Ewalina Jeyanesan's MSc Thesis R Markdown

Ewalina Jeyanesan

24/09/2020

Nomenclature and Abbreviations

- data = data frames with expression values proceeded by case/donor identification (including age, sex, PMI etc.)
- exp = data frames with expression values alone, no identifiers
- raw = original data: no transformations, normalization, or substitutions have been performed
- std.names = protein naming in data frame is standardized, usually by gene symbol
- syn and plasticity are used interchangeably to refer to plasticity marker data (depending on length of the variable name)
- immune refers to the neuroimmune data (shorted for more efficient code)
- frechet is used instead of Fréchet in the code, for ease of coding

Session Info (Ensure that only needed packages are loaded)

```
sessionInfo() # see what base R packages are already attached, R version etc.
```

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
## Matrix products: default
## BLAS:
/Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK:
/Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_CA.UTF-8/en_CA.UTF-8/en_CA.UTF-8/C/en_CA.UTF-8/en_CA.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.29
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.0.2  magrittr_1.5    tools_4.0.2     htmltools_0.5.0
```

```
##  [5] yaml_2.2.1      stringi_1.4.6   rmarkdown_2.3   stringr_1.4.0
##  [9] xfun_0.16       digest_0.6.25   rlang_0.4.7     evaluate_0.14
```

Load Packages

Certain packages are not available for direct installation from CRAN, these can be installed using the "devtools" package to access these directly from GitHub.

```
library(tidyverse) # clean, process, and visualize data (includes
dependencies like ggplot2 and dplyr)
library(janitor) # for various data cleaning functions
library(magrittr) # for managing data frames (i.e. set_rownames() and
set_colnames())
library(zoo) # for rollapply() function to process rolling windows
library(data.table) # for melt() function to reshape data
library(devtools) # used to install other packages not directly available
through CRAN
library(fitdistrplus) # for fitting models/ distributions to data
library(mosaic) # for calculating zscores
library(factoextra) # for WSS calculations and plot
library(pvclust) # for calculating p-values for hierarchical clusters
library(kmlShape) # for Frechet distance calculations
library(gplots) # for heatmap.2()
library(rlist) # for list.append() and other functions
library(plotrix) # for statistical measures including std.error and std.dev
library(qpcR) # for cbind() of varying lengths
library(gprofiler2) # gene enrichment tool
library(grid) # for plotting graphs
library(mice) # for data imputation
library(VIM) # for aggr() function
library(RSKC) # for robust and sparse k-means clustering
library(FactoMineR) # for PCA
library(readxl) # for reading and writing excel files
library(openxlsx) # for writing excel files with multiple sheets
library(dabestr) # for estimation statistics
library(Rmisc) # for confidence interval computation
library(clusteval) # for Jaccard Coefficient computation
library(gridExtra) # for grid.arrange function
library(ggpubr) # for ggarrange function
library(reshape2) # for melt function
library(dendextend) # rotating branches of dendrogram
#devtools::install_github("guiastrennec/ggplus")
library(ggplus) # for facet_multiple() function
```

Load Data Files

```
# Files for neuroimmune protein analyses
raw.data.file = read_csv("Raybiotech-All-Arrays-Result-Datafile.csv",
col_names = FALSE) # orginal raw data file
raybiotech.uniprotID.conversion = read_csv(file = "RaybiotechProteins.csv",
col_names = c("Raybiotech.Names","Uniprot.ID")) # RayBiotech protein names to
Uniprot IDS
uniprotID.genesymbol.conversion = read_csv(file = "DAVIDConversion.csv",
col_names = c("Uniprot.ID", "Gene.Symbol")) # Uniprot ID to standard gene
symbols


# Files for neural plasticity analyses
plasticity.data = read.csv("Synaptic Expression.csv", row.names = 1) # use
read.csv rather than read_csv to access row.names parameter
imputed.plasticity.data = read_csv("Average Imputed Dataframe.csv") # file
after imputing missing values


# Files for converting protein names to different aliases
gene.symbol.protein.names.conversion = read_csv("Gene Symbol to Our
Names.csv") # file with protein naming conversions
protein.names.long.format = read.xlsx("Gene Symbol Protein Name and Short
Name.xlsx") # file protein names


# Viral Receptor Information
uniprot.viral.receptors =  read_csv("Viral Receptors.csv") # list of human
virus receptors from Uniprot
# viral receptors matched to disease
paper.viral.receptors = read.csv("Viral Receptors and their Diseases.csv",
header = TRUE) # use read.csv rather than read_csv to access header parameter


# External Database files
syn.go.classification = read_csv("syngo_annotations_matching_user_input.csv")
# this file indiicates which proteins have known synaptic function
string.DB.export = read_csv("immune_cluster_string_interactions.csv") #
contains STRINGdb network for a cluster of interest
```

Clean Raw Neuroimmune Data File

```
# Replicate data file to perform data cleaning
raw.data.200.proteins = raw.data.file
# Move protein symbols to be in line with other identifier names (ex. age,
sex, etc.)
raw.data.200.proteins[5,7:206] = raw.data.200.proteins[2,7:206]
# Store LOD, Max, as a separate file, and remove them from this expression
data
LOD.Max.values = raw.data.200.proteins[(1:4),(6:206)]  %>%
row_to_names(2) %>%
column_to_rownames("Protein") %>%
  mutate_all(as.character) %>%  # first convert to characters, then ...
  mutate_all(as.numeric) # store values as numeric values
rownames(LOD.Max.values) = c("LOD", "Max")
raw.data.200.proteins = raw.data.200.proteins[-(1:4),]
# Set the column names as the protein symbols and reset row numbers
raw.data.200.proteins = raw.data.200.proteins %>%
row_to_names(1) %>%
set_rownames(NULL)
```

Create a data frame of the case/sample identifiers (i.e. case ID, age, age bin, sex, post-mortem interval (PMI), and PMI range) This is Table 1 in my thesis.

```
identifiers = raw.data.200.proteins[1:6]
# Convert the Years and PMI columns to numerical values, they are currently
stored as factors
identifiers$Years = as.numeric(as.character(identifiers$Years))
identifiers$PMI = as.numeric(as.character(identifiers$PMI))
# Change the PMI interval to PMI range for clarity
colnames(identifiers)[6] = "PMI Range"
```

For further processing create an expression file (i.e. measured protein concentrations alone, no identifiers)

```
raw.exp.200.proteins = raw.data.200.proteins[,-(1:6)] %>%
# Convert all expressions to numeric values
mutate_all(as.character) %>%
mutate_all(as.numeric)
```

For standardization purposes, the proteins will be referred to hereafter by their gene symbols. The RayBiotech company provided UniProt IDs for all the proteins in this array. The online tool “DAVID” was used to map the UniProt ID to the appropriate gene symbol. In the case that a UniProt ID was not recognized by DAVID, or if there are duplicates, a manual search of the protein name/ Uniprot ID was completed on the “Gene Cards” website to assign the gene symbol.

This is important for running analyses (like GO or inputting query terms into STRINGdb), as these analyses require the input of standard gene symbols - not protein names.

```r
# Create an "ID.matches" data file to store the RayBiotech protein names with
the correct gene symbol, merging the two
# original data files will be completed using the Uniprot ID
ID.matches = setNames(data.frame(matrix(ncol = 2, nrow = 200)),
c("Raybiotech.Names", "Gene.Symbol"))
ID.matches["Raybiotech.Names"] =
raybiotech.uniprotID.conversion["Raybiotech.Names"]

for (row.index in 1:nrow(ID.matches)){
  index = match(as.character(raybiotech.uniprotID.conversion[row.index,
"Uniprot.ID"]), as.vector(uniprotID.genesymbol.conversion$Uniprot.ID))
  ID.matches[row.index, "Gene.Symbol"] =
uniprotID.genesymbol.conversion[index, "Gene.Symbol"]
}
# Find the proteins that did not match to a gene symbol through DAVID
ID.NAs = ID.matches %>% filter(Gene.Symbol %in% NA)
# Manually add in gene symbols for "6Ckine", "Axl", "BTC", "LAP(TGFb1)",
"ST2", and "NRG1-b1"
ID.NAs$Gene.Symbol = c("CCL21", "AXL", "BTC", "LAP(TGFB1)", "IL1RL1",
"NRG1B1")

# Find gene symbols that are assigned to multiple proteins, count the number
of duplications
ID.duplicate.count = na.omit(ID.matches) %>%
group_by(Gene.Symbol) %>% # Group the data by Gene Symbol
tally() %>% # Tally how many time each Gene Symbol appears
filter(n > 1) # Filter for Gene Symbols that appear more than once
# Find the proteins that map onto the same gene symbol (duplicates)
ID.duplicates = ID.matches %>%
filter(Gene.Symbol %in% ID.duplicate.count$Gene.Symbol) %>%
arrange(Raybiotech.Names)
# Manually edit gene symbols to create distinct labels for duplicates
ID.duplicates$Gene.Symbol = c("IL12B", "IL12AB", "PDGFA", "PDGFAB",
"CXCL12A", "CXCL12B")

# Add the correct gene symbols to the NAs and duplicates in the ID.matches
data frame
ID.matches = ID.matches %>%
mutate(Gene.Symbol = toupper(Gene.Symbol)) %>% # Convert lowercase characters
to uppercase
mutate(Gene.Symbol = ifelse(Gene.Symbol %in% NA, # Add in the gene symbols
for all NA values
ID.NAs$Gene.Symbol[match(Raybiotech.Names,ID.NAs$Raybiotech.Names)],Gene.Symb
ol)) %>%
# Add in the new gene symbols that are distinct for each duplicate symbol
mutate(Gene.Symbol = ifelse(Gene.Symbol %in% ID.duplicate.count$Gene.Symbol,
ID.duplicates$Gene.Symbol[match(Raybiotech.Names,ID.duplicates$Raybiotech.Nam
es)],Gene.Symbol))
# Check the order of proteins for ID.matches and the columns of
```

```
raw.exp.200.proteins
identical(colnames(raw.exp.200.proteins), ID.matches$Raybiotech.Names)

## [1] TRUE

# Check the order of proteins for ID.matches and the columns of
raw.data.200.proteins
identical(colnames(raw.data.200.proteins)[7:206],
ID.matches$Raybiotech.Names)

## [1] TRUE

# Check the order of proteins for ID.matches and the columns of
LOD.Max.values
identical(colnames(LOD.Max.values), ID.matches$Raybiotech.Names)

## [1] TRUE
```

Create a table that shows conversions between RayBiotech target names, protein name (long format), and gene symbol This is Table 2 in my thesis.

```
table.for.protein.conversion = protein.names.long.format
table.for.protein.conversion$RayBiotech = NA # create empty column to store
values

for (i in 1:nrow(table.for.protein.conversion)){ # use for loop to assign
names to immune proteins
  index = which(ID.matches$Gene.Symbol %in% table.for.protein.conversion[i,
"Gene.Symbol"])
  if (length(index) == 0) {next}
  table.for.protein.conversion[i, "RayBiotech"] = ID.matches[index,
"Raybiotech.Names"]
}

table.for.protein.conversion[73:95, "RayBiotech"] =
gene.symbol.protein.names.conversion[73:95, "Protein"]

# write.csv(table.for.protein.conversion, "Table 2 Thesis.csv") # This is
file was manually edited for Greek symbols in Excel
```

Create new data frames using the standard gene symbols as column names

```
# Create a data frame with standard gene symbols
raw.exp.200.proteins.std.names = raw.exp.200.proteins %>%
set_colnames(ID.matches$Gene.Symbol)

# Do the same for our LOD, Max and Array Info data frame
LOD.Max.values.std.names = LOD.Max.values %>%
set_colnames(ID.matches$Gene.Symbol)
```

Below we can see that the only change between the old and new data frames is the use of standardized gene symbols to refer to the proteins.

raw.exp.200.proteins:

6Ckine	Axl	BTC	CCL28	CTACK	CXCL16	ENA-78	Eotaxin-3	GCP-2	GRO
6.0	40.5	0.0	0.0	0.5	1121.5	0.0	0.0	1.1	0.4
0.0	44.1	0.0	1.4	4.3	851.9	0.0	0.0	1.7	2.1
0.0	73.7	0.0	1.3	0.5	1026.9	0.0	0.0	1.0	1.3
0.0	27.1	0.0	2.2	1.7	621.0	0.0	0.0	1.0	0.8
10.1	107.7	7.2	38.3	7.7	757.4	0.0	3.5	1.0	4.7
0.0	32.8	0.0	25.6	4.2	856.6	9.7	0.0	5.6	3.8
0.0	71.5	0.0	0.0	0.2	1018.2	0.0	0.0	0.6	0.0
0.0	56.2	0.0	10.7	0.5	444.5	0.0	0.0	0.3	0.3
6.9	86.5	3.6	7.8	4.6	664.3	0.0	1.3	1.9	1.9
8.7	122.4	0.0	15.3	12.2	684.7	5.6	26.5	2.1	4.4

raw.exp.200.proteins.std.names:

CCL21	AXL	BTC	CCL28	CCL27	CXCL16	CXCL5	CCL26	CXCL6	CXCL1
6.0	40.5	0.0	0.0	0.5	1121.5	0.0	0.0	1.1	0.4
0.0	44.1	0.0	1.4	4.3	851.9	0.0	0.0	1.7	2.1
0.0	73.7	0.0	1.3	0.5	1026.9	0.0	0.0	1.0	1.3
0.0	27.1	0.0	2.2	1.7	621.0	0.0	0.0	1.0	0.8
10.1	107.7	7.2	38.3	7.7	757.4	0.0	3.5	1.0	4.7
0.0	32.8	0.0	25.6	4.2	856.6	9.7	0.0	5.6	3.8
0.0	71.5	0.0	0.0	0.2	1018.2	0.0	0.0	0.6	0.0
0.0	56.2	0.0	10.7	0.5	444.5	0.0	0.0	0.3	0.3
6.9	86.5	3.6	7.8	4.6	664.3	0.0	1.3	1.9	1.9
8.7	122.4	0.0	15.3	12.2	684.7	5.6	26.5	2.1	4.4

Selecting Proteins with Reliable Expression

The Multiplex ELISA used to measure the concentration of immune proteins reports an upper and lower limit of reliable detection for each protein. The upper limit is called the Maximum (Max) value, and the lower limit is the limit of detection (LOD) value. The first step is to determine which proteins have reliable expression. This study includes 30 post-mortem tissue samples; therefore, each protein has 30 concentration measurements. We decided that proteins with at least 10 measurements between the LOD and Max values (i.e. the reliable detection range) will be selected for further analysis.

To select proteins, we first categorize the measurements for each protein into four groups: Lowest, LOD, Max, Raw. For the values that fall below the LOD there are two interpretations. First, that there is a measuring error. Second, that during this period of the lifespan the protein expression actually falls to near zero levels. This study is cross-sectional in design, as a result, a rolling average will be used to determine whether four neighboring cases (in terms of age) age also show below LOD levels of expression.

```
# Add two empty rows to the top and bottom of our expression data frame so
the rollapply() function can
# calculate the rolling average for all 30 samples.

emptyrow = rep(NA, 200) # a vector with 200 NAs
sliding.window.dataframe = rbind(emptyrow, emptyrow, # bind two of these
empty rows to the beginning of the dataframe
                                 raw.exp.200.proteins,
                                 emptyrow, emptyrow)

# Build a dataframe to store the rolling averages.
sliding.window.means = sliding.window.dataframe %>%
# Here we use a rolling window of 5 cases to calculate the average.
  rollapply(width = 5, by = 1, FUN = mean, na.rm = TRUE, align = "center")
%>%
  data.frame(check.names = FALSE)
```

Using the table below, we will classify each protein expression into four classes (see below). Recall that each protein has its own LOD and Max values.

1. Lowest: protein expressions where both the raw expression value and rolling average are less than the LOD. We assign these expressions a value of 0.1 (i.e. smallest decimal place in our data set)

2. LOD: protein expressions where the raw expression value is less than the LOD, but the rolling average is greater than or equal to the LOD. We assign these expression values the LOD value.

3. Max: protein expressions with a raw expression value that is greater than the maximum reliable expression value. We round these expressions down to the Max value.

4. Raw: protein expressions that fall between the LOD and the Max values. In other words, the expression value falls within reliably detectable range, so we do not adjust it.

The table below gives a summary of this:

Classification	Assigned Value	Raw Expression Value	Rolling Average
LOWEST	0.1	< LOD Value	< LOD Value
LOD	LOD Value	< LOD Value	≥ LOD Value
MAX	Max Value	> Max Value	
RAW	Raw Expression Value	> LOD Score, < Max Value	

Build a data frame that classifies each measurement into one of our four groups (Lowest, LOD, Max, or Raw).

```
value.classification = as.data.frame(matrix(ncol = 200, nrow = 30))
colnames(value.classification) = colnames(raw.exp.200.proteins)

for (protein.index in 1:200){
  for (case.index in 1:30) {
    # Expression < LOD and Mean < LOD ==> Lowest
    if (raw.exp.200.proteins[case.index,protein.index] <
LOD.Max.values[1,protein.index]
        & sliding.window.means[case.index,protein.index] <
LOD.Max.values[1,protein.index]) {
      value.classification[case.index,protein.index] = "Lowest"
    # Expression < LOD and Mean >= LOD ==> LOD
    } else if (raw.exp.200.proteins[case.index,protein.index] <
LOD.Max.values[1,protein.index]
               & sliding.window.means[case.index,protein.index] >=
LOD.Max.values[1,protein.index]) {
      value.classification[case.index,protein.index] = "LOD"
    # Expression > Max ==> Max
    } else if (raw.exp.200.proteins[case.index,protein.index] >
LOD.Max.values[2,protein.index]) {
      value.classification[case.index,protein.index] = "Max"
    # LOD < Expression < Max ==> Raw
    } else {value.classification[case.index,protein.index] = "Raw"}
  }
}
```

Below, I have displayed the first 15 rows of a few select proteins to indicate the structure and content of this new "value.classification" data frame. Overall, it stores information about each of the 30 measurements for a single protein and indicates which of the aforementioned groups (Lowest, LOD, Max or Raw) they belong in.

value.classification:

MPIF-1	IFNg	TRAIL R3	Lipocalin-2
Lowest	Lowest	Raw	Raw
Lowest	Lowest	LOD	Raw
Lowest	Lowest	Raw	Raw
Lowest	Lowest	Raw	Raw
Lowest	Lowest	Raw	Raw
Lowest	Lowest	Raw	Raw
Lowest	Lowest	LOD	Raw
Lowest	Lowest	Raw	Raw
Lowest	Lowest	LOD	Raw
Lowest	Lowest	Raw	Raw
Lowest	Lowest	Raw	Raw
Lowest	Raw	Raw	Max
Lowest	Lowest	Raw	Max
Lowest	Lowest	Raw	Raw
Lowest	LOD	LOD	Raw

Next, I build a data frame that stores adjusted protein expression values for measurements that fall outside of the reliable detection range.

```
corrected.exp.200.proteins = as.data.frame(matrix(ncol = 200, nrow = 30))
colnames(corrected.exp.200.proteins) <- colnames(raw.exp.200.proteins)

for (protein.index in 1:200){
  for (case.index in 1:30) {
    # Expression < LOD and Mean < LOD ==> 0.1
    if (raw.exp.200.proteins[case.index,protein.index] <
LOD.Max.values[1,protein.index] &
        sliding.window.means[case.index,protein.index] <
LOD.Max.values[1,protein.index]) {
      corrected.exp.200.proteins[case.index,protein.index] = 0.1
    # Expression < LOD and Mean >= LOD ==> LOD
    } else if (raw.exp.200.proteins[case.index,protein.index] <
LOD.Max.values[1,protein.index] &
                sliding.window.means[case.index,protein.index] >=
LOD.Max.values[1,protein.index]) {
      corrected.exp.200.proteins[case.index,protein.index] =
LOD.Max.values[1,protein.index]
     # Expression > Max ==> Max
    } else if (raw.exp.200.proteins[case.index,protein.index] >
LOD.Max.values[2,protein.index]) {
      corrected.exp.200.proteins[case.index,protein.index] =
LOD.Max.values[2,protein.index]
    # LOD < Expression < Max ==> Raw
    } else {corrected.exp.200.proteins[case.index,protein.index] =
      raw.exp.200.proteins[case.index,protein.index]}
  }
}
```

Save the data frame with substituted values.

```
# write.csv(corrected.exp.200.proteins,
#           "Immune Protein (200) Expression - With Subsitiutions.csv",
row.names = F)
```

The next step is to use this classification data frame to filter for proteins that have at least 10 Raw measurements.

Selecting proteins with sufficient amount of raw expression data

In this section, we will determine how many expressions are in each of the four groups for all 200 proteins. Then we will select only the proteins that have 10 or more raw expression values after correcting for those outside of the LOD-Max range. We will then subset each of our relevant data frames to only include the proteins that meet this criterion.

Create a data frame that counts the number of expressions in each group for all 200 proteins.

```
# Replicate the value.classification df which contains the assigned group for
each expression
protein.classification.counts = value.classification %>%
  # Restructure the dataset into 2 columns of Protein-Type(Group) pairs
  gather("Protein", "Type", colnames(value.classification)) %>%
  # Group by Protein and Type
  dplyr::group_by(Protein, Type) %>%
  # Tally how many expressions there are for each Protein-Type pair
  dplyr::tally() %>%
  # Count the number of expressions fall in each group
  dplyr::mutate(`Number Raw` = ifelse(Type == "Raw", n, 0),
         `Number LOD` = ifelse(Type == "LOD", n, 0),
         `Number Lowest` = ifelse(Type == "Lowest", n, 0),
         `Number Max` = ifelse(Type == "Max", n, 0)) %>%
  # Restructure so each row contains a protein and the counts for each group
  dplyr::group_by(Protein) %>%
  dplyr::summarise(`Number Raw` = sum(`Number Raw`),
            `Number LOD` = sum(`Number LOD`),
            `Number Lowest` = sum(`Number Lowest`),
            `Number Max` = sum(`Number Max`))
```

See the table below to get a visual of what the first 10 rows of the 'protein.classification.counts' data frame looks like.

Protein	Number Raw	Number LOD	Number Lowest	Number Max
4-1BB	3	0	27	0
6Ckine	1	0	29	0
Activin A	0	0	30	0
AgRP	3	0	27	0
ALCAM	30	0	0	0
ANG-1	2	0	28	0
Angiogenin	30	0	0	0
Angiostatin	0	0	30	0
AR	3	0	27	0
Axl	30	0	0	0

Now, select only the proteins with a minimum of 10 raw expression values (i.e. 10 values within the reliable detection range).

```
# Filter for only proteins with minimum 10 expressions in the Raw group
selected.immune.proteins = protein.classification.counts %>% filter(`Number
Raw` >= 10) # There are 72 proteins that meet this criteria
```

selected.immune.proteins:

Protein	Number Raw	Number LOD	Number Lowest	Number Max
ALCAM	30	0	0	0
Angiogenin	30	0	0	0
Axl	30	0	0	0
bFGF	30	0	0	0
Cathepsin S	30	0	0	0
CD14	30	0	0	0
CEACAM-1	19	9	2	0
Contactin-2	30	0	0	0
CXCL16	30	0	0	0
Dtk	30	0	0	0
EG-VEGF	10	9	11	0
EGF R	30	0	0	0
ErbB3	25	2	3	0

Fas	23	7	0	0
Fcg RIIBC	19	5	6	0
Flt-3L	17	4	9	0
GDF-15	12	9	9	0
GDNF	12	5	13	0
gp130	30	0	0	0
HCC-1	22	5	3	0
HGF	30	0	0	0
HVEM	10	2	18	0
I-TAC	28	2	0	0
ICAM-1	30	0	0	0
ICAM-2	30	0	0	0
IGFBP-1	21	7	2	0
IGFBP-2	30	0	0	0
IGFBP-6	23	7	0	0
IL-11	25	5	0	0
IL-13 R1	13	9	8	0
IL-13 R2	17	10	3	0
IL-16	30	0	0	0
IL-1a	20	4	6	0
IL-1ra	25	4	1	0
IL-2 Rb	22	7	1	0
IL-2 Rg	10	5	15	0
IL-4	15	6	9	0
IL-6R	30	0	0	0
IL-8	19	5	6	0
IL-9	14	5	11	0
LAP(TGFb1)	30	0	0	0
LIF	16	10	4	0
LIMPII	30	0	0	0
Lipocalin-2	28	0	0	2
LYVE-1	30	0	0	0
MCP-1	26	4	0	0
MCSF	29	1	0	0
MCSF R	30	0	0	0
MICA	10	3	17	0

NAP-2	30	0	0	0
NrCAM	29	0	0	1
OPN	30	0	0	0
PARC	23	7	0	0
PDGF-AA	30	0	0	0
PDGF-AB	18	9	3	0
PDGF-BB	14	7	9	0
PECAM-1	27	3	0	0
PF4	28	2	0	0
RANTES	24	6	0	0
Resistin	21	8	1	0
SCF R	30	0	0	0
Siglec-5	24	3	3	0
TGFa	11	6	13	0
TIMP-1	30	0	0	0
TIMP-2	30	0	0	0
TNF RI	30	0	0	0
TNF RII	27	3	0	0
TNFa	22	7	1	0
TRAIL R3	18	8	4	0
Trappin-2	13	5	12	0
uPAR	20	9	1	0
VEGF R1	30	0	0	0

Create new data frames that contain only the 72 immune proteins with reliable expression.

```
# Save just the protein names as a vector
selected.immune.proteins = selected.immune.proteins$Protein

# Raw expression values
raw.exp.72.proteins = raw.exp.200.proteins %>%
  dplyr::select(all_of(selected.immune.proteins))

# LOD scores and max values
LOD.Max.72.proteins = LOD.Max.values %>%
  dplyr::select(all_of(selected.immune.proteins))

# Rolling averages
sliding.window.means.72.proteins = sliding.window.means %>%
  dplyr::select(all_of(selected.immune.proteins))

# Classification of each expression
value.classification.72.proteins = value.classification %>%
  dplyr::select(all_of(selected.immune.proteins))

# Adjusted expression values
corrected.exp.72.proteins = corrected.exp.200.proteins %>%
  dplyr::select(all_of(selected.immune.proteins))
```

Save the trimmed down data frame of our adjusted expression values for the 72 selected proteins.

```
# write.csv(corrected.exp.72.proteins,
#           "Immune Protein (72) Expression - With Subsitiutions.csv",
row.names = F)
```

Below, I have picked out the lifespan trajectory of 4 neuroimmune proteins. These highlight examples of proteins that were picked for further analysis and those that were not.

Recall that the number of raw measurements is what determines whether a protein is selected or not. Therefore, for simplicity's sake all values above the reliable limit will be labelled "Above MAX" and all values below the LOD will be labeled "Below LOD" in the following figure, with those in the reliable range labelled as "Reliable".

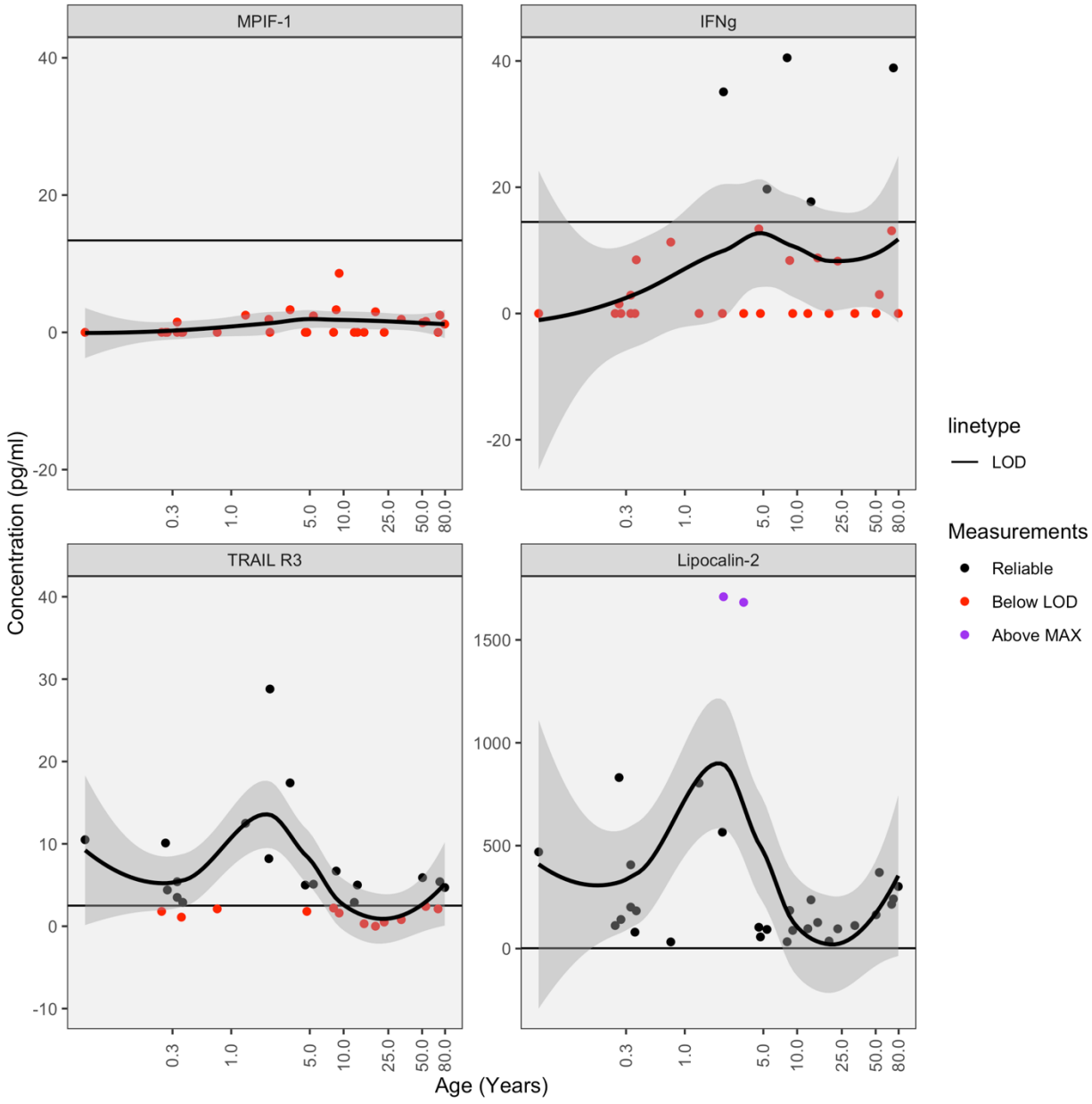
THIS IS FIGURE 2 in my thesis.

```
value.classification.2 = value.classification
value.classification.2[value.classification.2=="Max"] = "Above MAX"
value.classification.2[value.classification.2=="Lowest"] = "Below LOD"
value.classification.2[value.classification.2=="LOD"] = "Below LOD"
value.classification.2[value.classification.2=="Raw"] = "Reliable"
```

```r
# Bind the identifiers and adjusted protein expression values together
data.for.loess = cbind(identifiers, raw.exp.200.proteins) %>%
  # Restructure the data frame by grouping the columns together using the
common identifiers
  melt(id = colnames(identifiers)) %>%
  # Rename columns as Protein and Expression
  dplyr::rename(Protein = variable, Expression = value) %>%
  # Convert Protein from factor to character
  mutate(Protein = as.character(Protein))

# Bind the identifiers and adjusted protein expression groups together
labels.for.loess = cbind(identifiers, value.classification.2) %>%
  # Restructure the data frame by grouping the columns together using the
common identifiers
  melt(id = colnames(identifiers)) %>%
  # Rename columns as Protein and Label
  dplyr::rename(Protein = variable, Label = value) %>%
  # Convert Protein from factor to character
  mutate(Protein = as.character(Protein))

# Merge the data.for.loess and labels.for.loess data frames together
graphable.data = merge(data.for.loess, labels.for.loess, by =
c(colnames(identifiers), "Protein")) %>%
  # Convert Expression from character to numeric and Label from character to
factor
  mutate(Expression = as.numeric(Expression),
         Label = factor(Label, levels = c("Reliable", "Below LOD", "Above
MAX")))

# Create a vector that stores the LOD value for the example proteins
LOD.vector = c(LOD.Max.values[1, "MPIF-1"], LOD.Max.values[1, "IFNg"],
LOD.Max.values[1, "TRAIL R3"], LOD.Max.values[1, "Lipocalin-2"])


# Create a vector that stores the LOD value for the example proteins
MAX.vector = c(LOD.Max.values[2, "MPIF-1"], LOD.Max.values[2, "IFNg"],
LOD.Max.values[2, "TRAIL R3"], LOD.Max.values[2, "Lipocalin-2"])

selected.example = graphable.data %>% filter(Protein %in% c("TRAIL R3",
"IFNg", "MPIF-1", "Lipocalin-2"))
selected.example$LOD = NA # create an empty column with NAs to store the LOD
value for labelling on the graph
selected.example$MAX = NA # create an empty column with NAs to store the MAX
value for labelling on the graph


# Use the vectors to fill the data frame columns with the proteins' LOD and
the MAX values
for(i in 1:4){
```

```
  # CCL23
  if(selected.example[i, "Protein"] == "MPIF-1"){selected.example[i, "LOD"] =
LOD.vector[1]}
  if(selected.example[i, "Protein"] == "MPIF-1"){selected.example[i, "MAX"] =
MAX.vector[1]}
  # IFNG
  if(selected.example[i, "Protein"] == "IFNg"){selected.example[i, "LOD"] =
LOD.vector[2]}
  if(selected.example[i, "Protein"] == "IFNg"){selected.example[i, "MAX"] =
MAX.vector[2]}
  # TNFRSF10C
  if(selected.example[i, "Protein"] == "TRAIL R3"){selected.example[i, "LOD"]
= LOD.vector[3]}
  if(selected.example[i, "Protein"] == "TRAIL R3"){selected.example[i, "MAX"]
= MAX.vector[3]}
  # LCN2
  if(selected.example[i, "Protein"] == "Lipocalin-2"){selected.example[i,
"LOD"] = LOD.vector[4]}
  if(selected.example[i, "Protein"] == "Lipocalin-2"){selected.example[i,
"MAX"] = MAX.vector[4]}
}

# Set factor levels to plot the proteins in a specific order
selected.example$Protein = factor(selected.example$Protein, level = c("MPIF-
1", "IFNg", "TRAIL R3", "Lipocalin-2"))
selected.example$Years = as.numeric(selected.example$Years) # convert "Years"
column to a numeric value
selected.example$YMAX = NA # create an empty column to specify y-axis upper
limit (for visualization)
selected.example$YMIN = NA # create an empty column to specify x-axis lower
limit (for visualization)



# Manually set the max and min y axis values for visualization purposes
# The problem with automatic labels is that they sometimes fail to label the
absolute max / min values
selected.example[1,"YMAX"] = 20
selected.example[2,"YMAX"] = 60
selected.example[3,"YMAX"] = 40
selected.example[4,"YMAX"] = 40

selected.example[1,"YMIN"] = -10
selected.example[2,"YMIN"] = -40
selected.example[3,"YMIN"] = -20
selected.example[4,"YMIN"] = -10

# Set the colours for plotting
cols = c("Reliable" = "black", "Below LOD" = "red", "Above MAX" = "purple")
```

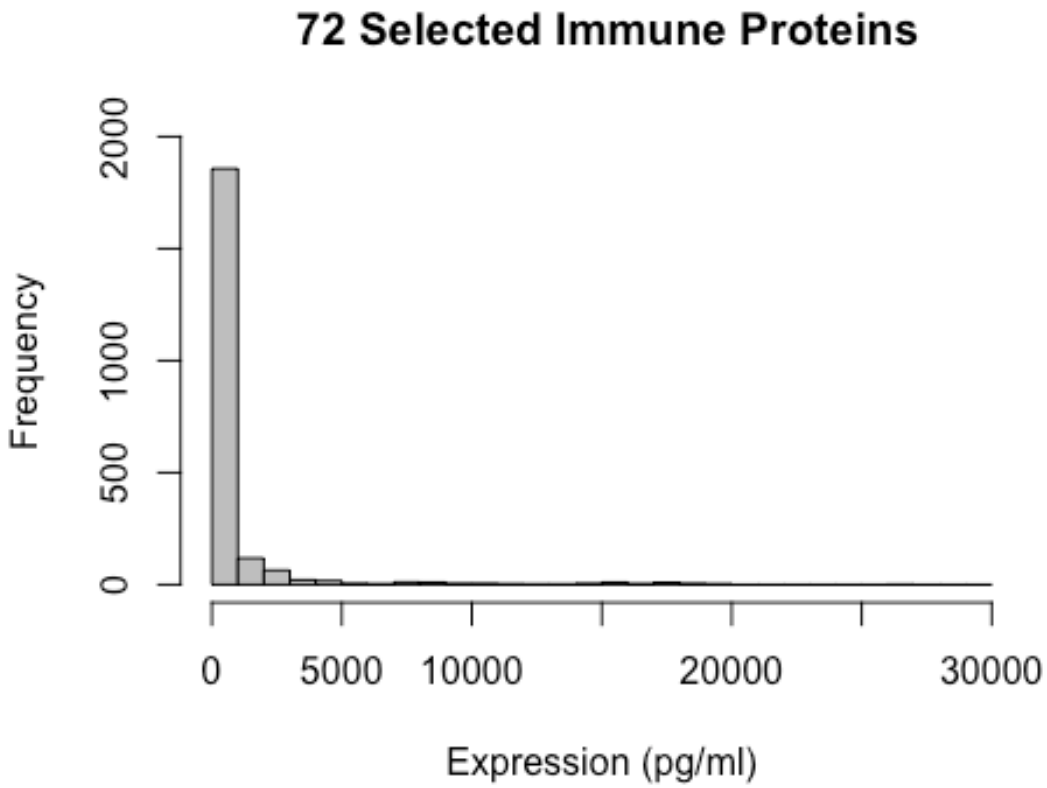
Plot the select curves using the ggplot code found below on page 30. Make sure the selected example is the input. MPIF-1 and IFNg are examples of proteins that are not selected. TRAIL R and Lipocalin-2 are examples of selected proteins.



Data Transformation

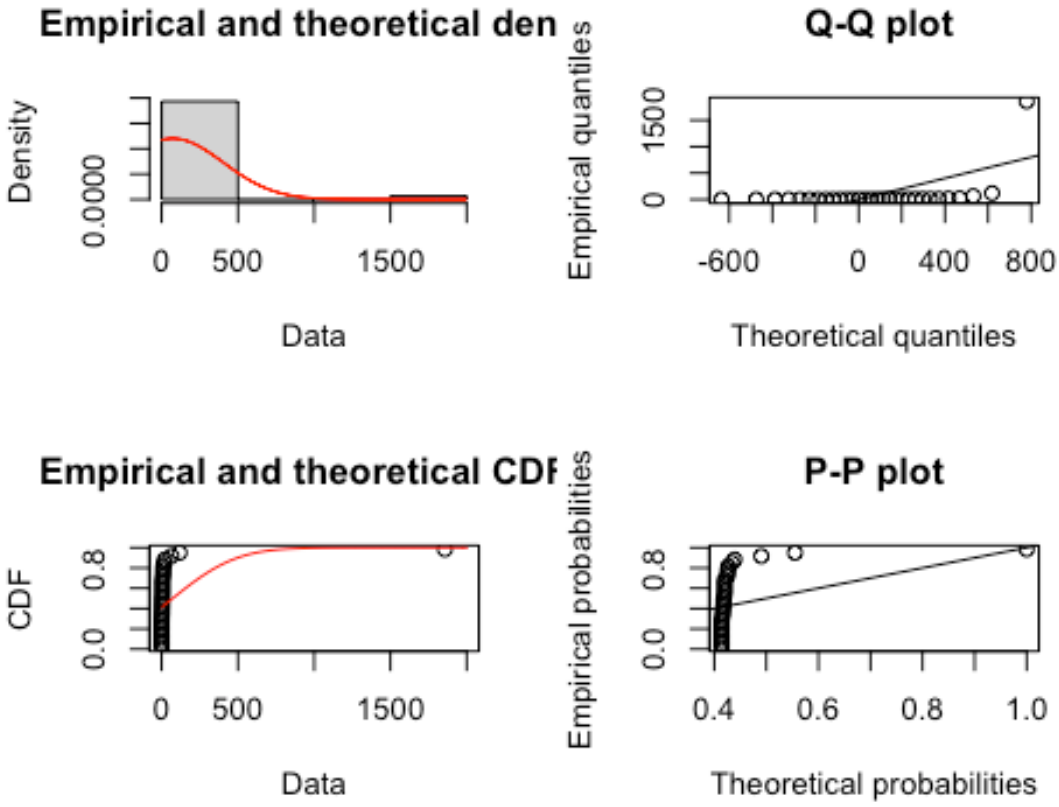
The proteins concentrations are orders of magnitude apart. Some proteins have less than 1pg/ml of expression, while others have over 2000pg/ml of expression. These values are not comparable. Moreover, many statistical tests and other analyses assume that data is normally distributed, while our data currently is not. In order to visualize the skewness of our data we plotted frequency histograms of the values and fitted them with a normal distribution.

```
# Raw Data with Substituted Values
corrected.exp.72.proteins = as.matrix(corrected.exp.72.proteins) # histogram
function works on a matrix
max(as.numeric(unlist(corrected.exp.72.proteins)), na.rm = TRUE) # use to set
max x-axis limit of frequency plot

## [1] 26506.9

min(as.numeric(unlist(corrected.exp.72.proteins)), na.rm = TRUE) # use to set
min x-axis limits of frequency plot

## [1] 0.1

# Plot frequency histogram
exp.freq.hist = hist(corrected.exp.72.proteins, freq = TRUE, main = " 72
Selected Immune Proteins",
           ylab= "Frequency", xlab= "Expression (pg/ml)", breaks =
seq(0,30000, by = 1000),
           xlim = c(0, 30000), ylim = c(0, 2000), labels = F, cex= 0.3,
col="grey")
```



By fitting a normal distribution to the data, we can see that the expressions are right-skewed.

```
# Fit a normal distribution to the frequency counts
exp.fit = fitdist(exp.freq.hist[["counts"]], "norm")
plot(exp.fit)
```

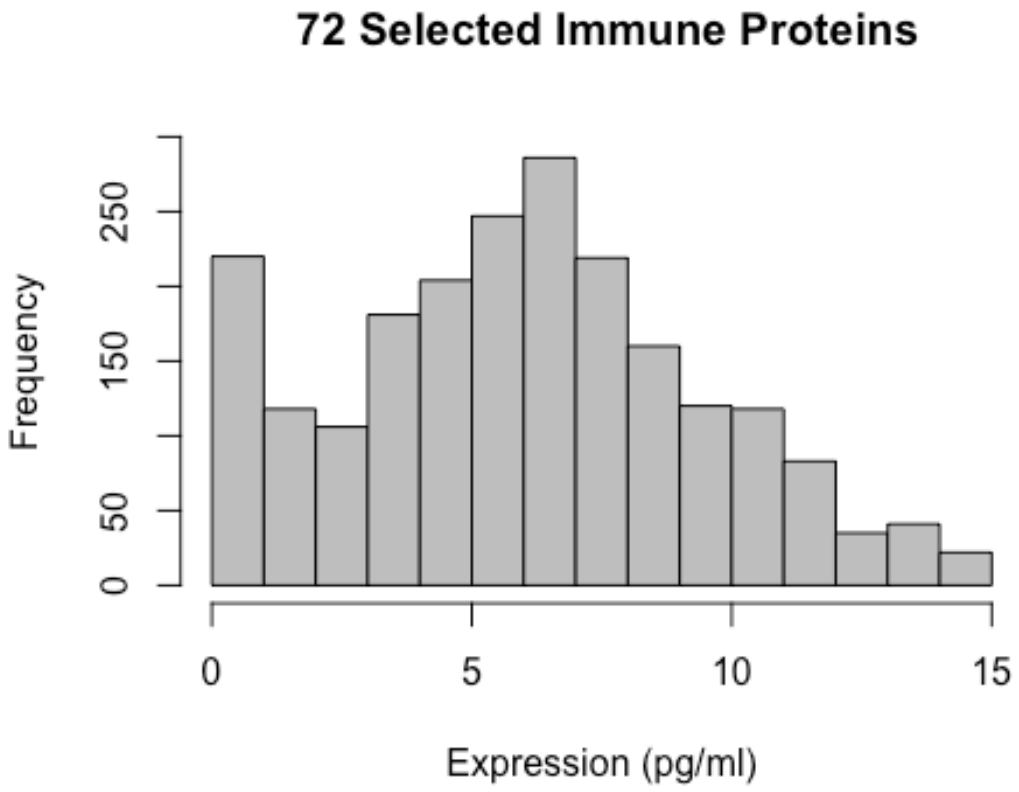


A log2 transformation is applied to the data to obtain a more normal distribution and to reduce the orders of magnitude difference between the concentration measurements of the proteins. A log2 was used as opposed to a base 10 transformation because log2 provides larger more efficient numbers than log10.

Log2 transformation of the data.

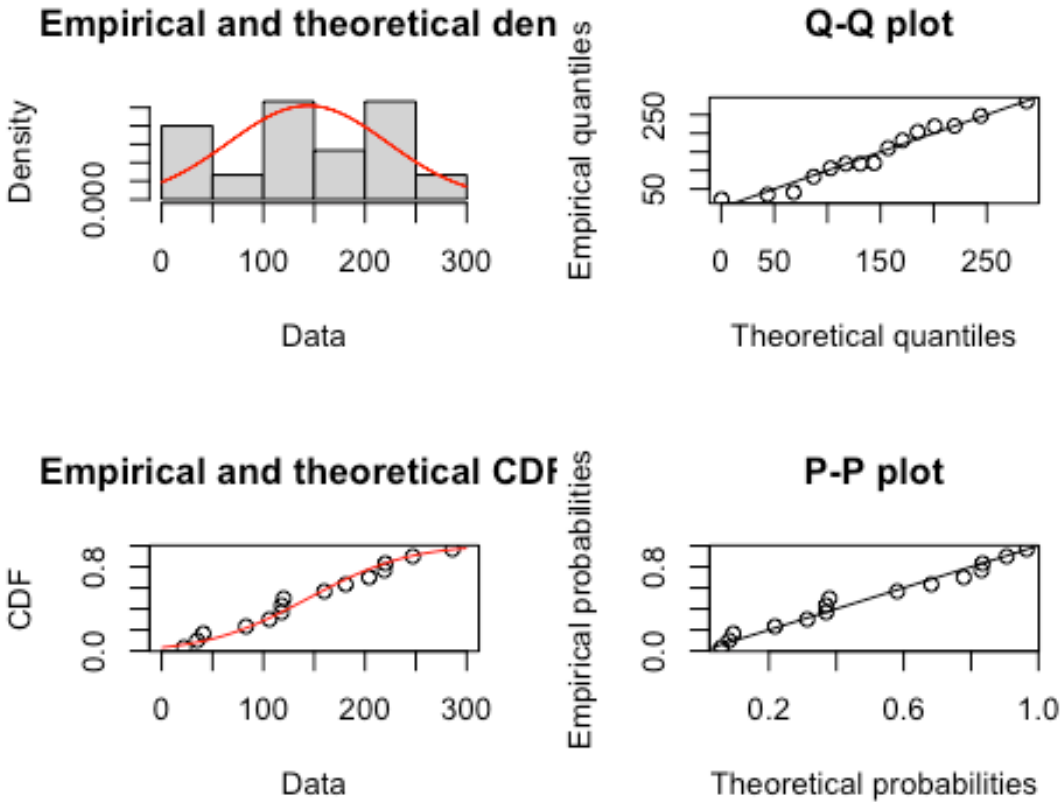
```
# The data contains values between 0 and 1. When performing log2
transformation these will become negatives. In order to avoid negative values
a constant of 1 is added to each measurement before performing log
transformation.
log2.exp.72.proteins = log2(corrected.exp.72.proteins + 1)

# Raw Data with Substituted Values
log2.exp.72.proteins = as.matrix(log2.exp.72.proteins) # histogram function
needs input to be a matrix
max(as.numeric(unlist(log2.exp.72.proteins)), na.rm = TRUE) # use to set max
x-axis limits of frequency plot

## [1] 14.69413

min(as.numeric(unlist(log2.exp.72.proteins)), na.rm = TRUE) # use to set min
x-axis limits of frequency plot
```

```
## [1] 0.1375035

# Plot frequency histogram
log.exp.freq.hist = hist(log2.exp.72.proteins, freq = TRUE, main = " 72
Selected Immune Proteins",
           ylab= "Frequency", xlab= "Expression (pg/ml)", breaks = seq(0,15,
by = 1),
           xlim = c(0, 15), ylim = c(0, 300), labels = F, cex= 0.3,
col="grey")
```



By fitting a normal distribution to the log2 data we can see that the values now more closely resemble a normal distribution. In the Q-Q plot if the solid line represents a normal distribution, the open circles represent the actual data. Here we see that the circles overlap much more with the solid line than before.

```
# Fit a normal distribution to the frequency counts
log.fit = fitdist(log.exp.freq.hist[["counts"]], "norm")
plot(log.fit)
```



Lifespan Trajectories of Neuroimmune Proteins (All 200)

To visualize the lifespan expression patterns of proteins, we have used a local regression method called LOESS (locally estimated scatterplot smoothing). This method works by smoothing the curve between the independent (age) and dependent (expression) variables using a rolling average. By smoothing the curve, we are able to better understand the patterns of expression, rather than using a scatterplot, where one outlier may misrepresent the level of expression at that stage on the lifespan.

Create and combine the data frames that contain the identifiers for each sample (ex. age, sex, PMI, etc.), the adjusted protein expression values and the classification for the measurements (i.e. Lowest, LOD, Max or Raw).

```
# Bind the identifiers and adjusted protein expression values together
data.for.loess = cbind(identifiers, raw.exp.200.proteins) %>%
  # Restructure the data frame by grouping the columns together using the
common identifiers
  melt(id = colnames(identifiers)) %>%
  # Rename columns as Protein and Expression
  dplyr::rename(Protein = variable, Expression = value) %>%
```

```
  # Convert Protein from factor to character
  mutate(Protein = as.character(Protein))

# Bind the identifiers and adjusted protein expression groups together

 value.classification.raw = value.classification
 colnames(value.classification.raw) = colnames(raw.exp.200.proteins)

labels.for.loess = cbind(identifiers, value.classification.raw) %>%
  # Restructure the data frame by grouping the columns together using the
common identifiers
  melt(id = colnames(identifiers)) %>%
  # Rename columns as Protein and Label
  dplyr::rename(Protein = variable, Label = value) %>%
  # Convert Protein from factor to character
  mutate(Protein = as.character(Protein))

# Merge the data.for.loess and labels.for.loess dataframes together
graphable.data = merge(data.for.loess, labels.for.loess, by =
c(colnames(identifiers), "Protein")) %>%
  # Convert Expression from character to numeric and Label from character to
factor
  mutate(Expression = as.numeric(Expression),
         Years = as.numeric(Years),
         Label = factor(Label, levels = c("Raw", "LOD", "Lowest", "Max")))


# Add the LOD and MAX values for plotting
graphable.data$LOD = NA # create an empty column with NAs to store the LOD
value for labelling on the graph
graphable.data$MAX = NA # create an empty column with NAs to store the MAX
value for labelling on the graph


# Use the vectors to fill the data frame columns with the proteins' LOD and
the MAX values

LOD.Max.values.raw = LOD.Max.values
colnames(LOD.Max.values.raw) = colnames(raw.exp.200.proteins)

for(i in 1:200){

index = min(which(graphable.data$Protein %in%
colnames(LOD.Max.values.raw)[i]))

graphable.data[index, "LOD"] = LOD.Max.values.raw[1, i]
graphable.data[index, "MAX"] = LOD.Max.values.raw[2, i]

}
```

```
graphable.data$Text = paste("LOD=", graphable.data$LOD, "pg/ml; MAX=",
graphable.data$MAX, "pg/ml")
```

Order the proteins correctly: R does not realize that Il1a should come before IL10, as it interprets the letter "a" as coming after the number "0".

```
protein.names.ordered.200 = sort(colnames(raw.exp.200.proteins))
protein.names.ordered.200
```

```
##   [1] "4-1BB"        "6Ckine"       "Activin A"    "AgRP"         "ALCAM"
##   [6] "ANG-1"        "Angiogenin"   "Angiostatin"  "AR"           "Axl"
##  [11] "b-NGF"        "B7-1"         "BCMA"         "BDNF"         "bFGF"
##  [16] "BLC"          "BMP-4"        "BMP-5"        "BMP-7"        "BTC"
##  [21] "Cathepsin S"  "CCL28"        "CD14"         "CD30"         "CD40"
##  [26] "CD40L"        "CEACAM-1"     "Contactin-2"  "Cripto-1"     "CTACK"
##  [31] "CXCL16"       "DAN"          "DKK-1"        "DR6"          "Dtk"
##  [36] "E-Cadherin"   "E-Selectin"   "EG-VEGF"      "EGF"          "EGF R"
##  [41] "ENA-78"       "Endoglin"     "Eotaxin"      "Eotaxin-2"
"Eotaxin-3"
##  [46] "EpCAM"        "ErbB3"        "Fas"          "FAS L"        "Fcg
RIIBC"
##  [51] "FGF-4"        "FGF-7"        "Flt-3L"       "Follistatin"  "G-CSF"
##  [56] "Galectin-7"   "GCP-2"        "GDF-15"       "GDNF"         "GH"
##  [61] "GITR"         "GM-CSF"       "gp130"        "GRO"          "HB-EGF"
##  [66] "HCC-1"        "HCC-4"        "HGF"          "HVEM"         "I-309"
##  [71] "I-TAC"        "ICAM-1"       "ICAM-2"       "ICAM-3"       "IFNg"
##  [76] "IGF-1"        "IGFBP-1"      "IGFBP-2"      "IGFBP-3"      "IGFBP-
4"
##  [81] "IGFBP-6"      "IL-1 RI"      "IL-10"        "IL-10 Rb"     "IL-11"
##  [86] "IL-12p40"     "IL-12p70"     "IL-13"        "IL-13 R1"     "IL-13
R2"
##  [91] "IL-15"        "IL-16"        "IL-17"        "IL-17B"       "IL-17F"
##  [96] "IL-17R"       "IL-18 BPa"    "IL-1a"        "IL-1b"        "IL-1ra"
## [101] "IL-2"         "IL-2 Ra"      "IL-2 Rb"      "IL-2 Rg"      "IL-21R"
## [106] "IL-23"        "IL-28A"       "IL-29"        "IL-31"        "IL-4"
## [111] "IL-5"         "IL-6"         "IL-6R"        "IL-7"         "IL-8"
## [116] "IL-9"         "Insulin"      "IP-10"        "L-Selectin"
"LAP(TGFb1)"
## [121] "LIF"          "LIGHT"        "LIMPII"       "Lipocalin-2"
"Lymphotactin"
## [126] "LYVE-1"       "MCP-1"        "MCP-2"        "MCP-3"        "MCP-4"
## [131] "MCSF"         "MCSF R"       "MDC"          "MICA"         "MICB"
## [136] "MIF"          "MIG"          "MIP-1a"       "MIP-1b"       "MIP-1d"
## [141] "MIP-3a"       "MIP-3b"       "MPIF-1"       "MSP"          "NAP-2"
## [146] "NGF R"        "NrCAM"        "NRG1-b1"      "NT-3"         "NT-4"
```

```
## [151] "OPG"          "OPN"          "PAI-1"        "PARC"         "PDGF
Rb"
## [156] "PDGF-AA"      "PDGF-AB"      "PDGF-BB"      "PECAM-1"      "PF4"
## [161] "PIGF"         "RAGE"         "RANTES"       "Resistin"     "SCF"
## [166] "SCF R"        "SDF-1a"       "SDF-1b"       "Shh-N"        "Siglec-
5"
## [171] "ST2"          "TARC"         "TECK"         "TGFa"         "TGFb1"
## [176] "TGFb2"        "TGFb3"        "Tie-2"        "TIM-1"        "TIMP-1"
## [181] "TIMP-2"       "TNF RI"       "TNF RII"      "TNFa"         "TNFb"
## [186] "TPO"          "TRAIL R3"     "TRAIL R4"     "Trappin-2"    "TREM-1"
## [191] "TSLP"         "uPAR"         "VCAM-1"       "VEGF"         "VEGF
R1"
## [196] "VEGF R2"      "VEGF R3"      "VEGF-C"       "VEGF-D"       "XEDAR"

protein.names.ordered.200 =
c("4-1BB", "6Ckine", "Activin A", "AgRP", "ALCAM", "ANG-1", "Angiogenin",
"Angiostatin","AR", "Axl",
"B7-1", "b-NGF", "BCMA", "BDNF", "bFGF", "BLC", "BMP-4", "BMP-5", "BMP-7",
"BTC", "Cathepsin S",
"CCL28", "CD14", "CD30", "CD40", "CD40L", "CEACAM-1", "Contactin-2",
"Cripto-1", "CTACK",
"CXCL16", "DAN", "DKK-1", "DR6", "Dtk", "E-Cadherin", "E-Selectin", "EG-
VEGF", "EGF", "EGF R",
"ENA-78", "Endoglin", "Eotaxin", "Eotaxin-2", "Eotaxin-3", "EpCAM", "ErbB3",
"Fas", "FAS L", "Fcg RIIBC",
"FGF-4", "FGF-7", "Flt-3L", "Follistatin",  "G-CSF", "Galectin-7", "GCP-2",
"GDF-15", "GDNF", "GH",
"GITR", "GM-CSF", "gp130", "GRO", "HB-EGF", "HCC-1", "HCC-4", "HGF", "HVEM",
"I-309",
 "I-TAC", "ICAM-1", "ICAM-2", "ICAM-3", "IFNg", "IGF-1", "IGFBP-1", "IGFBP-
2", "IGFBP-3", "IGFBP-4",
"IGFBP-6", "IL-1a", "IL-1b", "IL-1ra", "IL-1 RI", "IL-2", "IL-2 Ra", "IL-2
Rb", "IL-2 Rg", "IL-4",
"IL-5", "IL-6", "IL-6R", "IL-7", "IL-8", "IL-9", "IL-10", "IL-10 Rb", "IL-
11", "IL-12p40", "IL-12p70",
"IL-13", "IL-13 R1", "IL-13 R2", "IL-15", "IL-16", "IL-17", "IL-17B", "IL-
17F", "IL-17R", "IL-18 BPa",
"IL-21R", "IL-23", "IL-28A", "IL-29", "IL-31",  "Insulin", "IP-10", "L-
Selectin", "LAP(TGFb1)",
"LIF", "LIGHT", "LIMPII", "Lipocalin-2",  "Lymphotactin", "LYVE-1", "MCP-1",
"MCP-2", "MCP-3", "MCP-4",
"MCSF", "MCSF R", "MDC", "MICA", "MICB", "MIF", "MIG", "MIP-1a", "MIP-1b",
"MIP-1d",
"MIP-3a", "MIP-3b", "MPIF-1", "MSP", "NAP-2", "NGF R", "NrCAM", "NRG1-b1",
"NT-3", "NT-4",
"OPG", "OPN", "PAI-1", "PARC", "PDGF-AA", "PDGF-AB", "PDGF-BB", "PDGF Rb",
"PECAM-1", "PF4",
"PIGF", "RAGE", "RANTES", "Resistin", "SCF", "SCF R", "SDF-1a", "SDF-1b",
"Shh-N", "Siglec-5",
"ST2", "TARC", "TECK", "TGFa", "TGFb1", "TGFb2", "TGFb3", "Tie-2", "TIM-1",
```

```
"TIMP-1",
"TIMP-2", "TNFa", "TNFb", "TNF RI", "TNF RII", "TPO", "TRAIL R3", "TRAIL R4",
"Trappin-2", "TREM-1",
"TSLP", "uPAR", "VCAM-1", "VEGF", "VEGF R1", "VEGF R2", "VEGF R3", "VEGF-C",
"VEGF-D", "XEDAR")

# Factor the data frame to plot in the correct order:
graphable.data$Protein = factor(graphable.data$Protein, levels =
protein.names.ordered.200)


# Order the selected proteins as well
selected.immune.proteins.order =
  c("ALCAM", "Angiogenin", "Axl", "bFGF", "Cathepsin S", "CD14", "CEACAM-1",
"Contactin-2", "CXCL16", "Dtk", "EG-VEGF",
"EGF R", "ErbB3", "Fas", "Fcg RIIBC", "Flt-3L", "GDF-15", "GDNF", "gp130",
"HCC-1", "HGF", "HVEM",
"I-TAC", "ICAM-1", "ICAM-2", "IGFBP-1", "IGFBP-2", "IGFBP-6", "IL-1a", "IL-
1ra", "IL-2 Rb", "IL-2 Rg", "IL-4", "IL-6R",
"IL-8", "IL-9", "IL-11", "IL-13 R1", "IL-13 R2", "IL-16", "LAP(TGFb1)",
"LIF", "LIMPII", "Lipocalin-2",
"LYVE-1", "MCP-1", "MCSF", "MCSF R", "MICA", "NAP-2", "NrCAM", "OPN", "PARC",
"PDGF-AA", "PDGF-AB",
"PDGF-BB", "PECAM-1", "PF4", "RANTES", "Resistin", "SCF R", "Siglec-5",
"TGFa", "TIMP-1", "TIMP-2", "TNF RI",
"TNF RII", "TNFa", "TRAIL R3", "Trappin-2", "uPAR", "VEGF R1")
```

Use ggplot to visualize the lifespan trajectory the 200 neuroimmune and inflammatory proteins. Run the code once with raw data and then another time with the substituted data and LOD and MAX values lines. See supplementary file for output. Sections A and B respectively.

```
# Set cols as a vector of the colours that we will use for each Label in the
plot
cols = c("Raw" = "black", "LOD" = "blue", "Lowest" = "red", "Max" = "purple")


# Create the plots
raw.loess.curves = ggplot(graphable.data, aes(x = Years, y = Expression)) +
  geom_hline(aes(yintercept = LOD, linetype = "LOD"))+
  geom_hline(aes(yintercept = MAX, linetype = "MAX"))+
  scale_x_continuous(trans = "log2", breaks = c(0.3, 1, 5, 10, 25, 50, 80)) +
  geom_point(aes(color = Label), size = 1.5, shape = 19) +
  scale_color_manual(values = cols, name = "Expression Label") +
  facet_wrap(~Protein, ncol = 8, scales = "free_y") +
  stat_smooth(method = "loess", color = "black") +
  labs(title = "Immune Proteins - Raw Data", x = "Age (Years)", y =
"Expression (pg/ml)") +
  theme_bw() +
  theme(panel.background = element_rect(fill = 'gray95'),
        axis.text.x = element_text(angle=45,vjust=0.5),
        axis.title.y = element_text(size = 17),
        axis.title.x =element_text(size = 17),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        plot.title = element_text(hjust = 0.5, size = 20))



# Save them as a pdf
# pdf("LOESS Curves with LOD and MAX Labels for 200 Immune Proteins - Raw
Data.pdf")
# Use facet_multiple() so arrange the plots over multiple pages
# raw.facet.multiple.loess.curves =
#   facet_multiple(plot = raw.loess.curves,
#                          facets = 'Protein',
#                          ncol = 2,
#                          nrow = 2,
#                          scales = "free")
# Stop saving to pdf
# dev.off()
```

How many trajectory patterns are there? Using Ward.D2 clustering to group lifespan trajectories.

While a dynamic range of trajectories can be observed, some patterns are shared amongst multiple immune proteins. In other words, we do not see one unique "neuroimmune trajectory" but we also do not observe 72 unique trajectories. In order to determine the number of lifespan profiles observed in the human V1 we performed hierarchical clustering.

If we cluster based on expression values, it will segregate clusters based on the magnitude of expression. To avoid this, we will scale the data by calculating z-scores on a protein by protein basis.

```
# Create a data frame to store the z-score values, with the same dimensions
and column names as the log2 expression data
zscore.72.proteins <- as.data.frame(matrix(ncol = ncol(log2.exp.72.proteins),
nrow = nrow(log2.exp.72.proteins)))
colnames(zscore.72.proteins) = colnames(log2.exp.72.proteins)

# Calculate zscore on protein by protein basis
for (protein.index in 1:ncol(log2.exp.72.proteins)) {
  zscore.72.proteins[protein.index] =
zscore(log2.exp.72.proteins[,protein.index])
}
```

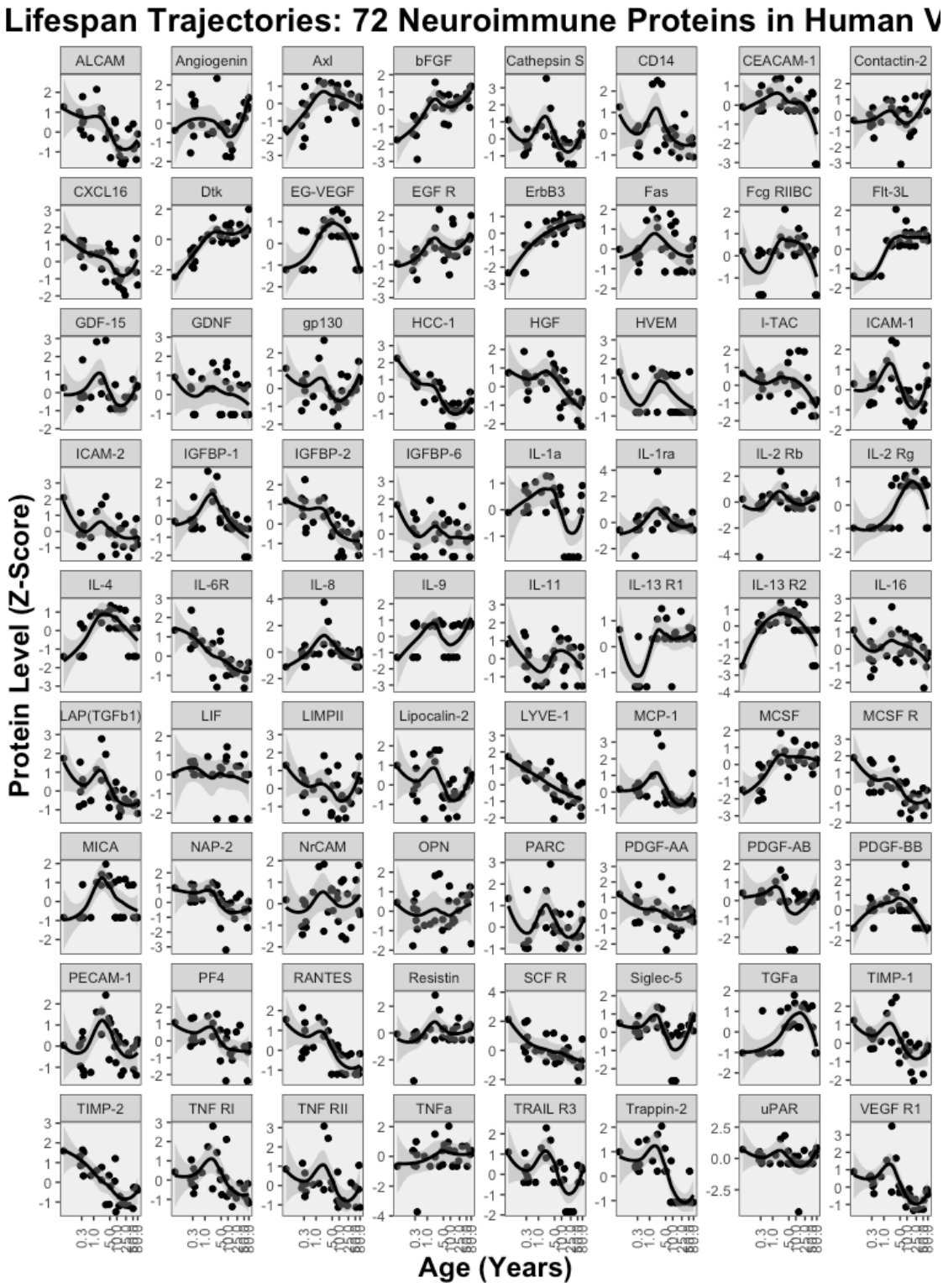
To determine the number of lifespan trajectories observed in our data set, we want to cluster the LOESS curves, not the individual 30 expression points (which may contain outliers and noise).

Plot the LOESS Curves for all 72 selected immune proteins. At the same time the loess values will be exported from the ggplot function and stored in the global environment.

This is FIGURE 5 in my thesis.

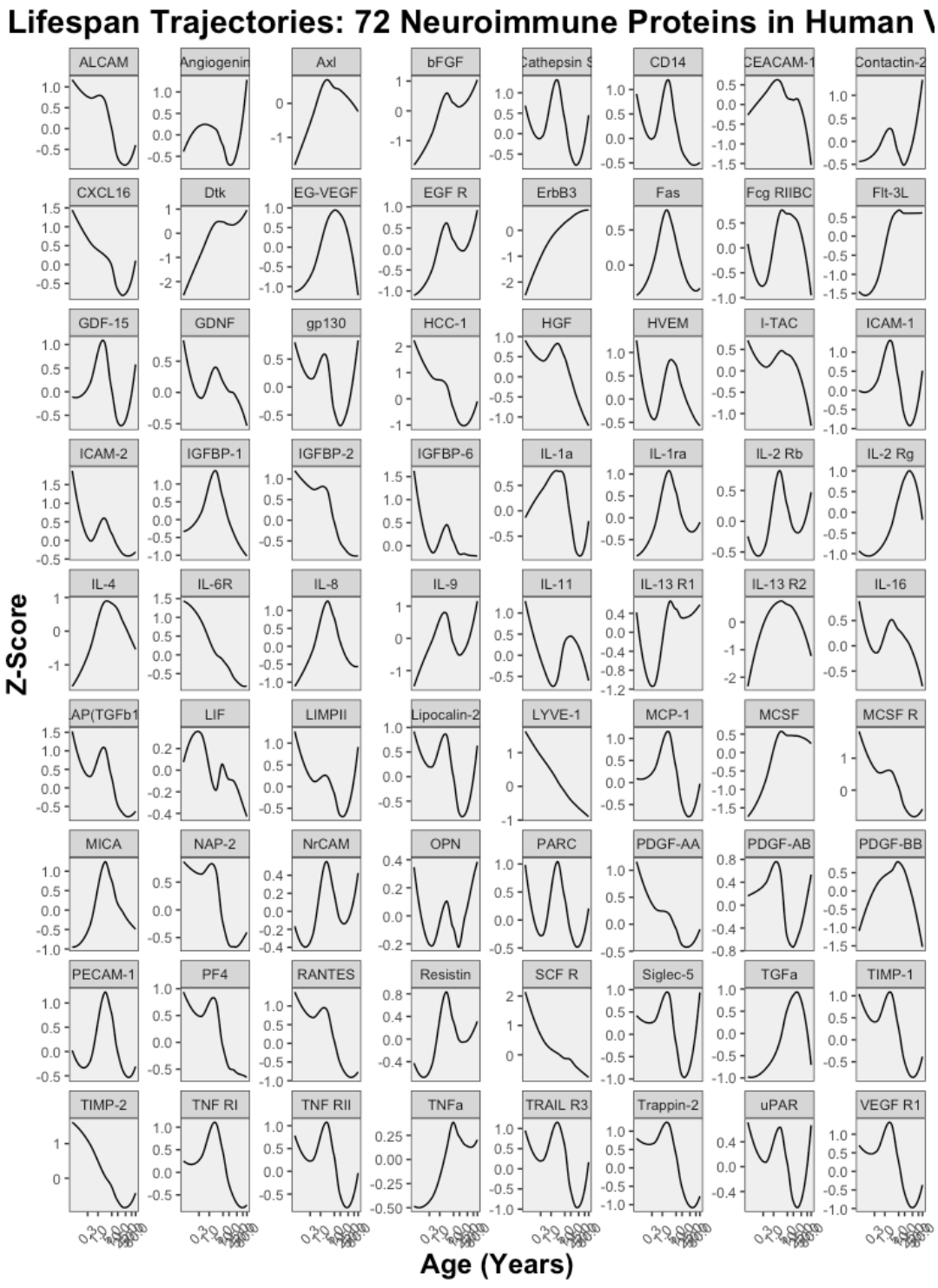
```
plot.data = cbind(identifiers$Years, zscore.72.proteins) # bind the z-score
values and the sample ages
colnames(plot.data)[1] = "Years" # reminder that the age must be in "Years",
remane column accordingly

  row.names(plot.data) <- NULL # set the row names simply the row numbers
  plot.data <- melt(plot.data, id = "Years") # create a long format data
frame

  # Set the column names for the data frame
  colnames(plot.data)[1] = "Years"
  colnames(plot.data)[2] = "Protein"
  colnames(plot.data)[3] = "Expression"

  # Ensure that the Age(Years) and Expression values are stored as numeric
```

```
variables and that the Protein names are characters
  plot.data$Years = as.numeric(plot.data$Years)
  plot.data$Protein = as.character(plot.data$Protein)
  plot.data$Expression = as.numeric(plot.data$Expression)

  # Need to reorder the proteins, as the immune protein names are being
organized alphabetically alone while, I want to order them alphabetically and
then numerically.
  plot.data$Protein = factor(plot.data$Protein, levels =
selected.immune.proteins.order)


  # Do not run the code further if there are any NAs
  if (anyNA(plot.data$Expression) == TRUE) {
    stop("Incomplete Data Set. NAs Present")
  }


# Plot the LOESS curves and export the loess values with 95% CI
ggplots = ggplot(plot.data, aes(Years, Expression))+
    scale_x_continuous(trans='log2',breaks = c(0.3, 1, 5, 10, 25, 50, 80)) +
# log2 transforms the x axis scale
    geom_point(size=1.5) +
    facet_wrap(~Protein, ncol = 8, scales = "free_y")+
    stat_smooth(method = "loess", colour = "black") +
    labs(x = "Age (Years)", y = "Protein Level (Z-Score)") +
    ggtitle("Lifespan Trajectories: 72 Neuroimmune Proteins in Human V1C")+
    theme_bw()+
    theme(panel.background = element_rect(fill = 'gray95'),
          axis.text.x = element_text(angle= 90,vjust=0.5),
          axis.title.y = element_text(size = 17, face = "bold"),
          axis.title.x =element_text(size = 17,face = "bold"),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          plot.title = element_text(hjust = 0.5, size = 20, face = "bold"))
ggplot.loess.export = ggplot_build(ggplots)$data


# Use facet_multiple() so arrange the plots over multiple pages
loess.72.stamp.collection = facet_multiple(plot = ggplots,
                        facets = 'Protein',
                        ncol = 8,
                        nrow = 9,
                        scales = "free_y")
# ggsave("LOESS Curves for 72 Immune Proteins Stamp Collection.png", device =
"png", width = 12, height = 15)


 loess.72.stamp.collection
```


Lifespan Trajectories: 72 Neuroimmune Proteins in Human V
ALCAM
Angiogenin
Axl
bFGF
Cathepsin S
CD14
CEACAM-1
Contactin-2
CXCL16
Dtk
EG-VEGF
EGF R
ErbB3
Fas
Fcg RIIBC
Flt-3L
GDF-15
GDNF
gp130
HCC-1
HGF
HVEM
I-TAC
ICAM-1
ICAM-2
IGFBP-1
IGFBP-2
IGFBP-6
IL-1a
IL-1ra
IL-2 Rb
IL-2 Rg
IL-4
IL-6R
IL-8
IL-9
IL-11
IL-13 R1
IL-13 R2
IL-16
LAP(TGFb1)
LIF
LIMPII
Lipocalin-2
LYVE-1
MCP-1
MCSF
MCSF R
MICA
NAP-2
NrCAM
OPN
PARC
PDGF-AA
PDGF-AB
PDGF-BB
PECAM-1
PF4
RANTES
Resistin
SCF R
Siglec-5
TGFa
TIMP-1
TIMP-2
TNF RI
TNF RII
TNFa
TRAIL R3
Trappin-2
uPAR
VEGF R1
Protein Level (Z-Score)
Age (Years)

Export and plot the loess values directly, to ensure that the values were stored correctly (i.e. values were labelled with the correct protein name)

```
 # Export LOESS Curves
  loess.CI.values = as.data.frame(rep(levels(plot.data$Protein), each = 80))
# create a dataframe with the protein names
  loess.CI.values = cbind(loess.CI.values, ggplot.loess.export[[2]]) # bind
the exported loess values to the correct protein
  colnames(loess.CI.values)[1] = "Protein" # set first column name

  # For plotting purposes, set factor again so that proteins are graphed in
the desired order
  loess.CI.values$Protein = factor(loess.CI.values$Protein,
                                   levels = selected.immune.proteins.order)

  # Exporting LOESS values through ggplot_build exports the x-values in a
log2 transformed format.
  # I want to get the actual values so I convert them back.
  loess.CI.values$x.values = 2^ (loess.CI.values$x)


  # Plot the loess values as a check that the z-score values are being
exported correctly
  gg.loess = ggplot(loess.CI.values, aes(x.values, y))+
    geom_line() +
    scale_x_continuous(trans='log2',breaks = c(0.3, 1, 5, 10, 25, 50,
80))+#transforms the x axis scale
    facet_wrap(~Protein, ncol = 8, scales = "free_y")+
    labs(x = "Age (Years)", y = "Z-Score") +
    ggtitle("Lifespan Trajectories: 72 Neuroimmune Proteins in Human V1C")+
    theme_bw()+
    theme(panel.background = element_rect(fill = 'gray95'),
          axis.text.x = element_text(angle=45,vjust=0.5),
          axis.title.y = element_text(size = 17, face = "bold"),
          axis.title.x =element_text(size = 17, face = "bold"),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          plot.title = element_text(hjust = 0.5, size = 20, face = "bold"))
  # pdf(paste("LOESS Values ", title, ".pdf"))
  stamp.gg.loess = facet_multiple(plot = gg.loess,
                          facets = 'Protein',
                          ncol = 8,
                          nrow = 9,
                          scales = "free_y")
  # dev.off()
 stamp.gg.loess
```



Create a data frame with these loess values in the wide format (for easily processing data in further analyses).

```
loess.72.proteins = dcast(loess.CI.values, formula =  x.values ~ Protein,
value.var="y")
```

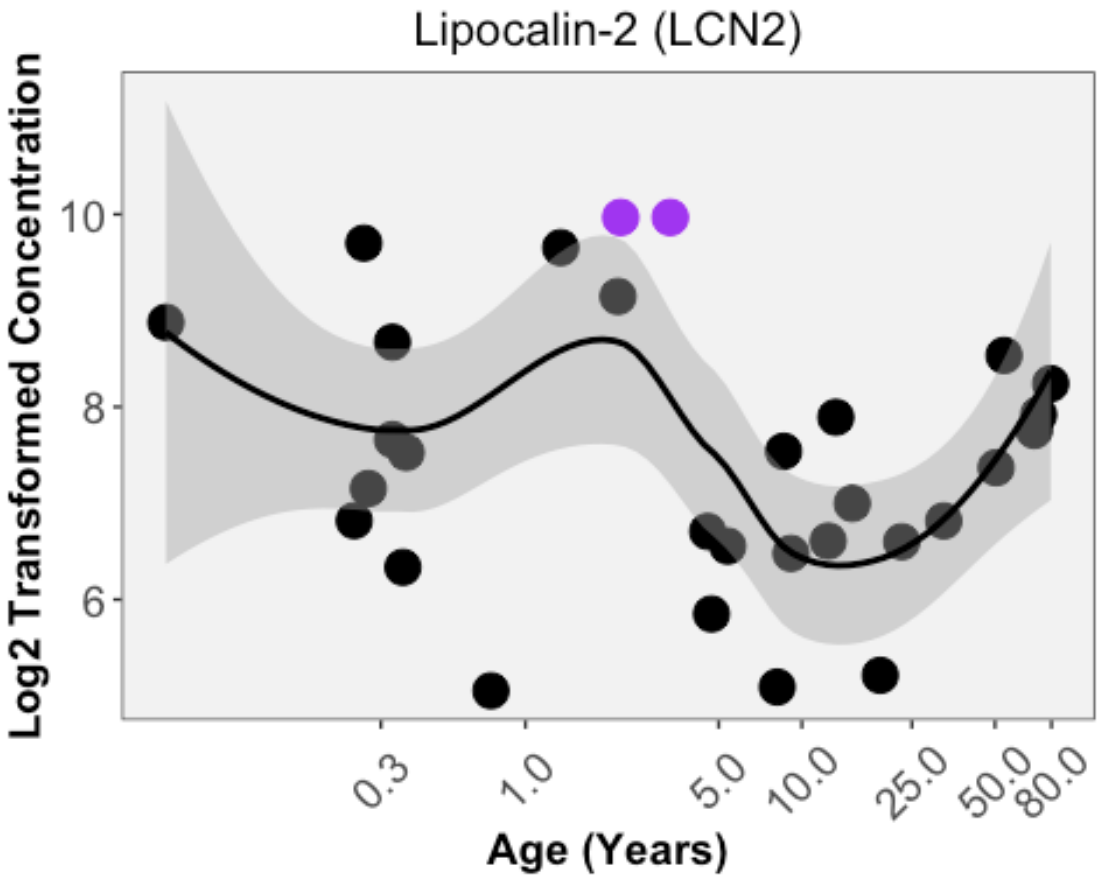
Workflow for Data Transformation

The above steps are all contribute to the workflow used to convert protein measurements in pg/ml to protein trajectories for analyses. Here, I will use Lipocalin-2 as my example and create a figure to encompass my workflow.

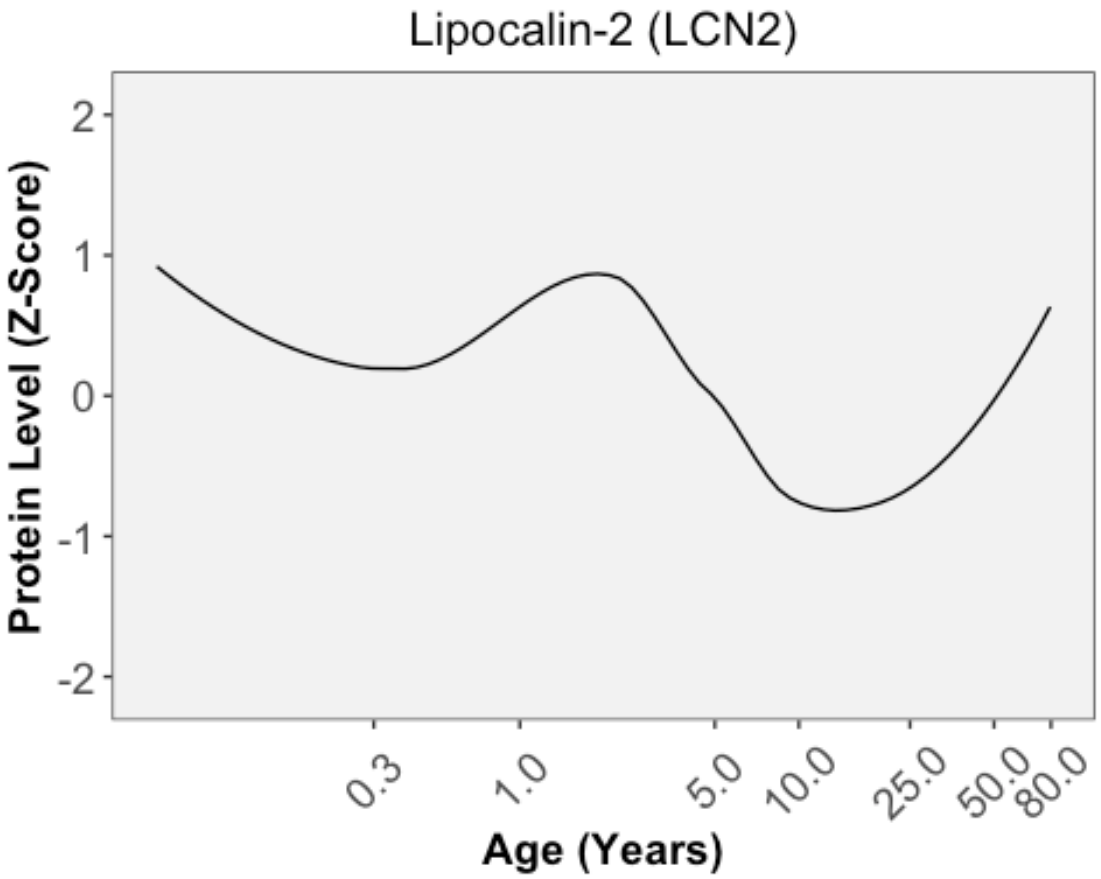
This corresponds to FIGURE 3 in my thesis.

```
# Extract the Lipocalin-2 LOD and MAX Values
LOD.LCN2 = LOD.Max.72.proteins[1, "Lipocalin-2"]
MAX.LCN2 = LOD.Max.72.proteins[2, "Lipocalin-2"]


# Part 1: Raw Expression
workflow = as.numeric(as.character(unlist(identifiers["Years"]))) # Get the
ages
workflow = cbind(workflow, raw.exp.200.proteins["Lipocalin-2"]) # get the
expression
workflow = cbind(workflow, value.classification.2["Lipocalin-2"]) # bind the
classification
colnames(workflow) = c("Years", "Expression", "Class") # set column names

# Part 2: Substituted Values
workflow2 = as.numeric(as.character(unlist(identifiers["Years"])))
workflow2 = cbind(workflow2, corrected.exp.72.proteins["Lipocalin-2"])
workflow2 = cbind(workflow2, value.classification.2["Lipocalin-2"])
colnames(workflow2) = c("Years", "Expression", "Class")

# Part 3: Log2 Expression
workflow3 = as.numeric(as.character(unlist(identifiers["Years"])))
workflow3 = cbind(workflow3, as.data.frame(log2.exp.72.proteins)["Lipocalin-
2"])
workflow3 = cbind(workflow3, value.classification.2["Lipocalin-2"])
colnames(workflow3) = c("Years", "Expression", "Class")

# Part 4: Protein Level (Z-Score)
workflow4 = as.numeric(as.character(unlist(identifiers["Years"])))
workflow4 = cbind(workflow4, zscore.72.proteins["Lipocalin-2"])
workflow4 = cbind(workflow4, value.classification.2["Lipocalin-2"])
colnames(workflow4) = c("Years", "Expression", "Class")

# Part 5: LOESS Curve
workflow5 = loess.72.proteins["x.values"]
workflow5 = cbind(workflow5, loess.72.proteins["Lipocalin-2"])
colnames(workflow5) = c("Years", "Expression")
```

```r
# Use the following code to graph each of the above (from Part 1:4) - must
change labels and titles etc. depending on which graph you want to make
# Uncomment the y-intercepts for MAX and LOD if making figures corresponding
to Part 1 or Part 2
ggplot(workflow3, aes(Years, Expression))+
  scale_x_continuous(trans='log2',breaks = c(0.3,1,5,10,25,50,80))+
#transforms the x axis scale
  geom_point(size= 5, shape=21, aes(color = Class, fill = Class))+
 #scale_y_continuous(limits = c(-300, 2000)) +
 #geom_hline(aes(yintercept = LOD.LCN2, linetype = "LOD"))+
 #geom_hline(aes(yintercept = MAX.LCN2, linetype = "MAX"))+
  scale_color_manual(values = c("black", "purple"),
                     aesthetics = c("colour", "fill"),
                     name = "Measurement",
                     breaks = c("Reliable", "Above MAX"),
                     labels = c("Reliable", "MAX")) +
  stat_smooth(method = "loess", colour = "black") +
  labs(x = "Age (Years)", y = "Log2 Transformed Concentration") +
  ggtitle("Lipocalin-2 (LCN2)")+
  theme_bw()+
  theme(panel.background = element_rect(fill = 'gray95'),
        axis.text.x = element_text(angle=45,vjust=0.5, size = 14),
        axis.text.y = element_text(size = 14),
        axis.title.y = element_text(size = 14, face = "bold"),
        axis.title.x =element_text(size = 14, face = "bold"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        plot.title = element_text(hjust = 0.5, size = 15),
        legend.position = "none")
```




```
# ggsave("Log2 LCN2.png", device = "png", width = 6, height = 4)


# Use the following code to graph Part 5
ggplot(workflow5, aes(Years, Expression))+
  scale_x_continuous(trans='log2',breaks = c(0.3,1,5,10,25,50,80))+
#transforms the x axis scale
  scale_y_continuous(limits = c(-2.1, 2.1)) +
  geom_line() +
  labs(x = "Age (Years)", y = "Protein Level (Z-Score)") +
  ggtitle("Lipocalin-2 (LCN2)")+
  theme_bw()+
  theme(panel.background = element_rect(fill = 'gray95'),
        axis.text.x = element_text(angle=45,vjust=0.5, size = 14),
        axis.text.y = element_text(size = 14),
        axis.title.y = element_text(size = 14, face = "bold"),
        axis.title.x =element_text(size = 14, face = "bold"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        plot.title = element_text(hjust = 0.5, size = 15),
        legend.position = "none")
```

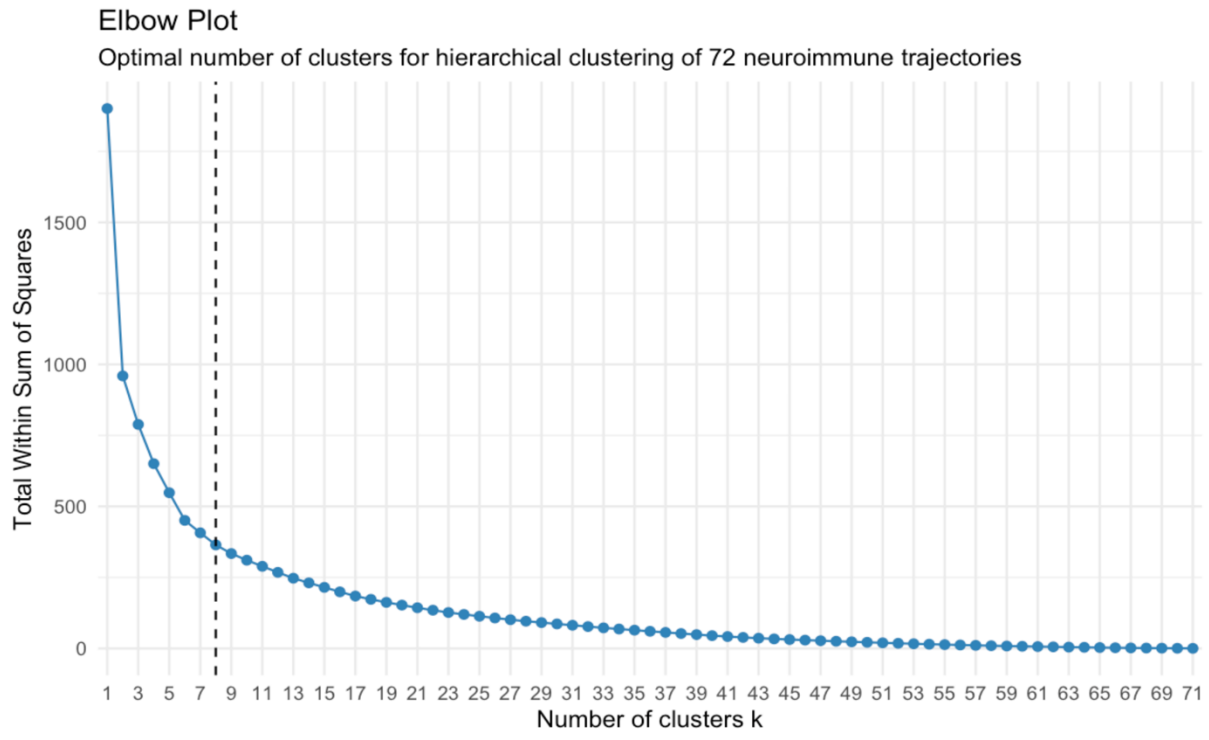



```
# ggsave("LOESS LCN2.png", device = "png", width = 6, height = 4)
```

Determine the number of clusters

To determine the number of clusters I used an elbow plot / WSS plot. This is FIGURE 6 in my thesis.

```
# Format the data frame according to the function specifications

# 1. Input should include only the loess values, do not include identifiers
# 2. The rows are the objects to be clustered

set.seed(123) # set seed for reproducibility
print(fviz_nbclust(t(loess.72.proteins[,2:ncol(loess.72.proteins)]),
FUNcluster = hcut, method = "wss", k.max = 71) +
    scale_x_discrete(breaks = seq(1, 71, 2), labels = seq(1, 71, 2), ) +
    labs(title = "Elbow Plot", subtitle = "Optimal number of clusters for
hierarchical clustering of 72 neuroimmune trajectories",
         y = "Total Within Sum of Squares") +
    geom_vline(xintercept = 8, linetype="dashed", color = "black") +
    theme(axis.text.x = element_text(size = 12)) +
    theme_minimal())
```

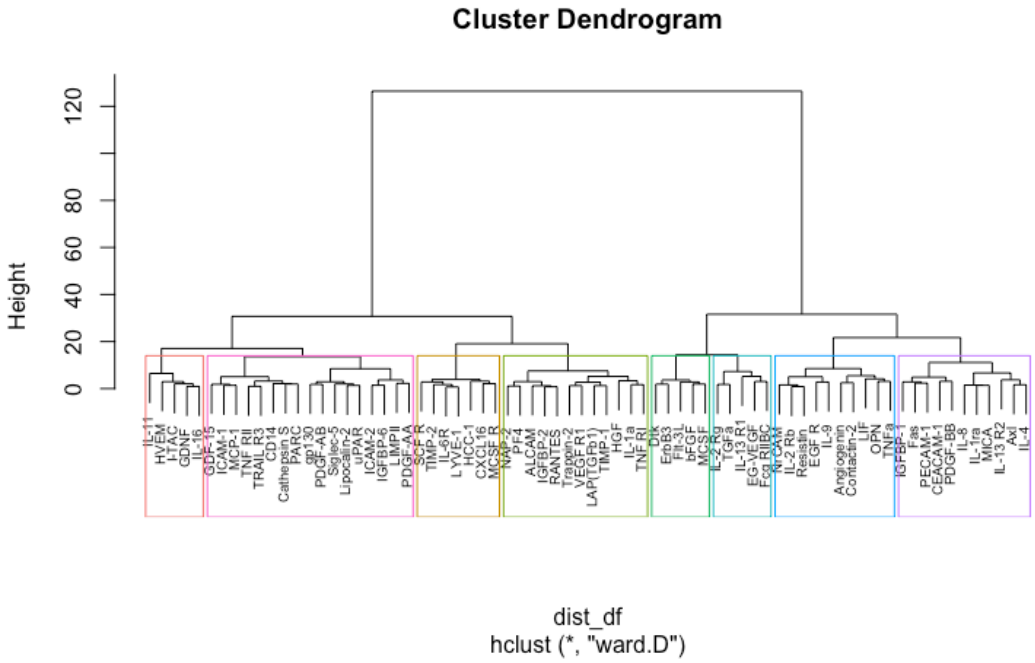


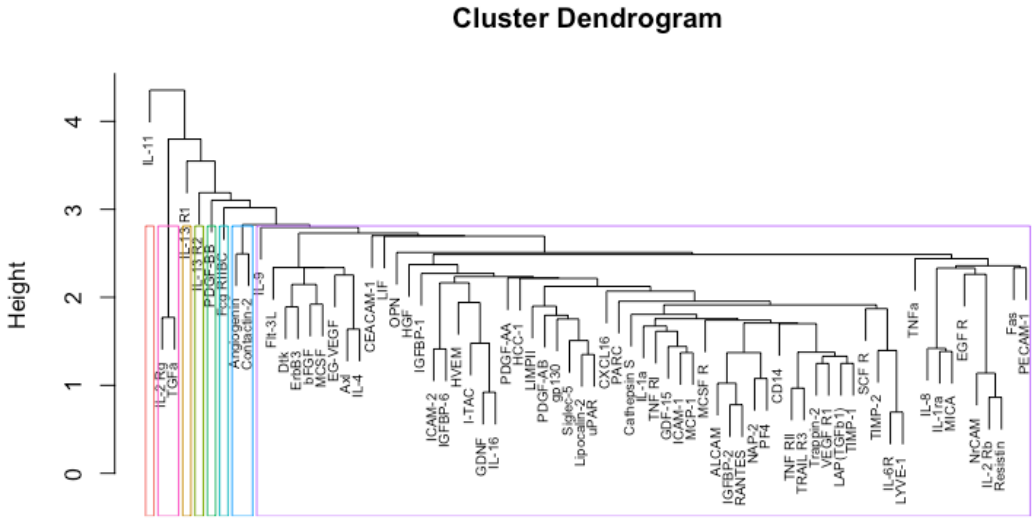

```
# ggsave("Optimal Number of Clusters 72.png", path = "Dropbox (Kathy
Murphy)/EJ RayBiotech Human V1 CB1-Immune/EJ RayBiotech Human V1 CB1-Immune
Images-Figures")
```

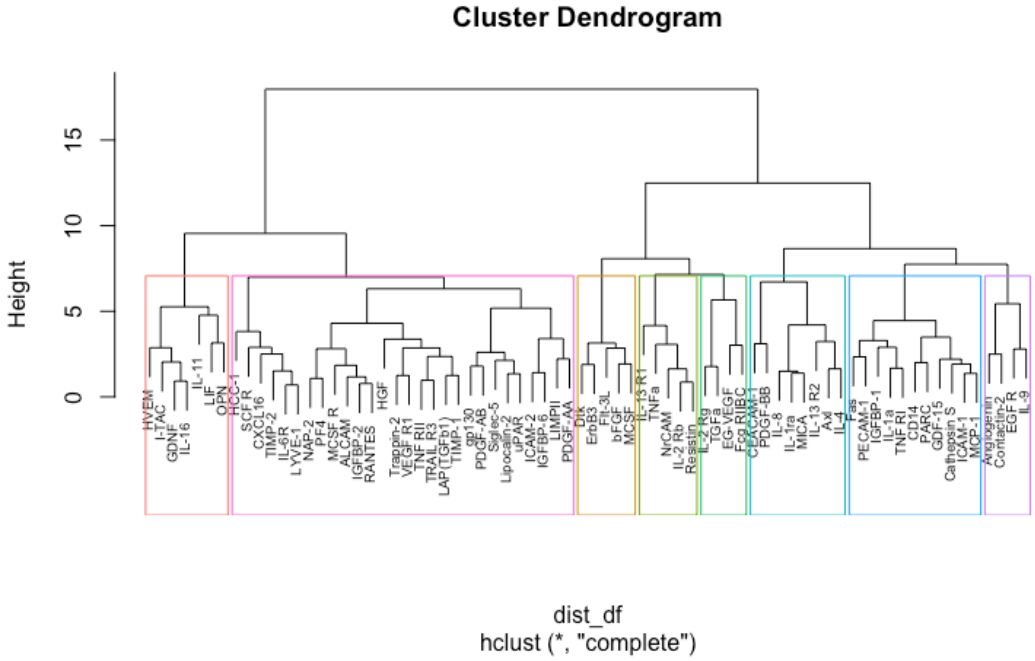
In the plot above, the bend or “elbow” occurs around 8 +/- 2 clusters. The bend signifies the number of clusters beyond which splitting the data has little effect of reducing the variance within a cluster. As a result, the bend is interpreted as the optimal number of clusters.

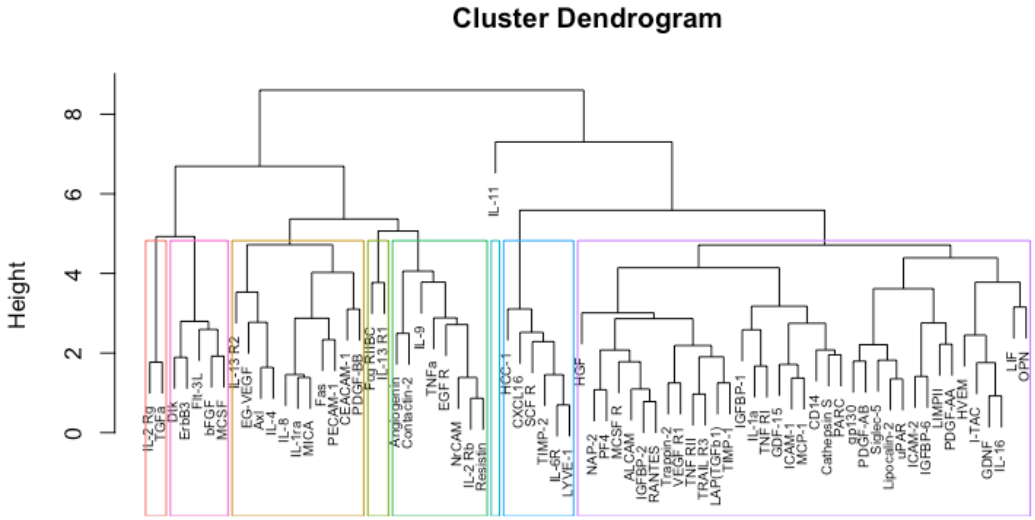
Next, a series of hierarchical clustering methods were tested. The different clusters produced from each method are saved. The last figure produced by this chunk of code is part A of FIGURE 7.

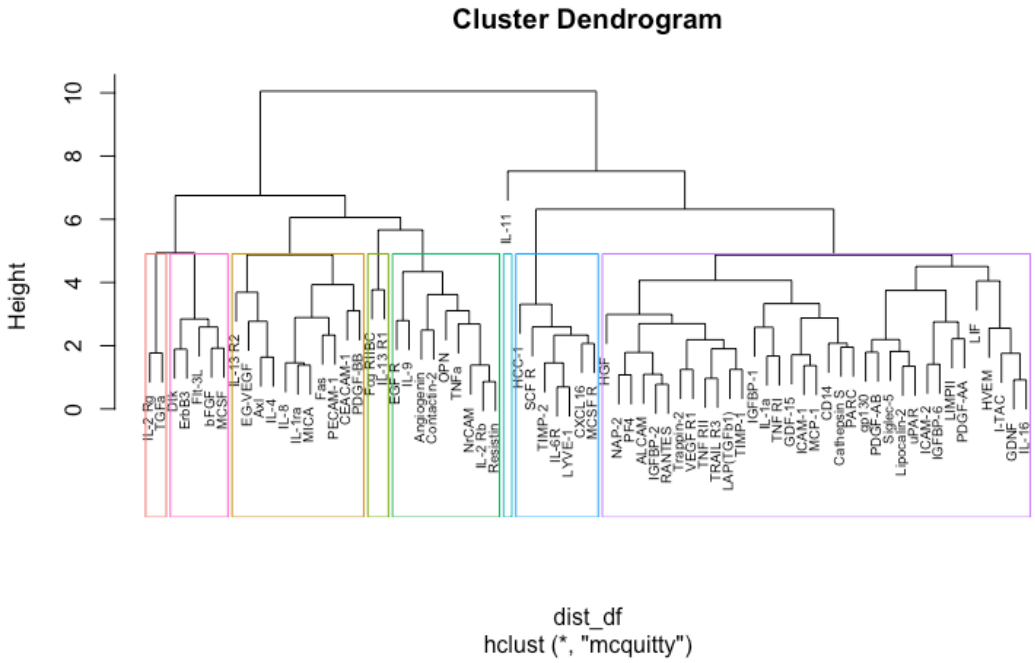
```
set.seed(123)

cluster.number = 8 # set the number of clusters

# Specify hierarchical clustering methods to try
hclust.methods = c("ward.D", "single", "complete", "average",
"mcquitty","median", "centroid", "ward.D2")

# Format the data to be processed by the clustering function
dendrogram.data = as.data.frame(loess.72.proteins[-1] %>% t()) # remember the
rows will be clustered


# Set colour scale for the clusters
cols = c( "#F8766D", "#FF61CC", "#CD9600", "#7CAE00", "#00BE67", "#00BFC4",
"#00A9FF", "#C77CFF")

# pdf("Dendrograms 8 Clusters 72 Immune Proteins.pdf", width = 8, height = 6)
for(i in 1:length(hclust.methods)){
  dist_df = dist(dendrogram.data, method = "euclidean") # get distance matrix
  immune.dendrogram = hclust(dist_df, method = hclust.methods[i]) # cluster
using different hierarchical methods
  split_tree = cutree(immune.dendrogram, k = cluster.number) # the 72
proteins split into their clusters
  name = paste(hclust.methods[i], "cluster.members", sep = ".") # create
names for each cluster (i.e. Cluster 1 etc.)
  assign(name, split(names(split_tree), split_tree)) # assign the names to
the clusters
  plot(immune.dendrogram, cex = 0.6) # plot the dendrogram
  rect.hclust(immune.dendrogram, k = cluster.number, border = cols) # add
rectangles around clusters
}
```

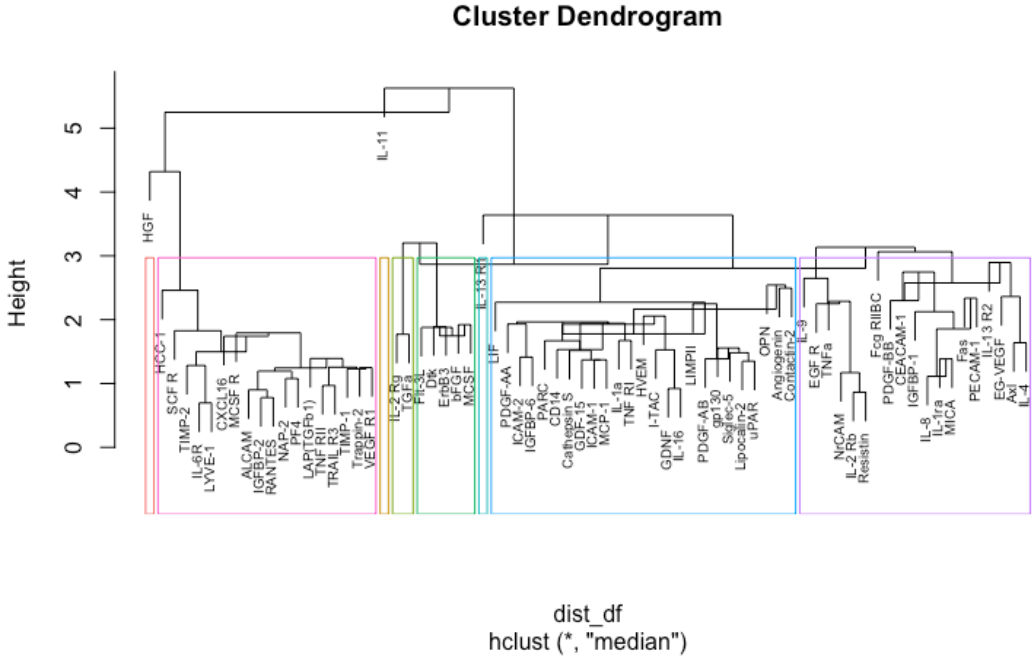

Cluster Dendrogram
Height
0
20
40
60
80
120
IL-11
HVEM
I-TAC
GDNF
IL-16
GDF-15
ICAM-1
MCP-1
TNF RII
TRAIL R3
CD14
Cathepsin S
PARC
gp130
PDGF-AB
Siglec-5
Lipocalin-2
uPAR
ICAM-2
IGFBP-6
LIMPII
PDGF-AA
SCF R
TIMP-2
IL-6R
LYVE-1
HCC-1
CXCL16
MCSF R
NAP-2
PF4
ALCAM
IGFBP-2
RANTES
Tragpin-2
VEGF R1
LAP(TGFb1)
TIMP-1
HGF
IL-1a
TNF RI
Dtk
ErbB3
Flt-3L
bFGF
MCSF
Axl
IL-4
IL-2 Rg
TGFa
IL-13 R1
EG-VEGF
Fcg RIIBC
NrCAM
IL-2 Rb
Resistin
EGF R
IL-9
Angiogenin
Contactin-2
LIF
OPN
TNFa
IGFBP-1
Fas
PECAM-1
CEACAM-1
PDGF-BB
IL-8
IL-1ra
MICA
IL-13 R2
Axl
IL-4
dist_df
hclust (*, "ward.D")

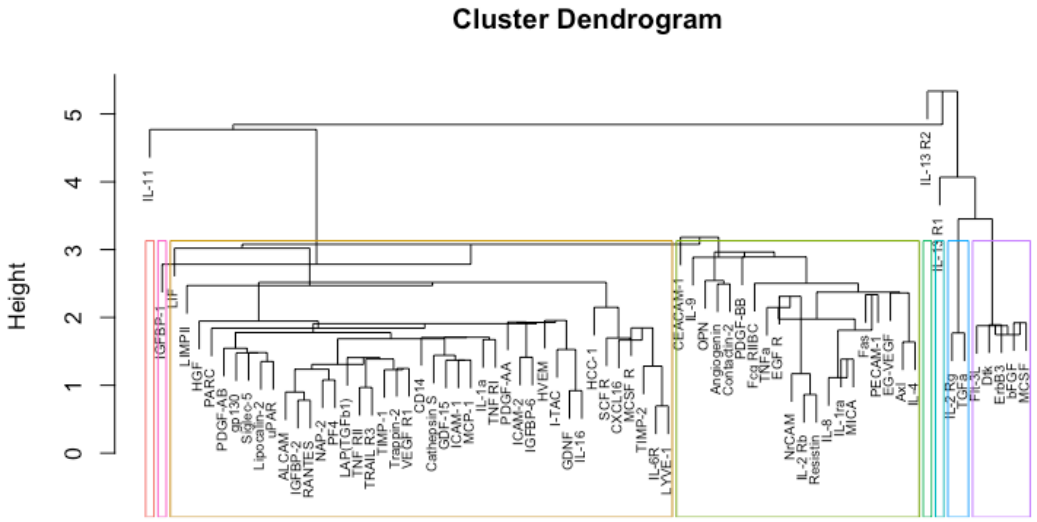

Cluster Dendrogram
Height
0
1
2
3
4
IL-11
IL-2 Rg
TGFa
IL-13 R1
IL-13 R2
PDGF-BB
Fcg RIIBC
Angiogenin
Contactin-2
IL-9
Flt-3L
Dtk
ErbB3
bFGF
MCSF
EG-VEGF
Axl
IL-4
CEACAM-1
LIF
OPN
HGF
IGFBP-1
ICAM-2
IGFBP-6
HVEM
I-TAC
GDNF
IL-16
PDGF-AA
HCC-1
LIMPII
PDGF-AB
gp130
Siglec-5
Lipocalin-2
uPAR
CXCL16
PARC
Cathepsin S
IL-1a
TNF RI
GDF-15
ICAM-1
MCP-1
MCSF R
ALCAM
IGFBP-2
RANTES
NAP-2
PF4
CD14
TNF RII
TRAIL R3
Tragpin-2
VEGF R1
LAP(TGFb1)
TIMP-1
SCF R
TIMP-2
IL-6R
LYVE-1
TNFa
IL-8
IL-1ra
MICA
EGF R
NrCAM
IL-2 Rb
Resistin
Fas
PECAM-1
dist_df
hclust (*, "single")

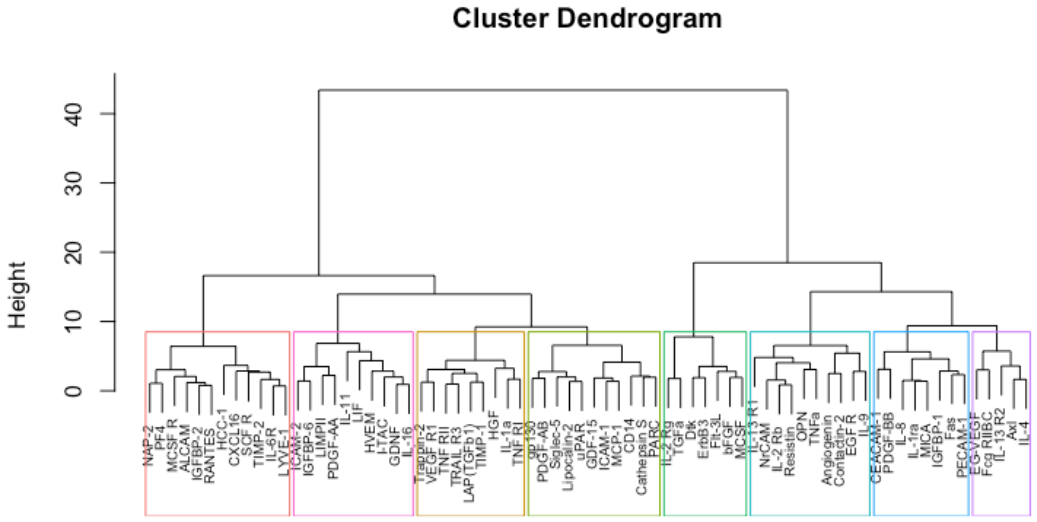

Cluster Dendrogram
Height
0
5
10
15
HVEM
I-TAC
GDNF
IL-16
IL-11
LIF
OPN
HCC-1
SCF R
CXCL16
TIMP-2
IL-6R
LYVE-1
NAP-2
PF4
MCSF R
ALCAM
IGFBP-2
RANTES
HGF
Trappin-2
VEGF R1
TNF RII
TRAIL R3
LAP(TGFb1)
TIMP-1
gp130
PDGF-AB
Siglec-5
Lipocalin-2
cPAR
ICAM-2
IGFBP-6
LIMPII
PDGF-AA
Dtk
ErbB3
FN-3L
bFGF
MCSF
IL-13 R1
TNFa
NrCAM
IL-2 Rb
Reslin
IL-2 Rg
TGFa
EG-VEGF
Fcg RIIBC
CEACAM-1
PDGF-BB
IL-8
IL-1ra
MICA
IL-13 R2
Axl
IL-4
Fas
PECAM-1
IGFBP-1
IL-1a
TNF RI
CD14
PARC
GDF-15
Cathepsin S
ICAM-1
Angiogenin
Contactin-2
EGF R
IL-9
dist_df
hclust (*, "complete")


Cluster Dendrogram
Height
0
2
4
6
8
IL-2 Rg
TGFa
Dtk
ErbB3
FN-3L
bFGF
MCSF
IL-13 R2
EG-VEGF
Axl
IL-4
IL-8
IL-1ra
MICA
Fas
PECAM-1
CEACAM-1
PDGF-BB
Fcg RIIBC
IL-13 R1
Angiogenin
Contactin-2
IL-9
TNFa
EGF R
NrCAM
IL-2 Rb
Reslin
IL-11
HCC-1
CXCL16
SCF R
TIMP-2
IL-6R
LYVE-1
HGF
NAP-2
PF4
MCSF R
ALCAM
IGFBP-2
RANTES
Trappin-2
VEGF R1
TNF RII
TRAIL R3
LAP(TGFb1)
TIMP-1
IGFBP-1
IL-1a
TNF RI
GDF-15
ICAM-1
MCP-1
CD14
Cathepsin S
PARC
gp130
PDGF-AB
Siglec-5
Lipocalin-2
cPAR
ICAM-2
IGFBP-6
LIMPII
PDGF-AA
HVEM
I-TAC
GDNF
IL-16
LIF
OPN
dist_df
hclust (*, "average")


Cluster Dendrogram
Height
0
2
4
6
8
10
IL-2 Rg
TGFa
Dtk
ErbB3
Flt-3L
bFGF
MCSF
IL-13 R2
EG-VEGF
Axi
IL-4
IL-8
IL-1ra
MICA
Fas
PECAM-1
CEACAM-1
PDGF-BB
Fcg RIIBC
EGF R
IL-13 R1
IL-9
Angiogenin
Contactin-2
OPN
TNFa
NrCAM
IL-2 Rb
Resistin
IL-11
HCC-1
SCF R
TIMP-2
IL-6R
LYVE-1
CXCL16
MCSF R
HGF
NAP-2
PF4
ALCAM
IGFBP-2
RANTES
Trappin-2
VEGF R1
TNF RII
TRAIL R3
LAP(TGFb1)
TIMP-1
IGFBP-1
IL-1a
TNF RI
GDF-15
ICAM-1
MCP-1
CD14
Cathepsin S
PARC
gp130
PDGF-AB
Siglec-5
Lipocalin-2
uPAR
ICAM-2
IGFBP-6
LIMPII
PDGF-AA
LIF
I-TAC
GDNF
IL-16
HVEM
dist_df
hclust (*, "mcquitty")


Cluster Dendrogram
Height
0
1
2
3
4
5
HGF
IL-11
HCC-1
SCF R
TIMP-2
IL-6R
LYVE-1
CXCL16
MCSF R
ALCAM
IGFBP-2
RANTES
NAP-2
PF4
LAP(TGFb1)
TNF RII
TRAIL R3
TIMP-1
Trappin-2
VEGF R1
IL-2 Rg
TGFa
Flt-3L
Dtk
ErbB3
bFGF
MCSF
IL-13 Rb
LIF
PDGF-AA
ICAM-2
IGFBP-6
PARC
CD14
Cathepsin S
GDF-15
ICAM-1
MCP-1
IL-1a
TNF RI
HVEM
I-TAC
GDNF
IL-16
LIMPII
PDGF-AB
gp130
Siglec-5
Lipocalin-2
uPAR
OPN
Angiogenin
Contactin-2
IL-9
EGF R
TNFa
NrCAM
IL-2 Rb
Resistin
Fcg RIIBC
PDGF-BB
CEACAM-1
IGFBP-1
IL-8
IL-1ra
MICA
Fas
PECAM-1
IL-13 R2
EG-VEGF
Axi
IL-4
dist_df
hclust (*, "median")


Cluster Dendrogram
Height
0
1
2
3
4
5
IL-11
IL-13 R2
IL-13 R1
IL-13
IGFBP-1
LIF
LIMPII
HGF
PARC
PDGF-AB
gp130
Siglec-5
Lipocalin-2
uPAR
ALCAM
IGFBP-2
RANTES
NAP-2
PF4
LAP(TGFb1)
TNF RII
TRAIL R3
TIMP-1
Trappin-2
VEGF R1
CD14
Cathepsin S
GDF-15
ICAM-1
MCP-1
IL-1a
TNF RI
PDGF-AA
ICAM-2
IGFBP-6
HVEM
I-TAC
GDNF
IL-16
HCC-1
SCF R
CXCL16
MCSF R
TIMP-2
IL-6R
LYVE-1
CEACAM-1
IL-9
OPN
Angiogenin
Contactin-2
PDGF-BB
Fcg RIIBC
TNFa
EGF R
NrCAM
IL-2 Rb
Resistin
IL-8
IL-1ra
MICA
Fas
PECAM-1
EG-VEGF
Axl
IL-4
IL-2 Rg
TGFa
Flt-3L
Dtk
ErbB3
bFGF
MCSF
dist_df
hclust (*, "centroid")


Cluster Dendrogram
Height
0
10
20
30
40
NAP-2
PF4
MCSF R
ALCAM
IGFBP-2
RANTES
HCC-1
CXCL16
SCF R
TIMP-2
IL-6R
LYVE-1
TCAM-2
IGFBP-6
LIMPII
PDGF-AA
IL-11
LIF
HVEM
I-TAC
GDNF
IL-16
Trappin-2
VEGF R1
TNF RII
TRAIL R3
LAP(TGFb1)
TIMP-1
HGF
IL-1a
TNF RI
gp130
PDGF-AB
Siglec-5
Lipocalin-2
uPAR
GDF-15
ICAM-1
MCP-1
CD14
Cathepsin S
PARC
IL-2 Rg
TGFa
Dtk
ErbB3
Flt-3L
bFGF
MCSF
IL-13 R1
NrCAM
IL-2 Rb
Resistin
OPN
TNFa
Angiogenin
Contactin-2
EGF R
IL-9
CEACAM-1
PDGF-BB
IL-8
IL-1ra
MICA
IGFBP-1
Fas
PECAM-1
EG-VEGF
Fcg RIIBC
IL-13 R2
Axl
IL-4
dist_df
hclust (*, "ward.D2")

```
# dev.off()

# png("Neuroimmune Dendrogram - No Extra Labels .png", height = 6, width =
11, units = "in", res = 800)
# plot(immune.dendrogram, cex = 0.5) # plot the dendrogram, add hang = -1
when getting tree only
# rect.hclust(immune.dendrogram, k = cluster.number, border = cols) # add
rectangles around clusters, remove when getting tree only
# dev.off()

# Get the order of proteins in the dendrogram
order.immune.dend = order.dendrogram(as.dendrogram(immune.dendrogram)) # get
the index of the proteins
order.immune.dend = rownames(dendrogram.data)[order.immune.dend] # convert
the index into protein symbols
```

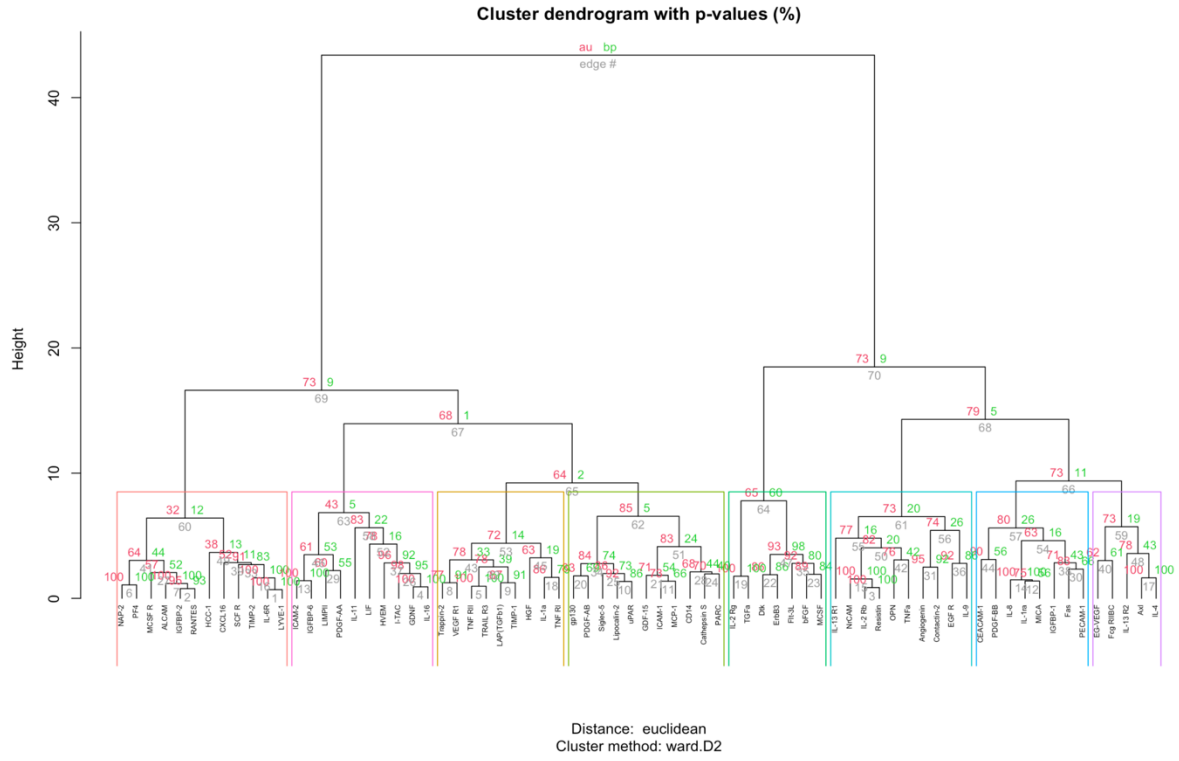
Ward.D2 is a widely used clustering method, it was chosen because of its ability to create compact clusters with minimal within-cluster variance. In addition, while most other hierarchical clustering methods formed clusters with only one member, or contained several small clusters and one large cluster, Ward.D2 formed did not.

Ward.D2 is an agglomerative clustering method. This means that originally each object to be clustered is treated as its own cluster. At each subsequent step of the algorithm, the two clusters with the smallest distance (here, I used Euclidean distance) are grouped together until the predetermined number of clusters is reached. This approach is suitable for our research question, as the goal is to group proteins together by the level of similarity in their trajectories.

P-values were calculated for the Ward.D2 clusters to indicate the robustness of these clusters. The function below returns two types of p-values. The first is an “approximately unbiased” (AU) p-value and the second is bootstrap probability (BP) value. The AU is considered to be the preferred p-value as it is derived using multiscale bootstrap resampling, which limits the bias found in the BP value.

```
# Calculate the p-values associated with the clusters.
set.seed(123)
res.pv = pvclust(loess.72.proteins[-1] , method.hclust = "ward.D2",
                 method.dist = "euclidean", nboot = 10000)

## Bootstrap (r = 0.5)... Done.
## Bootstrap (r = 0.6)... Done.
## Bootstrap (r = 0.7)... Done.
## Bootstrap (r = 0.8)... Done.
## Bootstrap (r = 0.9)... Done.
## Bootstrap (r = 1.0)... Done.
## Bootstrap (r = 1.1)... Done.
## Bootstrap (r = 1.2)... Done.
```

```
## Bootstrap (r = 1.3)... Done.
## Bootstrap (r = 1.4)... Done.

# png(" P-Values 72 Neuroimmune Proteins Dendrogram.png", width = 15, height
= 8, units="in", res=1200)
plot(res.pv, hang = -1, cex = 0.5) # plot the dendrogram with the
corresponding P-Values
rect.hclust(immune.dendrogram, k = 8, border = cols) # Highlight the clusters
```




```
# dev.off()
```

The clusters are automatically stored alphabetically (by the first element in each cluster) in the global environment. To order the clusters in a meaningful way, I will label them A to H, from left to right, as they appear on the dendrogram.

Original labelling of clusters:

```
## List of 8
##  $ 1: chr [1:12] "ALCAM" "CXCL16" "HCC-1" "IGFBP-2" ...
##  $ 2: chr [1:10] "Angiogenin" "Contactin-2" "EGF R" "IL-2 Rb" ...
##  $ 3: chr [1:5] "Axl" "EG-VEGF" "Fcg RIIBC" "IL-4" ...
##  $ 4: chr [1:7] "bFGF" "Dtk" "ErbB3" "Flt-3L" ...
##  $ 5: chr [1:11] "Cathepsin S" "CD14" "GDF-15" "gp130" ...
##  $ 6: chr [1:8] "CEACAM-1" "Fas" "IGFBP-1" "IL-1ra" ...
```

```
##  $ 7: chr [1:10] "GDNF" "HVEM" "I-TAC" "ICAM-2" ...
##  $ 8: chr [1:9] "HGF" "IL-1a" "LAP(TGFb1)" "TIMP-1" ...
```

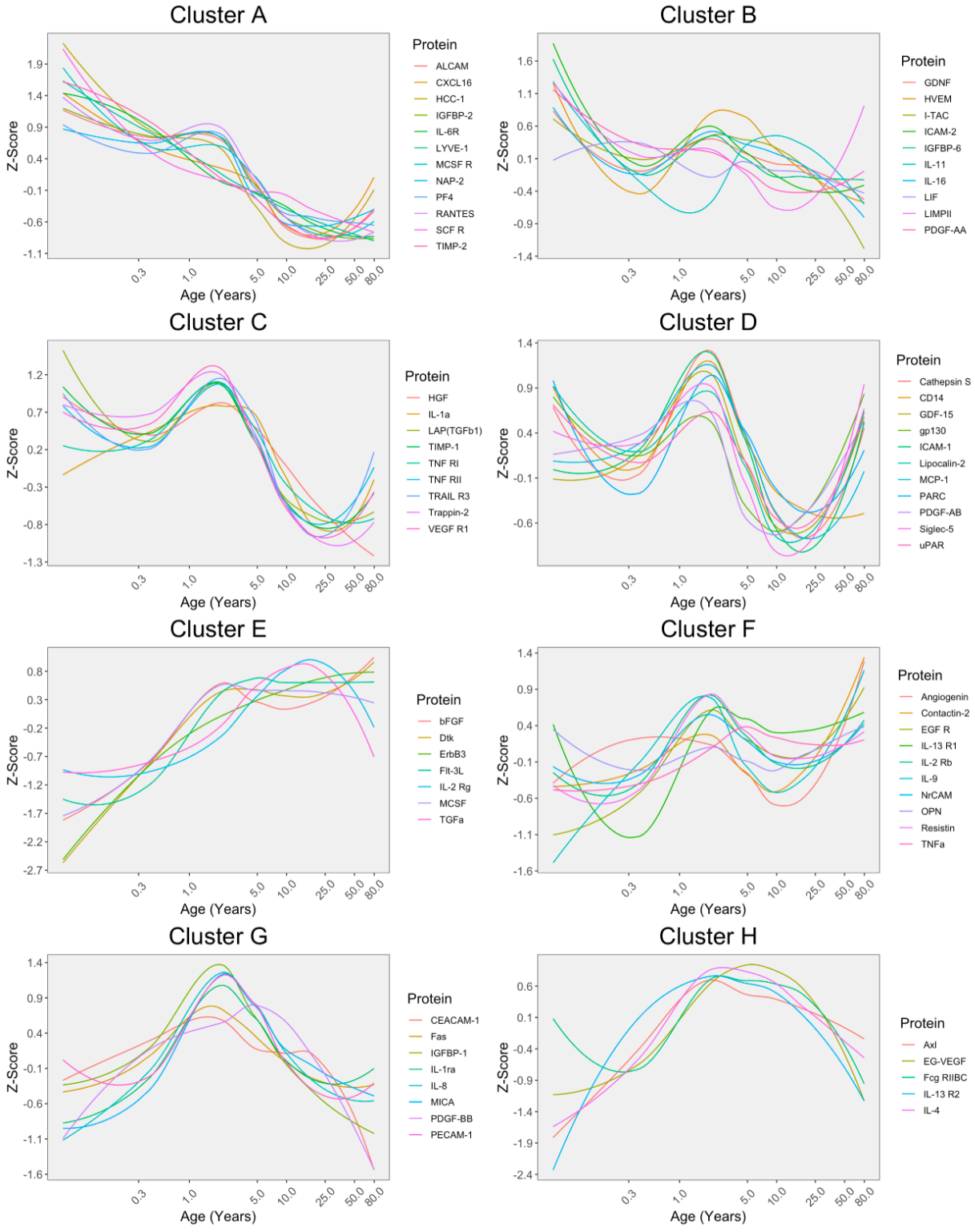
New labelling of clusters:

```
names(ward.D2.cluster.members) = c("A","F","H","E","D","G","B","C")
ward.D2.cluster.members =
ward.D2.cluster.members[order(names(ward.D2.cluster.members))]

## List of 8
##  $ A: chr [1:12] "ALCAM" "CXCL16" "HCC-1" "IGFBP-2" ...
##  $ B: chr [1:10] "GDNF" "HVEM" "I-TAC" "ICAM-2" ...
##  $ C: chr [1:9] "HGF" "IL-1a" "LAP(TGFb1)" "TIMP-1" ...
##  $ D: chr [1:11] "Cathepsin S" "CD14" "GDF-15" "gp130" ...
##  $ E: chr [1:7] "bFGF" "Dtk" "ErbB3" "Flt-3L" ...
##  $ F: chr [1:10] "Angiogenin" "Contactin-2" "EGF R" "IL-2 Rb" ...
##  $ G: chr [1:8] "CEACAM-1" "Fas" "IGFBP-1" "IL-1ra" ...
##  $ H: chr [1:5] "Axl" "EG-VEGF" "Fcg RIIBC" "IL-4" ...
```

Cluster Trajectories - Each Protein Labelled

```
cluster.labels = LETTERS[1:8] # create cluster labels from A to H

labelled.plot.list = list() # create a list that will store the plot for each
cluster

for (i in 1:cluster.number){
# extract the LOESS values for the proteins in a specific cluster
ward.d2.clustering.data =
cbind(loess.72.proteins$x,loess.72.proteins[ward.D2.cluster.members[[cluster.
labels[i]]]])
# the column for plotting the x-axis (Age in Years) is called "x"
colnames(ward.d2.clustering.data)[1] = "x"
# convert the data frame to a long format for plotting, make the protein
names a character value
melted.ward.d2.clustering.data = melt(ward.d2.clustering.data, id = "x") %>%
    mutate(variable = as.character(variable))

# Create plots
labelled.plot.list[[i]] = ggplot(melted.ward.d2.clustering.data, aes(x = x, y
= value, colour = variable)) +
                          geom_line() +
                          scale_x_continuous(trans='log2', breaks =
c(0.3,1,5,10,25,50,80))+ #transforms the x axis to log scale
                          # set y breaks and labels that increase by 0.5,
start labelling 0.1 below the min up to 0.1 more than the max value
                          scale_y_continuous(breaks =
seq(round(min(melted.ward.d2.clustering.data$value) - 0.1, 1),

round(max(melted.ward.d2.clustering.data$value) + 0.1, 1),
                                                          by = 0.5)) +
                          labs(x = "Age (Years)", y = "Z-Score", colour =
"Protein") +
                          theme_bw()+
                          theme(panel.background = element_rect(fill =
'gray95'),
                          axis.text.x = element_text(size = 12,
angle=45,vjust=0.5),
                          axis.text.y = element_text(size = 12),
                          axis.title.y = element_text(size = 15),
                          axis.title.x =element_text(size = 15),
                          panel.grid.major = element_blank(),
                          panel.grid.minor = element_blank(),
                          plot.title = element_text(hjust = 0.5, size = 25),
                          legend.text=element_text(size= 10),
                          legend.title = element_text(size = 15)) +
                          ggtitle(paste("Cluster", cluster.labels[i]))
}
```

```r
# Plots have varying sizes due to the varying size of the legend, must make
them all the same.
# Pick the plot with the largest legend size and use that for all plots
# Cannot use smallest legend or the other legends will not fit in the space
g.labelled.C = ggplotGrob(labelled.plot.list[[3]])

# Create a list that stores the plots with the new legend size
labelled.plot.list.new = list()
for (i in 1:cluster.number){
  g = ggplotGrob(labelled.plot.list[[i]])
  g$widths = g.labelled.C$widths
  labelled.plot.list.new[[i]] = g
}

# Plot all the figures on the same page

ggpubr::ggarrange(plotlist = labelled.plot.list.new, ncol = 2, nrow = 4)
```




```
#  ggsave(paste("Neuroimmune Protein Clusters - Proteins Lablled.png", sep =
""), device = "png")
```

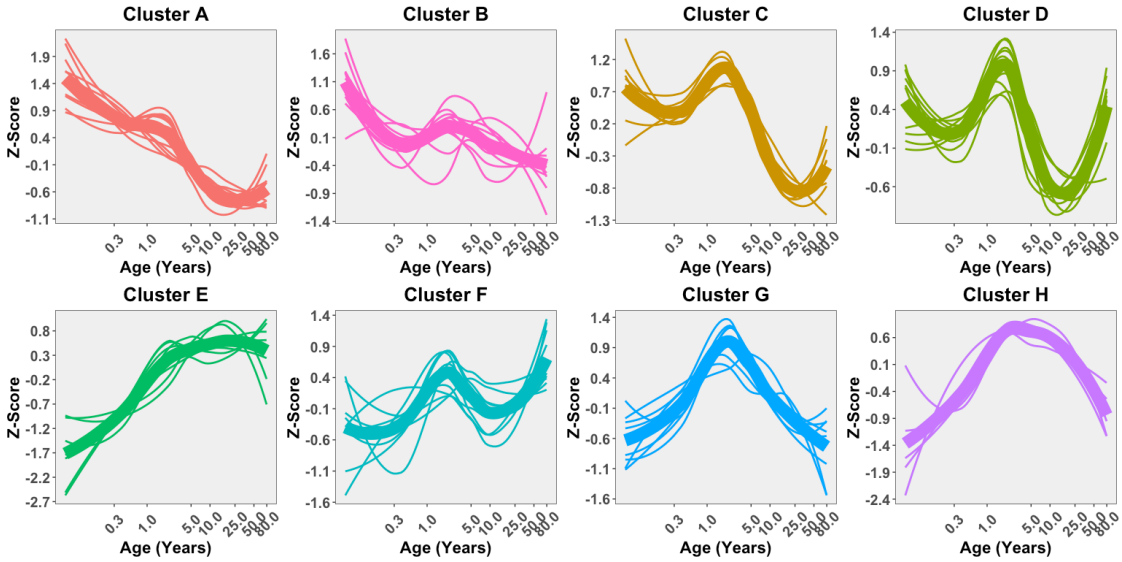
Create Average Curves

Plot Averaged Curves and save the average points as a data frame to the global environment. This is part B of FIGURE 7.

```
# Set colours for plotting.
cols = c( "#F8766D", "#FF61CC", "#CD9600", "#7CAE00", "#00BE67", "#00BFC4",
"#00A9FF", "#C77CFF")

# Create a data frame that contains the average curve values for all clusters
avg.loess.72.data = as.data.frame(matrix(ncol = 9, nrow = 80))
avg.loess.72.data[1] = loess.72.proteins$x # Set the x axis values (ie. the
loess age values)
# Set column names
colnames(avg.loess.72.data) = c("Years","Cluster A","Cluster B","Cluster
C","Cluster D",
                                "Cluster E","Cluster F","Cluster G","Cluster
H")

cluster.number = 8 # specify how many clusters

# Create a list that will hold the upper CI, lower CI and mean values for an
average cluster curve each curve will be its own sub-list
avg.curves.72.stats = list()
for(i in 1:8){
    sub.list = list()
    avg.curves.72.stats = list.append(avg.curves.72.stats, sub.list)
    names(avg.curves.72.stats)[[i]] = paste("Cluster", LETTERS[1:8][i])
}

plot.list = list() # create a list to store the plots

# Loop through each cluster and create an average plot
for (i in 1:cluster.number) {

# Extract the loess values for proteins in a specific cluster
ward.d2.clustering.data =
loess.72.proteins[ward.D2.cluster.members[[cluster.labels[i]]]]

mean.CI = as.data.frame(matrix(ncol = 3, nrow = 80)) # create data frame to
store upper CI, mean and lower CI
colnames(mean.CI) = c("Upper", "Average", "Lower") # set column names
for(row.number in 1:80) { # loop through all the rows and calculate the
upper, mean and lower CI
mean.CI[row.number,] =
CI(as.numeric(unlist(ward.d2.clustering.data[row.number, ])), ci = 0.95) #
calculate Mean, Upper and Lower CI
}
```

```
# Double check mean calculation using another function to verify the row
means
  if (identical(rowMeans(ward.d2.clustering.data), mean.CI$Average) == FALSE)
{
    stop("Means are being calculated incorrectly")
  }

# Add the x values to the data frame
ward.d2.clustering.data$x = loess.72.proteins$x
# Add the mean values to the average cluster data frame
avg.loess.72.data[i+1] = mean.CI$Average
# Add the mean values to the plotting data frame
ward.d2.clustering.data$Average = mean.CI$Average
# melt the data frame and store in separate variable for plotting
melted.ward.d2.clustering.data = reshape2::melt(ward.d2.clustering.data, id =
"x")
# merge the individual protein expression data frame with the upper and lower
CI data frame, mean has already been added
ward.d2.clustering.data = cbind(ward.d2.clustering.data, mean.CI[-2])
# take this data frame and store it in a list (will append all the other
clusters to the same list)
avg.curves.72.stats[[i]] = ward.d2.clustering.data
# plot the average curves using melted data frame
plot.list[[i]] = ggplot(melted.ward.d2.clustering.data, aes(x = x, y = value,
group = variable, size= ifelse(variable=="Average",4,0.1))) +
      geom_line(color=cols[i]) +
      scale_x_continuous(trans='log2',breaks = c(0.3,1,5,10,25,50,80))+ #
transforms the x axis to log scale
      scale_y_continuous(breaks =
seq(round(min(melted.ward.d2.clustering.data$value) - 0.1, 1),

round(max(melted.ward.d2.clustering.data$value) + 0.1, 1),
                                        by = 0.5)) +
      labs(y = "Z-Score", x = "Age (Years)") +
      # labs(y = NULL, x = NULL) + # add line when making plots with no
labels, remove the other
      theme_bw()+
      theme(panel.background = element_rect(fill = 'gray95'),
            axis.text.x = element_text(size = 15, angle = 45, vjust = 0.5,
face = "bold"),
            axis.text.y = element_text(size = 15, face = "bold"),
            axis.title.y = element_text(size = 17, face = "bold"),
            axis.title.x =element_text(size = 17, face = "bold"),
            title = element_text(face = "bold"),
            # axis.ticks = element_blank(), # add these lines when making
plots with no labels, remove the other
            # axis.text.x = element_blank(),
            # axis.text.y = element_blank(),
            legend.position = "none",
            panel.grid.major = element_blank(),
```

```
            panel.grid.minor = element_blank(),
            plot.title = element_text(hjust = 0.5, size = 20))+
      ggtitle(paste("Cluster", cluster.labels[i], sep = " "))
      #  ggtitle(NULL) # add line when making plots with no labels, remove
the other
}
do.call(grid.arrange, c(plot.list, ncol = 4))
```




```
# ggsave("Neuroimmune Average Plot Stamp Collection.png", device = "png",
height = 4, width = 8)



# Use code to get plots with no labels, ex. when adding to heatmap
# for(i in 1:length(plot.list)){
# plot(plot.list[[i]])
# ggsave(paste("Average Neuroimmune Curve no labels Cluster",
LETTERS[1:8][i], ".png"), height = 4, width = 4)
# }
```

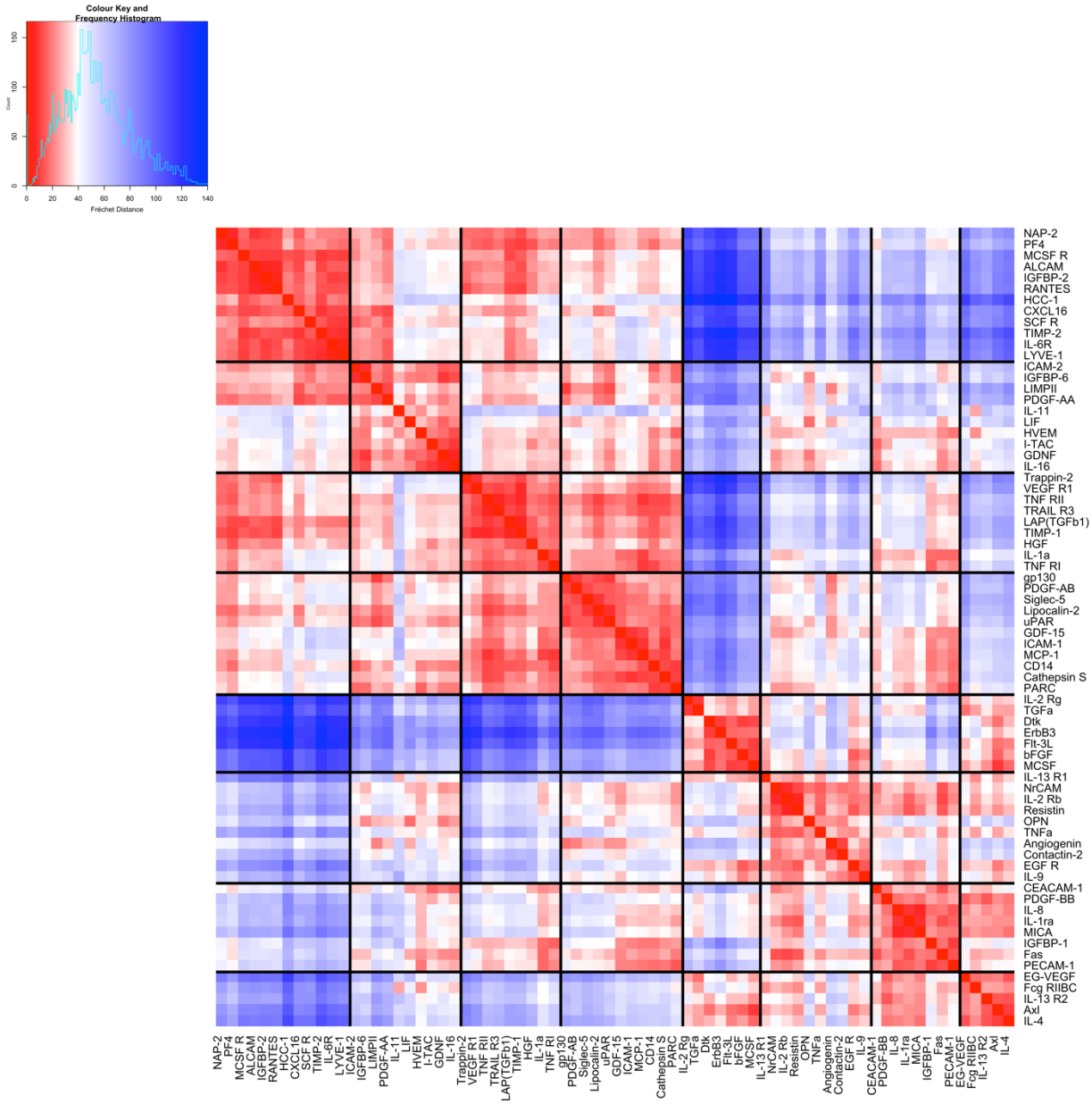
Fréchet Distances: Between Neuroimmune Proteins (72 x 72)

Fréchet distances, a similarity measure for time-series data, was used to assess the 72 immune protein trajectories. The advantages of using Fréchet over a Pearson's correlation is that Fréchet distances take into account both the location and ordering of points along the curves.

This is FIGURE 9 in my thesis.

```
# Create 72 x 72 Matrix

frechet.scores.proteins = expand.grid(colnames(loess.72.proteins[-1]),
colnames(loess.72.proteins[-1])) # get combination of proteins to compare
(including self-comparisons)
frechet.scores.proteins$Distance = NA # create empty column to store the
distances
colnames(frechet.scores.proteins)[1:2] = c("Variable 1", "Variable 2")
x.value = loess.72.proteins$x # set the x values (ages) for the trajectories
(here my proteins were measured in the same samples, so only 1 unique x-value
vector needed)

for(i in 1:nrow(frechet.scores.proteins)){
    trajectory1 =
as.numeric(unlist(loess.72.proteins[as.character(frechet.scores.proteins[i,1]
)])) # export the z-score values for the first protein
    trajectory2 =
as.numeric(unlist(loess.72.proteins[as.character(frechet.scores.proteins[i,2]
)])) # export the z-score values for the second protein
    frechet.scores.proteins[i,3] = as.numeric(distFrechetR(x.value,
trajectory1, x.value, trajectory2, FrechetSumOrMax = "sum")) # calculate
distance, use the sum method
}

# Store the median Frechet distances (without taking self-comparison into
account)
median.72.frechet.no.self = median(subset(frechet.scores.proteins$Distance,
!frechet.scores.proteins$Distance == 0))

# Use the median distance to determine which values should be white on the
heatmap
median.72.frechet = median(frechet.scores.proteins$Distance) # around 38.6 ,
so from 38 - 38.9999 will be white

# Use the max value to determine how high the color map must go (the lowest
value will be 0)

# Convert from long to wide format for plotting
frechet.72.heatmap.data = spread(frechet.scores.proteins, `Variable 1`,
Distance) %>%
```

```
                column_to_rownames("Variable 2")

# set colour scheme for data frame
paletteLength = 99 # indicates how many gradations are wanted in the color
key
col = colorRampPalette(c("red", "white", "blue"))(paletteLength) # make the
color scale
# assign where the breaks must occur
mybreaks = c(seq(0,37,length=49),  # Distances up to 37 are coloured on a red
gradient
             seq(38,39,length=2),  # Distances between 38-39 are coloured
white
             seq(40,135,length=49)) # Distances between 40 and 135 are
coloured using a blue gradient

# reorder the columns and rows by the  hierarchical clustering ward.D2
dendrogram
frechet.72.heatmap.data.2 = frechet.72.heatmap.data[order.immune.dend,
order.immune.dend]

# png("72 Protein Frechet Heatmap.png", height = 20, width = 20, units =
"in", res = 1000)
heatmap.2(as.matrix(frechet.72.heatmap.data.2), # specify data frame
          col = col, # specify colour scale
          breaks = mybreaks, # specify breaks
          Colv = order.immune.dend, # order of columns
          Rowv = order.immune.dend, # order of rows
          dendrogram = "none", # apply default dendrogram (no)
          trace = "none", # do not draw a trace
          sepwidth = c(0.2,0.2), # create lines showing clusters
          sepcolor = "black", # colour of line
          cexRow = 1.5, # font size rows
          cexCol = 1.5, # font size columns
          key.xlab = "Fréchet Distance", # Key x label
          key.title = "Colour Key and\n Frequency Histogram", # Key title
          key.ylab = "Count", # Key y label
          lhei = c(2, 8), # Height of  key and the plot
          lwid = c(2, 8), # Width  of  key and the plot
          colsep=c(12,22,31,42,49,59,67), # Where to draw column lines to
show clusters
          rowsep=c(12,22,31,42,49,59,67), # Where to draw row lines to show
clusters
          margins = c(10,10), # set margins for the plot
          key.xtickfun = function() { # labels for the Key
            breaks = pretty(parent.frame()$breaks)
            list(at = parent.frame()$scale01(breaks),
                 labels = breaks)
          }
          )
```


Colour Key and
Frequency Histogram
Count
Fréchet Distance
NAP-2
PF4
MCSF R
ALCAM
IGFBP-2
RANTES
HCC-1
CXCL16
SCF R
TIMP-2
IL-6R
LYVE-1
ICAM-2
IGFBP-6
LIMPII
PDGF-AA
IL-11
LIF
HVEM
I-TAC
GDNF
IL-16
Trappin-2
VEGF R1
TNF RII
TRAIL R3
LAP(TGFb1)
TIMP-1
HGF
IL-1a
TNF RI
gp130
PDGF-AB
Siglec-5
Lipocalin-2
uPAR
GDF-15
ICAM-1
MCP-1
CD14
Cathepsin S
PARC
IL-2 Rg
TGFa
Dtk
ErbB3
Flt-3L
bFGF
MCSF
IL-13 R1
NrCAM
IL-2 Rb
Resistin
OPN
TNFa
Angiogenin
Contactin-2
EGF R
IL-9
CEACAM-1
PDGF-BB
IL-8
IL-1ra
MICA
IGFBP-1
Fas
PECAM-1
EG-VEGF
Fcg RIIBC
IL-13 R2
Axl
IL-4

```
# dev.off()
```

Fréchet Distances: Within Neuroimmune Clusters

Fréchet distances were also calculated between each pair of trajectories WITHIN a cluster. This allows us to evaluate the level of variance within each cluster.

```
x.value = loess.72.proteins$x # x values for the proteins, all proteins have
the same x values in our data


# Create a list of lists that will store the frechet scores for each cluster
separately
frechet.scores = list()
for(i in 1:8){
    sub.list = list()
    frechet.scores = list.append(frechet.scores, sub.list)
    names(frechet.scores)[[i]] = paste("Cluster",
names(ward.D2.cluster.members)[[i]])
}


  for (i in 1:8) { # Determine all combinations (except self correlations)
    individual.clus = ward.D2.cluster.members[[i]] # get the names of
proteins in each cluster
    combinations = as.data.frame(combn(individual.clus, 2, simplify = TRUE))
# generate pairwise comparison list (no self-comparisons)
    scores.vector = vector() # create an empty vector to store the distance
values for each cluster

    for (a in 1:ncol(combinations)){
      trajectory1 = as.numeric(unlist(loess.72.proteins[combinations[1, a]]))
# numeric vector of the first trajectories z-score values
      trajectory2 = as.numeric(unlist(loess.72.proteins[combinations[2, a]]))
# numeric vector of the first trajectories z-score values
      score = as.numeric(distFrechetR(x.value, trajectory1, x.value,
trajectory2, FrechetSumOrMax = "sum")) # calculate the frechet distance
between them
      scores.vector = append(scores.vector, score) # add the distance to the
scores vector
    }

    frechet.scores[[i]] =  scores.vector # take the scores for this cluster
vector and put it into the frechet.scores
  }


# Save each of the values, one page for each cluster
list_of_datasets = list("Cluster A Distances" = frechet.scores[[1]], "Cluster
B Distances" = frechet.scores[[2]],
                        "Cluster C Distances" = frechet.scores[[3]], "Cluster
```

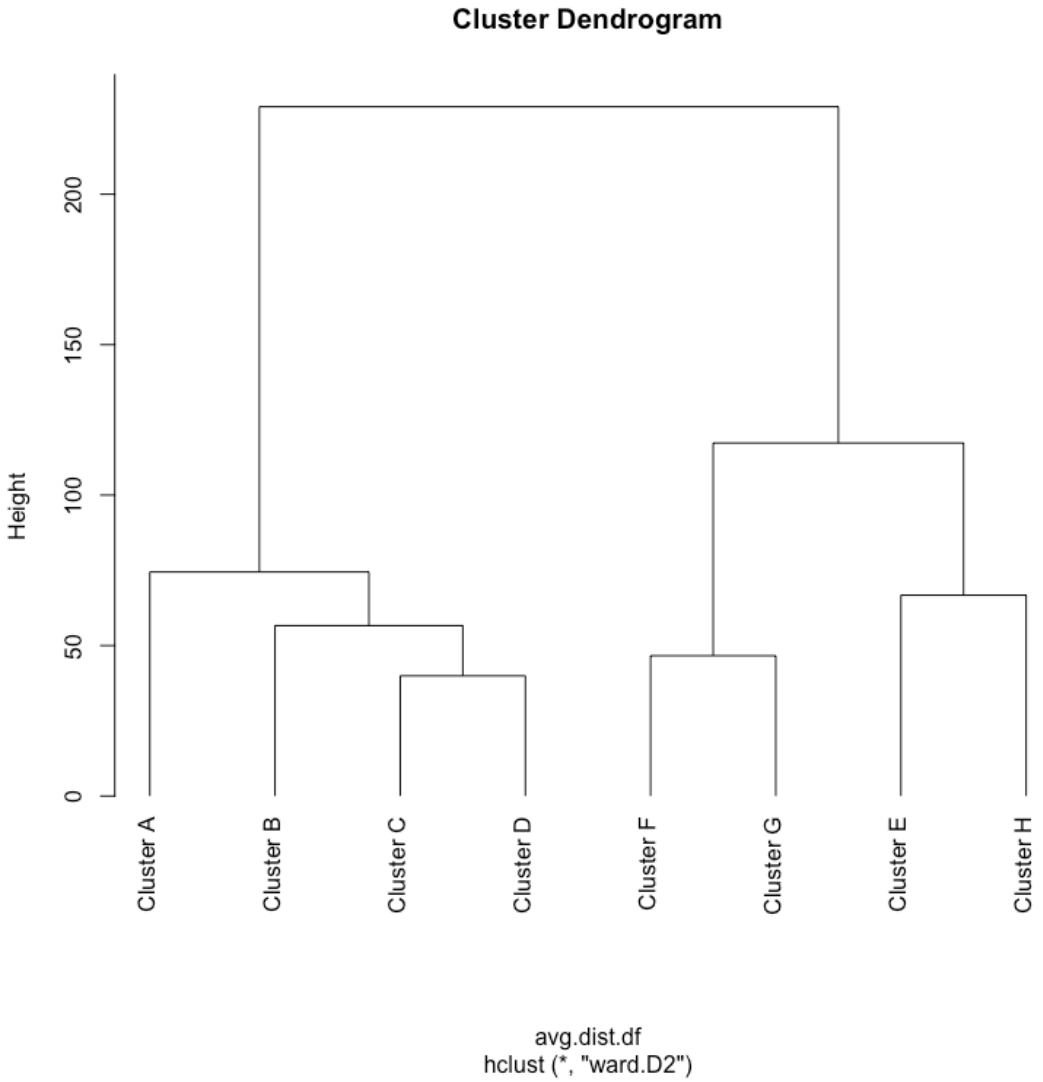
```
D Distances" = frechet.scores[[4]],
                          "Cluster E Distances" = frechet.scores[[5]], "Cluster
F Distances" = frechet.scores[[6]],
                          "Cluster G Distances" = frechet.scores[[7]], "Cluster
H Distances" = frechet.scores[[8]])
# openxlsx::write.xlsx(list_of_datasets, file = "Frechet within cluster for
neuroimmune only.xlsx")


# Create a long format data frame from the list

frechet.boxplot.data = as.data.frame(NA) # create an empty data frame
  for(i in 1:length(frechet.scores)){
    # Add the frechet scores from each cluster as a column to the data frame
    frechet.boxplot.data = qpcR:::cbind.na(frechet.boxplot.data,
as.data.frame(frechet.scores[[i]]))
    # Name each column by the cluster number
    colnames(frechet.boxplot.data)[i+1] = names(frechet.scores)[i]
  }

  frechet.boxplot.data = frechet.boxplot.data[-1] # remove the empty column
  frechet.boxplot.data = melt(frechet.boxplot.data, na.rm = TRUE) # covert
the data frame to the long format for estimation stats
```

Fréchet Distance Between Average Curves (8 x 8)

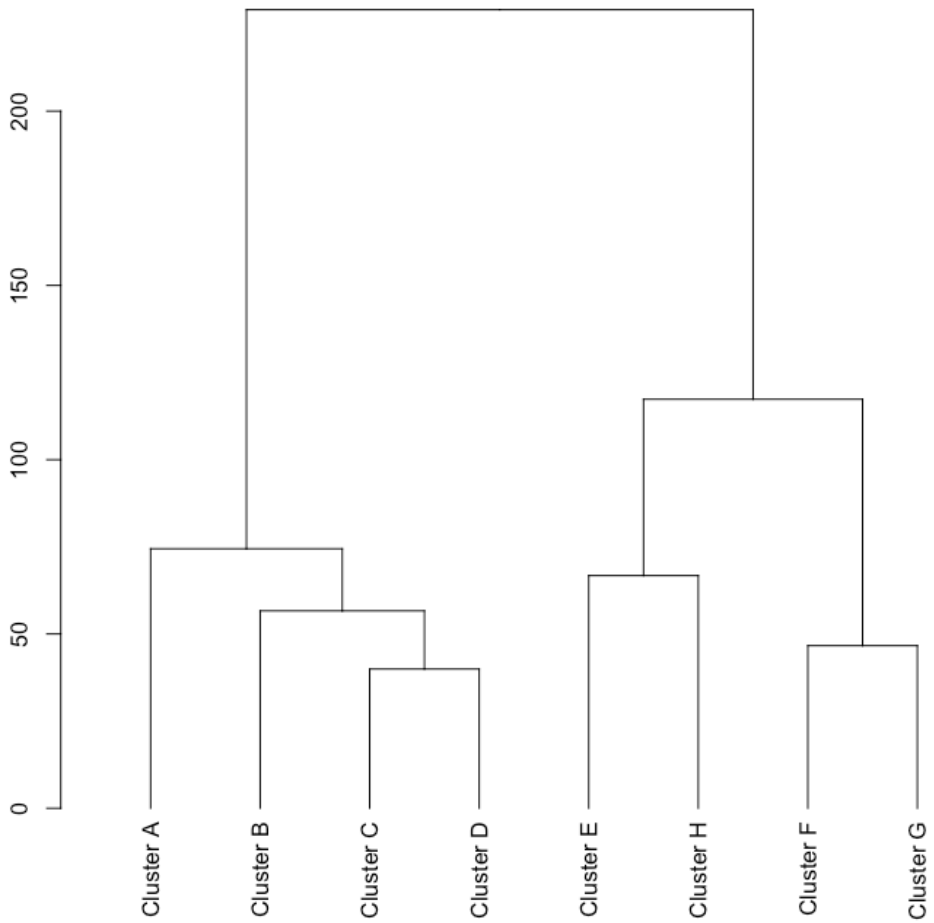
Finally, Fréchet distances were calculated between clusters using the average cluster curves.

```
# Create a list of all combinations (8 x 8)
frechet.scores.avg = expand.grid(colnames(avg.loess.72.data[-1]),
colnames(avg.loess.72.data[-1]))
frechet.scores.avg$Distance = NA # create empty column to store the distances
colnames(frechet.scores.avg)[1:2] = c("Variable 1", "Variable 2")
x.value = avg.loess.72.data$Years # set the x values (ages) for the
trajectories
for(i in 1:nrow(frechet.scores.avg)){
    trajectory1 =
as.numeric(unlist(avg.loess.72.data[as.character(unlist(frechet.scores.avg[i,
1]))])) # export the z-score values for the first protein
    trajectory2 =
as.numeric(unlist(avg.loess.72.data[as.character(unlist(frechet.scores.avg[i,
2]))])) # export the z-score values for the second protein
    frechet.scores.avg[i,3] = as.numeric(distFrechetR(x.value, trajectory1,
x.value, trajectory2, FrechetSumOrMax = "sum")) # use the sum method
}

# Store the median Frechet distances (with and without taking self-
```

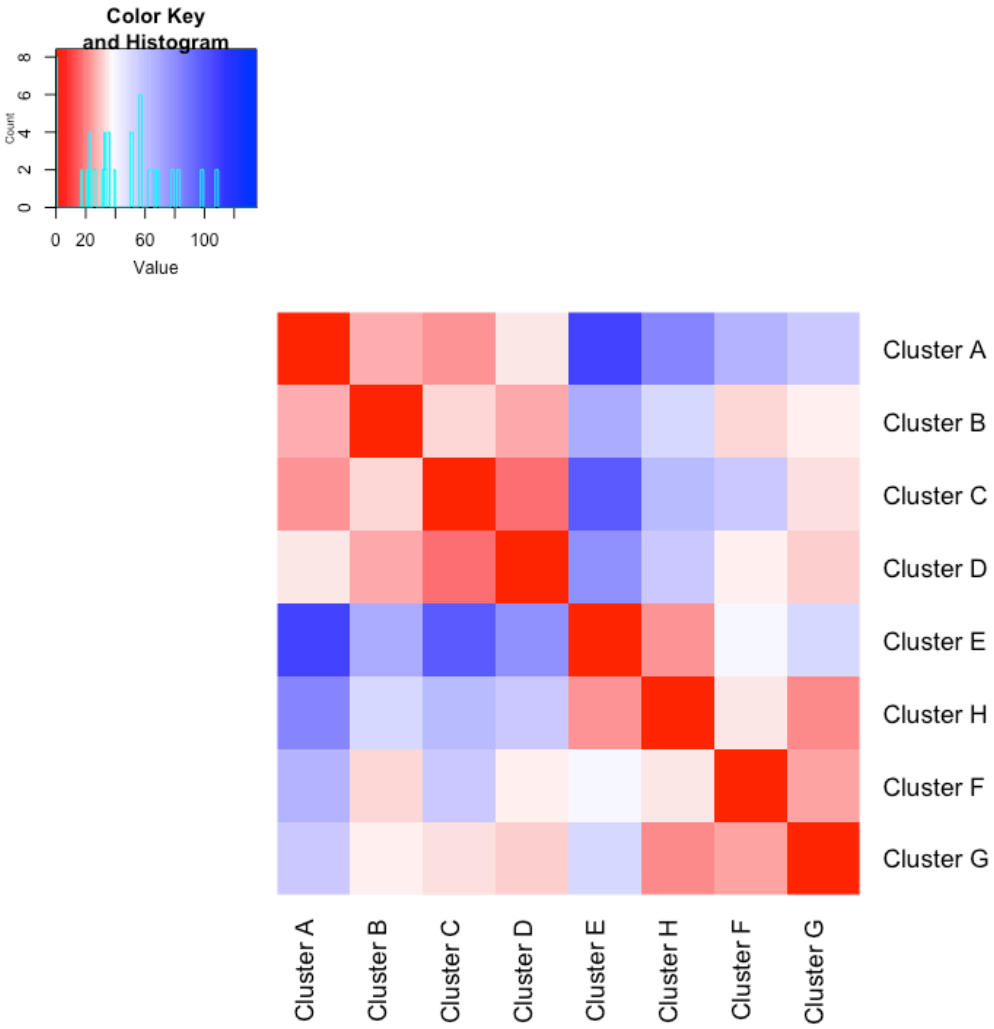
```
correlations into account)
median.avg.frechet = median(frechet.scores.avg$Distance)
median.avg.frechet.no.self = median(subset(frechet.scores.avg$Distance,
!frechet.scores.avg$Distance == 0))


# Convert from long to wide format for plotting
avg.heatmap.data = spread(frechet.scores.avg, `Variable 1`, Distance) %>%
                   column_to_rownames("Variable 2")

# Use the same colour scale for all frechet heatmaps in order to compare
across them.

paletteLength = 99 # indicates how many gradations are wanted in the color
key
col = colorRampPalette(c("red", "white", "blue"))(paletteLength) # make the
color scale
# assign where the breaks must occur
mybreaks = c(seq(0,37,length=49),  # Distances up to 37 are coloured on a red
gradient
             seq(38,39,length=2),  # Distances between 38-39 are coloured
white
             seq(40,135,length=49)) # Distances between 40 and 135 are
coloured using a blue gradient



# Get dendrogram for the clusters
avg.clust.dendrogram.data = as.data.frame(avg.heatmap.data)
avg.dist.df = dist(avg.clust.dendrogram.data, method = "euclidean")
avg.clust.dendrogram = hclust(avg.dist.df, method = "ward.D2")
avg.clust.dend.plot = plot(avg.clust.dendrogram, hang = -1, cex = 1)
```




```
new.order = c("Cluster A", "Cluster B", "Cluster C", "Cluster D",
              "Cluster E", "Cluster H", "Cluster F", "Cluster G")

avg.clust.dendrogram = as.dendrogram(avg.clust.dendrogram)
reordered.avg.clust.dendrogram = avg.clust.dendrogram %>%
  dendextend::rotate(as.character(new.order))

plot(reordered.avg.clust.dendrogram, hang = -1, cex = 1)
```



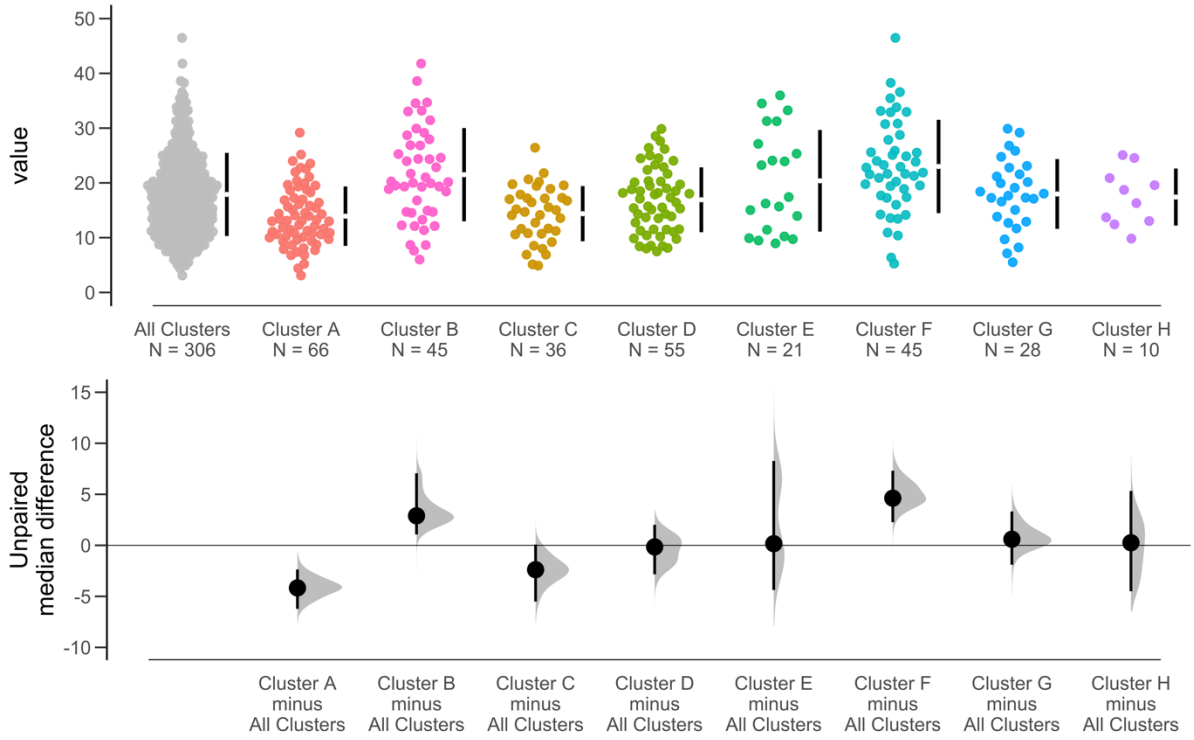

```
avg.heatmap.data = avg.heatmap.data[new.order, new.order]


# Use heatmap.2 to plot these values and organize rows and columns using
ward.D2 dendrogram
heatmap.2(as.matrix(avg.heatmap.data),
          col = col,
          Colv = new.order,
          Rowv = new.order,
          breaks = mybreaks,
          dendrogram = "none",
          trace = "none",
          margins = c(10,10))
```


Color Key
and Histogram
Count
0
2
4
6
8
0
20
60
100
Value
Cluster A
Cluster B
Cluster C
Cluster D
Cluster E
Cluster H
Cluster F
Cluster G
Cluster A
Cluster B
Cluster C
Cluster D
Cluster E
Cluster H
Cluster F
Cluster G

Estimation Statistics: Within Clusters

This is FIGURE 8 in my thesis.

```
within.estimation.stats.results = as.data.frame(matrix(ncol = 6, nrow = 8)) #
create empty data frame
colnames(within.estimation.stats.results) = c("Control", "Variable",
"Median", "Median.Diff", "Low.CI", "High.CI") # set row names

estimation.stats.data = frechet.boxplot.data
# Make a variable that stores all the values as one group, so that we can get
an overall median to compare groups against
estimation.stats.data["variable"] = "All Clusters"

# Now bind the frechet distances again, with each cluster differenitiated
estimation.stats.data = rbind(estimation.stats.data, frechet.boxplot.data)

two.group.unpaired =
  estimation.stats.data %>%
  dabest(variable, value,
         idx = c("All Clusters", "Cluster A", "Cluster B", "Cluster C",
"Cluster D", "Cluster E", "Cluster F", "Cluster G", "Cluster H"),
         paired = FALSE)

two.group.unpaired.mediandiff.within = median_diff(two.group.unpaired)

# png("Estimation Stats 72 Neuroimmune Proteins Within Frechet.png", height =
6, width = 10, units = "in", res = 1200)
plot(two.group.unpaired.mediandiff.within,
     palette = c("grey", "#F8766D", "#FF61CC", "#CD9600", "#7CAE00",
"#00BE67", "#00BFC4", "#00A9FF", "#C77CFF"),
     rawplot.ylim = c(0, 50),
     effsize.ylim = c(-10, 15),
     rawplot.type = "swarmplot")
```




```
# dev.off()

# Save values for future reference

within.estimation.stats.results["Control"] =
two.group.unpaired.mediandiff.within[["result"]][["control_group"]] # save
the control group name
within.estimation.stats.results["Variable"] =
two.group.unpaired.mediandiff.within[["result"]][["test_group"]] # save the
test group name
within.estimation.stats.results["Median"] =
two.group.unpaired.mediandiff.within[["summary"]][["median"]][2:9] # save the
median value for each cluster
within.estimation.stats.results["Median.Diff"] =
two.group.unpaired.mediandiff.within[["result"]][["difference"]] # save the
difference from the overall group
within.estimation.stats.results["Low.CI"] =
two.group.unpaired.mediandiff.within[["result"]][["bca_ci_low"]] # save the
lower bound 95 CI
within.estimation.stats.results["High.CI"] =
two.group.unpaired.mediandiff.within[["result"]][["bca_ci_high"]] # save the
upper bound 95 CI
```

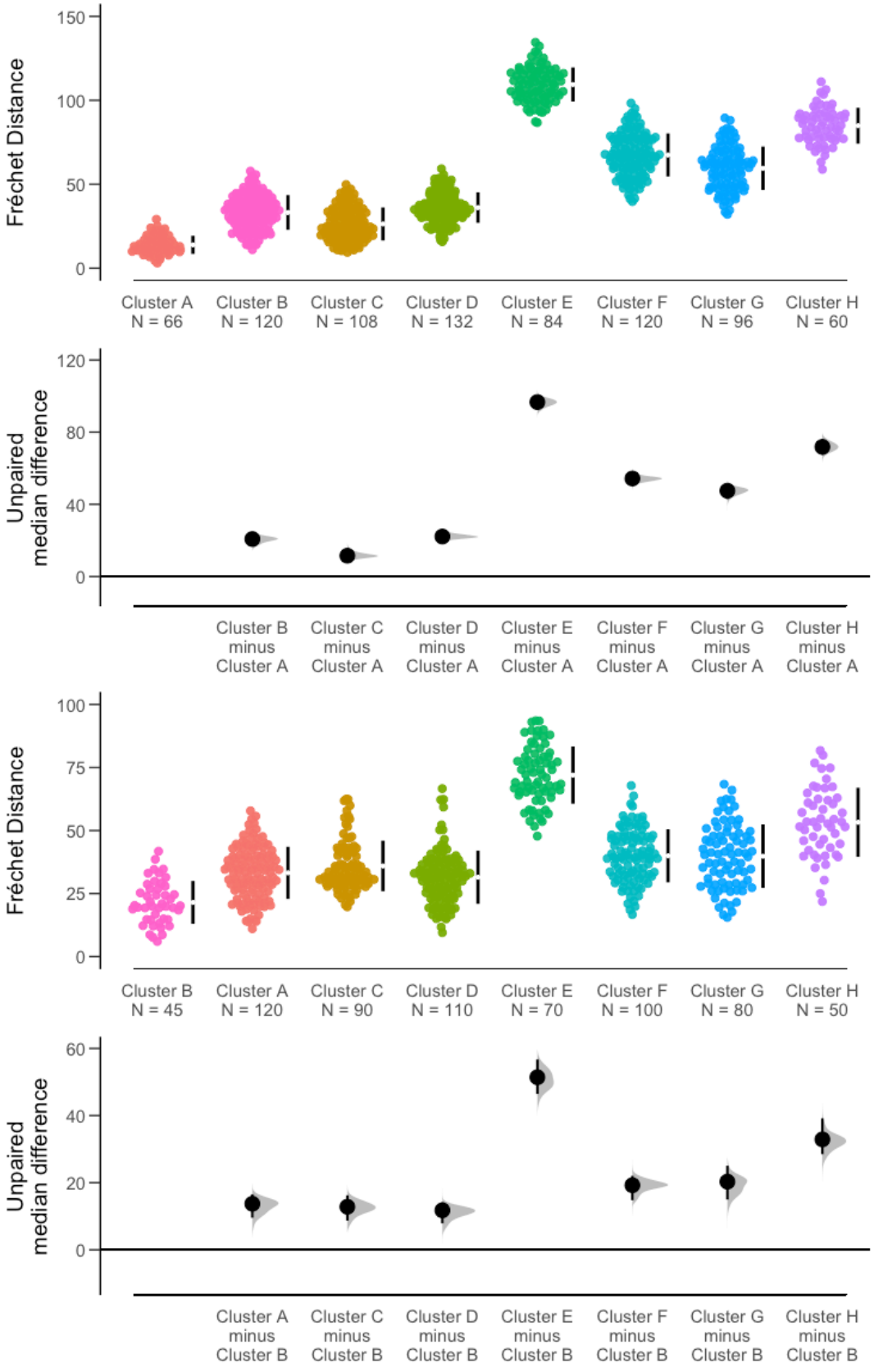
Estimation Statistics Between Clusters

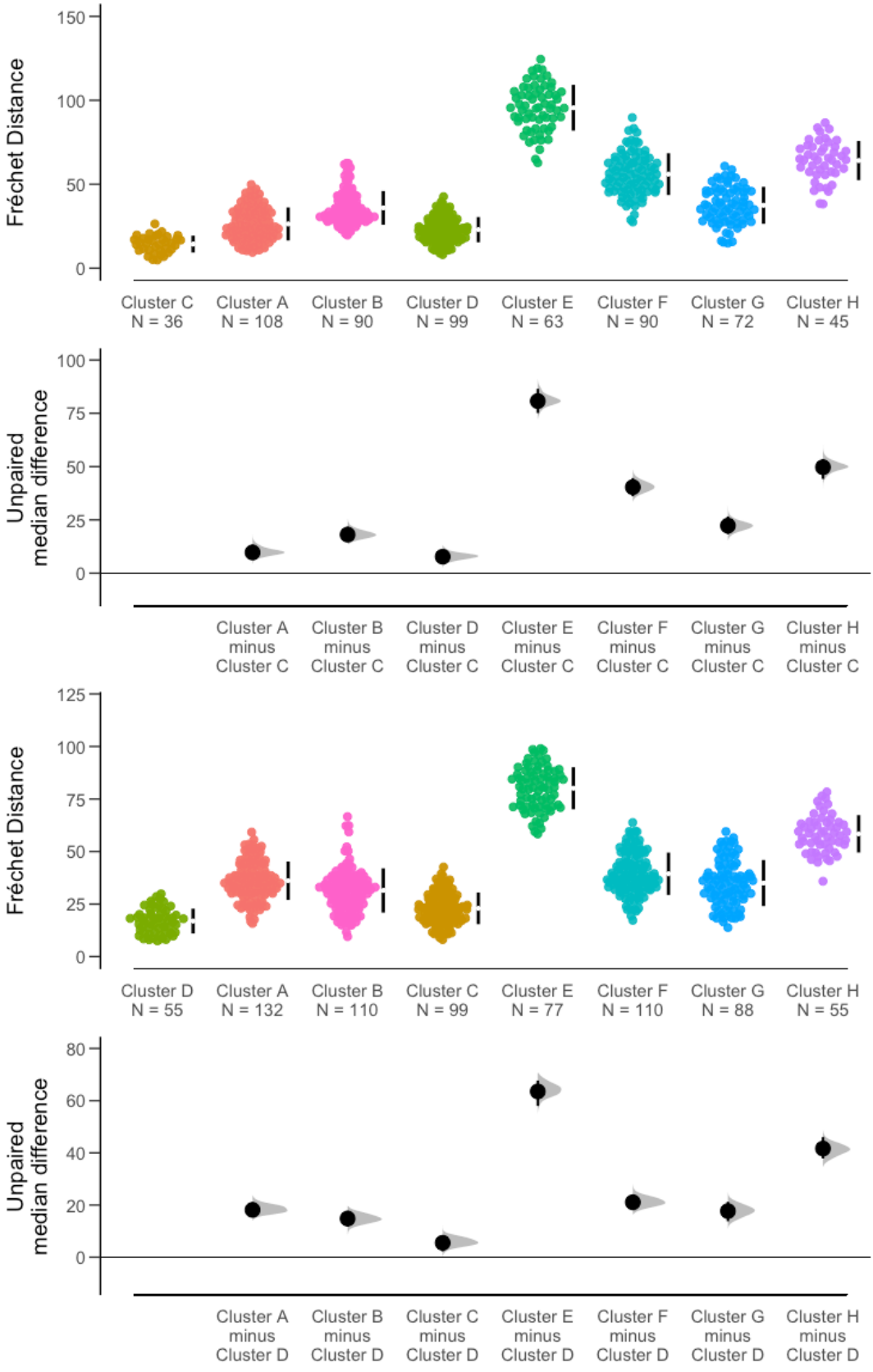
This is FIGURE 11 in my thesis.

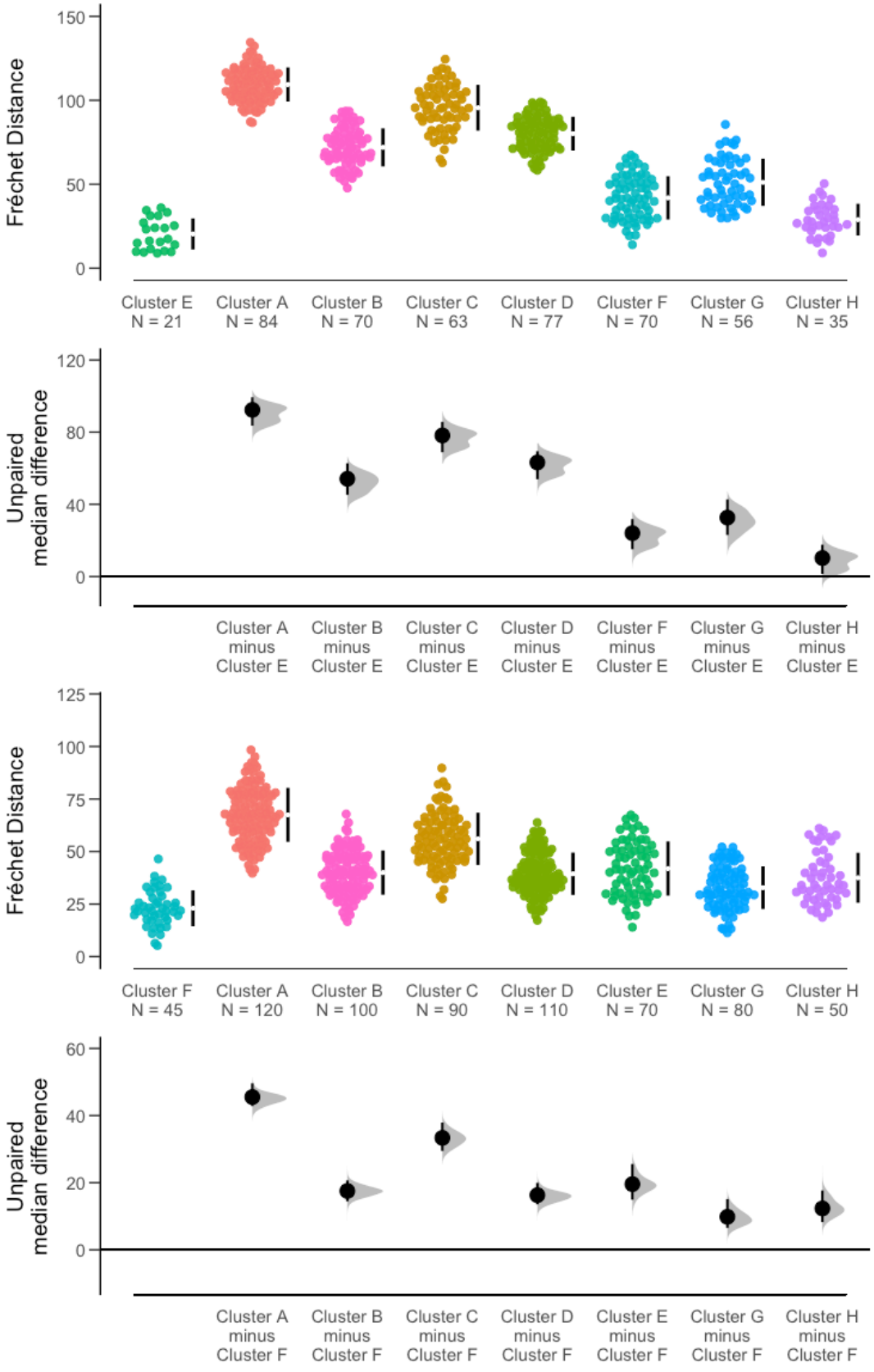
```
cols = c( "#F8766D", "#FF61CC", "#CD9600", "#7CAE00", "#00BE67", "#00BFC4",
"#00A9FF", "#C77CFF") # set colours
cluster.list = paste("Cluster", LETTERS[1:8]) # set labels
value.max = c(150, 100, 150, 120, 150, 120, 100, 120)
effect.max = c(120, 60, 100, 80, 120, 60, 60, 80)


total.between.estimation.stats.results = as.data.frame(matrix(ncol = 6)) #
create empty data frame to store estimation stats values
colnames(total.between.estimation.stats.results) = c("Control", "Variable",
"Median", "Median.Diff", "Low.CI", "High.CI") # set column names

median.within.cluster = as.data.frame(matrix(ncol = 2)) # create empty data
frame to store within cluster median
colnames(median.within.cluster) = c("Variable", "Median") # set column names

# The heatmap currently contains duplicate disatnces
frechet.scores.72.low = frechet.72.heatmap.data  # first store the values in
a different object for processing
frechet.scores.72.low[lower.tri(frechet.scores.72.low, diag=TRUE)] = NA #
subset the lower have (i.e. all unique values)

plot.list = list()


for (i in 1:8){ # loop through each cluster

  frechet.scores.between = as.data.frame(matrix(ncol = 1)) # create empty
data frame

  for (j in 1:8){ # loop through each cluster (combined with previous loop,
allows (8 x8) 64 cluster comparisons to be made)

    if(i == j){frechet.subset =
frechet.scores.72.low[ward.D2.cluster.members[[i]],
ward.D2.cluster.members[[j]]]} # get frechet scores for proteins
    if(i != j){frechet.subset =
frechet.72.heatmap.data[ward.D2.cluster.members[[i]],
ward.D2.cluster.members[[j]]]}

    frechet.subset = as.numeric(as.character(unlist(frechet.subset))) #
```

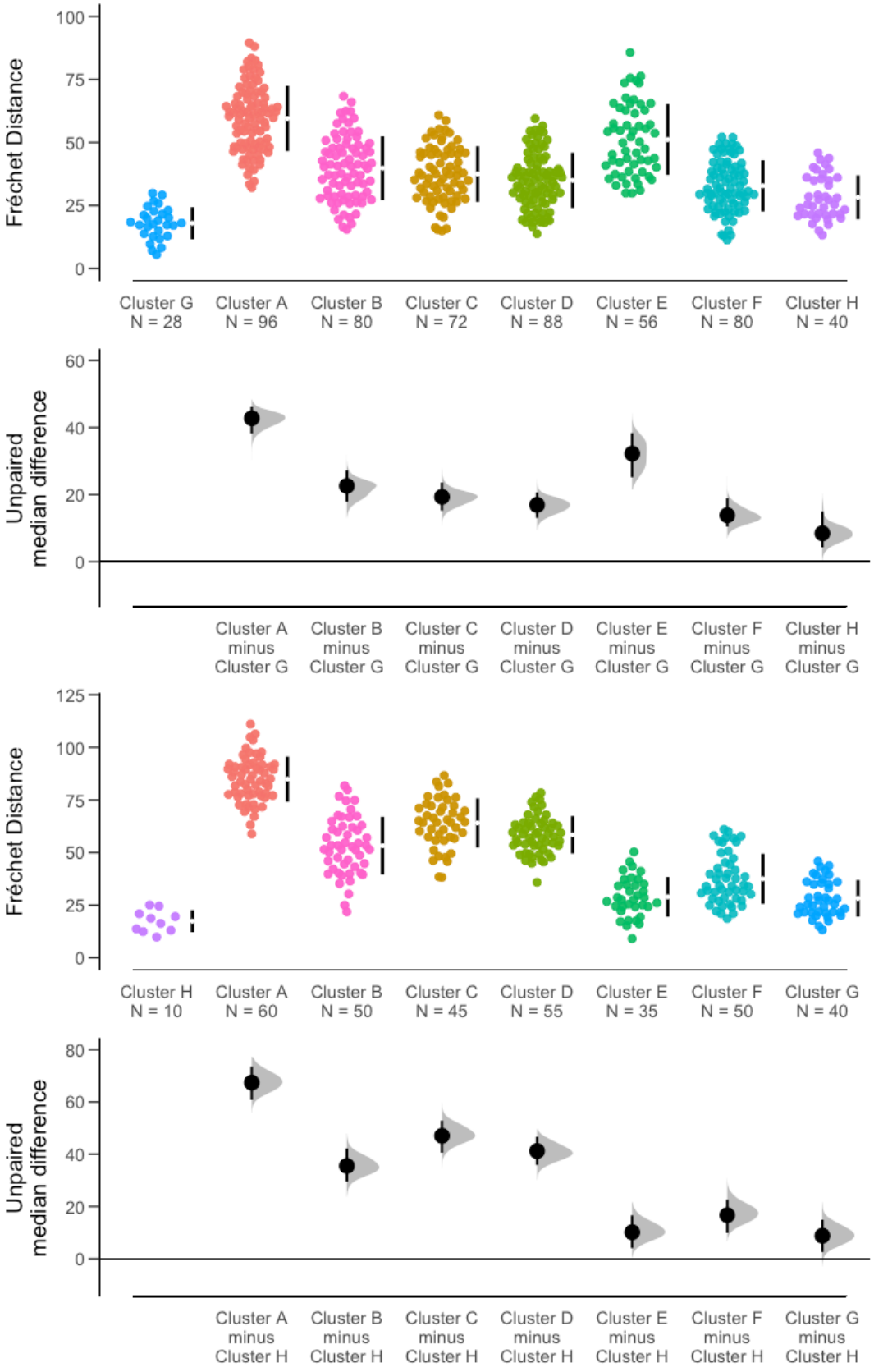
```
convert values to numeric

    frechet.scores.between = qpcR:::cbind.na(frechet.scores.between,
frechet.subset) # bind the frechet scores
    colnames(frechet.scores.between)[1+j] = paste("Cluster",
names(ward.D2.cluster.members[j])) # set column names
  }

  frechet.scores.between = frechet.scores.between[-1] # remove the empty
column
  frechet.scores.between = reshape2::melt(frechet.scores.between, na.rm =
TRUE) # covert the data frame to the long format for plotting


  # Between Cluster Estimation Stats

  between.estimation.stats.results = as.data.frame(matrix(ncol = 6, nrow =
7)) # create empty data frame to store between cluster estimation stats
  colnames(between.estimation.stats.results) = c("Control", "Variable",
"Median", "Median.Diff", "Low.CI", "High.CI") # set column names
  estimation.stats.data = frechet.scores.between # use the completed data
frame of frechet distances for one cluster as the estimation statistics data
  colnames(estimation.stats.data)[2] = "Fréchet Distance"


  control = cluster.list[[i]] # set the control variable
  comparison.groups = setdiff(cluster.list, cluster.list[[i]]) # set the
comparison groups

  # Prepare data for estimations statistics processing
  two.group.unpaired =
    estimation.stats.data %>%
    dabest(variable, `Fréchet Distance`,
           idx = c(control, comparison.groups),
           paired = FALSE)

  # Calculate effect size
  two.group.unpaired.mediandiff.between = median_diff(two.group.unpaired)

  between.cols = c(cols[[i]], setdiff(cols, cols[[i]]))

 plot.list[[i]] = plot(two.group.unpaired.mediandiff.between,
           palette = between.cols,
           rawplot.ylim = c(0, value.max[[i]]),
           effsize.ylim = c(-10, effect.max[[i]]),
           rawplot.type = "swarmplot")

  # Save the name of the control group
  between.estimation.stats.results["Control"] =
```

```
two.group.unpaired.mediandiff.between[["result"]][["control_group"]]
  # Save the name of each of the test groups
  between.estimation.stats.results["Variable"] =
two.group.unpaired.mediandiff.between[["result"]][["test_group"]]
  # Save the median frechet between each test group and the control group
  between.estimation.stats.results["Median"] =
two.group.unpaired.mediandiff.between[["summary"]][["median"]][2:8]
  # Save the median different between the control and the test group
  between.estimation.stats.results["Median.Diff"] =
two.group.unpaired.mediandiff.between[["result"]][["difference"]]
  # Save the low bound of the CI
  between.estimation.stats.results["Low.CI"] =
two.group.unpaired.mediandiff.between[["result"]][["bca_ci_low"]]
  # Save the upper bound of the CI
  between.estimation.stats.results["High.CI"] =
two.group.unpaired.mediandiff.between[["result"]][["bca_ci_high"]]
  # Bind the individual cluster stats to the data frame that will hold all
cluster values
  total.between.estimation.stats.results =
rbind(total.between.estimation.stats.results,
between.estimation.stats.results)



  # Create data frame to store median intracluster values of control groups
  median.self = as.data.frame(matrix(ncol = 2))
  colnames(median.self) = c("Variable", "Median") # set column names
  median.self["Variable"] =
two.group.unpaired.mediandiff.between[["result"]][["control_group"]][1] # get
cluster name
  median.self["Median"] =
two.group.unpaired.mediandiff.between[["summary"]][["median"]][1] # get
intracluster distance
  median.within.cluster = rbind(median.within.cluster, median.self) # bind
values to the data frame holding all cluster stats

}

# png("Estimation Statistics.png", height = 1500, width = 1200)
# do.call("grid.arrange", c(plot.list, ncol=2))
# dev.off()

for(i in 1:8){
# png(paste("Estimation Statistics Immune Neuroimmune Cluster ",
LETTERS[1:8][i], ".png", sep = ""), height = 6, width = 8, units = "in", res
= 1000)
  plot(plot.list[[i]])
# dev.off()
}
```


Fréchet Distance
150
100
50
0
Cluster A
N = 66
Cluster B
N = 120
Cluster C
N = 108
Cluster D
N = 132
Cluster E
N = 84
Cluster F
N = 120
Cluster G
N = 96
Cluster H
N = 60
Unpaired
median difference
120
80
40
0
Cluster B
minus
Cluster A
Cluster C
minus
Cluster A
Cluster D
minus
Cluster A
Cluster E
minus
Cluster A
Cluster F
minus
Cluster A
Cluster G
minus
Cluster A
Cluster H
minus
Cluster A
Fréchet Distance
100
75
50
25
0
Cluster B
N = 45
Cluster A
N = 120
Cluster C
N = 90
Cluster D
N = 110
Cluster E
N = 70
Cluster F
N = 100
Cluster G
N = 80
Cluster H
N = 50
Unpaired
median difference
60
40
20
0
Cluster A
minus
Cluster B
Cluster C
minus
Cluster B
Cluster D
minus
Cluster B
Cluster E
minus
Cluster B
Cluster F
minus
Cluster B
Cluster G
minus
Cluster B
Cluster H
minus
Cluster B


Fréchet Distance
150
100
50
0
Cluster C
N = 36
Cluster A
N = 108
Cluster B
N = 90
Cluster D
N = 99
Cluster E
N = 63
Cluster F
N = 90
Cluster G
N = 72
Cluster H
N = 45
Unpaired
median difference
100
75
50
25
0
Cluster A
minus
Cluster C
Cluster B
minus
Cluster C
Cluster D
minus
Cluster C
Cluster E
minus
Cluster C
Cluster F
minus
Cluster C
Cluster G
minus
Cluster C
Cluster H
minus
Cluster C
Fréchet Distance
125
100
75
50
25
0
Cluster D
N = 55
Cluster A
N = 132
Cluster B
N = 110
Cluster C
N = 99
Cluster E
N = 77
Cluster F
N = 110
Cluster G
N = 88
Cluster H
N = 55
Unpaired
median difference
80
60
40
20
0
Cluster A
minus
Cluster D
Cluster B
minus
Cluster D
Cluster C
minus
Cluster D
Cluster E
minus
Cluster D
Cluster F
minus
Cluster D
Cluster G
minus
Cluster D
Cluster H
minus
Cluster D


Fréchet Distance
150
100
50
0
Cluster E N = 21
Cluster A N = 84
Cluster B N = 70
Cluster C N = 63
Cluster D N = 77
Cluster F N = 70
Cluster G N = 56
Cluster H N = 35
Unpaired median difference
120
80
40
0
Cluster A minus Cluster E
Cluster B minus Cluster E
Cluster C minus Cluster E
Cluster D minus Cluster E
Cluster F minus Cluster E
Cluster G minus Cluster E
Cluster H minus Cluster E
Fréchet Distance
125
100
75
50
25
0
Cluster F N = 45
Cluster A N = 120
Cluster B N = 100
Cluster C N = 90
Cluster D N = 110
Cluster E N = 70
Cluster G N = 80
Cluster H N = 50
Unpaired median difference
60
40
20
0
Cluster A minus Cluster F
Cluster B minus Cluster F
Cluster C minus Cluster F
Cluster D minus Cluster F
Cluster E minus Cluster F
Cluster G minus Cluster F
Cluster H minus Cluster F

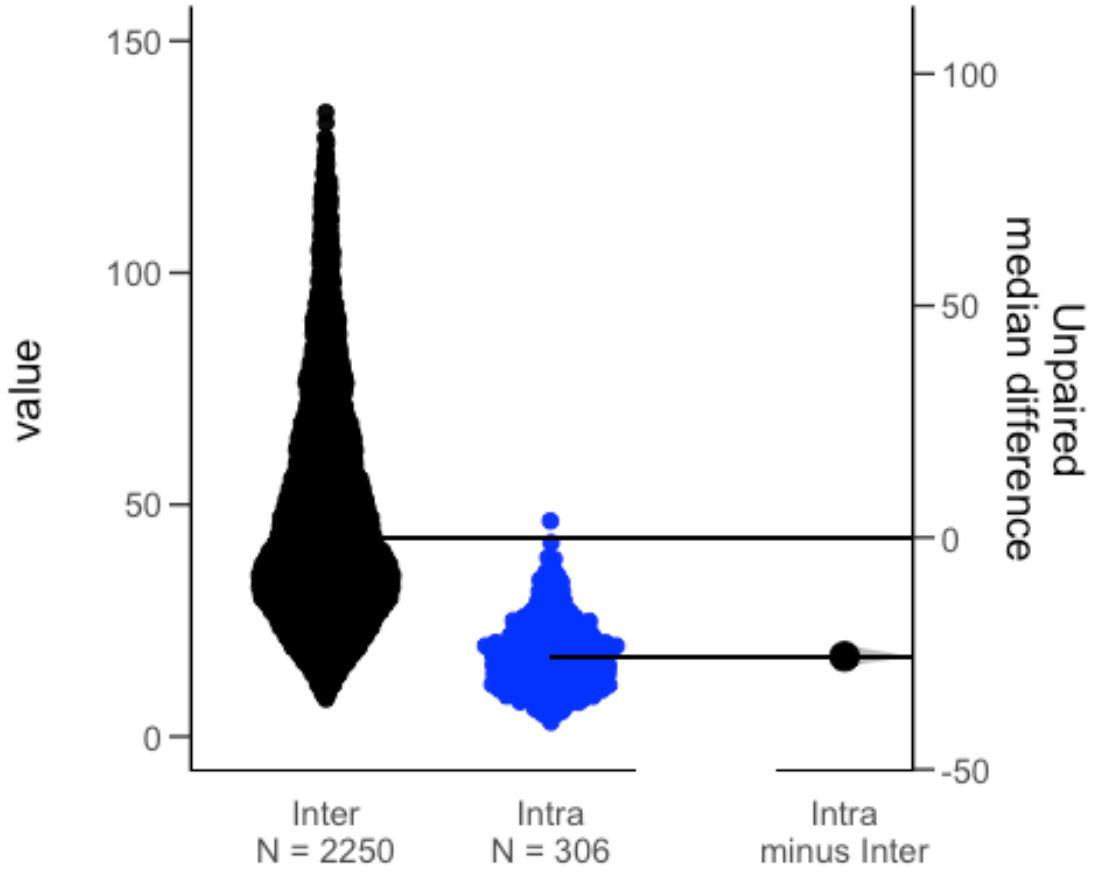

Fréchet Distance
100
75
50
25
0
Cluster G
N = 28
Cluster A
N = 96
Cluster B
N = 80
Cluster C
N = 72
Cluster D
N = 88
Cluster E
N = 56
Cluster F
N = 80
Cluster H
N = 40
Unpaired
median difference
60
40
20
0
Cluster A
minus
Cluster G
Cluster B
minus
Cluster G
Cluster C
minus
Cluster G
Cluster D
minus
Cluster G
Cluster E
minus
Cluster G
Cluster F
minus
Cluster G
Cluster H
minus
Cluster G
Fréchet Distance
125
100
75
50
25
0
Cluster H
N = 10
Cluster A
N = 60
Cluster B
N = 50
Cluster C
N = 45
Cluster D
N = 55
Cluster E
N = 35
Cluster F
N = 50
Cluster G
N = 40
Unpaired
median difference
80
60
40
20
0
Cluster A
minus
Cluster H
Cluster B
minus
Cluster H
Cluster C
minus
Cluster H
Cluster D
minus
Cluster H
Cluster E
minus
Cluster H
Cluster F
minus
Cluster H
Cluster G
minus
Cluster H

Intercluster vs Intracluster Distances

This is FIGURE 10 in my thesis.

```
# The heatmap currently contains duplicate distances
frechet.scores.72.low = frechet.72.heatmap.data  # first store the values in
a different object for processing
frechet.scores.72.low[lower.tri(frechet.scores.72.low, diag=TRUE)] = NA #
subset the lower have (i.e. all unique values)

# Create a folder that holds only the between cluster values:
all.between.cluster.values = as.data.frame(matrix(ncol = 3, nrow = 1))
colnames(all.between.cluster.values) = c("Variable 1", "Variable 2", "Value")

for (i in 1:8){ # loop through each cluster
# Extract the frechet distances that do not correspond to intracluster
comparisons
not.intracluster = frechet.scores.72.low[ward.D2.cluster.members[[i]],
setdiff(colnames(frechet.scores.72.low), ward.D2.cluster.members[[i]])]
# Convert the data frame to the long format
not.intracluster = reshape2::melt(setDT(not.intracluster, keep.rownames =
TRUE), "rn", na.rm = TRUE)
# Set column names
colnames(not.intracluster) = c("Variable 1", "Variable 2", "Value")
# Bind the indivudal cluster's  values to a data frame that will contain all
the clusters values
all.between.cluster.values = rbind(all.between.cluster.values,
not.intracluster)
}

all.between.cluster.values = all.between.cluster.values[-1,] # remove empty
row
all.between.cluster.values = cbind("Inter", all.between.cluster.values$Value)
# add a row that indicates that these values are intercluster distances

# Create a data frame that contains all intracluster distances, labelled as
such (i.e. "Intra")
only.intracluster = cbind("Intra", frechet.boxplot.data$value)
# Bind the inter- and intra-cluster values together with labels
intra.inter.est.stats = as.data.frame(rbind(all.between.cluster.values,
only.intracluster))
colnames(intra.inter.est.stats) = c("variable", "value") # set column names
# Make distances numeric
intra.inter.est.stats$value = as.numeric(unlist(intra.inter.est.stats$value))

# Prepare data for estimation statistics
group.unpaired =
intra.inter.est.stats %>%
  dabest(variable, value,
```

```
          idx = c("Inter", "Intra"),
          paired = FALSE)

# Perform statistical analysis
unpaired.mediandiff = median_diff(group.unpaired)

# png("Estimation Stats Intracluster Vs Intercluster.png", height = 6, width
= 10, units = "in", res = 1200)
print(plot(unpaired.mediandiff,
           palette = c("black", "blue"),
           rawplot.ylim = c(0, 150),
           rawplot.type = "swarmplot"))
```




```
# dev.off()
```

Unpacking the Molecular Functions, Cellular Components and Biological Processes of the Clusters

Version gProfiler: "e100_eg47_p14_7733820" Version GO Update: 2020-06-01 Version gProfiler2 (i.e. the function): 0.2.0

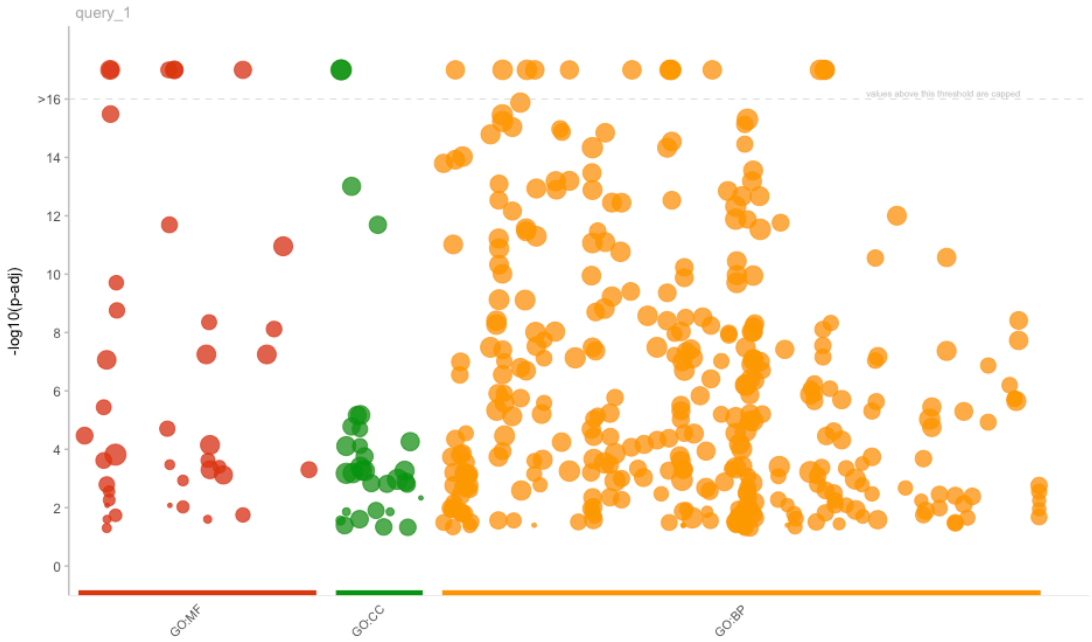
I will use gProfiler, an enrichment interface for the Gene Ontology (GO) website to determine the biological functions in the overall data set and in each cluster. GO contains annotations for tens of thousands of genes, these are can be by organism.

Notes for using gProfiler: when using a custom reference list "custom_bg" is needed, if using the entire human genome this can be removed. Additionally, domain_scope becomes "custom" when custom_bg is used. Otherwise use "annotated". This means the reference list is all genes in the human genome with annotated functions. 'evcodes" is set to TRUE to extract which genes map to which term. Finally, the correction method is the default "gSCS", which can deal specifically with terms in a hierarchy. The other correction methods are not suited for GO hierarchical data.

First, perform enrichment for all 72 unique genes to determine the enriched MF, CC and BP terms for the entire set of immune proteins.

```
# Get a list of gene symbols for all 72 proteins
immune.proteins.gene.symbol = ID.matches[which(ID.matches$Raybiotech.Names
%in% selected.immune.proteins), "Gene.Symbol"]
LAP = which(immune.proteins.gene.symbol == "LAP(TGFB1)") # brackets prevent
LAP(TGFB1) from being detected,
immune.proteins.gene.symbol[LAP] = "TGFB1" # so manually add it
allgenesgotable = gost(query = immune.proteins.gene.symbol, # set the query
terms
                       organism = "hsapiens", # organism of choice is homo
sapiens
                       ordered_query = FALSE,  # the order of proteins is not
important
                       multi_query = FALSE, # there are not multiple list,
consider all terms together
                       significant = TRUE, # only print of significant values
                       exclude_iea = TRUE, # exclude from results the
interactions based on weak evidence
                       measure_underrepresentation = FALSE, # results should
represent enriched terms not underestimated terms
                       evcodes = TRUE, # display a column that indicates
which proteins from query list map onto the enriched term
                       user_threshold = 0.05, # significance threshold, top
0.05
```

```
                        correction_method = "gSCS", # the custom correction
method for p-values
                        domain_scope = "annotated", # the background list of
terms to compare my query list is all genes with annotations
                        custom_bg = NULL, # not supplying a custom list
                        numeric_ns = "", # set namespace for fully numeric
gene ID, our is not, so leave at default
                        sources = c("GO:BP", "GO:MF", "GO:CC"), # which
versions of
                        as_short_link = FALSE) # get the results as a URL set
to FALSE
allgenesgoresults = allgenesgotable$result # export the enrichment results
gostplot(allgenesgotable, capped = TRUE, interactive = FALSE) # forms a
Manhattan plot
```

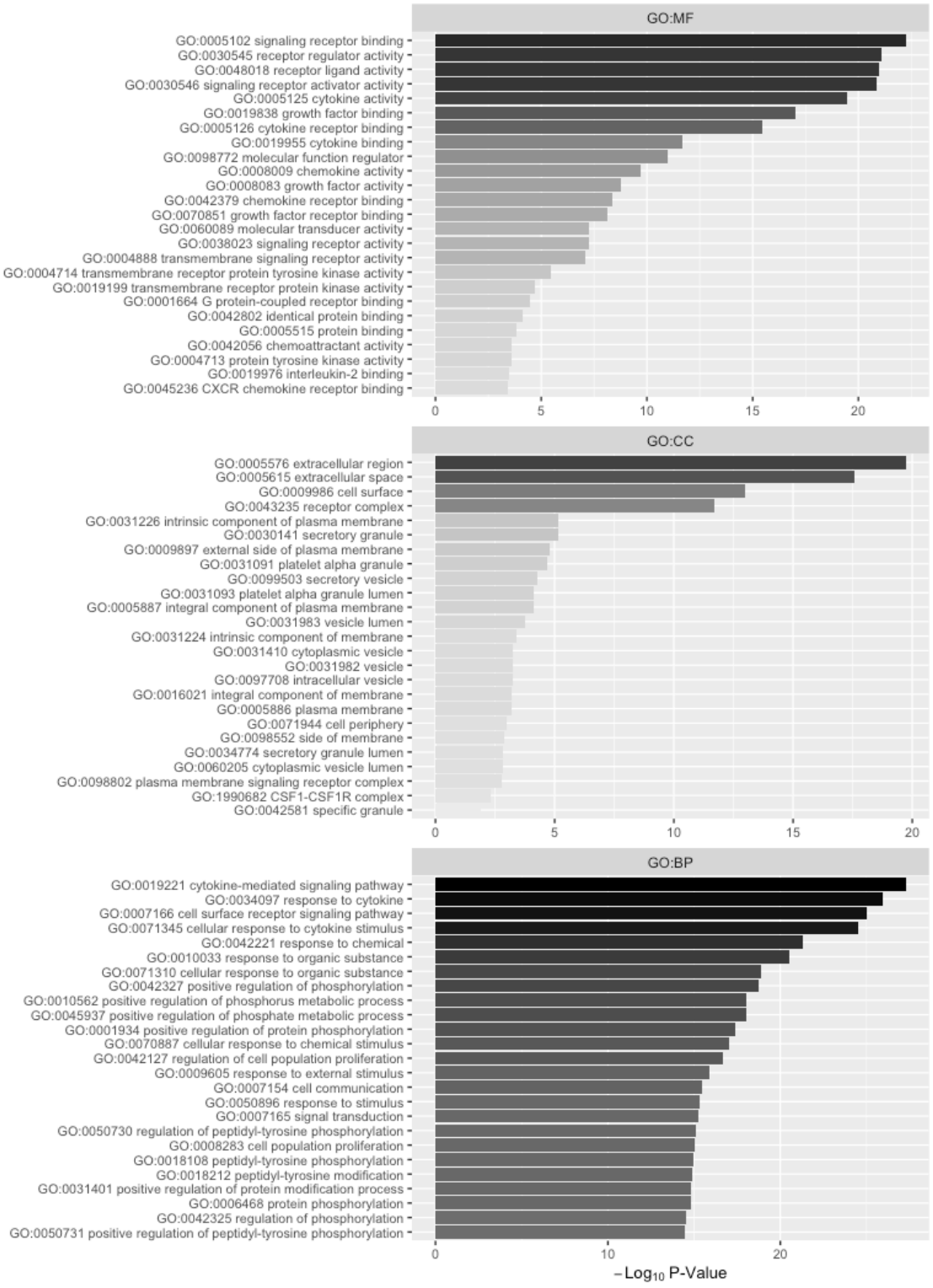


Note: PDGFAB was not detected by the GO database. This is because PDGFAB is a heterodimer, the two constituent proteins PDGFA and PDGFB are encoded by two separate genes, and are included in our data set of 72 proteins. Since PDGFAB is not recognized this analysis will consist of 71 proteins not 72.

Pick out the top 25 terms for each GO category and display results as a bar graph. This is FIGURE 12 in my thesis.

```
# Create lists of the various naming conventions for the GO categories
# To be used for labelling and data sub setting
# Must be in the same order
go.categories = c("GO:MF", "GO:CC", "GO:BP")
go.category.abbreviation = c("MF", "CC", "BP")
```

```
go.category.name = c("Molecular Functions", "Cellular Components",
"Biological Processes")


# Create a data frame for storing values needed for bar plot
forplotting = as.data.frame(ncol(16))

for (index.go.category in 1:length(go.categories)){
  # Subset the enrichment performed on all 72 proteins by the gene category
  category.subset = subset(allgenesgoresults, allgenesgoresults$source ==
go.categories[index.go.category])
  # Order the terms from greatest to lowest p-value
  category.subset = category.subset[order(category.subset$p_value, decreasing
= FALSE),]
  # Extract the top 25 terms
  forplotting = rbind(forplotting, category.subset[1:25,])
}

  # Combine the GO ID and the full GO term to create labels for the plot
  forplotting$labels = paste(forplotting$term_id, forplotting$term_name, sep
= " ")
  # Turn these labels to factors for graphing in this order
  forplotting$labels = factor(forplotting$labels, levels =
rev(forplotting$labels))
  # Order in the factors in the source column as well for facets
  forplotting$source = factor(forplotting$source, levels =
unique(as.character(unlist(forplotting$source))))


  # Use ggplot to create the bar plot
  print(ggplot(forplotting, aes(fill = -log10(p_value), x = -log10(p_value),
y = labels, group = source)) +
    facet_wrap(~source, ncol = 1, scales = "free") +
    geom_bar(stat = "identity") +
    scale_fill_gradient2(low = "white", high = "black", guide = F) +
    xlab(bquote(-Log[10] ~"P-Value")) +
    ylab(NULL)+
   # ggtitle(paste("Enriched GO", go.category.name[index.go.category])) +
    theme(plot.title = element_text(size = 10, hjust = 0.01),
          axis.text.y = element_text(size= 8),
          axis.text.x = element_text(size= 8),
          axis.title.x = element_text(size=10)))
```




```
 # ggsave("Top 25 GO Enrichments by Category.png", device = "png", height = 11, width = 8)
```

Clean the table to determine which proteins map onto the top 25 terms from each GO category.

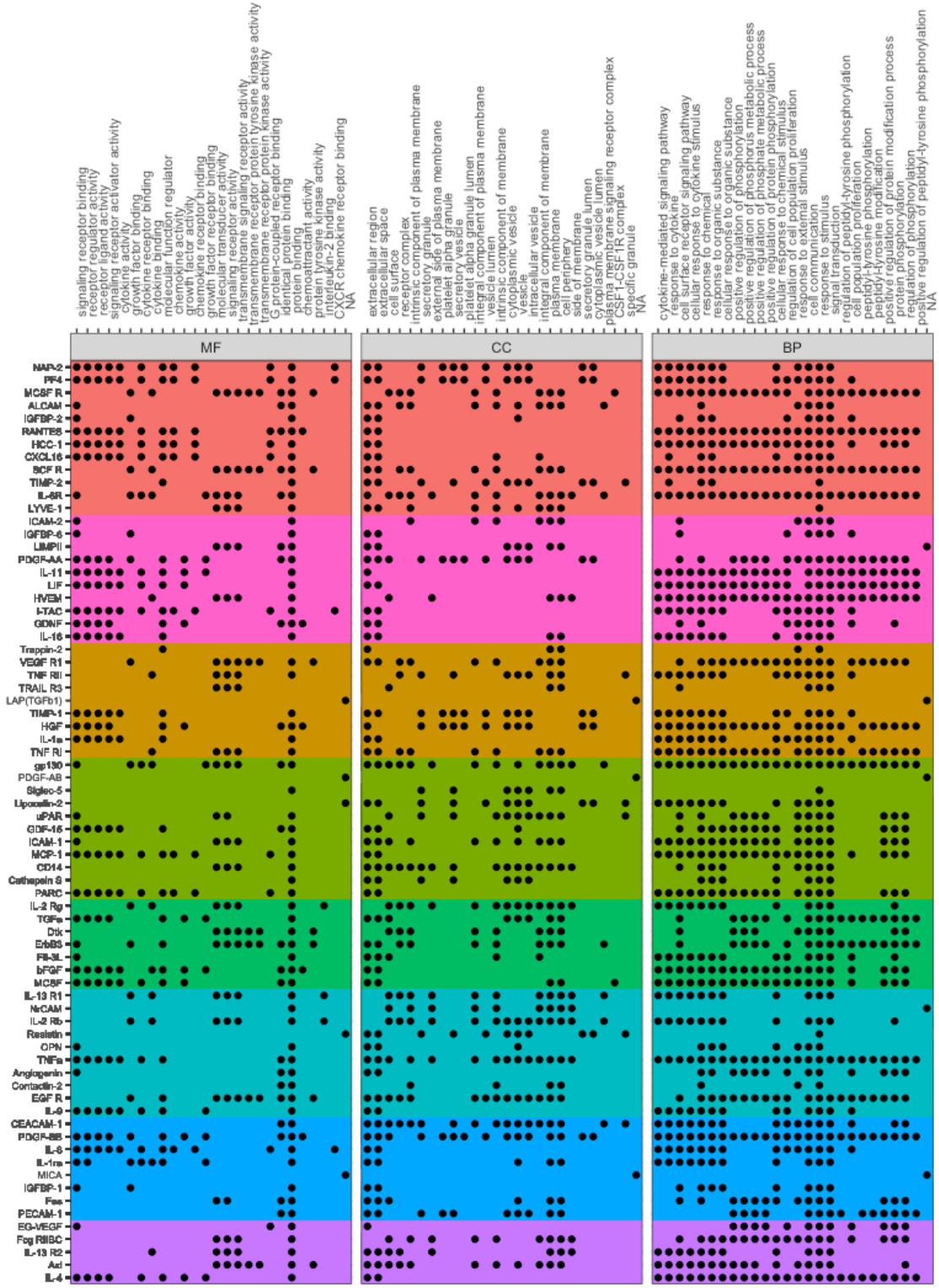
```
for (index.go.category in 1:length(go.categories)){
  # Subset the enrichment performed on all 72 proteins by the gene category
  category.subset = subset(allgenesgoresults, allgenesgoresults$source ==
go.categories[index.go.category])
  # Order the terms from greatest to lowest p-value
  category.subset = category.subset[order(category.subset$p_value, decreasing
= FALSE),]
  # Extract the top 25 terms
  protein.annotations = category.subset[1:25,]
  # Use tidyr to reformat the data to provide annotations for each protein
  protein.annotations = separate_rows(protein.annotations, intersection,
convert = TRUE)
  # Extract the necessary columns: GO ID, GO term name, and the protein gene
symbol
  protein.annotations = protein.annotations[c(16,9,11)]
  # Set column names
  colnames(protein.annotations)[1] = "query"
  # Assign a name and save these data frame to the global environment
  name = paste("protein.annotations",
go.category.abbreviation[index.go.category], sep = ".")
  assign(name, protein.annotations)
}
# Create a list of all categorical protein annotations
protein.annotations.list = list(protein.annotations.MF,
protein.annotations.CC, protein.annotations.BP)
```

Visualize how the individual proteins map onto the top 25 GO terms. This is FIGURE 13 in my thesis.

```
  # Create a vector to store the hierarchical order of proteins as gene
symbols
  order.immune.dend.gene.symbol = order.immune.dend
  for(i in 1:length(order.immune.dend.gene.symbol)){ # loops through the 72
neuroimmune proteins
    # get index of row in data frame that matches the vector
    gene.index = which(ID.matches$Raybiotech.Names %in%
order.immune.dend.gene.symbol[i])
    # use the gene symbol column for that row to fill the vector with the
gene symbol
    order.immune.dend.gene.symbol[i] = ID.matches[gene.index, "Gene.Symbol"]
  }


for (index.go.category in 1:length(protein.annotations.list)){
  # Determine which proteins map onto which term
```

```
  bubbleplotdata = protein.annotations.list[[index.go.category]]
  # The proteins that do not map onto any of the top terms need to be added
to this dataframe as NA values
  NA.dataframe = as.data.frame(setdiff(immune.proteins.gene.symbol,
as.character(unique(bubbleplotdata$query))))
  NA.dataframe = cbind(NA.dataframe, as.data.frame(matrix(ncol = 2, nrow =
ncol(NA.dataframe))))
  colnames(NA.dataframe) = c("query", "term_id", "term_name")
  NA.dataframe[2:3] = "NA"
  bubbleplotdata = bind_rows(bubbleplotdata, NA.dataframe)
  # Placeholder variable to indicate proteins that map onto terms
  bubbleplotdata$Present = 1
  # Obtain the order of terms based on their p-values
  # Subset the enrichment performed on all 72 proteins by the gene category
  category.subset = subset(allgenesgoresults, allgenesgoresults$source ==
go.categories[index.go.category])
  # Order the terms from greatest to lowest p-value
  category.subset = category.subset[order(category.subset$p_value, decreasing
= FALSE),]
  # Convert "TGFB1" to "LAP(TGFB1)"
  bubbleplotdata[which(as.character(unlist(bubbleplotdata$query)) ==
"TGFB1"), "query"] = "LAP(TGFB1)"
  # Convert the query and term_name columns from characters to factors to
preserve order columns and rows when graphing
  bubbleplotdata = bubbleplotdata %>%
  mutate(query = factor(query, levels = rev(order.immune.dend.gene.symbol)),
         Y = as.numeric(query))
  bubbleplotdata$term_name = factor(bubbleplotdata$term_name, levels =
category.subset[1:25, "term_name"])
  # Add protein names
  bubbleplotdata$protein = as.character(NA)
  for(row in 1:nrow(bubbleplotdata)){
  index = which(ID.matches$Gene.Symbol ==
as.character(unlist(bubbleplotdata[row, "query"])))
  bubbleplotdata[row, "protein"] = ID.matches[index, "Raybiotech.Names"]
  }
  # Add the cluster identity, loop through rows and determine the cluster
that the protein belongs in
  bubbleplotdata$cluster = as.character(NA)
  for(protein.index in 1:nrow(bubbleplotdata)){
    for(cluster.index in 1:length(ward.D2.cluster.members)){
    if(as.character(unlist(bubbleplotdata[protein.index,"protein"])) %in%
ward.D2.cluster.members[[cluster.index]]) {
      bubbleplotdata[protein.index,"cluster"] =
names(ward.D2.cluster.members)[[cluster.index]]}}}
  # Factor the cluster designations to indicate which proteins to plot first
  bubbleplotdata$cluster = factor(bubbleplotdata$cluster, levels =
unique(as.character(unlist(bubbleplotdata$cluster))))
  # Assign colours for the clusters
  colours = c( "#F8766D", "#FF61CC", "#CD9600", "#7CAE00", "#00BE67",
```

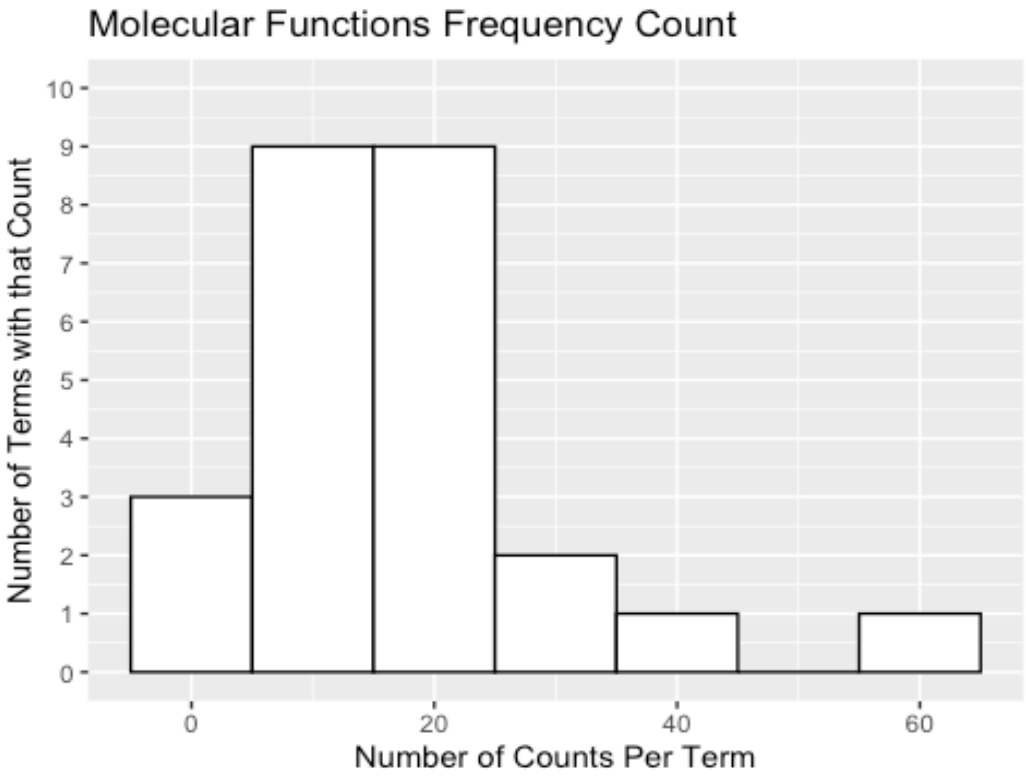
```
"#00BFC4", "#00A9FF", "#C77CFF")
  # Save bubbleplotdata to the global environment while specifying the GO
category
  name = paste("bubbleplotdata", go.category.abbreviation[index.go.category],
sep = "")
  assign(name, bubbleplotdata)
}

# Add a column to the individual data frames to indicate which GO category
they are representing
bubbleplotdataMF$source = "MF"
bubbleplotdataCC$source = "CC"
bubbleplotdataBP$source = "BP"

# Bind all three GO category data frames together to plot on one page
allbubbleplots = rbind(bubbleplotdataMF, bubbleplotdataCC, bubbleplotdataBP)
# Factor the different levels so that MF is plotted first and BP is plotted
last in the page
allbubbleplots$source = factor(allbubbleplots$source, levels =
unique(allbubbleplots$source))

# Use ggplot to plot the GO Counts
print(ggplot(allbubbleplots, aes(group = source)) +
  facet_wrap(~source, ncol = 3, scales = "free_x") +
  geom_rect(aes(ymin = Y - 0.5, ymax = Y + 0.5, xmin = -Inf, xmax = Inf, fill
= cluster)) +
  scale_fill_manual(breaks = LETTERS[1:8], values = colours) +
  geom_point(aes(x = term_name, y = Y)) +
  scale_x_discrete(position = "top") +
  scale_y_continuous(breaks = allbubbleplots$Y, labels =
allbubbleplots$protein, expand = c(0,0)) +
  labs(x = NULL, y = NULL, title = NULL) +
  theme_bw() +
  theme(axis.text.x = element_text(size = 8, angle = 90),
        axis.text.y = element_text(size = 6),
        axis.text.x.top = element_text(hjust = 0.05, vjust = 1),
        legend.position = "none"))
```

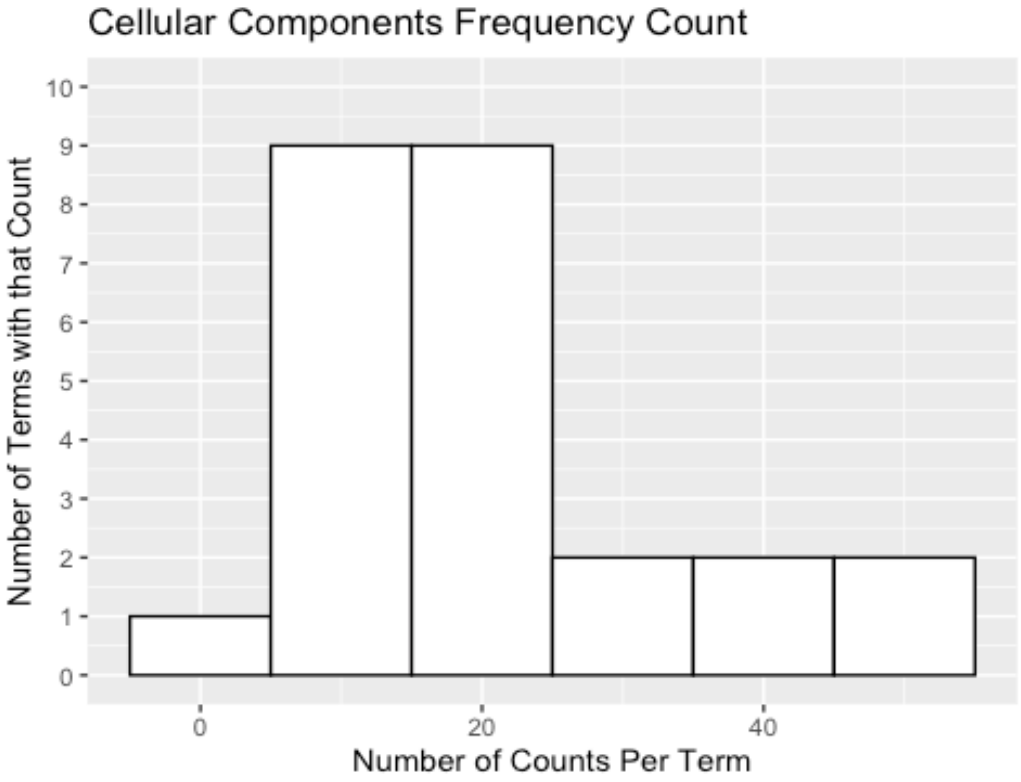

MF
CC
BP
signaling receptor binding
receptor regulator activity
receptor ligand activity
signaling receptor activator activity
cytokine activity
growth factor binding
cytokine receptor binding
cytokine binding
molecular function regulator
chemokine activity
growth factor activity
chemokine receptor binding
growth factor receptor binding
molecular transducer activity
signaling receptor activity
transmembrane signaling receptor activity
transmembrane receptor protein tyrosine kinase activity
transmembrane receptor protein kinase activity
G protein-coupled receptor binding
identical protein binding
protein binding
cytokine activity
protein tyrosine kinase activity
interleukin-2 binding
CXCR chemokine receptor binding
NA
extracellular region
extracellular space
cell surface
receptor complex
intrinsic component of plasma membrane
secretory granule
external side of plasma membrane
platelet alpha granule
secretory vesicle
platelet alpha granule lumen
integral component of plasma membrane
vesicle lumen
intrinsic component of membrane
cytoplasmic vesicle
vesicle
intracellular vesicle
integral component of membrane
cell periphery
plasma membrane
side of membrane
secretory granule lumen
cytoplasmic vesicle lumen
plasma membrane signaling receptor complex
CSF1-CSF1R complex
specific granule
NA
cytokine-mediated signaling pathway
response to cytokine
cellular response to cytokine stimulus
cellular response to chemical stimulus
response to chemical
response to organic substance
cellular response to organic substance
positive regulation of phosphorylation
positive regulation of phosphorus metabolic process
positive regulation of phosphate metabolic process
positive regulation of protein phosphorylation
cellular response to chemical stimulus
regulation of cell population proliferation
response to external stimulus
cell communication
response to stimulus
signal transduction
regulation of peptidyl-tyrosine phosphorylation
cell population proliferation
peptidyl-tyrosine phosphorylation
peptidyl-tyrosine modification
positive regulation of protein modification process
protein phosphorylation
regulation of phosphorylation
positive regulation of peptidyl-tyrosine phosphorylation
NA
MAP-2
PF4
MCSF R
ALCAM
IGFBP-2
RANTES
HCC-1
CXCL16
SCF R
TIMP-2
IL-6R
LYVE-1
ICAM-2
IGFBP-6
LIMPII
PDGF-AA
IL-11
LIF
HVEM
I-TAC
GDNF
IL-16
Trappin-2
VEGF R1
TNF RII
TRAIL R3
LAP(TGFb1)
TIMP-1
HGF
IL-1a
TNF RI
gp130
PDGF-AB
Siglec-5
Lipocalin-2
uPAR
GDF-15
ICAM-1
MCP-1
CD14
Cathepsin S
PARC
IL-2 Rg
TGFa
Dtk
ErbB3
Flt-3L
bFGF
MCSF
IL-13 R1
NrCAM
IL-2 Rb
Resistin
OPN
TNFa
Angiogenin
Contactin-2
EGF R
IL-9
CEACAM-1
PDGF-BB
IL-6
IL-1ra
MICA
IGFBP-1
Fas
PECAM-1
EG-VEGF
Fcg RIIBC
IL-13 R2
Axl
IL-4

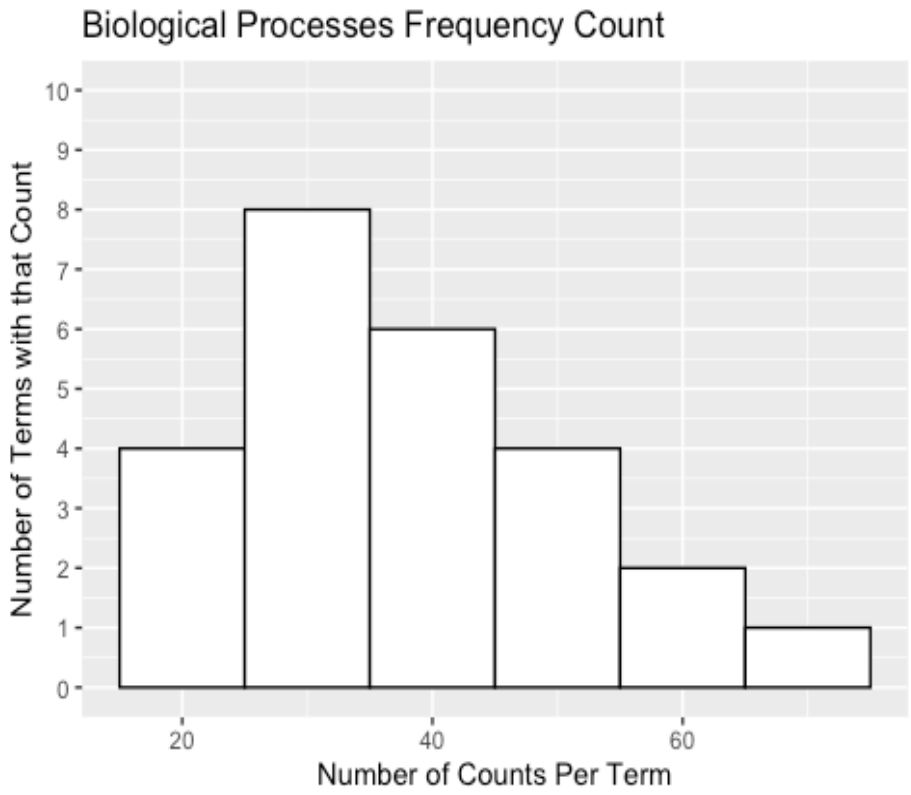
```
# ggsave("Top 25 GO Enrichments by Category per protein.png", device = "png",
height = 11, width = 8)

bubbleplotdata.list = list(bubbleplotdataMF, bubbleplotdataCC,
bubbleplotdataBP) # create a list that keeps the GO categories separate, if
needed
```

Calculate the frequency of each term for each GO category.

```
freq.list = list(MF.frequency = as.data.frame(table(bubbleplotdataMF[-
which(bubbleplotdataMF$term_id == "NA"),]$term_id)),
                 CC.frequency = as.data.frame(table(bubbleplotdataCC[-
which(bubbleplotdataCC$term_id == "NA"),]$term_id)),
                 BP.frequency = as.data.frame(table(bubbleplotdataBP[-
which(bubbleplotdataBP$term_id == "NA"),]$term_id)))

# Add the term name to the data frame, match using the term ids
for (index.go.category in 1:length(freq.list)){
  for(i in 1:25){
    for(j in 1:nrow(allgenesgoresults)){
      if(freq.list[[index.go.category]][["Var1"]][i] == allgenesgoresults[j,
"term_id"]){
        freq.list[[index.go.category]][["term_name"]][i] =
allgenesgoresults[j, "term_name"]}
  }
  }
}

# Visualize the number of terms that display a particular count
# Ex. how many terms have 20 proteins mapped to them etc.

for (index.go.category in 1:length(freq.list)){
frequency = freq.list[[index.go.category]]
print(ggplot(frequency, aes(Freq))+
  geom_histogram(stat = "bin", binwidth = 10, color="black", fill="white") +
  labs(x = "Number of Counts Per Term", y = "Number of Terms with that
Count") +
  ggtitle(paste(go.category.name[index.go.category], "Frequency Count", sep =
" ")) +
  scale_y_continuous(limits = c(0,10), breaks = seq(0,10,1)))
}
```

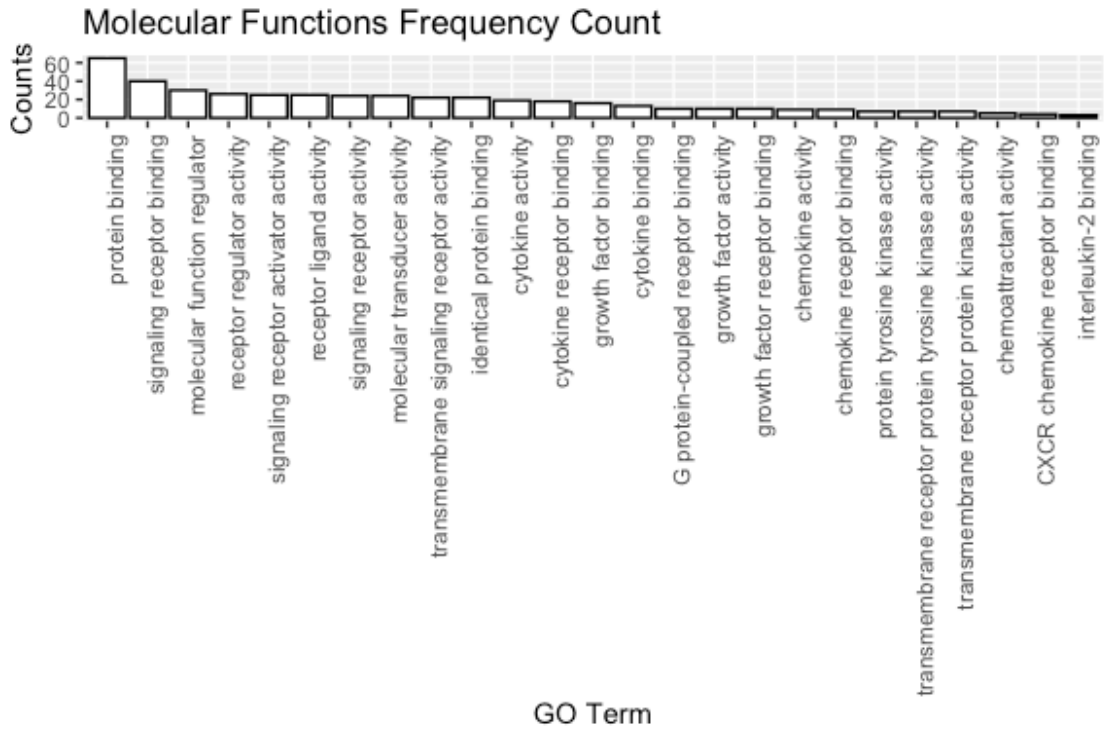

Molecular Functions Frequency Count
Number of Terms with that Count
10
9
8
7
6
5
4
3
2
1
0
0
20
40
60
Number of Counts Per Term

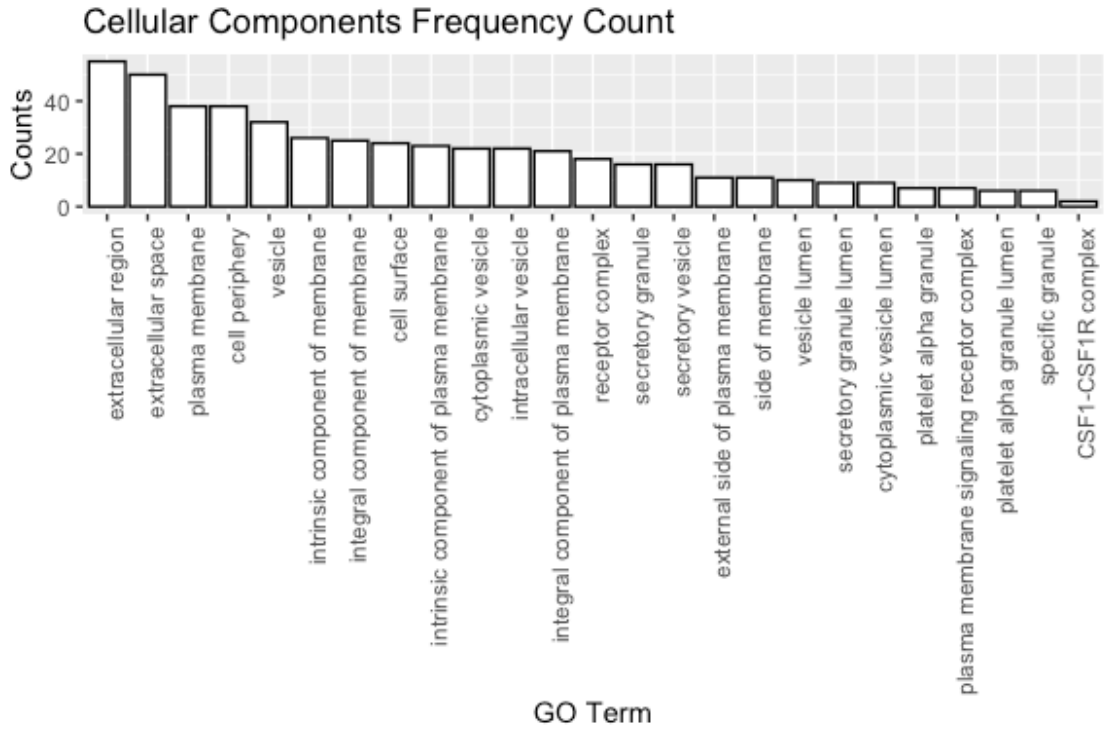

Cellular Components Frequency Count
Number of Terms with that Count
10
9
8
7
6
5
4
3
2
1
0
0
20
40
Number of Counts Per Term

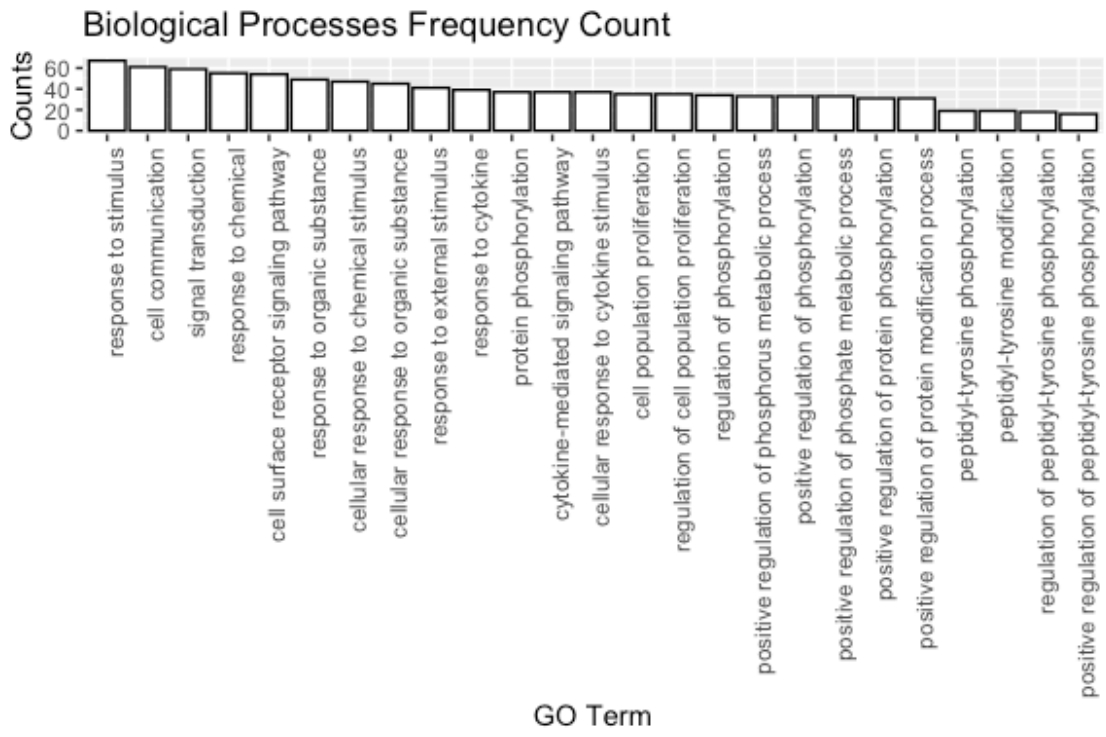


Counts per individual term.

```
for (index.go.category in 1:length(freq.list)){
frequency = freq.list[[index.go.category]]
frequency = frequency[order(frequency$Freq, decreasing = TRUE),]
frequency$term_name = factor(frequency$term_name, levels =
unique(frequency$term_name))
print(ggplot(frequency, aes(term_name, Freq))+
  geom_col(color="black", fill="white")+
  labs(x = "GO Term", y = "Counts")+
  ggtitle(paste(go.category.name[index.go.category], "Frequency Count", sep =
" ")) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)))
}
```


Molecular Functions Frequency Count
Counts
60
40
20
0
protein binding
signaling receptor binding
molecular function regulator
receptor regulator activity
signaling receptor activator activity
receptor ligand activity
signaling receptor activity
molecular transducer activity
transmembrane signaling receptor activity
identical protein binding
cytokine activity
cytokine receptor binding
growth factor binding
cytokine binding
G protein-coupled receptor binding
growth factor activity
growth factor receptor binding
chemokine activity
chemokine receptor binding
protein tyrosine kinase activity
transmembrane receptor protein tyrosine kinase activity
transmembrane receptor protein kinase activity
chemoattractant activity
CXCR chemokine receptor binding
interleukin-2 binding
GO Term


Cellular Components Frequency Count
Counts
40
20
0
extracellular region
extracellular space
plasma membrane
cell periphery
vesicle
intrinsic component of membrane
integral component of membrane
cell surface
intrinsic component of plasma membrane
cytoplasmic vesicle
intracellular vesicle
integral component of plasma membrane
receptor complex
secretory granule
secretory vesicle
external side of plasma membrane
side of membrane
vesicle lumen
secretory granule lumen
cytoplasmic vesicle lumen
platelet alpha granule
plasma membrane signaling receptor complex
platelet alpha granule lumen
specific granule
CSF1-CSF1R complex
GO Term



Perform an over representation analysis, where the proportion of all 71 proteins that map onto a term are compared to the proportion of proteins *in a cluster* that map onto that term. This analysis will reveal which of the terms are over- and under-represented in a cluster and help unpack functional significance.

```
go.categories = c("GO:MF", "GO:CC", "GO:BP")
go.category.abbreviation = c("MF", "CC", "BP")
go.category.name = c("Molecular Functions", "Cellular Components",
"Biological Processes")

# Create a data frame that will count the number of times a term occurs in
each cluster
for(index.go.category in 1:length(bubbleplotdata.list)){
  # Pull out the enrichment info for a single GO category
  plotdata = bubbleplotdata.list[[index.go.category]]
  # Remove the rows that contain NA (i.e. the proteins not mapped to any
term)
  plotdata = plotdata[-which(plotdata$term_id == "NA"),]
  # Get the terms that are unique
  terms = unique(na.omit(as.character(unlist(plotdata$term_id))))
  # Create a data frame to store the counts per term, per cluster
  plot.counts = as.data.frame(matrix(ncol = length(terms), nrow = 9)) %>%
  set_rownames(c("Cluster A","Cluster B","Cluster C","Cluster D", "Cluster
E","Cluster F","Cluster G","Cluster H", "Total"))
  # Use for loop to fill the data frame with the counts
```

```
  for(term.index in 1:length(terms)){
    term.count = subset(plotdata, plotdata$term_id == terms[term.index]) #
subset enrichment info by term
    plot.counts[9,term.index] = nrow(term.count) # save the total numbe rof
proteins that mao to that term across clusters
    colnames(plot.counts)[term.index] = as.character(unlist(term.count[3])) #
add column name
    for(cluster.index in 1:8){
      cluster.term.count = subset(term.count, term.count$cluster ==
LETTERS[1:8][cluster.index]) # subset the term count by cluster
      plot.counts[cluster.index,term.index] = nrow(cluster.term.count) # save
the number of times that term occurs in each cluster
    }
  }

# Create a data frame that will store for each term, for each cluster, the
number of hits, the total number of proteins in cluster
# and the expected rate based of total hits out of 71
total.counts = (data.frame(t(plot.counts)))$Total
names(total.counts) = colnames(plot.counts)
test.counts = plot.counts  %>%
    rownames_to_column("cluster") %>%
    filter(cluster != "Total") %>%
    melt(id.vars = "cluster", variable.name = "term", value.name = "count")
%>%
    mutate(observations = case_when(
      cluster == "Cluster A" ~ 12,
      cluster == "Cluster B" ~ 10,
      cluster == "Cluster C" ~ 9,
      cluster == "Cluster D" ~ 10, # 11 in total but this cluster contains
PDGFAB which is not detected by GO, so only 10
      cluster == "Cluster E" ~ 7,
      cluster == "Cluster F" ~ 10,
      cluster == "Cluster G" ~ 8,
      cluster == "Cluster H" ~ 5
    ))  %>%
    mutate(overall_rate = total.counts[match(term, names(total.counts))] /
71)

 # Execute Binomial Test: Conduct the binomial test and add the p-values and
confidence intervals to the data frame

# Function for p-value
  b.test.p = function(x, n, p) {
    binom.test(x, n, p, conf.level = 0.7)$p.value
  }

# Function for lower bound 95% CI
  b.test.CI.low = function(x, n, p) {
```

```
    binom.test(x, n, p, conf.level = 0.7)$conf.int[1]
  }

# Function for upper bound 95% CI
  b.test.CI.high = function(x, n, p) {
    binom.test(x, n, p, conf.level = 0.7)$conf.int[2]
  }

# Apply the above tests to each row of the test.counts data frame
  test.counts$p_value = mapply(b.test.p, test.counts$count,
test.counts$observations, test.counts$overall_rate)
  test.counts$CI_low = mapply(b.test.CI.low, test.counts$count,
test.counts$observations, test.counts$overall_rate)
  test.counts$CI_high = mapply(b.test.CI.high, test.counts$count,
test.counts$observations, test.counts$overall_rate)
# Assign a name and save each GO category's results to the global environment
  name = paste(go.category.abbreviation[index.go.category], "test.counts",
sep = ".")
  assign(name, test.counts)
}

# Create a data frame that stores the values.
test.counts.list = list(MF.test.counts, CC.test.counts, BP.test.counts)
```

Heatmap: we create an "adjusted p-value" in place of a test statistic (which the exact binomial test does not provide)

This is calculated using the equation:

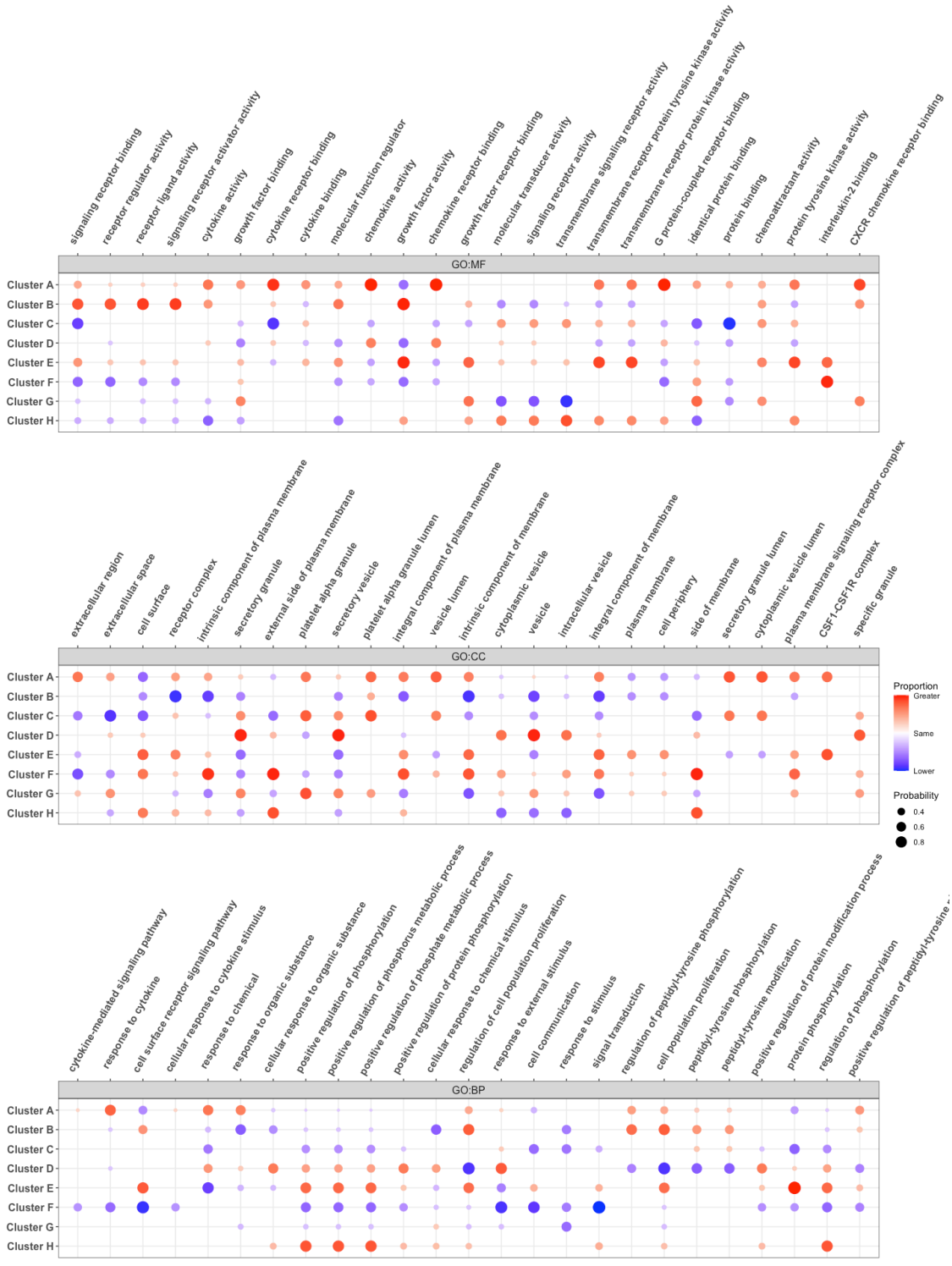
$$\text{adjusted p} - \text{value} = (-1)^k \times (1 - \text{p} - \text{value})$$

$$\text{Where } k = \begin{cases} 0, & \text{if proportion in cluster} - \text{proportion overall} \geq 0 \\ 1, & \text{if proportion in cluster} - \text{proportion overall} < 0 \end{cases}$$

This is FIGURE 14 in my thesis.

```
# Create a data frame to store the adjusted p-values a.k. probability values
GO.pvalues.all = as.data.frame(ncol(4))

for(index.go.category in 1:length(test.counts.list)){ # loop through each of
the test.counts for each GO category
  test.counts = test.counts.list[[index.go.category]] # extract a values
associated to a specific GO category
  # subtract the binomial test p-value from 1 to get a probability value,
store these as adjusted p-values
  GO.pvalues = test.counts %>%
    mutate(k = ifelse(count/observations - overall_rate > 0, 0, 1),
           adj_pval = (-1)^k * (1 - p_value)) %>%
    dplyr::select(cluster, term, adj_pval) %>% # select specific rows to save
    dcast(term ~ cluster) %>% # convert data frame to wide format
    column_to_rownames("term") # set the values in the column "term" as row
```

```
names for the data frame

GO.pvalues = as.matrix(GO.pvalues)
col = colorRampPalette(c("blue", "white", "red"))(20)

GO.pvalues = as.data.frame(GO.pvalues)
GO.pvalues$term = rownames(GO.pvalues)
GO.pvalues = melt(GO.pvalues, id = "term")
GO.pvalues[GO.pvalues == 0] = NA # set zeros as NAs so that they are blank in
the plot
GO.pvalues$source = go.categories[index.go.category]
GO.pvalues.all = rbind(GO.pvalues.all, GO.pvalues)
}


GO.pvalues.all$term = factor(GO.pvalues.all$term, levels =
unique(as.character(GO.pvalues.all$term)))
GO.pvalues.all$source = factor(GO.pvalues.all$source, levels = c("GO:MF",
"GO:CC", "GO:BP"))
GO.pvalues.all$variable = factor(GO.pvalues.all$variable, levels =
rev(c("Cluster A","Cluster B","Cluster C","Cluster D", "Cluster E","Cluster
F","Cluster G","Cluster H")))

clust.bubble = ggplot(GO.pvalues.all, aes(x = term, y = variable, color =
value, size = abs(value), group = source)) +
facet_wrap(~source, scales = "free", ncol = 1) +
geom_point() +
scale_color_gradient2(low = "blue", mid = "white", high = "red",
                      breaks = c(min(na.omit(GO.pvalues$value)), 0,
max(na.omit(GO.pvalues$value))),
                      labels = c("Lower", "Same" ,"Greater")) +
scale_x_discrete(position = "top") +
theme_bw() +
theme(axis.text.x = element_text(size = 12, angle = 60, face = "bold"),
      axis.text.y = element_text(size = 12, face = "bold"),
      strip.text.x = element_text(size = 12),
      axis.text.x.top = element_text(vjust = 0.05, hjust = 0.05))+
labs(x = NULL, y = NULL, colour = "Proportion", size = "Probability")
clust.bubble
```




```
# ggsave("GO Enrichment Bubble Plots.png", width = 15, height = 20, units = "in")
```

Relationship to synaptic proteins and classic plasticity markers

Our lab has previously explored the lifespan trajectories of 23 well-known synaptic plasticity markers. This includes glutamatergic receptor proteins, GABAergic receptor proteins as well as myelin associated proteins.

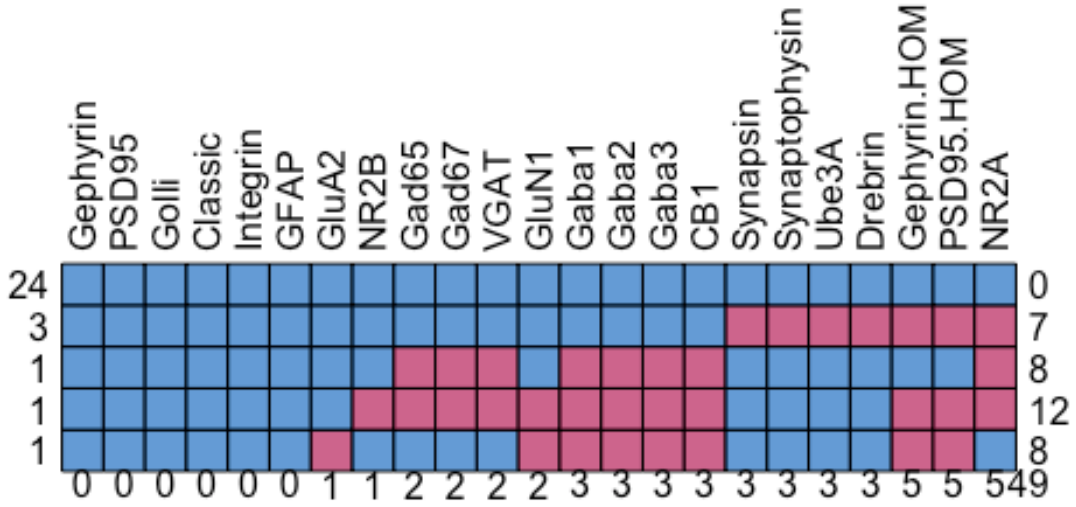
To determine which neuroimmune proteins show similar lifespan patterns to these plasticity markers, I will perform the clustering analysis on the 95 protein trajectories (23 synaptic trajectories, 72 immune trajectories).

Synaptic and Plasticity Marker Data Preparation

There are missing values in the data set, where a particular protein may not have been measured in a sample, or there were unreliable measurements. Note: this data was collected using Western Blots not a Multiplex ELISA like the immune data.

Below we can observe the nature of the missing data.

```
# Show missing data in pink hue
md.pattern(plasticity.data, rotate.names = TRUE)
```



```
##    Gephyrin PSD95 Golli Classic Integrin GFAP GluA2 NR2B Gad65 Gad67 VGAT
GluN1
## 24        1     1     1       1        1    1     1    1     1     1    1
1
## 3         1     1     1       1        1    1     1    1     1     1    1
1
## 1         1     1     1       1        1    1     1    1     0     0    0
1
## 1         1     1     1       1        1    1     1    0     0     0    0
0
## 1         1     1     1       1        1    1     0    1     1     1    1
0
##          0     0     0       0        0    0     1    1     2     2    2
2
##    Gaba1 Gaba2 Gaba3 CB1 Synapsin Synaptophysin Ube3A Drebrin Gephyrin.HOM
## 24     1     1     1   1        1             1     1       1            1
## 3      1     1     1   1        0             0     0       0            0
## 1      0     0     0   0        1             1     1       1            1
## 1      0     0     0   0        1             1     1       1            0
## 1      0     0     0   0        1             1     1       1            0
##        3     3     3   3        3             3     3       3            5
##    PSD95.HOM NR2A
## 24         1    1  0
## 3          0    0  7
## 1          1    0  8
## 1          0    0 12
## 1          0    1  8
##            5    5 49

# Add identifiers to the plasticity.data
plasticity.data = cbind(identifiers, plasticity.data)
```

The output tells us that 24 samples are complete, 3 samples that are missing the same 7 proteins, and 3 samples are missing measurements for 8-12 proteins.

In order to create a complete data set we imputed the missing values. To be specific, we used the predictive mean matching method for imputing these values.

Predictive mean matching is an imputation method that works as follows:

- Select a small set of 'candidate donors' (in our case we will use 5) from all complete cases that have predicted values that are the closest to the predicted value for the missing entry.
- From the candidates, we randomly choose one donor, and this donor's observed value fills in for the missing entry.

Impute the values. We will use the mice() function from the mice package in R. The package version used originally to create the imputed data was mice 3.6.0.

```
# Replace spaces in column names with periods
colnames(plasticity.data) = gsub(" ", ".", colnames(plasticity.data))

# Impute the data using mice()
mice.imputed.data = mice(data = plasticity.data[7:29], m = 5, method = "pmm",
maxit = 50, seed = 500)
# m - Refers to 5 imputed data sets
# maxit - Refers to no. of iterations taken to impute missing values
# method - Refers to method used in imputation. we used predictive mean
matching
# seed - An integer that is used as argument by the set.seed() for offsetting
the random number generator

# Store each of the 5 data frames created and set the row names as the case
number
mice.imputation.1 = complete(mice.imputed.data, 1) %>%
  set_rownames(rownames(plasticity.data))

mice.imputation.2 = complete(mice.imputed.data, 2) %>%
  set_rownames(rownames(plasticity.data))

mice.imputation.3 = complete(mice.imputed.data, 3) %>%
  set_rownames(rownames(plasticity.data))

mice.imputation.4 = complete(mice.imputed.data, 4) %>%
  set_rownames(rownames(plasticity.data))

mice.imputation.5 = complete(mice.imputed.data, 5) %>%
  set_rownames(rownames(plasticity.data))

# Create one data frame with averaged values across all 5 data frames
imputed.plasticity.data.new =
  (mice.imputation.1 + mice.imputation.2 +  mice.imputation.3 +
mice.imputation.4 + mice.imputation.5) / 5 %>%
  # Round this to 4 digits
  round(digits = 4)
imputed.plasticity.data.new = cbind(identifiers, imputed.plasticity.data.new)
```

Save the imputed dataset and the average dataset. These can be found in our supplementary materials. Note that from this point on, I am working with the original imputed data frame, not the example provided above.

```
# write.csv(imputed.plasticity.data, "Average Imputed Dataframe.csv" )
# write.csv(mice.imputation.1, "Imputed Mice Dataframe 1.csv")
# write.csv(mice.imputation.2, "Imputed Mice Dataframe 2.csv")
# write.csv(mice.imputation.3, "Imputed Mice Dataframe 3.csv")
# write.csv(mice.imputation.4, "Imputed Mice Dataframe 4.csv")
# write.csv(mice.imputation.5, "Imputed Mice Dataframe 5.csv")
# capture.output(mice.imputed.data, "Imputation Lists.txt")
```

```r
# Bind the identifiers to the original imputed file, and remove the column
containing case IDs
imputed.plasticity.data = cbind(identifiers, imputed.plasticity.data[-1])

# Rename proteins with more specific aliases
colnames(imputed.plasticity.data)[10] = "GABAA-a1"
colnames(imputed.plasticity.data)[11] = "GABAA-a2"
colnames(imputed.plasticity.data)[12] = "GABAA-a3"
colnames(imputed.plasticity.data)[15] = "PSD-95"
colnames(imputed.plasticity.data)[17] = "PSD-95.HOM"
colnames(imputed.plasticity.data)[22] = "GluN2A"
colnames(imputed.plasticity.data)[23] = "GluN2B"
colnames(imputed.plasticity.data)[26] = "Integrin-B3"

# Two MBPs due to presence of Golli and Classic MBP - manually specify
colnames(imputed.plasticity.data)[24] = "Golli-MBP"
colnames(imputed.plasticity.data)[25] = "Classic-MBP"
```

The following tables provides a quick look at the first 10 rows for Gephyrin.HOM in the original and imputed data frames. You can see that originally, case 271 is missing a measurement. This is no longer the case after imputing the missing values. The protein names have been changed to standardized formats where applicable.

Missing Values:

	PSD95	Gephyrin.HOM
271	0.2626	NA
1283	0.4206	1.9504
1055	0.7190	1.3041
1296	0.4399	0.3094
1102	0.6551	1.8156
135	0.8145	0.7790
569	0.3436	2.1769
166	0.8684	0.9385
774	0.7564	0.4157
1063	0.7545	1.2366

Imputed Values:

PSD-95	Gephyrin.HOM
0.2626	1.3227
0.4206	1.9504
0.7190	1.3041

0.4399	0.3094
0.6551	1.8156
0.8145	0.7790
0.3436	2.1769
0.8684	0.9385
0.7564	0.4157
0.7545	1.2366

To determine which immune proteins show similar protein trajectories to these synaptic proteins, we scaled the synaptic data (protein z-score), and then plotted LOESS curves, similar to our immune proteins. The collection of 23 plasticity trajectories are seen below.

It is not necessary to log transform the plasticity markers' expression values as they have already been normalized to a control sample. The control sample is a mixture of all samples that were run on the Western Blot.

As seen from the histograms below, the frequency distribution does not change greatly when values are imputed

Frequency Histogram: Raw Data with Missing Values

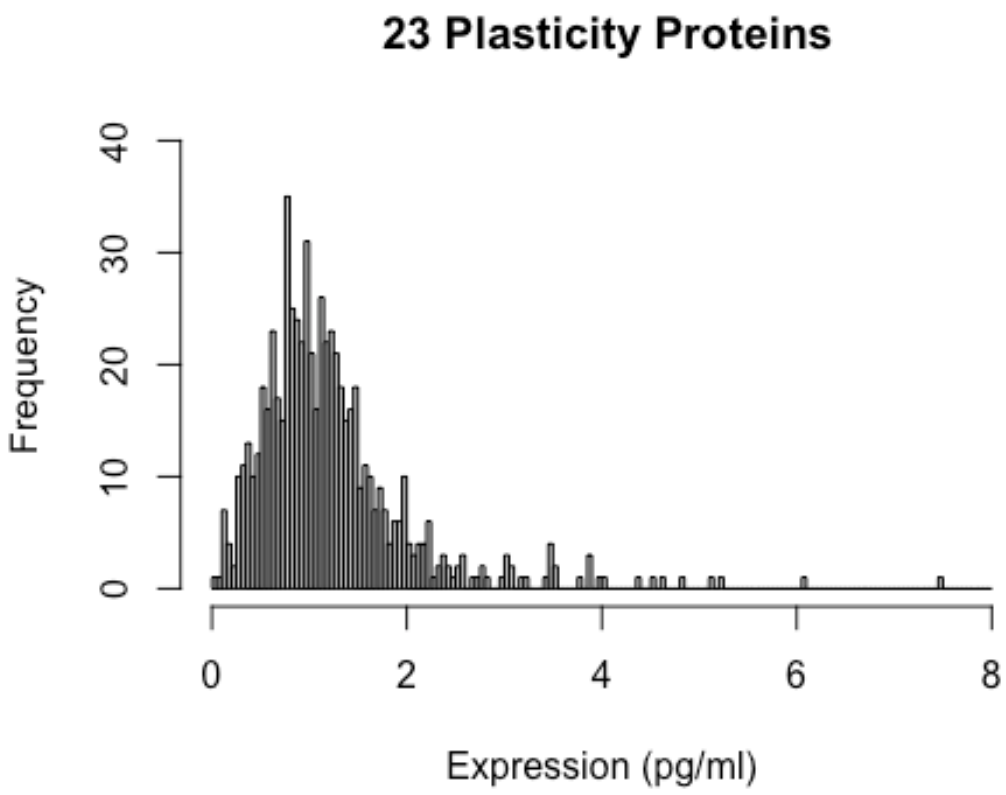
```
plasticity.exp = as.matrix(plasticity.data[7:29]) # histogram function
requires matrix input
max(as.numeric(unlist(plasticity.exp)), na.rm = TRUE) # use to set max x-axis
limits of frequency plot
```

```
## [1] 7.4946
```

```
min(as.numeric(unlist(plasticity.exp)), na.rm = TRUE) # use to set min x-axis
limits of frequency plot
```

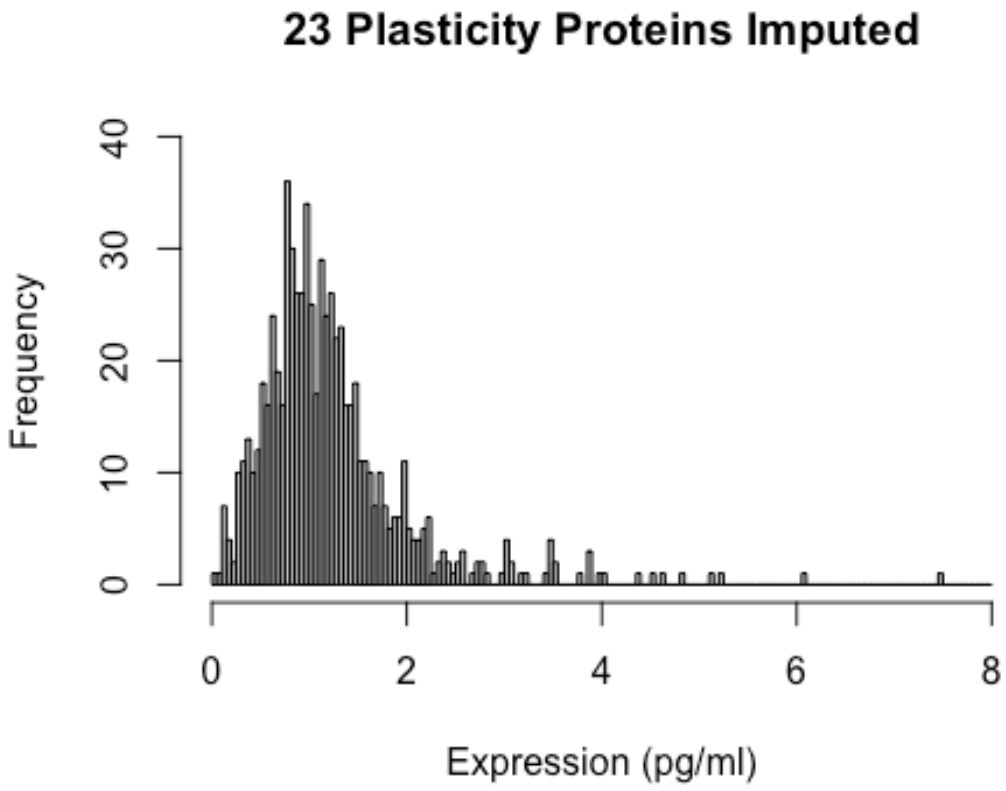
```
## [1] 0.0419
```

```
plasticity.exp.freq.hist = hist(plasticity.exp, freq = TRUE, main = " 23
Plasticity Proteins",
           ylab= "Frequency", xlab= "Expression (pg/ml)", breaks = seq(0,8,
by = 0.05),
           xlim = c(0, 8), ylim = c(0, 40), labels = F, cex= 0.3, col="grey")
```



Frequency Histogram: Raw Data with Imputed

```
imputed.plasticity.exp = as.matrix(imputed.plasticity.data[7:29]) # histogram function requires matrix input
max(as.numeric(unlist(imputed.plasticity.exp)), na.rm = TRUE) # use to set max x-axis limits of frequency plot

## [1] 7.4946

min(as.numeric(unlist(imputed.plasticity.exp)), na.rm = TRUE) # use to set min x-axis limits of frequency plot

## [1] 0.0419

imputed.plasticity.exp.freq.hist = hist(imputed.plasticity.exp, freq = TRUE, main = "23 Plasticity Proteins Imputed",
           ylab= "Frequency", xlab= "Expression (pg/ml)", breaks = seq(0,8, by = 0.05),
           xlim = c(0, 8), ylim = c(0, 40), labels = F, cex= 0.3, col="grey")
```



We observe that the distribution of values does not change drastically when imputing missing values. This is a desired outcome.

Scale the plasticity data

```
# Create a data frame to store the z-score values, with the same dimensions
and column names as the log2 expression data
zscore.23.plasticity.proteins = as.data.frame(matrix(ncol =
ncol(imputed.plasticity.exp), nrow = nrow(imputed.plasticity.exp)))
colnames(zscore.23.plasticity.proteins) = colnames(imputed.plasticity.exp)

# Calculate z-score on protein by protein basis, remove NA values before
performing calculations
for (protein.index in 1:ncol(zscore.23.plasticity.proteins)) {
  zscore.23.plasticity.proteins[protein.index] <-
zscore(imputed.plasticity.exp[,protein.index], na.rm = TRUE)
}
```

Classify Proteins as Imputed or Not

Create a data frame that classifies the synaptic proteins as either “Raw” or “Imputed”.

```
# Replicate Neural_Protein_Exp_23
plasticity.value.classification = as.data.frame(plasticity.exp) %>%
  # Classify each protein expression as "Raw" or "Imputed" - > if empty then
it will be imputed
  mutate_all(funs(ifelse(is.na(.),"Imputed","Raw")))
# Set colnames
colnames(plasticity.value.classification) = colnames(imputed.plasticity.exp)
```

Plot Plasticity Protein LOESS Values

See the supplementary file, Section C, for these figures/ plots.

```
# Bind the identifiers and adjusted protein expression values together
data.for.loess = cbind(identifiers, zscore.23.plasticity.proteins)  %>%
  # Restructure the data frame by grouping the columns together using the
common identifiers
  melt(id = colnames(identifiers))  %>%
  # Rename columns as Protein and Expression
  dplyr::rename(Protein = variable, Expression = value) %>%
  # Convert Protein from factor to character
  mutate(Protein = as.character(Protein))

# Bind the identifiers and adjusted protein expression groups together
labels.for.loess = cbind(identifiers, plasticity.value.classification) %>%
  # Restructure the dataframe by grouping the columns together using the
common identifiers
  melt(id = colnames(identifiers)) %>%
  # Rename columns as Protein and Label
  dplyr::rename(Protein = variable, Label = value) %>%
```

```
  # Convert Protein from factor to character
  mutate(Protein = as.character(Protein))

# Merge the data.for.loess and labels.for.loess dataframes together
graphable.data = merge(data.for.loess, labels.for.loess, by =
c(colnames(identifiers), "Protein")) %>%
  # Convert Expression from character to numeric and Label from character to
factor
  mutate(Expression = as.numeric(Expression),
         Label = factor(Label, levels = c("Raw", "Imputed")))

# Convert the Protein column to a factor to specify the order for facets
graphable.data$Protein = factor(graphable.data$Protein, levels =
as.character(unique(unlist(graphable.data$Protein))))

# Create a vector of the colours that we will use for each Label in the plot
cols = c("Raw" = "black", "Imputed" = "Green")

# Create the plots
plasticity.loess.curves = ggplot(graphable.data, aes(x = Years, y =
Expression)) +
  scale_x_continuous(trans = "log2", breaks = c(0.3, 1, 5, 10, 25, 50, 80)) +
  geom_point(aes(color = Label), size = 1.5, shape = 19) +
  scale_color_manual(values = cols, name = "Expression Label") +
  facet_wrap(~Protein, ncol = 8, scales = "free_y") +
  stat_smooth(method = "loess", color = "black") +
  labs(title = "Plasticity Markers - Imputed Data", x = "Age (Years)", y =
"Z-Score") +
  theme_bw() +
  theme(panel.background = element_rect(fill = 'gray95'),
        axis.text.x = element_text(angle=45,vjust=0.5),
        axis.title.y = element_text(size = 17),
        axis.title.x =element_text(size = 17),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        plot.title = element_text(hjust = 0.5, size = 20))
ggplot.loess.export = ggplot_build(plasticity.loess.curves)$data


# pdf("LOESS Curves for 23 Plasticity Proteins.pdf")
# Use facet_multiple() so the pdf can have the plots span over multiple pages
plasticity.facet.loess = facet_multiple(plot = plasticity.loess.curves,
                                        facets = 'Protein',
                                        ncol = 2,
                                        nrow = 2,
                                        scales = "free")

# dev.off()
# plasticity.facet.loess
```

Export Plasticity LOESS Values

See the supplementary file, Section D, for these figures/plots.

```
 # Export LOESS Curves
  plasticity.loess.CI.values =
as.data.frame(rep(levels(graphable.data$Protein), each = 80)) # create a
dataframe with the protein names
  plasticity.loess.CI.values = cbind(plasticity.loess.CI.values,
ggplot.loess.export[[2]]) # bind the exported loess values to the correct
protein
  colnames(plasticity.loess.CI.values)[1] = "Protein" # set first column name

  # Exporting LOESS values through ggplot_build exports the x-values in a
log2 transformed format.
  # I want to get the actual values so I convert them back.
  plasticity.loess.CI.values$x.values = 2^ (plasticity.loess.CI.values$x)

  # Plot the loess values as a check that the z-score values are being
exported correctly
  gg.loess = ggplot(plasticity.loess.CI.values, aes(x.values, y))+
    geom_line() +
    scale_x_continuous(trans='log2',breaks = c(0.3, 1, 5, 10, 25, 50,
80))+#transforms the x axis scale
    facet_wrap(~Protein, ncol = 8, scales = "free_y")+
    labs(x = "Age (Years)", y = "Z-Score") +
    ggtitle("Lifespan Trajectories: 72 Neuroimmune Proteins in Human V1C")+
    theme_bw()+
    theme(panel.background = element_rect(fill = 'gray95'),
          axis.text.x = element_text(angle=45,vjust=0.5),
          axis.title.y = element_text(size = 17),
          axis.title.x =element_text(size = 17),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          plot.title = element_text(hjust = 0.5, size = 20))
  # pdf(paste("LOESS Values ", title, ".pdf"))
  stamp.gg.loess = facet_multiple(plot = gg.loess,
                          facets = 'Protein',
                          ncol = 2,
                          nrow = 2,
                          scales = "free_y")

 # dev.off()
 # stamp.gg.loess
```

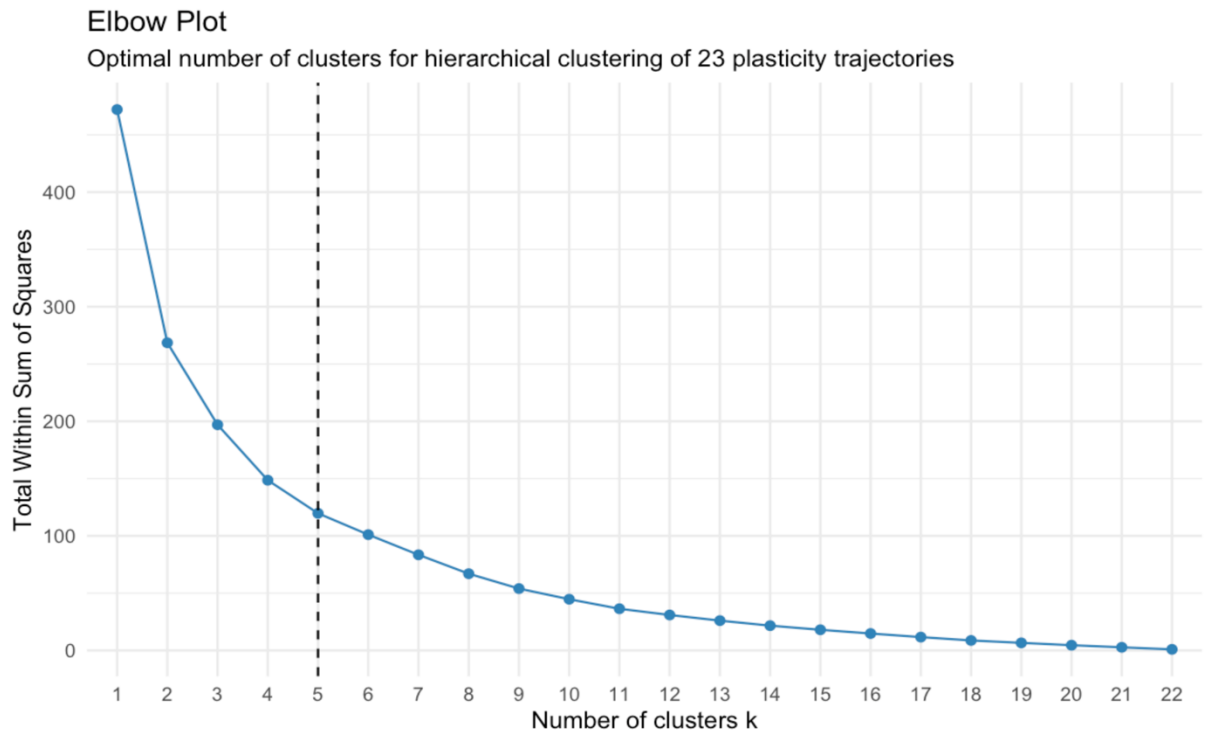
Create a data frame with these loess values in the wide format (for easily processing data in further analyses).

```
loess.23.proteins = dcast(plasticity.loess.CI.values, formula =  x.values ~
Protein, value.var="y")
```

WSS for the Plasticity Markers

This is FIGURE 15 in my thesis.

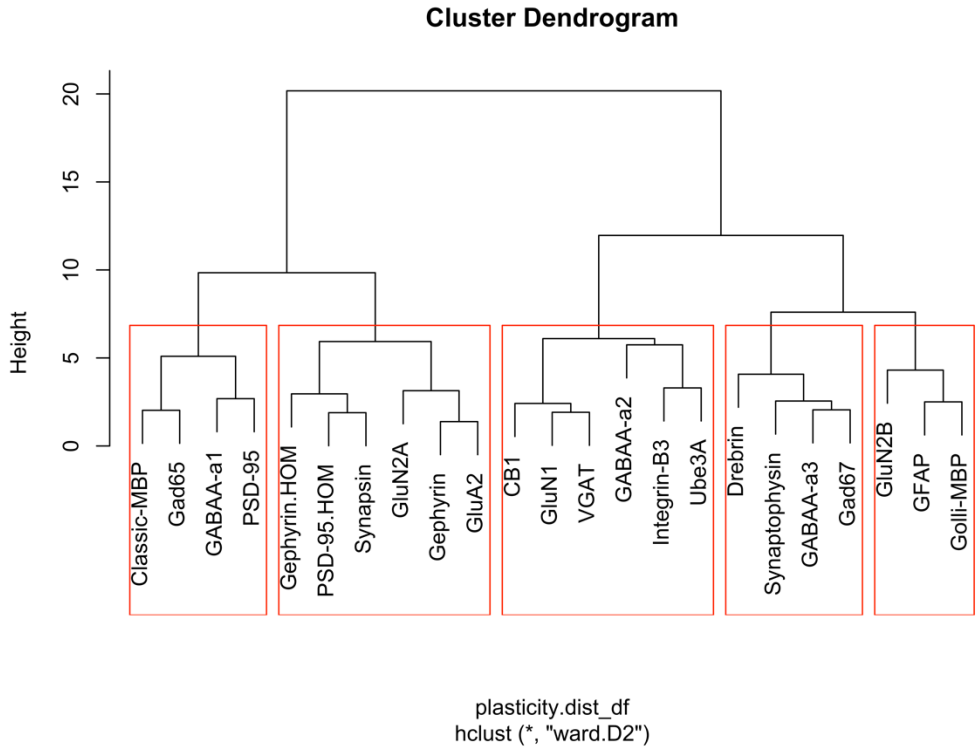
```
data.for.optimal.clus = as.data.frame(t(loess.23.proteins[-1])) # format data
frame so that the proteins are rows
set.seed(123) # set seed for reproducibility
print(fviz_nbclust(data.for.optimal.clus, FUNcluster = hcut, method = "wss",
k.max = 22) +
        scale_x_discrete(breaks = seq(1, 22, 1), labels = seq(1, 22, 1), ) +
        geom_vline(xintercept = 5, linetype="dashed", color = "black") +
        labs(title = "Elbow Plot", subtitle = "Optimal number of clusters for
hierarchical clustering of 23 plasticity trajectories",
             y = "Total Within Sum of Squares") +
        theme(axis.text.x = element_text(size = 12)) +
        theme_minimal())
```



Plasticity Marker Dendrogram

This is part A of FIGURE 16 in my thesis.

```
  plasticity.dendrogram.data = as.data.frame(loess.23.proteins[-1] %>% t()) #
transpose data frame, do not use age column
  plasticity.dist_df = dist(plasticity.dendrogram.data, method = "euclidean")
# calcluate distance matrix
  plasticity.dendrogram = hclust(plasticity.dist_df, method = "ward.D2") #
perform hierarchical clustering
  plasticity.split_tree = cutree(plasticity.dendrogram, k = 5) # solit the
tree into 5 branches
  name = paste("ward.D2", "23.cluster.members", sep = ".") # create a name to
store the cluster members
  assign(name, split(names(plasticity.split_tree), plasticity.split_tree)) #
assign the names to the cluster designation object

# png("Plasticity Dendrogram - No Extra Labels .png", height = 6, width = 8,
units = "in", res = 800)
 plasticity.dend.plot = plot(plasticity.dendrogram, cex = 1) # plot the
dendrogram, add hang = -1 when getting tree with no labels
 rect.hclust(plasticity.dendrogram, k = 5, border = "red") # add rectangles #
remove when simply getting tree with no labels
```

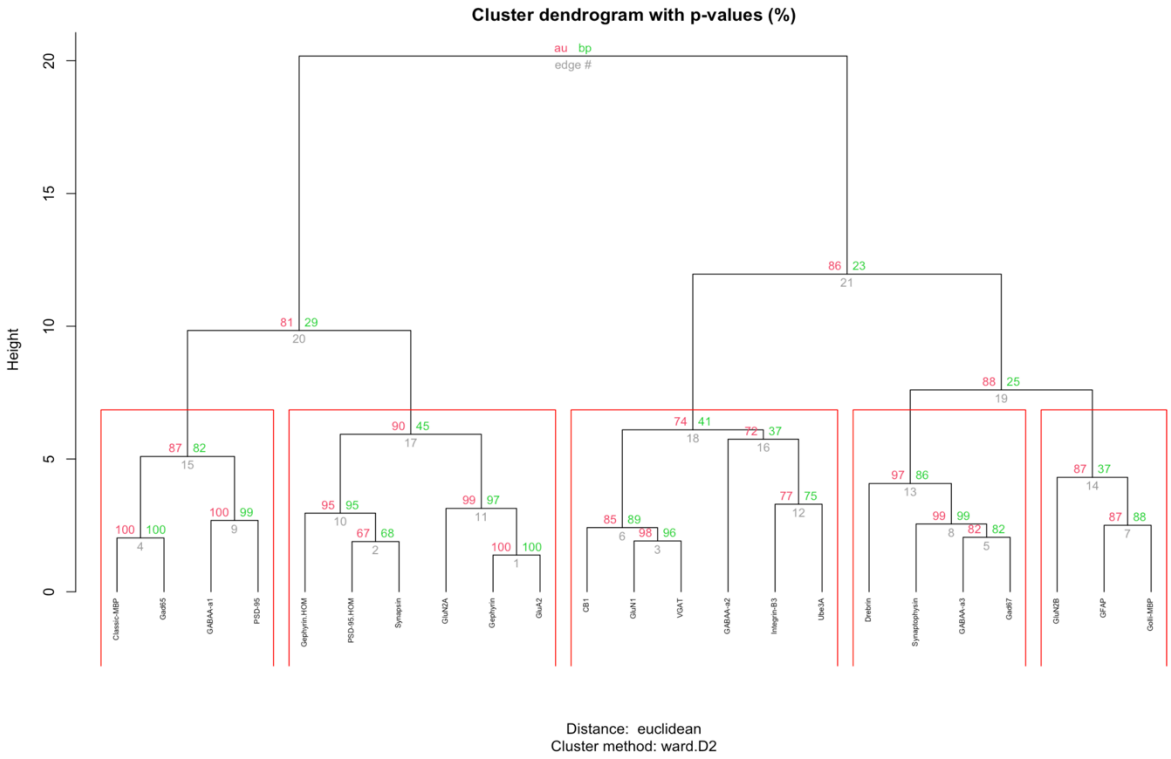



```
# dev.off()
```

```
names(ward.D2.23.cluster.members) = c("3", "1", "4", "2", "5")
ward.D2.23.cluster.members =
ward.D2.23.cluster.members[order(names(ward.D2.23.cluster.members))]

order.plasticity.dend =
order.dendrogram(as.dendrogram(plasticity.dendrogram))
order.plasticity.dend =
rownames(plasticity.dendrogram.data)[order.plasticity.dend]
```

Obtain P-Values for Hierarchical Clustering

```
# 23 Synaptic Only

set.seed(123)
cluster.23.pv = pvclust(loess.23.proteins[-1] , method.hclust = "ward.D2",
                 method.dist = "euclidean", nboot = 10000)

## Bootstrap (r = 0.5)... Done.
## Bootstrap (r = 0.6)... Done.
## Bootstrap (r = 0.7)... Done.
## Bootstrap (r = 0.8)... Done.
## Bootstrap (r = 0.9)... Done.
## Bootstrap (r = 1.0)... Done.
## Bootstrap (r = 1.1)... Done.
## Bootstrap (r = 1.2)... Done.
## Bootstrap (r = 1.3)... Done.
## Bootstrap (r = 1.4)... Done.

# png(" P-Values 23 Plasticity Trajectories Dendrogram.png", width = 8,
height = 6, units="in", res=1200)
plot(cluster.23.pv, hang = -1, cex = 0.5)  # plot the dendrogram with the
corresponding P-Values
rect.hclust(plasticity.dendrogram, k = 5, border = "red") # Highlight the
clusters
```


Cluster dendrogram with p-values (%)
au bp
edge #
Height
0
5
10
15
20
81 29
20
86 23
21
88 25
19
87 82
15
90 45
17
74 41
18
97 86
13
87 37
14
100 100
4
100 99
9
95 95
10
99 97
11
85 89
6
72 37
16
99 99
8
87 88
7
67 68
2
100 100
1
98 96
3
77 75
12
82 82
5
Classic-MBP
Gad65
GABAA-α1
PSD-95
Gephyrin-HOM
PSD-95-HOM
Synapsin
GluNDA
Gephyrin
GluA2
CB1
GluN1
VGluT
GABAA-α2
Integrin-B3
Ubc9A
Drebrin
Synaptophysin
GABAA-α3
Gad67
GluN2B
GFAP
Golli-MBP
Distance: euclidean
Cluster method: ward.D2

```
# dev.off()
```

Create Average Plasticity Curves

Plot averaged Curves and save the average points as a data frame to the global environment.

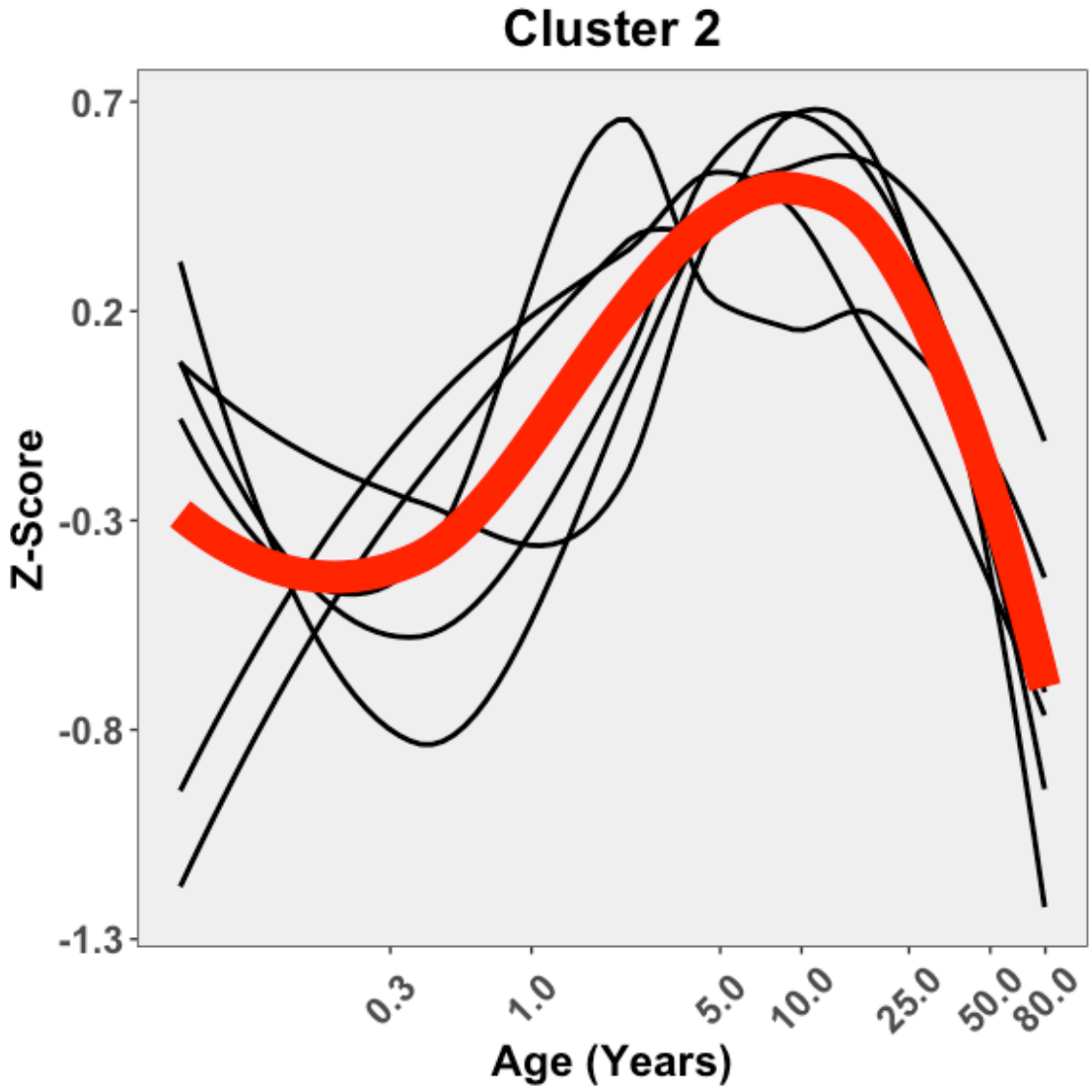
This is part B of FIGURE 16 in my thesis.

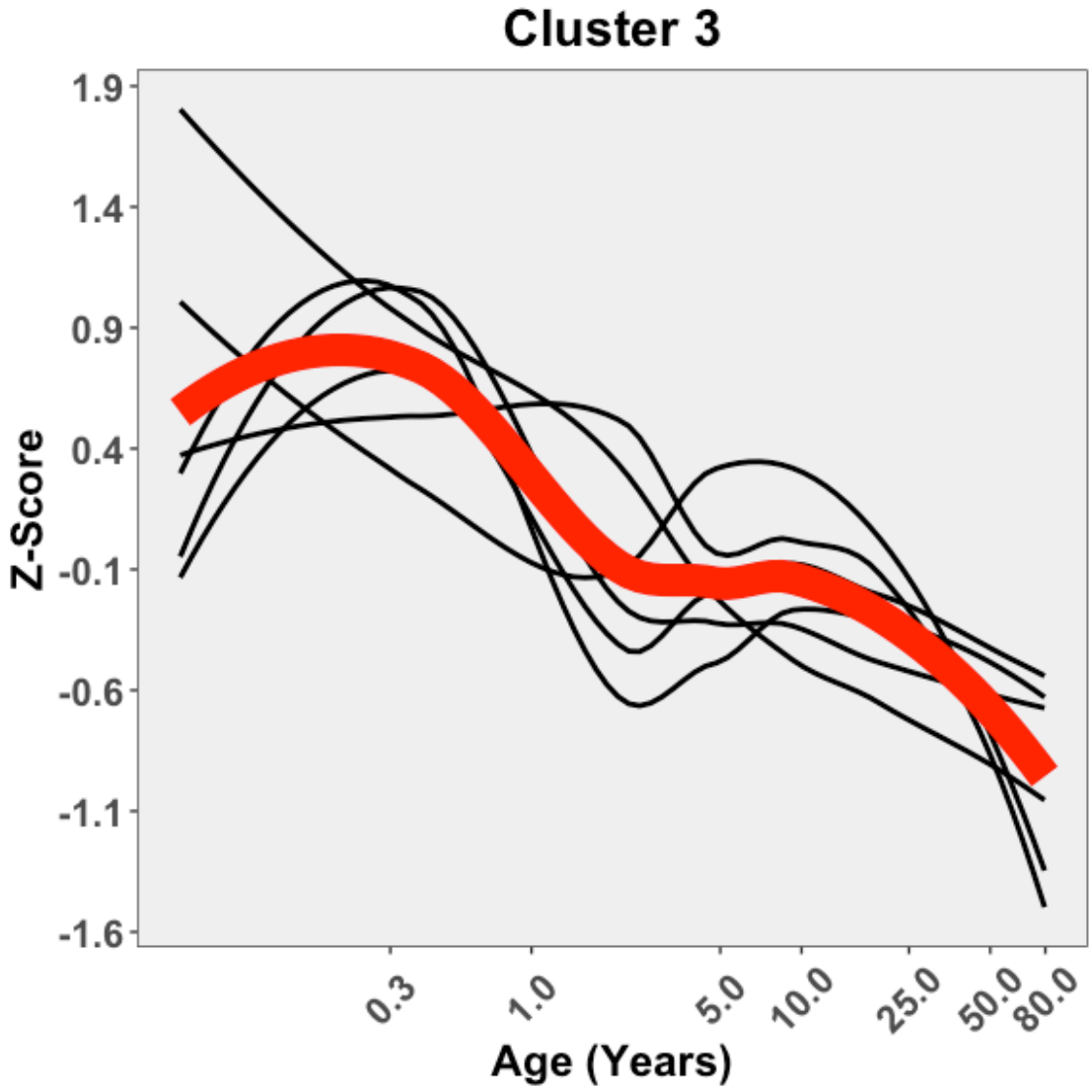
```
# Create a data frame that contains the average curve values for all clusters
avg.loess.23.data = as.data.frame(matrix(ncol = 6, nrow = 80))
avg.loess.23.data[1] = loess.23.proteins$x # Set the x axis values (ie. the
loess age values)
# Set column names
colnames(avg.loess.23.data) = c("Years","Cluster 1","Cluster 2","Cluster
3","Cluster 4",
                                "Cluster 5")

cluster.number = 5 # specify how many clusters

# Create a list that will hold the upper CI, lower CI and mean values for an
average cluster curve each curve will be its own sub-list
avg.curves.23.stats = list()
for(i in 1:5){
    sub.list = list()
    avg.curves.23.stats = list.append(avg.curves.23.stats, sub.list)
    names(avg.curves.23.stats)[[i]] = paste("Cluster", LETTERS[1:5][i])
}

plot.list = list() # create a list to store the plots

# Loop through each cluster and create an average plot
for (i in 1:cluster.number) {

# Extract the loess values for proteins in a specific cluster
ward.d2.clustering.data =
loess.23.proteins[ward.D2.23.cluster.members[[as.character(i)]]]

mean.CI = as.data.frame(matrix(ncol = 3, nrow = 80)) # create data frame to
store upper CI, mean and lower CI
colnames(mean.CI) = c("Upper", "Average", "Lower") # set column names
for(row.number in 1:80) { # loop through all the rows and calculate the
upper, mean and lower CI
mean.CI[row.number,] =
CI(as.numeric(unlist(ward.d2.clustering.data[row.number, ])), ci = 0.95) #
calculate Mean, Upper and Lower CI
}
```

```
# Double check mean calculation using another function to verify the row
means
  if (identical(rowMeans(ward.d2.clustering.data), mean.CI$Average) == FALSE)
{
    stop("Means are being calculated incorrectly")
  }

# Add the x values to the data farme
ward.d2.clustering.data$x = loess.23.proteins$x
# Add the mean values to the averge cluster data frame
avg.loess.23.data[i+1] = mean.CI$Average
# Add the mean values to the plotting data frame
ward.d2.clustering.data$Average = mean.CI$Average
# melt the data frame and store in separate variable for plotting
melted.ward.d2.clustering.data = reshape2::melt(ward.d2.clustering.data, id =
"x")
# merge the individual protein expression data frame with the mean, upper and
lower CI data frame
ward.d2.clustering.data = cbind(ward.d2.clustering.data, mean.CI)
# take this data frame and store it in a list (will append all the other
clusters to the same list)
avg.curves.23.stats[[i]] = ward.d2.clustering.data
# plot the average curves using melted data frame
print(ggplot(melted.ward.d2.clustering.data, aes(x = x, y = value, group =
variable, size= ifelse(variable=="Average",3, 0.01),
                                                 color =
ifelse(variable=="Average", "red", "black"))) +
      scale_color_manual(values = c("black", "red"))+
      geom_line() +
      scale_x_continuous(trans='log2',breaks = c(0.3,1,5,10,25,50,80))+ #
transforms the x axis to log scale
      scale_y_continuous(breaks =
seq(round(min(melted.ward.d2.clustering.data$value) - 0.1, 1),

round(max(melted.ward.d2.clustering.data$value) + 0.1, 1),
                                       by = 0.5)) +
       labs(y = "Z-Score", x = "Age (Years)") +
      # labs(y = NULL, x = NULL) + # add line when making plots with no
labels, remove the other
      theme_bw()+
      theme(panel.background = element_rect(fill = 'gray95'),
            axis.text.x = element_text(size = 15, angle = 45, vjust = 0.5,
face = "bold"),
            axis.text.y = element_text(size = 15, face = "bold"),
            axis.title.y = element_text(size = 17, face = "bold"),
            axis.title.x =element_text(size = 17, face = "bold"),
            title = element_text(face = "bold"),
            # axis.ticks = element_blank(), # add lines when making plots
with no labels, remove the other
            # axis.text.x = element_blank(),
```

```
            # axis.text.y = element_blank(),
            legend.position = "none",
            panel.grid.major = element_blank(),
            panel.grid.minor = element_blank(),
            plot.title = element_text(hjust = 0.5, size = 20))+
       ggtitle(paste("Cluster", i, sep = " "))
      # ggtitle(NULL) # add line when making plots with no labels, remove the
other
      )

  #   ggsave(paste("23 Average Curve no labels Cluster ", i, ".png", sep =
""), device = "png", height = 6, width = 6)
}
```


Cluster 1
0.8
0.3
-0.2
-0.7
-1.2
-1.7
-2.2
Z-Score
0.3
1.0
5.0
10.0
25.0
50.0
80.0
Age (Years)


Cluster 2
0.7
0.2
-0.3
-0.8
-1.3
Z-Score
0.3
1.0
5.0
10.0
25.0
50.0
80.0
Age (Years)


Cluster 3
1.9
1.4
0.9
0.4
-0.1
-0.6
-1.1
-1.6
Z-Score
0.3
1.0
5.0
10.0
25.0
50.0
80.0
Age (Years)


Cluster 4
0.7
0.2
-0.3
Z-Score
0.3
1.0
5.0
10.0
25.0
50.0
80.0
Age (Years)


```
cluster.number = 8 # set back to 8 for all other clusters
```

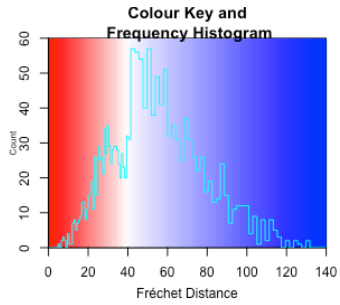
Fréchet Distances and Estimation Statistics

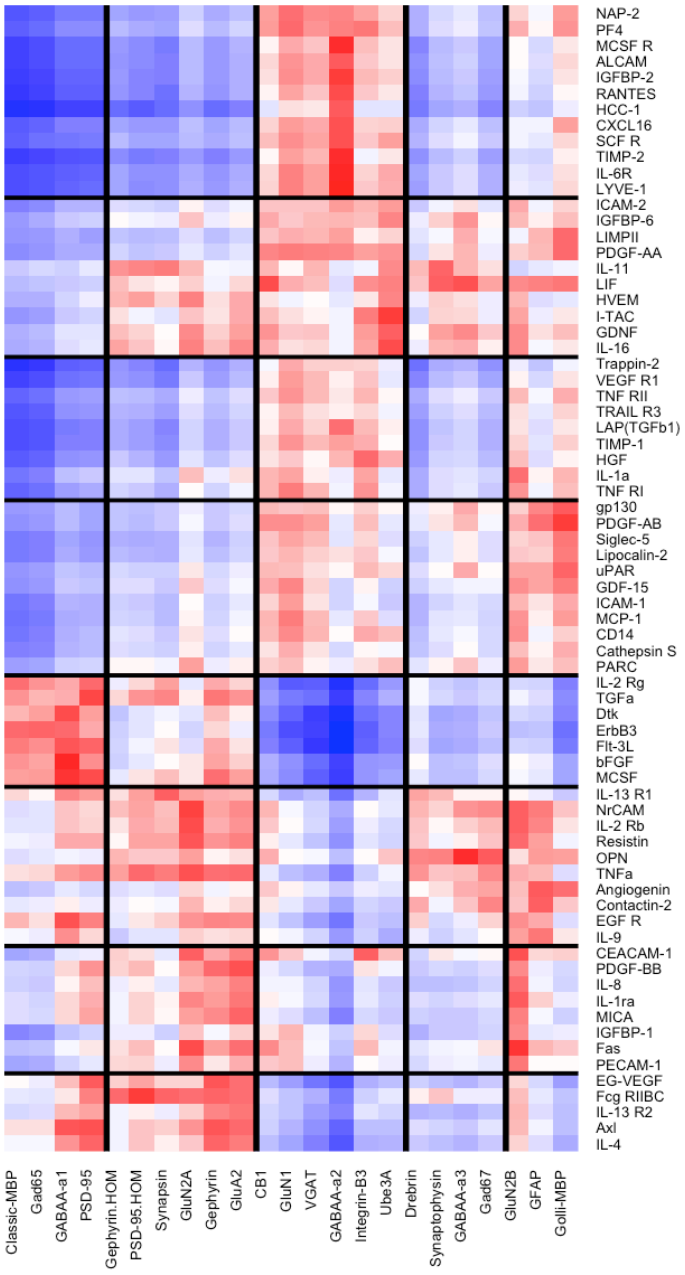
Calculate the Fréchet distance between the neuroimmune proteins and the plasticity marker data.

This is FIGURE 18 in my thesis.

```
# Create 72 x 23 Matrix
frechet.scores.proteins = expand.grid(colnames(loess.72.proteins[-1]),
colnames(loess.23.proteins[-1]))  # get combination of proteins to compare
(including self-comparisons)
frechet.scores.proteins$Distance = NA # create empty column to store the
distances
colnames(frechet.scores.proteins)[1:2] = c("Variable 1", "Variable 2")
x.value = loess.72.proteins$x # set the x values (ages) for the trajectories
(here my proteins were measured in the same samples, so only 1 unique x-value
vector needed)

for(i in 1:nrow(frechet.scores.proteins)){
    trajectory1 =
as.numeric(unlist(loess.72.proteins[as.character(frechet.scores.proteins[i,1]
)])) # export the z-score values for the first protein
    trajectory2 =
as.numeric(unlist(loess.23.proteins[as.character(frechet.scores.proteins[i,2]
)])) # export the z-score values for the second protein
    frechet.scores.proteins[i,3] = as.numeric(distFrechetR(x.value,
trajectory1, x.value, trajectory2, FrechetSumOrMax = "sum")) # calculate
distance, use the sum method
}


# Determine median distance
median.immune.synaptic.frechet = median(frechet.scores.proteins$Distance) #
around 42.8


# Convert from long to wide format for plotting
frechet.immune.synaptic.heatmap.data = spread(frechet.scores.proteins,
`Variable 2`, Distance) %>%
                column_to_rownames("Variable 1")

# Use the max value to determine how high the color map must go (the lowest
value will be 0)
max(frechet.immune.synaptic.heatmap.data)

## [1] 126.286

# Set colour scheme for data frame : Use same colour scale for comparisons
paletteLength = 99 # indicates how many gradations are wanted in the color
key
```

```
col = colorRampPalette(c("red", "white", "blue"))(paletteLength) # make the
color scale
# assign where the breaks must occur
mybreaks = c(seq(0,37,length=49),  # Distances up to 37 are coloured on a red
gradient
             seq(38,39,length=2),  # Distances between 38-39 are coloured
white
             seq(40,135,length=49)) # Distances between 40 and 135 are
coloured using a blue gradient

# reorder the columns and rows by the  hierarchical clustering ward.D2
dendrogram
frechet.immune.synaptic.heatmap.data.2 =
frechet.immune.synaptic.heatmap.data[order.immune.dend,
order.plasticity.dend]

# png("Neuroimmune-Plasticity Frechet Heatmap.png", height = 15, width = 10,
units = "in", res = 1000)
heatmap.2(as.matrix(frechet.immune.synaptic.heatmap.data.2), # specify data
frame
          col = col, # specify colour scale
          breaks = mybreaks, # specify breaks
          Rowv = order.immune.dend, # order of columns
          Colv = order.plasticity.dend, # order of rows
          dendrogram = "none", # apply default dendrogram (no)
          trace = "none", # do not draw a trace
          sepwidth = c(0.2,0.2), # create lines showing clusters
          sepcolor = "black", # colour of line
          cexRow = 1, # font size rows
          cexCol = 1, # font size columns
          key.xlab = "Fréchet Distance", # Key x label
          key.title = "Colour Key and\n Frequency Histogram", # Key title
          key.ylab = "Count", # Key y label
          lhei = c(2, 8), # Height of  key and the plot
          lwid = c(4, 8), # Width  of  key and the plot
          colsep = c(4, 10, 16, 20), # Where to draw column lines to show
clusters
          rowsep=c(12,22,31,42,49,59,67), # Where to draw row lines to show
clusters
          margins = c(8,8), # set margins for the plot
          key.xtickfun = function() { # labels for the Key
            breaks = pretty(parent.frame()$breaks)
            list(at = parent.frame()$scale01(breaks),
                 labels = breaks)
          }
          )
```


Colour Key and
Frequency Histogram
Count
0
10
20
30
40
50
60
0
20
40
60
80
100
120
140
Fréchet Distance

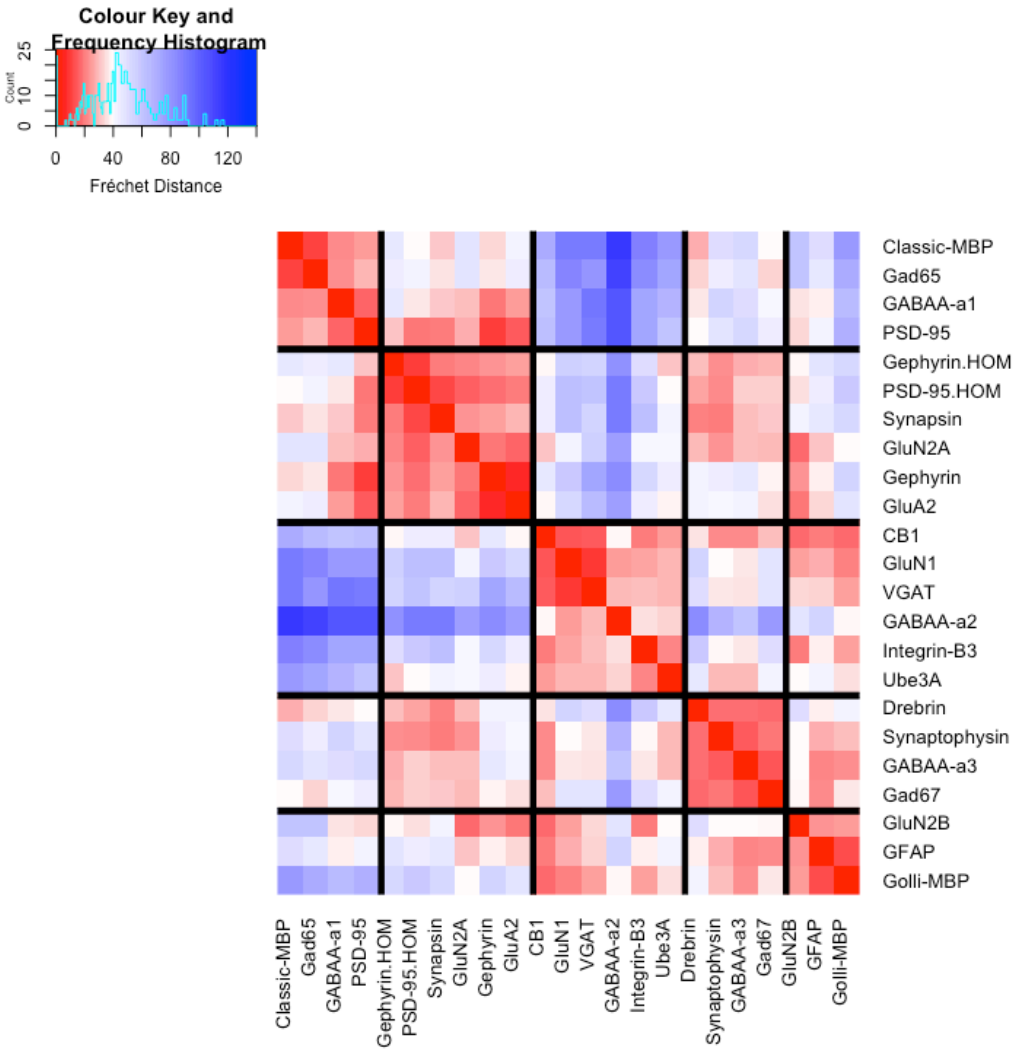

NAP-2
PF4
MCSF R
ALCAM
IGFBP-2
RANTES
HCC-1
CXCL16
SCF R
TIMP-2
IL-6R
LYVE-1
ICAM-2
IGFBP-6
LIMPII
PDGF-AA
IL-11
LIF
HVEM
I-TAC
GDNF
IL-16
Trappin-2
VEGF R1
TNF RII
TRAIL R3
LAP(TGFb1)
TIMP-1
HGF
IL-1a
TNF RI
gp130
PDGF-AB
Siglec-5
Lipocalin-2
uPAR
GDF-15
ICAM-1
MCP-1
CD14
Cathepsin S
PARC
IL-2 Rg
TGFa
Dtk
ErbB3
Flt-3L
bFGF
MCSF
IL-13 R1
NrCAM
IL-2 Rb
Resistin
OPN
TNFa
Angiogenin
Contactin-2
EGF R
IL-9
CEACAM-1
PDGF-BB
IL-8
IL-1ra
MICA
IGFBP-1
Fas
PECAM-1
EG-VEGF
Fcg RIIBC
IL-13 R2
Axl
IL-4
Classic-MBP
Gad65
GABAA-a1
PSD-95
Gephyrin.HOM
PSD-95.HOM
Synapsin
GluN2A
Gephyrin
GluA2
CB1
GluN1
VGAT
GABAA-a2
Integrin-B3
Ube3A
Drebrin
Synaptophysin
GABAA-a3
Gad67
GluN2B
GFAP
Golli-MBP

```
# dev.off()
```

Estimation Statistics Plasticity Only

```
# Create 23 x 23 Matrix
frechet.scores.proteins = expand.grid(colnames(loess.23.proteins[-1]),
colnames(loess.23.proteins[-1]))  # get combination of proteins to compare
(including self comparisons)
frechet.scores.proteins$Distance = NA # create empty column to store the
distances
colnames(frechet.scores.proteins)[1:2] = c("Variable 1", "Variable 2")
x.value = loess.23.proteins$x # set the x values (ages) for the trajectories
(here my proteins were measured in the same samples, so only 1 unique x-value
vector needed)

for(i in 1:nrow(frechet.scores.proteins)){
    trajectory1 =
as.numeric(unlist(loess.23.proteins[as.character(frechet.scores.proteins[i,1]
)])) # export the z-score values for the first protein
    trajectory2 =
as.numeric(unlist(loess.23.proteins[as.character(frechet.scores.proteins[i,2]
)])) # export the z-score values for the second protein
    frechet.scores.proteins[i,3] = as.numeric(distFrechetR(x.value,
trajectory1, x.value, trajectory2, FrechetSumOrMax = "sum")) # calculate
distance, use the sum method
}


# Determine median distance
median.plasticity.frechet = median(frechet.scores.proteins$Distance) # around
36

# Convert from long to wide format for plotting
frechet.plasticity.heatmap.data = spread(frechet.scores.proteins, `Variable
2`, Distance) %>%
                column_to_rownames("Variable 1")


# Ensure that the color scale covers the max
max(frechet.plasticity.heatmap.data)

## [1] 112.6456

# Use the max value to determine how high the color map must go (the lowest
value will be 0)


# Set colour scheme for data frame : Use same colour scale as neuroimmune for
comparisons
paletteLength = 99 # indicates how many gradations are wanted in the color
key
col = colorRampPalette(c("red", "white", "blue"))(paletteLength) # make the
```

```
color scale
# assign where the breaks must occur
mybreaks = c(seq(0,37,length=49),  # Distances up to 37 are coloured on a red
gradient
             seq(38,39,length=2),  # Distances between 38-39 are coloured
white
             seq(40,135,length=49)) # Distances between 40 and 135 are
coloured using a blue gradient

# reorder the columns and rows by the  hierarchical clustering ward.D2
dendrogram
frechet.plasticity.heatmap.data.2 =
frechet.plasticity.heatmap.data[order.plasticity.dend, order.plasticity.dend]

# png("23 Protein Frechet Heatmap.png", height = 15, width = 15, units =
"in", res = 1000)
heatmap.2(as.matrix(frechet.plasticity.heatmap.data.2), # specify data frame
          col = col, # specify colour scale
          breaks = mybreaks, # specify breaks
          Rowv = order.plasticity.dend, # order of columns
          Colv = order.plasticity.dend, # order of rows
          dendrogram = "none", # apply default dendrogram (no)
          trace = "none", # do not draw a trace
          sepwidth = c(0.2,0.2), # create lines showing clusters
          sepcolor = "black", # colour of line
          cexRow = 1, # font size rows
          cexCol = 1, # font size columns
          key.xlab = "Fréchet Distance", # Key x label
          key.title = "Colour Key and\n Frequency Histogram", # Key title
          key.ylab = "Count", # Key y label
          lhei = c(2, 8), # Height of  key and the plot
          lwid = c(3, 8), # Width  of  key and the plot
          colsep = c(4, 10, 16, 20), # Where to draw column lines to show
clusters
          rowsep= c(4, 10, 16, 20), # Where to draw row lines to show
clusters
          margins = c(10,10), # set margins for the plot
          key.xtickfun = function() { # labels for the Key
            breaks = pretty(parent.frame()$breaks)
            list(at = parent.frame()$scale01(breaks),
                 labels = breaks)
          }
          )
```


Colour Key and
Frequency Histogram
Count
0
10
25
0
40
80
120
Fréchet Distance
Classic-MBP
Gad65
GABAA-a1
PSD-95
Gephyrin.HOM
PSD-95.HOM
Synapsin
GluN2A
Gephyrin
GluA2
CB1
GluN1
VGAT
GABAA-a2
Integrin-B3
Ube3A
Drebrin
Synaptophysin
GABAA-a3
Gad67
GluN2B
GFAP
Golli-MBP

```
# dev.off()
```

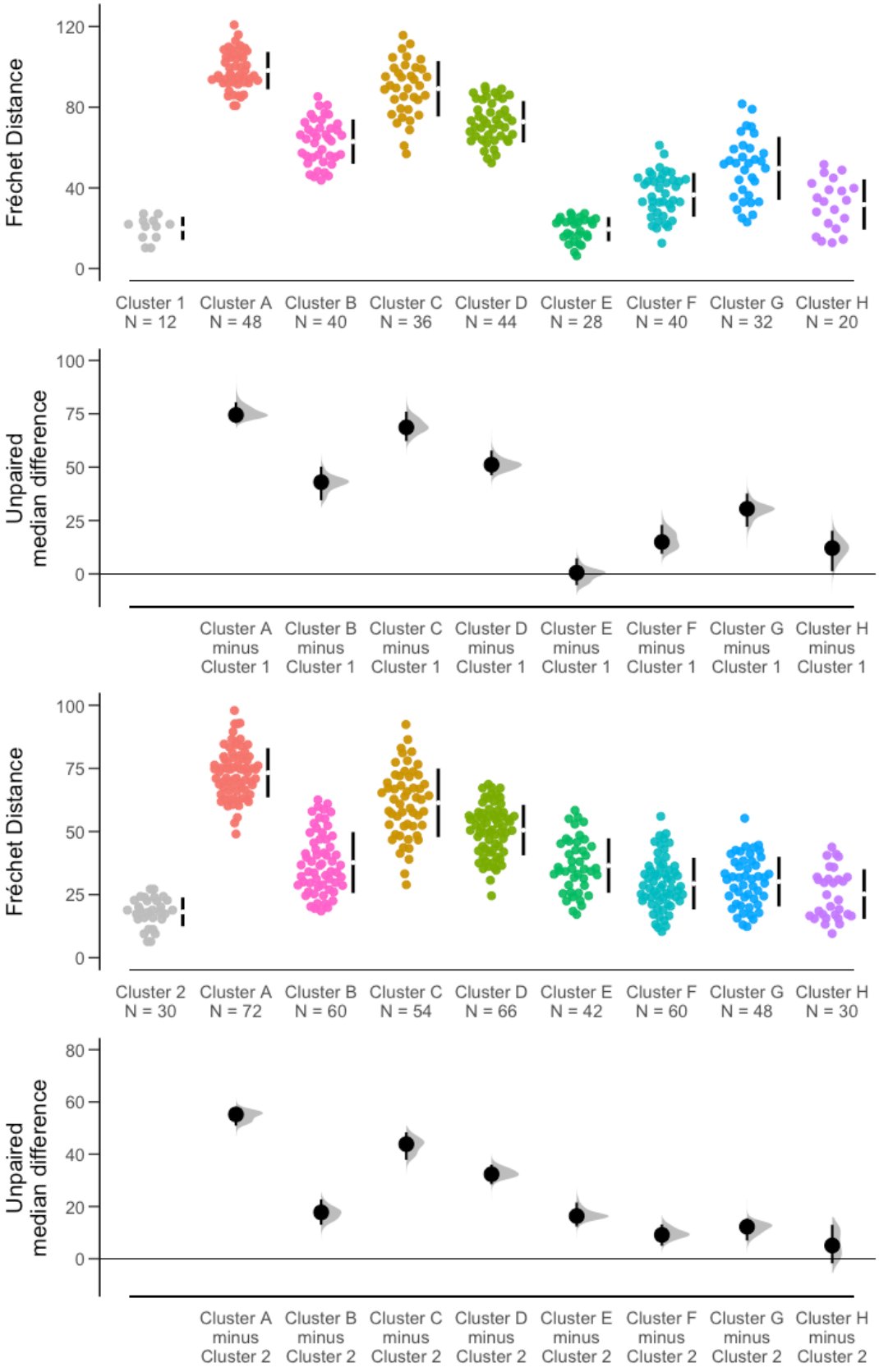
Estimation Statistics Between Immune and Plasticity Data

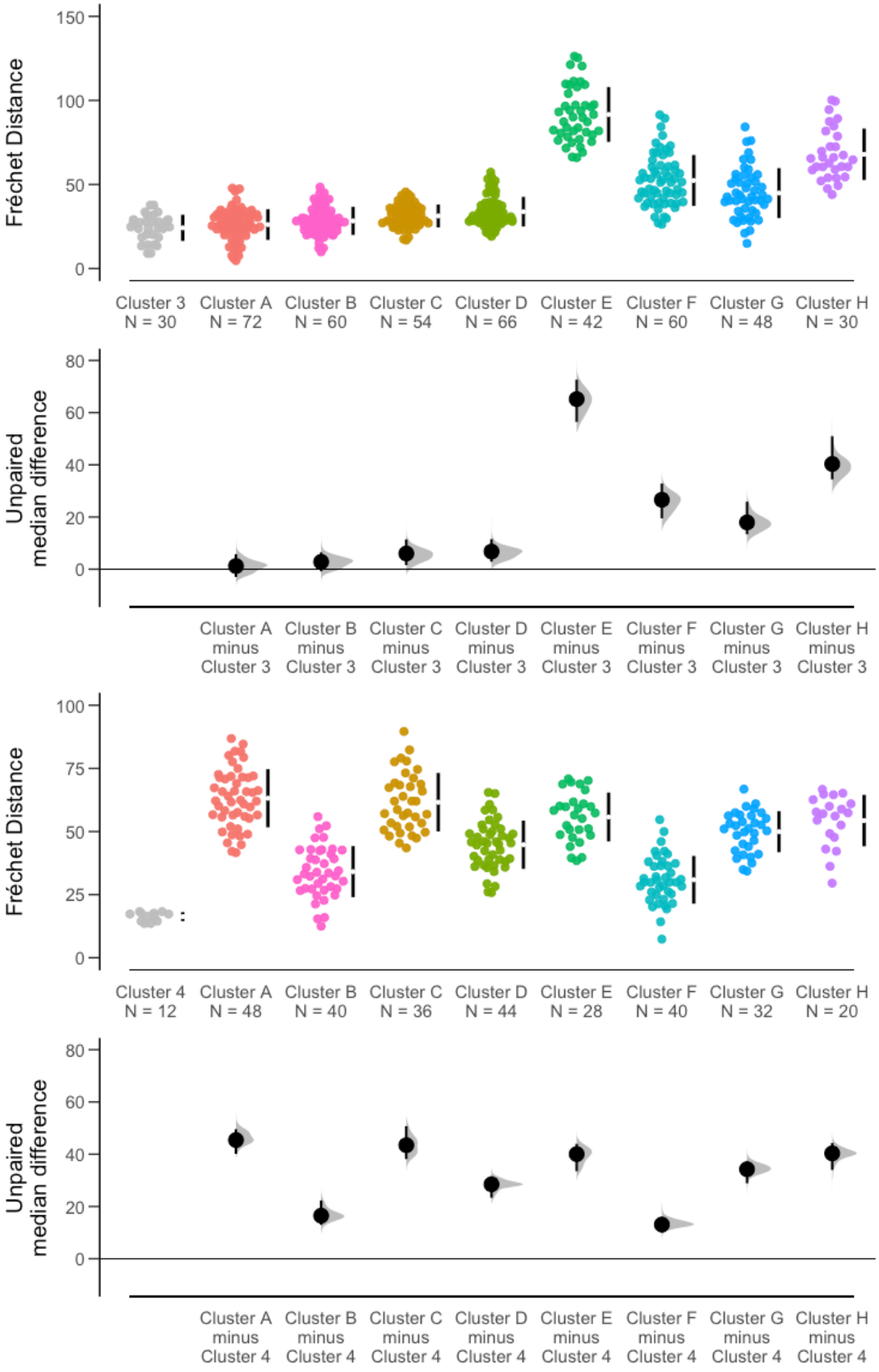
This is FIGURE 19 in my thesis.

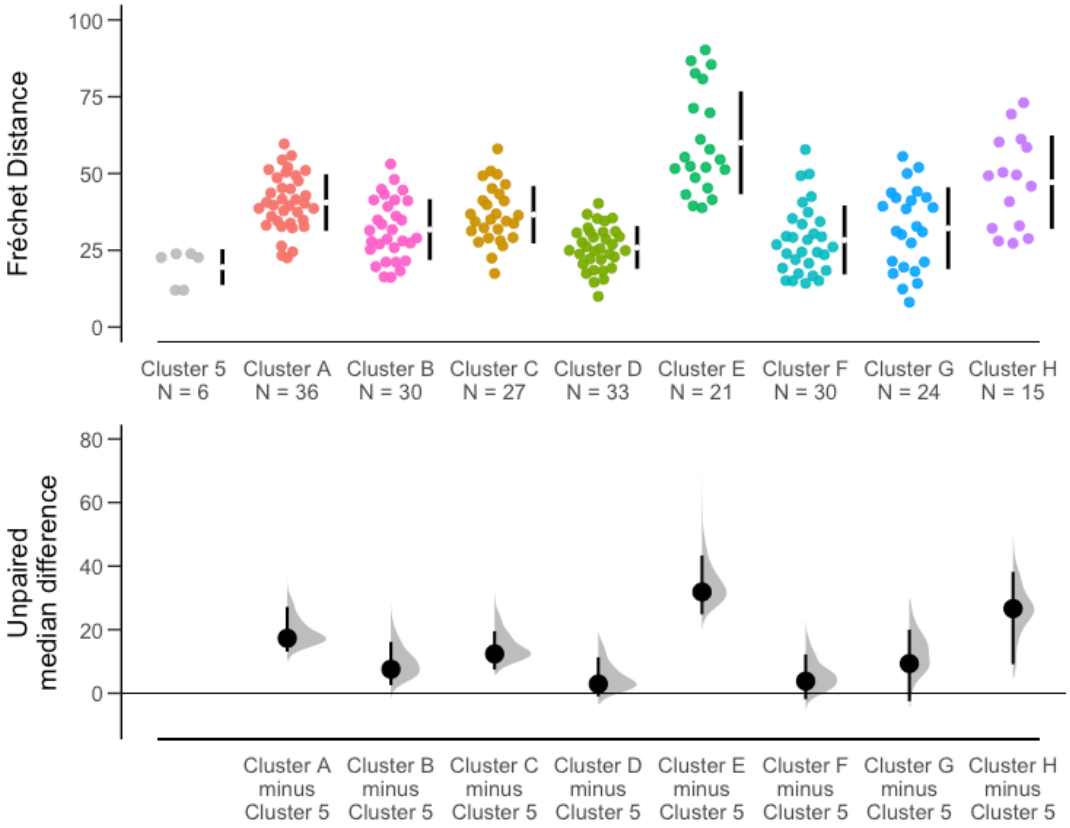
```
cluster.list = c("Cluster 1", "Cluster 2", "Cluster 3", "Cluster 4", "Cluster
5")
cluster.list.immune = c("Cluster A", "Cluster B", "Cluster C", "Cluster D",
"Cluster E", "Cluster F", "Cluster G", "Cluster H")
value.max = c(125, 100, 150, 100, 100)
effect.max = c(100, 80, 80, 80, 80)

total.between.estimation.stats.results.plasticity = as.data.frame(matrix(ncol
= 6)) # create empty data frame to store estimation stats values
colnames(total.between.estimation.stats.results.plasticity) = c("Control",
"Variable", "Median", "Median.Diff", "Low.CI", "High.CI") # set column names

median.within.cluster.plasticity = as.data.frame(matrix(ncol = 2)) # create
empty data frame to store within cluster median
colnames(median.within.cluster.plasticity) = c("Variable", "Median") # set
column names


for (i in 1:5){ # loop through each cluster

  frechet.scores.between.plasticity = as.data.frame(matrix(ncol = 1)) #
create empty data frame

  for (j in 1:8){ # loop through each cluster (combined with previous loop,
allows (8 x8) 64 comparisons to be made)

    frechet.subset =
frechet.immune.synaptic.heatmap.data[ward.D2.cluster.members[[j]],
ward.D2.23.cluster.members[[i]]] # get frechet scores for proteins
    frechet.subset = as.numeric(as.character(unlist(frechet.subset))) #
convert values to numeric

    frechet.scores.between.plasticity =
qpcR:::cbind.na(frechet.scores.between.plasticity, frechet.subset) # bind the
frechet scores
    colnames(frechet.scores.between.plasticity)[1+j] = paste("Cluster",
names(ward.D2.cluster.members[j])) # set column names
  }

  frechet.scores.between.plasticity = frechet.scores.between.plasticity[-1] #
remove the empty column
  frechet.scores.between.plasticity =
reshape2::melt(frechet.scores.between.plasticity, na.rm = TRUE) # covert the
data frame to the long format for plotting
```

```
  # Between Cluster Estimation Stats

  between.estimation.stats.results.plasticity = as.data.frame(matrix(ncol =
6, nrow = 8)) # create empty data frame to store between cluster estimation
stats
  colnames(between.estimation.stats.results.plasticity) = c("Control",
"Variable", "Median", "Median.Diff", "Low.CI", "High.CI") # set column names
  estimation.stats.data = frechet.scores.between.plasticity # use the
completed data frame of frechet distances for one cluster as the estimation
statistics data
  colnames(estimation.stats.data)[2] = "Fréchet Distance"

  # Get the intracluster correlations for the plasticity cluster
  estimation.stats.self =
as.data.frame(as.numeric(unlist(frechet.plasticity.heatmap.data[ward.D2.23.cl
uster.members[[i]], ward.D2.23.cluster.members[[i]]])))
  colnames(estimation.stats.self) = "Fréchet Distance" # set column name
  estimation.stats.self$variable = cluster.list[[i]] # specify which
plastciicty cluster is being currently examined
  estimation.stats.self = estimation.stats.self[c(2,1)] # reorder the data
frame to match the estimation.stats.data for rbinding

    if (identical(sum(estimation.stats.self == 0),
length(ward.D2.23.cluster.members[[i]])) == FALSE) {
    stop("More zeros than simply self-comparisons")
  }

  estimation.stats.self[estimation.stats.self == 0] = NA # remove self-
comparisons (as these will have a floor effect)
  estimation.stats.data = rbind(estimation.stats.self, estimation.stats.data)
  estimation.stats.data = na.omit(estimation.stats.data)


  control = cluster.list[[i]] # set the control variable
  comparison.groups = cluster.list.immune # set the comparison groups

  # Prepare data for estimations statistics processing
  two.group.unpaired =
    estimation.stats.data %>%
    dabest(variable, `Fréchet Distance`,
           idx = c(control, comparison.groups),
           paired = FALSE)

  # Calculate effect size
  two.group.unpaired.mediandiff.between = median_diff(two.group.unpaired)

# png(paste("Estimation Statistics Plasticity Cluster ", LETTERS[i], ".png",
sep = ""), height = 6, width = 8, units = "in", res = 1000)
```

```
  print(plot(two.group.unpaired.mediandiff.between,
  palette = c("grey", "#F8766D", "#FF61CC", "#CD9600", "#7CAE00", "#00BE67",
"#00BFC4", "#00A9FF", "#C77CFF"),
  rawplot.ylim = c(0, value.max[[i]]),
  effsize.ylim = c(-10, effect.max[[i]]),
  rawplot.type = "swarmplot"))
#  dev.off()

  # Save the name of the control group
  between.estimation.stats.results.plasticity["Control"] =
two.group.unpaired.mediandiff.between[["result"]][["control_group"]]
  # Save the name of each of the test groups
  between.estimation.stats.results.plasticity["Variable"] =
two.group.unpaired.mediandiff.between[["result"]][["test_group"]]
  # Save the median frechet between each test group and the control group
  between.estimation.stats.results.plasticity["Median"] =
two.group.unpaired.mediandiff.between[["summary"]][["median"]][2:9]
  # Save the median different between the control and the test group
  between.estimation.stats.results.plasticity["Median.Diff"] =
two.group.unpaired.mediandiff.between[["result"]][["difference"]]
  # Save the low bound of the CI
  between.estimation.stats.results.plasticity["Low.CI"] =
two.group.unpaired.mediandiff.between[["result"]][["bca_ci_low"]]
  # Save the upper bound of the CI
  between.estimation.stats.results.plasticity["High.CI"] =
two.group.unpaired.mediandiff.between[["result"]][["bca_ci_high"]]
  # Bind the individual cluster stats to the data frame that will hold all
cluster values
  total.between.estimation.stats.results.plasticity =
rbind(total.between.estimation.stats.results.plasticity,
between.estimation.stats.results.plasticity)



  # Create data frame to store median intracluster values of control groups
  median.self = as.data.frame(matrix(ncol = 2))
  colnames(median.self) = c("Variable", "Median") # ste column names
  median.self["Variable"] =
two.group.unpaired.mediandiff.between[["result"]][["control_group"]][1] # get
cluster name
  median.self["Median"] =
two.group.unpaired.mediandiff.between[["summary"]][["median"]][1] # get
intracluster distance
  median.within.cluster.plasticity = rbind(median.within.cluster.plasticity,
median.self) # bind values to the data frame holding all cluster stats

}
```


Fréchet Distance
120
80
40
0
Cluster 1 N = 12
Cluster A N = 48
Cluster B N = 40
Cluster C N = 36
Cluster D N = 44
Cluster E N = 28
Cluster F N = 40
Cluster G N = 32
Cluster H N = 20
Unpaired median difference
100
75
50
25
0
Cluster A minus Cluster 1
Cluster B minus Cluster 1
Cluster C minus Cluster 1
Cluster D minus Cluster 1
Cluster E minus Cluster 1
Cluster F minus Cluster 1
Cluster G minus Cluster 1
Cluster H minus Cluster 1
Fréchet Distance
100
75
50
25
0
Cluster 2 N = 30
Cluster A N = 72
Cluster B N = 60
Cluster C N = 54
Cluster D N = 66
Cluster E N = 42
Cluster F N = 60
Cluster G N = 48
Cluster H N = 30
Unpaired median difference
80
60
40
20
0
Cluster A minus Cluster 2
Cluster B minus Cluster 2
Cluster C minus Cluster 2
Cluster D minus Cluster 2
Cluster E minus Cluster 2
Cluster F minus Cluster 2
Cluster G minus Cluster 2
Cluster H minus Cluster 2

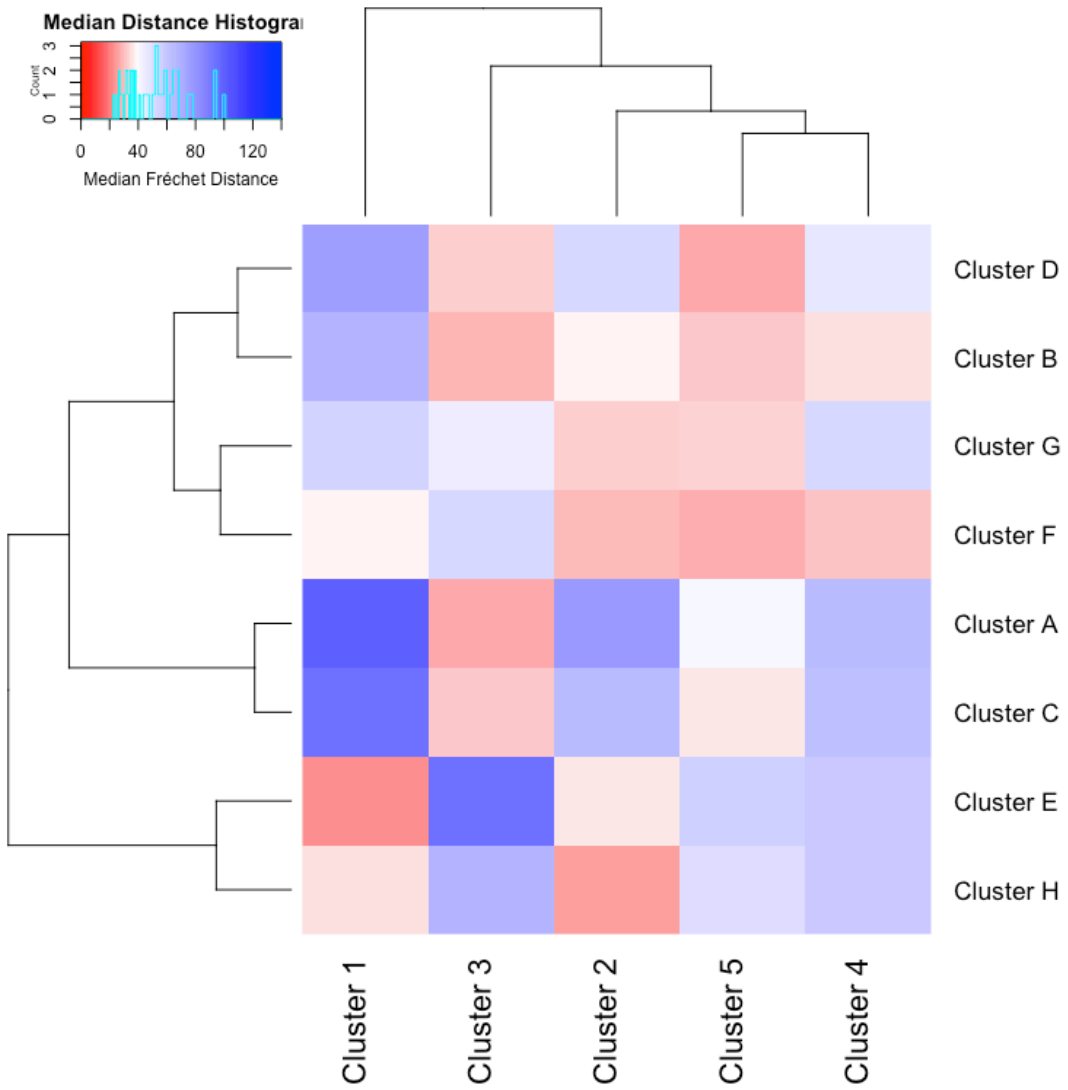

Fréchet Distance
150
100
50
0
Cluster 3 N = 30
Cluster A N = 72
Cluster B N = 60
Cluster C N = 54
Cluster D N = 66
Cluster E N = 42
Cluster F N = 60
Cluster G N = 48
Cluster H N = 30
Unpaired median difference
80
60
40
20
0
Cluster A minus Cluster 3
Cluster B minus Cluster 3
Cluster C minus Cluster 3
Cluster D minus Cluster 3
Cluster E minus Cluster 3
Cluster F minus Cluster 3
Cluster G minus Cluster 3
Cluster H minus Cluster 3
Fréchet Distance
100
75
50
25
0
Cluster 4 N = 12
Cluster A N = 48
Cluster B N = 40
Cluster C N = 36
Cluster D N = 44
Cluster E N = 28
Cluster F N = 40
Cluster G N = 32
Cluster H N = 20
Unpaired median difference
80
60
40
20
0
Cluster A minus Cluster 4
Cluster B minus Cluster 4
Cluster C minus Cluster 4
Cluster D minus Cluster 4
Cluster E minus Cluster 4
Cluster F minus Cluster 4
Cluster G minus Cluster 4
Cluster H minus Cluster 4


Fréchet Distance
100
75
50
25
0
Cluster 5
N = 6
Cluster A
N = 36
Cluster B
N = 30
Cluster C
N = 27
Cluster D
N = 33
Cluster E
N = 21
Cluster F
N = 30
Cluster G
N = 24
Cluster H
N = 15
Unpaired
median difference
80
60
40
20
0
Cluster A
minus
Cluster 5
Cluster B
minus
Cluster 5
Cluster C
minus
Cluster 5
Cluster D
minus
Cluster 5
Cluster E
minus
Cluster 5
Cluster F
minus
Cluster 5
Cluster G
minus
Cluster 5
Cluster H
minus
Cluster 5

Make Summary Figure

This is FIGURE 20 in my thesis.

```
immune.plasticity.summary =
total.between.estimation.stats.results.plasticity[-1,1:3] # remove empty
median(unlist(immune.plasticity.summary$Median)) # This is the median value

## [1] 41.31507

immune.plasticity.summary.2 = dcast(immune.plasticity.summary, Variable ~
Control, value.var = "Median") %>%
   column_to_rownames("Variable")

# Set colour scheme for data frame : Use same colour scale as neuroimmune for
comparisons
paletteLength = 99 # indicates how many gradations are wanted in the color
key
col = colorRampPalette(c("red", "white", "blue"))(paletteLength) # make the
color scale
# assign where the breaks must occur
mybreaks = c(seq(0,37,length=49),  # Distances up to 37 are coloured on a red
gradient
             seq(38,39,length=2),  # Distances between 38-39 are coloured
white
             seq(40,135,length=49)) # Distances between 40 and 135 are
coloured using a blue gradient



# png("Immune Plasticity Summary Heatmap.png", height = 11, width = 12, units
= "in", res = 800)
heatmap.2(as.matrix(immune.plasticity.summary.2),
          col = col,
          breaks = mybreaks,
          dendrogram = "both",
          hclustfun = function(d) hclust(d, method="ward.D2"),
          trace = "none",
          key.xlab = "Median Fréchet Distance",
          key.title = "Median Distance Histogram",
          key.ylab = "Count",
          lhei = c(2, 8),
          lwid = c(3, 8),
          margins = c(8,8),
          key.xtickfun = function() {
            breaks = pretty(parent.frame()$breaks)
            list(at = parent.frame()$scale01(breaks),
                 labels = breaks)
          }
          )
```

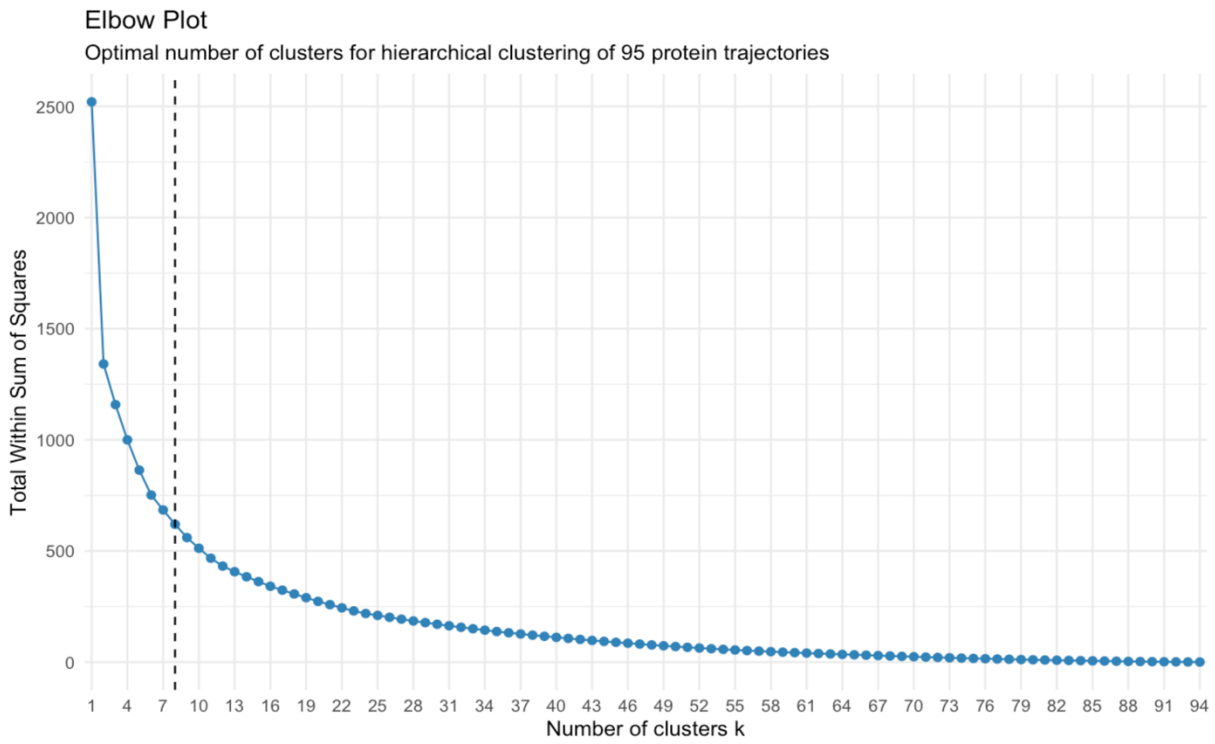

Median Distance Histogra
Count
0 1 2 3
0 40 80 120
Median Fréchet Distance
Cluster D
Cluster B
Cluster G
Cluster F
Cluster A
Cluster C
Cluster E
Cluster H
Cluster 1
Cluster 3
Cluster 2
Cluster 5
Cluster 4

```
# dev.off()
```

WSS on all 95 trajectories

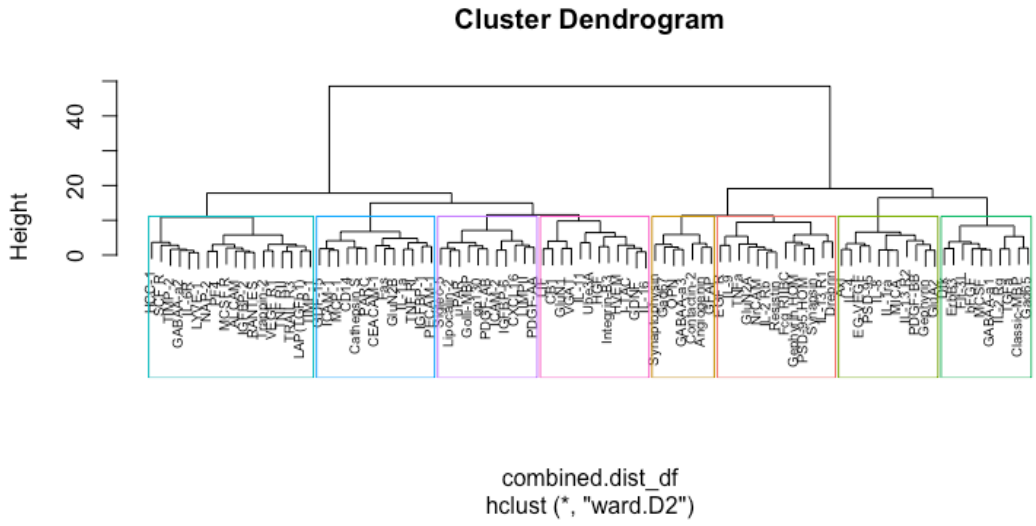
This is FIGURE 21 in my thesis.

```
# Create a data frame with all 95 values
loess.95.proteins = cbind(loess.72.proteins, loess.23.proteins[-1]) # remove
row, so that we do not have the ages twice

# Format the data frame according to the function specifications

# 1. Input should include only the loess values, do not include identifiers
# 2. The rows are the objects to be clustered
  set.seed(123) # set seed for reproducibility
  print(fviz_nbclust(t(loess.95.proteins[,2:ncol(loess.95.proteins)]),
FUNcluster = hcut, method = "wss", k.max = 94) +
    scale_x_discrete(breaks = seq(1, 94, 3), labels = seq(1, 94, 3), ) +
    geom_vline(xintercept = 8, linetype="dashed", color = "black") +
    labs(title = "Elbow Plot", subtitle = "Optimal number of clusters for
hierarchical clustering of 95 protein trajectories",
         y = "Total Within Sum of Squares") +
    theme(axis.text.x = element_text(size = 12)) +
    theme_minimal())
```



Perform hierarchical clustering and get a dendrogram. This is part A of FIGURE 22 in my thesis.

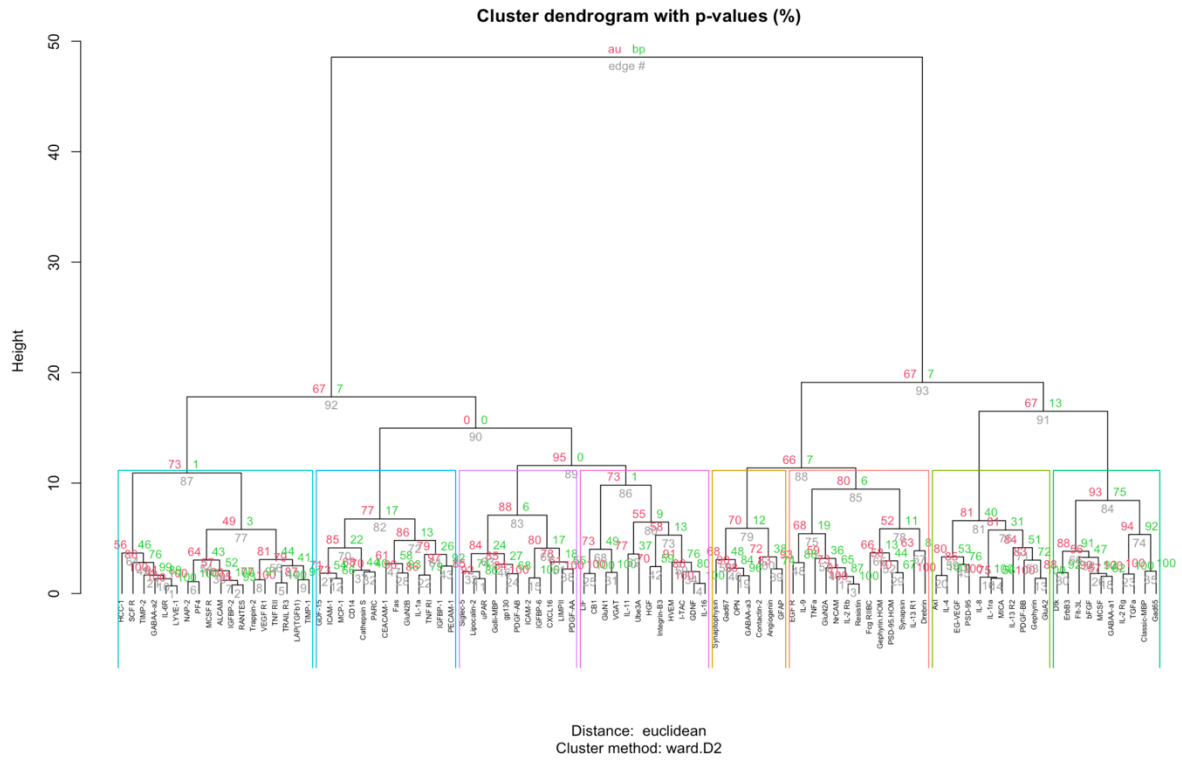
```r
set.seed(1890)

# Set colours for dendrogram: colours are ordered according to SynGO (see
below) --> This step was added later
cols.95.dend = c("#00BFC4", "#00A9FF", "#C77CFF", "#FF61CC", "#CD9600",
"#F8766D", "#7CAE00", "#00BE67")

  combined.dendrogram.data = as.data.frame(loess.95.proteins[-1] %>% t()) #
transpose data frame, do not use age column
  combined.dist_df = dist(combined.dendrogram.data, method = "euclidean") #
calcluate distance matrix
  combined.dendrogram = hclust(combined.dist_df, method = "ward.D2") #
perform hierarchical clustering
  combined.split_tree = cutree(combined.dendrogram, k = 8) # solit the tree
into 8 branches
  name = paste("ward.D2", "95.cluster.members", sep = ".") # create a name to
store the cluster members
  assign(name, split(names(combined.split_tree), combined.split_tree)) #
assign the names to the cluster designation object

# png("Combined Data Dendrogram.png", width = 15, height = 8, units = "in",
res = 800)
 combined.dend.plot = plot(combined.dendrogram, cex = 0.6) # plot the
dendrogram
 rect.hclust(combined.dendrogram, k = 8, border = cols.95.dend) # add
rectangles
```




```r
# dev.off()
```

Obtain P-Values for Hierarchical Clustering

```
# 95 Protein Trajectories
set.seed(123)
cluster.95.pv = pvclust(loess.95.proteins[-1] , method.hclust = "ward.D2",
                 method.dist = "euclidean", nboot = 10000)

## Bootstrap (r = 0.5)... Done.
## Bootstrap (r = 0.6)... Done.
## Bootstrap (r = 0.7)... Done.
## Bootstrap (r = 0.8)... Done.
## Bootstrap (r = 0.9)... Done.
## Bootstrap (r = 1.0)... Done.
## Bootstrap (r = 1.1)... Done.
## Bootstrap (r = 1.2)... Done.
## Bootstrap (r = 1.3)... Done.
## Bootstrap (r = 1.4)... Done.

# png(" P-Values 95 Protein Trajectories Dendrogram.png", width = 8, height = 6, units="in", res=1200)
plot(cluster.95.pv, hang = -1, cex = 0.5)  # plot the dendrogram with the corresponding P-Values
rect.hclust(combined.dendrogram, k = 8, border = cols.95.dend) # Highlight the clusters
```




```
# dev.off()
```

SynGO protein annotations

To determine which proteins have classic synaptic properties, we used the database SynGO to annotate the 95 proteins with synapse-based functions. I simply took the 95 gene symbols and entered them into the SynGO search, and downloaded a file indicating which proteins have synaptic functions. The chart can be viewed below. We then arranged the clusters based on the number of SynGO recognized proteins they contained (from most to least). If two clusters had the same number of SynGO proteins they were organized alphabetically.

```
syn.go.proteins = unique(as.character(unlist(syn.go.classification[1]))) # 20
proteins match synaptic functions or components

# Counts of synaptic proteins per cluster

# Current cluster 1 = 1 SynGO
# Current cluster 2 = 4 SynGO
# Current cluster 3 = 3 SynGO
# Current cluster 4 = 2 SynGO
# Current cluster 5 = 1 SynGO
# Current cluster 6 = 0 SynGO
# Current cluster 7 = 6 SynGO
# Current cluster 8 = 5 SynGO

# Order is therefore 7,8,2,3,4,1,5,6

names(ward.D2.95.cluster.members) =  c("F", "C", "D", "E", "G", "H", "A",
"B")
ward.D2.95.cluster.members =
ward.D2.95.cluster.members[order(names(ward.D2.95.cluster.members))]
```

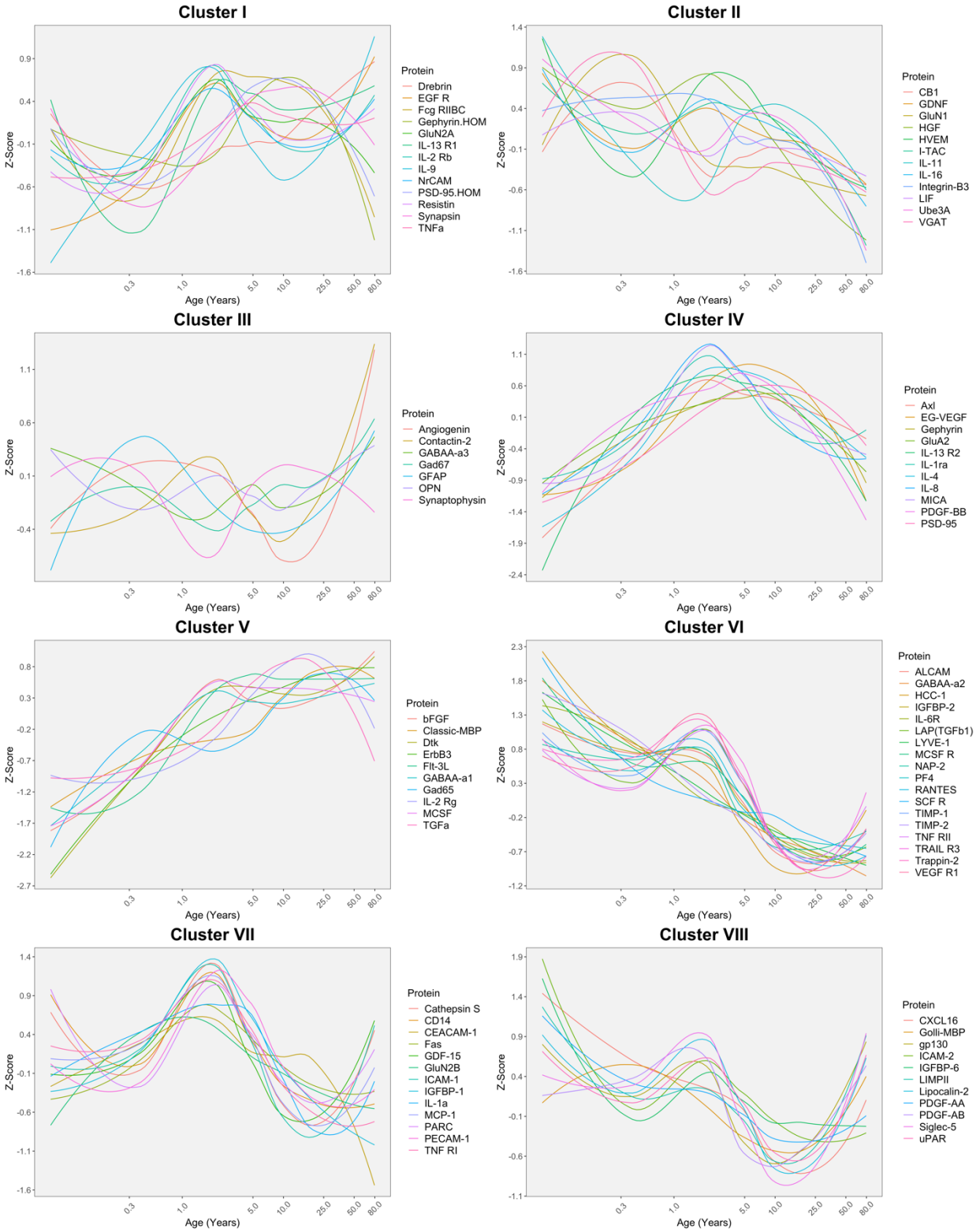
Combined Clusters - Proteins Individually Labelled

This is FIGURE 24 in my thesis.

```
cluster.labels = LETTERS[1:8] # create cluster labels from A to H
cluster.labels.title = c("I", "II", "III", "IV", "V", "VI", "VII", "VIII")

labelled.combined.plot.list = list() # create a list that will store the plot
for each cluster

for (i in 1:cluster.number){
# extract the LOESS values for the proteins in a specific cluster
ward.d2.clustering.data =
cbind(loess.95.proteins$x,loess.95.proteins[ward.D2.95.cluster.members[[clust
er.labels[i]]]])
# the column for plotting the x-axis (Age in Years) is called "x"
colnames(ward.d2.clustering.data)[1] = "x"
```

```
# convert the data frame to a long format for plotting, make the protein
names a character value
melted.ward.d2.clustering.data = melt(ward.d2.clustering.data, id = "x") %>%
    mutate(variable = as.character(variable))

# Create plots
labelled.combined.plot.list[[i]] = ggplot(melted.ward.d2.clustering.data,
aes(x = x, y = value, colour = variable)) +
                            geom_line() +
                            scale_x_continuous(trans='log2', breaks =
c(0.3,1,5,10,25,50,80))+ #transforms the x axis to log scale
                            # set y breaks and labels that increase by 0.5,
start labelling 0.1 below the min up to 0.1 more than the max value
                            scale_y_continuous(breaks =
seq(round(min(melted.ward.d2.clustering.data$value) - 0.1, 1),

round(max(melted.ward.d2.clustering.data$value) + 0.1, 1),
                                                                by = 0.5)) +
                            labs(x = "Age (Years)", y = "Z-Score", colour =
"Protein") +
                            theme_bw()+
                            theme(panel.background = element_rect(fill =
'gray95'),
                            axis.text.x = element_text(size = 12,
angle=45,vjust=0.5),
                            axis.text.y = element_text(size = 12),
                            axis.title.y = element_text(size = 15),
                            axis.title.x =element_text(size = 15),
                            panel.grid.major = element_blank(),
                            panel.grid.minor = element_blank(),
                            plot.title = element_text(hjust = 0.5, size = 25,
face = "bold"),
                            legend.text=element_text(size= 15),
                            legend.title = element_text(size = 15)) +
                            ggtitle(paste("Cluster", cluster.labels.title[i]))
}


# Plots have varying sizes due to the varying size of the legend, must make
them all the same.
# Pick the plot with the largest legend size and use that for all plots
(largest, will have most members)
# Cannot use smallest legend or the other legends will not fit in the
allocated space
g.labelled.C = ggplotGrob(labelled.combined.plot.list[[3]])

# Create a list that stores the plots with the new legend size
labelled.combined.plot.list.new = list()
for (i in 1:cluster.number){
```

```
  g = ggplotGrob(labelled.combined.plot.list[[i]])
  g$widths = g.labelled.C$widths
  g$heights = g.labelled.C$heights
  labelled.combined.plot.list.new[[i]] = g
}

# Plot all the figures on the same page
ggpubr::ggarrange(plotlist = labelled.combined.plot.list.new, ncol = 2, nrow
= 4)
```



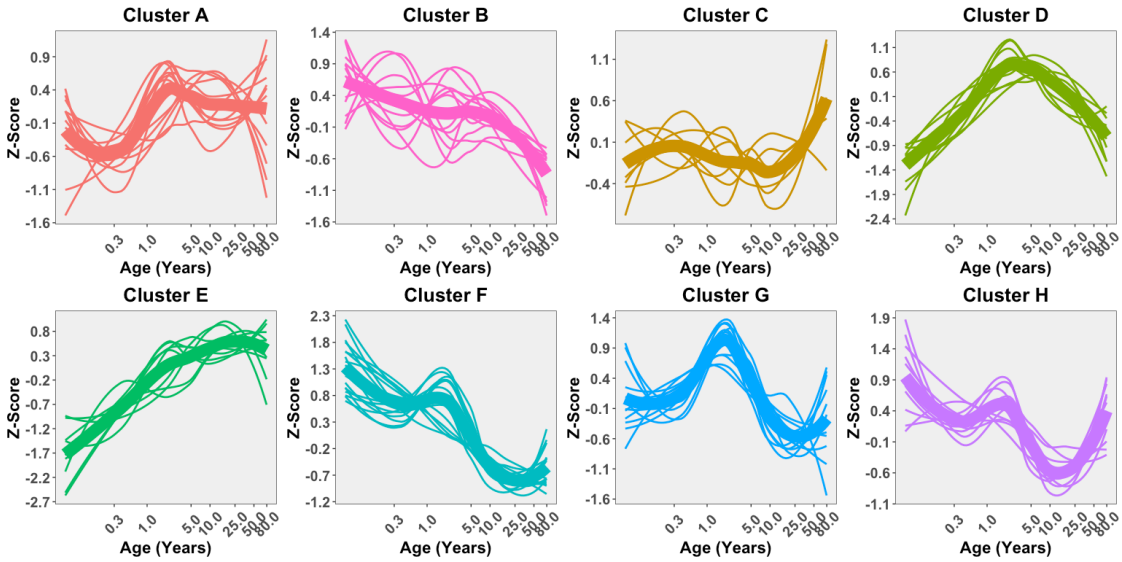

```
# ggsave("Combined Protein Clusters - Proteins Lablled.png", device = "png",
height = 25, width = 20)
```

Combined Data - Averaged Curves

This is part B of FIGURE 22 in my thesis. Note that the Clusters are labelled with the alphabet for ease in writing the code. These labels were then edited to Roman Numerals outside of R. For example, "A" = "I"; "B" = "II" etc. in my thesis.

```
# Set colours for plotting.
cols = c( "#F8766D", "#FF61CC", "#CD9600", "#7CAE00", "#00BE67", "#00BFC4",
"#00A9FF", "#C77CFF")

# Create a data frame that contains the average curve values for all clusters
avg.loess.95.data = as.data.frame(matrix(ncol = 9, nrow = 80))
avg.loess.95.data[1] = loess.95.proteins$x # Set the x axis values (ie. the
loess age values)
# Set column names
colnames(avg.loess.95.data) = c("Years","Cluster A","Cluster B","Cluster
C","Cluster D",
                                "Cluster E","Cluster F","Cluster G","Cluster
H")

cluster.number = 8 # specify how many clusters

# Create a list that will hold the upper CI, lower CI and mean values for an
average cluster curve each curve will be its own sub list
avg.curves.95.stats = list()
for(i in 1:8){
    sub.list = list()
    avg.curves.95.stats = list.append(avg.curves.95.stats, sub.list)
    names(avg.curves.95.stats)[[i]] = paste("Cluster", LETTERS[1:8][i])
}

avg.95.plot.list = list() # create a list to store the plots

# Loop through each cluster and create an average plot
for (i in 1:cluster.number) {

# Extract the loess values for proteins in a specific cluster
ward.d2.clustering.data =
loess.95.proteins[ward.D2.95.cluster.members[[cluster.labels[i]]]]

mean.CI = as.data.frame(matrix(ncol = 3, nrow = 80)) # create data frame to
store upper CI, mean and lower CI
colnames(mean.CI) = c("Upper", "Average", "Lower") # set column names
for(row.number in 1:80) { # loop through all the rows and calculate the
upper, mean and lower CI
mean.CI[row.number,] =
CI(as.numeric(unlist(ward.d2.clustering.data[row.number, ])), ci = 0.95) #
calculate Mean, Upper and Lower CI
}
```

```
# Double check mean calculation using another function to verify the row
means
  if (identical(rowMeans(ward.d2.clustering.data), mean.CI$Average) == FALSE)
{
    stop("Means are being calculated incorrectly")
  }

# Add the x values to the data farme
ward.d2.clustering.data$x = loess.95.proteins$x
# Add the mean values to the averge cluster data frame
avg.loess.95.data[i+1] = mean.CI$Average
# Add the mean values to the plotting data frame
ward.d2.clustering.data$Average = mean.CI$Average
# melt the data frame and store in separate variable for plotting
melted.ward.d2.clustering.data = reshape2::melt(ward.d2.clustering.data, id =
"x")
# merge the individual protein expression data frame with the mean, upper and
lower CI data frame
ward.d2.clustering.data = cbind(ward.d2.clustering.data, mean.CI)
# take this data frame and store it in a list (will append all the other
clusters to the same list)
avg.curves.95.stats[[i]] = ward.d2.clustering.data
# plot the average curves using melted data frame
avg.95.plot.list[[i]] = ggplot(melted.ward.d2.clustering.data, aes(x = x, y =
value, group = variable, size= ifelse(variable=="Average",4,0.1))) +
      geom_line(color=cols[i]) +
      scale_x_continuous(trans='log2',breaks = c(0.3,1,5,10,25,50,80))+ #
transforms the x axis to log scale
      scale_y_continuous(breaks =
seq(round(min(melted.ward.d2.clustering.data$value) - 0.1, 1),

round(max(melted.ward.d2.clustering.data$value) + 0.1, 1),
                                        by = 0.5)) +
      labs(y = "Z-Score", x = "Age (Years)") +
      theme_bw()+
      theme(panel.background = element_rect(fill = 'gray95'),
            axis.text.x = element_text(size = 15, angle = 45, vjust = 0.5,
face = "bold"),
            axis.text.y = element_text(size = 15, face = "bold"),
            axis.title.y = element_text(size = 17, face = "bold"),
            axis.title.x =element_text(size = 17, face = "bold"),
            title = element_text(face = "bold"),
            legend.position = "none",
            panel.grid.major = element_blank(),
            panel.grid.minor = element_blank(),
            plot.title = element_text(hjust = 0.5, size = 20))+
      ggtitle(paste("Cluster", cluster.labels[i], sep = " "))
}
do.call(grid.arrange, c(avg.95.plot.list, ncol = 4))
```


Cluster A
Cluster B
Cluster C
Cluster D
Cluster E
Cluster F
Cluster G
Cluster H
Z-Score
Age (Years)
0.3
1.0
5.0
10.0
25.0
50.0
80.0

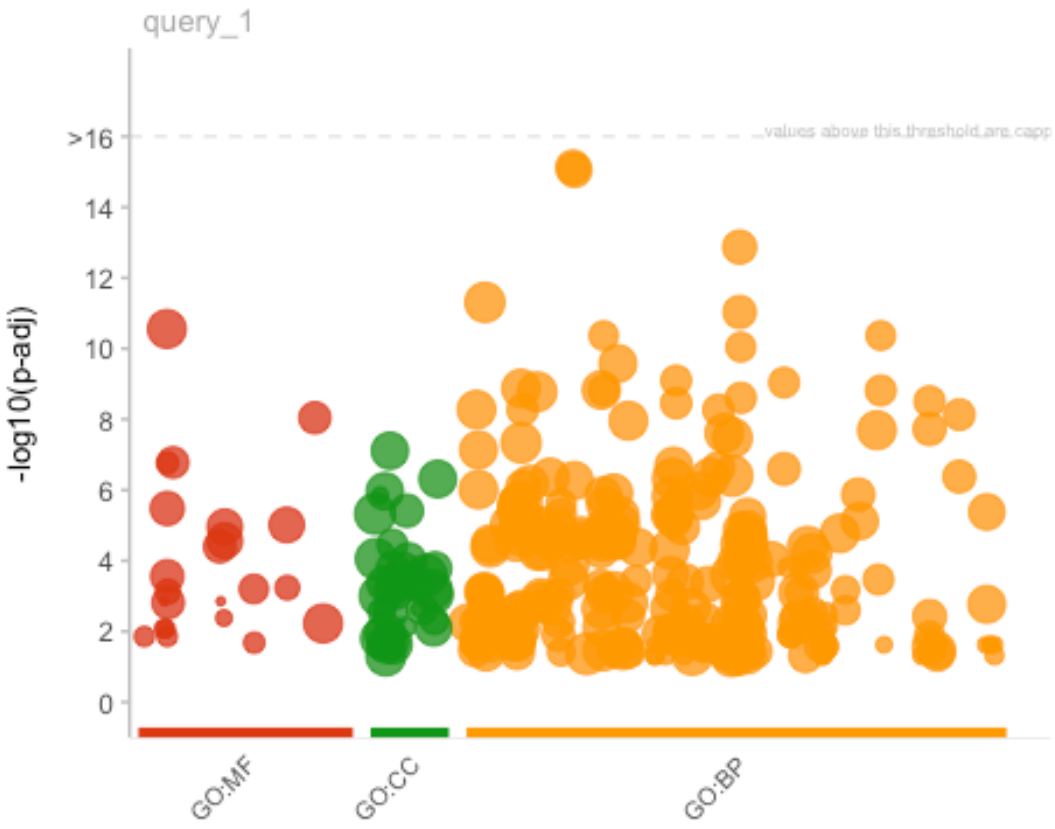
Immune Only Cluster : Functional Partners

The immune only cluster is interesting. It drops to zero in adolescences, a time of onset for SZ. So what are these proteins that are low in expression, and what other proteins (outside of our data set) do they interact with?

I used STRINGdb to predict the functional partners for the proteins in the immune only cluster. I added the partners to the first shell until an integrated network (i.e. all proteins are connected to each other) was formed. This required the addition of 23 predicted functional partners. Below we have the MF, CC, and BP for all the entire network.

```
string.DB.partners = unique(as.character(unlist(string.DB.export[1:2]))) # 33
terms, note that Cluster H contains PDGFAB which is not a unique gene, so it
does not map onto any hits in the STRINGDdb. It is removed from this
analysis.

# I use the alias "SIGLEC5" to replace "OBBP2", as "OBBP2" is not recognized
by the gProfiler algorithm

string.DB.partners[string.DB.partners == "OBBP2"] = "SIGLEC5"

# Perform GO enrichment analysis

cluster.H.gotable = allgenesgotable = gost(query = string.DB.partners, # set
the query terms
                      organism = "hsapiens", # organism of choice is homo
sapiens
                      ordered_query = FALSE,  # the order of proteins is not
important
                      multi_query = FALSE, # there are not multiple list,
consider all terms together
                      significant = TRUE, # only print of significant values
                      exclude_iea = TRUE, # exclude from results the
interactions based on weak evidence
                      measure_underrepresentation = FALSE, # results should
represent enriched terms not underestimated terms
                      evcodes = TRUE, # display a column that indicates
which proteins from query list map onto the enriched term
                      user_threshold = 0.05, # significance threshold, top
0.05
                      correction_method = "gSCS", # the custom correction
method for p-values
                      domain_scope = "annotated", # the background list of
terms to compare my query list is all genes with annotations
                      custom_bg = NULL, # not supplying a custom list
                      numeric_ns = "", # set namespace for fully numeric
gene ID, our is not, so leave at default
                      sources = c("GO:BP", "GO:MF", "GO:CC"), # which
versions of
```

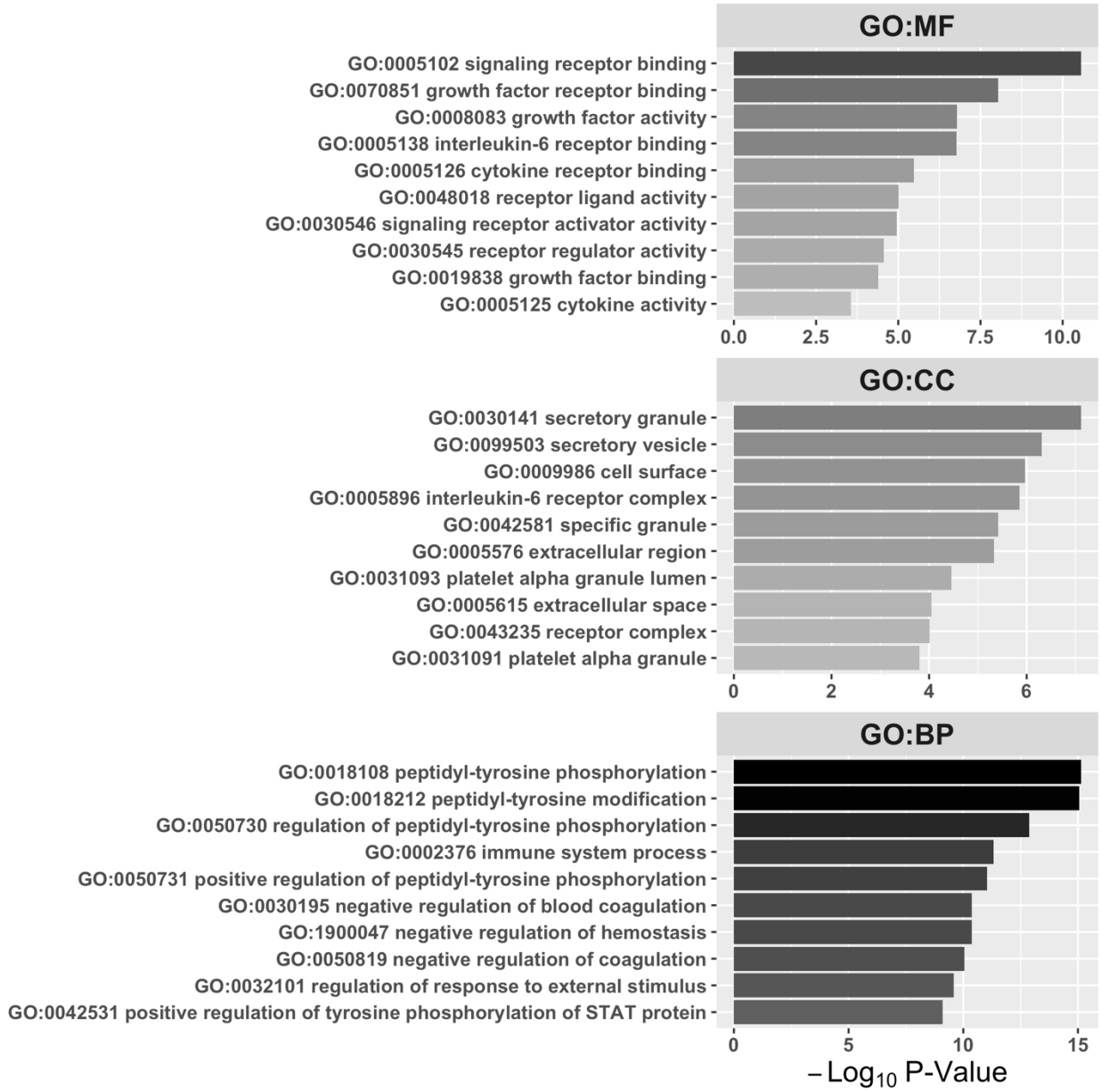
```
                        as_short_link = FALSE) # get the results as a URL set
to FALSE
cluster.H.go.results = cluster.H.gotable$result  # export the enrichment
results
gostplot(cluster.H.gotable, capped = TRUE, interactive = FALSE) # This forms
a Manhatten plot
```



Take the top 10 terms from each GO category and plot their p-values using a bar plot. This is part B of FIGURE 25 in my thesis.

```
# Create lists of the various names for the GO categories to be used for
labelling and data sub setting
go.categories = c("GO:MF", "GO:CC", "GO:BP")
go.category.abbreviation = c("MF", "CC", "BP")
go.category.name = c("Molecular Functions", "Cellular Components",
"Biological Processes")


forplotting = as.data.frame(ncol(16))

for (index.go.category in 1:length(go.categories)){
  # Subset the enrichment performed on all 72 proteins by the gene category
  category.subset = subset(cluster.H.go.results, cluster.H.go.results$source
```

```
== go.categories[index.go.category])
  # Order the terms from greatest to lowest p-value
  category.subset = category.subset[order(category.subset$p_value, decreasing
= FALSE),]
  # Extract the top 25 terms
  forplotting = rbind(forplotting, category.subset[1:10,])

}

  # Combine the GO ID and the full GO term to create labels for the plot
  forplotting$labels = paste(forplotting$term_id, forplotting$term_name, sep
= " ")
  # Turn these labels to factors for graphing in this order
  forplotting$labels = factor(forplotting$labels, levels =
rev(forplotting$labels))
  #Reverse order here as well for facets
  forplotting$source = factor(forplotting$source, levels =
unique(as.character(unlist(forplotting$source))))


  # Use ggplot to create the bar chart
  print(ggplot(forplotting, aes(fill = -log10(p_value), x = -log10(p_value),
y = labels, group = source)) +
    facet_wrap(~source, ncol = 1, scales = "free") +
    geom_bar(stat = "identity") +
    scale_fill_gradient2(low = "white", high = "black", guide = F) +
    xlab(bquote(-Log[10] ~"P-Value")) +
    ylab(NULL)+
   # ggtitle(paste("Enriched GO", go.category.name[index.go.category])) +
    theme(plot.title = element_text(size = 15, hjust = 0.01),
          strip.text = element_text(size = 15, face = "bold"),
          axis.text.y = element_text(size= 10, face = "bold"),
          axis.text.x = element_text(size= 10, face = "bold"),
          axis.title.x = element_text(size=15, face = "bold")))
```




```
 #  ggsave("Top 10 GO Enrichments by Category - Immune Cluster with Predicted
Partners.png", device = "png", height = 8, width = 8)
```

Viral Receptors

This is FIGURE 31 in my thesis.

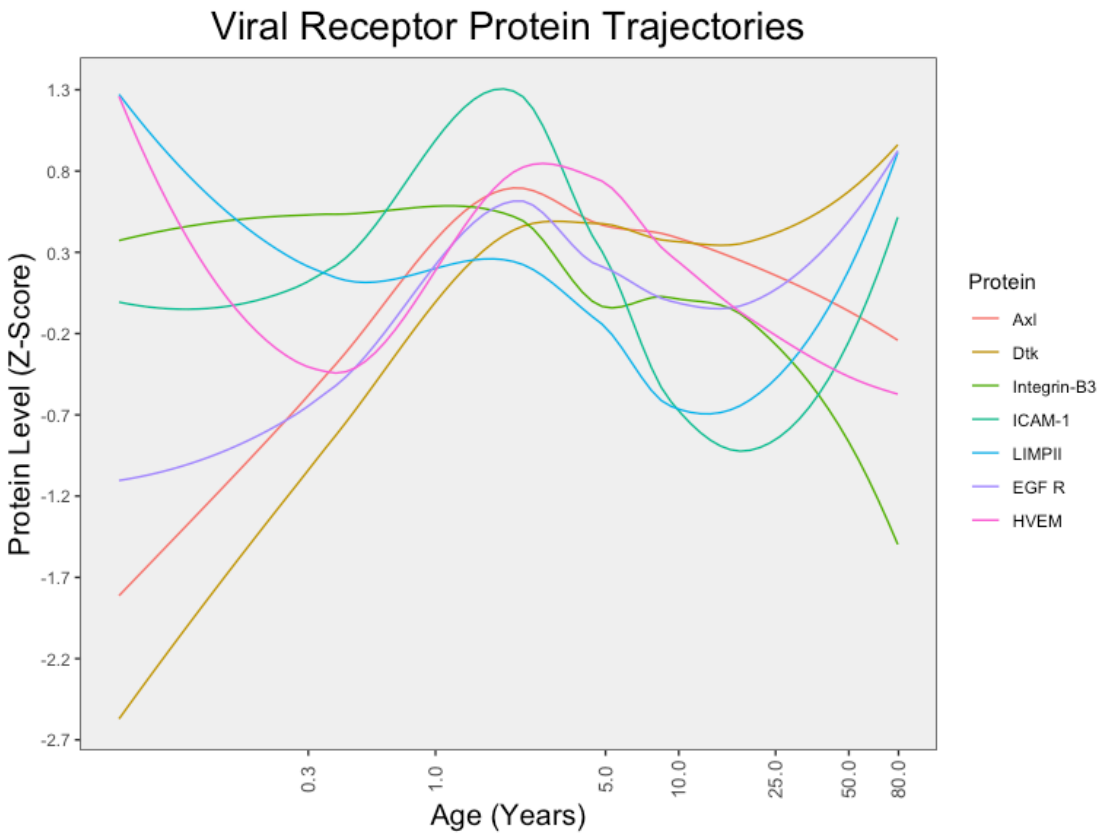
```
viral.receptor.gene.name = uniprot.viral.receptors$`Gene names`
viral.receptor.gene.name = strsplit(viral.receptor.gene.name, " ")
viral.receptor.gene.name = unlist(viral.receptor.gene.name)

# Pick out genes corresponding to proteins in our data set using the Uniprot
keyword search
selected.viral.genes = intersect(toupper(viral.receptor.gene.name),
as.character(unlist(gene.symbol.protein.names.conversion[1])))

# Match to diseases using the Paper
intersect(as.character(unlist(gene.symbol.protein.names.conversion[1])),toupp
er(as.character(unlist(paper.viral.receptors[7]))[-1]))

## [1] "AXL"     "CEACAM1" "EGFR"    "ICAM1"   "SCARB2"  "TYRO3"   "ITGB3"

# Note that TNFRSF14 does not appear in the paper (but it is a hepatitis
entry mediator)


#  Organize viral receptor proteins by the family they fall into
colnames(paper.viral.receptors) =
as.character(unlist(paper.viral.receptors[1,]))
paper.viral.receptors = paper.viral.receptors[-1,]
selected.viral.genes.info = paper.viral.receptors[paper.viral.receptors$Gene
%in% c(selected.viral.genes, "Ceacam1"),]

# Note that CEACAM1 is not matched to any human diseases in the paper, rather
it is mapped to mouse hepatitis.
selected.viral.genes = selected.viral.genes[-8] #  remove CEACAM1 from the
list


# Get protein names so that we can extract curves from the loess data frame
selected.viral.proteins = selected.viral.genes # vector to store protein
names
for (i in 1:length(selected.viral.genes)){
  index = which(table.for.protein.conversion$Gene.Symbol ==
selected.viral.genes[i])
  selected.viral.proteins[i] = table.for.protein.conversion[index,
"RayBiotech"]
}


# Integrin needs to be specified as B3 subunit
selected.viral.proteins[5] = "Integrin-B3"
```

```
# Order selected.viral.proteins so that the proteins in the same family are
close to one another.
selected.viral.proteins = selected.viral.proteins[c(2,3,5,7,1,6,4)]

# Extract LOESS values for these proteins
viral.loess = cbind(loess.95.proteins$x.values,
loess.95.proteins[selected.viral.proteins])
colnames(viral.loess)[1] = "Years"
melted.viral.loess = melt(viral.loess, id = "Years")

viral.plot.faceted = ggplot(melted.viral.loess, aes(x = Years, y = value)) +
  facet_wrap(~variable, nrow = 7, scales = "free_y") +
  geom_line(size = 2) +
  scale_x_continuous(trans = "log2", breaks = c(0.3, 1, 5, 10, 25, 50, 80)) +
  scale_y_continuous(breaks = seq(round(min(melted.viral.loess$value) - 0.1,
1),
                                  round(max(melted.viral.loess$value) + 0.1,
1),
                                  by = 0.5)) +
  labs(title = "Viral Protein", x = "Age (Years)", y = "Protein Level (Z-
Score)") +
  labs(NULL) +
  theme_bw() +
  theme(panel.background = element_rect(fill = 'gray95'),
        title = element_text(size =25, face = "bold"),
        strip.text.x = element_text(size = 20, face = "bold"),
        axis.text.x = element_text(size = 15, angle = 90, vjust=0.5),
        axis.text.y = element_text(size = 15),
        axis.title.y = element_text(size = 20),
        axis.title.x =element_text(size = 20),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        plot.title = element_text(hjust = 0.5, size = 20))
#  ggsave("Viral Receptors Entry.png", height = 20, width = 5)
```

Plot all viral proteins on one graph to compare their trajectories more easily. This is FIGURE 30 in my thesis.

```
viral.loess.plot =
  ggplot(melted.viral.loess, aes(x = Years, y = value, group = variable,
color = variable)) +
  geom_line() +
  scale_x_continuous(trans = "log2", breaks = c(0.3, 1, 5, 10, 25, 50, 80)) +
  scale_y_continuous(breaks = seq(round(min(melted.viral.loess$value) - 0.1,
1),
                                  round(max(melted.viral.loess$value) + 0.1,
```

```
1),
                                  by = 0.5)) +
  labs(title = "Viral Receptor Protein Trajectories", x = "Age (Years)",
       y = "Protein Level (Z-Score)", color = "Protein") +
  labs(NULL) +
  theme_bw() +
  theme(panel.background = element_rect(fill = 'gray95'),
        axis.text.x = element_text(size = 10, angle = 90, vjust=0.5),
        axis.title.y = element_text(size = 15),
        axis.title.x =element_text(size = 15),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        plot.title = element_text(hjust = 0.5, size = 20))
viral.loess.plot
```




```
# ggsave("Viral Receptor Lifespan Trajectories.png", device = "png", height =
6, width = 8)
```

Age-based clustering using protein expression values

Our data show that many proteins in the immune and synaptic categories have similar expression patterns at different points in their lifespan. Specifically, trajectories with similar protein expression levels in young childhood and in aging. This led us to wonder if protein expression can even be used to differentiate age of samples.

To test this, we performed RSKC clustering on the samples, using protein expression as the input. Neuroimmune and plasticity data were separately considered. Then they were combined.

Determine optimal number of clusters using RSKC

"Clest" is the function used to determine number of clusters for RSKC. We use this algorithm here on the data sets separately and pick a value that is suitable for both.

```
# Plasticity data is suited for 6 clusters
set.seed(120)
Clest(imputed.plasticity.exp, 10, 0.1, B = 15, B0 = 5, nstart = 1000,
      L1 = 5, beta = 0.3, pca = TRUE, silent=FALSE)

## RSKC will be performed (maxK-1)*(B0*1*(1+1)+B*(1+1))= 360 times
##
## Assessing a reference data 1 out of 5
##  Assessing a random partition 1 out of 1
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##
## Assessing a reference data 2 out of 5
##  Assessing a random partition 1 out of 1
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##
## Assessing a reference data 3 out of 5
##  Assessing a random partition 1 out of 1
```

```
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##
## Assessing a reference data 4 out of 5
##  Assessing a random partition 1 out of 1
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##
## Assessing a reference data 5 out of 5
##  Assessing a random partition 1 out of 1
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##
## Assessing the observed data
##  Assessing a random partition 1 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 2 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
```

```
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 3 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 4 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 5 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 6 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 7 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
```

```
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 8 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 9 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 10 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 11 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 12 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
```

```
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 13 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 14 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 15 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10

## $call
## Clest(d = imputed.plasticity.exp, maxK = 10, alpha = 0.1, B = 15,
##     B0 = 5, nstart = 1000, L1 = 5, beta = 0.3, pca = TRUE, silent = FALSE)
##
## $K
## [1] 6
##
## $result.table
##       test.stat    obsCER    refCER P-value
## 2   0.048888889 0.3555556 0.3066667     0.4
## 3   0.017777778 0.3555556 0.3377778     0.6
## 4   0.088888889 0.3555556 0.2666667     0.6
## 5  -0.057777778 0.2222222 0.2800000     0.0
## 6  -0.066666667 0.1555556 0.2222222     0.0
```

```
## 7   0.031111111 0.2000000 0.1688889     0.6
## 8  -0.017777778 0.1555556 0.1733333     0.2
## 9  -0.008888889 0.1333333 0.1422222     0.4
## 10  0.053333333 0.1555556 0.1022222     1.0
##
## $referenceCERs
##            k=2         3         4         5         6          7
8
## [1,] 0.5333333 0.3111111 0.1555556 0.3111111 0.2444444 0.06666667
0.02222222
## [2,] 0.0000000 0.3111111 0.1333333 0.2666667 0.2000000 0.26666667
0.22222222
## [3,] 0.5333333 0.3555556 0.3555556 0.2666667 0.2444444 0.20000000
0.24444444
## [4,] 0.4666667 0.5333333 0.4222222 0.2444444 0.1555556 0.15555556
0.20000000
## [5,] 0.0000000 0.1777778 0.2666667 0.3111111 0.2666667 0.15555556
0.17777778
##               9         10
## [1,] 0.08888889 0.06666667
## [2,] 0.13333333 0.11111111
## [3,] 0.20000000 0.13333333
## [4,] 0.17777778 0.11111111
## [5,] 0.11111111 0.08888889
##
## $observedCERs
##             k=2         3          4          5          6         7
8
##  [1,] 0.0000000 0.5333333 0.44444444 0.15555556 0.08888889 0.2000000
0.11111111
##  [2,] 0.3555556 0.3333333 0.40000000 0.28888889 0.33333333 0.3777778
0.20000000
##  [3,] 0.0000000 0.3777778 0.24444444 0.15555556 0.08888889 0.1111111
0.08888889
##  [4,] 0.0000000 0.0000000 0.35555556 0.02222222 0.06666667 0.1333333
0.17777778
##  [5,] 0.4666667 0.4000000 0.31111111 0.20000000 0.15555556 0.2000000
0.15555556
##  [6,] 0.5333333 0.4000000 0.42222222 0.33333333 0.28888889 0.2000000
0.13333333
##  [7,] 0.5333333 0.2666667 0.37777778 0.28888889 0.20000000 0.1777778
0.13333333
##  [8,] 0.3555556 0.3111111 0.40000000 0.17777778 0.20000000 0.2000000
0.20000000
##  [9,] 0.3555556 0.2000000 0.22222222 0.20000000 0.24444444 0.3111111
0.24444444
## [10,] 0.3555556 0.2222222 0.31111111 0.22222222 0.15555556 0.1111111
0.13333333
## [11,] 0.5333333 0.6444444 0.08888889 0.31111111 0.15555556 0.2444444
0.15555556
```

```
## [12,] 0.4666667 0.3555556 0.26666667 0.20000000 0.15555556 0.1111111
0.08888889
## [13,] 0.4666667 0.4000000 0.37777778 0.37777778 0.04444444 0.2222222
0.17777778
## [14,] 0.5333333 0.3777778 0.40000000 0.40000000 0.24444444 0.1555556
0.15555556
## [15,] 0.2000000 0.3111111 0.31111111 0.31111111 0.22222222 0.1333333
0.13333333
##                9         10
##  [1,] 0.08888889 0.15555556
##  [2,] 0.20000000 0.33333333
##  [3,] 0.13333333 0.15555556
##  [4,] 0.13333333 0.17777778
##  [5,] 0.13333333 0.15555556
##  [6,] 0.11111111 0.17777778
##  [7,] 0.11111111 0.04444444
##  [8,] 0.06666667 0.11111111
##  [9,] 0.26666667 0.24444444
## [10,] 0.15555556 0.08888889
## [11,] 0.13333333 0.15555556
## [12,] 0.08888889 0.13333333
## [13,] 0.20000000 0.17777778
## [14,] 0.13333333 0.11111111
## [15,] 0.04444444 0.06666667

# Immune is suited for 4+ clusters, we can use clusters 6 to be able to
compare the results directly with known plasticity markers.
set.seed(120)
Clest(log2.exp.72.proteins, 10, 0.1, B = 15, B0 = 5, nstart = 1000,
      L1 = 9, beta = 0.05, pca = TRUE, silent=FALSE)

## RSKC will be performed (maxK-1)*(B0*1*(1+1)+B*(1+1))= 360 times
##
## Assessing a reference data 1 out of 5
##  Assessing a random partition 1 out of 1
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##
## Assessing a reference data 2 out of 5
##  Assessing a random partition 1 out of 1
##  K= 2
##  K= 3
##  K= 4
```

```
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##
## Assessing a reference data 3 out of 5
##  Assessing a random partition 1 out of 1
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##
## Assessing a reference data 4 out of 5
##  Assessing a random partition 1 out of 1
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##
## Assessing a reference data 5 out of 5
##  Assessing a random partition 1 out of 1
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##
## Assessing the observed data
##  Assessing a random partition 1 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
```

```
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 2 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 3 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 4 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 5 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 6 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
```

```
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 7 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 8 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 9 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 10 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 11 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
```

```
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 12 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 13 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 14 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10
##  Assessing a random partition 15 out of 15
##  K= 2
##  K= 3
##  K= 4
##  K= 5
##  K= 6
##  K= 7
##  K= 8
##  K= 9
##  K= 10

## $call
## Clest(d = log2.exp.72.proteins, maxK = 10, alpha = 0.1, B = 15,
##     B0 = 5, nstart = 1000, L1 = 9, beta = 0.05, pca = TRUE, silent =
FALSE)
##
```

```
## $K
## [1] 4
##
## $result.table
##       test.stat    obsCER    refCER P-value
## 2   0.284444444 0.5333333 0.2488889     0.8
## 3  -0.146666667 0.2666667 0.4133333     0.2
## 4  -0.111111111 0.1777778 0.2888889     0.0
## 5  -0.044444444 0.2222222 0.2666667     0.2
## 6   0.008888889 0.1777778 0.1688889     0.6
## 7  -0.026666667 0.1777778 0.2044444     0.4
## 8  -0.040000000 0.1333333 0.1733333     0.2
## 9  -0.035555556 0.1333333 0.1688889     0.6
## 10  0.062222222 0.1777778 0.1155556     0.8
##
## $referenceCERs
##            k=2         3         4         5         6         7
8
## [1,] 0.0000000 0.3555556 0.3333333 0.2666667 0.3333333 0.2000000
0.13333333
## [2,] 0.3555556 0.4666667 0.4444444 0.4000000 0.1333333 0.3111111
0.24444444
## [3,] 0.0000000 0.4222222 0.2222222 0.2222222 0.0000000 0.1111111
0.06666667
## [4,] 0.3555556 0.6000000 0.1777778 0.2666667 0.1333333 0.1111111
0.15555556
## [5,] 0.5333333 0.2222222 0.2666667 0.1777778 0.2444444 0.2888889
0.26666667
##               9         10
## [1,] 0.08888889 0.08888889
## [2,] 0.26666667 0.22222222
## [3,] 0.08888889 0.11111111
## [4,] 0.11111111 0.04444444
## [5,] 0.28888889 0.11111111
##
## $observedCERs
##             k=2         3          4          5          6          7
##  [1,] 0.5555556 0.0000000 0.08888889 0.06666667 0.04444444 0.11111111
##  [2,] 0.5555556 0.2888889 0.20000000 0.28888889 0.17777778 0.11111111
##  [3,] 0.0000000 0.3333333 0.15555556 0.22222222 0.22222222 0.20000000
##  [4,] 0.5333333 0.2444444 0.06666667 0.13333333 0.17777778 0.06666667
##  [5,] 0.4666667 0.6222222 0.08888889 0.11111111 0.13333333 0.17777778
##  [6,] 0.5555556 0.1555556 0.24444444 0.26666667 0.22222222 0.20000000
##  [7,] 0.5333333 0.2222222 0.17777778 0.22222222 0.17777778 0.17777778
##  [8,] 0.0000000 0.2888889 0.17777778 0.15555556 0.28888889 0.28888889
##  [9,] 0.3555556 0.5333333 0.31111111 0.31111111 0.22222222 0.26666667
## [10,] 0.5555556 0.1111111 0.17777778 0.15555556 0.08888889 0.08888889
## [11,] 0.5333333 0.2666667 0.13333333 0.15555556 0.08888889 0.11111111
## [12,] 0.4666667 0.1111111 0.24444444 0.31111111 0.24444444 0.24444444
## [13,] 0.3555556 0.6000000 0.35555556 0.24444444 0.11111111 0.11111111
```

```
## [14,] 0.0000000 0.4000000 0.00000000 0.24444444 0.11111111 0.11111111
## [15,] 0.5555556 0.2666667 0.31111111 0.33333333 0.20000000 0.22222222
##                8          9         10
##  [1,] 0.06666667 0.13333333 0.06666667
##  [2,] 0.17777778 0.13333333 0.24444444
##  [3,] 0.22222222 0.20000000 0.22222222
##  [4,] 0.08888889 0.08888889 0.08888889
##  [5,] 0.13333333 0.15555556 0.22222222
##  [6,] 0.24444444 0.17777778 0.20000000
##  [7,] 0.13333333 0.13333333 0.11111111
##  [8,] 0.17777778 0.15555556 0.17777778
##  [9,] 0.26666667 0.24444444 0.26666667
## [10,] 0.11111111 0.13333333 0.11111111
## [11,] 0.13333333 0.11111111 0.15555556
## [12,] 0.15555556 0.15555556 0.17777778
## [13,] 0.13333333 0.08888889 0.17777778
## [14,] 0.11111111 0.11111111 0.15555556
## [15,] 0.31111111 0.24444444 0.17777778
```

Function for plotting whiskers on a log scale

Inability to plot accurate whiskers on a log scale is a common problem. The solution was found at stackoverflow: https://stackoverflow.com/questions/38753628/ggplot-boxplot-length-of-whiskers-with-logarithmic-axis

```
# Function to use boxplot.stats to set the box-and-whisker locations
mybxp = function(x) {
  bxp = log2(boxplot.stats(2^x)[["stats"]])
  names(bxp) = c("ymin","lower", "middle","upper","ymax")
  return(bxp)
}

# Function to use boxplot.stats for the outliers
myout = function(x) {
  data.frame(y=log2(boxplot.stats(2^x)[["out"]]))
}
```

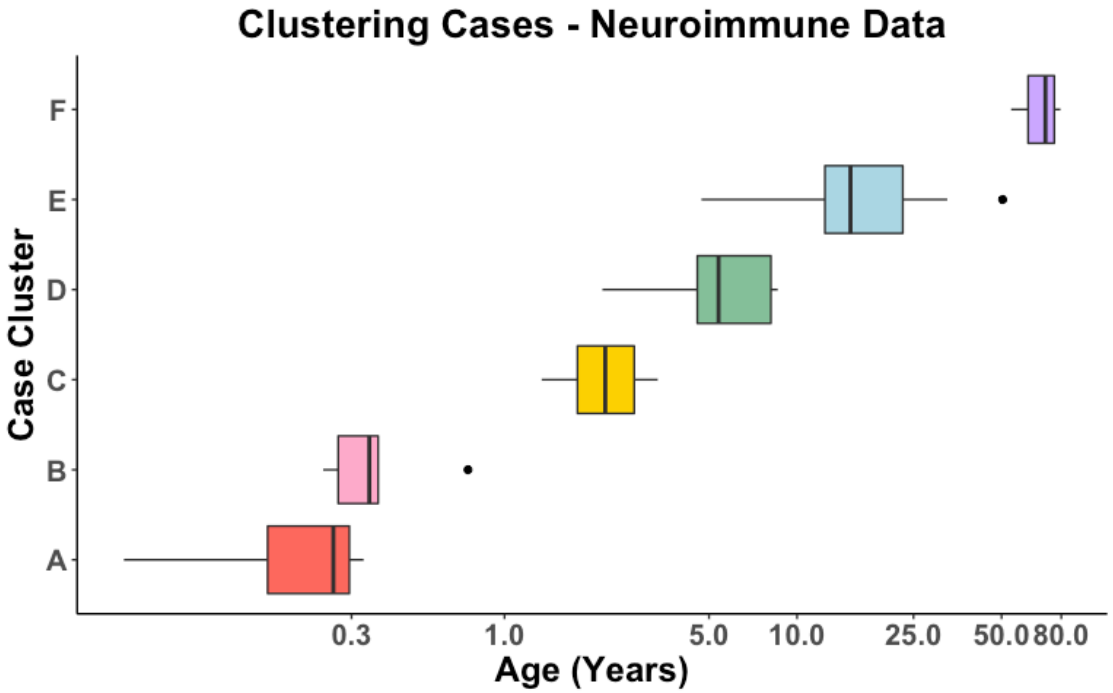
RSKC Immune

This is part A of FIGURE 26 in my thesis.

```
# Immune Data

colour.palette.rskc =  c("#FF6961", "#FFAACC","#FFD300", "#86C29C",
"#ADD8E6","#CCAAFF")
set.seed(120)
rskc.output.immune = RSKC(
  log2.exp.72.proteins,
```

```
  6,
  alpha = 0.1,
  L1 = 9,
  nstart = 1000,
  silent = FALSE,
  scaling = TRUE,
  correlation = FALSE
)

## step a: old wwss 79.2004  new wwss 78.31779
## step b: wbss 98.6 wbss dif -0.16

clust.identity = rskc.output.immune$labels # export cluster labels for each
case
full.data.immune = cbind(identifiers$Years, clust.identity,
log2.exp.72.proteins) # bind the cases, cluster identification of each case
and the exp for all proteins for each case
full.data.immune  = as.data.frame(full.data.immune)
colnames(full.data.immune)[1] = "Age"
full.data.immune$clust.identity = as.factor(full.data.immune$clust.identity)
# make a factor so that you can group by the different clusters when plotting


apply.colors.to.cases =
colour.palette.rskc[as.numeric(full.data.immune$clust.identity)]
original.case.clust = full.data.immune$clust.identity #original order of
clusters
cluster.ages = tapply(full.data.immune$Age,
                      full.data.immune$clust.identity,
                      median,
                      na.rm = TRUE) #calculate the median of each age group

#reorder clusters in order of increasing median age, and append colours in
original order
clust.colors = rbind(cluster.ages, colour.palette.rskc)[, order(cluster.ages,
colour.palette.rskc)]
numeric.cluster.order = order(cluster.ages) #determine the order of the
clusters based on median age
original.case.clust = factor(original.case.clust,
levels(original.case.clust)[numeric.cluster.order]) #reorder the factor
levels
clust.colors.cols = clust.colors[2, ] #prints new order of colors
original.case.clust.immune = original.case.clust

immune.cluster.ages = cluster.ages


# Plot Clusters Horizontally
full.data.immune$clust.identity = factor(full.data.immune$clust.identity,
levels =  c("5", "4", "3", "1", "6", "2"))
```

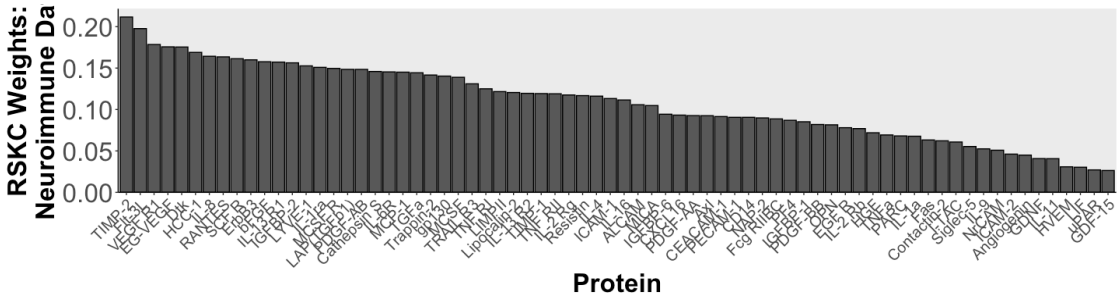
```
ggplot(full.data.immune, aes(Age, clust.identity, fill = clust.identity)) +
  stat_summary(fun.data=mybxp, geom="boxplot", width = 0.75) +
  stat_summary(fun.data=myout, geom="point") +
  scale_fill_manual(values =  colour.palette.rskc) +
  scale_x_continuous(trans = "log2", breaks = c(0.3, 1, 5, 10, 25, 50, 80)) +
  labs(x = "Age (Years)", y = "Case Cluster", title = "Clustering Cases -
Neuroimmune Data") +
  scale_y_discrete(labels=c("5" = "A", "4" = "B", "3" = "C", "1" = "D", "6" =
"E", "2" = "F")) +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none",
        plot.title = element_text(face = "bold", size = 20, hjust = 0.5),
        axis.title.x = element_text(face = "bold", size = 18),
        axis.title.y = element_text(face = "bold", size = 18),
        axis.text.x = element_text(face = "bold", size=15),
        axis.text.y = element_text(face = "bold", size=15))
```




```
# ggsave("Neuroimmune RSKC Age Progression Plot.png", device = "png", height
= 5, width = 8)
```

RSKC Immune Weights: This is part B of FIGURE 26 in my thesis.

```
rskc.weights = as.data.frame(rskc.output.immune$weights) # export weights
from rskc function
rskc.weights$Protein = rownames(rskc.weights)
colnames(rskc.weights)[1] = "Weights"

rskc.weights.immune = rskc.weights # save for future use

# Order the weights, in decreasing order
rskc.weights = rskc.weights[order(rskc.weights$Weights, decreasing = TRUE),]
rskc.weights$Protein = factor(rskc.weights$Protein, levels =
rskc.weights$Protein)

top.rskc.immune = rskc.weights[1:10,]

# Plot the weights as a histogram (Averaged of All Iterations)
ggplot(data = rskc.weights, aes(Protein, Weights)) +
       geom_col(colour = "black") +
       scale_y_continuous(
          expand = c(0, 0),
          name = paste("RSKC Weights:\nNeuroimmune Data"),
          lim = c(0, max(rskc.weights$Weights) + 0.01)) +
      theme(
          axis.line.y = element_line(),
          axis.line.x = element_line(),
          panel.grid = element_blank(),
          axis.text.x = element_text(angle = 45, hjust = 1, size = 15),
          axis.text.y = element_text(angle = 0, vjust = 0.5, size = 25),
          axis.title.x = element_text(size = 25, face = "bold"),
          axis.title.y = element_text(size = 25, face = "bold"),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank()
        )
```



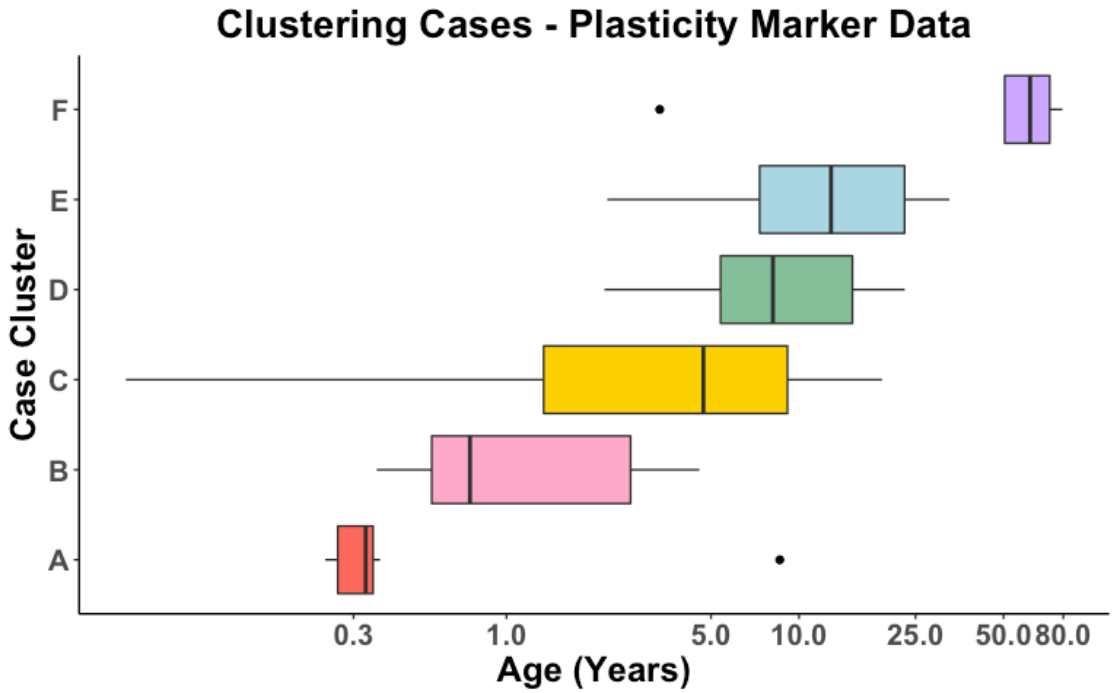

```
# ggsave("Immune RSKC Weights.png", height = 7, width = 20)
```

Plasticity Data

This is part A of FIGURE 28 in my thesis.

```
# Plasticity Data

colour.palette.rskc =  c("#FF6961", "#FFAACC","#FFD300", "#86C29C",
"#ADD8E6","#CCAAFF")
set.seed(120)
rskc.output.plasticity = RSKC(imputed.plasticity.exp, 6, alpha = 0.1, L1 = 5,
nstart = 1000, silent = FALSE, scaling = TRUE, correlation = FALSE)

## step a: old wwss 45.93452  new wwss 46.43965
## step b: wbss 59.76 wbss dif 0

clust.identity = rskc.output.plasticity$labels # export cluster labels for
each case
full.data = cbind(identifiers$Years, clust.identity, imputed.plasticity.exp)
# bind the cases, cluster identification of each case and the exp for all
proteins for each case
full.data  = as.data.frame(full.data)
colnames(full.data)[1] = "Age"
full.data$clust.identity = as.factor(full.data$clust.identity) # make a
factor so that you can group by the different clusters when plotting
levels(full.data$clust.identity) #view levels of cluster (1-X)

## [1] "1" "2" "3" "4" "5" "6"

apply.colors.to.cases =
colour.palette.rskc[as.numeric(full.data$clust.identity)]
original.case.clust = full.data$clust.identity #original order of clusters
cluster.ages = tapply(full.data$Age, full.data$clust.identity,
median,na.rm=TRUE) #calculate the mean of each age group
clust.colors =
rbind(cluster.ages,colour.palette.rskc)[,order(cluster.ages,colour.palette.rs
kc)] #reorder clusters in order of increasing median age, and append colours
in original order
numeric.cluster.order = order(cluster.ages) #determine the order of the
clusters based on median age
original.case.clust =
factor(original.case.clust,levels(original.case.clust)[numeric.cluster.order]
) #reorder the factor levels
clust.colors.cols = clust.colors[2,] #prints new order of colors
original.case.clust.plasticity = original.case.clust

plasticity.cluster.ages = cluster.ages

# Plot RSKC clusters horizontally
full.data$clust.identity = factor(full.data$clust.identity, levels =  c("1",
"5", "6", "4", "2", "3"))
```

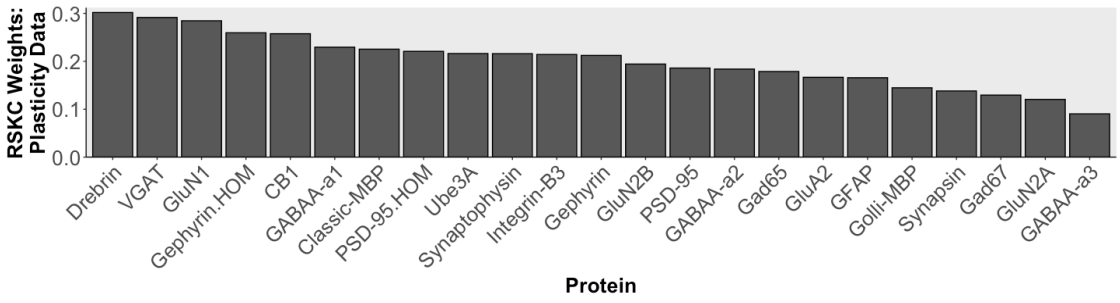
```
ggplot(full.data, aes(Age, clust.identity, fill = clust.identity)) +
  stat_summary(fun.data=mybxp, geom="boxplot", width = 0.75) +
  stat_summary(fun.data=myout, geom="point") +
  scale_fill_manual(values =  colour.palette.rskc) +
  scale_x_continuous(trans = "log2", breaks = c(0.3, 1, 5, 10, 25, 50, 80)) +
  labs(x = "Age (Years)", y = "Case Cluster", title = "Clustering Cases -
Plasticity Marker Data") +
  scale_y_discrete(labels=c("1" = "A", "5" = "B", "6" = "C", "4" = "D", "2" =
"E", "3" = "F")) +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none",
        plot.title = element_text(face = "bold", size = 20, hjust = 0.5),
        axis.title.x = element_text(face = "bold", size = 18),
        axis.title.y = element_text(face = "bold", size = 18),
        axis.text.x = element_text(face = "bold", size=15),
        axis.text.y = element_text(face = "bold", size=15))
```




```
# ggsave("Plasticity RSKC Age Progression Plot.png", device = "png", height =
5, width = 8)
```

RSKC Plasticity Weights: This is part B of FIGURE 28 in my thesis.

```
rskc.weights = as.data.frame(rskc.output.plasticity$weights) # export weights
from rskc function
rskc.weights$Protein = rownames(rskc.weights)
colnames(rskc.weights)[1] = "Weights"

rskc.weights.plasticity = rskc.weights

# Order the weights, in decreasing order
rskc.weights = rskc.weights[order(rskc.weights$Weights, decreasing = TRUE),]
rskc.weights$Protein = factor(rskc.weights$Protein, levels =
rskc.weights$Protein)


# Plot the weights as a histogram (Averaged of All Iterations)
ggplot(data = rskc.weights, aes(Protein, Weights)) +
       geom_col(colour = "black") +
       scale_y_continuous(
          expand = c(0, 0),
          name = paste("RSKC Weights:\nPlasticity Data"),
          lim = c(0, max(rskc.weights$Weights) + 0.01)) +
      theme(
          axis.line.y = element_line(),
          axis.line.x = element_line(),
          panel.grid = element_blank(),
          axis.text.x = element_text(size = 20, angle = 45, hjust = 1),
          axis.text.y = element_text(size = 20, angle = 0, vjust = 0.5),
          axis.title.x = element_text(size = 20, face = "bold"),
          axis.title.y = element_text(size = 20, face = "bold"),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank()
        )
```

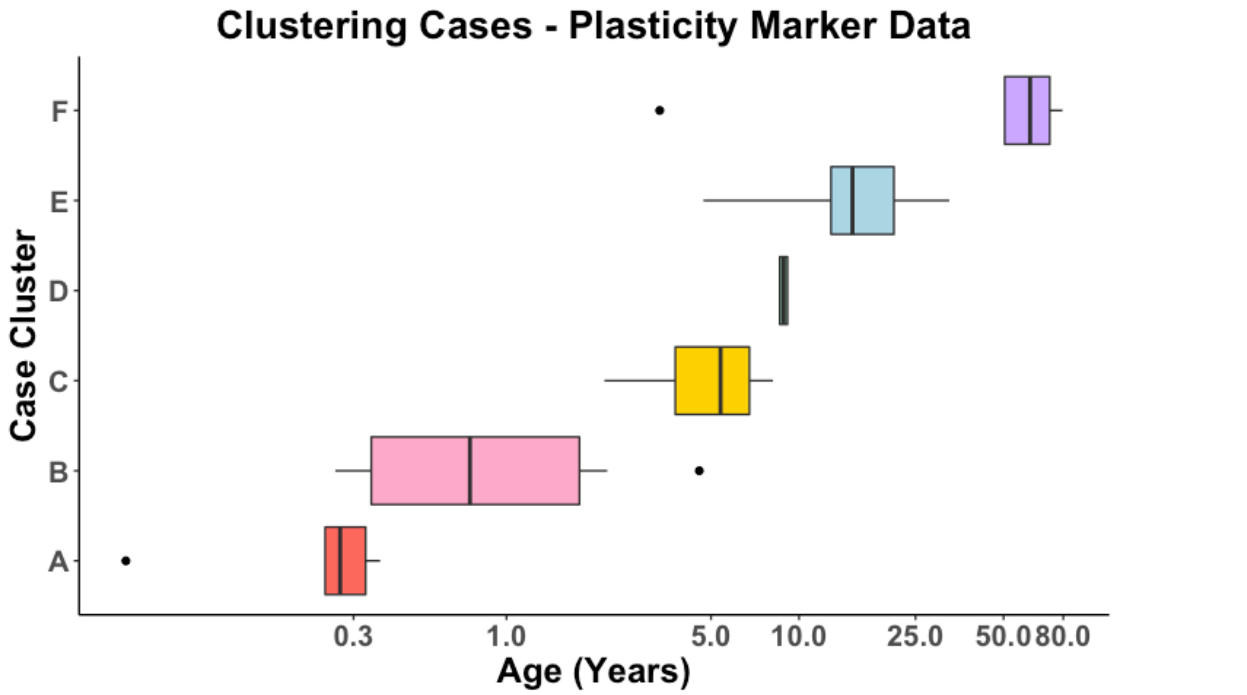



```
# ggsave("Plasticity RSKC Weights.png", height = 7, width = 20)
```

Combined Data

```
# Combined Data


combined.exp = cbind(log2.exp.72.proteins, imputed.plasticity.exp)

colour.palette.rskc =  c("#FF6961", "#FFAACC","#FFD300", "#86C29C",
"#ADD8E6","#CCAAFF")
set.seed(120)
rskc.output.combined = RSKC(combined.exp, 6, alpha = 0.1, L1 = 10, nstart =
1000, silent = FALSE, scaling = TRUE, correlation = FALSE)

## step a: old wwss 96.41799  new wwss 97.86437
## step b: wbss 113.54 wbss dif 0

clust.identity = rskc.output.combined$labels # export cluster labels for each
case
full.data.combined = cbind(identifiers$Years, clust.identity, combined.exp) #
bind the cases, cluster identification of each case and the exp for all
proteins for each case
full.data.combined  = as.data.frame(full.data.combined)
colnames(full.data.combined)[1] = "Age"
full.data.combined$clust.identity =
as.factor(full.data.combined$clust.identity) # make a factor so that you can
group by the different clusters when plotting
levels(full.data.combined$clust.identity) #view levels of cluster (1-X)

## [1] "1" "2" "3" "4" "5" "6"

apply.colors.to.cases =
colour.palette.rskc[as.numeric(full.data.combined$clust.identity)]
original.case.clust = full.data.combined$clust.identity #original order of
clusters
cluster.ages = tapply(full.data.combined$Age,
full.data.combined$clust.identity, median,na.rm=TRUE) #calculate the mean of
each age group
clust.colors =
rbind(cluster.ages,colour.palette.rskc)[,order(cluster.ages,colour.palette.rs
kc)] #reorder clusters in order of increasing median age, and append colours
in original order
numeric.cluster.order = order(cluster.ages) #determine the order of the
clusters based on median age
original.case.clust =
factor(original.case.clust,levels(original.case.clust)[numeric.cluster.order]
) #reorder the factor levels
clust.colors.cols = clust.colors[2,] #prints new order of colors
original.case.clust.combined = original.case.clust
```

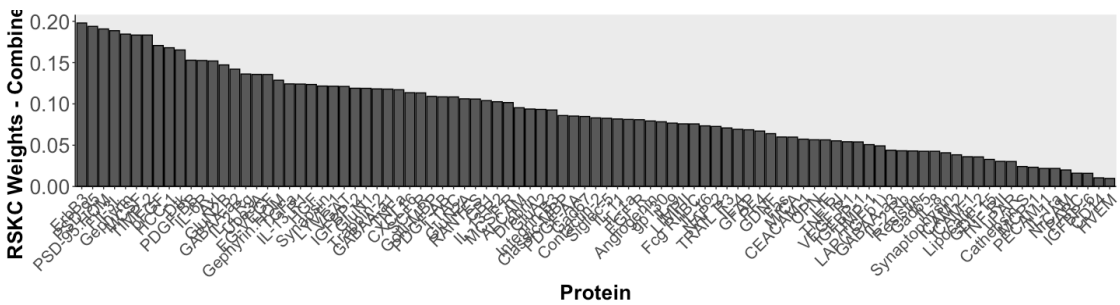
```
combined.cluster.ages = cluster.ages


# Plot RSKC clusters horizontally
full.data.combined$clust.identity = factor(full.data.combined$clust.identity,
levels =  c("2", "6", "3", "5", "1", "4"))

ggplot(full.data.combined, aes(Age, clust.identity, fill = clust.identity)) +
  stat_summary(fun.data=mybxp, geom="boxplot", width = 0.75) +
  stat_summary(fun.data=myout, geom="point") +
  scale_fill_manual(values =  colour.palette.rskc) +
  scale_x_continuous(trans = "log2", breaks = c(0.3, 1, 5, 10, 25, 50, 80)) +
  labs(x = "Age (Years)", y = "Case Cluster", title = "Clustering Cases -
Plasticity Marker Data") +
  scale_y_discrete(labels=c("1" = "E", "5" = "D", "6" = "B", "4" = "F", "2" =
"A", "3" = "C")) +
  theme(panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        panel.background = element_blank(),
        axis.line = element_line(colour = "black"),
        legend.position = "none",
        plot.title = element_text(face = "bold", size = 20, hjust = 0.5),
        axis.title.x = element_text(face = "bold", size = 18),
        axis.title.y = element_text(face = "bold", size = 18),
        axis.text.x = element_text(face = "bold", size=15),
        axis.text.y = element_text(face = "bold", size=15))
```



```
# ggsave("Combined RSKC Age Progression Plot.png", device = "png", height =
5, width = 8)
```

Combined Data Weights

```
rskc.weights = as.data.frame(rskc.output.combined$weights) # export weights
from rskc function
rskc.weights$Protein = rownames(rskc.weights)
colnames(rskc.weights)[1] = "Weights"

rskc.weights.combined = rskc.weights

# Order the weights, in decreasing order
rskc.weights = rskc.weights[order(rskc.weights$Weights, decreasing = TRUE),]
rskc.weights$Protein = factor(rskc.weights$Protein, levels =
rskc.weights$Protein)

# Plot the weights as a histogram (Averaged of All Iterations)
ggplot(data = rskc.weights, aes(Protein, Weights)) +
       geom_col(colour = "black") +
       scale_y_continuous(
          expand = c(0, 0),
          name = paste("RSKC Weights - Combined Data"),
          lim = c(0, max(rskc.weights$Weights) + 0.01)) +
       theme(
          axis.line.y = element_line(),
          axis.line.x = element_line(),
          panel.grid = element_blank(),
          axis.text.x = element_text(size = 16, angle = 45, hjust = 1),
          axis.text.y = element_text(size = 20, angle = 0, vjust = 0.5),
          axis.title.x = element_text(size = 20, face = "bold"),
          axis.title.y = element_text(size = 20, face = "bold"),
          panel.grid.major = element_blank(),
          panel.grid.minor = element_blank()
        )
```



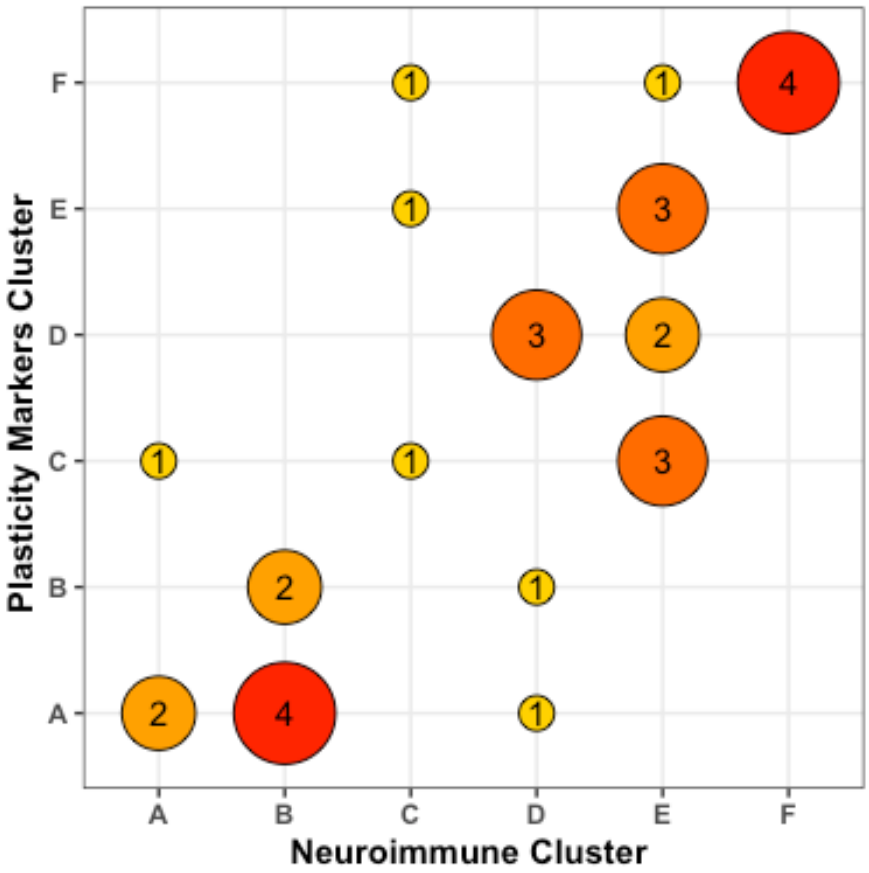

```
# ggsave("Combined RSKC Weights.png", height = 4, width = 15)
```

Evaluate similarity between Age Clusters: Immune Vs Classic Plasticity Markers

This is part A and B of FIGURE 29 in my thesis.

```
# Visualize the Immune Vs. Plasticity Cluster Designations


original.case.clust.immune # informs the median age of numeric immune
clusters

##  [1] 5 4 5 4 4 5 4 4 4 3 1 3 3 1 6 1 1 1 6 6 6 6 6 6 6 2 2 2 2
## Levels: 5 4 3 1 6 2

original.case.clust.plasticity # informs the median age of numeric plasticity
clusters

##  [1] 6 1 1 1 1 1 5 1 5 6 4 2 3 5 6 4 4 1 6 2 2 4 6 4 2 3 3 3 3 3
## Levels: 1 5 6 4 2 3

combined.results = cbind(full.data.immune[1:2], full.data[2]) # bind the two
cluster designations together with ages

# combine the clustering results into one data frame
# order clusters based on median age, A = Youngest, G = Oldest
combined.results = combined.results %>%
set_colnames(c("Age", "immune_cluster", "plasticity_cluster")) %>%
mutate(immune_cluster = case_when(
immune_cluster == 1 ~ "D",
immune_cluster == 2 ~ "F",
immune_cluster == 3 ~ "C",
immune_cluster == 4 ~ "B",
immune_cluster == 5 ~ "A",
immune_cluster == 6 ~ "E"

)) %>%
mutate(plasticity_cluster = case_when(
plasticity_cluster == 1 ~ "A",
plasticity_cluster == 2 ~ "E",
plasticity_cluster == 3 ~ "F",
plasticity_cluster == 4 ~ "D",
plasticity_cluster == 5 ~ "B",
plasticity_cluster == 6 ~ "C"

))

# Tally the number of occurrences for each combination (ex. How many cases
are cluster A in both,
# How many are in A when clustering using immune and then B when using
plasticity data,
```

```
# how many follow the reverse pattern etc.)
plot.results = combined.results %>%
group_by(immune_cluster, plasticity_cluster) %>%
tally()


# Plot with Clusters as discrete bins
clust.bubble = ggplot(plot.results, aes(x = immune_cluster, y =
plasticity_cluster)) +
geom_point(aes(size = n, fill = n), colour = "black", shape = 21) +
geom_text(aes(label = n)) +
coord_fixed() +
scale_fill_gradient2(low = "white", mid = "yellow", high = "red") +
scale_size_continuous(range = c(5,15)) +
theme_bw() +
theme(legend.position = "none",
      axis.title.x = element_text(face = "bold"),
      axis.title.y = element_text(face = "bold"),
      axis.text.x = element_text(face= "bold"),
      axis.text.y = element_text(face= "bold")) +
labs(x = "Neuroimmune Cluster ", y = "Plasticity Markers Cluster ")
clust.bubble
```



```r
# ggsave("RSKC Similarity Bubble Plot.png", height = 6, width = 6)


kable(combined.results, align = "c")
```

Age	immune_cluster	plasticity_cluster
0.05	A	C
0.24	B	A
0.26	A	A
0.27	B	A
0.33	B	A
0.33	A	A
0.36	B	B
0.37	B	A
0.75	B	B
1.34	C	C
2.16	D	D
2.21	C	E
3.34	C	F
4.56	D	B
4.71	E	C
5.39	D	D
8.14	D	D
8.59	D	A
9.13	E	C
12.45	E	E
13.27	E	E
15.22	E	D
19.21	E	C
22.98	E	D
32.61	E	E
50.43	E	F
53.90	F	F
69.30	F	F
71.91	F	F
79.50	F	F

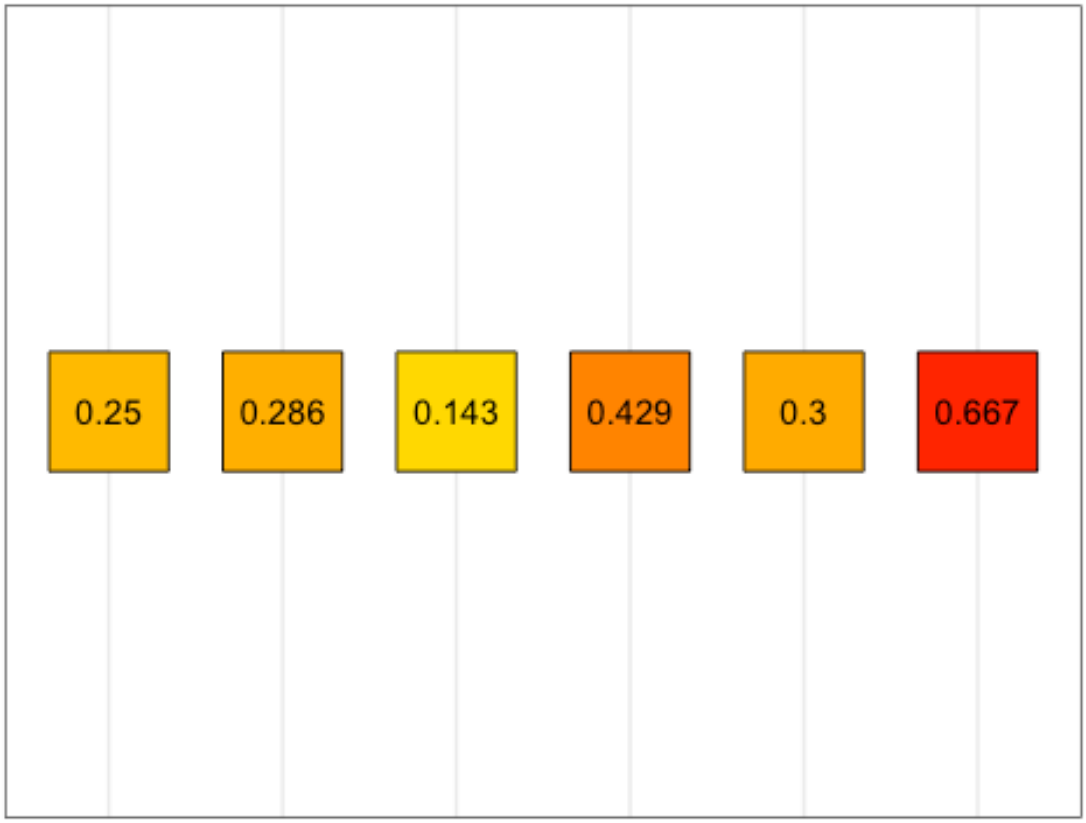
Perform statistical comparisons (Jaccard Similarity Coefficient)

This is part C of FIGURE 29 in my thesis.

```
# Calculate Rand Index between the immune and plasticity clusters designation
overall


immune.cluster.designation = combined.results$immune_cluster # get case
cluster designation using immune
plasticity.cluster.designation = combined.results$plasticity_cluster # get
case cluster designation using plasticity

# Loop through both of the above and convert letters to numbers, as rand
index function only takes numeric input
# Below A = 1, B = 2, C = 3, D = 4, E = 5, F = 6

for (a in 1:30){
if(immune.cluster.designation[a] == LETTERS[1]){immune.cluster.designation[a]
= 1}
if(immune.cluster.designation[a] == LETTERS[2]){immune.cluster.designation[a]
= 2}
if(immune.cluster.designation[a] == LETTERS[3]){immune.cluster.designation[a]
= 3}
if(immune.cluster.designation[a] == LETTERS[4]){immune.cluster.designation[a]
= 4}
if(immune.cluster.designation[a] == LETTERS[5]){immune.cluster.designation[a]
= 5}
if(immune.cluster.designation[a] == LETTERS[6]){immune.cluster.designation[a]
= 6}
}

for (a in 1:30){
if(plasticity.cluster.designation[a] ==
LETTERS[1]){plasticity.cluster.designation[a] = 1}
if(plasticity.cluster.designation[a] ==
LETTERS[2]){plasticity.cluster.designation[a] = 2}
if(plasticity.cluster.designation[a] ==
LETTERS[3]){plasticity.cluster.designation[a] = 3}
if(plasticity.cluster.designation[a] ==
LETTERS[4]){plasticity.cluster.designation[a] = 4}
if(plasticity.cluster.designation[a] ==
LETTERS[5]){plasticity.cluster.designation[a] = 5}
if(plasticity.cluster.designation[a] ==
LETTERS[6]){plasticity.cluster.designation[a] = 6}
}

# The values are stored as a character string, change to numeric
immune.cluster.designation = as.numeric(immune.cluster.designation)
```

```
plasticity.cluster.designation = as.numeric(plasticity.cluster.designation)

# Calculate Rand Index and Jaccard

similarity.measurements =
clusteval::cluster_similarity(immune.cluster.designation,
plasticity.cluster.designation,
  similarity = "jaccard",
  method = "independence")

comembership_table(immune.cluster.designation,
plasticity.cluster.designation)

## $n_11
## [1] 24
##
## $n_10
## [1] 49
##
## $n_01
## [1] 41
##
## $n_00
## [1] 321

# Calculate Jaccard coefficient between individual clusters

jaccard.values = as.data.frame(matrix(ncol = 6, nrow = 1))
colnames(jaccard.values) = paste("Cluster", LETTERS[1:6])

for(cluster.index in 1:6){

evaluate.clusters = combined.results[2:3]
evaluate.clusters$Immune.Present = NA
evaluate.clusters$Plasticity.Present = NA

for (a in 1: nrow(evaluate.clusters)){ # Use binary values to indicate when a
case is found in a cluster (1) and when it is not (0)

  if (evaluate.clusters[a, "immune_cluster"] ==
LETTERS[cluster.index]){evaluate.clusters[a, "Immune.Present"] = 1
  } else(evaluate.clusters[a, "Immune.Present"] = 0)


  if (evaluate.clusters[a, "plasticity_cluster"] ==
LETTERS[cluster.index]){evaluate.clusters[a, "Plasticity.Present"] = 1
  } else(evaluate.clusters[a, "Plasticity.Present"] = 0)

evaluate.clusters$Sums = rowSums(evaluate.clusters[3:4]) # calculate sums,
```

```
in.both = sum(evaluate.clusters$Sums == 2) # if present in both neuroimmune
and plasticity cluster then this will be a value of 2
in.either.or.both = sum(evaluate.clusters$Sums == 2 | evaluate.clusters$Sums
== 1 ) # if present in one or either sum will be 1
jaccard.coefficient = in.both/in.either.or.both

}

jaccard.values[cluster.index] = jaccard.coefficient # store the Jaccard index
for each cluster in a vector

}

jaccard.values.2 = reshape2::melt(jaccard.values) # melt the data to convert
it to the long format and visualize

# Plot using ggplot2
RSKC.jaccard  = ggplot(jaccard.values.2, aes(x = variable, y = 0)) +
  geom_point(aes(fill = value), colour = "black", shape = 22, size = 20) +
  geom_text(aes(label = round(value, 3))) +
  scale_fill_gradient2(low = "white", mid = "yellow", high = "red") +
  scale_x_discrete(position = "bottom") +
  theme_bw() +
  theme(legend.position = "none",
        axis.text = element_blank(),
        axis.ticks = element_blank(),
        panel.grid.major.y = element_blank(),
        panel.grid.minor.y = element_blank()) +
  labs(x = "RSKC Jaccard Index", y = NULL)
# ggsave("RSKC Jaccard per Cluster.png", height = 5, width = 8)

RSKC.jaccard
```



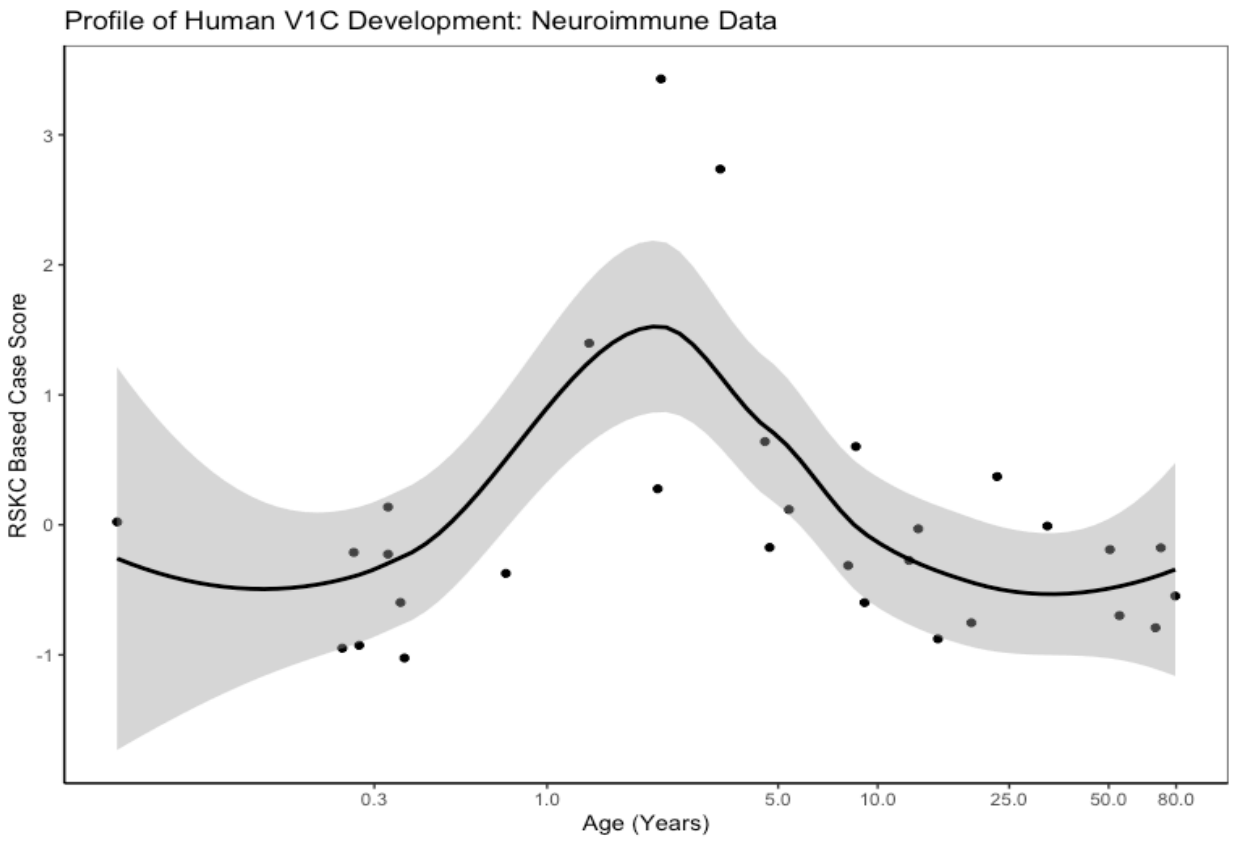
RSKC Jaccard Index

Creating a Profile of Human V1 Development Using RSKC Weights

This FIGURE 27 in my thesis.

72 Neuroimmune Proteins:

```
# Comapring RSKC Weighted Cases

rskc.weighted.72 = as.data.frame(log2.exp.72.proteins)
rskc.weighted.72 = rskc.weighted.72[order(colnames(rskc.weighted.72),
decreasing = FALSE)]

rskc.weights.immune = rskc.weights.immune[order(rskc.weights.immune$Protein,
decreasing = FALSE),]

for (i in 1:ncol(rskc.weighted.72)){
  rskc.weighted.72[,i] = rskc.weighted.72[,i] * rskc.weights.immune[i,
"Weights"]
}

rskc.weighted.72$Sum = rowSums(rskc.weighted.72)
rskc.weighted.72$Zscore = zscore(rskc.weighted.72$Sum)


rskc.weighted.72 = cbind(identifiers, rskc.weighted.72)

ggplot(rskc.weighted.72, aes(x = Years, y = Zscore)) +
  geom_point() +
  labs(title = "Profile of Human V1C Development: Neuroimmune Data",
       x = "Age (Years)",
       y = "RSKC Based Case Score") +
  stat_smooth(method = "loess", color = "black") +
  scale_x_continuous(trans = "log2", breaks = c(0.3, 1, 5, 10, 25, 50, 80)) +
  theme_bw() +
  theme(axis.line.y = element_line(),
        axis.line.x = element_line(),
        panel.grid = element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```

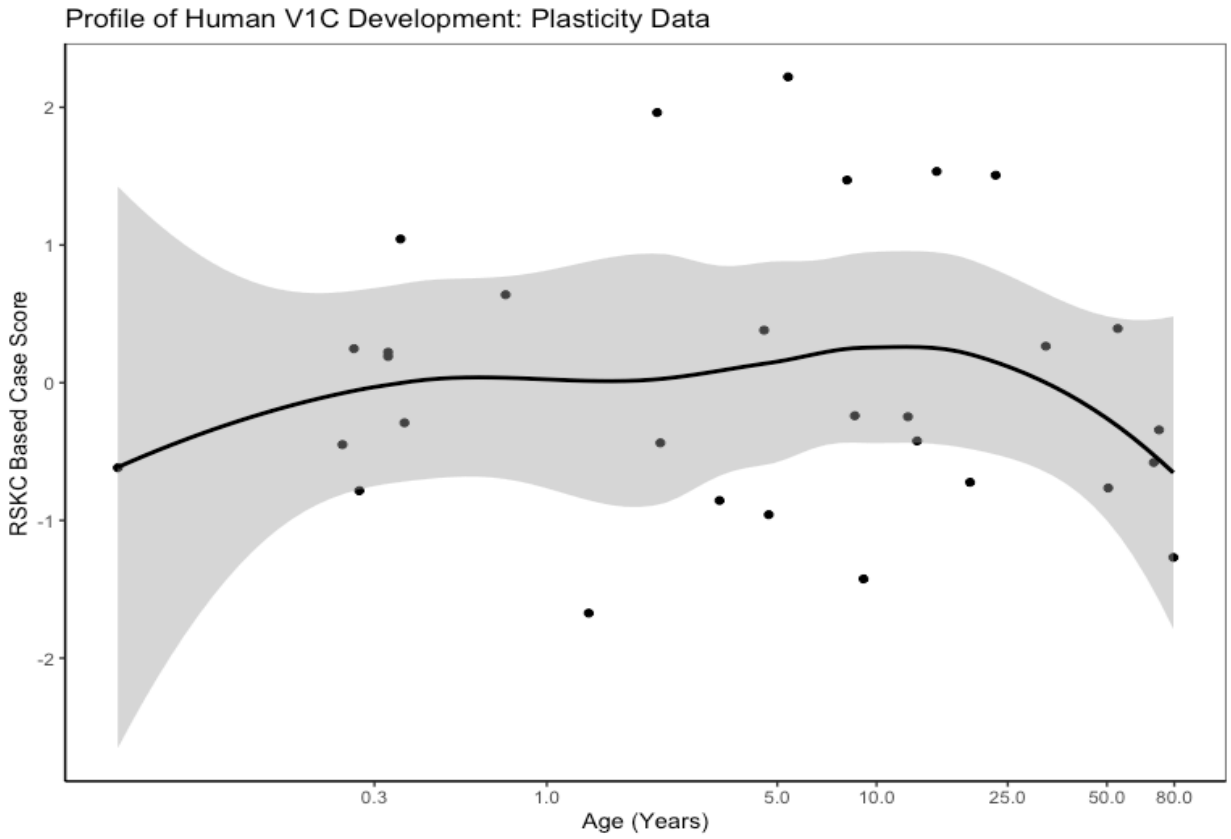



```
# ggsave("Profile of Human V1C Development: Neuroimmune Data.png", device =
"png", height = 6, width = 8)
```

23 Classic Plasticity Markers:

```
# Create a data frame to store the weighted expression values
rskc.weighted.23 = as.data.frame(imputed.plasticity.exp)

# Order the columns alphabetically, so that we can apply the same order to
the weights data frame
rskc.weighted.23 = rskc.weighted.23[order(colnames(rskc.weighted.23),
decreasing = FALSE)]

# Order the proteins in the weights data frame in the same order as the
expression data frame
rskc.weights.plasticity =
rskc.weights.plasticity[order(rskc.weights.plasticity$Protein, decreasing =
FALSE),]

for (i in 1:ncol(rskc.weighted.23)){
  rskc.weighted.23[,i] = rskc.weighted.23[,i] * rskc.weights.plasticity[i,
"Weights"]
}
```

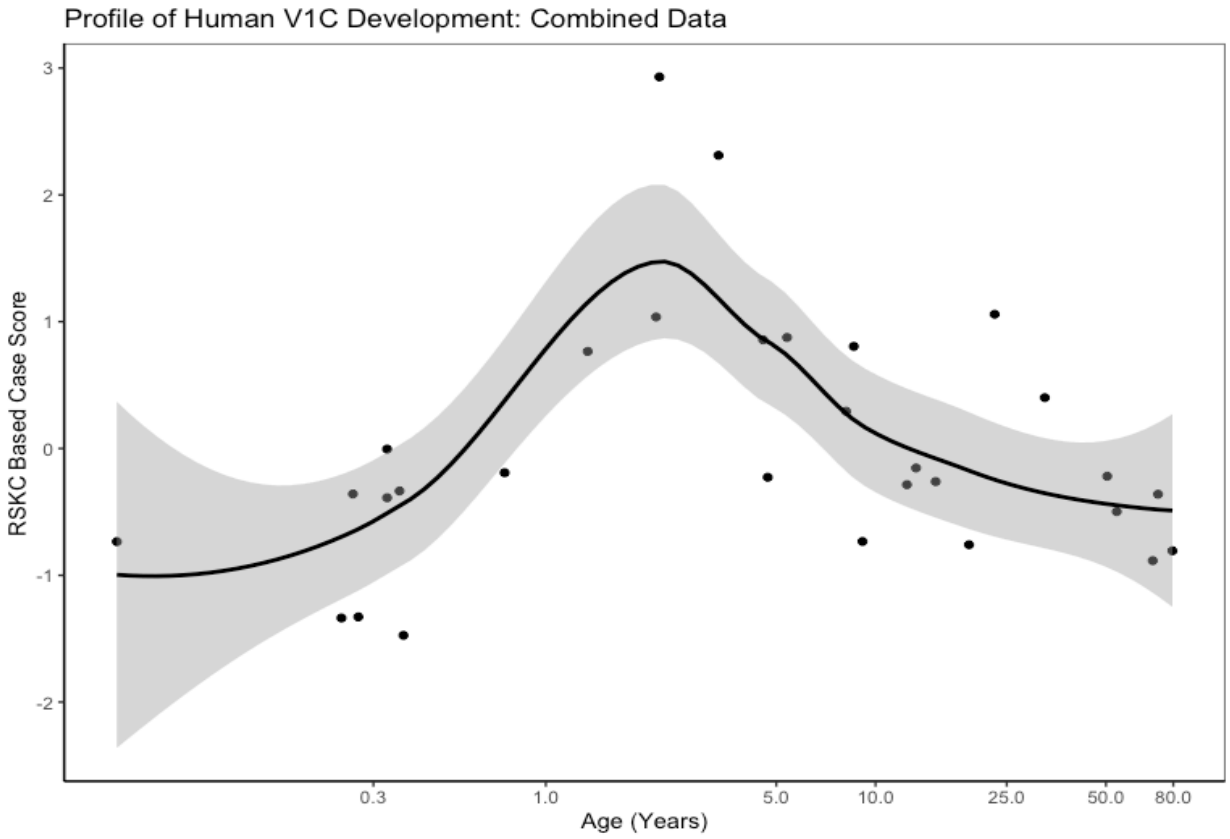
```
rskc.weighted.23$Sum = rowSums(rskc.weighted.23)
rskc.weighted.23$Zscore = zscore(rskc.weighted.23$Sum)


rskc.weighted.23 = cbind(identifiers, rskc.weighted.23)

ggplot(rskc.weighted.23, aes(x = Years, y = Zscore)) +
  geom_point() +
  labs(title = "Profile of Human V1C Development: Plasticity Data",
       x = "Age (Years)",
       y = "RSKC Based Case Score") +
  stat_smooth(method = "loess", color = "black") +
  scale_x_continuous(trans = "log2", breaks = c(0.3, 1, 5, 10, 25, 50, 80)) +
  theme_bw() +
  theme(axis.line.y = element_line(),
        axis.line.x = element_line(),
        panel.grid = element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```




```
# ggsave("Profile of Human V1C Development: Plasticity Data.png", device =
"png", height = 6, width = 8)
```

95 Combined Trajectories:

```
# Create a data frame to store the weighted expression values
rskc.weighted.95 = as.data.frame(combined.exp)

# Order the columns alphabetically, so that we can apply the same order to
the weights data frame
rskc.weighted.95 = rskc.weighted.95[order(colnames(rskc.weighted.95),
decreasing = FALSE)]

# Order the proteins in the weights data frame in the same order as the
expression data frame
rskc.weights.combined =
rskc.weights.combined[order(rskc.weights.combined$Protein, decreasing =
FALSE),]

# Multiply the protein expression by the protein weight
for (i in 1:ncol(rskc.weighted.95)){
  rskc.weighted.95[,i] = rskc.weighted.95[,i] * rskc.weights.combined[i,
"Weights"]
}

rskc.weighted.95$Sum = rowSums(rskc.weighted.95) # calculate the row sum
rskc.weighted.95$Zscore = zscore(rskc.weighted.95$Sum) # calculate zscores on
the row sums

# Bind the identifers to the weighted expression
rskc.weighted.95 = cbind(identifiers, rskc.weighted.95)

# Plot the weights
ggplot(rskc.weighted.95, aes(x = Years, y = Zscore)) +
  geom_point() +
  labs(title = "Profile of Human V1C Development: Combined Data",
       x = "Age (Years)",
       y = "RSKC Based Case Score") +
  stat_smooth(method = "loess", color = "black") +
  scale_x_continuous(trans = "log2", breaks = c(0.3, 1, 5, 10, 25, 50, 80)) +
  theme_bw() +
  theme(axis.line.y = element_line(),
        axis.line.x = element_line(),
        panel.grid = element_blank(),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())
```




```
# ggsave("Profile of Human V1C Development: Combined Data.png", device =
"png", height = 6, width = 8)
```

Get Interquartile Ranges for Immune and Plasticity Age-based Clusters

```
# Turn the cluster designation column into a factor so that we can group
cases in a cluster together and calculate the interquartile range
combined.results$immune_cluster = factor(combined.results$immune_cluster,
levels = LETTERS[1:6]) # immune
combined.results$plasticity_cluster =
factor(combined.results$plasticity_cluster, levels = LETTERS[1:6]) #
plasticity

# Create an empty data frame to store the IQR, assign column names
case.cluster.IQR = as.data.frame(matrix(ncol = 6, nrow = 6))
colnames(case.cluster.IQR) =
  c("Median Immune", "25th Percentile Immune", "75th Percentile Immune",
"Median Plasticity", "25th Percentile Plasticity", "75th Percentile
Plasticity")

# Fill the data frame un using the summarise, median and quantile functions

# Column containing median age for each immune cluster
case.cluster.IQR["Median Immune"] = as.data.frame(ddply(combined.results,
"immune_cluster", summarise, IMMED = median(Age)))["IMMED"]
# Column containing the 25th percentile for each immune cluster
case.cluster.IQR["25th Percentile Immune"] =
as.data.frame(ddply(combined.results, "immune_cluster", summarise, IM25 =
quantile(Age, .25)))["IM25"]
# Column containing the 75th percentile for each immune cluster
case.cluster.IQR["75th Percentile Immune"] =
as.data.frame(ddply(combined.results, "immune_cluster", summarise, IM75 =
quantile(Age, .75)))["IM75"]
# Column containing median age for each plasticity cluster
case.cluster.IQR["Median Plasticity"] = as.data.frame(ddply(combined.results,
"plasticity_cluster", summarise, PLMED = median(Age)))["PLMED"]
# Column containing the 25th percentile for each plasticity cluster
case.cluster.IQR["25th Percentile Plasticity"] =
as.data.frame(ddply(combined.results, "plasticity_cluster", summarise, PL25 =
quantile(Age, .25)))["PL25"]
# Column containing the 75th percentile for each plasticity cluster
case.cluster.IQR["75th Percentile Plasticity"] =
as.data.frame(ddply(combined.results, "plasticity_cluster", summarise, PL75 =
quantile(Age, .75)))["PL75"]
```

Ewalina Jeyanesan's MSc Thesis Markdown- Revisions

Ewalina Jeyanesan

24/09/2020

Note - This file is supplemental to the main thesis markdown.

This markdown is supplemental to the original markdown titled "Ewalina Jeyanesan's MSc Thesis R Markdown" found above. That file was generated in preparing my thesis for the draft submissions to my supervisory committee meeting. The current markdown was created while making the revisions indicted by the committee and contains some code verification steps as well.

Session Info (Ensure that only needed packages are loaded)

```
sessionInfo() # see what base R packages are already attached, R version etc.
```

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
## Matrix products: default
## BLAS:
/Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK:
/Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_CA.UTF-8/en_CA.UTF-8/en_CA.UTF-8/C/en_CA.UTF-8/en_CA.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.29
##
## loaded via a namespace (and not attached):
##  [1] compiler_4.0.2  magrittr_1.5    tools_4.0.2     htmltools_0.5.0
##  [5] yaml_2.2.1      stringi_1.4.6   rmarkdown_2.3   stringr_1.4.0
##  [9] xfun_0.16       digest_0.6.25   rlang_0.4.7     evaluate_0.14
```

Load Packages

Certain packages are not available for direct installation from CRAN, these can be installed using the "devtools" package to access these directly from GitHub.

```
library(tidyverse) # clean, process, and visualize data (includes
dependencies like ggplot2 and dplyr)
library(janitor) # for various data cleaning functions
library(magrittr) # for managing data frames (i.e. set_rownames() and
set_colnames())
library(zoo) # for rollapply() function to process rolling windows
library(data.table) # for melt() function to reshape data
library(devtools) # used to install other packages not directly available
through CRAN
library(fitdistrplus) # for fitting models/ distributions to data
library(mosaic) # for calculating zscores
library(factoextra) # for WSS calculations and plot
library(pvclust) # for calculating p-values for hierarchical clusters
library(kmlShape) # for Frechet distance calculations
library(gplots) # for heatmap.2()
library(rlist) # for list.append() and other functions
library(plotrix) # for statistical measures including std.error and std.dev
library(qpcR) # for cbind() of varying lengths
library(gprofiler2) # gene enrichment tool
library(grid) # for plotting graphs
library(mice) # for data imputation
library(VIM) # for aggr() function
library(RSKC) # for robust and sparse k-means clustering
library(FactoMineR) # for PCA
library(readxl) # for reading and writing excel files
library(openxlsx) # for writing excel files with multiple sheets
library(dabestr) # for estimation statistics
library(Rmisc) # for confidence interval computation
library(clusteval) # for Jaccard Coefficient computation
library(gridExtra) # for grid.arrange function
library(ggpubr) # for ggarrange function
library(reshape2) # for melt function
library(dendextend) # rotating branches of dendrogram
#devtools::install_github("guiastrennec/ggplus")
library(ggplus) # for facet_multiple() function
```

Load Data Files

This contains all the values and analyses performed. Saves time when double checking the analyses.

```
load("/Users/ewalinajeyanesan/Dropbox (Kathy Murphy)/Ewalina/Thesis/R Markdown and Supplement/Markdowns/Sept 2nd Draft Submission/Thesis Submission Global Environment.RData")
```

Frequency Histogram - Normalization Verification

This additional check ensures that the data remains normally distributed once all the data preparation steps have been completed for the neuroimmune data. This is important as ward.D2 and other statistical measures assume that the data is normally distributed.

First complete it for the neuroimmune data alone.

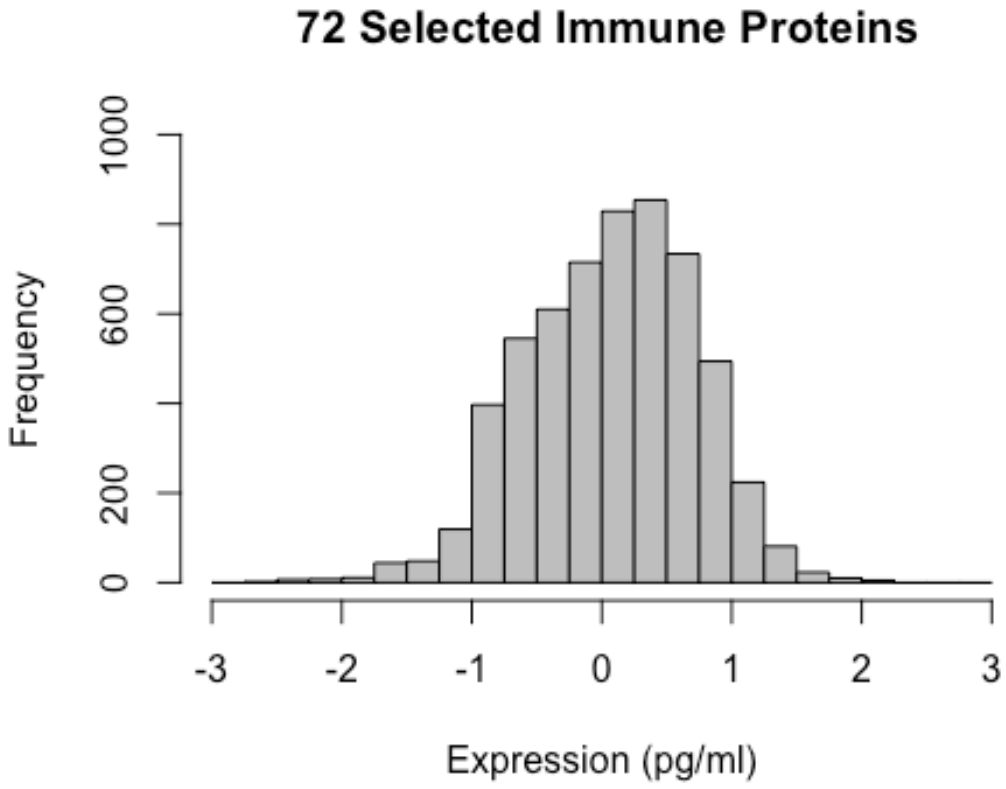
```
loess.72.proteins.check = as.matrix(loess.72.proteins[,-1]) # histogram function needs input to be a matrix
max(as.numeric(unlist(loess.72.proteins.check)), na.rm = TRUE) # use to set max x-axis limits of frequency plot
```

```
## [1] 2.230528
```

```
min(as.numeric(unlist(loess.72.proteins.check)), na.rm = TRUE) # use to set min x-axis limits of frequency plot
```

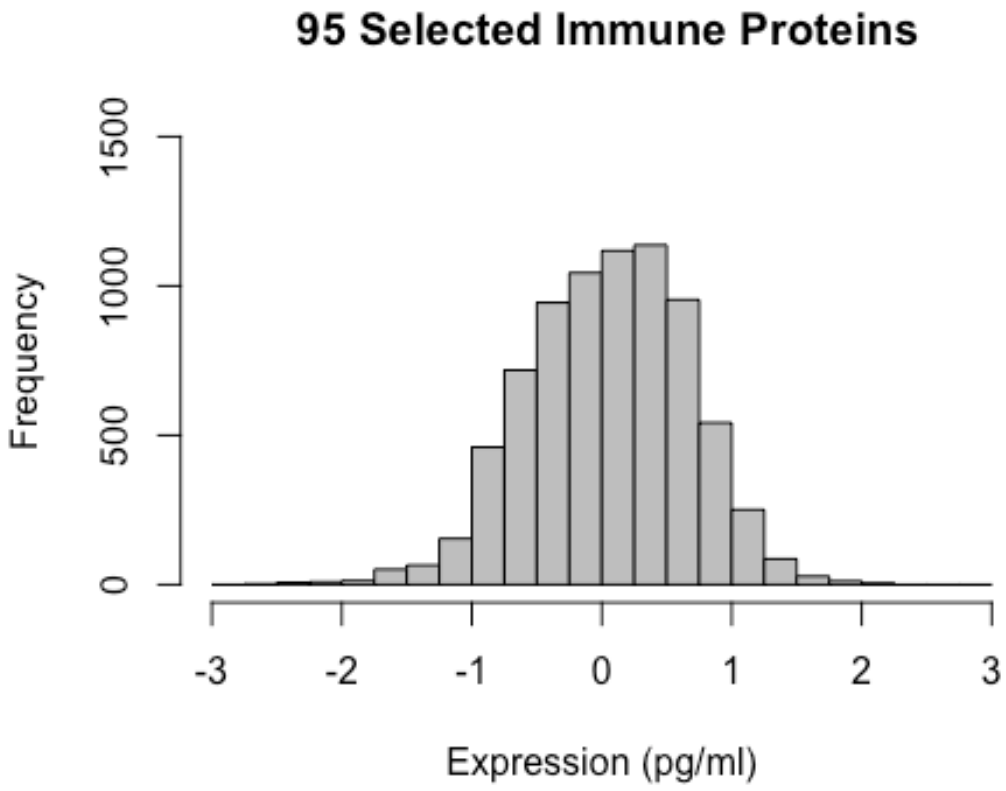
```
## [1] -2.571198
```

```
# Plot frequency histogram
loess.freq.hist = hist(loess.72.proteins.check, freq = TRUE, main = " 72 Selected Immune Proteins",
                       ylab= "Frequency", xlab= "Expression (pg/ml)", breaks = seq(-3,3, by = 0.25),
                       xlim = c(-3,3), ylim = c(0,1000), labels = F, cex= 0.3, col="grey")
```



Next, verify the normal distribution when looking at all 95 protein trajectories together. Recall that there is no extensive workflow for the plasticity marker data alone. So this verification step is not needed when looking at that dataset by itself.

```
loess.95.proteins.check = as.matrix(loess.95.proteins[,-1]) # histogram
function needs input to be a matrix
max(as.numeric(unlist(loess.95.proteins.check)), na.rm = TRUE) # use to set
max x-axis limits of frequency plot

## [1] 2.230528

min(as.numeric(unlist(loess.95.proteins.check)), na.rm = TRUE) # use to set
min x-axis limits of frequency plot

## [1] -2.571198

# Plot frequency histogram
loess.freq.hist = hist(loess.95.proteins.check, freq = TRUE, main = " 95
Selected Immune Proteins",
                       ylab= "Frequency", xlab= "Expression (pg/ml)", breaks
= seq(-3,3, by = 0.25),
                       xlim = c(-3,3), ylim = c(0,1500), labels = F, cex=
0.3, col="grey")
```



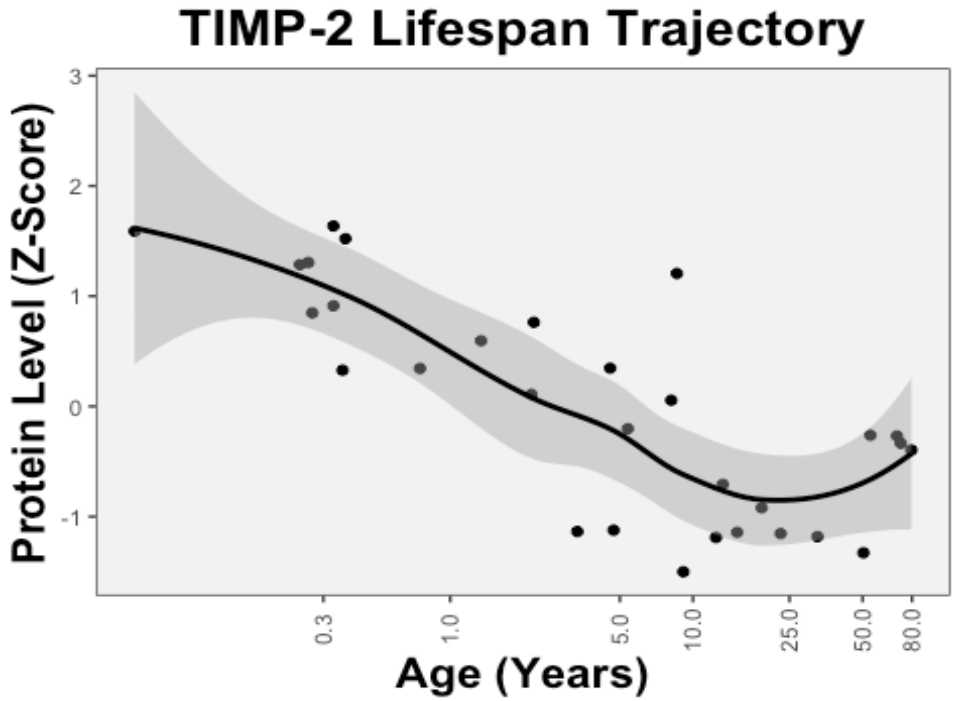
Images for Defense Presentation

The defense was delivered on zoom, the slides were horizontal (landscape), while the thesis was submitted as a vertical (portrait) orientation PDF. As a result, some of the figures were reorganized to be landscape instead of portrait for the presentation.

```
# Change width and height of 72 stamp collection, ie. number of rows and
columns changed
loess.72.stamp.collection = facet_multiple(plot = ggplots,
                                           facets = 'Protein',
                                           ncol = 9,
                                           nrow = 8,
                                           scales = "free_y")
# ggsave("Defence 72 Trajectories Stamp Collection.png", width = 11, height =
8)
```

For my defense, I focused in on some proteins to speak about more detail. I therefore, needed to extract those from the 72 protein stamp collection and graph them on their own (i.e. create larger image). In my thesis, I used a stamp collection of all these together, rather than creating a separate figure for each protein.

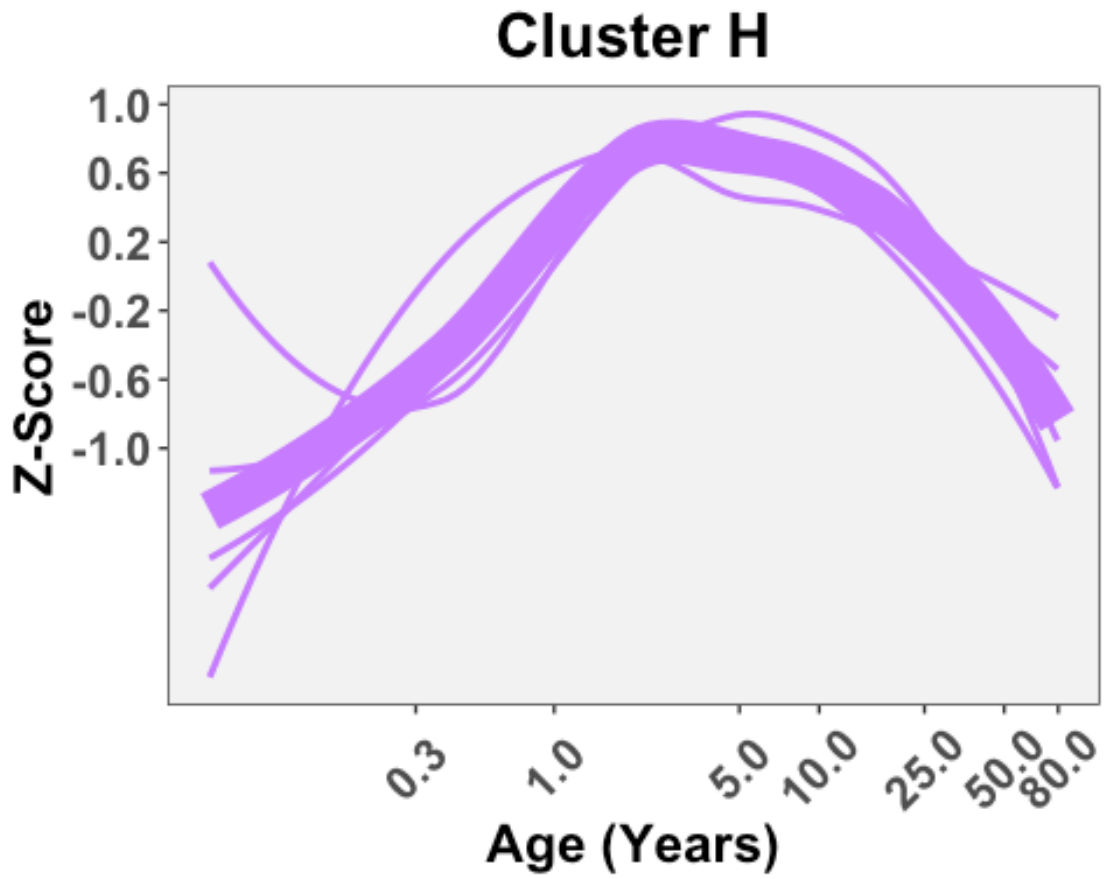
```
# Plot an individual protein
individual.zscore.plot = cbind(identifiers$Years, zscore.72.proteins) #
create a data frame containing all values
individual.zscore.plot = individual.zscore.plot[c(1, 66)] # change second
number to change protein extracted
colnames(individual.zscore.plot) = c("Age", "Protein") # set column names

# Plot data using ggplot2 function
ggplot(individual.zscore.plot, aes(Age, Protein))+
  scale_x_continuous(trans='log2',breaks = c(0.3, 1, 5, 10, 25, 50, 80)) + #
log2 transforms the x axis scale
  geom_point(size=1.5) +
  stat_smooth(method = "loess", colour = "black") +
  labs(x = "Age (Years)", y = "Protein Level (Z-Score)") +
  ggtitle("TIMP-2 Lifespan Trajectory")+ # change name when protein extracted
changes
  theme_bw()+
  theme(panel.background = element_rect(fill = 'gray95'),
        axis.text.x = element_text(angle= 90,vjust=0.5),
        axis.title.y = element_text(size = 17, face = "bold"),
        axis.title.x =element_text(size = 17,face = "bold"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        plot.title = element_text(hjust = 0.5, size = 20, face = "bold"))
```




```
# ggsave("TIMP-2 Lifespan Trajectory.png", height = 6, width = 8) # change
name when protein extracted changes
```

Similarly, I also wanted to pull out some individual clusters in my presentation. For visualization, I created an average curve for each neuroimmune protein cluster with labels. To do this I used the following code. In my thesis, I used a stamp collection of all these together, rather than creating a separate figure for each protein cluster.

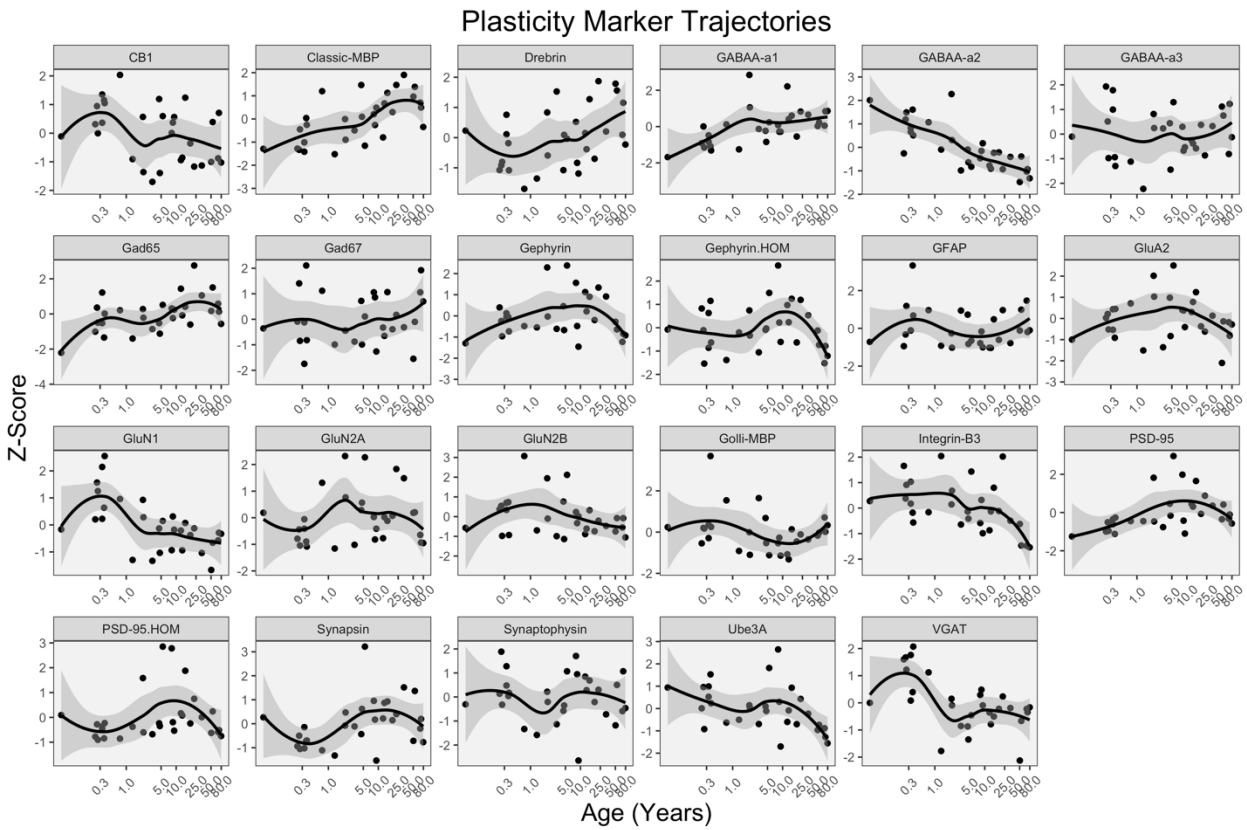
```
# Average with labels
ggplot(melted.ward.d2.clustering.data, aes(x = x, y = value, group =
variable, size= ifelse(variable=="Average", 4, 0.1))) +
  geom_line(color=cols[i]) +
  scale_x_continuous(trans='log2',breaks = c(0.3,1,5,10,25,50,80))+ #
transforms the x axis to log scale
  scale_y_continuous(breaks = seq(-1,1.4, 0.4)) +
  labs(y = "Z-Score", x = "Age (Years)") +
  theme_bw() +
  guides(group = TRUE) +
  theme(panel.background = element_rect(fill = 'gray95'),
        axis.text.x = element_text(size = 15, angle = 45, vjust = 0.5, face =
"bold"),
        axis.text.y = element_text(size = 15, face = "bold"),
        axis.title.y = element_text(size = 17, face = "bold"),
        axis.title.x =element_text(size = 17, face = "bold"),
        title = element_text(face = "bold"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        legend.position = "none",
        plot.title = element_text(hjust = 0.5, size = 20))+
  ggtitle(paste("Cluster", cluster.labels[i], sep = " "))
```




```
# ggsave("Cluster H - Individual plot with labels.png", height = 6, width =
8)
```

In my thesis, I jump straight into the clustering of the 23 synaptic proteins (protein trajectories are visualized in their clusters), and a supplemental file was created with these trajectories (as I already have 31 images in my thesis). However, I wanted to include a stamp collection of them in my presentation.

```
# Plot 23 Classic plasticity marker trajectories
plasticity.loess.curves = ggplot(graphable.data, aes(x = Years, y =
Expression)) +
  scale_x_continuous(trans = "log2", breaks = c(0.3, 1, 5, 10, 25, 50, 80)) +
  geom_point(size = 1.5, shape = 19) +
  facet_wrap(~Protein, ncol = 8, scales = "free_y") +
  stat_smooth(method = "loess", color = "black") +
  labs(title = "Plasticity Marker Trajectories", x = "Age (Years)", y = "Z-
Score") +
  theme_bw() +
  theme(panel.background = element_rect(fill = 'gray95'),
        axis.text.x = element_text(angle=45,vjust=0.5),
        axis.title.y = element_text(size = 17),
        axis.title.x =element_text(size = 17),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank(),
        plot.title = element_text(hjust = 0.5, size = 20))
plasticity.facet.loess = facet_multiple(plot = plasticity.loess.curves,
                                        facets = 'Protein',
                                        ncol = 6,
                                        nrow = 4,
                                        scales = "free")
plasticity.facet.loess
```




```
# ggsave("Stamp Collection 23 Plasticity Markers.png", height = 8, width = 12)
```

Fréchet Distances (Cluster Verification for Plasticity Marker Clustering Alone and Combined Data)

Similar to the neuroimmune clusters, to ensure that the clustering was in fact placing similar proteins together, I had to verify that proteins in the same cluster (i.e. intracluster Fréchet distance) is smaller than the distance between proteins in varying clusters (i.e. intercluster Fréchet distance).

This is done in a series of steps:

- Calculate within cluster distances
- Calculate between cluster distances
- Perform estimation statistics on them

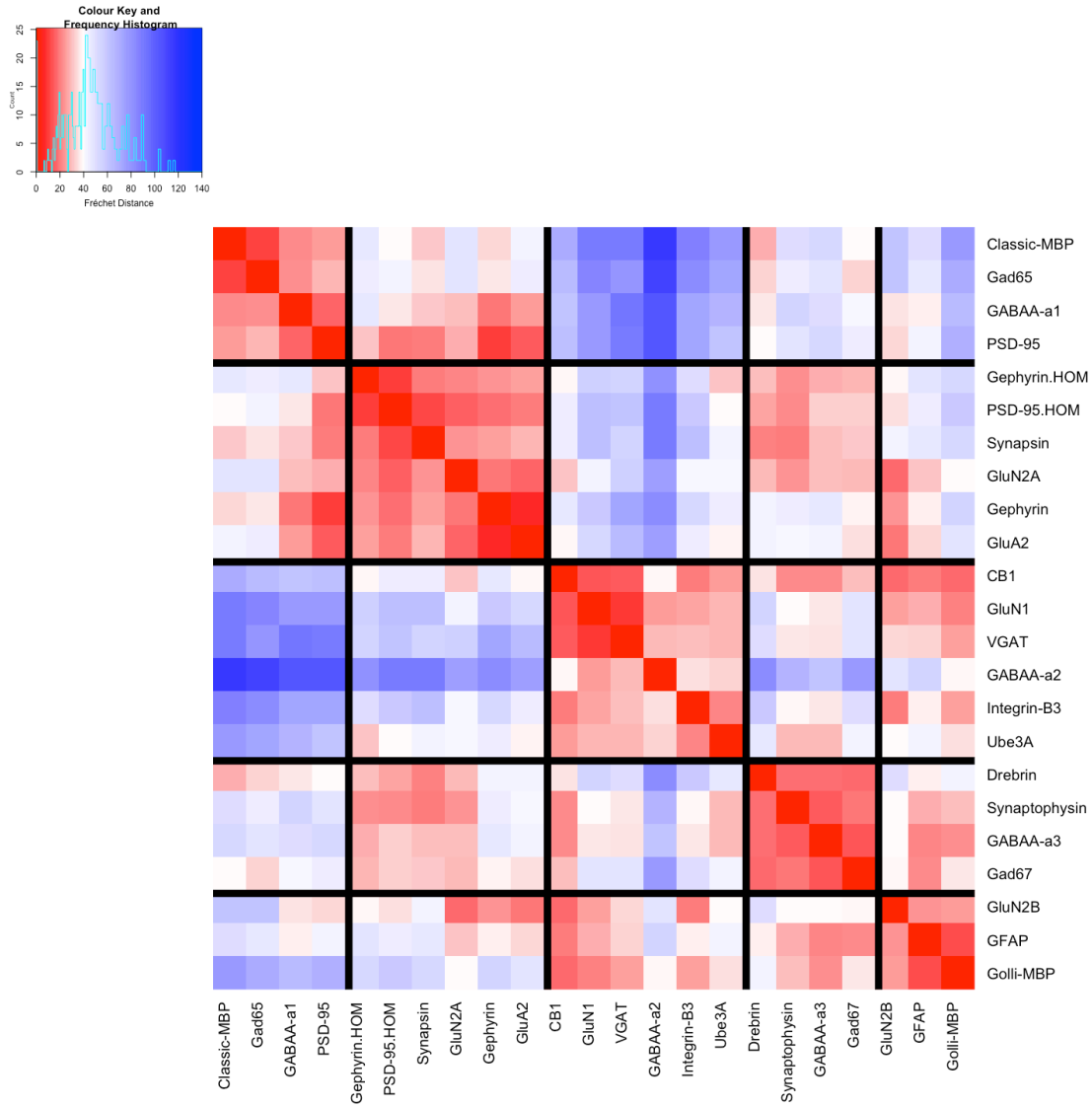
Plasticity Marker Data

Create a 23 x 23 Matrix

```
# Get combinations of proteins to compare (including self-comparisons)
frechet.scores.proteins.23 = expand.grid(colnames(loess.23.proteins[-1]),
colnames(loess.23.proteins[-1]))
# Create an empty column to store the distances
frechet.scores.proteins.23$Distance = NA
colnames(frechet.scores.proteins.23)[1:2] = c("Variable 1", "Variable 2")
x.value = loess.23.proteins$x # set the x values (ages) for the trajectories
(here my proteins were measured in the same samples, so only 1 unique x-value
vector needed)

for(i in 1:nrow(frechet.scores.proteins.23)){
  # export the z-score values for the first protein
  trajectory1 =
as.numeric(unlist(loess.23.proteins[as.character(frechet.scores.proteins.23[i
,1])]))
  # export the z-score values for the second protein
  trajectory2 =
as.numeric(unlist(loess.23.proteins[as.character(frechet.scores.proteins.23[i
,2])]))
  # calculate distance, use the sum method
  frechet.scores.proteins.23[i,3] = as.numeric(distFrechetR(x.value,
trajectory1, x.value, trajectory2, FrechetSumOrMax = "sum"))
}

# Store the median Frechet distances (without taking self-comparison into
account)
median.23.frechet.no.self =
median(subset(frechet.scores.proteins.23$Distance,
```

```
!frechet.scores.proteins.23$Distance == 0))

# Store the median Frechet distances (taking self-comparison into account)
median.23.frechet = median(frechet.scores.proteins.23$Distance)

# Convert from long to wide format for plotting
frechet.23.heatmap.data = spread(frechet.scores.proteins.23, `Variable 1`,
Distance) %>%
  column_to_rownames("Variable 2")

# set colour scheme for data frame, same colour scheme as 72 neuroimmune
proteins will be used
paletteLength = 99 # indicates how many gradations are wanted in the color
key
col = colorRampPalette(c("red", "white", "blue"))(paletteLength) # make the
color scale
# assign where the breaks must occur
mybreaks = c(seq(0,37,length=49),  # Distances up to 37 are coloured on a red
gradient
             seq(38,39,length=2),  # Distances between 38-39 are coloured
white
             seq(40,135,length=49)) # Distances between 40 and 135 are
coloured using a blue gradient



# reorder the columns and rows by the  hierarchical clustering ward.D2
dendrogram
frechet.23.heatmap.data.2 = frechet.23.heatmap.data[order.plasticity.dend,
order.plasticity.dend]

# png("23 Protein Frechet Heatmap.png", height = 15, width = 15, units =
"in", res = 1000)
heatmap.2(as.matrix(frechet.23.heatmap.data.2), # specify data frame
          col = col, # specify colour scale
          breaks = mybreaks, # specify breaks
          Colv = order.plasticity.dend, # order of columns
          Rowv = order.plasticity.dend, # order of rows
          dendrogram = "none", # apply default dendrogram (no)
          trace = "none", # do not draw a trace
          sepwidth = c(0.2,0.2), # create lines showing clusters
          sepcolor = "black", # colour of line
          cexRow = 1.5, # font size rows
          cexCol = 1.5, # font size columns
          key.xlab = "Fréchet Distance", # Key x label
          key.title = "Colour Key and\n Frequency Histogram", # Key title
          key.ylab = "Count", # Key y label
          lhei = c(2, 8), # Height of  key and the plot
          lwid = c(2, 8), # Width  of  key and the plot
```

```
          colsep=c(4,10,16,20), # Where to draw column lines to show clusters
          rowsep=c(4,10,16,20), # Where to draw row lines to show clusters
          margins = c(10,10), # set margins for the plot
          key.xtickfun = function() { # labels for the Key
            breaks = pretty(parent.frame()$breaks)
            list(at = parent.frame()$scale01(breaks),
                 labels = breaks)
          })
```




```
# dev.off()
```

Intracluster Fréchet Distances – 23

```
x.value = loess.23.proteins$x # x values for the proteins, all proteins have
the same x values in our data


# Create a list of lists that will store the frechet scores for each cluster
separately
frechet.scores.23 = list()
for(i in 1:5){
  sub.list = list()
  frechet.scores.23 = list.append(frechet.scores.23, sub.list)
  names(frechet.scores.23)[[i]] = paste("Cluster", seq(1,5,1)[[i]])
}


for (i in 1:5) { # Determine all combinations (except self-correlations)
  individual.clus = ward.D2.23.cluster.members[[i]] # get the names of
proteins in each cluster
  combinations = as.data.frame(combn(individual.clus, 2, simplify = TRUE)) #
generate pariwise comparison list (no self-comparisons)
  scores.vector = vector() # create an empty vector to store the distance
values for each cluster

  for (a in 1:ncol(combinations)){
    trajectory1 = as.numeric(unlist(loess.23.proteins[combinations[1, a]])) #
numeric vector of the first trajectories z-score values
    trajectory2 = as.numeric(unlist(loess.23.proteins[combinations[2, a]])) #
numeric vector of the first trajectories z-score values
    score = as.numeric(distFrechetR(x.value, trajectory1, x.value,
trajectory2, FrechetSumOrMax = "sum")) # calculate the frechet distance
between them
    scores.vector = append(scores.vector, score) # add the distance to the
scores vector
  }

  frechet.scores.23[[i]] =  scores.vector # take the scores for this cluster
vector and put it into the frechet.scores.23
}


# Save each of the values, one page for each cluster
list_of_datasets.23 = list("Cluster 1 Distances" = frechet.scores.23[[1]],
"Cluster 2 Distances" = frechet.scores.23[[2]],
```

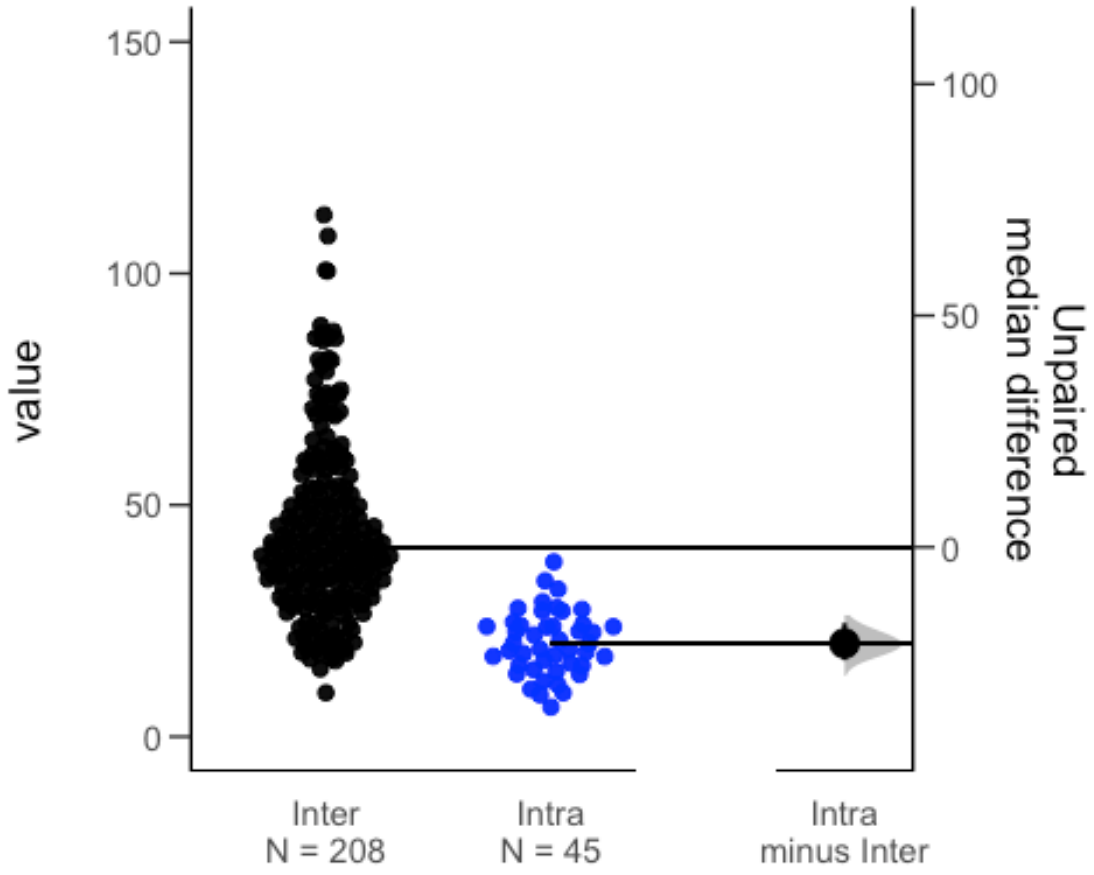
```
                            "Cluster 3 Distances" = frechet.scores.23[[3]],
"Cluster 4 Distances" = frechet.scores.23[[4]],
                            "Cluster 5 Distances" = frechet.scores.23[[5]])
# openxlsx::write.xlsx(list_of_datasets, file = "Frechet within cluster for
plasticity markers only.xlsx")


# Create a long format data frame from the list

frechet.boxplot.data.23 = as.data.frame(NA) # create an empty data frame
for(i in 1:length(frechet.scores.23)){
  # Add the frechet scores from each cluster as a column to the data frame
  frechet.boxplot.data.23 = qpcR:::cbind.na(frechet.boxplot.data.23,
as.data.frame(frechet.scores.23[[i]]))
  # Name each column by the cluster number
  colnames(frechet.boxplot.data.23)[i+1] = names(frechet.scores.23)[i]
}

frechet.boxplot.data.23 = frechet.boxplot.data.23[-1] # remove the empty
column
frechet.boxplot.data.23 = melt(frechet.boxplot.data.23, na.rm = TRUE) #
covert the data frame to the long format for estimation stats
```

Intracluster Vs. Intercluster

This is FIGURE 17 in my thesis.

```
# Inter- Vs. Intracluster 23

# The heatmap currently contains duplicate distances
frechet.scores.23.low = frechet.23.heatmap.data  # first store the values in
a different object for processing
frechet.scores.23.low[lower.tri(frechet.scores.23.low, diag=TRUE)] = NA #
subset the lower have (i.e. all unique values)

# Create a folder that holds only the between cluster values:
all.between.cluster.values = as.data.frame(matrix(ncol = 3, nrow = 1))
colnames(all.between.cluster.values) = c("Variable 1", "Variable 2", "Value")

for (i in 1:5){ # loop through each cluster
  # Extract the frechet distances that do not correspond to intracluster
comparisons
  not.intracluster = frechet.scores.23.low[ward.D2.23.cluster.members[[i]],
setdiff(colnames(frechet.scores.23.low), ward.D2.23.cluster.members[[i]])]
  # Convert the data frame to the long format
  not.intracluster = reshape2::melt(setDT(not.intracluster, keep.rownames =
TRUE), "rn", na.rm = TRUE)
  # Set column names
```

```
  colnames(not.intracluster) = c("Variable 1", "Variable 2", "Value")
  # Bind the indivudal cluster's  values to a data frame that will contain
all the clusters values
  all.between.cluster.values = rbind(all.between.cluster.values,
not.intracluster)
}

all.between.cluster.values = all.between.cluster.values[-1,] # remove empty
row
all.between.cluster.values = cbind("Inter", all.between.cluster.values$Value)
# add a row that indicates that these values are intercluster distances

# Create a data frame that contains all intracluster distances, labelled as
such (i.e. "Intra")
only.intracluster = cbind("Intra", frechet.boxplot.data.23$value)
# Bind the inter- and intra-cluster values together with labels
intra.inter.est.stats = as.data.frame(rbind(all.between.cluster.values,
only.intracluster))
colnames(intra.inter.est.stats) = c("variable", "value") # set column names
# Make distances numeric
intra.inter.est.stats$value = as.numeric(unlist(intra.inter.est.stats$value))

# Prepare data for estimation statistics
group.unpaired.plasticity.alone =
  intra.inter.est.stats %>%
  dabest(variable, value,
         idx = c("Inter", "Intra"),
         paired = FALSE)

# Perform statistical analysis
unpaired.mediandiff.plasticity.alone =
median_diff(group.unpaired.plasticity.alone)

# png("Estimation Stats Intracluster Vs Intercluster 23.png", height = 6,
width = 10, units = "in", res = 1200)
plot(unpaired.mediandiff.plasticity.alone,
           palette = c("black", "blue"),
           rawplot.ylim = c(0, 150),
           rawplot.type = "swarmplot")
```




```
# dev.off()
```

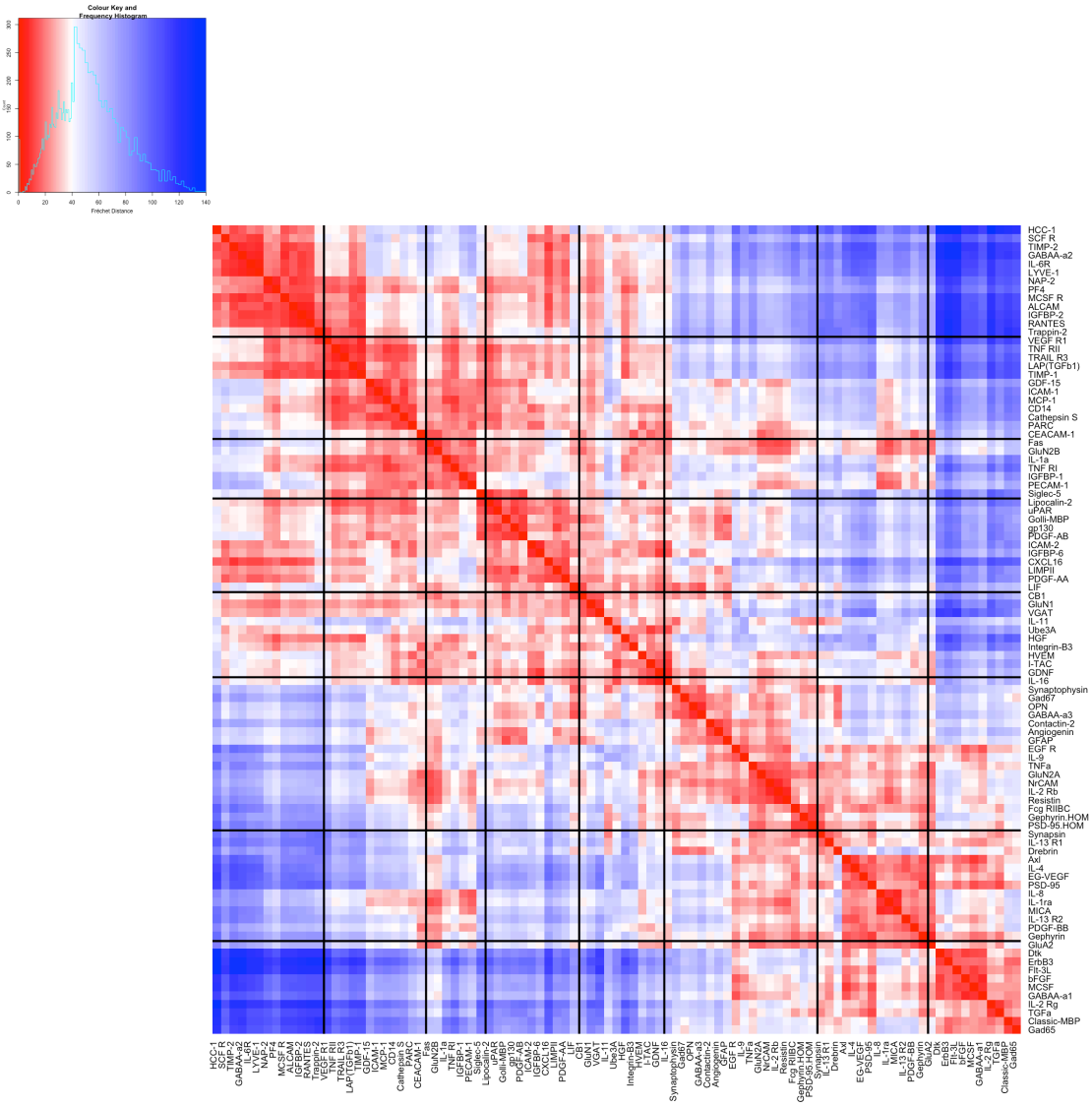
Combined Data

Create 95 x 95 Matrix

```
# Get combination of proteins to compare (including self comparisons)
frechet.scores.proteins.95 = expand.grid(colnames(loess.95.proteins[-1]),
colnames(loess.95.proteins[-1]))
frechet.scores.proteins.95$Distance = NA # create empty column to store the
distances
colnames(frechet.scores.proteins.95)[1:2] = c("Variable 1", "Variable 2")
x.value = loess.95.proteins$x # set the x values (ages) for the trajectories
(here my proteins were measured in the same samples, so only 1 unique x-value
vector needed)

for(i in 1:nrow(frechet.scores.proteins.95)){
  # Export the z-score values for the first protein
  trajectory1 =
as.numeric(unlist(loess.95.proteins[as.character(frechet.scores.proteins.95[i
,1])]))
  # Export the z-score values for the second protein
```

```
  trajectory2 =
as.numeric(unlist(loess.95.proteins[as.character(frechet.scores.proteins.95[i
,2])]))
  # Calculate distance, use the sum method
  frechet.scores.proteins.95[i,3] = as.numeric(distFrechetR(x.value,
trajectory1, x.value, trajectory2, FrechetSumOrMax = "sum"))
}

# Store the median Frechet distances (without taking self-comparison into
account)
median.95.frechet.no.self =
median(subset(frechet.scores.proteins.95$Distance,
!frechet.scores.proteins.95$Distance == 0))

# Store the median Frechet distances (while taking self-comparison into
account)
median.95.frechet = median(frechet.scores.proteins.95$Distance)

# Convert from long to wide format for plotting
frechet.95.heatmap.data = spread(frechet.scores.proteins.95, `Variable 1`,
Distance) %>%
  column_to_rownames("Variable 2")

# set colour scheme for data frame - same as figure 9 in thesis
paletteLength = 99 # indicates how many gradations are wanted in the color
key
col = colorRampPalette(c("red", "white", "blue"))(paletteLength) # make the
color scale
# assign where the breaks must occur
mybreaks = c(seq(0,37,length=49),  # Distances up to 37 are coloured on a red
gradient
             seq(38,39,length=2),  # Distances between 38-39 are coloured
white
             seq(40,135,length=49)) # Distances between 40 and 135 are
coloured using a blue gradient



order.combined.dend = order.dendrogram(as.dendrogram(combined.dendrogram)) #
get the index of the proteins
order.combined.dend = rownames(combined.dendrogram.data)[order.combined.dend]


# reorder the columns and rows by the  hierarchical clustering ward.D2
dendrogram
frechet.95.heatmap.data.2 = frechet.95.heatmap.data[order.combined.dend,
order.combined.dend]

# png("95 Protein Frechet Heatmap.png", height = 30, width = 30, units =
```

```
"in", res = 1000)
heatmap.2(as.matrix(frechet.95.heatmap.data.2), # specify data frame
          col = col, # specify colour scale
          breaks = mybreaks, # specify breaks
          Colv = order.combined.dend, # order of columns
          Rowv = order.combined.dend, # order of rows
          dendrogram = "none", # apply default dendrogram (no)
          trace = "none", # do not draw a trace
          sepwidth = c(0.2,0.2), # create lines showing clusters
          sepcolor = "black", # colour of line
          cexRow = 1.5, # font size rows
          cexCol = 1.5, # font size columns
          key.xlab = "Fréchet Distance", # Key x label
          key.title = "Colour Key and\n Frequency Histogram", # Key title
          key.ylab = "Count", # Key y label
          lhei = c(2, 8), # Height of  key and the plot
          lwid = c(2, 8), # Width  of  key and the plot
          colsep=c(13,25,32,43,53,71,84), # Where to draw column lines to
show clusters
          rowsep=c(13,25,32,43,53,71,84), # Where to draw row lines to show
clusters
          margins = c(10,10), # set margins for the plot
          key.xtickfun = function() { # labels for the Key
            breaks = pretty(parent.frame()$breaks)
            list(at = parent.frame()$scale01(breaks),
                 labels = breaks)
          }
)
```


Colour Key and
Frequency Histogram
Count
Frechet Distance
HCC-1
SCF R
TIMP-2
GABAA-a2
IL-6R
LYVE-1
NAP-2
PF4
MCSF R
ALCAM
IGFBP-2
RANTES
Trappin-2
VEGF R1
TNF RII
TRAIL R3
LAP(TGFb1)
TIMP-1
GDF-15
ICAM-1
MCP-1
CD14
Cathepsin S
PARC
CEACAM-1
Fas
GluN2B
IL-1a
TNF RI
IGFBP-1
PECAM-1
Siglec-5
Lipocalin-2
uPAR
Golli-MBP
gp130
PDGF-AB
ICAM-2
IGFBP-6
CXCL16
LIMPII
PDGF-AA
LIF
CB1
GluN1
VGAT
IL-11
Ube3A
HGF
Integrin-B3
HVEM
I-TAC
GDNF
IL-16
Synaptophysin
Gad67
OPN
GABAA-a3
Contactin-2
Angiogenin
GFAP
EGF R
IL-9
TNFa
GluN2A
NrCAM
IL-2 Rb
Resistin
Fcg RIIBC
Gephyrin.HOM
PSD-95.HOM
Synapsin
IL-13 R1
Drebrin
Axl
IL-4
EG-VEGF
PSD-95
IL-8
IL-1ra
MICA
IL-13 R2
PDGF-BB
Gephyrin
GluA2
Dtk
ErbB3
Flt-3L
bFGF
MCSF
GABAA-a1
IL-2 Rg
TGFa
Classic-MBP
Gad65

```
# dev.off()
```

Fréchet Distances – 95

```
# Intracluster Values

x.value = loess.95.proteins$x # x values for the proteins, all proteins have
the same x values in our data

# Create a list of lists that will store the frechet scores for each cluster
separately
frechet.scores.95 = list()
for(i in 1:8){
  sub.list = list()
  frechet.scores.95 = list.append(frechet.scores.95, sub.list)
  names(frechet.scores.95)[[i]] = paste("Cluster",
names(ward.D2.cluster.members)[[i]])
}


for (i in 1:8) { # Determine all combinations (except self-correlations)
  individual.clus = ward.D2.95.cluster.members[[i]] # get the names of
proteins in each cluster
  combinations = as.data.frame(combn(individual.clus, 2, simplify = TRUE)) #
generate pariwise comparison list (no self-comparisons)
  scores.vector = vector() # create an empty vector to store the distance
values for each cluster

  for (a in 1:ncol(combinations)){
    trajectory1 = as.numeric(unlist(loess.95.proteins[combinations[1, a]])) #
numeric vector of the first trajectories z-score values
    trajectory2 = as.numeric(unlist(loess.95.proteins[combinations[2, a]])) #
numeric vector of the first trajectories z-score values
    score = as.numeric(distFrechetR(x.value, trajectory1, x.value,
trajectory2, FrechetSumOrMax = "sum")) # calculate the frechet distance
between them
    scores.vector = append(scores.vector, score) # add the distance to the
scores vector
  }

  frechet.scores.95[[i]] =  scores.vector # take the scores for this cluster
vector and put it into the frechet.scores.95
}


# Save each of the values, one page for each cluster
list_of_datasets.95 = list("Cluster A Distances" = frechet.scores.95[[1]],
"Cluster B Distances" = frechet.scores.95[[2]],
```

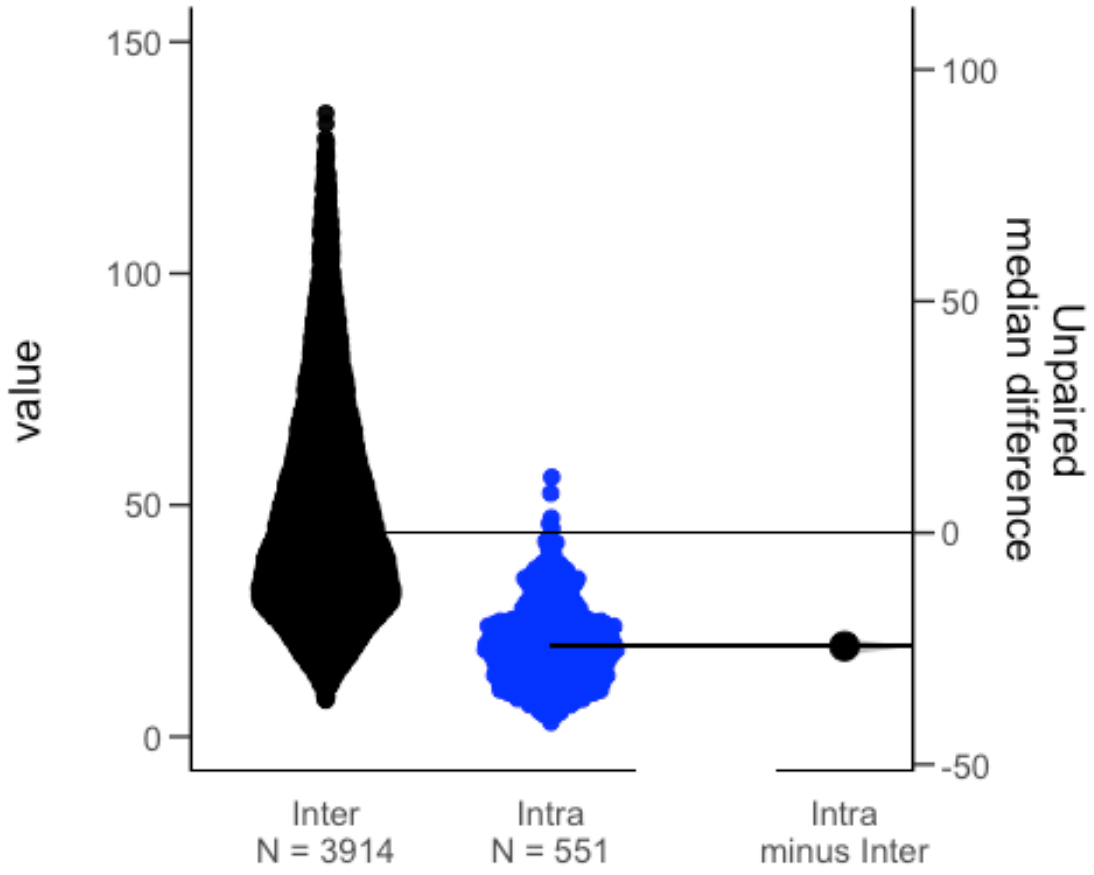
```
                            "Cluster C Distances" = frechet.scores.95[[3]],
"Cluster D Distances" = frechet.scores.95[[4]],
                            "Cluster E Distances" = frechet.scores.95[[5]],
"Cluster F Distances" = frechet.scores.95[[6]],
                            "Cluster G Distances" = frechet.scores.95[[7]],
"Cluster H Distances" = frechet.scores.95[[8]])
# openxlsx::write.xlsx(list_of_datasets, file = "Frechet within cluster for
combined data.xlsx")



# Create a long format data frame from the list

frechet.boxplot.data.95 = as.data.frame(NA) # create an empty data frame
for(i in 1:length(frechet.scores.95)){
  # Add the frechet scores from each cluster as a column to the data frame
  frechet.boxplot.data.95 = qpcR:::cbind.na(frechet.boxplot.data.95,
as.data.frame(frechet.scores.95[[i]]))
  # Name each column by the cluster number
  colnames(frechet.boxplot.data.95)[i+1] = names(frechet.scores.95)[i]
}

frechet.boxplot.data.95 = frechet.boxplot.data.95[-1] # remove the empty
column
frechet.boxplot.data.95 = melt(frechet.boxplot.data.95, na.rm = TRUE) #
covert the data frame to the long format for estimation stats
```

Intracluster Vs. Intercluster

This is FIGURE 23 in my thesis.

```
# Inter Vs. Intracluster 95

# The heatmap currently contains duplicate distances
frechet.scores.95.low = frechet.95.heatmap.data  # first store the values in
a different object for processing
frechet.scores.95.low[lower.tri(frechet.scores.95.low, diag=TRUE)] = NA #
subset the lower have (i.e. all unique values)

# Create a folder that holds only the between cluster values:
all.between.cluster.values = as.data.frame(matrix(ncol = 3, nrow = 1))
colnames(all.between.cluster.values) = c("Variable 1", "Variable 2", "Value")

for (i in 1:8){ # loop through each cluster
  # Extract the frechet distances that do not correspond to intracluster
comparisons
  not.intracluster = frechet.scores.95.low[ward.D2.95.cluster.members[[i]],
setdiff(colnames(frechet.scores.95.low), ward.D2.95.cluster.members[[i]])]
```

```
  # Convert the data frame to the long format
  not.intracluster = reshape2::melt(setDT(not.intracluster, keep.rownames =
TRUE), "rn", na.rm = TRUE)
  # Set column names
  colnames(not.intracluster) = c("Variable 1", "Variable 2", "Value")
  # Bind the individual cluster's  values to a data frame that will contain
all the clusters values
  all.between.cluster.values = rbind(all.between.cluster.values,
not.intracluster)
}

all.between.cluster.values = all.between.cluster.values[-1,] # remove empty
row
all.between.cluster.values = cbind("Inter", all.between.cluster.values$Value)
# add a row that indicates that these values are intercluster distances

# Create a data frame that contains all intracluster distances, labelled as
such (i.e. "Intra")
only.intracluster = cbind("Intra", frechet.boxplot.data.95$value)
# Bind the inter- and intra-cluster values together with labels
intra.inter.est.stats = as.data.frame(rbind(all.between.cluster.values,
only.intracluster))
colnames(intra.inter.est.stats) = c("variable", "value") # set column names
# Make distances numeric
intra.inter.est.stats$value = as.numeric(unlist(intra.inter.est.stats$value))

# Prepare data for estimation statistics
group.unpaired.combined =
  intra.inter.est.stats %>%
  dabest(variable, value,
         idx = c("Inter", "Intra"),
         paired = FALSE)

# Perform statistical analysis
unpaired.mediandiff.combined = median_diff(group.unpaired.combined)

# png("Estimation Stats Intracluster Vs Intercluster 95.png", height = 6,
width = 10, units = "in", res = 1200)
print(plot(unpaired.mediandiff.combined,
           palette = c("black", "blue"),
           rawplot.ylim = c(0, 150),
           rawplot.type = "swarmplot"))
```




```
# dev.off()
```

Note: You can also perform a within cluster estimation statistics test to get a comparison of level of heterogeneity between clusters and you can also perform estimation statistics on the between clusters as well. Nonetheless, for the purpose of my thesis, these analyses were not extensively explored for the plasticity-only clusters and the combined data set as these analyses were used in a different context than the neuroimmune data (i.e. to find candidate plasticity processes for neuroimmune proteins - not to determine the number of trajectories in the cortex). Use the code found in the main thesis markdown to compute these values if desired.

Nonetheless, note that for the plasticity marker data these analyses would be non-informative as the cluster sizes are very small (all clusters are six proteins or less) thus bootstrapping to compare them will yield very large confidence intervals that are not completely informative.

References

NOTE: This code was used to get the citations in the proper format as provided by RStudio. A concise reference list is found after the code on page 234.

*****

guiastrennec/ggplus Version 0.1: The source code and available functions in the package are found at the following URL: https://rdrr.io/github/guiastrennec/ggplus/f/

R Core Team (2013). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

RStudio Team (2020). RStudio: Integrated Development for R. RStudio, PBC, Boston, MA URL http://www.rstudio.com/.

Citations for packages below

```
# Create a list of packages to be cited.
session.info = sessionInfo()
session.info
```

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
## Matrix products: default
## BLAS:
/Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
## LAPACK:
/Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_CA.UTF-8/en_CA.UTF-8/en_CA.UTF-8/C/en_CA.UTF-8/en_CA.UTF-8
##
## attached base packages:
## [1] stats4    grid      stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] ggplus_0.1             dendextend_1.14.0      reshape2_1.4.4
##  [4] ggpubr_0.4.0           gridExtra_2.3          clusteval_0.1
##  [7] Rmisc_1.5              plyr_1.8.6             dabestr_0.3.0
## [10] openxlsx_4.1.5         readxl_1.3.1           FactoMineR_2.3
## [13] RSKC_2.4.2             flexclust_1.4-0        modeltools_0.2-23
## [16] VIM_6.0.0              colorspace_1.4-1       mice_3.11.0
## [19] gprofiler2_0.2.0       qpcR_1.4-1             robustbase_0.93-6
```

```
## [22] minpack.lm_1.2-1       plotrix_3.7-8          rlist_0.4.6.1
## [25] gplots_3.0.4           kmlShape_0.9.5         kml_2.4.1
## [28] longitudinalData_2.4.1 misc3d_0.8-4           rgl_0.100.54
## [31] clv_0.3-2.2            cluster_2.1.0          class_7.3-17
## [34] pvclust_2.2-0          factoextra_1.0.7       mosaic_1.7.0
## [37] Matrix_1.2-18          mosaicData_0.18.0      ggformula_0.9.4
## [40] ggstance_0.3.4         lattice_0.20-41        fitdistrplus_1.1-1
## [43] survival_3.2-3         MASS_7.3-52            devtools_2.3.1
## [46] usethis_1.6.1          data.table_1.13.0      zoo_1.8-8
## [49] magrittr_1.5           janitor_2.0.1          forcats_0.5.0
## [52] stringr_1.4.0          dplyr_1.0.2            purrr_0.3.4
## [55] readr_1.3.1            tidyr_1.1.2            tibble_3.0.3
## [58] ggplot2_3.3.2          tidyverse_1.3.0        knitr_1.29
##
## loaded via a namespace (and not attached):
##   [1] tidyselect_1.1.0        htmlwidgets_1.5.1       ranger_0.12.1
##   [4] munsell_0.5.0           codetools_0.2-16        miniUI_0.1.1.1
##   [7] withr_2.2.0             highr_0.8               rstudioapi_0.11
##  [10] leaps_3.1               ggsignif_0.6.0          vcd_1.4-7
##  [13] labeling_0.3            polyclip_1.10-0         farver_2.0.3
##  [16] rprojroot_1.3-2         vctrs_0.3.3             generics_0.0.2
##  [19] xfun_0.16               R6_2.4.1                ggbeeswarm_0.6.0
##  [22] manipulateWidget_0.10.1 bitops_1.0-6            assertthat_0.2.1
##  [25] promises_1.1.1          scales_1.1.1            nnet_7.3-14
##  [28] beeswarm_0.2.3          gtable_0.3.0            processx_3.4.3
##  [31] rlang_0.4.7             scatterplot3d_0.3-41    splines_4.0.2
##  [34] rstatix_0.6.0           lazyeval_0.2.2          broom_0.7.0
##  [37] mosaicCore_0.6.0        yaml_2.2.1              abind_1.4-5
##  [40] modelr_0.1.8            crosstalk_1.1.0.1       backports_1.1.9
##  [43] httpuv_1.5.4            tools_4.0.2             ellipsis_0.3.1
##  [46] ggdendro_0.1.21         sessioninfo_1.1.1       Rcpp_1.0.5
##  [49] RCurl_1.98-1.2          ps_1.3.4                prettyunits_1.1.1
##  [52] viridis_0.5.1           cowplot_1.0.0           haven_2.3.1
##  [55] ggrepel_0.8.2           fs_1.5.0                lmtest_0.9-37
##  [58] reprex_0.3.0            mvtnorm_1.1-1           pkgload_1.1.0
##  [61] hms_0.5.3               mime_0.9                evaluate_0.14
##  [64] xtable_1.8-4            leaflet_2.0.3           rio_0.5.16
##  [67] testthat_2.3.2          compiler_4.0.2          KernSmooth_2.23-17
##  [70] crayon_1.3.4            htmltools_0.5.0         mgcv_1.8-33
##  [73] later_1.1.0.1           lubridate_1.7.9         DBI_1.1.0
##  [76] tweenr_1.0.1            dbplyr_1.4.4            boot_1.3-25
##  [79] car_3.0-9               cli_2.0.2               gdata_2.18.0
##  [82] parallel_4.0.2          pkgconfig_2.0.3         flashClust_1.01-2
##  [85] foreign_0.8-80          laeken_0.5.1            sp_1.4-2
##  [88] plotly_4.9.2.1          xml2_1.3.2              vipor_0.4.5
##  [91] webshot_0.5.2           rvest_0.3.6             snakecase_0.11.0
##  [94] callr_3.4.3             digest_0.6.25           rmarkdown_2.3
##  [97] cellranger_1.1.0        curl_4.3                shiny_1.5.0
## [100] gtools_3.8.2            nlme_3.1-149            lifecycle_0.2.0
## [103] jsonlite_1.7.0          carData_3.0-4           desc_1.2.0
```

```
## [106] viridisLite_0.3.0       fansi_0.4.1             pillar_1.4.6
## [109] fastmap_1.0.1           httr_1.4.2              DEoptimR_1.0-8
## [112] pkgbuild_1.1.0          glue_1.4.2              remotes_2.2.0
## [115] zip_2.1.1               ggforce_0.3.2           stringi_1.4.6
## [118] blob_1.2.1              caTools_1.18.0          memoise_1.1.0
## [121] e1071_1.7-3
```

```
packages = session.info[["basePkgs"]] # Get package version information etc.
packages = append(packages, names(session.info[["otherPkgs"]]))

# Packages not attached (i.e. loaded only) have not been used in the
analyses, so cite packages that are attached.

for (i in 1:length(packages)){
  print(citation(package = packages[i]))
}
```

```
##
## The 'stats4' package is part of R.  To cite R in publications use:
##
##   R Core Team (2020). R: A language and environment for statistical
##   computing. R Foundation for Statistical Computing, Vienna, Austria.
##   URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2020},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please cite it
## when using it for data analysis. See also 'citation("pkgname")' for
## citing R packages.
##
##
## The 'grid' package is part of R.  To cite R in publications use:
##
##   R Core Team (2020). R: A language and environment for statistical
##   computing. R Foundation for Statistical Computing, Vienna, Austria.
##   URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
```

```
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2020},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please cite it
## when using it for data analysis. See also 'citation("pkgname")' for
## citing R packages.
##
##
## The 'stats' package is part of R.  To cite R in publications use:
##
##   R Core Team (2020). R: A language and environment for statistical
##   computing. R Foundation for Statistical Computing, Vienna, Austria.
##   URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2020},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please cite it
## when using it for data analysis. See also 'citation("pkgname")' for
## citing R packages.
##
##
## The 'graphics' package is part of R.  To cite R in publications use:
##
##   R Core Team (2020). R: A language and environment for statistical
##   computing. R Foundation for Statistical Computing, Vienna, Austria.
##   URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2020},
##     url = {https://www.R-project.org/},
##   }
```

```
##
## We have invested a lot of time and effort in creating R, please cite it
## when using it for data analysis. See also 'citation("pkgname")' for
## citing R packages.
##
##
## The 'grDevices' package is part of R.  To cite R in publications use:
##
##   R Core Team (2020). R: A language and environment for statistical
##   computing. R Foundation for Statistical Computing, Vienna, Austria.
##   URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2020},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please cite it
## when using it for data analysis. See also 'citation("pkgname")' for
## citing R packages.
##
##
## The 'utils' package is part of R.  To cite R in publications use:
##
##   R Core Team (2020). R: A language and environment for statistical
##   computing. R Foundation for Statistical Computing, Vienna, Austria.
##   URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2020},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please cite it
## when using it for data analysis. See also 'citation("pkgname")' for
## citing R packages.
##
##
```

```
## The 'datasets' package is part of R.  To cite R in publications use:
##
##   R Core Team (2020). R: A language and environment for statistical
##   computing. R Foundation for Statistical Computing, Vienna, Austria.
##   URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2020},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please cite it
## when using it for data analysis. See also 'citation("pkgname")' for
## citing R packages.
##
##
## The 'methods' package is part of R.  To cite R in publications use:
##
##   R Core Team (2020). R: A language and environment for statistical
##   computing. R Foundation for Statistical Computing, Vienna, Austria.
##   URL https://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2020},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please cite it
## when using it for data analysis. See also 'citation("pkgname")' for
## citing R packages.
##
##
## To cite R in publications use:
##
##   R Core Team (2020). R: A language and environment for statistical
##   computing. R Foundation for Statistical Computing, Vienna, Austria.
##   URL https://www.R-project.org/.
##
```

```
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistical Computing},
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Computing},
##     address = {Vienna, Austria},
##     year = {2020},
##     url = {https://www.R-project.org/},
##   }
##
## We have invested a lot of time and effort in creating R, please cite it
## when using it for data analysis. See also 'citation("pkgname")' for
## citing R packages.
##
##
## To cite package 'ggplus' in publications use:
##
##   Benjamin Guiastrennec (2020). ggplus: Set of additional functions for
##   ggplot2. R package version 0.1.
##   https://github.com/guiastrennec/ggplus
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {ggplus: Set of additional functions for ggplot2},
##     author = {Benjamin Guiastrennec},
##     year = {2020},
##     note = {R package version 0.1},
##     url = {https://github.com/guiastrennec/ggplus},
##   }
##
## ATTENTION: This citation information has been auto-generated from the
## package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
##
##
## The methods within the package can be cited as:
##
##   Tal Galili (2015). dendextend: an R package for visualizing,
##   adjusting, and comparing trees of hierarchical clustering.
##   Bioinformatics. DOI: 10.1093/bioinformatics/btv428
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     author = {Tal Galili},
##     title = {dendextend: an R package for visualizing, adjusting, and
comparing trees of hierarchical clustering},
##     journal = {Bioinformatics},
```

```
##     year = {2015},
##     doi = {10.1093/bioinformatics/btv428},
##     url =
{https://academic.oup.com/bioinformatics/article/31/22/3718/240978/dendextend
-an-R-package-for-visualizing-adjusting},
##     eprint = {https://academic.oup.com/bioinformatics/article-
pdf/31/22/3718/17122682/btv428.pdf},
##   }
##
## This free open-source software implements academic research by the
## authors and co-workers. If you use it, please support the project by
## citing the appropriate journal articles.
##
##
## To cite reshape2 in publications use:
##
##   Hadley Wickham (2007). Reshaping Data with the reshape Package.
##   Journal of Statistical Software, 21(12), 1-20. URL
##   http://www.jstatsoft.org/v21/i12/.
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {Reshaping Data with the {reshape} Package},
##     author = {Hadley Wickham},
##     journal = {Journal of Statistical Software},
##     year = {2007},
##     volume = {21},
##     number = {12},
##     pages = {1--20},
##     url = {http://www.jstatsoft.org/v21/i12/},
##   }
##
##
## To cite package 'ggpubr' in publications use:
##
##   Alboukadel Kassambara (2020). ggpubr: 'ggplot2' Based Publication
##   Ready Plots. R package version 0.4.0.
##   https://CRAN.R-project.org/package=ggpubr
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {ggpubr: 'ggplot2' Based Publication Ready Plots},
##     author = {Alboukadel Kassambara},
##     year = {2020},
##     note = {R package version 0.4.0},
##     url = {https://CRAN.R-project.org/package=ggpubr},
##   }
##
```

```
##
## To cite package 'gridExtra' in publications use:
##
##   Baptiste Auguie (2017). gridExtra: Miscellaneous Functions for "Grid"
##   Graphics. R package version 2.3.
##   https://CRAN.R-project.org/package=gridExtra
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {gridExtra: Miscellaneous Functions for "Grid" Graphics},
##     author = {Baptiste Auguie},
##     year = {2017},
##     note = {R package version 2.3},
##     url = {https://CRAN.R-project.org/package=gridExtra},
##   }
##
##
## To cite package 'clusteval' in publications use:
##
##   John A. Ramey (2012). clusteval: Evaluation of Clustering Algorithms.
##   R package version 0.1. https://CRAN.R-project.org/package=clusteval
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {clusteval: Evaluation of Clustering Algorithms},
##     author = {John A. Ramey},
##     year = {2012},
##     note = {R package version 0.1},
##     url = {https://CRAN.R-project.org/package=clusteval},
##   }
##
## ATTENTION: This citation information has been auto-generated from the
## package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
##
##
## To cite package 'Rmisc' in publications use:
##
##   Ryan M. Hope (2013). Rmisc: Rmisc: Ryan Miscellaneous. R package
##   version 1.5. https://CRAN.R-project.org/package=Rmisc
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {Rmisc: Rmisc: Ryan Miscellaneous},
##     author = {Ryan M. Hope},
##     year = {2013},
##     note = {R package version 1.5},
```

```
##     url = {https://CRAN.R-project.org/package=Rmisc},
##   }
##
## ATTENTION: This citation information has been auto-generated from the
## package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
##
##
## To cite plyr in publications use:
##
##   Hadley Wickham (2011). The Split-Apply-Combine Strategy for Data
##   Analysis. Journal of Statistical Software, 40(1), 1-29. URL
##   http://www.jstatsoft.org/v40/i01/.
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {The Split-Apply-Combine Strategy for Data Analysis},
##     author = {Hadley Wickham},
##     journal = {Journal of Statistical Software},
##     year = {2011},
##     volume = {40},
##     number = {1},
##     pages = {1--29},
##     url = {http://www.jstatsoft.org/v40/i01/},
##   }
##
##
## To cite dabestr in publications, please use:
##
##   Moving beyond P values: Everyday data analysis with estimation plots.
##   (2019) Joses Ho, Tayfun Tumkaya, Sameer Aryal, Hyungwon Choi, Adam
##   Claridge-Chang. Nature Methods 2019, 1548-710. doi:
##   https://doi.org/10.1038/s41592-019-0470-3
##
## A BibTeX entry for LaTeX users is
##
##   @Misc{,
##     title = {Moving beyond P values: Everyday data analysis with
estimation plots.},
##     author = {Joses Ho and Tayfun Tumkaya and Sameer Aryal and Hyungwon
Choi and Adam Claridge-Chang},
##     year = {2019},
##     doi = {10.1038/s41592-019-0470-3},
##     url = {https://rdcu.be/bHhJ4},
##   }
##
##
## To cite package 'openxlsx' in publications use:
##
```

```
##   Philipp Schauberger and Alexander Walker (2020). openxlsx: Read,
##   Write and Edit xlsx Files. R package version 4.1.5.
##   https://CRAN.R-project.org/package=openxlsx
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {openxlsx: Read, Write and Edit xlsx Files},
##     author = {Philipp Schauberger and Alexander Walker},
##     year = {2020},
##     note = {R package version 4.1.5},
##     url = {https://CRAN.R-project.org/package=openxlsx},
##   }
##
##
## To cite package 'readxl' in publications use:
##
##   Hadley Wickham and Jennifer Bryan (2019). readxl: Read Excel Files. R
##   package version 1.3.1. https://CRAN.R-project.org/package=readxl
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {readxl: Read Excel Files},
##     author = {Hadley Wickham and Jennifer Bryan},
##     year = {2019},
##     note = {R package version 1.3.1},
##     url = {https://CRAN.R-project.org/package=readxl},
##   }
##
##
## To cite FactoMineR in publications use:
##
##   Sebastien Le, Julie Josse, Francois Husson (2008). FactoMineR: An R
##   Package for Multivariate Analysis. Journal of Statistical Software,
##   25(1), 1-18. 10.18637/jss.v025.i01
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {{FactoMineR}: A Package for Multivariate Analysis},
##     author = {S\'ebastien L\^e and Julie Josse and Fran\c{c}ois Husson},
##     journal = {Journal of Statistical Software},
##     year = {2008},
##     volume = {25},
##     number = {1},
##     pages = {1--18},
##     doi = {10.18637/jss.v025.i01},
##   }
##
```

```
##
## To cite RSKC in publications use:
##
##   Yumi Kondo, Matias Salibian-Barrera, Ruben Zamar (2016). RSKC: An R
##   Package for a Robust and Sparse K-Means Clustering Algorithm. Journal
##   of Statistical Software, 72(5), 1-26. doi:10.18637/jss.v072.i05
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {{RSKC}: An {R} Package for a Robust and Sparse K-Means
Clustering Algorithm},
##     author = {Yumi Kondo and Matias Salibian-Barrera and Ruben Zamar},
##     journal = {Journal of Statistical Software},
##     year = {2016},
##     volume = {72},
##     number = {5},
##     pages = {1--26},
##     doi = {10.18637/jss.v072.i05},
##   }
##
##
## To cite package flexclust in publications use:
##
##   Friedrich Leisch. A Toolbox for K-Centroids Cluster Analysis.
##   Computational Statistics and Data Analysis, 51 (2), 526-544, 2006.
##
## Visualization methods are explained in:
##
##   Friedrich Leisch. Neighborhood graphs, stripes and shadow plots for
##   cluster visualization. Statistics and Computing, 20 (4), 457-469,
##   2010.
##
## Group constraints are explained in:
##
##   Friedrich Leisch and Bettina Gruen. Extending standard cluster
##   algorithms to allow for group constraints. In Alfredo Rizzi and
##   Maurizio Vichi, editors, Compstat 2006-Proceedings in Computational
##   Statistics, pages 885-892. Physica Verlag, Heidelberg, Germany, 2006.
##
## Function qtclust() is explained in:
##
##   Theresa Scharl and Friedrich Leisch. The Stochastic QT-clust
##   Algorithm: Evaluation of Stability and Variance on Time-course
##   Microarray Data. In Alfredo Rizzi and Maurizio Vichi, editors,
##   Compstat 2006-Proceedings in Computational Statistics, pages
##   1015-1022. Physica Verlag, Heidelberg, Germany, 2006.
##
## Using flexclust for market segmentation is explained in:
##
```

```
##   Sara Dolnicar, Bettina Gruen, and Friedrich Leisch. Market
##   Segmentation Analysis --- Understanding It, Doing It, and Making It
##   Useful. Management for Professionals. Springer, Singapore, 2018.
##   doi:10.1007/978-981-10-8818-6
##
## Papers are available from
## https://homepage.boku.ac.at/leisch/papers/fl-publications.html
## To see these entries in BibTeX format, use 'print(<citation>,
## bibtex=TRUE)', 'toBibtex(.)', or set
## 'options(citation.bibtex.max=999)'.
##
##
## To cite package 'modeltools' in publications use:
##
##   Torsten Hothorn, Friedrich Leisch and Achim Zeileis (2020).
##   modeltools: Tools and Classes for Statistical Models. R package
##   version 0.2-23. https://CRAN.R-project.org/package=modeltools
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {modeltools: Tools and Classes for Statistical Models},
##     author = {Torsten Hothorn and Friedrich Leisch and Achim Zeileis},
##     year = {2020},
##     note = {R package version 0.2-23},
##     url = {https://CRAN.R-project.org/package=modeltools},
##   }
##
## ATTENTION: This citation information has been auto-generated from the
## package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
##
##
## To cite VIM in publications use:
##
##   Alexander Kowarik, Matthias Templ (2016). Imputation with the R
##   Package VIM. Journal of Statistical Software, 74(7), 1-16.
##   doi:10.18637/jss.v074.i07
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {Imputation with the {R} Package {VIM}},
##     author = {Alexander Kowarik and Matthias Templ},
##     journal = {Journal of Statistical Software},
##     year = {2016},
##     volume = {74},
##     number = {7},
##     pages = {1--16},
##     doi = {10.18637/jss.v074.i07},
```

```
##   }
##
##
## To cite colorspace in publications use
##
## Zeileis A, Fisher JC, Hornik K, Ihaka R, McWhite CD, Murrell P,
## Stauffer R, Wilke CO (2019). "colorspace: A Toolbox for Manipulating
## and Assessing Colors and Palettes." arXiv 1903.06490, arXiv.org E-Print
## Archive. <URL: http://arxiv.org/abs/1903.06490>.
##
## If you use HCL-based color palettes, please cite:
##
## Zeileis A, Hornik K, Murrell P (2009). "Escaping RGBland: Selecting
## Colors for Statistical Graphics." _Computational Statistics \& Data
## Analysis_, *53*(9), 3259-3270. doi: 10.1016/j.csda.2008.11.033 (URL:
## https://doi.org/10.1016/j.csda.2008.11.033).
##
## If you use HCL-based color palettes in meteorological visualizations,
## please cite:
##
## Stauffer R, Mayr GJ, Dabernig M, Zeileis A (2009). "Somewhere over the
## Rainbow: How to Make Effective Use of Colors in Meteorological
## Visualizations." _Bulletin of the American Meteorological Society_,
## *96*(2), 203-216. doi: 10.1175/BAMS-D-13-00155.1 (URL:
## https://doi.org/10.1175/BAMS-D-13-00155.1).
##
## To see these entries in BibTeX format, use 'print(<citation>,
## bibtex=TRUE)', 'toBibtex(.)', or set
## 'options(citation.bibtex.max=999)'.
##
##
## To cite mice in publications use:
##
##   Stef van Buuren, Karin Groothuis-Oudshoorn (2011). mice: Multivariate
##   Imputation by Chained Equations in R. Journal of Statistical
##   Software, 45(3), 1-67. URL https://www.jstatsoft.org/v45/i03/.
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {{mice}: Multivariate Imputation by Chained Equations in R},
##     author = {Stef {van Buuren} and Karin Groothuis-Oudshoorn},
##     journal = {Journal of Statistical Software},
##     year = {2011},
##     volume = {45},
##     number = {3},
##     pages = {1-67},
##     url = {https://www.jstatsoft.org/v45/i03/},
##   }
##
```

```
##
## To cite package 'gprofiler2' in publications use:
##
##   Liis Kolberg and Uku Raudvere (2020). gprofiler2: Interface to the
##   'g:Profiler' Toolset. R package version 0.2.0.
##   https://CRAN.R-project.org/package=gprofiler2
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {gprofiler2: Interface to the 'g:Profiler' Toolset},
##     author = {Liis Kolberg and Uku Raudvere},
##     year = {2020},
##     note = {R package version 0.2.0},
##     url = {https://CRAN.R-project.org/package=gprofiler2},
##   }
##
## ATTENTION: This citation information has been auto-generated from the
## package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
##
##
## To cite package 'qpcR' in publications use:
##
##   Andrej-Nikolai Spiess (2018). qpcR: Modelling and Analysis of
##   Real-Time PCR Data. R package version 1.4-1.
##   https://CRAN.R-project.org/package=qpcR
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {qpcR: Modelling and Analysis of Real-Time PCR Data},
##     author = {Andrej-Nikolai Spiess},
##     year = {2018},
##     note = {R package version 1.4-1},
##     url = {https://CRAN.R-project.org/package=qpcR},
##   }
##
## ATTENTION: This citation information has been auto-generated from the
## package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
##
##
## To cite robustbase in publications use:
##
## To cite package 'robustbase' in publications use:
##
##   Martin Maechler, Peter Rousseeuw, Christophe Croux, Valentin Todorov,
##   Andreas Ruckstuhl, Matias Salibian-Barrera, Tobias Verbeke, Manuel
##   Koller, c("Eduardo", "L. T.") Conceicao and Maria Anna di Palma
```

```
##   (2020). robustbase: Basic Robust Statistics R package version 0.93-6.
##   URL http://CRAN.R-project.org/package=robustbase
##
## To cite the multivariate class/methods framework use:
##
##   Valentin Todorov, Peter Filzmoser (2009). An Object-Oriented
##   Framework for Robust Multivariate Analysis. Journal of Statistical
##   Software, 32(3), 1-47. URL http://www.jstatsoft.org/v32/i03/.
##
## To see these entries in BibTeX format, use 'print(<citation>,
## bibtex=TRUE)', 'toBibtex(.)', or set
## 'options(citation.bibtex.max=999)'.
##
##
## To cite package 'minpack.lm' in publications use:
##
##   Timur V. Elzhov, Katharine M. Mullen, Andrej-Nikolai Spiess and Ben
##   Bolker (2016). minpack.lm: R Interface to the Levenberg-Marquardt
##   Nonlinear Least-Squares Algorithm Found in MINPACK, Plus Support for
##   Bounds. R package version 1.2-1.
##   https://CRAN.R-project.org/package=minpack.lm
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {minpack.lm: R Interface to the Levenberg-Marquardt Nonlinear
Least-Squares
## Algorithm Found in MINPACK, Plus Support for Bounds},
##     author = {Timur V. Elzhov and Katharine M. Mullen and Andrej-Nikolai
Spiess and Ben Bolker},
##     year = {2016},
##     note = {R package version 1.2-1},
##     url = {https://CRAN.R-project.org/package=minpack.lm},
##   }
##
## ATTENTION: This citation information has been auto-generated from the
## package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
##
##
## To cite package 'plotrix' in publications use:
##
##   Lemon, J. (2006) Plotrix: a package in the red light district of R.
##   R-News, 6(4): 8-12.
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     year = {2006},
##     title = {Plotrix: a package in the red light district of R},
```

```
##     journal = {R-News},
##     volume = {6},
##     number = {4},
##     pages = {8-12},
##     author = {Lemon J},
##   }
##
##
## To cite package 'rlist' in publications use:
##
##   Kun Ren (2016). rlist: A Toolbox for Non-Tabular Data Manipulation. R
##   package version 0.4.6.1. https://CRAN.R-project.org/package=rlist
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {rlist: A Toolbox for Non-Tabular Data Manipulation},
##     author = {Kun Ren},
##     year = {2016},
##     note = {R package version 0.4.6.1},
##     url = {https://CRAN.R-project.org/package=rlist},
##   }
##
## ATTENTION: This citation information has been auto-generated from the
## package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
##
##
## To cite package 'gplots' in publications use:
##
##   Gregory R. Warnes, Ben Bolker, Lodewijk Bonebakker, Robert Gentleman,
##   Wolfgang Huber, Andy Liaw, Thomas Lumley, Martin Maechler, Arni
##   Magnusson, Steffen Moeller, Marc Schwartz and Bill Venables (2020).
##   gplots: Various R Programming Tools for Plotting Data. R package
##   version 3.0.4. https://CRAN.R-project.org/package=gplots
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {gplots: Various R Programming Tools for Plotting Data},
##     author = {Gregory R. Warnes and Ben Bolker and Lodewijk Bonebakker and
Robert Gentleman and Wolfgang Huber and Andy Liaw and Thomas Lumley and
Martin Maechler and Arni Magnusson and Steffen Moeller and Marc Schwartz and
Bill Venables},
##     year = {2020},
##     note = {R package version 3.0.4},
##     url = {https://CRAN.R-project.org/package=gplots},
##   }
##
##
```

```
## To cite package 'kmlShape' in publications use:
##
##   Christophe Genolini (2016). kmlShape: K-Means for Longitudinal Data
##   using Shape-Respecting Distance. R package version 0.9.5.
##   https://CRAN.R-project.org/package=kmlShape
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {kmlShape: K-Means for Longitudinal Data using Shape-
Respecting Distance},
##     author = {Christophe Genolini},
##     year = {2016},
##     note = {R package version 0.9.5},
##     url = {https://CRAN.R-project.org/package=kmlShape},
##   }
##
##
## To cite kml in publications use:
##
##   Christophe Genolini, Xavier Alacoque, Marianne Sentenac, Catherine
##   Arnaud (2015). kml and kml3d: R Packages to Cluster Longitudinal
##   Data. Journal of Statistical Software, 65(4), 1-34. URL
##   http://www.jstatsoft.org/v65/i04/.
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {{kml} and {kml3d}: {R} Packages to Cluster Longitudinal
Data},
##     author = {Christophe Genolini and Xavier Alacoque and Marianne
Sentenac and Catherine Arnaud},
##     journal = {Journal of Statistical Software},
##     year = {2015},
##     volume = {65},
##     number = {4},
##     pages = {1--34},
##     url = {http://www.jstatsoft.org/v65/i04/},
##   }
##
##
## To cite package 'longitudinalData' in publications use:
##
##   Christophe Genolini (2016). longitudinalData: Longitudinal Data. R
##   package version 2.4.1.
##   https://CRAN.R-project.org/package=longitudinalData
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
```

```
##     title = {longitudinalData: Longitudinal Data},
##     author = {Christophe Genolini},
##     year = {2016},
##     note = {R package version 2.4.1},
##     url = {https://CRAN.R-project.org/package=longitudinalData},
##   }
##
##
## To cite misc3d in publications use:
##
##   Dai Feng, Luke Tierney (2008). Computing and Displaying Isosurfaces
##   in R. Journal of Statistical Software 28(1). URL
##   http://www.jstatsoft.org/v28/i01/.
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {Computing and Displaying Isosurfaces in {R}},
##     author = {Dai Feng and Luke Tierney},
##     journal = {Journal of Statistical Software},
##     year = {2008},
##     volume = {28},
##     number = {1},
##     url = {http://www.jstatsoft.org/v28/i01/},
##   }
##
##
## To cite package 'rgl' in publications use:
##
##   Daniel Adler, Duncan Murdoch and others (2020). rgl: 3D Visualization
##   Using OpenGL. R package version 0.100.54.
##   https://CRAN.R-project.org/package=rgl
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {rgl: 3D Visualization Using OpenGL},
##     author = {Daniel Adler and Duncan Murdoch and others},
##     year = {2020},
##     note = {R package version 0.100.54},
##     url = {https://CRAN.R-project.org/package=rgl},
##   }
##
## ATTENTION: This citation information has been auto-generated from the
## package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
##
##
## To cite package 'clv' in publications use:
##
```

```
##   Lukasz Nieweglowski (2020). clv: Cluster Validation Techniques. R
##   package version 0.3-2.2. https://CRAN.R-project.org/package=clv
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {clv: Cluster Validation Techniques},
##     author = {Lukasz Nieweglowski},
##     year = {2020},
##     note = {R package version 0.3-2.2},
##     url = {https://CRAN.R-project.org/package=clv},
##   }
##
## ATTENTION: This citation information has been auto-generated from the
## package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
##
##
## To cite the R package 'cluster' in publications use:
##
##   Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., Hornik,
##   K.(2019).  cluster: Cluster Analysis Basics and Extensions. R package
##   version 2.1.0.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {cluster: Cluster Analysis Basics and Extensions},
##     author = {Martin Maechler and Peter Rousseeuw and Anja Struyf and Mia
Hubert and Kurt Hornik},
##     year = {2019},
##     note = {R package version 2.1.0 --- For new features, see the
'Changelog' file (in the package source)},
##   }
##
##
## To cite the class package in publications use:
##
##   Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with
##   S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0
##
## A BibTeX entry for LaTeX users is
##
##   @Book{,
##     title = {Modern Applied Statistics with S},
##     author = {W. N. Venables and B. D. Ripley},
##     publisher = {Springer},
##     edition = {Fourth},
##     address = {New York},
##     year = {2002},
```

```
##     note = {ISBN 0-387-95457-0},
##     url = {http://www.stats.ox.ac.uk/pub/MASS4},
##   }
##
##
## To cite package 'pvclust' in publications use:
##
##   Ryota Suzuki, Yoshikazu Terada and Hidetoshi Shimodaira (2019).
##   pvclust: Hierarchical Clustering with P-Values via Multiscale
##   Bootstrap Resampling. R package version 2.2-0.
##   https://CRAN.R-project.org/package=pvclust
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {pvclust: Hierarchical Clustering with P-Values via Multiscale
Bootstrap
## Resampling},
##     author = {Ryota Suzuki and Yoshikazu Terada and Hidetoshi Shimodaira},
##     year = {2019},
##     note = {R package version 2.2-0},
##     url = {https://CRAN.R-project.org/package=pvclust},
##   }
##
## ATTENTION: This citation information has been auto-generated from the
## package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
##
##
## To cite package 'factoextra' in publications use:
##
##   Alboukadel Kassambara and Fabian Mundt (2020). factoextra: Extract
##   and Visualize the Results of Multivariate Data Analyses. R package
##   version 1.0.7. https://CRAN.R-project.org/package=factoextra
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {factoextra: Extract and Visualize the Results of Multivariate
Data Analyses},
##     author = {Alboukadel Kassambara and Fabian Mundt},
##     year = {2020},
##     note = {R package version 1.0.7},
##     url = {https://CRAN.R-project.org/package=factoextra},
##   }
##
##
## To cite mosaic in publications, please use:
##
##   R. Pruim, D. T. Kaplan and N. J. Horton. The mosaic Package: Helping
```

```
##   Students to 'Think with Data' Using R (2017). The R Journal,
##   9(1):77-102.
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     author = {Randall Pruim and Daniel T Kaplan and Nicholas J Horton},
##     title = {The mosaic Package: Helping Students to 'Think with Data'
Using R},
##     journal = {The R Journal},
##     volume = {9},
##     number = {1},
##     pages = {77--102},
##     year = {2017},
##     url = {https://journal.r-project.org/archive/2017/RJ-2017-
024/index.html},
##   }
##
##
## To cite package 'Matrix' in publications use:
##
##   Douglas Bates and Martin Maechler (2019). Matrix: Sparse and Dense
##   Matrix Classes and Methods. R package version 1.2-18.
##   https://CRAN.R-project.org/package=Matrix
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {Matrix: Sparse and Dense Matrix Classes and Methods},
##     author = {Douglas Bates and Martin Maechler},
##     year = {2019},
##     note = {R package version 1.2-18},
##     url = {https://CRAN.R-project.org/package=Matrix},
##   }
##
##
## To cite package 'mosaicData' in publications use:
##
##   Randall Pruim, Daniel Kaplan and Nicholas Horton (2020). mosaicData:
##   Project MOSAIC Data Sets. R package version 0.18.0.
##   https://CRAN.R-project.org/package=mosaicData
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {mosaicData: Project MOSAIC Data Sets},
##     author = {Randall Pruim and Daniel Kaplan and Nicholas Horton},
##     year = {2020},
##     note = {R package version 0.18.0},
##     url = {https://CRAN.R-project.org/package=mosaicData},
```

```
##   }
##
## ATTENTION: This citation information has been auto-generated from the
## package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
##
##
## To cite package 'ggformula' in publications use:
##
##   Daniel Kaplan and Randall Pruim (2020). ggformula: Formula Interface
##   to the Grammar of Graphics. R package version 0.9.4.
##   https://CRAN.R-project.org/package=ggformula
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {ggformula: Formula Interface to the Grammar of Graphics},
##     author = {Daniel Kaplan and Randall Pruim},
##     year = {2020},
##     note = {R package version 0.9.4},
##     url = {https://CRAN.R-project.org/package=ggformula},
##   }
##
##
## To cite package 'ggstance' in publications use:
##
##   Lionel Henry, Hadley Wickham and Winston Chang (2020). ggstance:
##   Horizontal 'ggplot2' Components. R package version 0.3.4.
##   https://CRAN.R-project.org/package=ggstance
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {ggstance: Horizontal 'ggplot2' Components},
##     author = {Lionel Henry and Hadley Wickham and Winston Chang},
##     year = {2020},
##     note = {R package version 0.3.4},
##     url = {https://CRAN.R-project.org/package=ggstance},
##   }
##
##
## To cite the lattice package in publications use:
##
##   Sarkar, Deepayan (2008) Lattice: Multivariate Data Visualization with
##   R. Springer, New York. ISBN 978-0-387-75968-5
##
## A BibTeX entry for LaTeX users is
##
##   @Book{,
##     title = {Lattice: Multivariate Data Visualization with R},
```

```
##     author = {Deepayan Sarkar},
##     publisher = {Springer},
##     address = {New York},
##     year = {2008},
##     note = {ISBN 978-0-387-75968-5},
##     url = {http://lmdvr.r-forge.r-project.org},
##   }
##
##
## To cite fitdistrplus in publications use:
##
##   Marie Laure Delignette-Muller, Christophe Dutang (2015).
##   fitdistrplus: An R Package for Fitting Distributions. Journal of
##   Statistical Software, 64(4), 1-34. URL
##   http://www.jstatsoft.org/v64/i04/.
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {{fitdistrplus}: An {R} Package for Fitting Distributions},
##     author = {Marie Laure Delignette-Muller and Christophe Dutang},
##     journal = {Journal of Statistical Software},
##     year = {2015},
##     volume = {64},
##     number = {4},
##     pages = {1--34},
##     url = {http://www.jstatsoft.org/v64/i04/},
##   }
##
## Please cite both the package and R when using them for data analysis.
## See also 'citation()' for citing R.
##
##
## Therneau T (2020). _A Package for Survival Analysis in R_. R package
## version 3.2-3, <URL: https://CRAN.R-project.org/package=survival>.
##
## Terry M. Therneau, Patricia M. Grambsch (2000). _Modeling Survival
## Data: Extending the Cox Model_. Springer, New York. ISBN 0-387-98784-3.
##
## To see these entries in BibTeX format, use 'print(<citation>,
## bibtex=TRUE)', 'toBibtex(.)', or set
## 'options(citation.bibtex.max=999)'.
##
##
## To cite the MASS package in publications use:
##
##   Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with
##   S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0
##
## A BibTeX entry for LaTeX users is
```

```
##
##   @Book{,
##     title = {Modern Applied Statistics with S},
##     author = {W. N. Venables and B. D. Ripley},
##     publisher = {Springer},
##     edition = {Fourth},
##     address = {New York},
##     year = {2002},
##     note = {ISBN 0-387-95457-0},
##     url = {http://www.stats.ox.ac.uk/pub/MASS4/},
##   }
##
##
## To cite package 'devtools' in publications use:
##
##   Hadley Wickham, Jim Hester and Winston Chang (2020). devtools: Tools
##   to Make Developing R Packages Easier. R package version 2.3.1.
##   https://CRAN.R-project.org/package=devtools
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {devtools: Tools to Make Developing R Packages Easier},
##     author = {Hadley Wickham and Jim Hester and Winston Chang},
##     year = {2020},
##     note = {R package version 2.3.1},
##     url = {https://CRAN.R-project.org/package=devtools},
##   }
##
##
## To cite package 'usethis' in publications use:
##
##   Hadley Wickham and Jennifer Bryan (2020). usethis: Automate Package
##   and Project Setup. R package version 1.6.1.
##   https://CRAN.R-project.org/package=usethis
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {usethis: Automate Package and Project Setup},
##     author = {Hadley Wickham and Jennifer Bryan},
##     year = {2020},
##     note = {R package version 1.6.1},
##     url = {https://CRAN.R-project.org/package=usethis},
##   }
##
##
## To cite package 'data.table' in publications use:
##
##   Matt Dowle and Arun Srinivasan (2020). data.table: Extension of
```

```
##   `data.frame`. R package version 1.13.0.
##   https://CRAN.R-project.org/package=data.table
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {data.table: Extension of `data.frame`},
##     author = {Matt Dowle and Arun Srinivasan},
##     year = {2020},
##     note = {R package version 1.13.0},
##     url = {https://CRAN.R-project.org/package=data.table},
##   }
##
##
## To cite zoo in publications use:
##
##   Achim Zeileis and Gabor Grothendieck (2005). zoo: S3 Infrastructure
##   for Regular and Irregular Time Series. Journal of Statistical
##   Software, 14(6), 1-27. doi:10.18637/jss.v014.i06
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {zoo: S3 Infrastructure for Regular and Irregular Time
Series},
##     author = {Achim Zeileis and Gabor Grothendieck},
##     journal = {Journal of Statistical Software},
##     year = {2005},
##     volume = {14},
##     number = {6},
##     pages = {1--27},
##     doi = {10.18637/jss.v014.i06},
##   }
##
##
## To cite package 'magrittr' in publications use:
##
##   Stefan Milton Bache and Hadley Wickham (2014). magrittr: A
##   Forward-Pipe Operator for R. R package version 1.5.
##   https://CRAN.R-project.org/package=magrittr
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {magrittr: A Forward-Pipe Operator for R},
##     author = {Stefan Milton Bache and Hadley Wickham},
##     year = {2014},
##     note = {R package version 1.5},
##     url = {https://CRAN.R-project.org/package=magrittr},
##   }
```

```
##
## ATTENTION: This citation information has been auto-generated from the
## package DESCRIPTION file and may need manual editing, see
## 'help("citation")'.
##
##
## To cite package 'janitor' in publications use:
##
##   Sam Firke (2020). janitor: Simple Tools for Examining and Cleaning
##   Dirty Data. R package version 2.0.1.
##   https://CRAN.R-project.org/package=janitor
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {janitor: Simple Tools for Examining and Cleaning Dirty Data},
##     author = {Sam Firke},
##     year = {2020},
##     note = {R package version 2.0.1},
##     url = {https://CRAN.R-project.org/package=janitor},
##   }
##
##
## To cite package 'forcats' in publications use:
##
##   Hadley Wickham (2020). forcats: Tools for Working with Categorical
##   Variables (Factors). R package version 0.5.0.
##   https://CRAN.R-project.org/package=forcats
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {forcats: Tools for Working with Categorical Variables
(Factors)},
##     author = {Hadley Wickham},
##     year = {2020},
##     note = {R package version 0.5.0},
##     url = {https://CRAN.R-project.org/package=forcats},
##   }
##
##
## To cite package 'stringr' in publications use:
##
##   Hadley Wickham (2019). stringr: Simple, Consistent Wrappers for
##   Common String Operations. R package version 1.4.0.
##   https://CRAN.R-project.org/package=stringr
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
```

```
##     title = {stringr: Simple, Consistent Wrappers for Common String
Operations},
##     author = {Hadley Wickham},
##     year = {2019},
##     note = {R package version 1.4.0},
##     url = {https://CRAN.R-project.org/package=stringr},
##   }
##
##
## To cite package 'dplyr' in publications use:
##
##   Hadley Wickham, Romain François, Lionel Henry and Kirill Müller
##   (2020). dplyr: A Grammar of Data Manipulation. R package version
##   1.0.2. https://CRAN.R-project.org/package=dplyr
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {dplyr: A Grammar of Data Manipulation},
##     author = {Hadley Wickham and Romain François and Lionel {
##               Henry} and Kirill Müller},
##     year = {2020},
##     note = {R package version 1.0.2},
##     url = {https://CRAN.R-project.org/package=dplyr},
##   }
##
##
## To cite package 'purrr' in publications use:
##
##   Lionel Henry and Hadley Wickham (2020). purrr: Functional Programming
##   Tools. R package version 0.3.4.
##   https://CRAN.R-project.org/package=purrr
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {purrr: Functional Programming Tools},
##     author = {Lionel Henry and Hadley Wickham},
##     year = {2020},
##     note = {R package version 0.3.4},
##     url = {https://CRAN.R-project.org/package=purrr},
##   }
##
##
## To cite package 'readr' in publications use:
##
##   Hadley Wickham, Jim Hester and Romain Francois (2018). readr: Read
##   Rectangular Text Data. R package version 1.3.1.
##   https://CRAN.R-project.org/package=readr
##
```

```
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {readr: Read Rectangular Text Data},
##     author = {Hadley Wickham and Jim Hester and Romain Francois},
##     year = {2018},
##     note = {R package version 1.3.1},
##     url = {https://CRAN.R-project.org/package=readr},
##   }
##
##
## To cite package 'tidyr' in publications use:
##
##   Hadley Wickham (2020). tidyr: Tidy Messy Data. R package version
##   1.1.2. https://CRAN.R-project.org/package=tidyr
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {tidyr: Tidy Messy Data},
##     author = {Hadley Wickham},
##     year = {2020},
##     note = {R package version 1.1.2},
##     url = {https://CRAN.R-project.org/package=tidyr},
##   }
##
##
## To cite package 'tibble' in publications use:
##
##   Kirill Müller and Hadley Wickham (2020). tibble: Simple Data Frames.
##   R package version 3.0.3. https://CRAN.R-project.org/package=tibble
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {tibble: Simple Data Frames},
##     author = {Kirill Müller and Hadley Wickham},
##     year = {2020},
##     note = {R package version 3.0.3},
##     url = {https://CRAN.R-project.org/package=tibble},
##   }
##
##
## To cite ggplot2 in publications, please use:
##
##   H. Wickham. ggplot2: Elegant Graphics for Data Analysis.
##   Springer-Verlag New York, 2016.
##
## A BibTeX entry for LaTeX users is
##
```

```
##   @Book{,
##     author = {Hadley Wickham},
##     title = {ggplot2: Elegant Graphics for Data Analysis},
##     publisher = {Springer-Verlag New York},
##     year = {2016},
##     isbn = {978-3-319-24277-4},
##     url = {https://ggplot2.tidyverse.org},
##   }
##
##
##   Wickham et al., (2019). Welcome to the tidyverse. Journal of Open
##   Source Software, 4(43), 1686, https://doi.org/10.21105/joss.01686
##
## A BibTeX entry for LaTeX users is
##
##   @Article{,
##     title = {Welcome to the {tidyverse}},
##     author = {Hadley Wickham and Mara Averick and Jennifer Bryan and
Winston Chang and Lucy D'Agostino McGowan and Romain François and Garrett
Grolemund and Alex Hayes and Lionel Henry and Jim Hester and Max Kuhn and
Thomas Lin Pedersen and Evan Miller and Stephan Milton Bache and Kirill
Müller and Jeroen Ooms and David Robinson and Dana Paige Seidel and Vitalie
Spinu and Kohske Takahashi and Davis Vaughan and Claus Wilke and Kara Woo and
Hiroaki Yutani},
##     year = {2019},
##     journal = {Journal of Open Source Software},
##     volume = {4},
##     number = {43},
##     pages = {1686},
##     doi = {10.21105/joss.01686},
##   }
##
##
## To cite the 'knitr' package in publications use:
##
##   Yihui Xie (2020). knitr: A General-Purpose Package for Dynamic Report
##   Generation in R. R package version 1.29.
##
##   Yihui Xie (2015) Dynamic Documents with R and knitr. 2nd edition.
##   Chapman and Hall/CRC. ISBN 978-1498716963
##
##   Yihui Xie (2014) knitr: A Comprehensive Tool for Reproducible
##   Research in R. In Victoria Stodden, Friedrich Leisch and Roger D.
##   Peng, editors, Implementing Reproducible Computational Research.
##   Chapman and Hall/CRC. ISBN 978-1466561595
##
## To see these entries in BibTeX format, use 'print(<citation>,
## bibtex=TRUE)', 'toBibtex(.)', or set
## 'options(citation.bibtex.max=999)'.
```

References

Achim Zeileis and Gabor Grothendieck (2005). zoo: S3 Infrastructure for Regular and Irregular Time Series. Journal of Statistical Software, 14(6), 1-27. doi:10.18637/jss.v014.i06

Alboukadel Kassambara (2020). ggpubr: 'ggplot2' Based Publication Ready Plots. R package version 0.4.0. https://CRAN.R-project.org/package=ggpubr

Alboukadel Kassambara and Fabian Mundt (2020). factoextra: Extract and Visualize the Results of Multivariate Data Analyses. R package version 1.0.7. https://CRAN.R-project.org/package=factoextra

Alexander Kowarik, Matthias Templ (2016). Imputation with the R Package VIM. Journal of Statistical Software, 74(7), 1-16. doi:10.18637/jss.v074.i07

Andrej-Nikolai Spiess (2018). qpcR: Modelling and Analysis of Real-Time PCR Data. R package version 1.4-1. https://CRAN.R-project.org/package=qpcR

Baptiste Auguie (2017). gridExtra: Miscellaneous Functions for "Grid" Graphics. R package version 2.3. https://CRAN.R-project.org/package=gridExtra

Benjamin Guiastrennec (2020). ggplus: Set of additional functions for ggplot2. R package version 0.1. https://github.com/guiastrennec/ggplus

Christophe Genolini (2016). kmlShape: K-Means for Longitudinal Data using Shape-Respecting Distance. R package version 0.9.5. https://CRAN.R-project.org/package=kmlShape

Christophe Genolini (2016). longitudinalData: Longitudinal Data. R package version 2.4.1. https://CRAN.R-project.org/package=longitudinalData

Christophe Genolini, Xavier Alacoque, Marianne Sentenac, Catherine Arnaud (2015). kml and kml3d: R Packages to Cluster Longitudinal Data. Journal of Statistical Software, 65(4), 1-34. URL http://www.jstatsoft.org/v65/i04/.

Dai Feng, Luke Tierney (2008). Computing and Displaying Isosurfaces in R. Journal of Statistical Software 28(1). URL http://www.jstatsoft.org/v28/i01/.

Daniel Adler, Duncan Murdoch and others (2020). rgl: 3D Visualization Using OpenGL. R package version 0.100.54. https://CRAN.R-project.org/package=rgl

Daniel Kaplan and Randall Pruim (2020). ggformula: Formula Interface to the Grammar of Graphics. R package version 0.9.4. https://CRAN.R-project.org/package=ggformula

Douglas Bates and Martin Maechler (2019). Matrix: Sparse and Dense Matrix Classes and Methods. R package version 1.2-18. https://CRAN.R-project.org/package=Matrix

Friedrich Leisch and Bettina Gruen. Extending standard cluster algorithms to allow for group constraints. In Alfredo Rizziand Maurizio Vichi, editors, Compstat 2006-Proceedings in Computational Statistics, pages 885-892. Physica Verlag, Heidelberg, Germany, 2006.

Friedrich Leisch. A Toolbox for K-Centroids Cluster Analysis. Computational Statistics and Data Analysis, 51 (2), 526-544, 2006.

Friedrich Leisch. Neighborhood graphs, stripes and shadow plots for cluster visualization. Statistics and Computing, 20(4), 457-469, 2010.

Gregory R. Warnes, Ben Bolker, Lodewijk Bonebakker, Robert Gentleman, Wolfgang Huber, Andy Liaw, Thomas Lumley, Martin Maechler, Arni Magnusson, Steffen Moeller, Marc Schwartz and Bill Venables (2020). gplots: Various R Programming Tools for Plotting Data. R package version 3.0.4. https://CRAN.R-project.org/package=gplots

H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016.

Hadley Wickham (2007). Reshaping Data with the reshape Package. Journal of Statistical Software, 21(12), 1-20. URL http://www.jstatsoft.org/v21/i12/.

Hadley Wickham (2011). The Split-Apply-Combine Strategy for Data Analysis. Journal of Statistical Software, 40(1), 1-29. URL http://www.jstatsoft.org/v40/i01/.

Hadley Wickham (2019). stringr: Simple, Consistent Wrappers for Common String Operations. R package version 1.4.0. https://CRAN.R-project.org/package=stringr

Hadley Wickham (2020). forcats: Tools for Working with Categorical Variables (Factors). R package version 0.5.0. https://CRAN.R-project.org/package=forcats

Hadley Wickham (2020). tidyr: Tidy Messy Data. R package version 1.1.2. https://CRAN.R-project.org/package=tidyr

Hadley Wickham and Jennifer Bryan (2019). readxl: Read Excel Files. R package version 1.3.1.

Hadley Wickham and Jennifer Bryan (2020). usethis: Automate Package and Project Setup. R package version 1.6.1. https://CRAN.R-project.org/package=usethis

Hadley Wickham, Jim Hester and Romain Francois (2018). readr: Read Rectangular Text Data. R package version 1.3.1. https://CRAN.R-project.org/package=readr

Hadley Wickham, Jim Hester and Winston Chang (2020). devtools: Tools to Make Developing R Packages Easier. R package version 2.3.1. https://CRAN.R-project.org/package=devtools

Hadley Wickham, Romain François, Lionel Henry and Kirill Müller (2020). dplyr: A Grammar of Data Manipulation. R package version 1.0.2. https://CRAN.R-project.org/package=dplyr

https://CRAN.R-project.org/package=clusteval
https://CRAN.R-project.org/package=clv
https://CRAN.R-project.org/package=readxl
https://CRAN.R-project.org/package=survival>.

John A. Ramey (2012). clusteval: Evaluation of Clustering Algorithms. R package version 0.1.

Kirill Müller and Hadley Wickham (2020). tibble: Simple Data Frames. R package version 3.0.3. https://CRAN.R-project.org/package=tibble

Kun Ren (2016). rlist: A Toolbox for Non-Tabular Data Manipulation. R package version 0.4.6.1. https://CRAN.R-project.org/package=rlist

Lemon, J. (2006) Plotrix: a package in the red light district of R. R-News, 6(4): 8-12.

Liis Kolberg and Uku Raudvere (2020). gprofiler2: Interface to the 'g:Profiler' Toolset. R package version 0.2.0. https://CRAN.R-project.org/package=gprofiler2

Lionel Henry and Hadley Wickham (2020). purrr: Functional Programming Tools. R package version 0.3.4. https://CRAN.R-project.org/package=purrr

Lionel Henry, Hadley Wickham and Winston Chang (2020). ggstance: Horizontal 'ggplot2' Components. R package version 0.3.4. https://CRAN.R-project.org/package=ggstance

Lukasz Nieweglowski (2020). clv: Cluster Validation Techniques. R package version 0.3-2.2. Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., Hornik, K.(2019). cluster: Cluster Analysis Basics and Extensions. R

Marie Laure Delignette-Muller, Christophe Dutang (2015). fitdistrplus: An R Package for Fitting Distributions. Journal of Statistical Software, 64(4), 1-34. URL http://www.jstatsoft.org/v64/i04/.

Martin Maechler, Peter Rousseeuw, Christophe Croux, Valentin Todorov, Andreas Ruckstuhl, Matias Salibian-Barrera, Tobias Verbeke, Manuel Koller, c("Eduardo", "L. T.") Conceicao and Maria Anna di Palma (2020). robustbase: Basic Robust Statistics R package version 0.93-6. URL http://CRAN.R-project.org/package=robustbase

Matt Dowle and Arun Srinivasan (2020). data.table: Extension of `data.frame`. R package version 1.13.0. https://CRAN.R-project.org/package=data.table

Moving beyond P values: Everyday data analysis with estimation plots. (2019) Joses Ho, Tayfun Tumkaya, Sameer Aryal, Hyungwon Choi, Adam Claridge-Chang. Nature Methods 2019, 1548-710. doi: https://doi.org/10.1038/s41592-019-0470-3 package version 2.1.0.

Philipp Schauberger and Alexander Walker (2020). openxlsx: Read, Write and Edit xlsx Files. R package version 4.1.5. https://CRAN.R-project.org/package=openxlsx

R Core Team (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

R. Pruim, D. T. Kaplan and N. J. Horton. The mosaic Package: Helping Students to 'Think with Data' Using R (2017). The R Journal, 9(1):77-102.

Randall Pruim, Daniel Kaplan and Nicholas Horton (2020). mosaicData: Project MOSAIC Data Sets. R package version 0.18.0. https://CRAN.R-project.org/package=mosaicData

Ryan M. Hope (2013). Rmisc: Rmisc: Ryan Miscellaneous. R package version 1.5. https://CRAN.R-project.org/package=Rmisc

Ryota Suzuki, Yoshikazu Terada and Hidetoshi Shimodaira (2019). pvclust: Hierarchical Clustering with P-Values via Multiscale Bootstrap Resampling. R package version 2.2-0. https://CRAN.R-project.org/package=pvclust

Sam Firke (2020). janitor: Simple Tools for Examining and Cleaning Dirty Data. R package version 2.0.1. https://CRAN.R-project.org/package=janitor

Sara Dolnicar, Bettina Gruen, and Friedrich Leisch. Market Segmentation Analysis --- Understanding It, Doing It, and Making It Useful. Management for Professionals. Springer, Singapore, 2018. doi:10.1007/978-981-10-8818-6

Sarkar, Deepayan (2008) Lattice: Multivariate Data Visualization with R. Springer, New York. ISBN 978-0-387-75968-5

Sebastien Le, Julie Josse, Francois Husson (2008). FactoMineR: An R Package for Multivariate Analysis. Journal of Statistical Software, 25(1), 1-18. 10.18637/jss.v025.i01

Stauffer R, Mayr GJ, Dabernig M, Zeileis A (2009). “Somewhere over the Rainbow: How to Make Effective Use of Colors in Meteorological Visualizations.” _Bulletin of the American Meteorological Society_, *96*(2), 203-216. doi:10.1175/BAMS-D-13-00155.1 (URL: https://doi.org/10.1175/BAMS-D-13-00155.1).

Stef van Buuren, Karin Groothuis-Oudshoorn (2011). mice: Multivariate Imputation by Chained Equations in R. Journal of Statistical Software, 45(3), 1-67. URL https://www.jstatsoft.org/v45/i03/.

Stefan Milton Bache and Hadley Wickham (2014). magrittr: A Forward-Pipe Operator for R. R package version 1.5. https://CRAN.R-project.org/package=magrittr

Terry M. Therneau, Patricia M. Grambsch (2000). _Modeling Survival Data: Extending the Cox Model_. Springer, New York. ISBN 0-387-98784-3.

Theresa Scharl and Friedrich Leisch. The Stochastic QT-clust Algorithm: Evaluation of Stability and Variance on Time-course Microarray Data. In Alfredo Rizzi and Maurizio Vichi, editors, Compstat 2006-Proceedings in Computational Statistics, pages 1015-1022. Physica Verlag, Heidelberg, Germany, 2006.

Therneau T (2020). _A Package for Survival Analysis in R_. R package version 3.2-3, <URL: https://CRAN.R-project.org/package=survival>.

Timur V. Elzhov, Katharine M. Mullen, Andrej-Nikolai Spiess and Ben Bolker (2016). minpack.lm: R Interface to the Levenberg-Marquardt Nonlinear Least-Squares Algorithm Found in MINPACK, Plus Support for Bounds. R package version 1.2-1. https://CRAN.R-project.org/package=minpack.lm

Torsten Hothorn, Friedrich Leisch and Achim Zeileis (2020). modeltools: Tools and Classes for Statistical Models. R package version 0.2-23. https://CRAN.R-project.org/package=modeltools

Valentin Todorov, Peter Filzmoser (2009). An Object-Oriented Framework for Robust Multivariate Analysis. Journal of Statistical Software, 32(3), 1-47. URL http://www.jstatsoft.org/v32/i03/.

Venables, W. N. & Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth Edition. Springer, New York. ISBN 0-387-95457-0

Wickham et al., (2019). Welcome to the tidyverse. Journal of Open Source Software, 4(43), 1686, https://doi.org/10.21105/joss.01686

Yihui Xie (2020). knitr: A General-Purpose Package for Dynamic Report Generation in R. R package version 1.29.

Yihui Xie (2015) Dynamic Documents with R and knitr. 2nd edition. Chapman and Hall/CRC. ISBN 978-1498716963

Yihui Xie (2014) knitr: A Comprehensive Tool for Reproducible Research in R. In Victoria Stodden, Friedrich Leisch and Roger D. Peng, editors, Implementing Reproducible Computational Research. Chapman and Hall/CRC. ISBN 978-1466561595

Yumi Kondo, Matias Salibian-Barrera, Ruben Zamar (2016). RSKC: An R Package for a Robust and Sparse K-Means Clustering Algorithm. Journal of Statistical Software, 72(5), 1-26. doi:10.18637/jss.v072.i05

Zeileis A, Fisher JC, Hornik K, Ihaka R, McWhite CD, Murrell P, Stauffer R, Wilke CO (2019). “colorspace: A Toolbox for Manipulating and Assessing Colors and Palettes.” arXiv 1903.06490, arXiv.org E-Print Archive. <URL: http://arxiv.org/abs/1903.06490>.

Zeileis A, Hornik K, Murrell P (2009). “Escaping RGBland: Selecting Colors for Statistical Graphics.” _Computational Statistics \& Data Analysis_, *53*(9), 3259-3270. doi: 10.1016/j.csda.2008.11.033 (URL: https://doi.org/10.1016/j.csda.2008.11.033).