Visualization in bioinformatics

Abhijit Dasgupta, PhD

Networks

Visualizing a proteomic network

We read a dataset that contains the network relationships between different proteins

```
library(ggnetwork)
datf <- rio::import('data/string_graph.txt')
head(datf)
```

```
    node1  node2 node1_string_id node2_string_id node1_external_id
1   CXCR3   CCR7         1855969         1843829   ENSP00000362795
2   ITGA4    EED         1858446         1845338   ENSP00000380227
3    SMC3  CENPK         1854200         1843648   ENSP00000354720
4 HNRNPA1 LUC7L3         1852510         1843556   ENSP00000341826
5    SMC2    RB1         1847012         1845924   ENSP00000286398
6   RBBP4  CENPK         1855919         1843648   ENSP00000362592
  node2_external_id neighborhood fusion cooccurence homology coexpression
1   ENSP00000246657            0      0           0    0.847        0.000
2   ENSP00000263360            0      0           0    0.000        0.000
3   ENSP00000242872            0      0           0    0.000        0.000
4   ENSP00000240304            0      0           0    0.000        0.000
5   ENSP00000267163            0      0           0    0.000        0.136
6   ENSP00000242872            0      0           0    0.000        0.000
  experimental knowledge textmining combined_score
1        0.000       0.9      0.878          0.913
2        0.566       0.0      0.312          0.688
3        0.000       0.9      0.081          0.904
```

Visualizing a proteomic network

The **igraph** package allows the creation of network graphs.

However, here, we're only using it for data ingestion

```
pacman::p_load(igraph)
grs <- graph_from_data_frame(datf[,c('node1','node2')],
                             directed = F)
grs
```

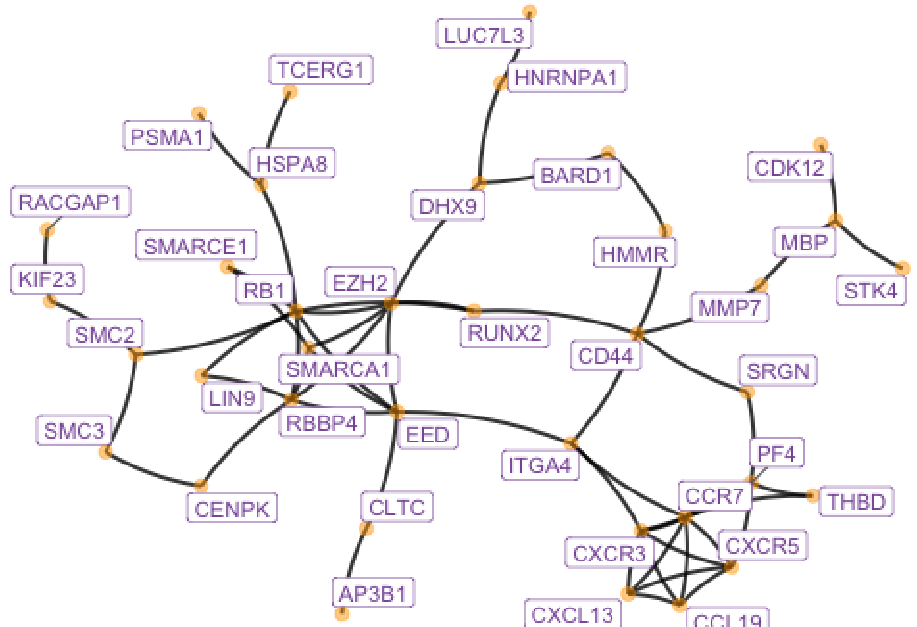
```
  IGRAPH 473d753 UN-- 37 58 --
  + attr: name (v/c)
  + edges from 473d753 (vertex names):
   [1] CXCR3   --CCR7     ITGA4   --EED      SMC3    --CENPK    HNRNPA1--LUC7L3
   [5] SMC2    --RB1      RBBP4   --CENPK    CXCR5   --CXCL13   CD44    --RUNX2
   [9] CXCR5   --PF4      PF4     --THBD     SMARCA1--EZH2      HMMR    --BARD1
  [13] MBP     --MMP7     CCL19   --CCR7     RBBP4   --EZH2     RUNX2   --RB1
  [17] RB1     --HSPA8    DHX9    --BARD1    CXCL13  --CCR7     SMC2    --KIF23
  [21] CD44    --HMMR     ITGA4   --CD44     RB1     --SMARCE1  ITGA4   --CCR7
  [25] MBP     --STK4     RBBP4   --LIN9     RB1     --EED      CXCR5   --CCR7
  [29] PSMA1   --HSPA8    RBBP4   --SMARCA1  CXCR3   --ITGA4    MBP     --CDK12
  + ... omitted several edges
```

We see that this object holds the different connections.

Visualizing a proteomic network

We can then transform this data into `ggplot`-friendly data, to use `ggplot` for the plotting

```
library(intergraph)
ggdf <- ggnetwork(asNetwork(grs),
                  layout='fruchtermanreingold')
ggplot(ggdf, aes(x = x, y = y,
                 xend = xend, yend = yend)) +
  geom_edges(color = "black",
             curvature = 0.1,
             size = 0.95, alpha = 0.8)+
  geom_nodes(aes(x = x, y = y),
             size = 3,
             alpha = 0.5,
             color = "orange") +
  geom_nodelabel_repel(aes(label = vertex.names),
                       size=4, color="#8856a7") +
  theme_blank() + theme(legend.position = "none")
```



Composing different genomic data into tracks

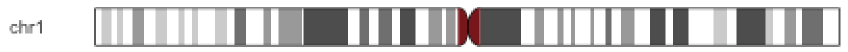
The `ggbio` package

The **ggbio** package has several functions that allow graphical representations of different genomic entities.

You will see a lot of use of `autoplot`, which is a software technique to create default visualizations based on the type of entry.

An ideogram

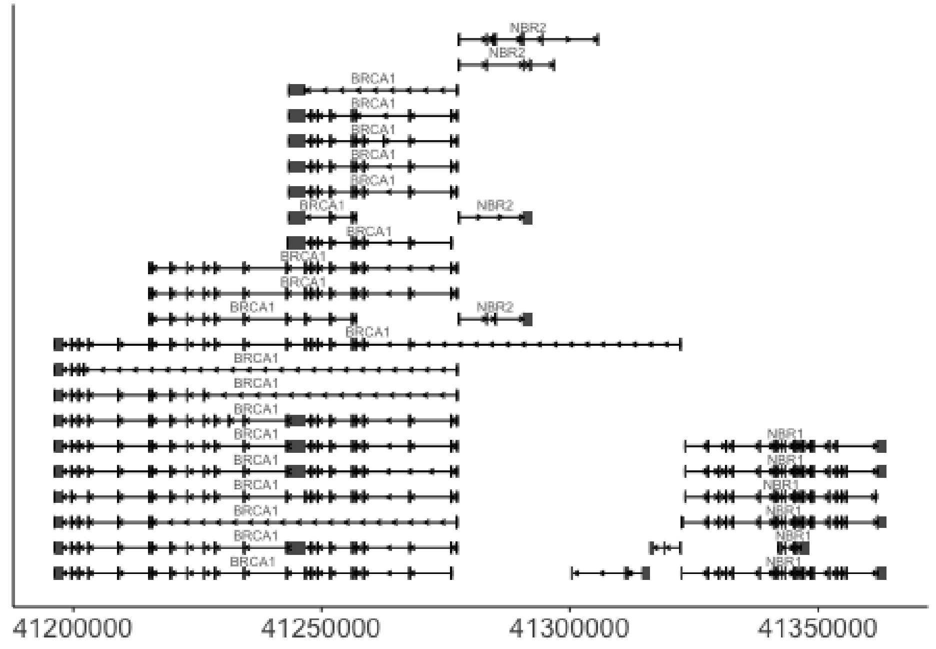
```
library(ggbio) # p_install('ggbio', try.bioconductor=
p.ideo <- Ideogram(genome = 'hg19')
p.ideo
```



The ggbio package

Visualizing the gene model

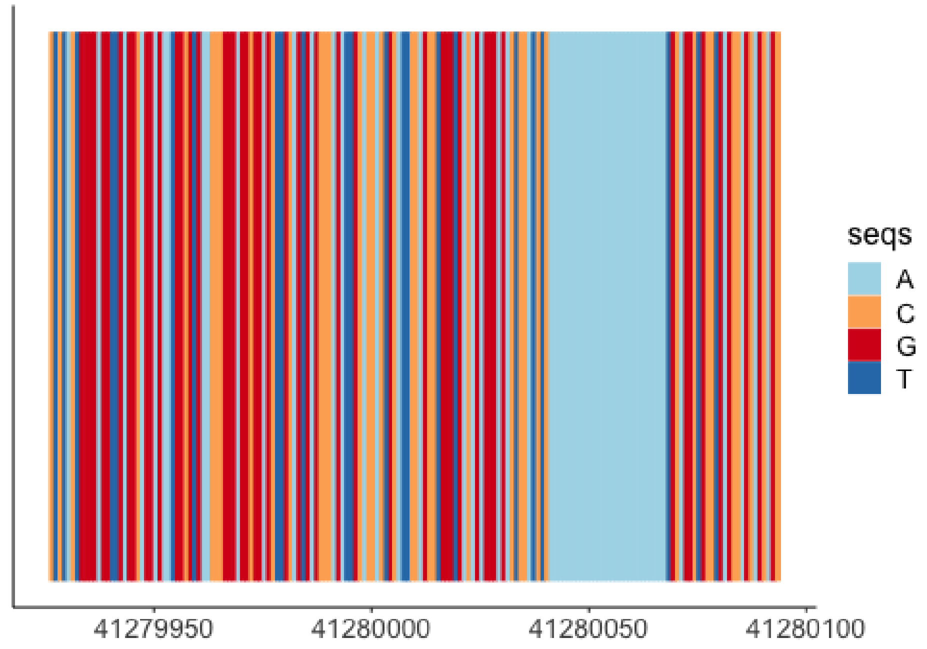
```r
pacman::p_load(Homo.sapiens)
data(genesymbol, package='biovizBase')
wh <- genesymbol[c('BRCA1','NBR1')]
wh <- range(wh, ignore.strand=T)
p.txdb <- autoplot(Homo.sapiens, which = wh)
p.txdb
```



The ggbio package

A reference track

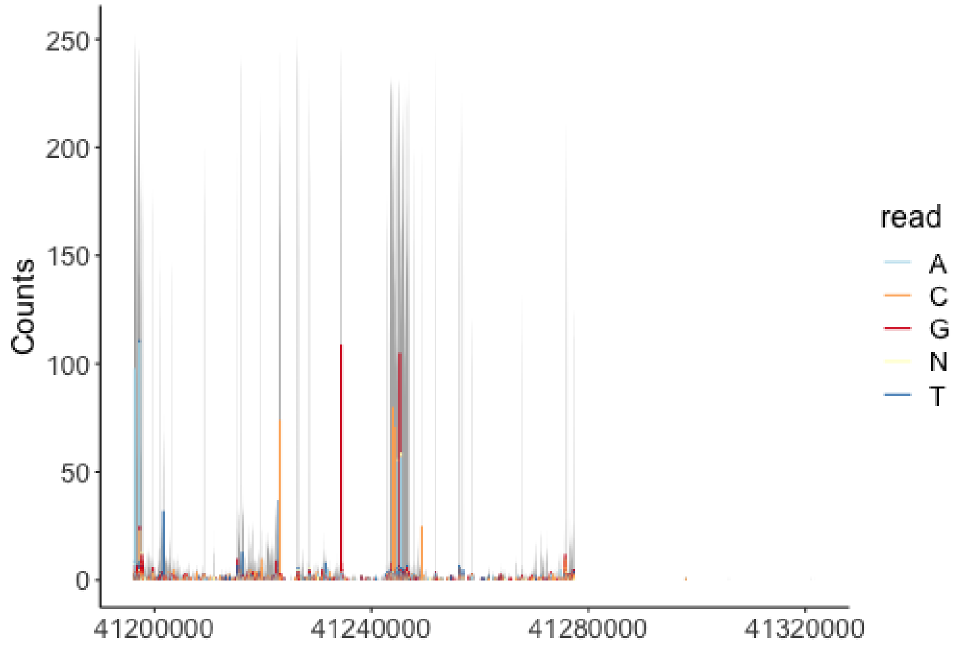
```r
library(BSgenome.Hsapiens.UCSC.hg19)
bg <- BSgenome.Hsapiens.UCSC.hg19
p.bg <- autoplot(bg, which=wh)
p.bg + zoom(1/1000)
```



The ggbio package

An alignment track with mismatch proportions

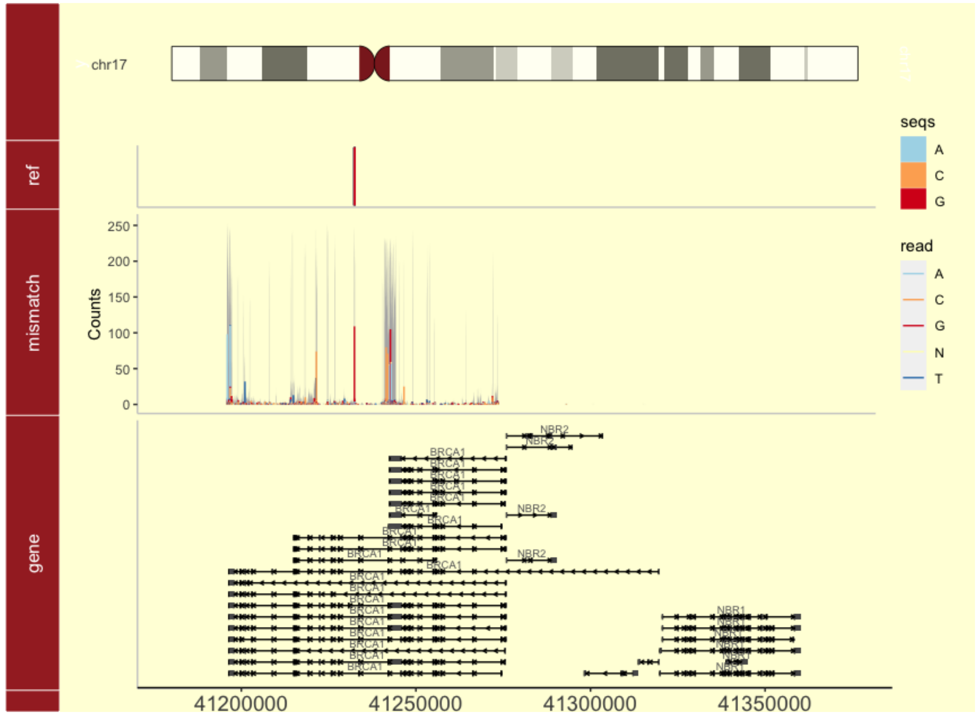
```
library(BSgenome.Hsapiens.UCSC.hg19)
fl.bam <- system.file("extdata", "wg-brca1.sorted.bam
wh <- keepSeqlevels(wh, "chr17")
bg <- BSgenome.Hsapiens.UCSC.hg19
p.mis <- autoplot(fl.bam, bsgenome = bg, which = wh,
p.mis
```



The ggbio package

Putting it into tracks

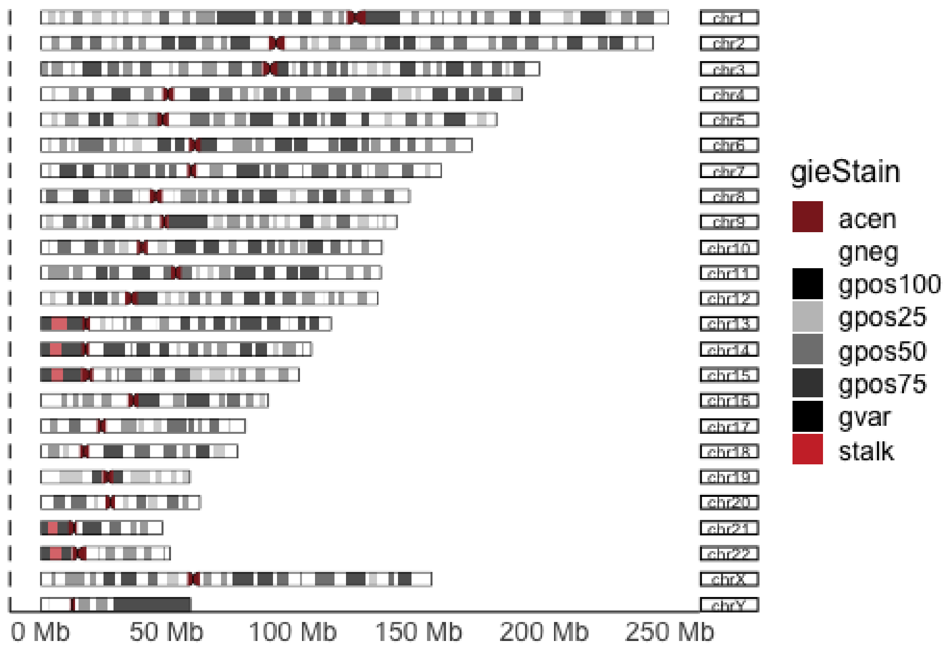
```
pacman::p_load(GenomicRanges)
gr17 <- GRanges("chr17", IRanges(41234415, 41234569))
tks <-  tracks(p.ideo,
                   ref=p.bg,
                   mismatch=p.mis,
                   gene=p.txdb,
                   heights=c(2,1,3,4))+
  xlim(gr17) +
  theme_tracks_sunset()
print(tks)
```



The ggbio package

A karyogram

```r
data(ideoCyto, package = "biovizBase")
autoplot(ideoCyto$hg19, layout = "karyogram",
         cytobands = TRUE)
```



P-values and Manhattan plots

A very simple example

```r
library(tidyverse)
clinical <- rio::import('data/BreastCancer_Clinical.xlsx') %>% janitor::clean_names()
proteome <- rio::import('data/BreastCancer_Expression.xlsx') %>% janitor::clean_names()
final_data <- clinical %>%
    inner_join(proteome, by = c('complete_tcga_id' = 'tcga_id')) %>%
    dplyr::filter(gender == 'FEMALE') %>%
    dplyr::select(complete_tcga_id, age_at_initial_pathologic_diagnosis, er_status, starts_with("np"))
head(final_data)
```

```
  complete_tcga_id age_at_initial_pathologic_diagnosis er_status  np_958782  np_958785  np_958786  np_000436
1     TCGA-A2-A0CM                                  40  Negative  0.6834035  0.6944241  0.6980976  0.6870771
2     TCGA-BH-A18Q                                  56  Negative  0.1953407  0.2154129  0.2154129  0.2053768
3     TCGA-A7-A0CE                                  57  Negative -1.1231731 -1.1231731 -1.1168605 -1.1294857
4     TCGA-D8-A142                                  74  Negative  0.5385958  0.5422105  0.5422105  0.5349810
5     TCGA-AO-A0J6                                  61  Negative  0.8311317  0.8565398  0.8565398  0.8367780
6     TCGA-A2-A0YM                                  67  Negative  0.6558497  0.6581426  0.6558497  0.6558497
   np_958781  np_958780  np_958783  np_958784  np_112598  np_001611
1  0.6870771  0.6980976  0.6980976  0.6980976 -2.6521501 -0.9843733
2  0.2154129  0.2154129  0.2154129  0.2154129 -1.0357599 -0.5172257
3 -1.1294857 -1.1200168 -1.1231731 -1.1231731  2.2445844 -2.5750648
4  0.5422105  0.5422105  0.5422105  0.5422105 -0.1482049  0.2674902
5  0.8650092  0.8565398  0.8508936  0.8508936 -0.9671961  2.8383705
6  0.6512639  0.6581426  0.6558497  0.6558497 -1.9695337  1.3070365
```

A very simple example

```r
results <- final_data %>%
    summarise_at(vars(starts_with('np')),
                 ~wilcox.test(. ~ er_status)$p.value)
results
```

```
   np_958782 np_958785 np_958786 np_000436 np_958781 np_958780 np_958783 np_958784 np_112598    np_001611
1 0.6988415 0.6910103 0.6832121 0.6910103 0.6832121 0.6910103 0.6910103 0.6832121 0.9957714 0.0001218627
```

`.` is the placeholder for what's specified inside the `vars()`.

This isn't in the right format for me to plot

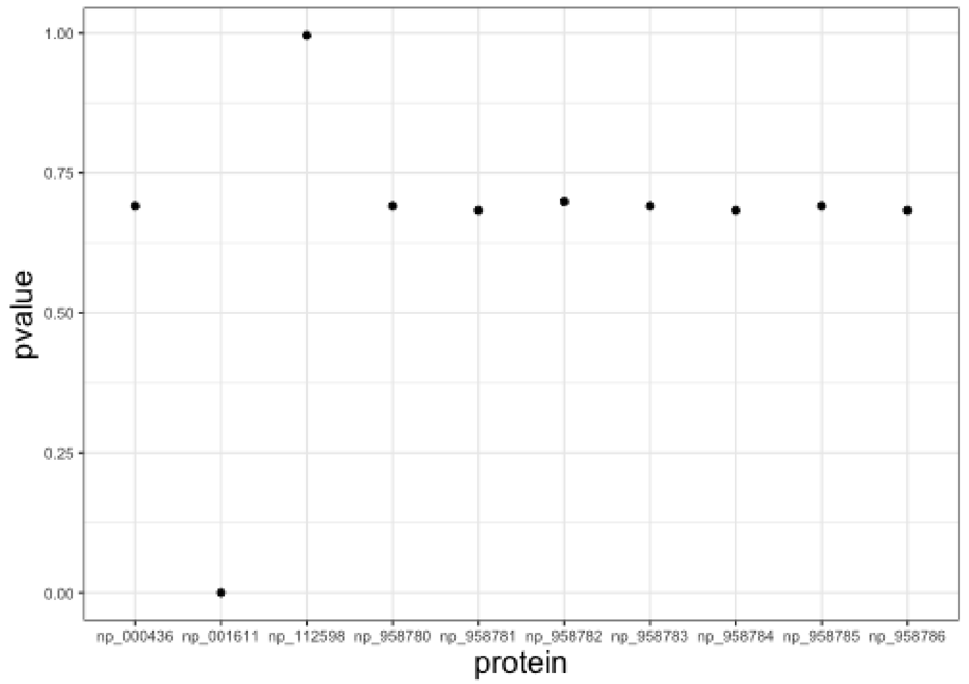
A very simple example

```
results %>% tidyr::pivot_longer(cols=everything(),
                                names_to='protein',
                                values_to='pvalue')
```

```
  # A tibble: 10 x 2
     protein     pvalue
     <chr>        <dbl>
   1 np_958782 0.699
   2 np_958785 0.691
   3 np_958786 0.683
   4 np_000436 0.691
   5 np_958781 0.683
   6 np_958780 0.691
   7 np_958783 0.691
   8 np_958784 0.683
   9 np_112598 0.996
  10 np_001611 0.000122
```

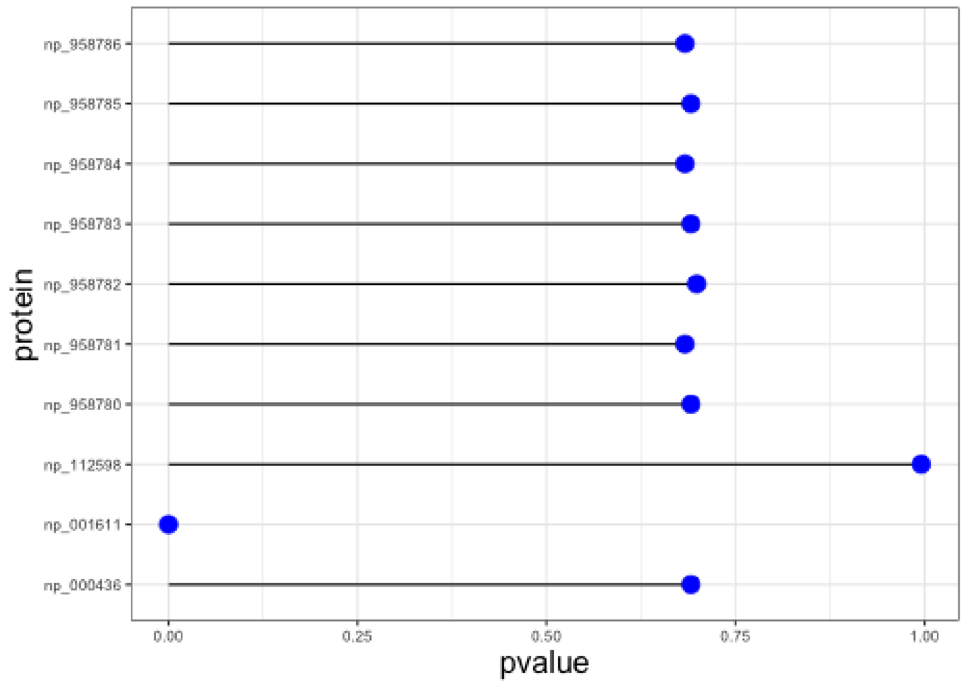
A very simple example

```r
theme_439 <- theme_bw() +
    theme(axis.title = element_text(size=16),
          axis.text = element_text(size=8))


results %>% pivot_longer(
  cols=everything(),
  names_to='protein',
  values_to='pvalue') %>%
  ggplot(aes(x = protein, y = pvalue)) +
  geom_point() +
  theme_439
```



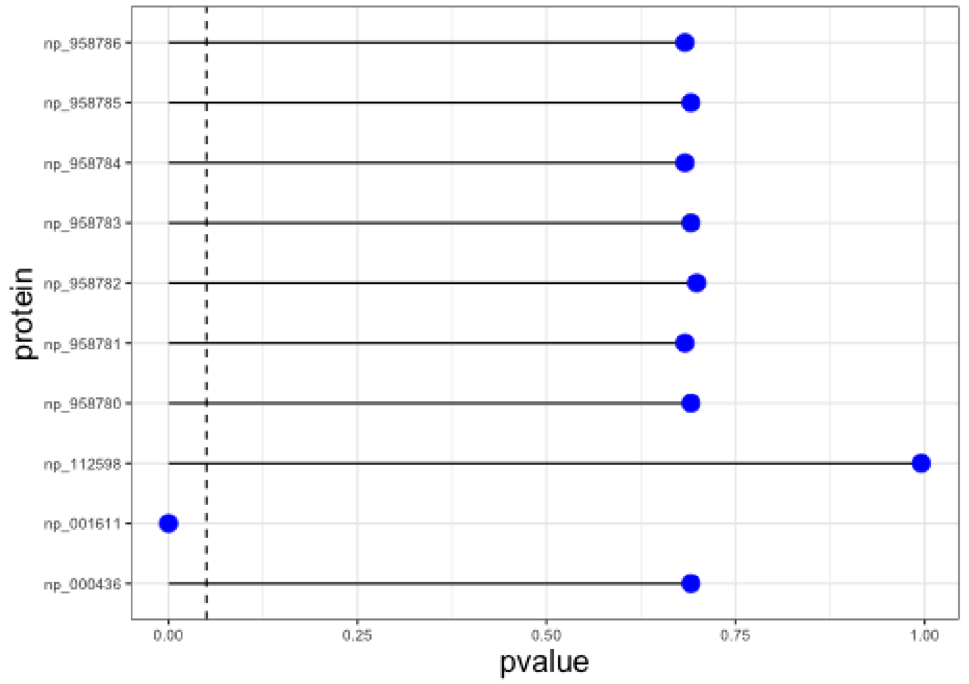
A very simple example

```
pacman::p_load('ggalt')
results %>% pivot_longer(
  cols=everything(),
  names_to = 'protein',
  values_to = 'pvalue') %>%
  ggplot(aes(x = protein, y = pvalue)) +
  geom_point() +
  geom_lollipop(point.colour='blue', point.size=4)+
  coord_flip()+
  theme_439
```



A very simple example

```
results %>% pivot_longer(
  cols=everything(),
  names_to = 'protein',
  values_to = 'pvalue') %>%
  ggplot(aes(x = protein, y = pvalue)) +
      geom_point() +
      geom_lollipop(point.colour='blue', point.size=4
      geom_hline(yintercept = 0.05, linetype=2)+
      coord_flip() +
      theme_439
```

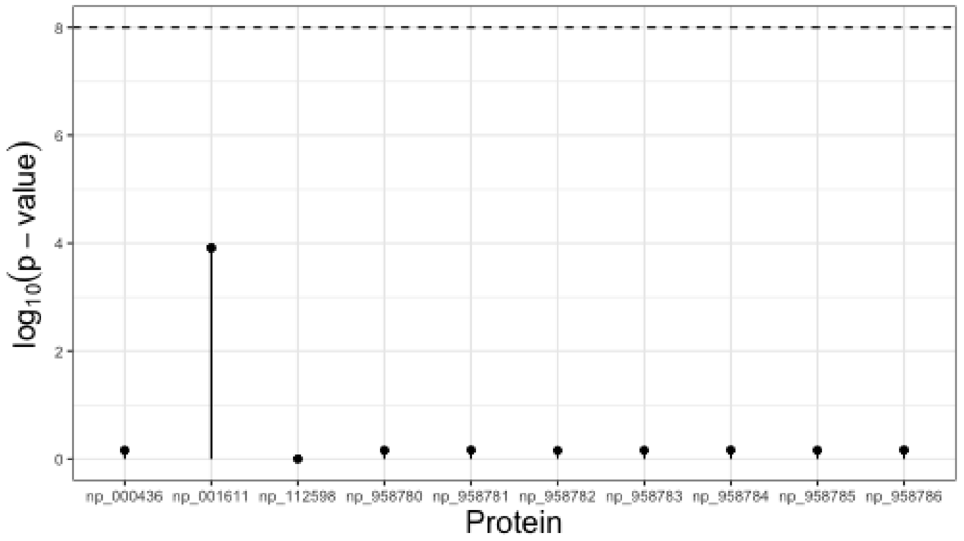


Manhattan plot

A Manhattan plot is used to visualize a set of p-values from unit-based tests

It plots the negative log p-value at each unit

```r
results %>% pivot_longer(
  cols=everything(),
  names_to = 'protein',
  values_to = 'pvalue') %>%
  ggplot(aes(x = protein, y = -log10(pvalue))) +
    geom_point() +
    geom_lollipop() +
    geom_hline(yintercept = 8, linetype=2)+
    labs(x = 'Protein',
         y = expression(log[10](p-value))) +
    theme_439
```



Manhattan plot

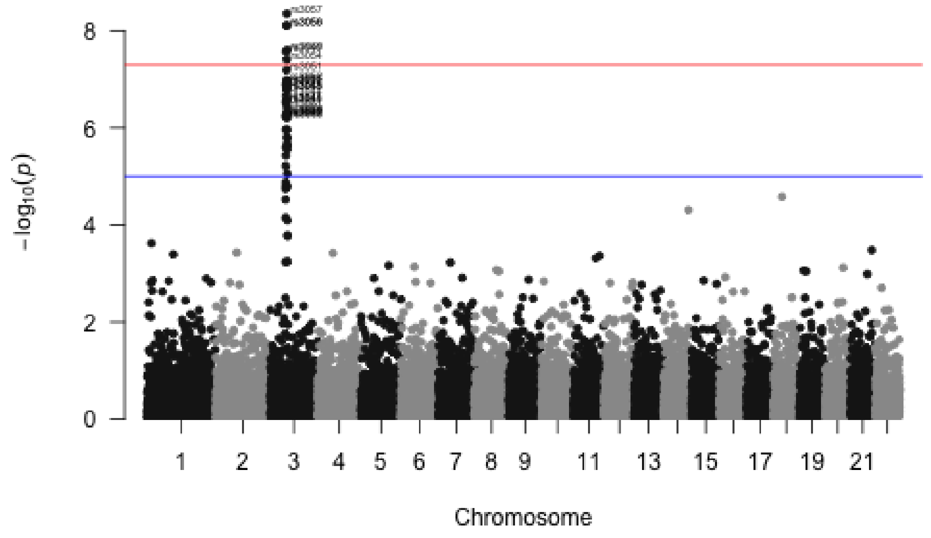
There is a specialized package for doing Manhattan plots and quantile plots for GWAS data

This package is meant to work with PLINK output, but the function is generic

```
library(qqman)
manhattan(gwasResults)
```

Manhattan plot

```r
library(qqman)
manhattan(gwasResults,
          annotatePval = 1e-6,
          annotateTop=F)
```



Heatmaps

Let us count the ways

There are several ways of doing heatmaps in R:

- https://jokergoo.github.io/ComplexHeatmap-reference/book/
- http://sebastianraschka.com/Articles/heatmaps_in_r.html
- https://plot.ly/r/heatmaps/
- http://moderndata.plot.ly/interactive-heat-maps-for-r/
- http://www.siliconcreek.net/r/simple-heatmap-in-r-with-ggplot2
- https://rud.is/b/2016/02/14/making-faceted-heatmaps-with-ggplot2/

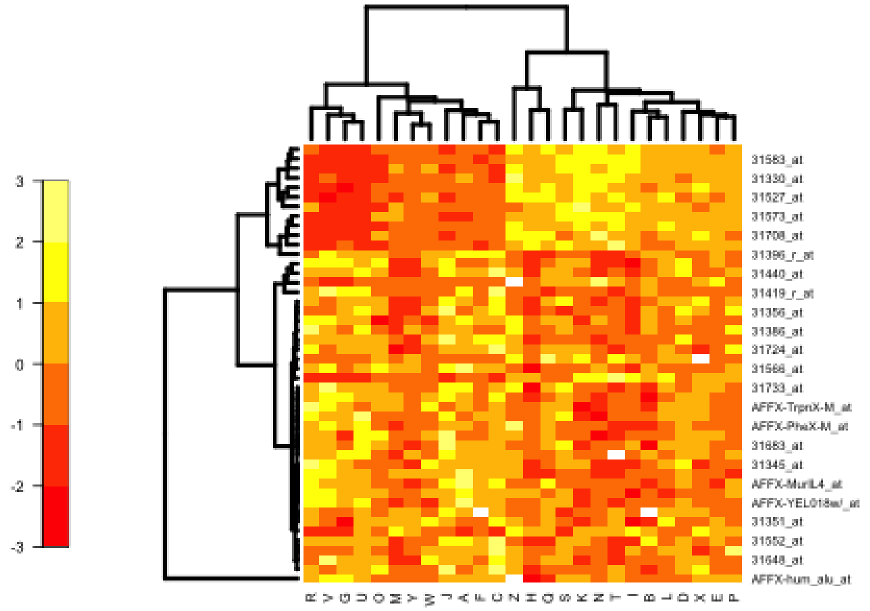
Some example data

```r
library(Biobase)
#data(sample.ExpressionSet)
exdat <-  readRDS('data/exprset.rds')
library(limma)
design1 <- model.matrix(~type, data=pData(exdat))
lm1 <- lmFit(exprs(exdat), design1)
lm1 <- eBayes(lm1) # compute linear model for each probeset
geneID <- rownames(topTable(lm1, coef = 2, number = 100,
                            adjust.method = 'none',
                            p.value = 0.05))
exdat2 <- exdat[geneID,] # Keep features with p-values < 0.05

head(exdat2)
```

```
  ExpressionSet (storageMode: lockedEnvironment)
  assayData: 1 features, 26 samples
    element names: exprs, se.exprs
  protocolData: none
  phenoData
    sampleNames: A B ... Z (26 total)
    varLabels: sex type score
    varMetadata: labelDescription
  featureData: none
  experimentData: use 'experimentData(object)'
  Annotation: hgu95av2
```

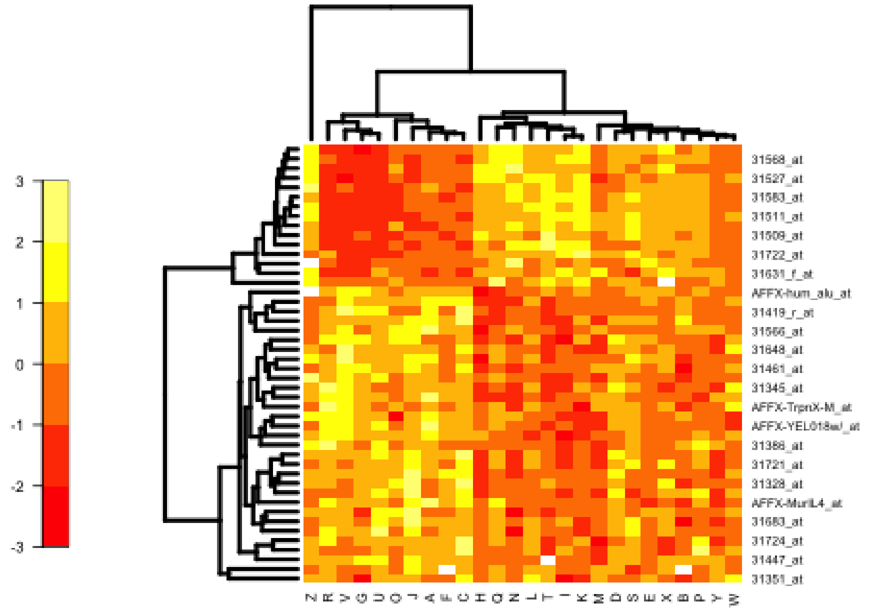
Using Heatplus

```
# BiocManager::install('Heatplus')
library(Heatplus)
reg1 <- regHeatmap(exprs(exdat2), legend=2, col=heat.
                   breaks=-3:3)
plot(reg1)
```



Using Heatplus

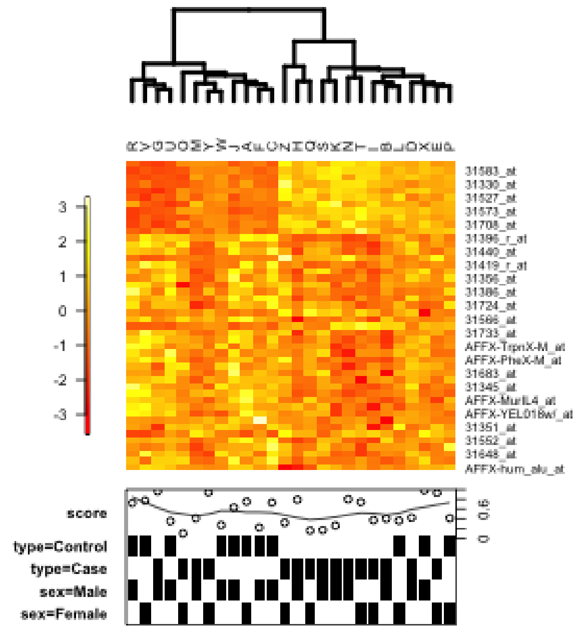
```
corrdist <- function(x) as.dist(1-cor(t(x)))
hclust.avl <- function(x) hclust(x, method='average')
reg2 <- regHeatmap(exprs(exdat2), legend=2,
                   col=heat.colors,
                   breaks=-3:3,
                   dendrogram =
                       list(clustfun=hclust.avl,
                            distfun=corrdist))
plot(reg2)
```



Using Heatplus

Adding annotations

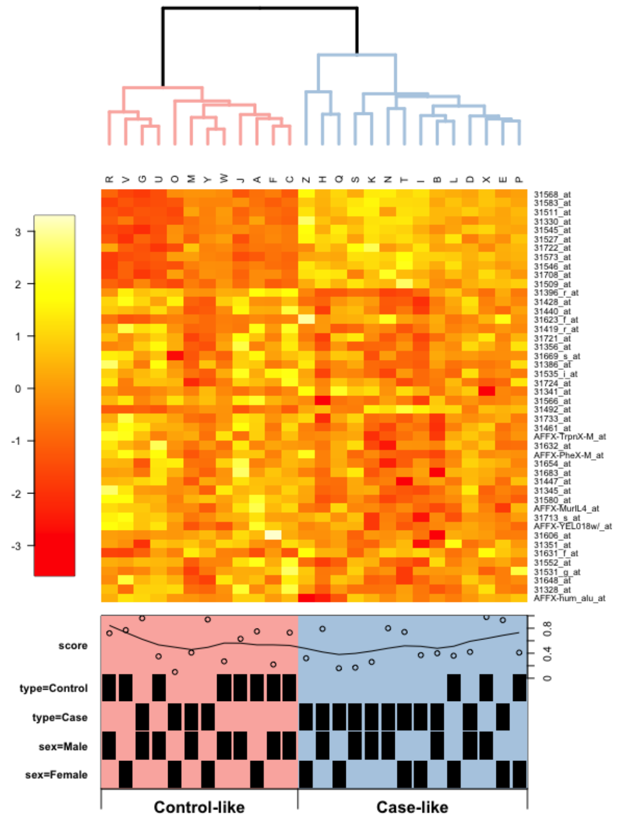
```
ann1 <- annHeatmap(exprs(exdat2),
                   ann=pData(exdat2),
                   col = heat.colors)
plot(ann1)
```



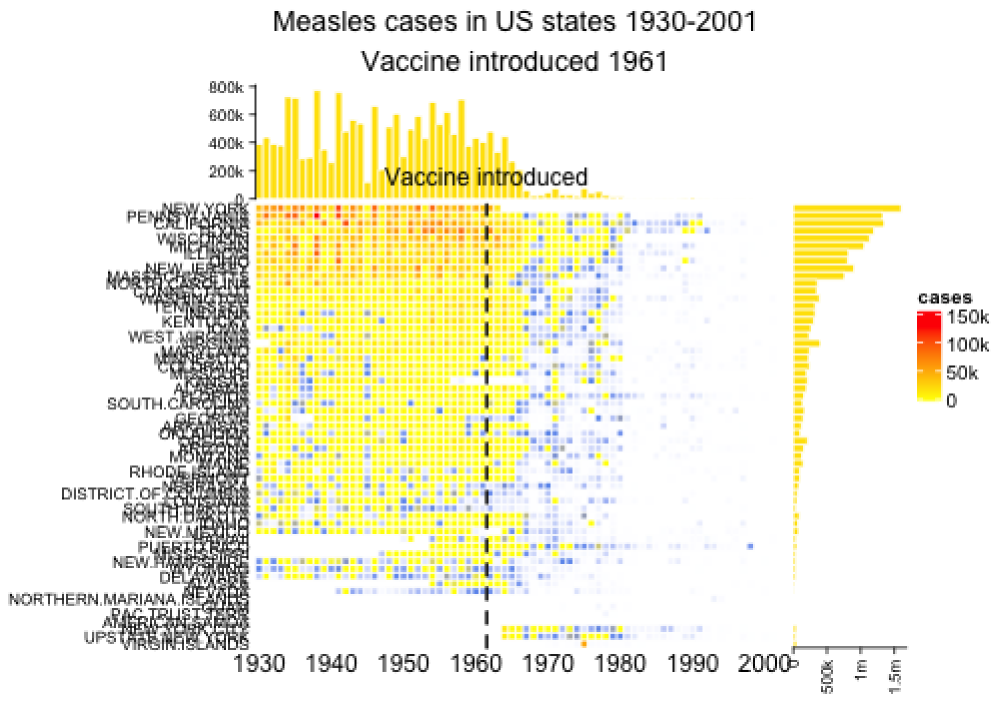
Using Heatplus

Adding annotations

```
ann2 <- annHeatmap(exprs(exdat2),
                   ann=pData(exdat2),
                   col = heat.colors,
        cluster =
            list(cuth=7500,
                 label=c('Control-like','Case-like')))
plot(ann2)
```



Using ComplexHeatmap

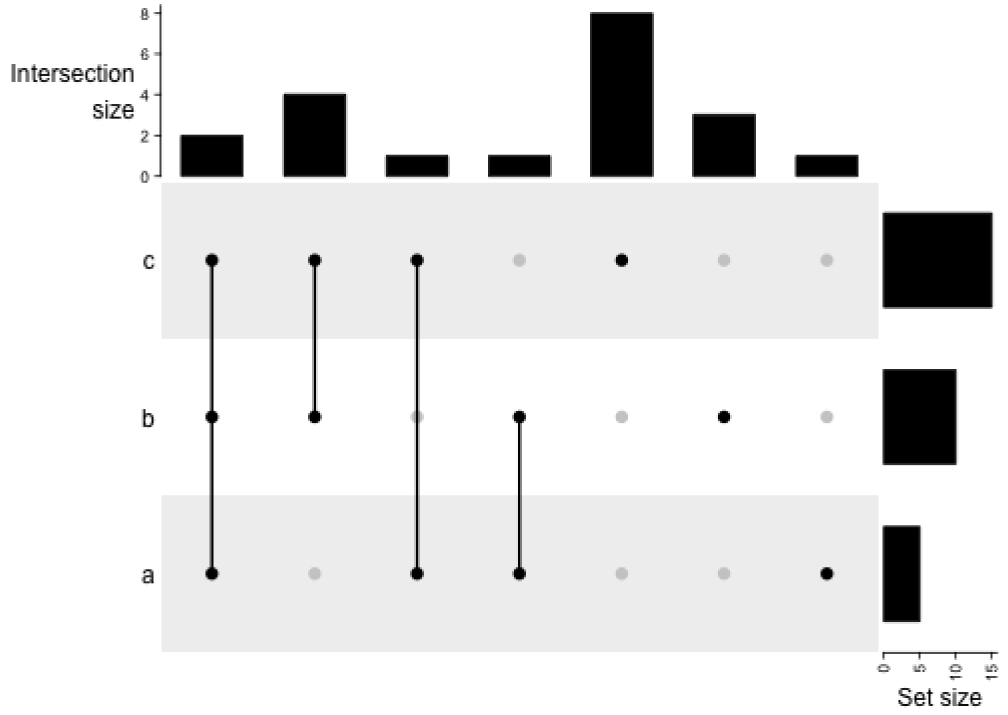


Source code here

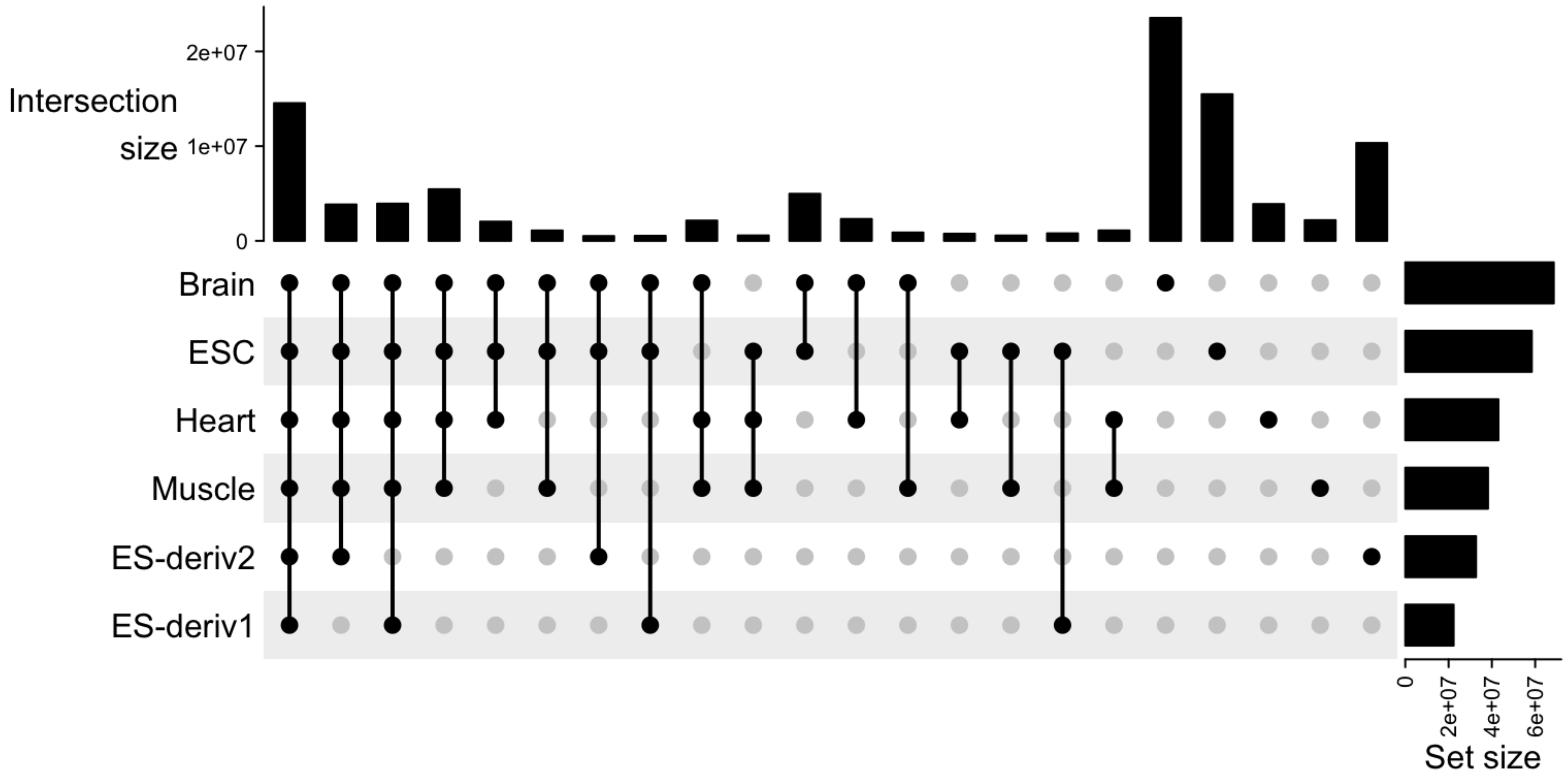
UpSet plots

UpSet plots are nice visualizations for looking at commonalities (complex intersections) between sets of objects.

We used UpSet plots to look at missing value patterns

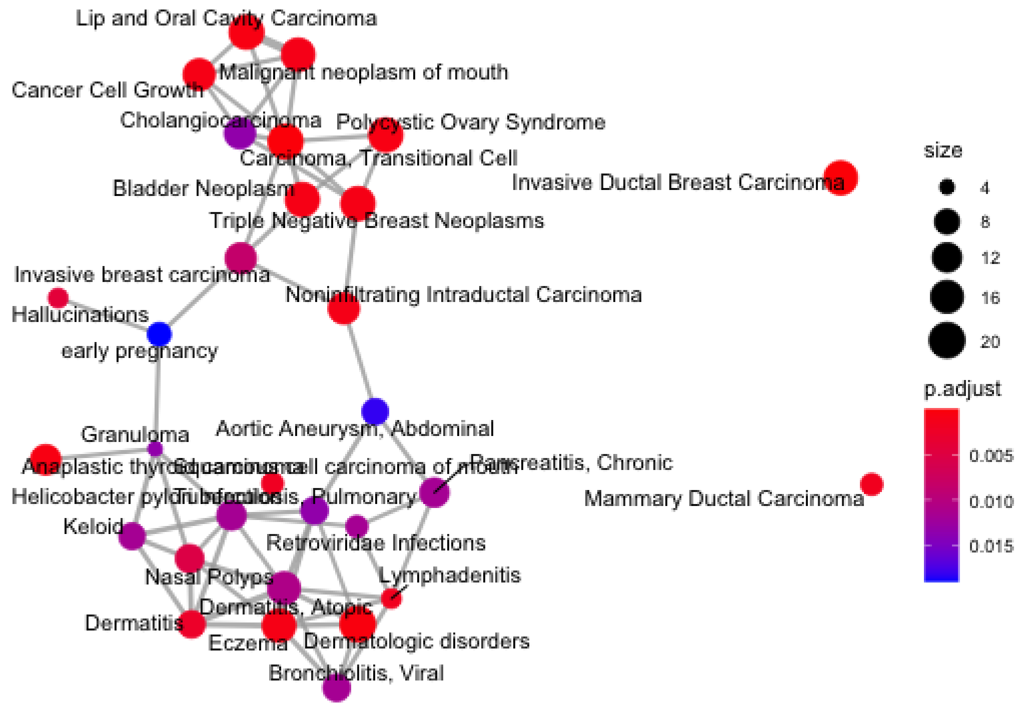


UpSet plots


Intersection size
2e+07
1e+07
0
Brain
ESC
Heart
Muscle
ES-deriv2
ES-deriv1
0
2e+07
4e+07
6e+07
Set size

clusterProfiler

Enrichment network based on GSEA



Playing with Seurat

Example data

```r
library(Seurat)
# pbmc.data <- Read10X(data.dir='data/hg19/')
# pbmc <- CreateSeuratObject(counts = pbmc.data, project='pbmc3k', min.cells=3, min.features=200)
pbmc <- readRDS('data/pbmc.rds')
pbmc
```

```
  An object of class Seurat
  13714 features across 2700 samples within 1 assay
  Active assay: RNA (13714 features, 0 variable features)
```

```r
names(pbmc)
```

```
  [1] "RNA"
```

```r
slotNames(pbmc)
```

```
   [1] "assays"       "meta.data"    "active.assay" "active.ident" "graphs"       "neighbors"    "reductions"
   [8] "project.name" "misc"         "version"      "commands"     "tools"
```

Adding QC metrics and plotting

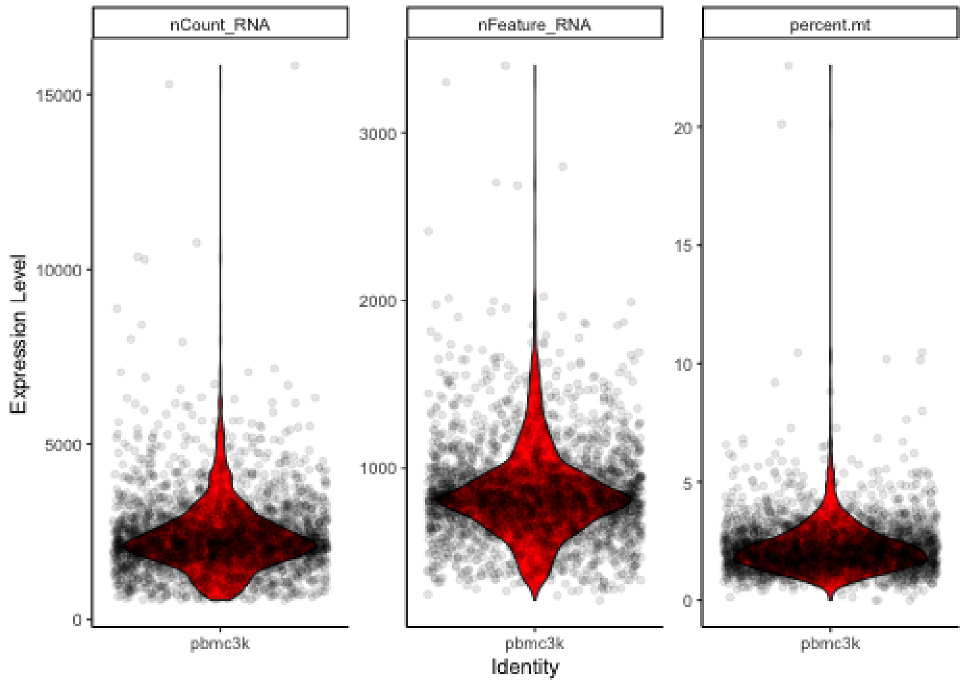
We'll calculate mitochondrial QC metrics (percentage counts originating from mitochondrial genes)

```r
pbmc[['percent.mt']] <- PercentageFeatureSet(pbmc, pattern = '^MT-')
head(pbmc@meta.data)
```

```
               orig.ident nCountRNA nFeature_RNA percent.mt
AAACATACAACCAC     pbmc3k      2419          779  3.0177759
AAACATTGAGCTAC     pbmc3k      4903         1352  3.7935958
AAACATTGATCAGC     pbmc3k      3147         1129  0.8897363
AAACCGTGCTTCCG     pbmc3k      2639          960  1.7430845
AAACCGTGTATGCG     pbmc3k       980          521  1.2244898
AAACGCACTGGTAC     pbmc3k      2163          781  1.6643551
```

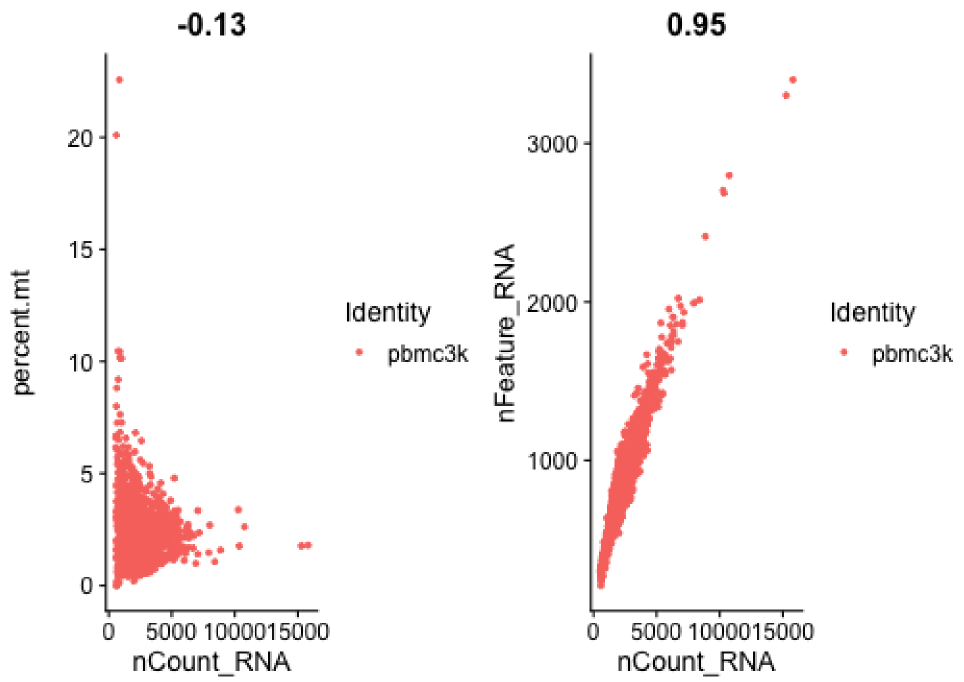
Visualizing metrics

```r
# plt <- VlnPlot(object = pbmc,
#   features = c('nFeature_RNA',
#                'nCount_RNA',
#                'percent.mt'))

plot_data <- pbmc@meta.data %>%
  tidyr::gather(variable, value, -orig.ident)

ggplot(plot_data, aes(orig.ident, value)) +
  geom_violin(fill = 'red') +
  geom_jitter(width=0.5, alpha = 0.1) +
  facet_wrap(~variable, nrow = 1,
             scales = 'free_y') +
  labs(x = 'Identity',y = 'Expression Level') +
  theme_classic()
```



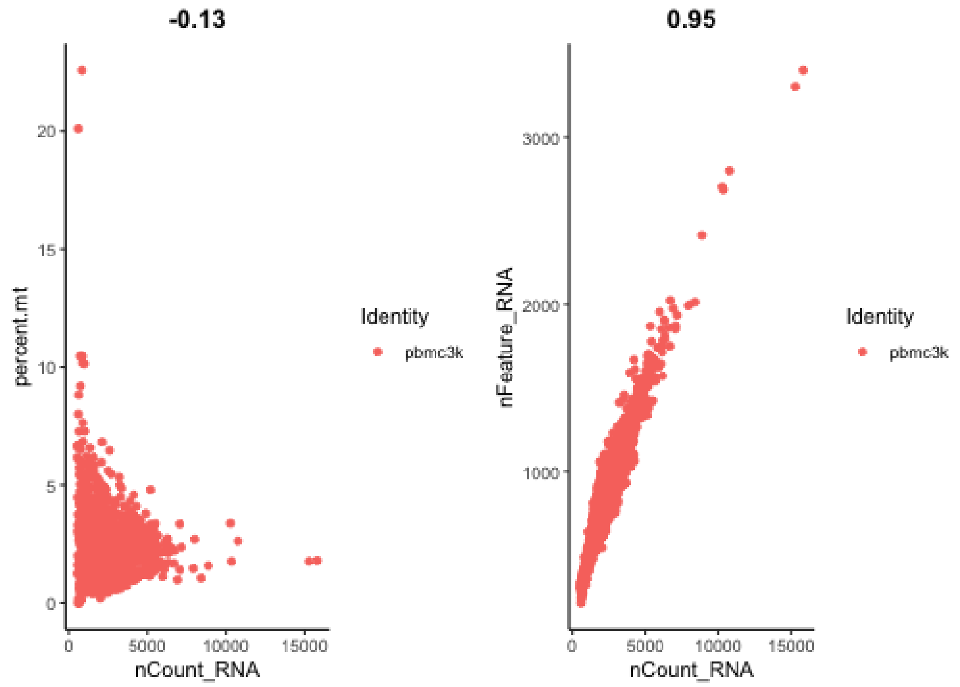
Visualizing feature-feature relationships

```
plot1 <- FeatureScatter(object = pbmc,
                        feature1 = "nCount_RNA",
                        feature2 = "percent.mt")
plot2 <- FeatureScatter(object = pbmc,
                        feature1 = "nCount_RNA",
                        feature2 = "nFeature_RNA")
CombinePlots(plots = list(plot1, plot2))
```



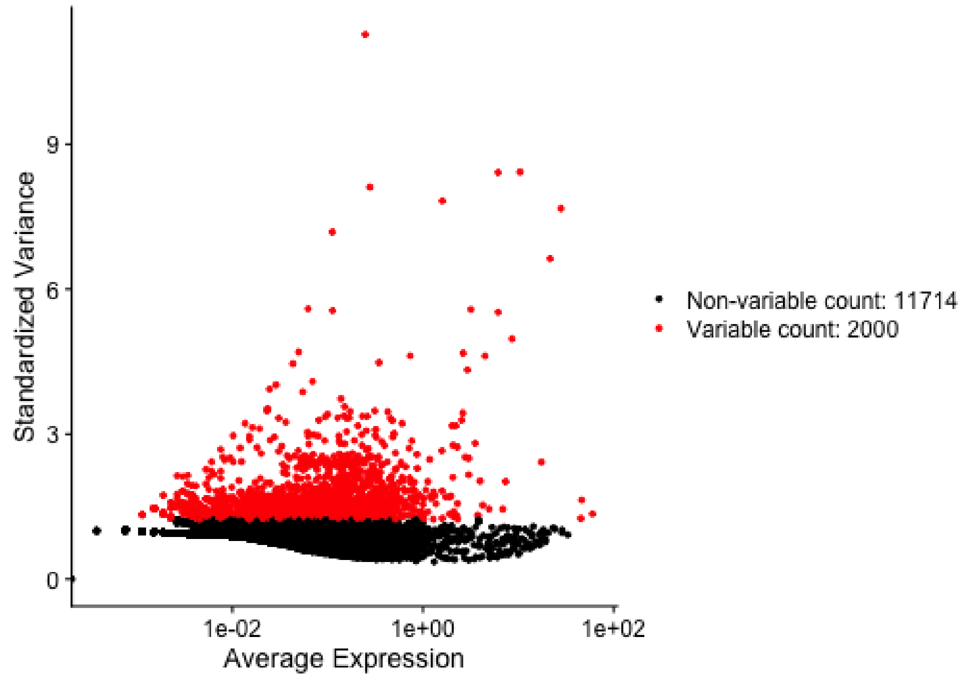
Visualizing feature-feature relationships

```r
cormatrix <- cor(pbmc@meta.data %>% dplyr::select(-or
plt1 <-
  ggplot(pbmc@meta.data,
         aes(x = nCount_RNA,
             y = percent.mt,
             group = orig.ident,
             color = orig.ident)) +
  geom_point() +
    theme_classic() +
    labs(color = 'Identity',
         title=as.character(round(cormatrix['nCount_R
  theme(plot.title = element_text(face = 'bold', hjus

plt2 <-
  ggplot(pbmc@meta.data,
         aes(x = nCount_RNA,
             y = nFeature_RNA,
             group = orig.ident,
             color = orig.ident)) +
  geom_point() +
  theme_classic() +
  labs(color = 'Identity',
       title=as.character(round(cormatrix['nCount_RNA
  theme(plot.title = element_text(face = 'bold', hjus

ggpubr::ggarrange(plt1, plt2, nrow = 1, ncol=2)
```



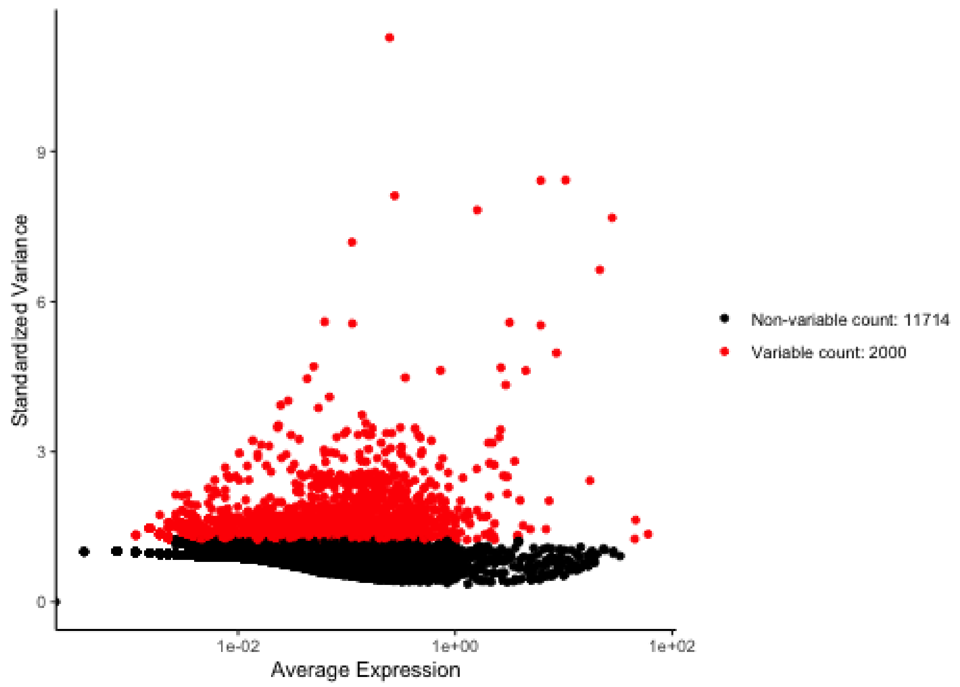
Feature selection

```
pbmc <- subset(x = pbmc,
    subset = nFeature_RNA > 200 & nFeature_RNA < 2500
pbmc <- NormalizeData(object = pbmc,
                      normalization.method = "LogNorm
                      scale.factor = 10000)
# This is stored in pbmc[['RNA']]@meta.features

pbmc <- FindVariableFeatures(object = pbmc,
                             selection.method = "vst"
                             nfeatures = 2000)


# Identify the 10 most highly variable genes
top10 <- head(x = VariableFeatures(object = pbmc), 10

# plot variable features with and without labels
plot1 <- VariableFeaturePlot(object = pbmc)
plot1
```



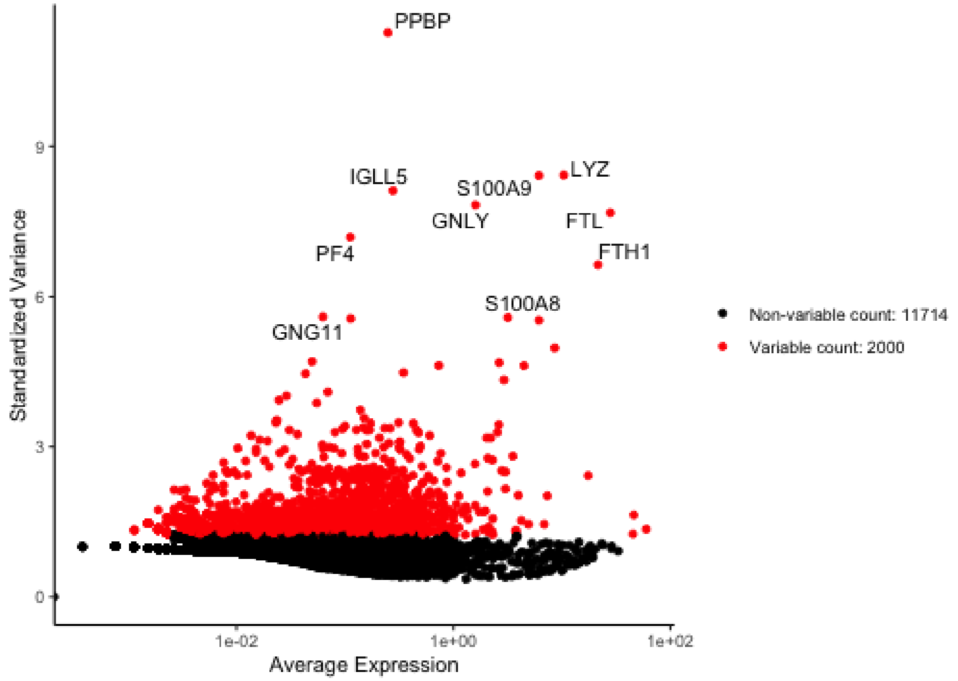
Feature selection

```r
plt_data <- pbmc[['RNA']]@meta.features %>%
    rownames_to_column(var='id')
topvars <- pbmc[['RNA']]@var.features
plt_data <- plt_data %>%
    mutate(indic = ifelse(id %in% topvars,
                          'Variable count',
                          'Non-variable count'))
bl <- plt_data %>%
    dplyr::count(indic) %>%
    glue::glue_data("{indic}: {n}")
names(bl) <- c('Non-variable count','Variable count')
plt_data <- plt_data %>%
  mutate(indic = bl[indic])
plt11 <- ggplot(plt_data,
                aes(x = vst.mean,
                    y = vst.variance.standardized,
                    color = indic)) +
  geom_point() +
  scale_x_log10() +
  scale_color_manual(values = c('black','red')) +
  labs(x = 'Average Expression', y = 'Standardized Va
  theme_classic()
plt11
```



Feature selection

```
# plot2 <- LabelPoints(plot = plot1, points = top10,
plt12 <- plt11 + ggrepel::geom_text_repel(data = plt_
                                           aes(label =
                                           color = 'bl
plt12
```



There's a lot more

We'll stop our sampling here.

- Many Bioconductor packages do use ggplot, however some use base graphics
  - Faster
- Key is to find where the data is stored, and use that to create visualizations
- Bioconductor tends to create
  - One monolithic object
  - Containing different information in slots
  - combined by lists
- `slotNames` and `names` are your friends