

Week 5 slides

BIOF 439

Goals

1. Visualization in biomedical applications and models
2. Visualizing spatial data using maps
3. Bioinformatics

Biomedical applications

Survival analysis

Survival analysis

Survival analysis is the analysis of time-to-event data. In biomedical research, *event* can mean

1. death
2. recurrence or relapse (cancer, infection)
3. equipment failure (pacemakers, orthopedic implants)

You need two pieces of information:

1. the time of the event or the time you last saw the subject, and
2. the status (dead/alive) the last time you saw the subject. If subject is alive, you call that observation *censored*, since you didn't observe the subject long enough to see the event happen.

Survival analysis

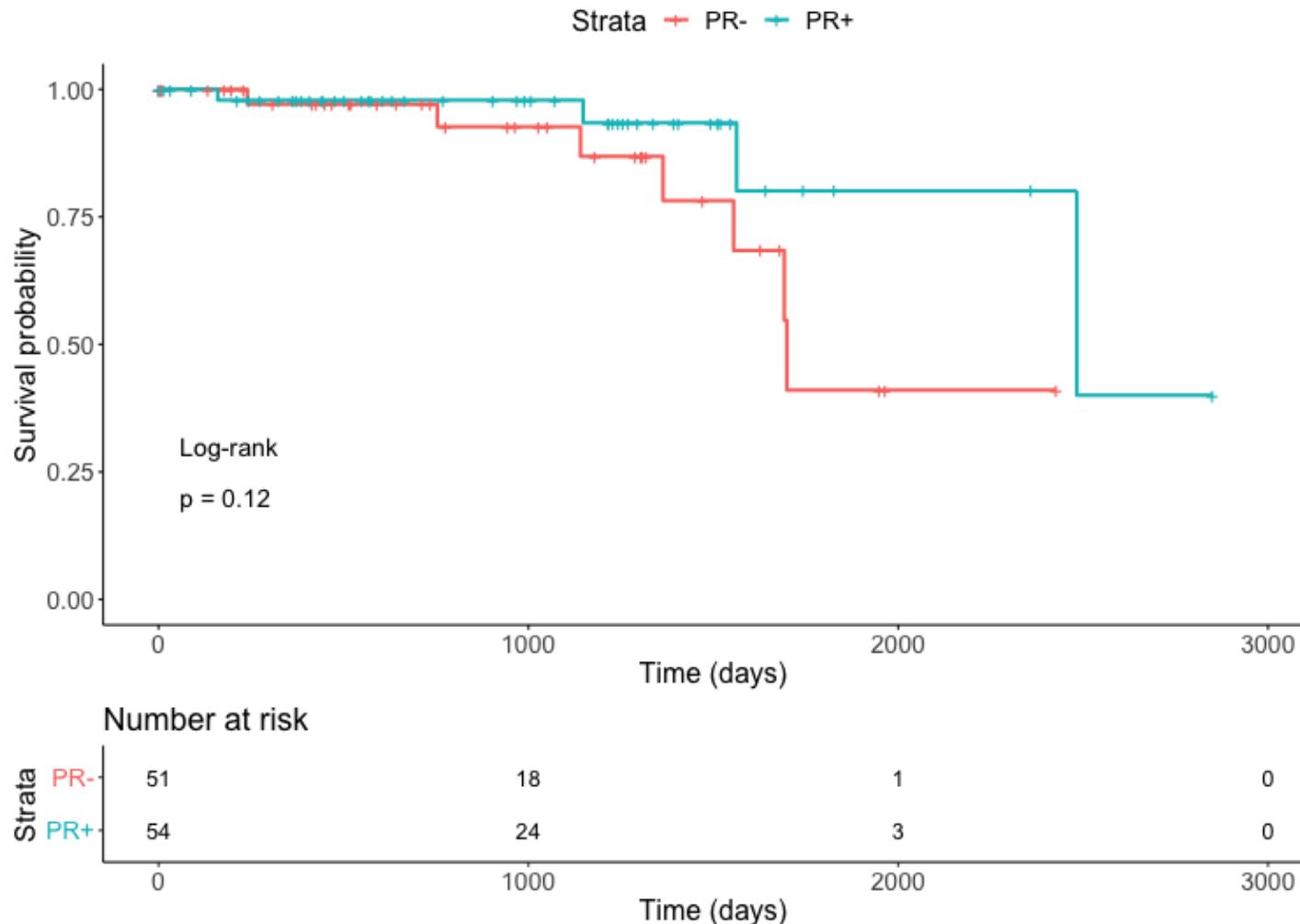
The basic graphical tool in survival analysis is the *Kaplan-Meier curve*, which shows the proportion of subjects **still surviving** at any given time, with 100% alive at time 0.

The Kaplan-Meier curve is computed using the function `survival::survfit` and uses the same formula interface as usual models, with one little quirk.

```
library(survival)
brca <- readxl::read_excel('data/BreastCancer_Clinical.xlsx', .name_repair = 'universal')
fit <- survfit(Surv(OS.Time, OS.event) ~ PR.Status, data = brca)
```

We have a composite dependent variable, comprising both the time and status of each event. It is encapsulated in the function `Surv` (note capitalization)

Kaplan-Meier curve



- Small ticks are times when subjects are censored
- The log-rank test tests if the stratified curves are statistically different
- The risk table gives the number still alive at different times.

Kaplan-Meier curve

The plot is generated by the following code:

```
library(survminer)
(plt <- ggsurvplot(fit, pval=TRUE, pval.method = TRUE, risk.table=TRUE,
                    legend.labs = c('PR-', 'PR+'),
                    xlab = 'Time (days)'))
```

- The actual plot can be accessed using `plt$plot` and is a regular **ggplot2** object
- The table can be accessed using `plt$table` and is *also* a **ggplot2** object

There is a lot of customization of these plots that are possible. See
<https://rpkgs.datanovia.com/survminer/index.html> for tutorials and details.

Regression results

Displaying the results of a regression model

We're used to seeing the results of a regression analysis in tables, but graphically displaying them may make them easier to understand.

We'll use a toy example using the mpg data.

We fit a linear regression model using

```
mpg <- mpg %>%
  mutate(year = as.factor(year)) %>%
  mutate(trans = str_extract(trans, '[[:alnum:]]+')) %>%
  mutate(cyl = as.factor(cyl))
fit <- lm(cty ~ year + trans + cyl + drv + class, data=mpg)
```

This models city mpg against year, type of transmission, number of cylinders, type of drive, and class of car.

Displaying the results of a regression model

As a table, these results look like

```
knitr::kable(broom::tidy(fit))
```

term	estimate	std.error	statistic	p.value
(Intercept)	20.5622426	1.1227750	18.3137694	0.0000000
year2008	0.4624090	0.2712424	1.7047815	0.0896468
transmanual	0.4024746	0.3106162	1.2957299	0.1964264
cyl5	-2.1058003	1.0607113	-1.9852718	0.0483563
cyl6	-3.4162456	0.3620780	-9.4351106	0.0000000
cyl8	-5.5938201	0.4951472	-11.2972877	0.0000000
drvf	2.5030933	0.4872506	5.1371787	0.0000006
drvrr	-0.0873527	0.5793938	-0.1507656	0.8802988
classcompact	-1.6774636	1.1124558	-1.5078924	0.1330167
classmidsize	-2.2914265	1.1438440	-2.0032683	0.0463754
classminivan	-4.3516621	1.2936355	-3.3639014	0.0009065

Obviously this needs cleaning up

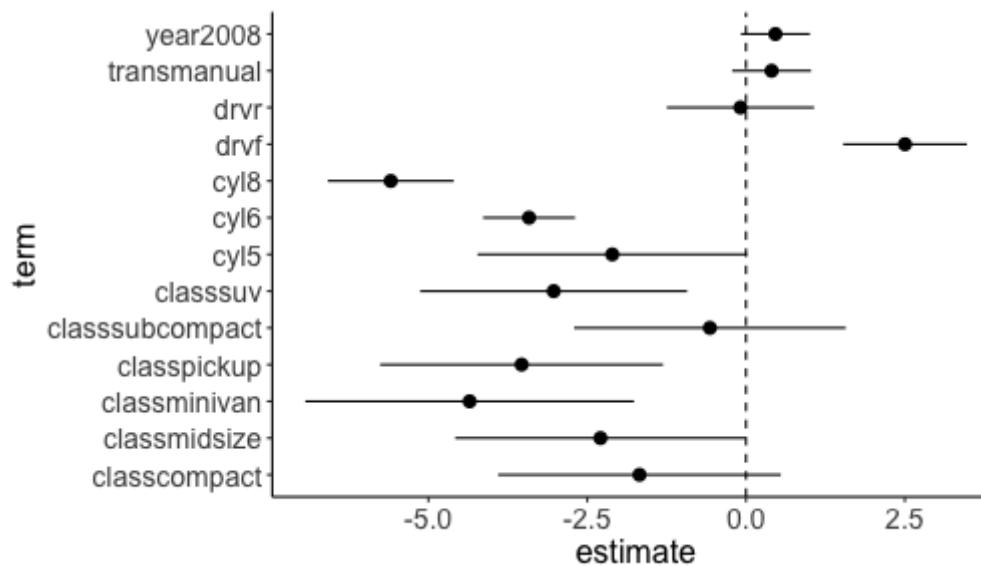
We'd fix the terms, and reduce the number of decimal places to 2

The point is, it's relatively hard to understand the relative effects of different levels.

Displaying the results of a regression model

As a figure, it's probably a bit better

```
results = broom::tidy(fit) %>% filter(term != "(Intercept)")  
ggplot(results)+  
  geom_pointrange(aes(x=term, y=estimate,  
                      ymin = estimate-2*std.error, ymax=estimate+2*std.error))+  
  geom_hline(yintercept=0, linetype=2) +  
  coord_flip() +  
  theme_439
```



We plot the estimate and 95% confidence interval for each term

Provide reference line of 0

Can see increasing and decreasing patterns within variable levels

We'll work on cleaning this picture up in a tutorial

A learning note

You've been gaining some experience in creating graphics.

I'll increasingly be providing complete code, and will expect you to work on parsing that code and understanding what the different parts are doing

One of the nice things about learning an open-source language is that there are plenty of examples out there, and you're allowed (for the most part) to copy others' code. So developing the skill of understanding other people's code is really useful and somewhat important.

Reach out to me if it really feels mysterious and arcane. I'm not teaching mystical arts here.

Logistic regression

We'll use the PimaIndianDiabetes2 data from the **mlbench** package

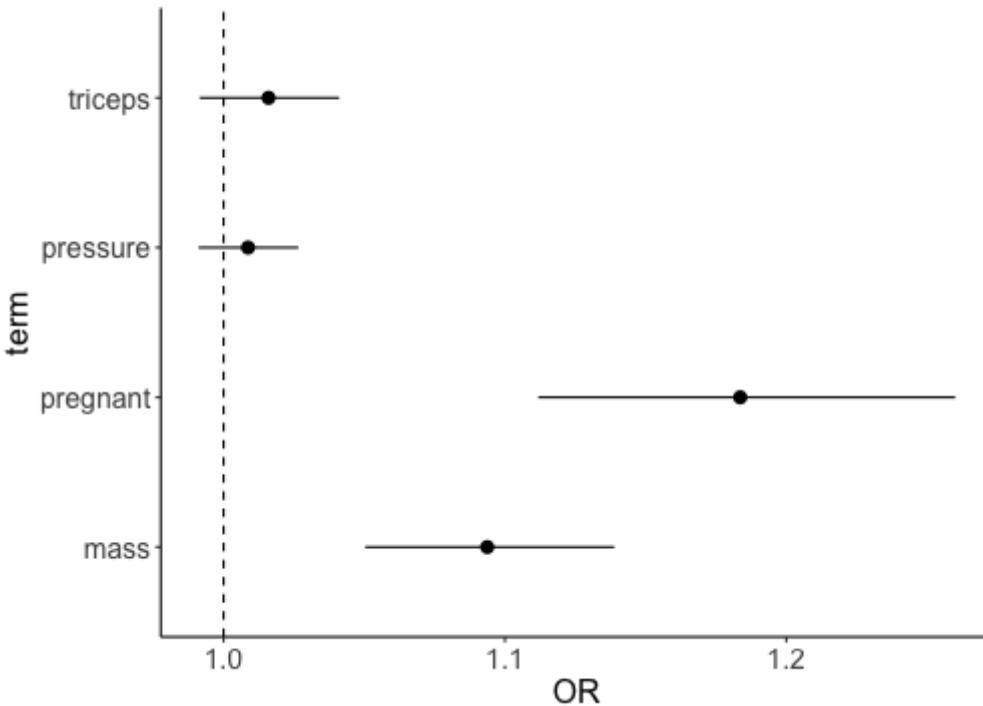
```
library(mlbench); data("PimaIndiansDiabetes2") # 2 commands separated by ;
fit_logistic <- glm(diabetes ~ pregnant + pressure + triceps + mass,
                      data = PimaIndiansDiabetes2,
                      family='binomial') # This option makes this logistic regression
```

The unifying step to displaying the results of most regression modeling is `broom::tidy`. We'll transform the results a bit, since logistic regression provides log-odds ratios and we will display the odds ratios.

```
results <- broom::tidy(fit_logistic)
results <- results %>%
  mutate(OR = exp(estimate),
        LCB = exp(estimate - 2*std.error),
        UCB = exp(estimate + 2*std.error)) %>%
  filter(term != '(Intercept)')
```

Logistic regression

```
ggplot(results)+  
  geom_pointrange(aes(x = term, y = OR, ymin = LCB, ymax = UCB))+  
  geom_hline(yintercept = 1, linetype=2)+  
  coord_flip()
```



The graph makes it clear that number of pregnancies and body mass significantly influence the odds of being diabetic, but blood pressure and tricep size do not

Cox proportional hazards regression

For survival data, Cox regression is typically used to model the *hazard rate*, i.e. the instantaneous risk of having an event.

We'll use the breast cancer dataset to do this, after some data munging

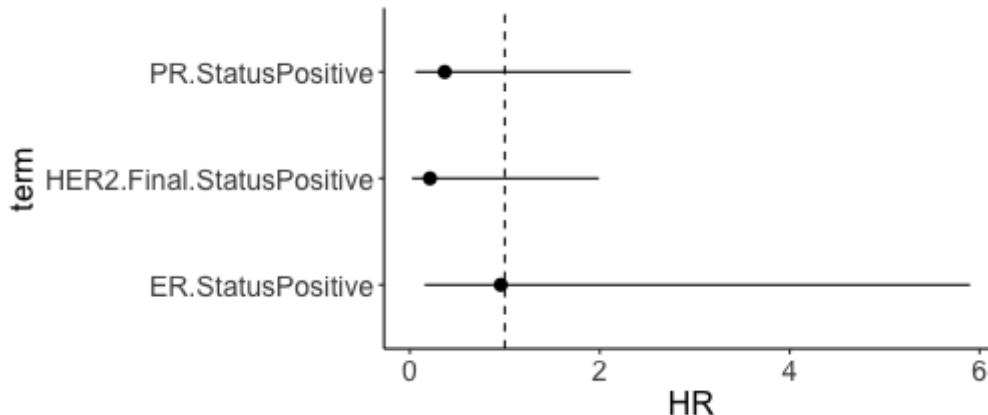
```
library(survival)
brca1 <- brca %>%
  mutate(ER.Status = ifelse(ER.Status == 'Indeterminate', NA, ER.Status),
        HER2.Final.Status = ifelse(HER2.Final.Status=='Equivocal', NA, HER2.Final.Status))
fit_cox <- coxph(Surv(OS.Time,OS.event) ~ ER.Status + PR.Status + HER2.Final.Status, data=brca1)
(results <- broom::tidy(fit_cox))
```

```
# A tibble: 3 x 5
  term            estimate std.error statistic p.value
  <chr>           <dbl>     <dbl>      <dbl>    <dbl>
1 ER.StatusPositive -0.0428   0.909    -0.0471   0.962
2 PR.StatusPositive  -1.00    0.923    -1.08     0.278
3 HER2.Final.StatusPositive -1.55    1.12     -1.39    0.166
```

Notice that a Cox regression does not have an intercept term

Cox regression

```
results <- results %>%
  mutate(HR = exp(estimate),
        LCB = exp(estimate - 2*std.error),
        UCB = exp(estimate + 2*std.error))
ggplot(results)+  
  geom_pointrange(aes(x = term, y = HR,  
                      ymin = LCB, ymax = UCB)) +  
  geom_hline(yintercept = 1, linetype=2)+  
  coord_flip() +  
  theme_439
```



These results may seem counter-intuitive

One may need to do some model building to see what model fits the data well, i.e. what variables and transformed variables should be included in the model.

Creating a function for these plots

We've basically used the same code to create the graphs in all three cases, albeit with some modest data transformations.

A general rule is that if you're using the same code more than twice, you should probably make it into a function, so you avoid copy-and-paste of code and the mistakes that engenders

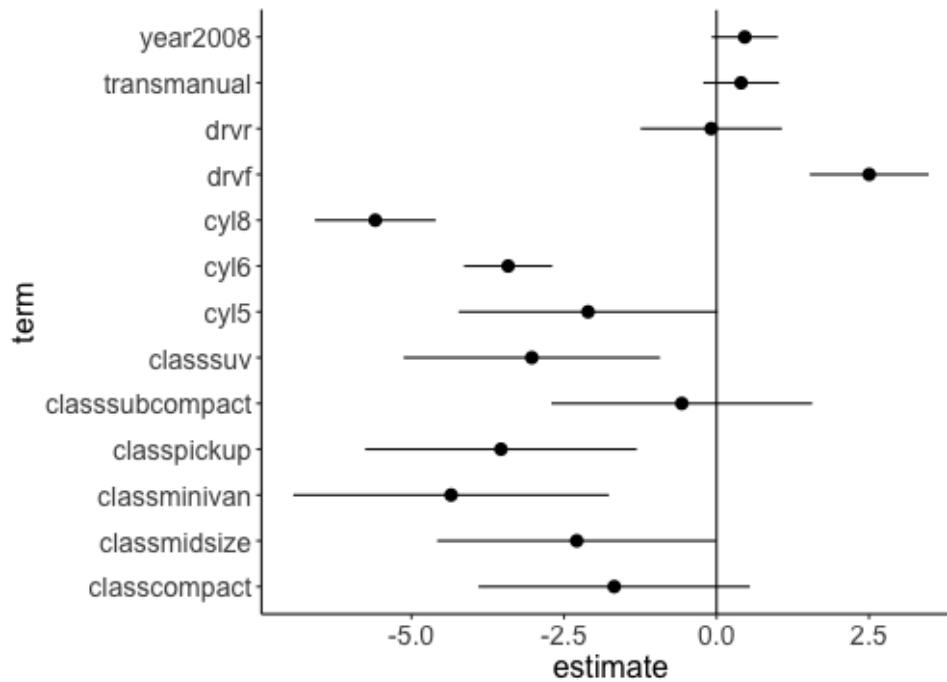
```
plt_reg_results <- function(results, reflevel=0){  
  require(ggplot2)  
  plt <- ggplot(results)+  
    geom_pointrange(aes(x = term, y = estimate,  
                        ymin = LCB, ymax = UCB))+  
    geom_hline(yintercept = reflevel)+  
    coord_flip()  
  return(plt)  
}
```

You should document what format results should be in, and what reflevel means using comments

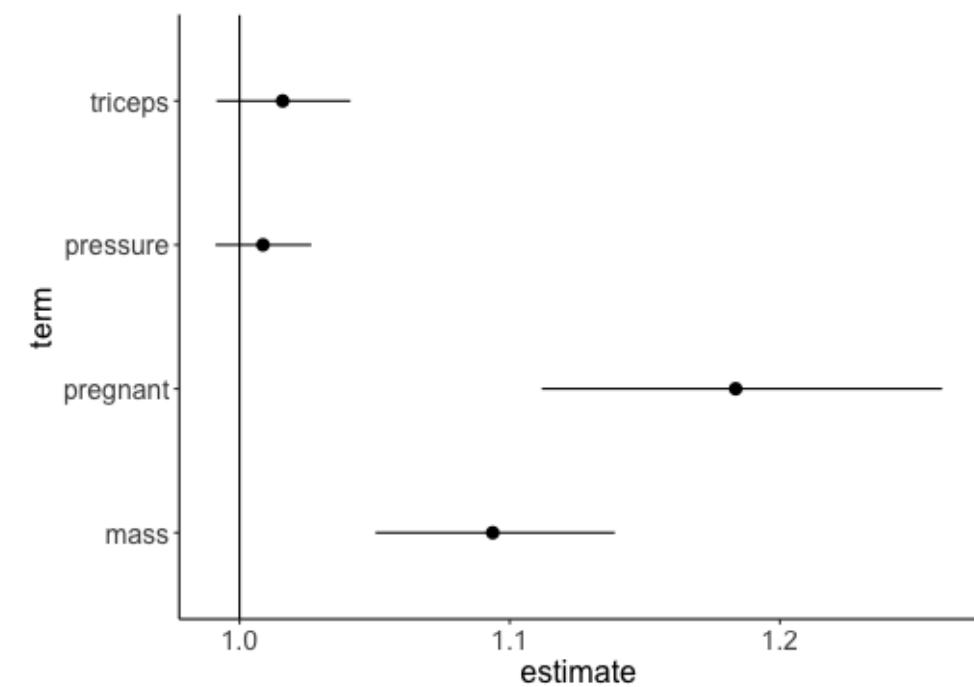
With such a function, you might need to do a bit more data munging, but creating the plots becomes a unified process.

Creating a function for these plots

```
broom::tidy(fit) %>%
  mutate(LCB = estimate - 2*std.error,
        UCB = estimate + 2*std.error) %>%
  filter(term != '(Intercept)') %>%
  plt_reg_results()
```



```
broom::tidy(fit_logistic) %>%
  mutate(LCB = exp(estimate - 2*std.error),
        UCB = exp(estimate + 2*std.error),
        estimate = exp(estimate)) %>%
  filter(term != '(Intercept)') %>%
  plt_reg_results(reflevel=1)
```



Maps

Toolsets

Visualizing spatial and geographic data is a specialized area on its own

R has increasing capabilities in this regard, and its increasingly mature. Some of the packages that we might need are

Data

- sf
- sp
- raster
- spData
- rnaturalearthdata

Visualization

- ggplot + ggmap + geom_sf
- tmap
- leaflet

Several parts of this lecture are inspired by [Chapter 8](#) of Geocomputation with R by Lovelace, Nowosad and Muenchow (2019), also available on [Amazon](#)

Toolset

We'll start of loading the following packages

```
library(ggplot2)  
library(sf)  
library(spData)
```

The **sf** package provides [simple features access](#) for R, and helps to store and process geographic data within the tidyverse framework, while linking to several geospatial packages that are standard in the geography world.

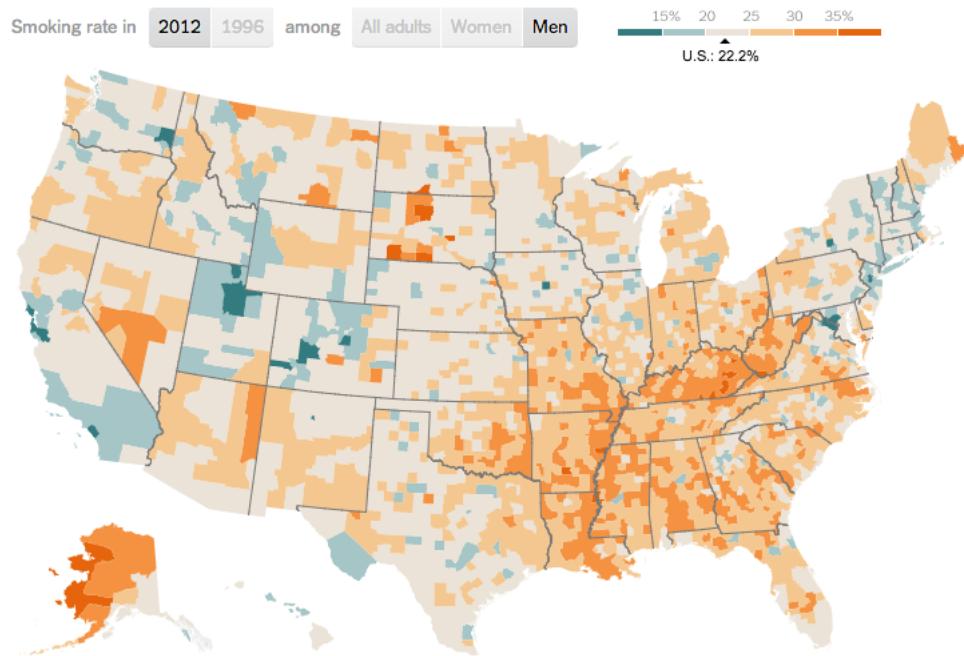


To use **sf** you may need to install some additional software. At the very least you will need to install the R packages **rgdal** and **rgeos**.

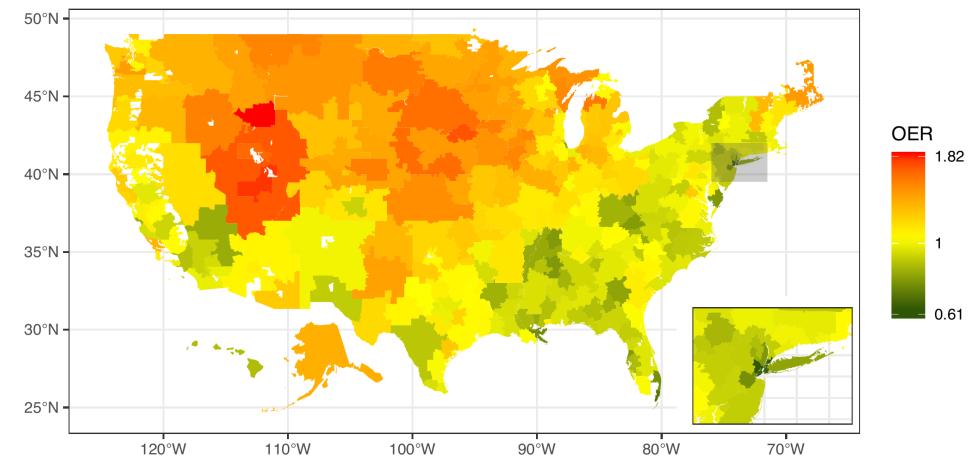
Additional information is available [here](#)

Chloropleth maps

Chloropleth maps are maps with some geometries filled in to signify levels of some variable.



Smoking rates in USA in 2012 (NY Times, March 24, 2014)



Observed to Expected Ratios (OERs) for Rates of Primary Total Knee Arthroplasty Among White Medicare Beneficiaries by Health Referral Region ([Ward & Dasgupta, 2020](#))

A chloropleth of life expectancy

We'll start off with a world map

```
library(sf); library(spData)  
  
ggplot(data = world) +  
  geom_sf() # a special geometry for plotting maps
```



There are several ways of getting map geometries, which are specifications of polygons.

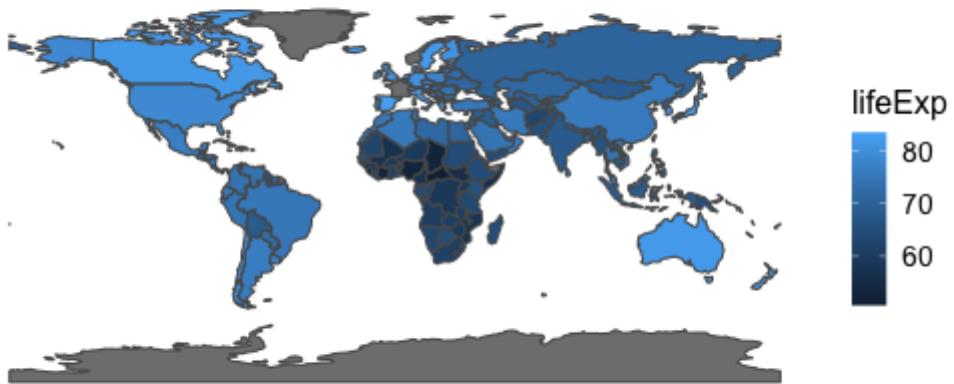
If you look at `world`, you'll see it's a `data.frame`, with one column named `geometry`. This column provides the shapes of the polygons, and what `geom_sf` looks for

A chloropleth of life expectancy

If you look at world, it also provides life expectancy estimates from 2014 (World Bank). The data set is tidy, with one row corresponding to one country. We'll use our known **ggplot2** way of filling things in.

```
ggplot(data = spData::world) +  
  geom_sf(aes(fill = lifeExp)) # a special geometry for plotting maps
```

We need a more distinctive color palette.

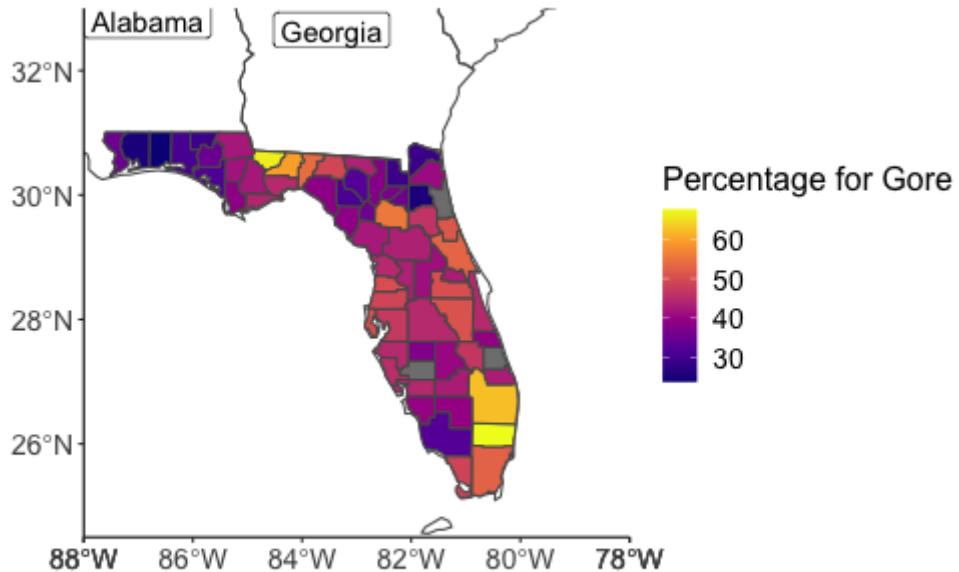


A chloropleth of life expectancy

If you look at `world`, it also provides life expectancy estimates from 2014 (World Bank). The data set is tidy, with one row corresponding to one country. We'll use our known **ggplot2** way of filling things in.

```
ggplot(data = spData::world) +  
  geom_sf(aes(fill = lifeExp)) + # a special geometry for plotting maps  
  viridis::scale_fill_viridis('Life Exp', option='plasma',  
    trans='sqrt', labels = scales::label_number_si())
```

The electoral picture in Florida, 2000



We'll develop this map in a tutorial

Using tmap

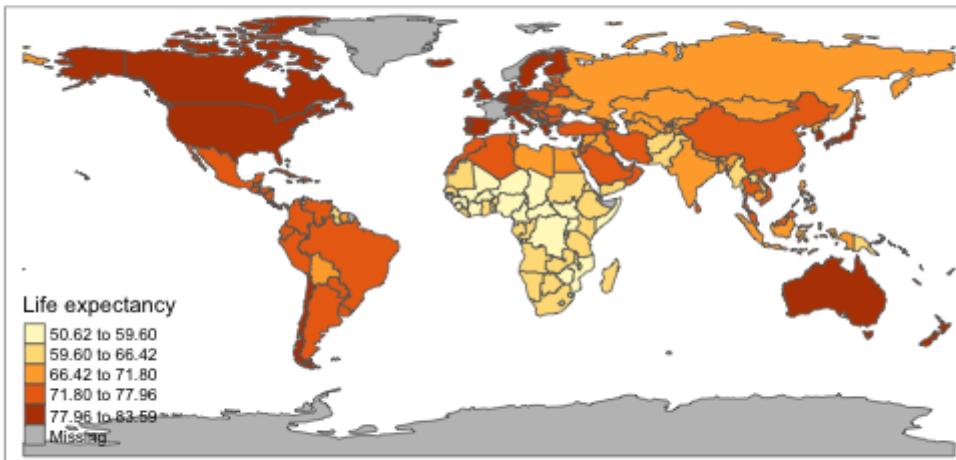
Using tmap

The **tmap** package uses many syntactical structures similar to **ggplot2**, but can be nicer in some ways

```
library(tmap)
tm_shape(spData::world) + tm_polygons(col = "lifeExp", style='jenks',
                                         title = 'Life expectancy')
```

It's more "publication-ready" by default

It makes some nice choices



tmap (interactive)

```
tmap_mode('view')
tm_shape(spData::world) + tm_polygons(col = 'pop', style='jenks',
                                         palette='Reds', title='Population',
                                         popup.vars = c('pop'), id='name_long')
```

```
<div id="htmlwidget-2b4b579aa17ced1b59d6" style="width:100%;height:100%;" class="tmap-interactive">
<script type="application/json" data-for="htmlwidget-2b4b579aa17ced1b59d6">{"x":
```

Just need one line of code
for interactivity!

It gains interactivity using
leaflet, which we'll see in a
bit

Street maps

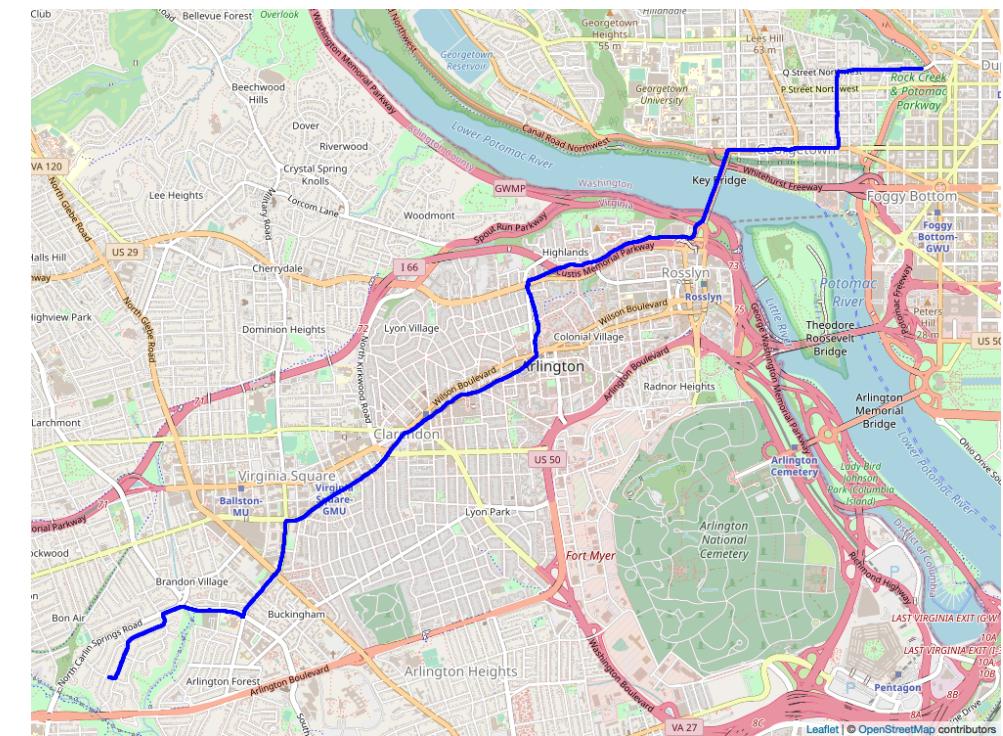
The easiest ways to overlay data on street maps is with the **leaflet** package.

```
library(leaflet)
library(leaflet.providers)
load(file.path(here::here('slides','lectures','data', 'exdata.rda')))
leaflet(gpx) %>% addTiles() %>% addCircleMarkers(~Longitude, ~Latitude, radius=1
```

Street maps

You can also use the **mapview** package, which calls **leaflet** and has a bit more compact syntax

```
load(file.path(here::here('slides', 'lectures', 'data'),
gpx <- sf::st_as_sf(gpx,
  coords=c("Longitude", "Latitude"),
  crs = 4267) # Need to make sf object
mapview::mapview(gpx, color='blue',
  map.type = 'OpenStreetMap.Mapnik',
  cex = 0.2, # size of points
  legend=FALSE)
```



Street maps

You can also use the **mapview** package, which calls **leaflet** and has a bit more compact syntax

```
load(file.path(here::here('slides', 'lectures', 'data'),  
gpx <- st_as_sf(gpx,  
                  coords=c("Longitude", "Latitude"),  
                  crs = 4267) # Need to make sf object  
mapview::mapview(gpx, color='blue',  
                  map.type = 'Stamen.Watercolor',  
                  cex = 0.2, # size of points  
                  legend=FALSE)
```

You can also have some stylistic fun with maps.

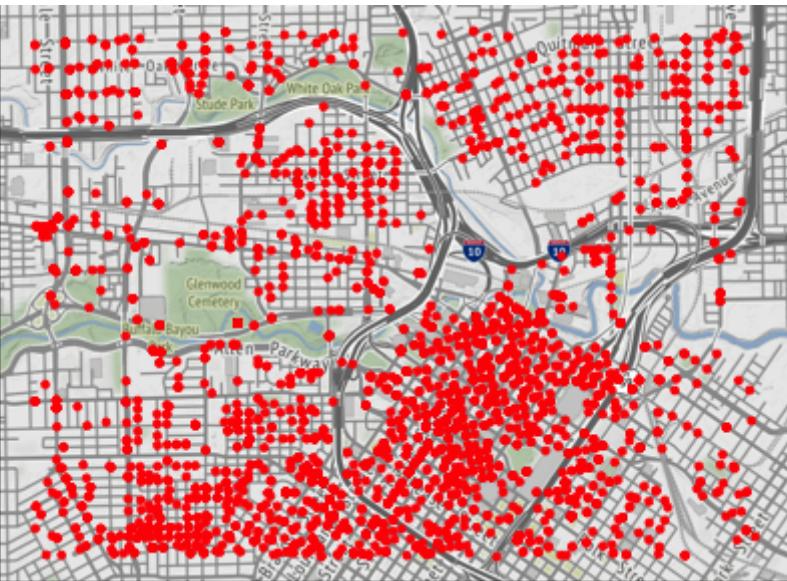
More possibilities at <http://leaflet-extras.github.io/leaflet-providers/preview/>



Dot density maps

```
library(ggmap)
crime1 <- crime %>%
  filter(between(crime$lon, -95.4, -95.34) &
         between(crime$lat, 29.746, 29.784))

qmapplot(lon, lat, data=crime1, maptype='toner-lite',
          color = I('red'))
```

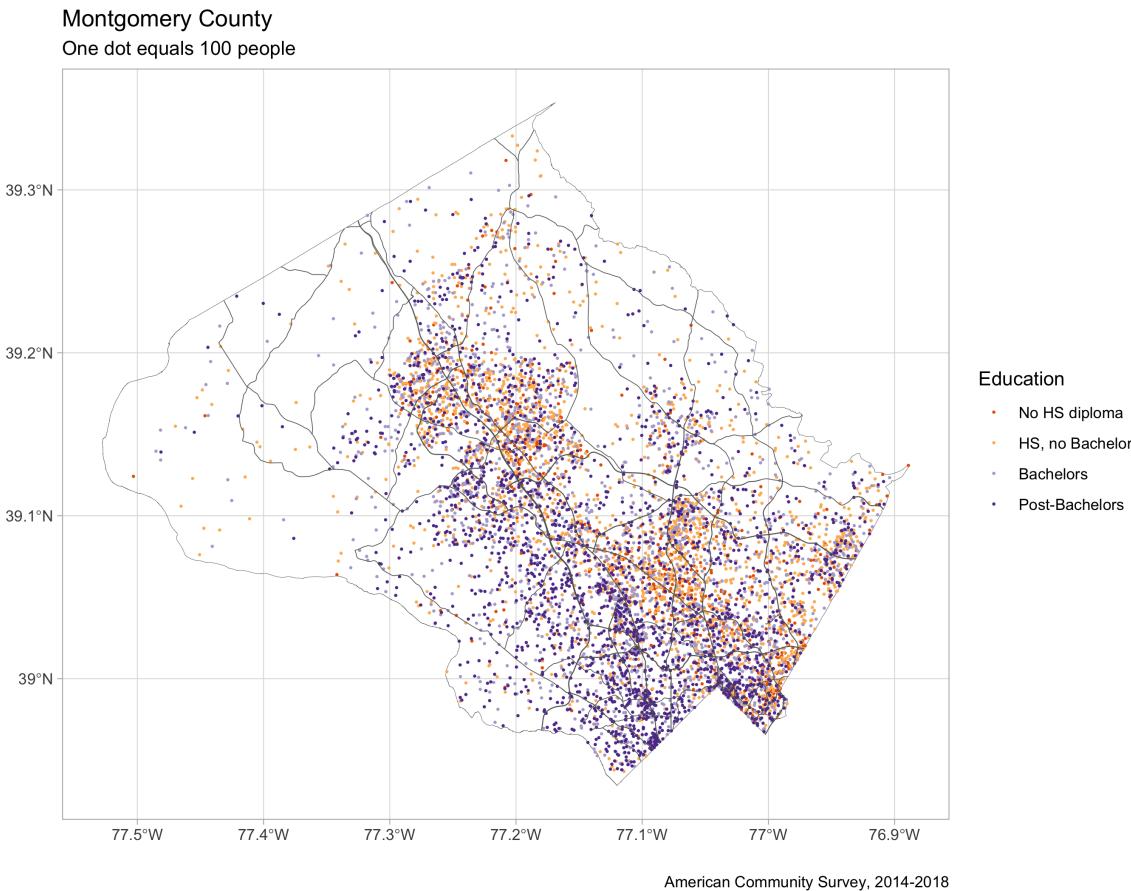


ggmap was built for Google Maps

Google Maps requires a credit card now

Better option is Stamen Maps, which uses OpenStreetMap data

Dot density maps

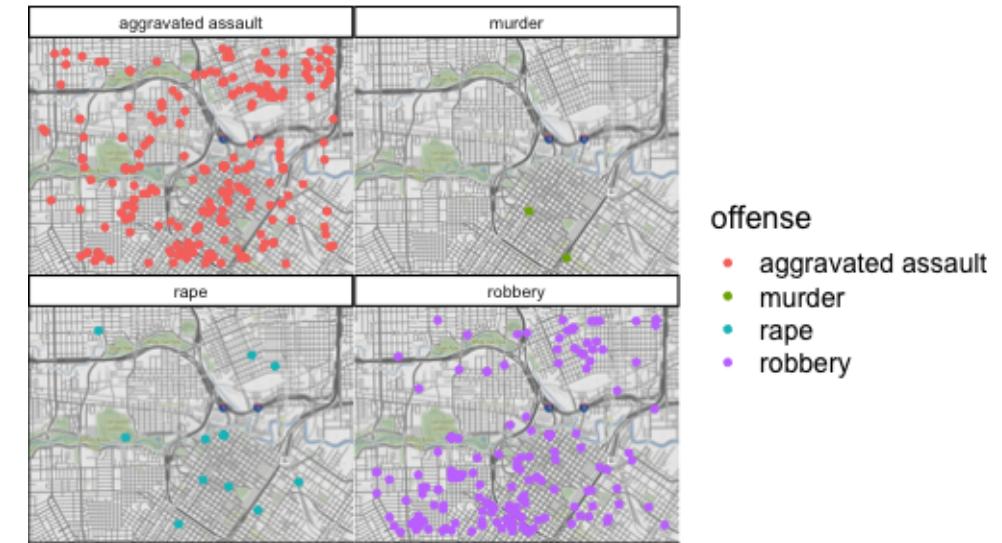


Code is available [here](#)

Based on [this blog](#) by Tarak Shah

Facetted maps

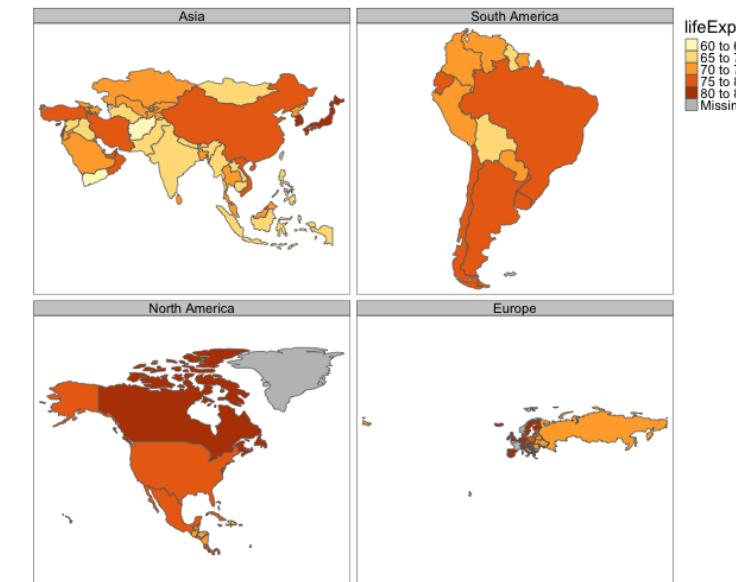
```
library(ggmap)
crime1 <- crime1 %>%
  filter(!(offense %in% c('auto theft', 'theft',
                         'burglary')))
qmapplot(lon, lat, data=crime1,
          maptype='toner-background',
          color = offense) +
  facet_wrap(~offense)
```



Facetted maps

```
library(tmap)
world1 <- world %>%
  filter(continent %in% c('Europe', 'Asia',
                          'North America',
                          'South America')) %>%
  mutate(continent = fct_reorder(continent,
                                  lifeExp, na.rm=T))

tm_shape(world1)+tm_polygons(col='lifeExp') +
  tm_facets(by='continent', ncol=2)
```



layout: true

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	node2_external_id	neighborhood	fusion	cooccurrence	homology	coexpression
1	CXCR3	CCR7	1855969	1843829	ENSP0000362795	ENSP0000246657	0	0	0	0.847	0.000
2	ITGA4	EED	1858446	1845338	ENSP0000380227	ENSP0000263360	0	0	0	0.000	0.000
3	SMC3	CENPK	1854200	1843648	ENSP0000354720	ENSP0000242872	0	0	0	0.000	0.000
4	HNRNPA1	LUC7L3	1852510	1843556	ENSP0000341826	ENSP0000240304	0	0	0	0.000	0.000
5	SMC2	RB1	1847012	1845924	ENSP0000286398	ENSP0000267163	0	0	0	0.000	0.136
6	RBBP4	CENPK	1855919	1843648	ENSP0000362592	ENSP0000242872	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
```

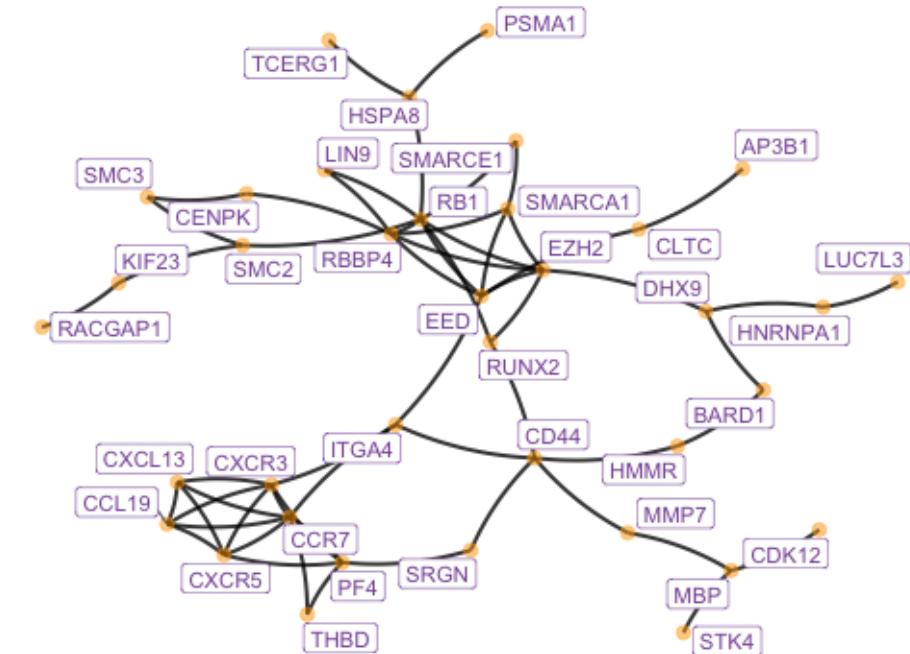
```
IGRAPH 339beab UN-- 37 58 --
+ attr: name (v/c)
+ edges from 339beab (vertex names):
 [1] CXCR3 --CCR7   ITGA4  --EED    SMC3   --CENPK   HNRNPA1--LUC7L3
 [5] SMC2   --RB1    RBBP4  --CENPK   CXCR5  --CXCL13  CD44   --RUNX2
 [9] CXCR5  --PF4    PF4    --THBD    SMARCA1--EZH2   HMMR   --BARD1
[13] 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

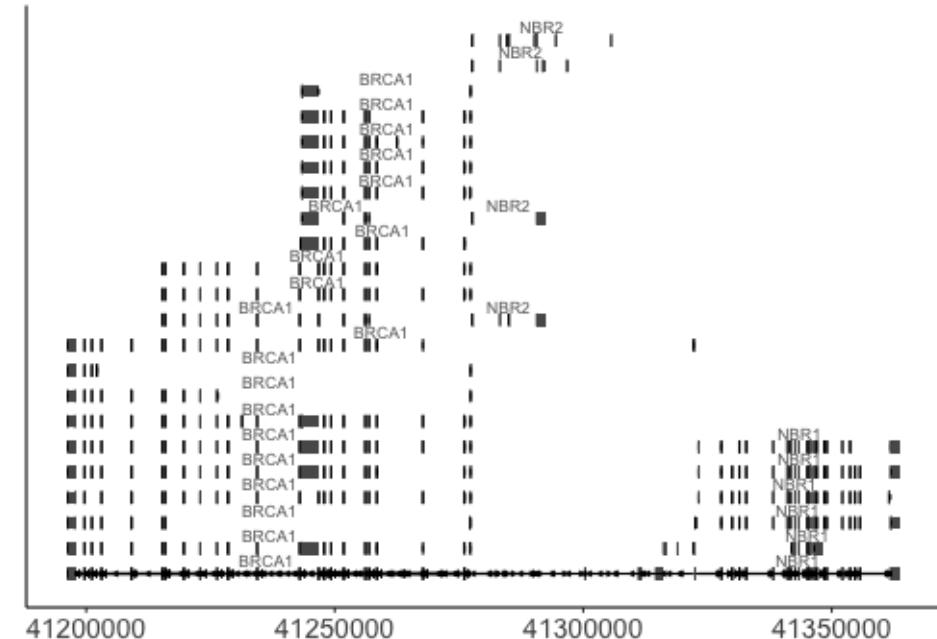
```
library(ggbio) # p_install('ggbio', try.bioconductor=TRUE)
p.ideo <- Ideogram(genome = 'hg19')
p.ideo
```



The ggbio package

Visualizing the gene model

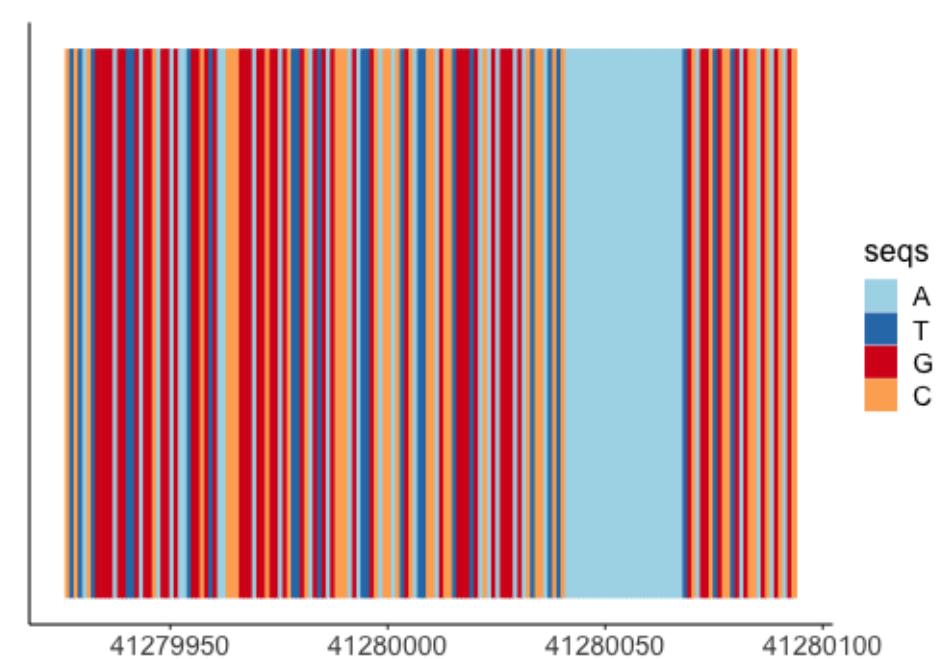
```
pacman::p_load(Homo.sapiens)
data(genesymbol, package='biovizBase')
wh <- genesymbol[c('BRCA1','NBR1')]
wh <- range(wh, ignore.strand=T)
p.txdb <- autoplot(Homo.sapiens, which = wh)
p.txdb
```



The ggbio package

A reference track

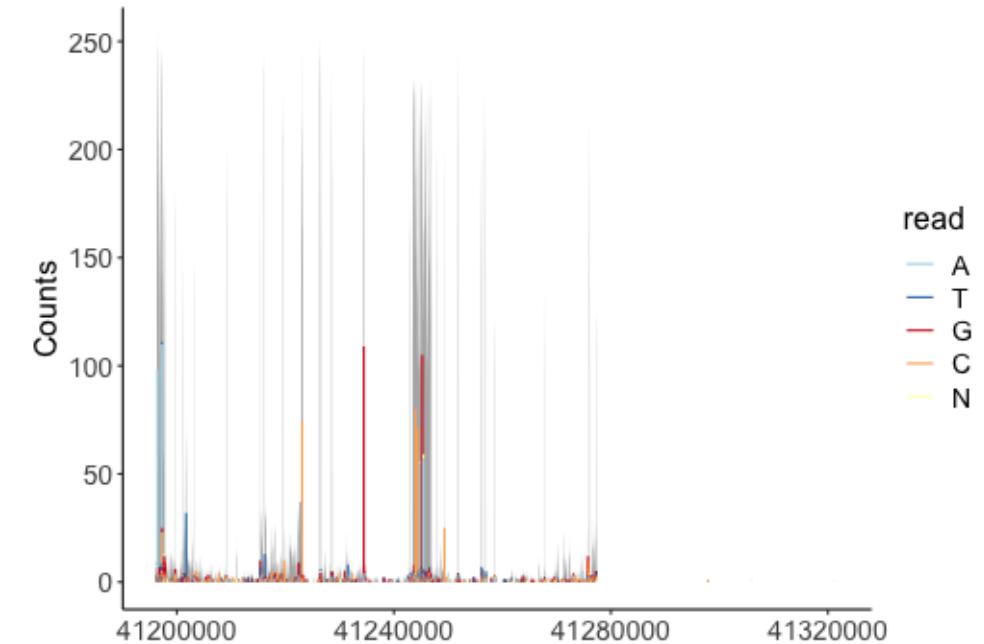
```
library(BSgenome.Hsapiens.UCSC.hg19)
bg <- BSgenome.Hsapiens.UCSC.hg19
p.bg <- autoplot(bg, which=wh)
p.bg + zoom(1/1000)
```



The ggbio package

An alignment track with mismatch proportions

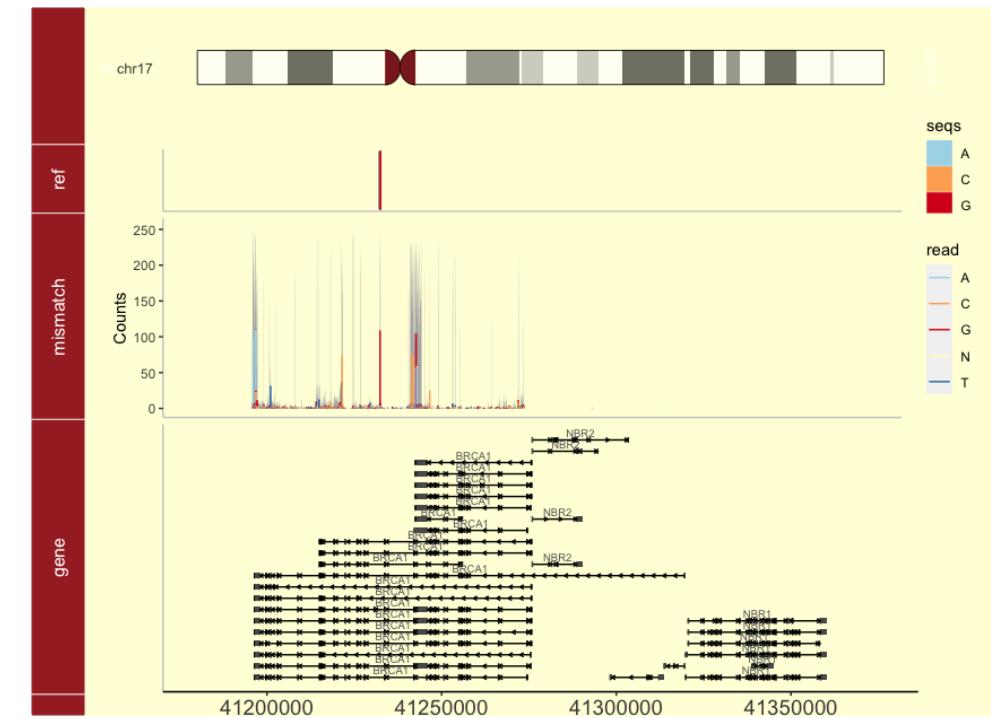
```
library(BSgenome.Hsapiens.UCSC.hg19)
fl.bam <- system.file("extdata", "wg-brca1.sorted.bam")
wh <- keepSeqlevels(wh, "chr17")
bg <- BSgenome.Hsapiens.UCSC.hg19
p.mis <- autoplot(fl.bam, bsgenome = bg, which = wh,
p.mis
```



The ggbio package

Putting it into tracks

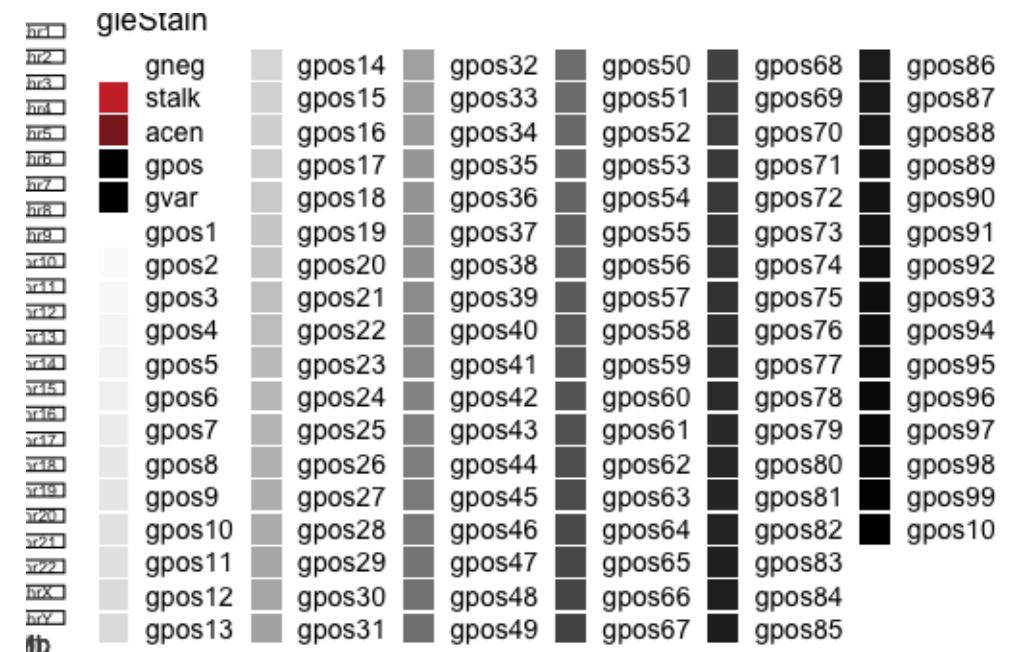
```
pacman::p_load(GenomicRanges)
gr17 <- GRanges("chr17", IRanges(41234415, 41234569))
tks <- tracks(p.ideo,
              ref=p.bg,
              mismatch=p.mis,
              gene=p.txdb,
              heights=c(2,1,3,4))+  
  xlim(gr17) +
  theme_tracks_sunset()
print(tks)
```



The ggbio package

A karyogram

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



P-values and Manhattan plots

A very simple example

```
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
1	TCGA-A2-A0CM		40	Negative 0.6834035
2	TCGA-BH-A18Q		56	Negative 0.1953407
3	TCGA-A7-A0CE		57	Negative -1.1231731
4	TCGA-D8-A142		74	Negative 0.5385958
5	TCGA-A0-A0J6		61	Negative 0.8311317
6	TCGA-A2-A0YM		67	Negative 0.6558497
	np_958785 np_958786 np_000436 np_958781 np_958780 np_958783 np_958784			
1	0.6944241 0.6980976 0.6870771 0.6870771 0.6980976 0.6980976 0.6980976			
2	0.2154129 0.2154129 0.2053768 0.2154129 0.2154129 0.2154129 0.2154129			
3	-1.1231731 -1.1168605 -1.1294857 -1.1294857 -1.1200168 -1.1231731 -1.1231731			
4	0.5422105 0.5422105 0.5349810 0.5422105 0.5422105 0.5422105 0.5422105			
5	0.8565398 0.8565398 0.8367780 0.8650092 0.8565398 0.8508936 0.8508936			
6	0.6581426 0.6558497 0.6558497 0.6512639 0.6581426 0.6558497 0.6558497			
	np_112598 np_001611			
1	-2.6521501 -0.9843733			
2	-1.0357599 -0.5172257			
3	2.2445844 -2.5750648			
4	-0.1482049 0.2674902			

A very simple example

```
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
1 0.6988415 0.6910103 0.6832121 0.6910103 0.6832121 0.6910103 0.6910103
np_958784 np_112598 np_001611
1 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

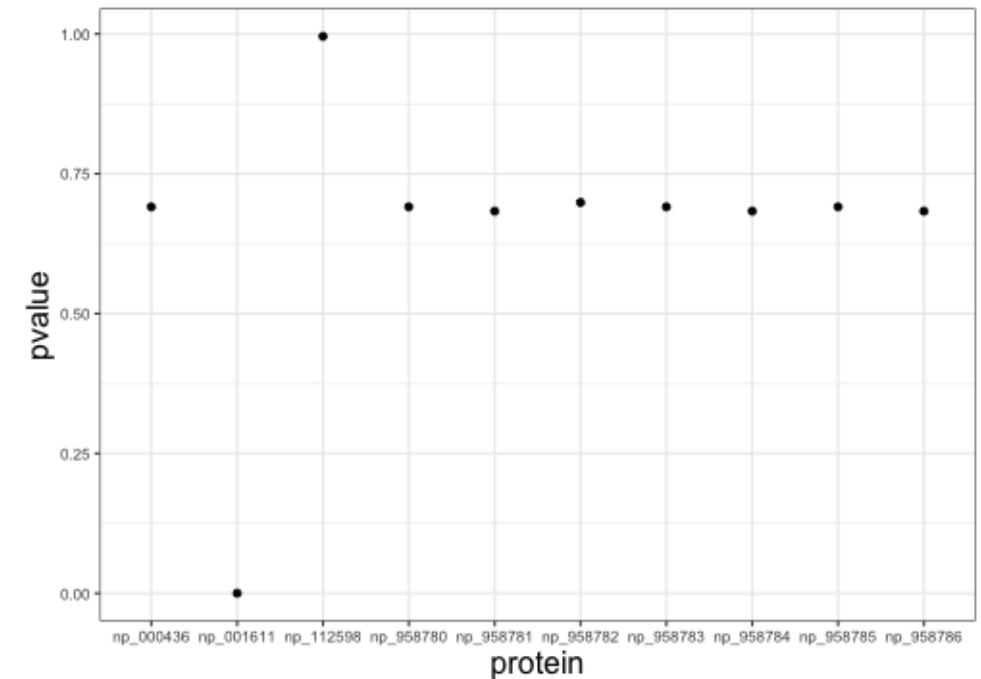
A very simple example

```
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
```

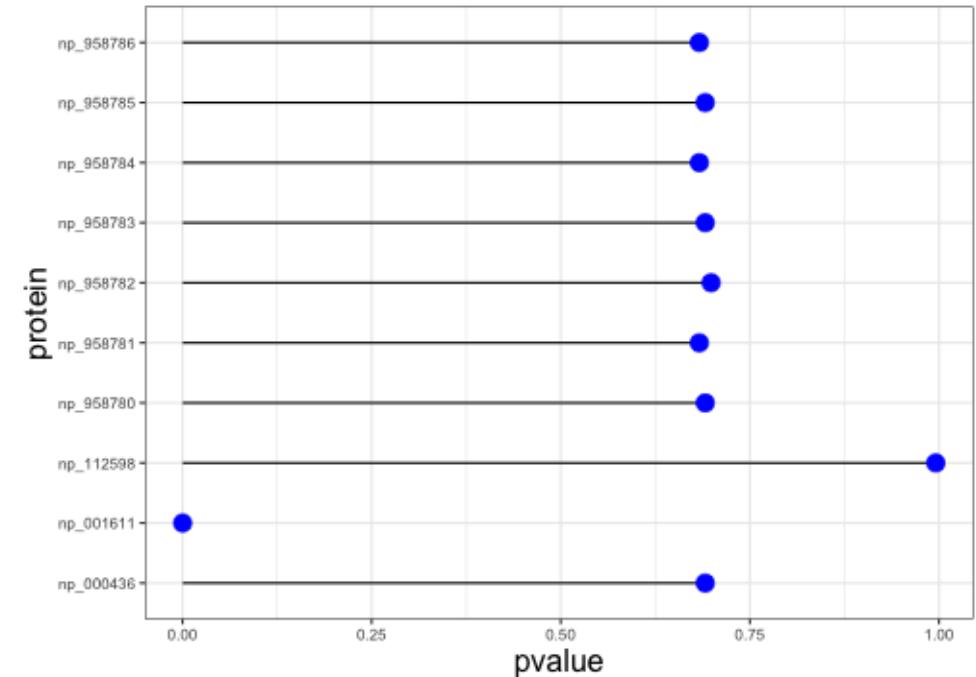
A very simple example

```
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
```



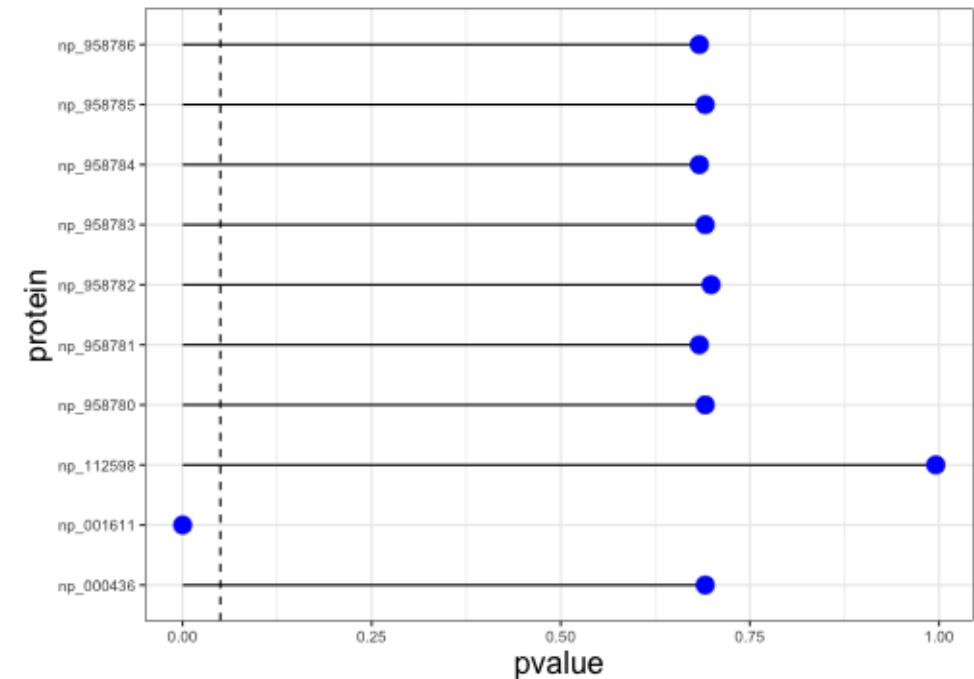
A very simple example

```
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
```

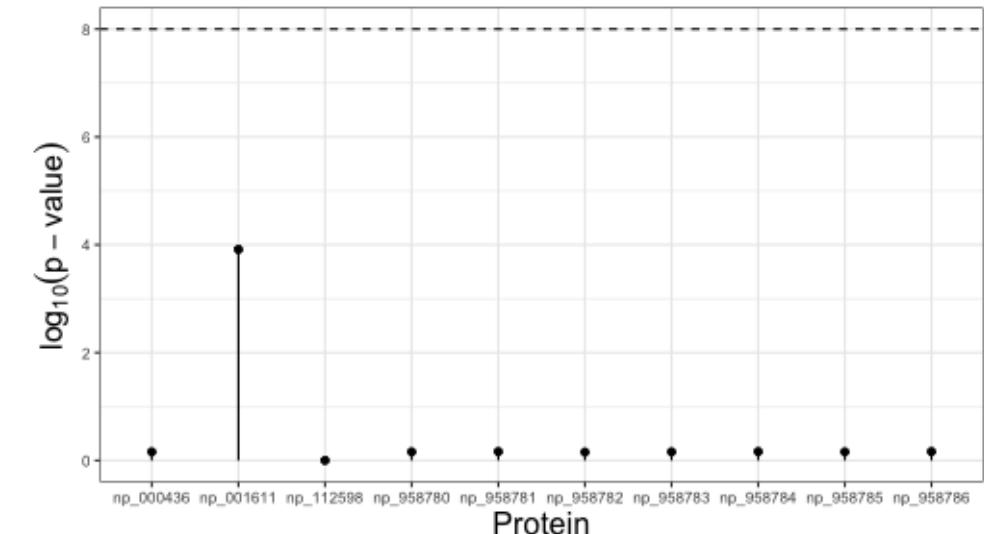


Manhattan plot

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

```
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

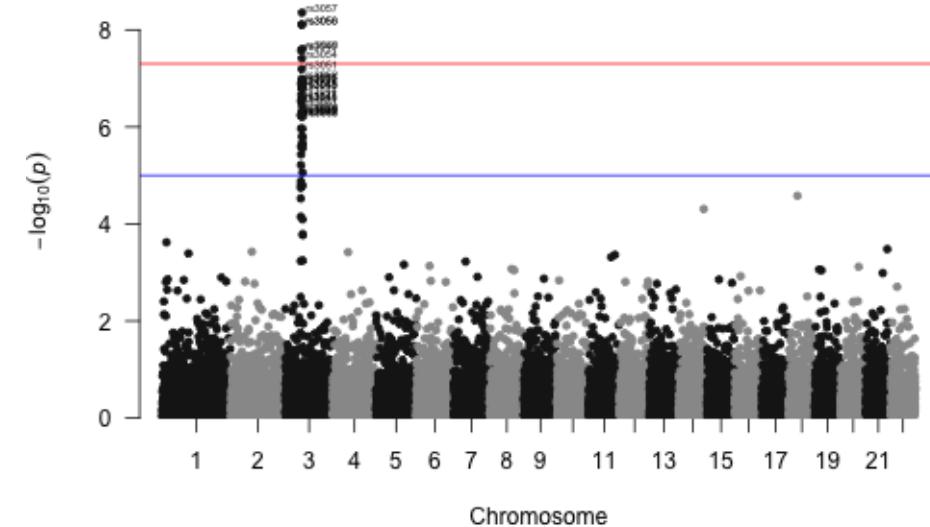
There is a specialized package for doing Manhattan plots and quantile plots for GWAS data

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

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

Manhattan plot

```
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

```
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: 6 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
```

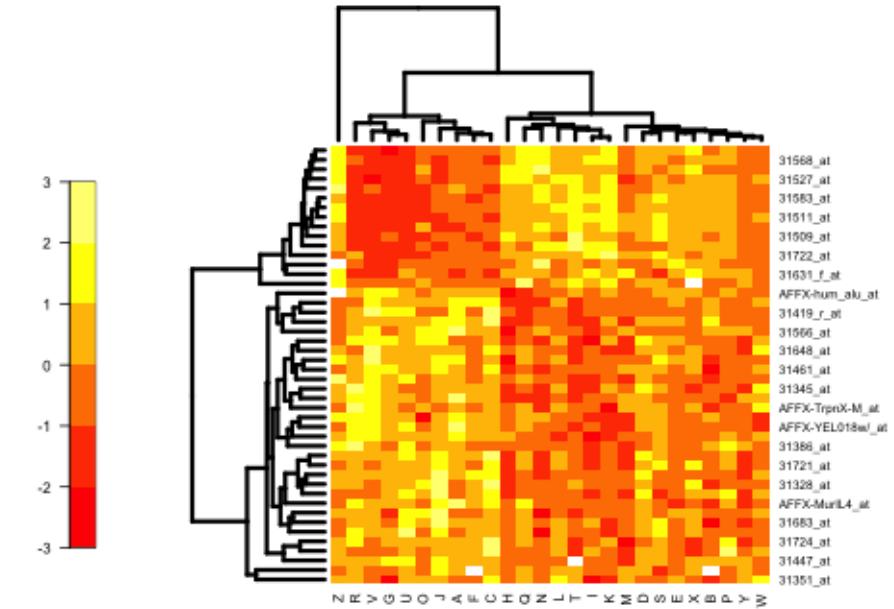
Using Heatplus

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



Using Heatplus

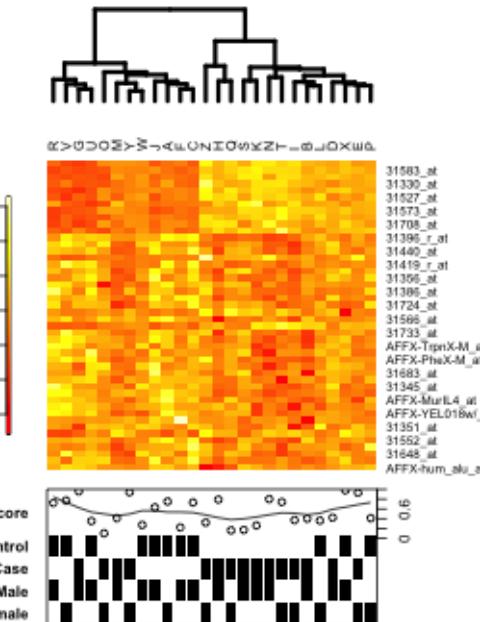
```
corrdist <- function(x) as.dist(1-cor(t(x)))
hclust.avl <- function(x) hclust(x, method='average')
reg2 <- Heatplus:::regHeatmap(exprs(exdat2), legend=2,
                               col=heat.colors,
                               breaks=-3:3,
                               dendrogram =
      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



Using Heatplus

Adding annotations

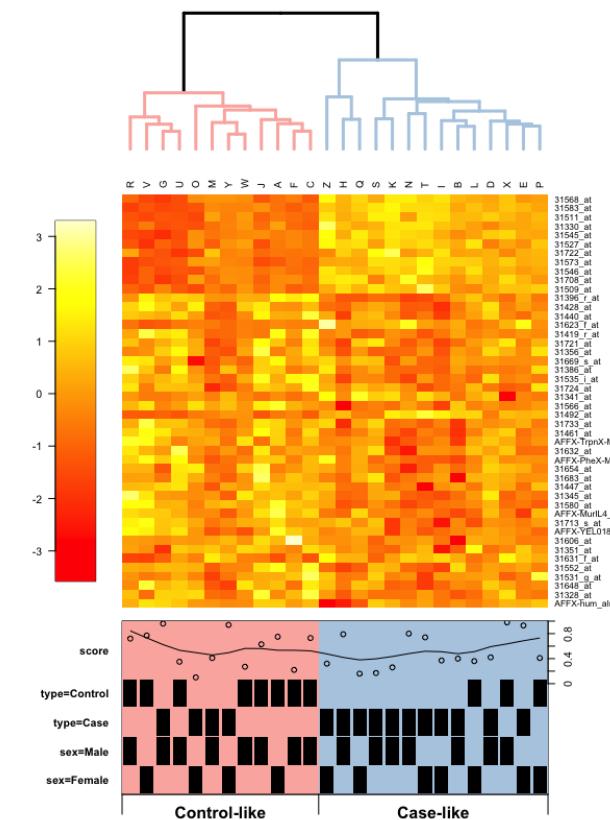
```
ann1 <- Heatplus:::annHeatmap(exprs(exdat2),  
                               ann=pData(exdat2),  
                               col = heat.colors)  
plot(ann1)
```



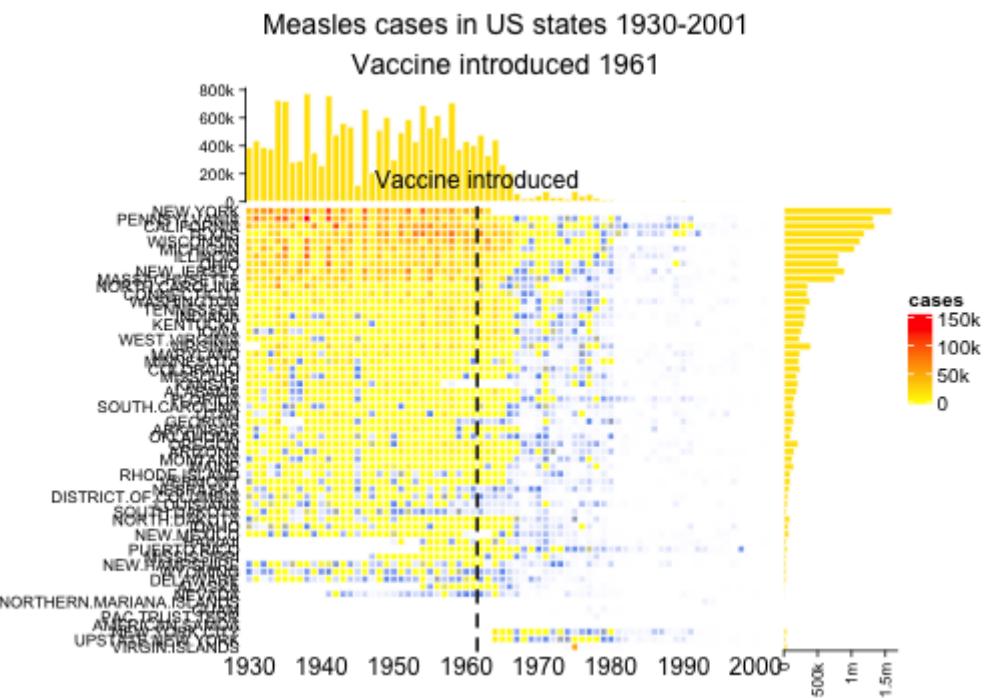
Using Heatplus

Adding annotations

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



Using ComplexHeatmap

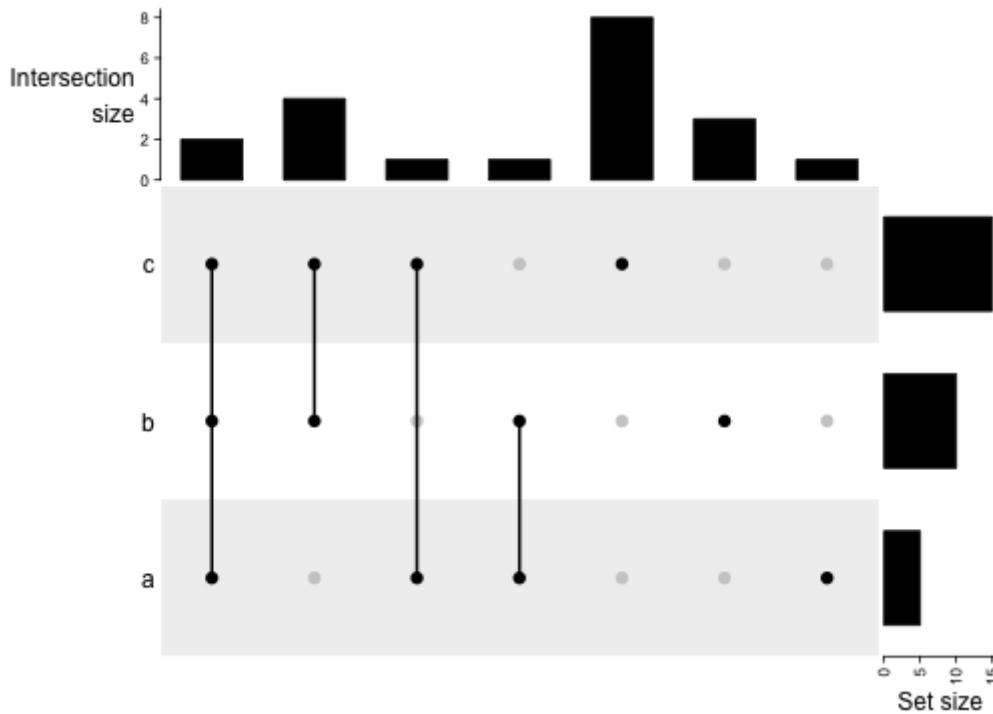


Source code [here](#)

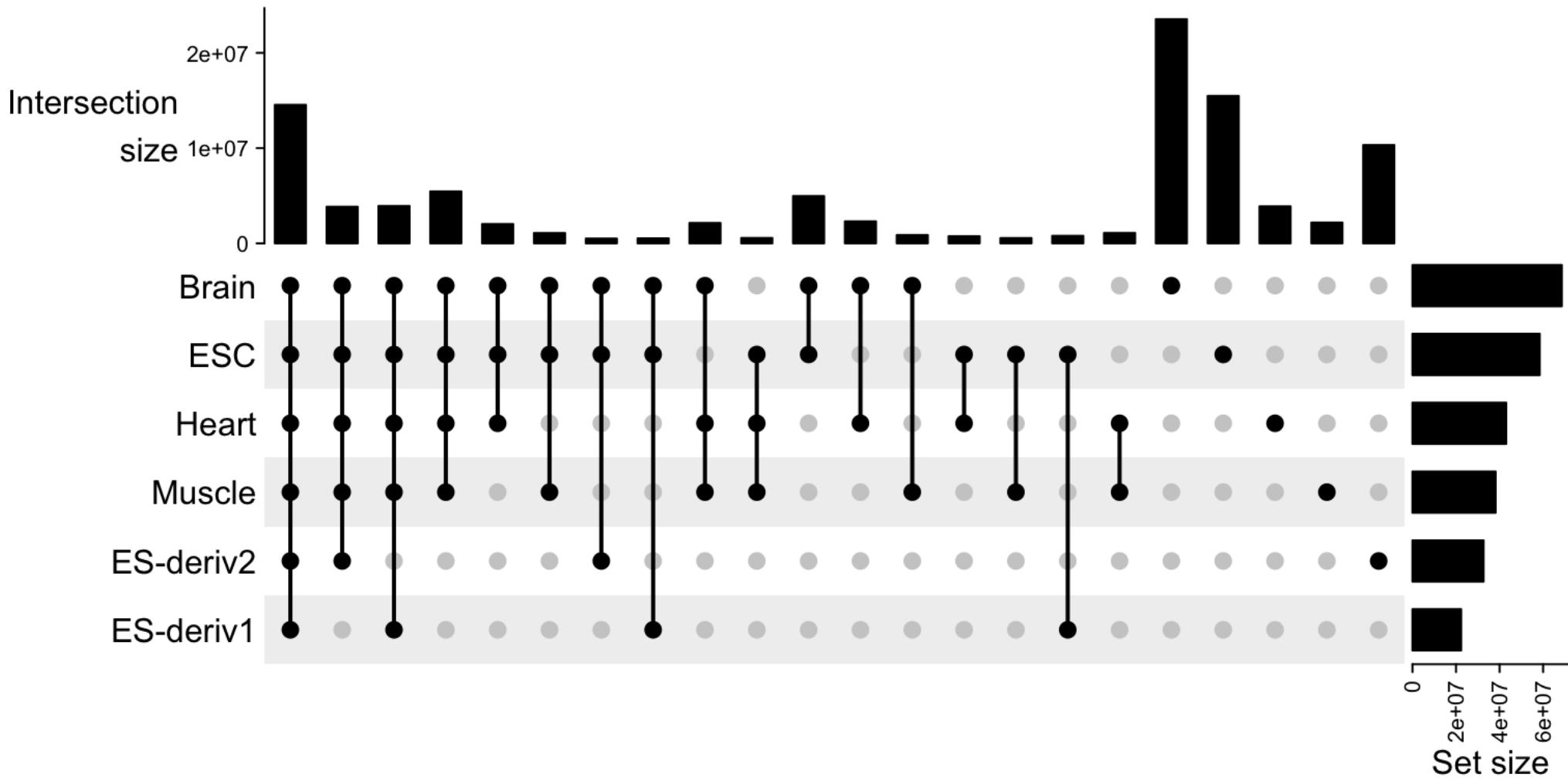
UpSet plots

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

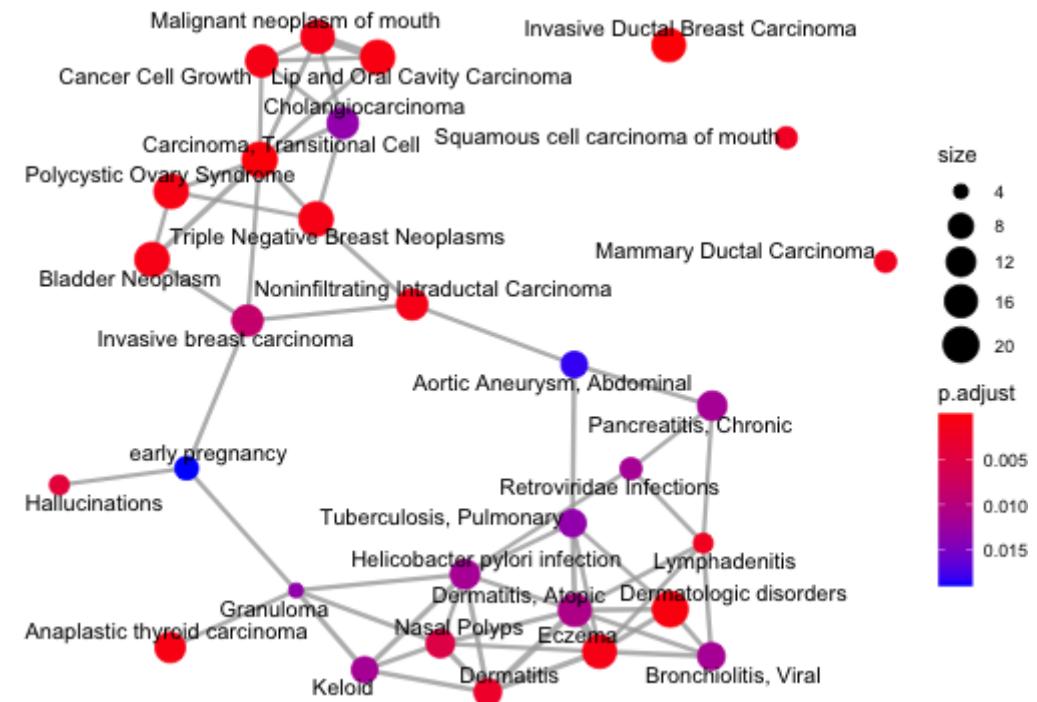


UpSet plots



clusterProfiler

Enrichment network based on GSEA



Playing with Seurat

Example data

```
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)

```
names(pbmc)
```

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

```
slotNames(pbmc)
```

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

Adding QC metrics and plotting

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

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

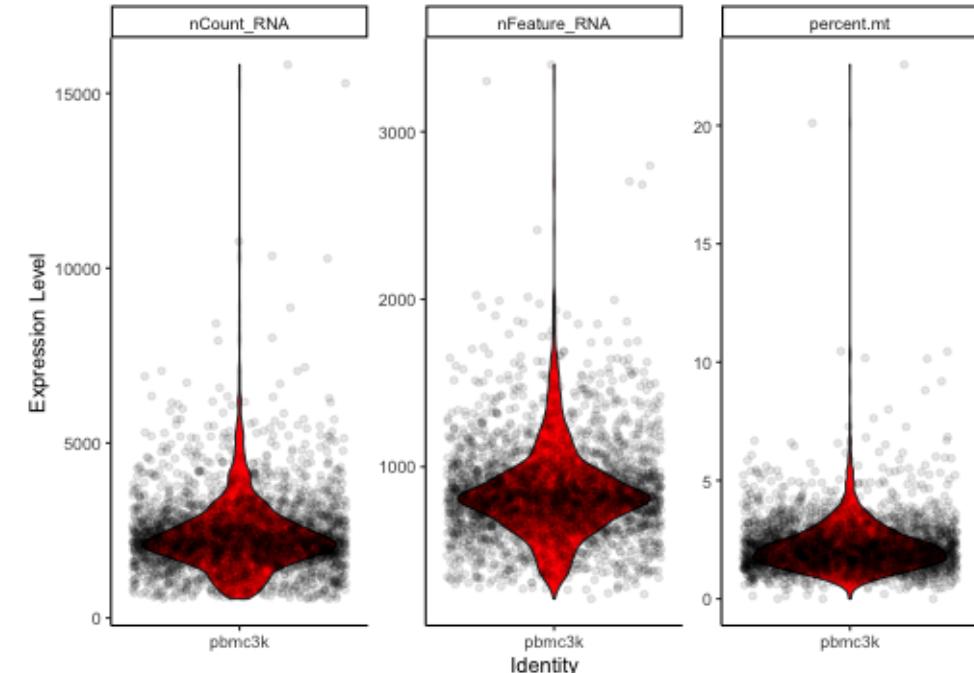
	orig.ident	nCount_RNA	nFeature_RNA	percent.mt
AAACATACAACCAC	pbmc3k	2419	779	3.0177759
AAACATTGAGCTAC	pbmc3k	4903	1352	3.7935958
AAACATTGATCAGC	pbmc3k	3147	1129	0.8897363
AAACCGTGCTTCGG	pbmc3k	2639	960	1.7430845
AAACCGTGTATGCG	pbmc3k	980	521	1.2244898
AAACGCACTGGTAC	pbmc3k	2163	781	1.6643551

Visualizing metrics

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

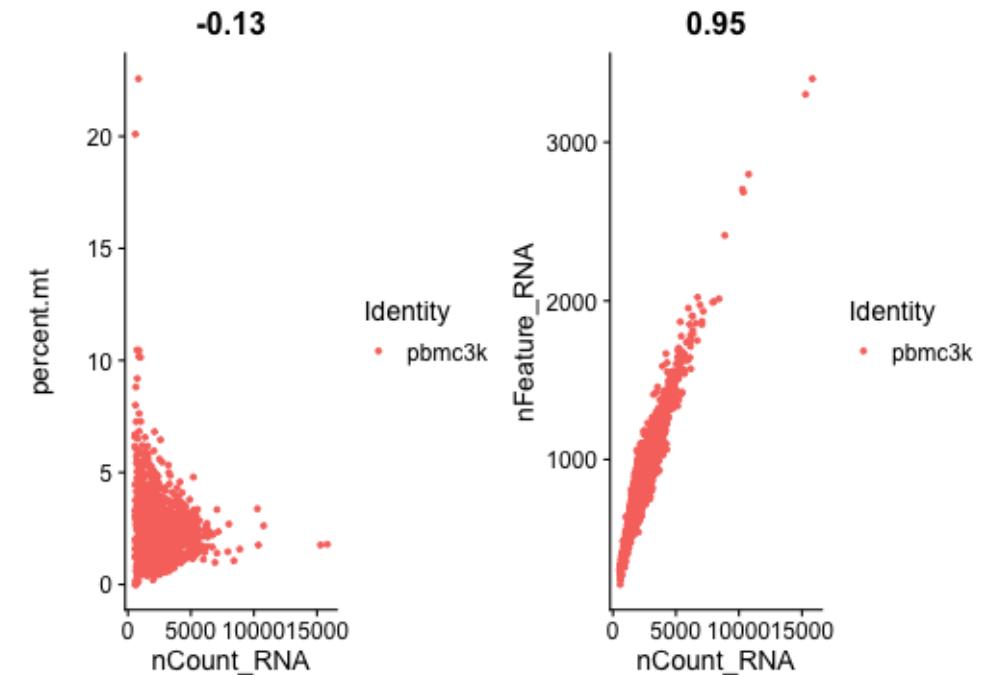
plot_data <- pbmc@meta.data %>%
  tidyverse::gather(variable, value, -orig.ident)

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))
```

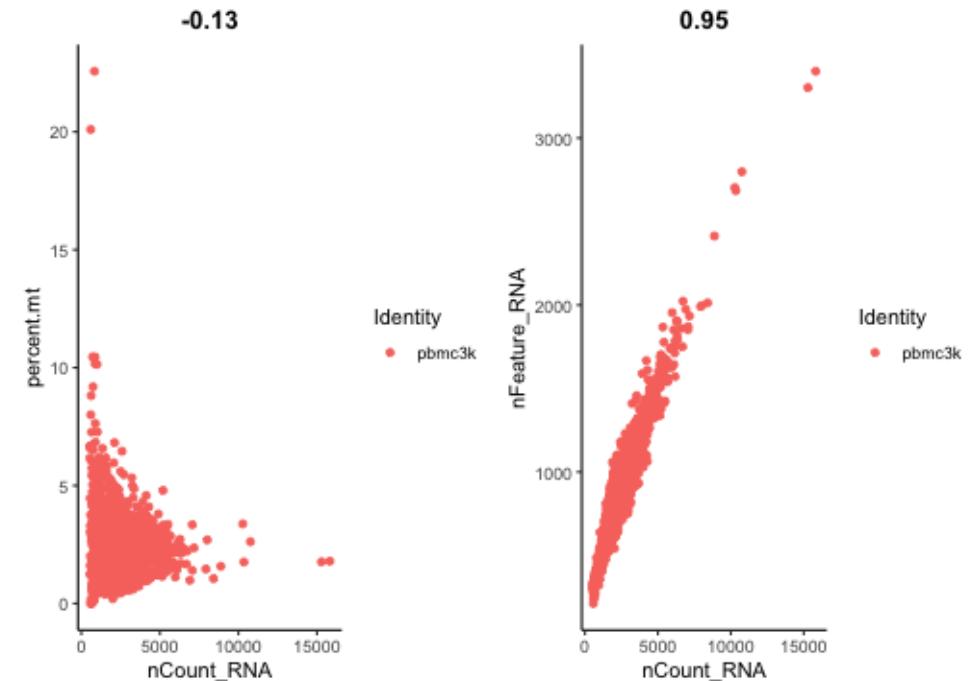


Visualizing feature-feature relationships

```

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

```



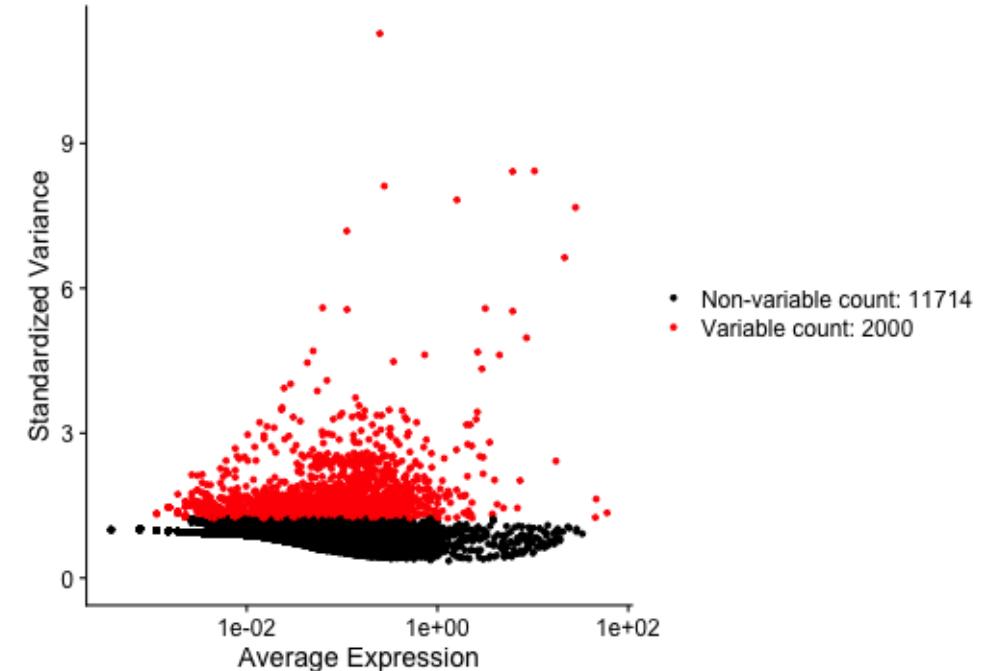
Feature selection

```
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
```

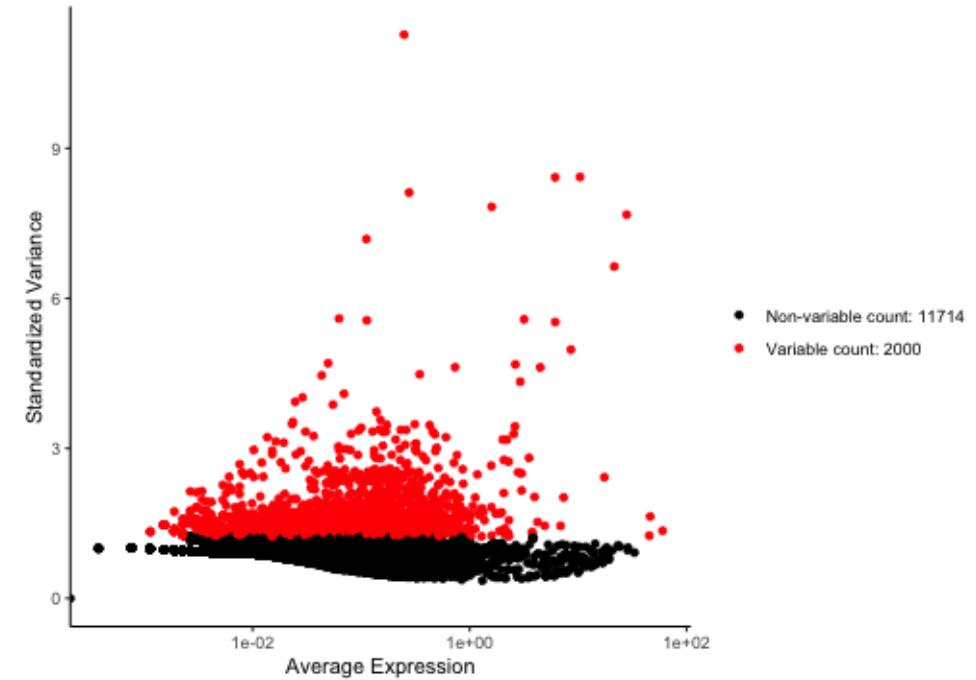


Feature selection

```

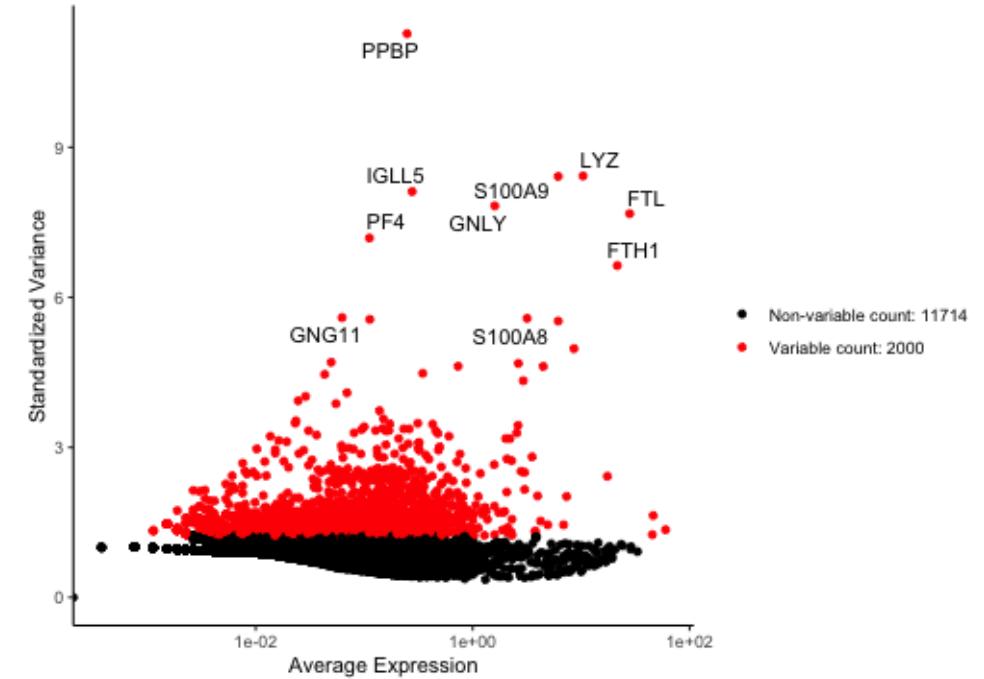
plt_data <- pbmc[['RNA']]@meta.features %>%
  rownames_to_column(var='id')
topvars <- pbmc[['RNA']]@var.features
plt_data <- plt_data %>%
  mutate(indic = ifelse(id %in% topvars,
                        'Variable count',
                        'Non-variable count'))
bl <- plt_data %>%
  dplyr::count(indic) %>%
  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