Joins, and more plotting

Abhijit Dasgupta

Fall, 2019

Goals today

- Learn how to join data sets (merging)
- See how to transform data sets to help our plotting
- Some additional plot types, and customization

Data

This data set is taken from a breast cancer proteome database available here and modified for this exercise.

- Clinical data: CSV|XLSX
- Proteome data: CSV|XLSX

Joins

Putting data sets together

- Quite often, data on individuals lie in different tables
  - Clinical, demographic and bioinformatic data
  - Drug, procedure, and payment data (think Medicare)
  - Personal health data across different healthcare entities

Joining data sets

We already talked about `cbind` and `rbind`:

`cbind`

```
knitr::include_graphics('img/addcol.png')
```

`rbind`

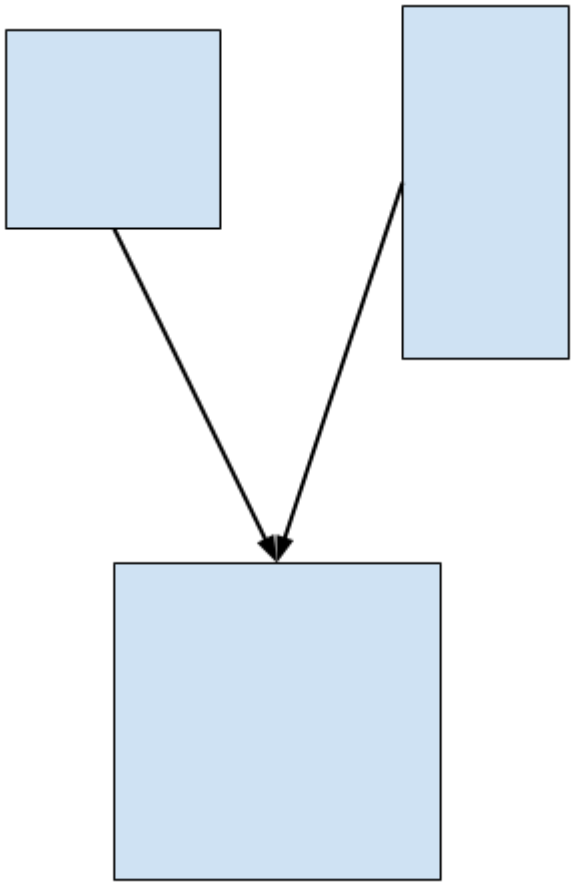
```
knitr::include_graphics('img/addrow.png')
```

Joining data sets

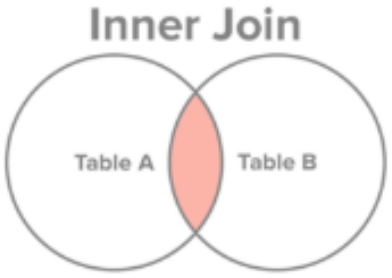
We will talk about more general ways of joining two datasets

We will assume:

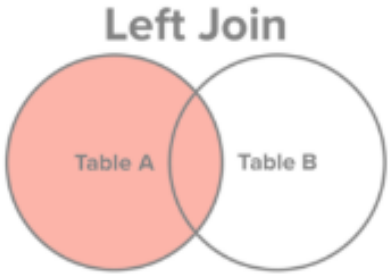
1. We have two rectangular data sets (so `data.frame` or `tibble`)
2. There is at least one variable (column) in common, even if they have different names
    - ID number
    - SSN (Social Security number)
    - Identifiable information



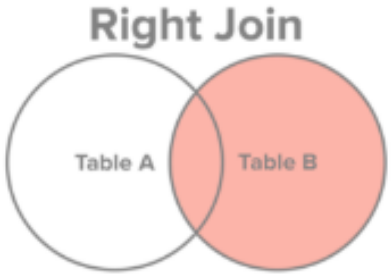
Joining data sets



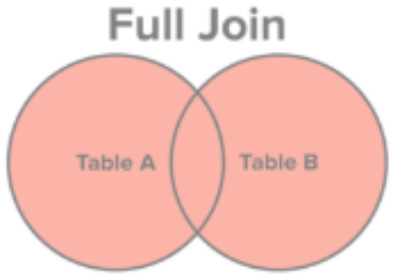
Select all records from Table A and Table B, where the join condition is met.



Select all records from Table A, along with records from Table B for which the join condition is met (if at all).



Select all records from Table B, along with records from Table A for which the join condition is met (if at all).



Select all records from Table A and Table B, regardless of whether the join condition is met or not.

inner_join	left_join	right_join	outer_join

The "join condition" are the common variables in the two datasets, i.e. rows are selected if the values of the common variables in the left dataset matches the values of the common variables in the right dataset

Data example

```
library(readxl)
clinical <- read_excel('data/BreastCancer_Clinical.xlsx',
                       .name_repair='universal')
proteome <- read_excel('data/BreastCancer_Expression.xlsx',
                       .name_repair='universal')
```

```
clinical
```

```
#>  # A tibble: 105 x 30
#>    Complete.TCGA.ID Gender Age.at.Initial.Pathologic
#>    <chr>            <chr>
#>  1 TCGA-A2-A0T2     FEMALE
#>  2 TCGA-A2-A0CM     FEMALE
#>  3 TCGA-BH-A18V     FEMALE
#>    PR.Status HER2.Final.Status Tumor Tumor..T1.Coded
#>    <chr>     <chr>             <chr> <chr>
#>  1 Negative  Negative          T3    T_Other
#>  2 Negative  Negative          T2    T_Other
#>  3 Negative  Negative          T2    T_Other
#>    Metastasis Metastasis.Coded AJCC.Stage Converted.S
#>    <chr>      <chr>            <chr>      <chr>
#>  1 M1         Positive         Stage IV   No_Conver
#>  2 M0         Negative         Stage IIA  Stage II
#>  3 M0         Negative         Stage IIB  No_Conver
#>    Vital.Status Days.to.Date.of.Last.Contact Days.to.
#>    <chr>                                    <dbl>
```

```
proteome
```

```
#>  # A tibble: 83 x 11
#>    TCGA_ID      NP_958782 NP_958785 NP_958786 NP_0
#>    <chr>            <dbl>     <dbl>     <dbl>
#>  1 TCGA-AO-A12D      1.10      1.11      1.11
#>  2 TCGA-C8-A131      2.61      2.65      2.65
#>  3 TCGA-AO-A12B    -0.660    -0.649    -0.654    -
#>    NP_958783 NP_958784 NP_112598 NP_001611
#>        <dbl>     <dbl>     <dbl>     <dbl>
#>  1     1.11      1.11     -1.52      0.483
#>  2     2.65      2.65      3.91     -1.05
#>  3    -0.649    -0.649    -0.618     1.22
#>  # … with 80 more rows
```

Data example

```
library(readxl)
clinical <- read_excel('data/BreastCancer_Clinical.xlsx',
                       .name_repair = 'universal')
proteome <- read_excel('data/BreastCancer_Expression.xlsx',
                         .name_repair = 'universal')
```

```
clinical[,1:2]
```

```
#>  # A tibble: 105 x 2
#>    Complete.TCGA.ID Gender
#>    <chr>            <chr>
#>  1 TCGA-A2-A0T2     FEMALE
#>  2 TCGA-A2-A0CM     FEMALE
#>  3 TCGA-BH-A18V     FEMALE
#>  # … with 102 more rows
```

```
proteome[,1:2]
```

```
#>  # A tibble: 83 x 2
#>    TCGA_ID      NP_958782
#>    <chr>            <dbl>
#>  1 TCGA-AO-A12D     1.10
#>  2 TCGA-C8-A131     2.61
#>  3 TCGA-AO-A12B    -0.660
#>  # … with 80 more rows
```

We see that both have the same ID variable, but with different names and different orders

Data example

Let's make sure that the ID's are truly IDs, i.e. each row has a unique value

```
length(unique(clinical$Complete.TCGA.ID)) == nrow(clinical)
```

```
#>  [1] TRUE
```

```
length(unique(proteome$TCGA_ID)) == nrow(proteome)
```

```
#>  [1] FALSE
```

Data example

For convenience we'll keep the first instance for each ID in the `proteome` data

```
proteome <- proteome %>% filter(!duplicated(TCGA_ID))
```

> `duplicated` = TRUE if a previous row contains the same value

```
length(unique(proteome$TCGA_ID)) == nrow(proteome)
```

```
#>  [1] TRUE
```

Inner join

```
common_rows <- inner_join(clinical[,1:6], proteome, by=c('Complete.TCGA.ID'='TCGA_ID'))
```

```
#>  # A tibble: 77 x 16
#>    Complete.TCGA.ID Gender Age.at.Initial.Pathologic.Diagnosis ER.Status
#>    <chr>            <chr>                                <dbl> <chr>
#>  1 TCGA-A2-A0CM     FEMALE                                  40 Negative
#>  2 TCGA-BH-A18Q     FEMALE                                  56 Negative
#>  3 TCGA-A7-A0CE     FEMALE                                  57 Negative
#>    PR.Status HER2.Final.Status NP_958782 NP_958785 NP_958786 NP_000436
#>    <chr>     <chr>                <dbl>     <dbl>     <dbl>     <dbl>
#>  1 Negative  Negative             0.683     0.694     0.698     0.687
#>  2 Negative  Negative             0.195     0.215     0.215     0.205
#>  3 Negative  Negative            -1.12     -1.12     -1.12     -1.13
#>    NP_958781 NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
#>        <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
#>  1     0.687     0.698     0.698     0.698     -2.65    -0.984
#>  2     0.215     0.215     0.215     0.215     -1.04    -0.517
#>  3    -1.13     -1.12     -1.12     -1.12       2.24    -2.58
#>  # … with 74 more rows
```

Note that we have all the columns from both datasets, but only 77 rows, which is the common set of IDs from the two datasets

> If you don't include the `by` option, R will attempt to match values of any columns with the same names

Left join

```
left_rows <- left_join(clinical[,1:6], proteome, by=c('Complete.TCGA.ID'='TCGA_ID'))
```

```
#>  # A tibble: 105 x 16
#>    Complete.TCGA.ID Gender Age.at.Initial.Pathologic.Diagnosis ER.Status
#>    <chr>            <chr>                                 <dbl> <chr>
#>  1 TCGA-A2-A0T2     FEMALE                                   66 Negative
#>  2 TCGA-A2-A0CM     FEMALE                                   40 Negative
#>  3 TCGA-BH-A18V     FEMALE                                   48 Negative
#>    PR.Status HER2.Final.Status NP_958782 NP_958785 NP_958786 NP_000436
#>    <chr>     <chr>                <dbl>     <dbl>     <dbl>     <dbl>
#>  1 Negative  Negative            NA        NA        NA        NA
#>  2 Negative  Negative             0.683     0.694     0.698     0.687
#>  3 Negative  Negative            NA        NA        NA        NA
#>    NP_958781 NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
#>        <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
#>  1    NA        NA        NA        NA         NA       NA
#>  2     0.687     0.698     0.698     0.698     -2.65    -0.984
#>  3    NA        NA        NA        NA         NA       NA
#>  # … with 102 more rows
```

We get 105 rows, which is all the rows of `clinical`, combined with the rows of `proteome` with common IDs. The rest of the rows get `NA` for the proteome columns.

Right join

```
right_rows <- right_join(clinical[,1:6], proteome, by=c('Complete.TCGA.ID'='TCGA_ID'))
```

```
#>  # A tibble: 80 x 16
#>    Complete.TCGA.ID Gender Age.at.Initial.Pathologic.Diagnosis ER.Status
#>    <chr>            <chr>                                <dbl> <chr>
#>  1 TCGA-AO-A12D     FEMALE                                  43 Negative
#>  2 TCGA-C8-A131     FEMALE                                  82 Negative
#>  3 TCGA-AO-A12B     FEMALE                                  63 Positive
#>    PR.Status HER2.Final.Status NP_958782 NP_958785 NP_958786 NP_000436
#>    <chr>     <chr>                 <dbl>     <dbl>     <dbl>     <dbl>
#>  1 Negative  Positive              1.10      1.11      1.11      1.11
#>  2 Negative  Negative              2.61      2.65      2.65      2.65
#>  3 Positive  Negative             -0.660    -0.649    -0.654    -0.632
#>    NP_958781 NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
#>        <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
#>  1     1.12      1.11      1.11      1.11     -1.52      0.483
#>  2     2.65      2.65      2.65      2.65      3.91     -1.05
#>  3    -0.640    -0.654    -0.649    -0.649    -0.618     1.22
#>  # … with 77 more rows
```

Here we get 80 rows, which is all the rows of `proteome`, along with the rows of `clinical` with common IDs, but with the columns of `clinical` appearing first.

Outer/Full Join

```
full_rows <- full_join(clinical[,1:6], proteome, by=c('Complete.TCGA.ID'='TCGA_ID'))
```

```
#>  # A tibble: 108 x 16
#>    Complete.TCGA.ID Gender Age.at.Initial.Pathologic.Diagnosis ER.Status
#>    <chr>            <chr>                                 <dbl> <chr>
#>  1 TCGA-A2-A0T2     FEMALE                                   66 Negative
#>  2 TCGA-A2-A0CM     FEMALE                                   40 Negative
#>  3 TCGA-BH-A18V     FEMALE                                   48 Negative
#>    PR.Status HER2.Final.Status NP_958782 NP_958785 NP_958786 NP_000436
#>    <chr>     <chr>                <dbl>     <dbl>     <dbl>     <dbl>
#>  1 Negative  Negative            NA        NA        NA        NA
#>  2 Negative  Negative             0.683     0.694     0.698     0.687
#>  3 Negative  Negative            NA        NA        NA        NA
#>    NP_958781 NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
#>        <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
#>  1    NA        NA        NA        NA         NA       NA
#>  2     0.687     0.698     0.698     0.698     -2.65    -0.984
#>  3    NA        NA        NA        NA         NA       NA
#>  # … with 105 more rows
```

Here we obtain 108 rows and 16 columns. So we've expanded the data in both rows and columns, putting missing values in where needed.

Joins

In each of `inner_join`, `left_join`, `right_join` and `full_join`, the number of columns always increases

There are also two joins where the number of columns don't increase. They aren't really "joins" in that sense, but really fancy filters on a dataset

```
tbl <- tribble(~Join,~Use,~Description,
               "semi_join", "semi_join(A,B)", "Keep rows in A where ID matches some ID value in B",
               'anti_join', 'anti_join(A,B)', 'Keep rows in A where ID does NOT match any ID value in B')
knitr::kable(tbl, format='html')
```

Join	Use	Description
semi_join	semi_join(A,B)	Keep rows in A where ID matches some ID value in B
anti_join	anti_join(A,B)	Keep rows in A where ID does NOT match any ID value in B

These just filter the rows of `A` without adding any columns of `B`.

Are there protein expression differences between ER +ve and ER -ve breast cancers

Create analytic dataset

```
final_data <- clinical %>%
  inner_join(proteome, by=c("Complete.TCGA.ID"="TCGA_ID")) %>%
  filter(Gender =='FEMALE') %>%
  select(Complete.TCGA.ID, Age.at.Initial.Pathologic.Diagnosis, ER.Status,
         starts_with("NP")) # grabs all the protein data
```

```
#>  # A tibble: 75 x 13
#>    Complete.TCGA.ID Age.at.Initial.Pathologic.Diagnosis ER.Status NP_958782
#>    <chr>                                          <dbl> <chr>         <dbl>
#>  1 TCGA-A2-A0CM                                      40 Negative      0.683
#>  2 TCGA-BH-A18Q                                      56 Negative      0.195
#>  3 TCGA-A7-A0CE                                      57 Negative     -1.12
#>    NP_958785 NP_958786 NP_000436 NP_958781 NP_958780 NP_958783 NP_958784
#>        <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
#>  1     0.694     0.698     0.687     0.687     0.698     0.698     0.698
#>  2     0.215     0.215     0.205     0.215     0.215     0.215     0.215
#>  3    -1.12     -1.12     -1.13     -1.13     -1.12     -1.12     -1.12
#>    NP_112598 NP_001611
#>        <dbl>     <dbl>
#>  1     -2.65    -0.984
#>  2     -1.04    -0.517
#>  3      2.24    -2.58
#>  # … with 72 more rows
```

Protein-specific graphs

We want to graph each protein separately, while maintaining alignment with ER status and age.

The R trick is to make this wide table long, so you can split on the rows

```
final_data2 <- final_data %>% tidyr::gather(protein, expression, starts_with('NP')) %>%
  arrange(Complete.TCGA.ID)
```

```
#>  # A tibble: 750 x 5
#>    Complete.TCGA.ID Age.at.Initial.Pathologic.Diagnosis ER.Status protein
#>    <chr>                                          <dbl> <chr>     <chr>
#>  1 TCGA-A2-A0CM                                      40 Negative  NP_958782
#>  2 TCGA-A2-A0CM                                      40 Negative  NP_958785
#>  3 TCGA-A2-A0CM                                      40 Negative  NP_958786
#>    expression
#>         <dbl>
#>  1      0.683
#>  2      0.694
#>  3      0.698
#>  # … with 747 more rows
```

Facetted plots (Trellis graphics)

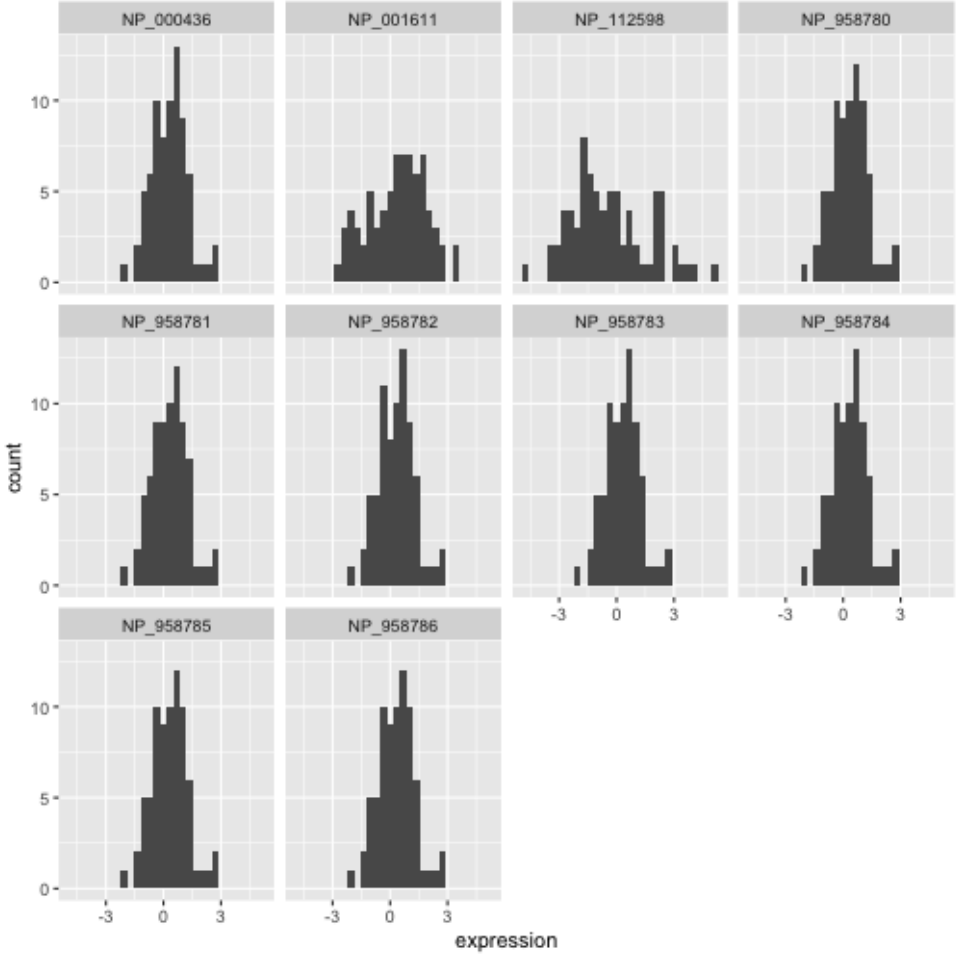
Realize that the `group` or `color` or `fill` or similar modifications of geoms are really splitting the data based on the grouping variable and then plotting the data from each of the divided datasets

Split-apply-combine

Facetted plots (Trellis graphics)

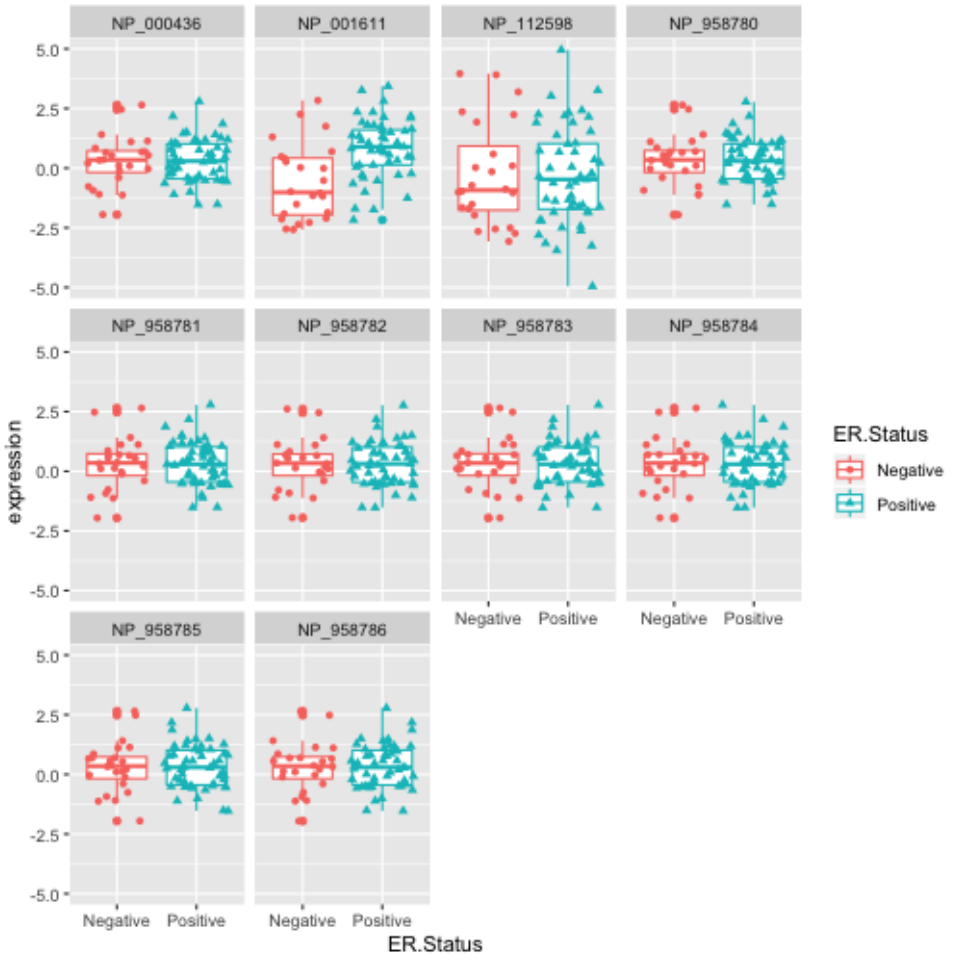
```
ggplot(final_data2, aes(x = expression)) +
  geom_histogram() +
  facet_wrap(~protein)
```

Here we're splitting the **rows** of the data based on the value of `protein` (which are the protein names), and the plotting a histogram of expression for each subgroup, and then putting all the plots back together



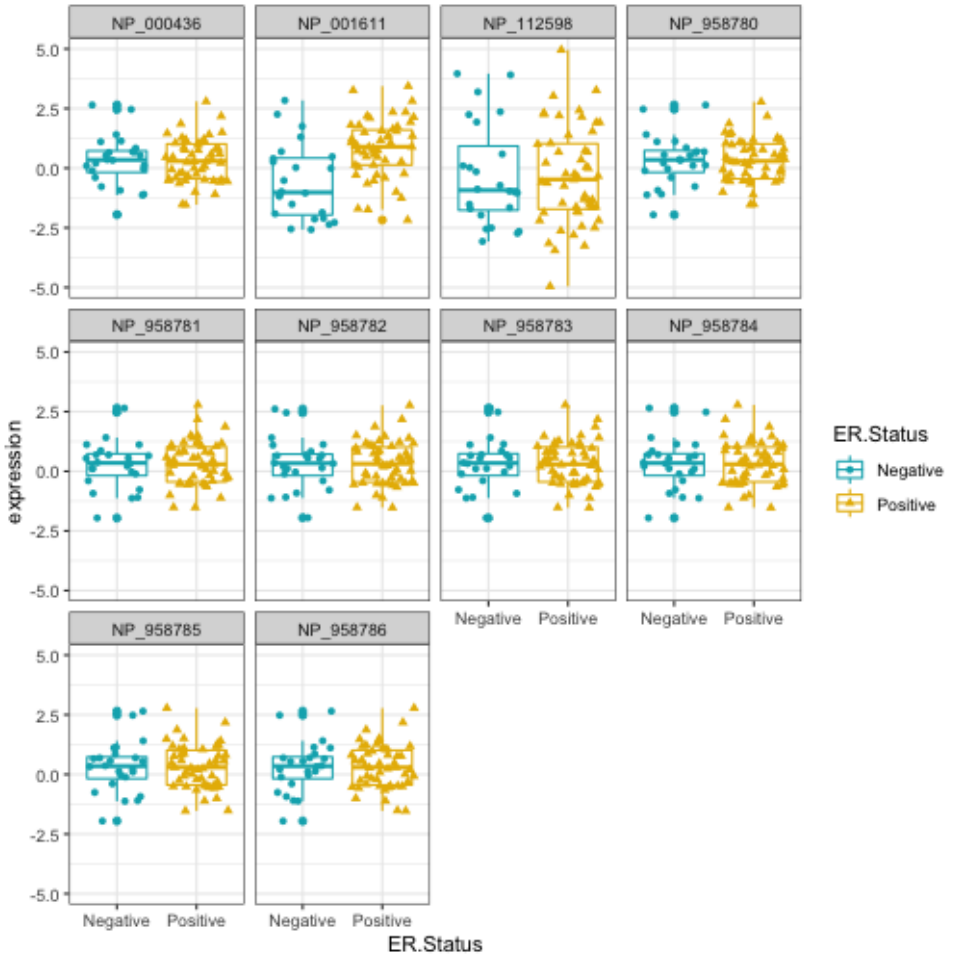
Facetted plots (Trellis graphics)

```
p <- ggplot(final_data2,
       aes(x = ER.Status, y = expression,
           color = ER.Status,
           shape=ER.Status)) +
  geom_boxplot() +
  geom_jitter() +
  facet_wrap(~protein)

p
```



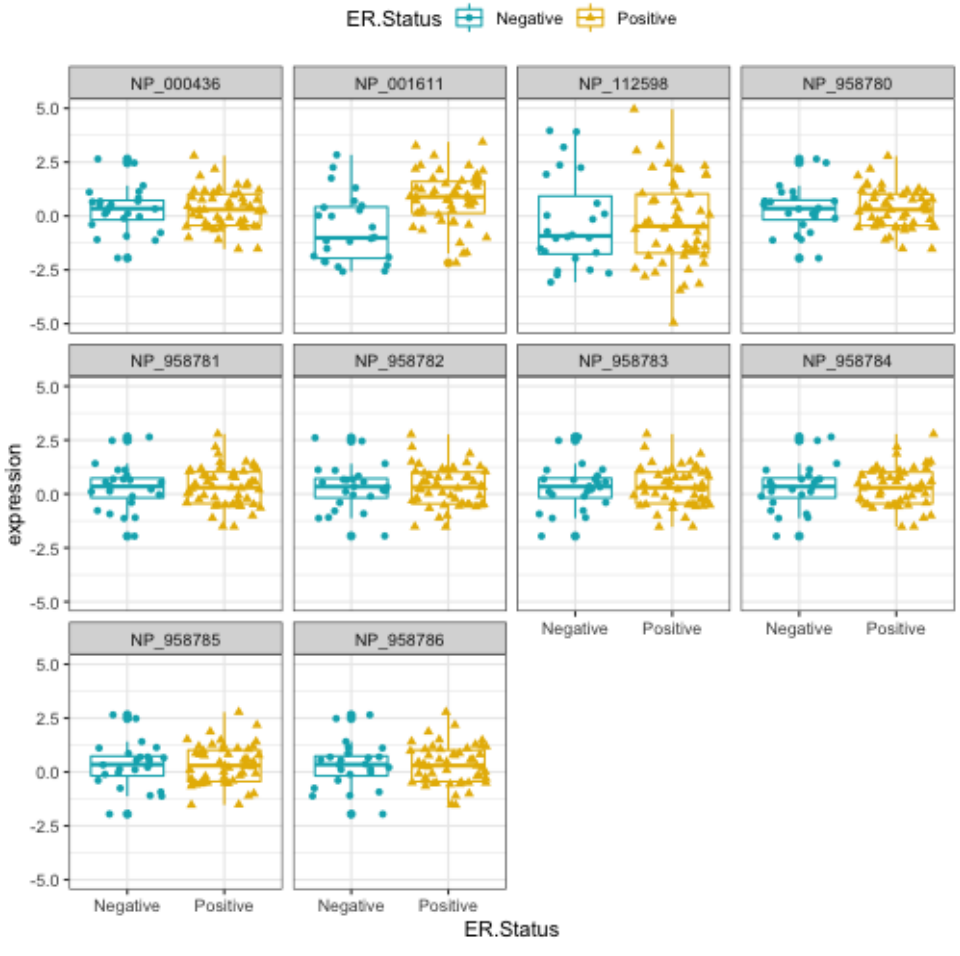
Facetted plots (Trellis graphics)

```
p <- ggplot(final_data2,
       aes(x = ER.Status, y = expression,
           color = ER.Status,
           shape=ER.Status)) +
  geom_boxplot() +
  geom_jitter()+
  facet_wrap(~protein) +
  theme_bw() +
  scale_color_manual(values = c("#00AFBB", "#E7B800"))

p
```



Facetted plots (Trellis graphics)

```
p <- ggplot(final_data2,
       aes(x = ER.Status, y = expression,
           color = ER.Status,
           shape=ER.Status)) +
  geom_boxplot() +
  geom_jitter()+
  facet_wrap(~protein) +
  theme_bw() +
  scale_color_manual(values = c("#00AFBB", "#E7B800")) +
  theme(legend.position='top')

p
```

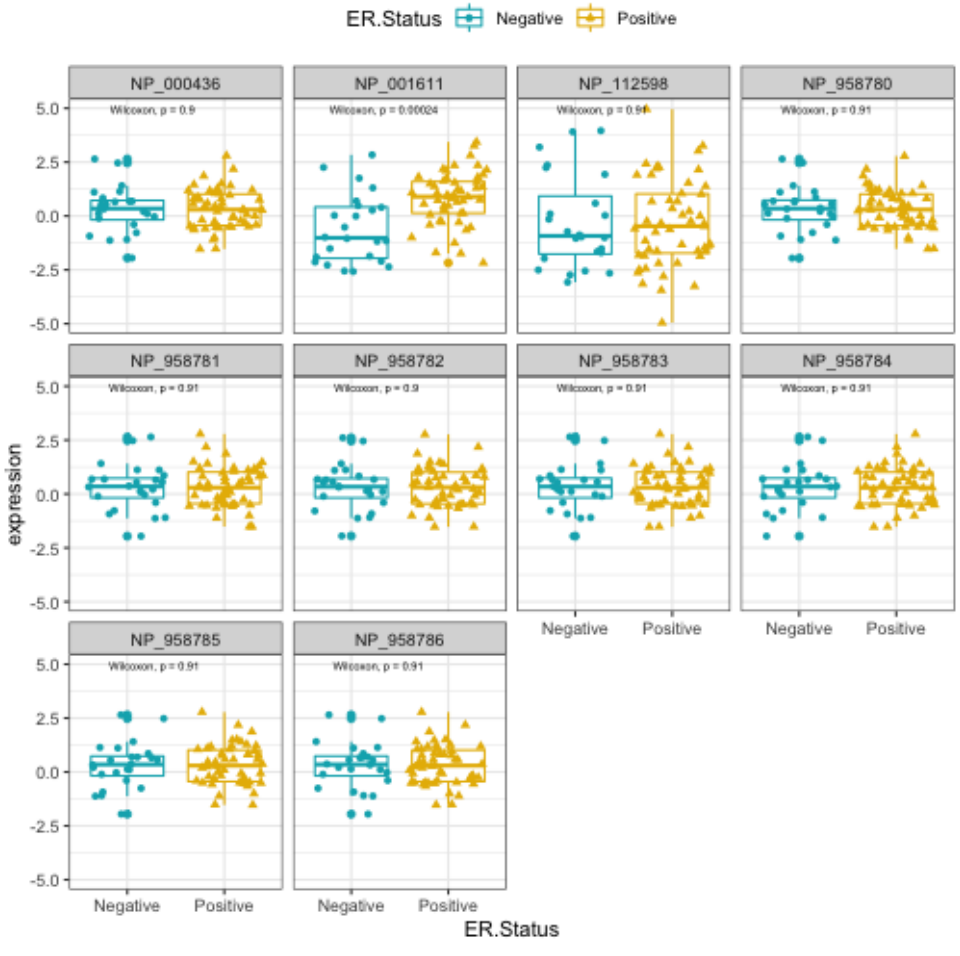


Facetted plots (Trellis graphics)

```
p <- ggplot(final_data2,
       aes(x = ER.Status, y = expression,
           color = ER.Status,
           shape=ER.Status)) +
  geom_boxplot() +
  geom_jitter()+
  facet_wrap(~protein) +
  scale_color_manual(values = c("#00AFBB", "#E7B800")
  theme_bw() +
  theme(legend.position='top')
p + ggpubr::stat_compare_means(size = 2, na.rm=T)
```

Adding statistics

The statistics are computed on each subgroup, so proving that this is really an example of split-apply-combine



> Design is choice. The theory of the visual display of quantitative information consists of principles that generate design options and that guide choices among options. The principles should not be applied rigidly or in a peevish spirit; they are not logically or mathematically certain; and it is better to violate any principle than to place graceless or inelegant marks on paper. Most principles of design should be greeted with some skepticism, for word authority can dominate our vision, and we may come to see only through the lenses of word authority rather than with our own eyes.
>
> --- Edward Tufte, *The Visual Display of Quantitative Data*

Considerations

Tufte's Principles of Graphical Integrity

1. Show data variation, not design variation
2. Do not use graphics to quote data out of context
3. Use clear, detailed, thorough labelling.
4. Representation of numbers should be directly proportional to numerical quantities
5. Don't use more dimensions than the data require

Tufte's Principles of Graphical Integrity

1. Show data variation, not design variation
    - Don't get fancy, let the data speak
2. Do not use graphics to quote data out of context
    - Maintain accuracy
3. Use clear, detailed, thorough labelling.
    - Use annotations to make your point
4. Representation of numbers should be directly proportional to numerical quantities
    - This is essential for fair representation
5. Don't use more dimensions than the data require
    - Be appropriate in use of 3D graphics, for example

Tufte's Fundamental Principles of Design

1. Show comparisons
2. Show causality
3. Use multivariate data
4. Completely integrate modes (like text, images, numbers)
5. Establish credibility
6. Focus on content

7 Basic Rules for Making Charts and Graphs

1. Check the data
2. Explain encodings
3. Label axes
4. Include units
5. Keep your geometry in check
6. Include your sources
7. Consider your audience

Nathan Yau, Flowing Data
https://flowingdata.com/2010/07/22/7-basic-rules-for-making-charts-and-graphs/

Presentations and formal papers

Work on formating

- Fonts
- Colors
- Glyphs
- Labeling
- Panels/Facets and organization

Check any particular requirements from publisher

- Resolution
- File type
- Typically TIFF at 300dpi is required

A note on TIFFs

- R creates graphs at 72dpi by default
- I've had most success creating PDFs or SVGs and converting them
- Adobe Acrobat Pro will save PDFs to TIFFs, as will Adobe Illustrator for SVGs
- Make sure you use LZW compression, otherwise you'll fail file size requirements

Using Ghostscript

```
gs -q -dNOPAUSE -dBATCH -sDEVICE=tiff24nc -sCompression=lzw -r300x300 -sOutputFile=<output file> <input file>
```

On Windows, replace `gs` with `gswin32c`

Using ggplot (appears to give right DPI, but doesn't seem to compress, so files are too big)

```
ggsave('out.tiff', units='in', width=4, height=4, compression = 'lzw', dpi = 300)
```

File Formats

In general, you will generate a graphics file for your plot by calling a function which will have the same name as the desired file format (svg, pdf, jpeg, etc).

```
library(ggplot2, quietly = TRUE)
svg(filename="myPlot.svg", width = 3, height=3, pointsize = 8)
ggplot(cars, aes(x=speed)) + geom_density()
dev.off()
```

The second command opens a file for output, the third generates the plot, and the fourth command (dev.off()) finishes writing the file and closes it. By default, graphics go to the last graphics "device" you created and dev.off closes the last graphics device created.

A shortcut for this in `ggplot2` is

```
ggplot(cars, aes(x = speed)) + geom_density()
ggsave('myPlot.svg') # Type is picked up from last 3 letters after .
```

Vector versus Raster (pixel) graphics

```
pdf(file = "test.pdf", width=3, height=3)
ggplot(cars, aes(x=speed)) + geom_density()
dev.off()
png(filename = "test.png", width=3, height=3, units = "in",res = 100)
ggplot(cars, aes(x=speed)) + geom_density()
dev.off()
```

File Size

test.pdf: 9KB

test.png: 16KB

Raster graphics take more space but give worse results! In general, you will be better off using vector graphcics when makeing plots and graphs.

Error Bars

You create error bars in ggplot by adding an extra plot argument, geom_errorbar. You specify the top and bottom "y" position of the error bar, and optionally the width. You will have to calculate where the error bars should be and choose what they should represent (standard deviation, standard error, 95% confidence interval).

Calculating Standard Error

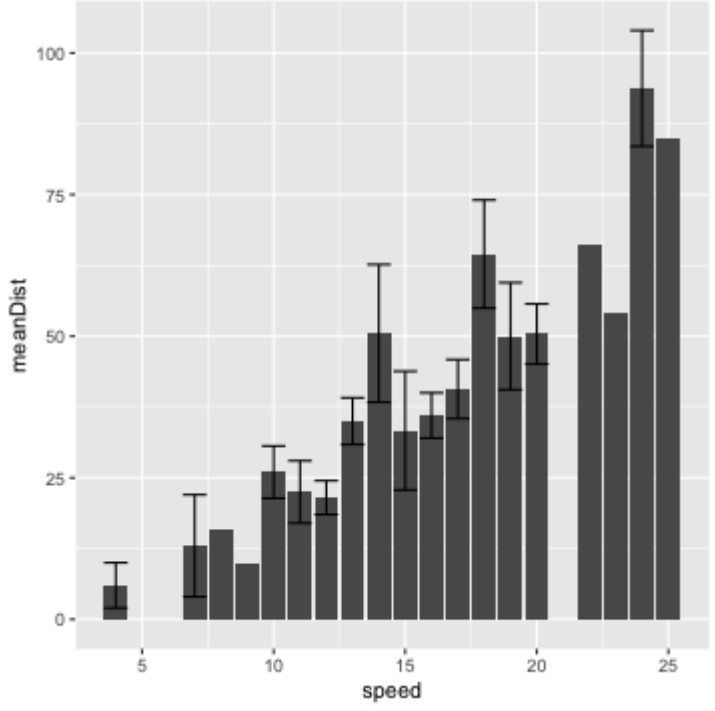
A standard method to achieve error bars would be to calculate standard error and store the value in a column.
Be careful! There is a standard error function in R (stderr) that has nothing to do with standard errors! But you can define your on function to calculate it or use a package that supplies the standard error function.

```
sem <- function(x) sqrt(var(x, na.rm=T)/sum(!is.na(x)))
cars %>% group_by(speed) %>%
  summarize(meanDist = mean(dist),
            semDist = sem(dist))
```

```
#>  # A tibble: 19 x 3
#>     speed meanDist semDist
#>     <dbl>    <dbl>   <dbl>
#>   1     4      6      4
#>   2     7     13      9
#>   3     8     16     NA
#>   4     9     10     NA
#>   5    10     26      4.62
#>   6    11     22.5    5.5
#>   7    12     21.5    2.99
#>   8    13     35      4.12
#>   9    14     50.5   12.1
#>  10    15     33.3   10.5
#>  11    16     36      4
#>  12    17     40.7    5.21
#>  13    18     64.5    9.54
```

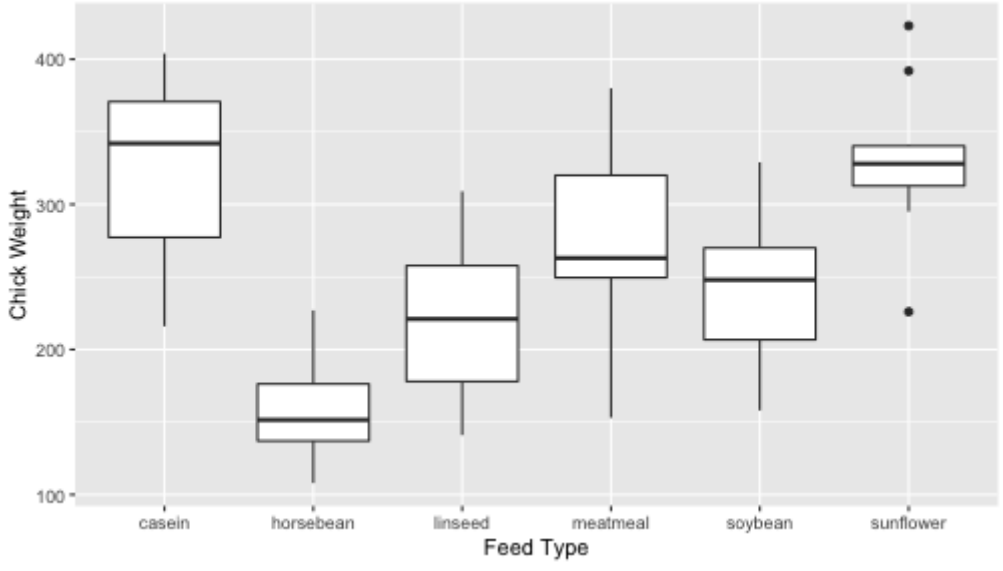
Error Bar Example

```
sem <- function(x) sqrt(var(x, na.rm=T)/sum(!is.na(x)))
cars %>% group_by(speed) %>% summarize(meanDist = mean(dist),
                                       semDist = sem(dist)) %>%
ggplot(aes(x=speed, y=meanDist)) + geom_bar(stat="identity") +
  geom_errorbar(aes(ymin=meanDist - semDist, ymax= meanDist+semDist))
```



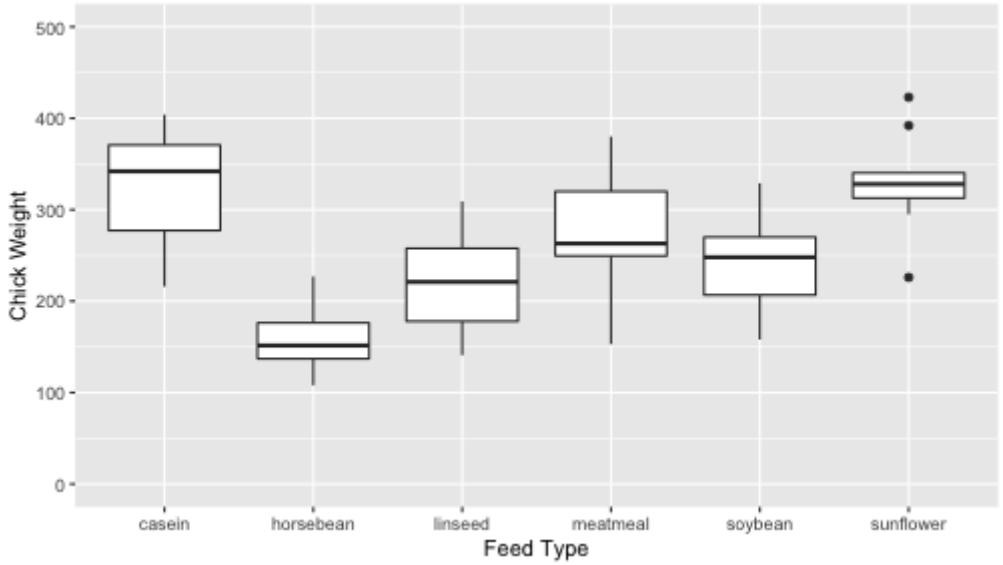
Changing the axis title

```
ggplot(chickwts, aes(x=feed, y=weight)) + geom_boxplot() +
  labs(x = 'Feed Type', y = 'Chick Weight')
```



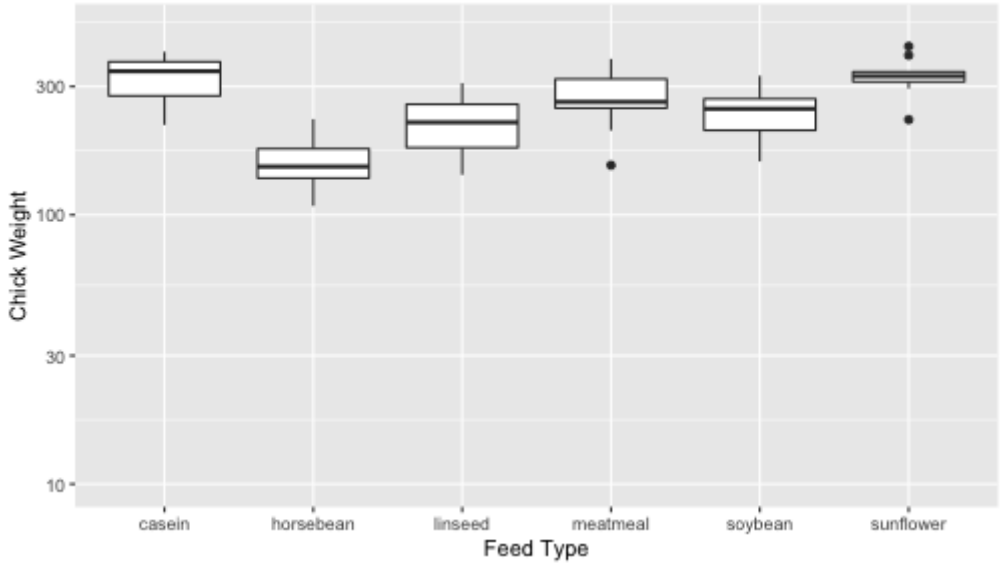
Changing the axis limits

```
ggplot(chickwts, aes(x=feed, y=weight)) + geom_boxplot() +
  scale_y_continuous("Chick Weight", limits=c(0,500)) +
  scale_x_discrete("Feed Type")
```



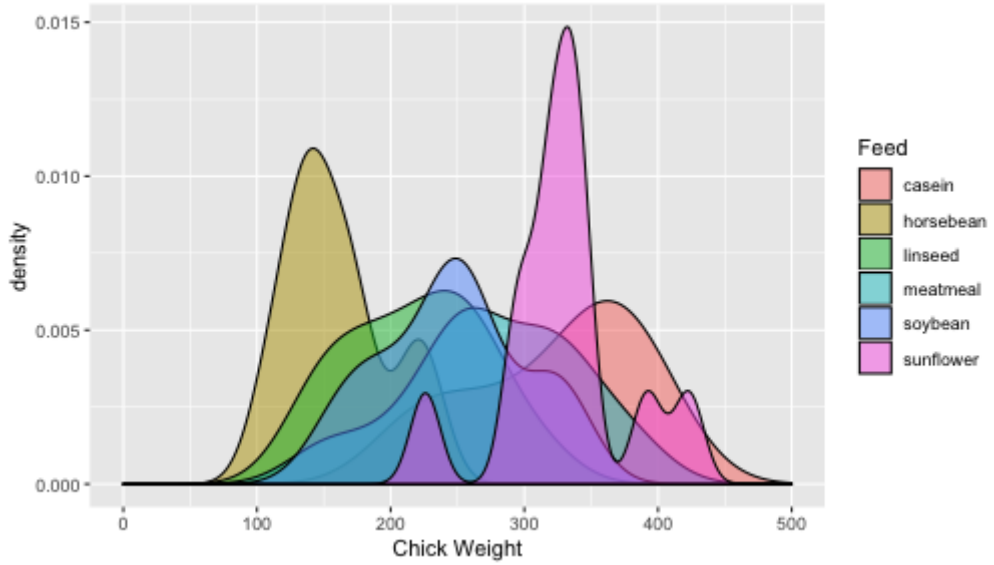
Log scale

```
ggplot(chickwts, aes(x=feed, y=weight)) + geom_boxplot() +
  scale_y_log10("Chick Weight", limits=c(10,500)) +
  scale_x_discrete("Feed Type")
```



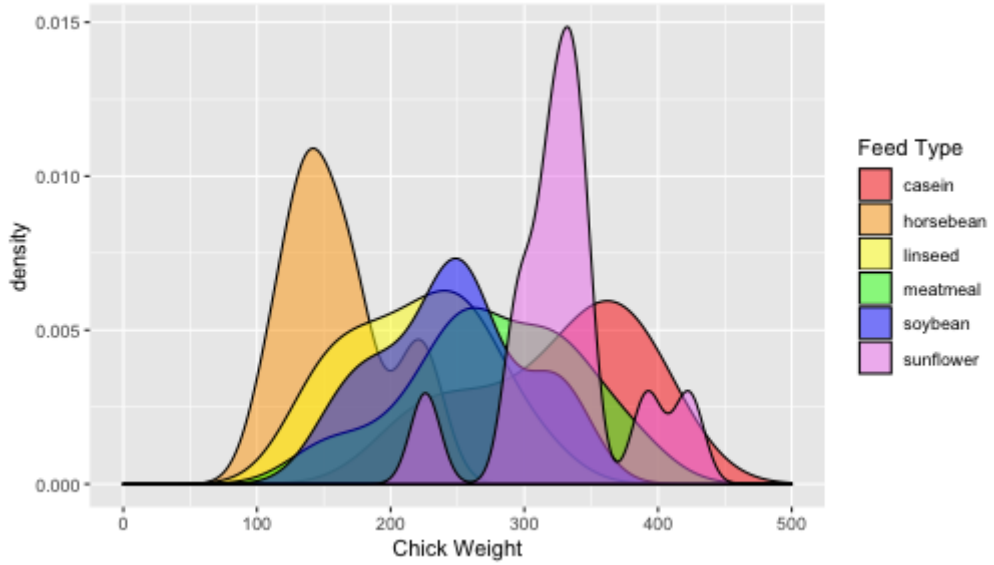
Coloring of Factors

```
ggplot(chickwts, aes(x=weight, fill=feed)) + geom_density(alpha=0.5) +
  scale_x_continuous("Chick Weight", limits=c(0,500))+
  labs(fill = 'Feed')
```



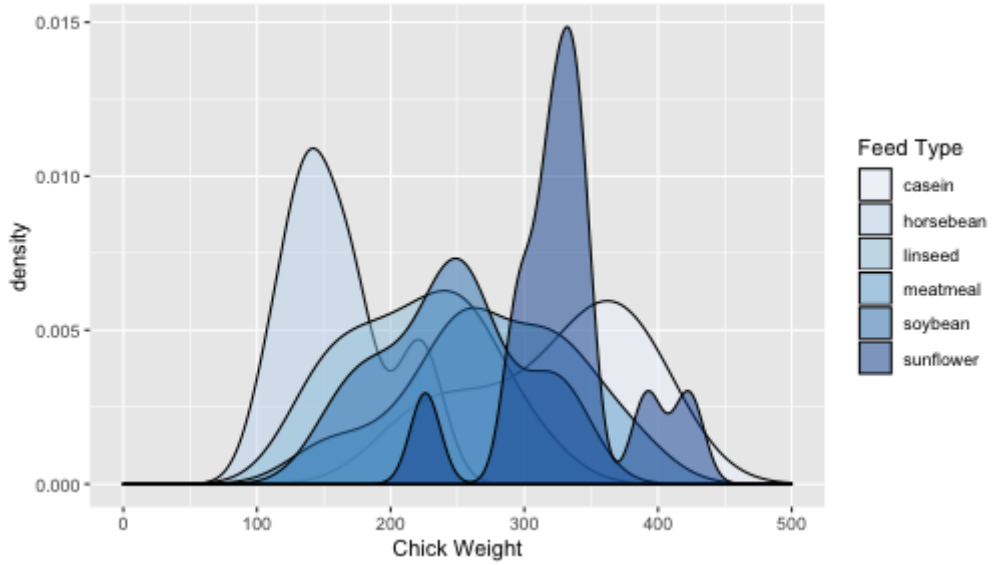
Manual Color Scale

```
ggplot(chickwts, aes(x=weight, fill=feed)) + geom_density(alpha=0.5) +
  scale_x_continuous("Chick Weight", limits=c(0,500)) +
  scale_fill_manual("Feed Type",values = c("red","orange","yellow","green","blue","violet"))
```



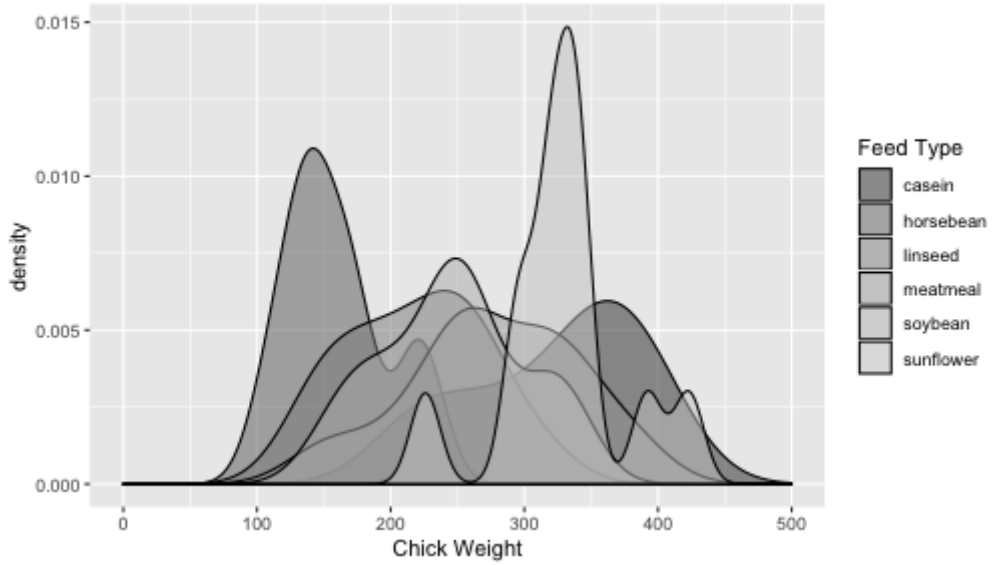
Fill Color Brewer

```
ggplot(chickwts, aes(x=weight, fill=feed)) + geom_density(alpha=0.5) +
  scale_x_continuous("Chick Weight", limits=c(0,500)) +
  scale_fill_brewer("Feed Type")
```



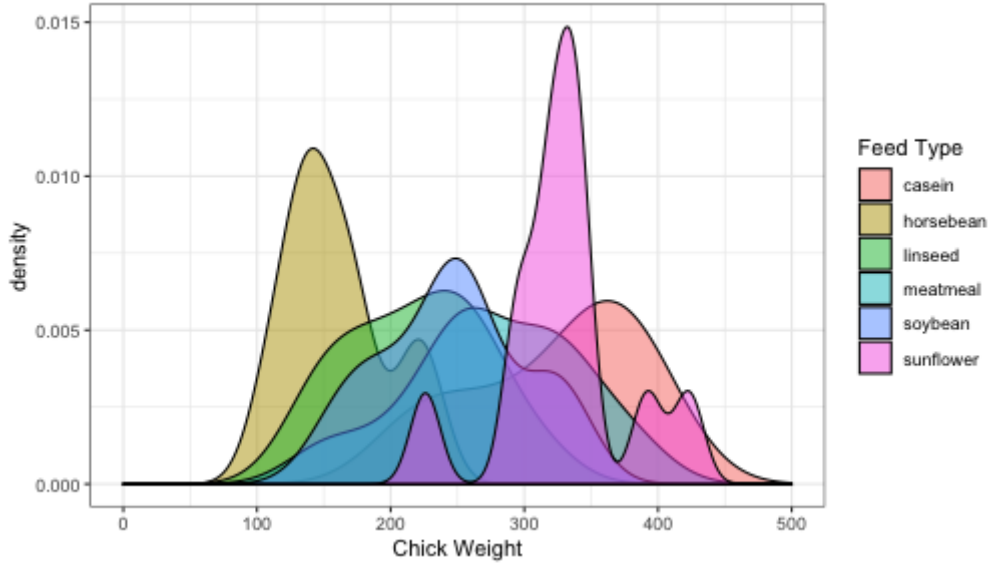
Fill Color Brewer

```
ggplot(chickwts, aes(x=weight, fill=feed)) + geom_density(alpha=0.5) +
  scale_x_continuous("Chick Weight", limits=c(0,500)) +
  scale_fill_grey("Feed Type")
```



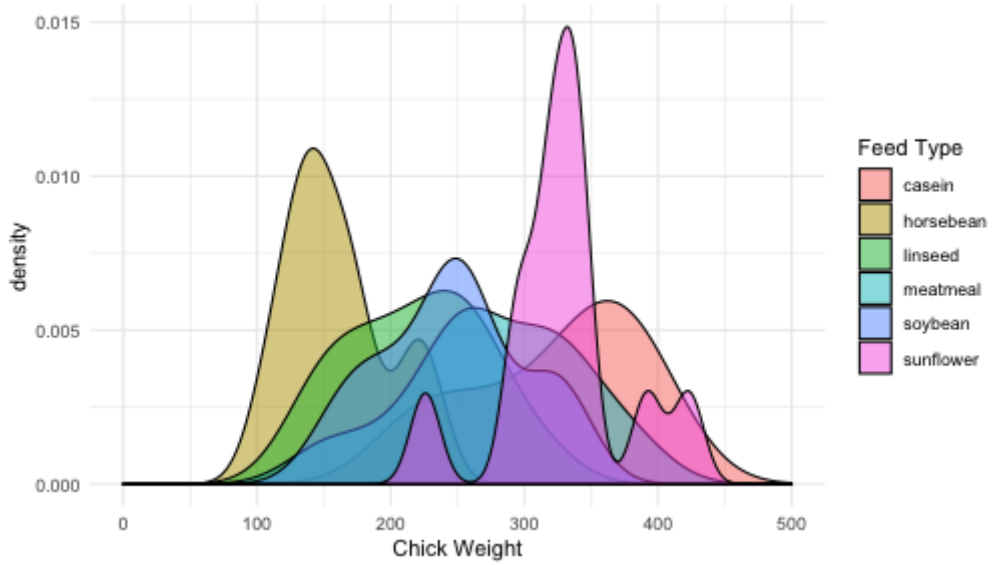
Changing Plot "Theme"

```
ggplot(chickwts, aes(x=weight, fill=feed)) + geom_density(alpha=0.5) +
  scale_x_continuous("Chick Weight", limits=c(0,500)) +
  scale_fill_discrete("Feed Type") + theme_bw()
```



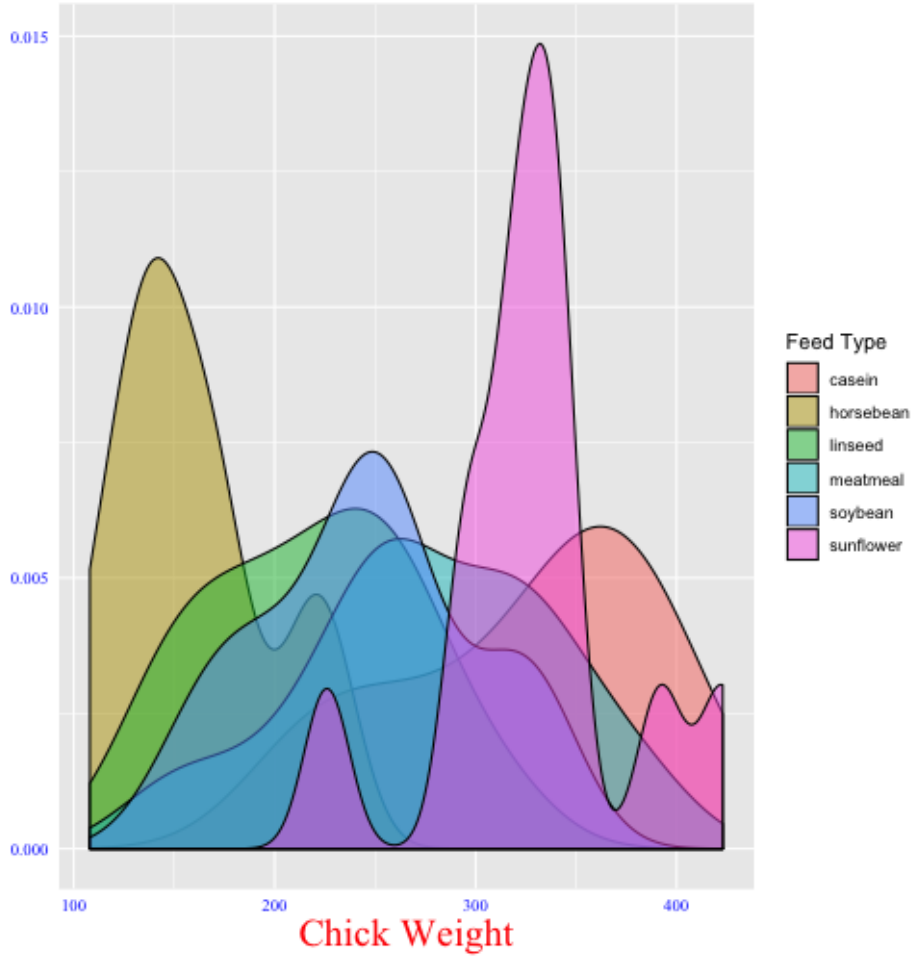
Minimal Theme

```
ggplot(chickwts, aes(x=weight, fill=feed)) + geom_density(alpha=0.5) +
  scale_x_continuous("Chick Weight", limits=c(0,500)) +
  scale_fill_discrete("Feed Type") + theme_minimal()
```



Personal theme

```
mytheme <- theme(axis.ticks = element_blank(),
                 axis.text = element_text(color = 'bl
                 axis.title = element_text(color = 'r
                                           size = 20)
ggplot(chickwts, aes(x = weight, fill = feed))+geom_d
  labs(x = 'Chick Weight', y = '', fill  = 'Feed Type
  mytheme
```

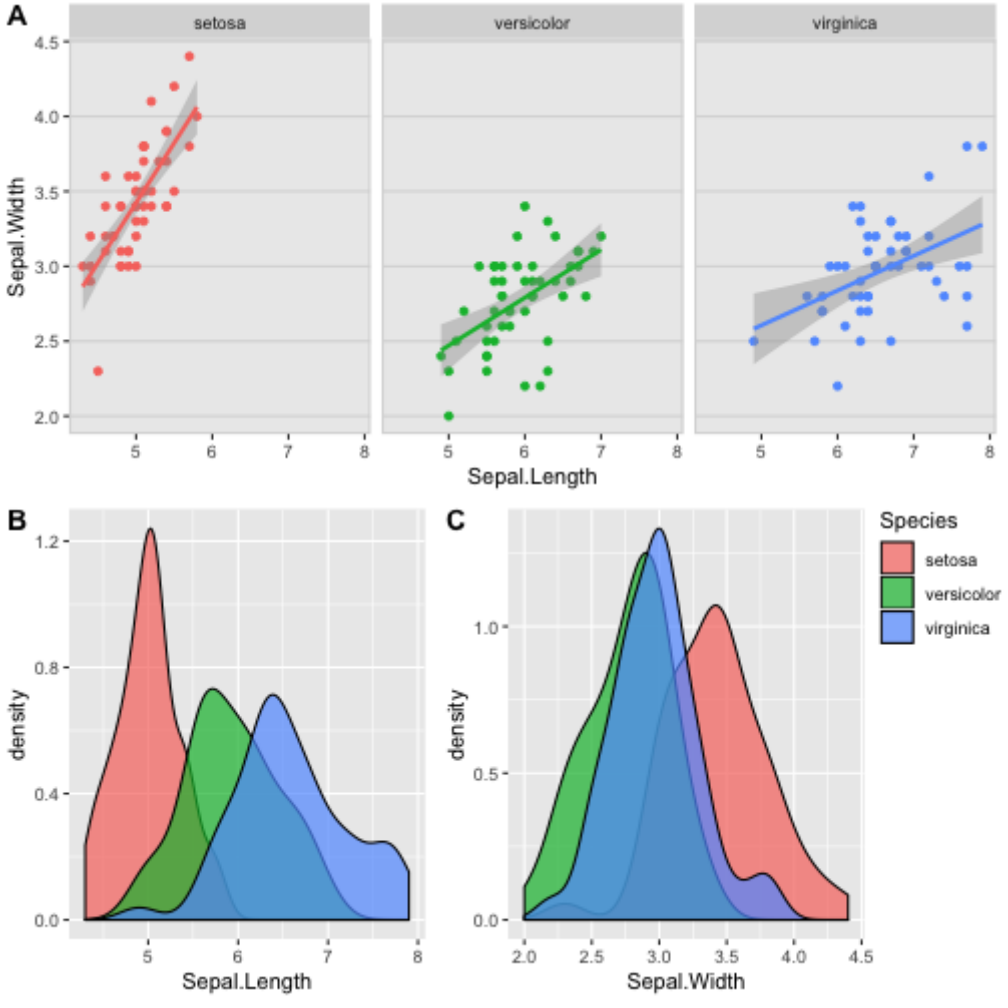


Multiple Graphs

The packages `cowplot` and `ggpubr` make putting different graphs on the same panel pretty straightforward.

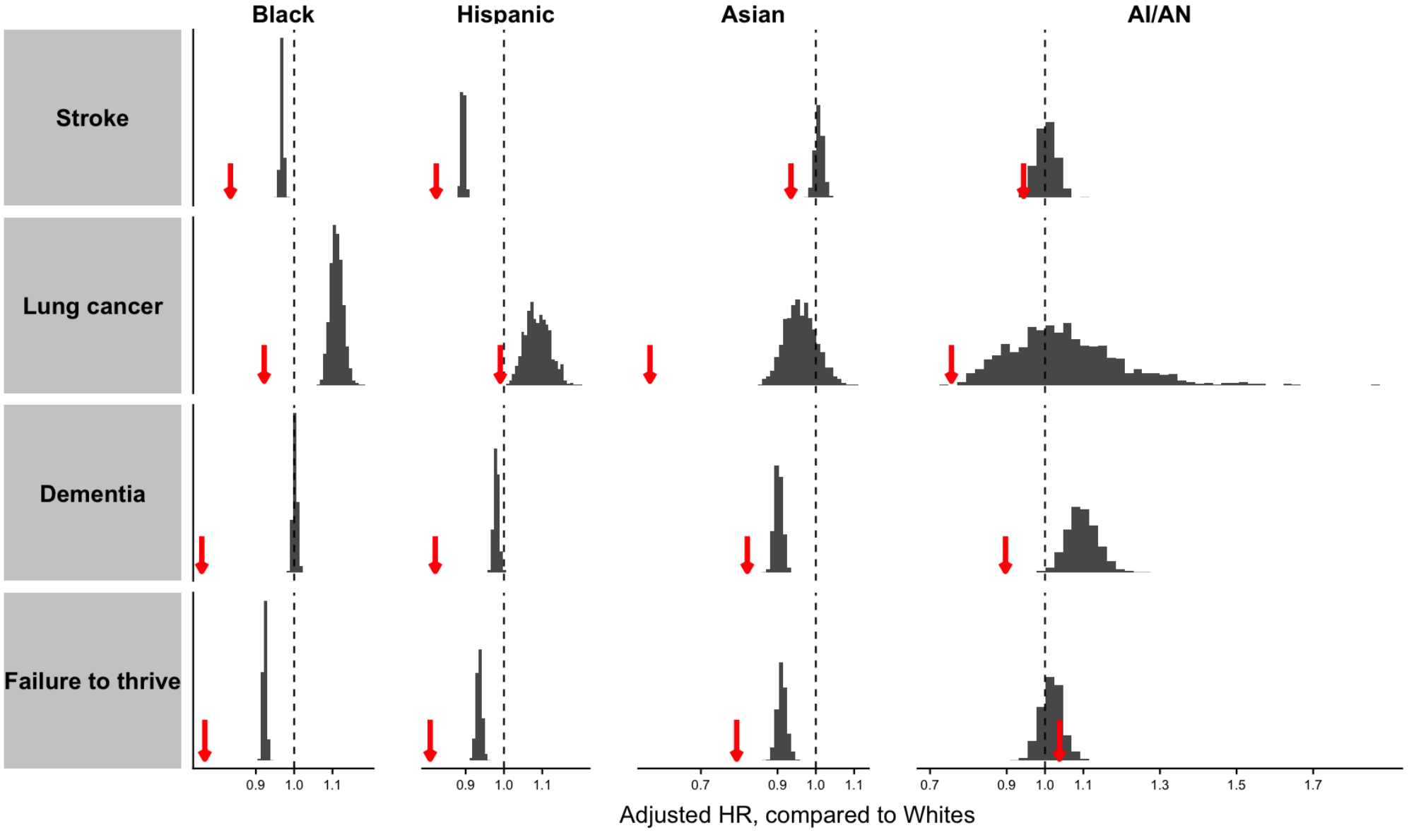
```
# install.packages('cowplot')
library(cowplot)
p1 <- ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point() + facet_grid(. ~ Species) + stat_smooth(method = "lm") +
  background_grid(major = 'y', minor = "none") +
  panel_border() + theme(legend.position = "none")

# plot B
p2 <- ggplot(iris, aes(Sepal.Length, fill = Species)) +
  geom_density(alpha = .7) + theme(legend.justification = "top")
p2a <- p2 + theme(legend.position = "none")

# plot C
p3 <- ggplot(iris, aes(Sepal.Width, fill = Species)) +
  geom_density(alpha = .7) + theme(legend.position = "none")

# legend
legend <- get_legend(p2)

# align all plots vertically
plots <- align_plots(p1, p2a, p3, align = 'v', axis = 'l')

# put together bottom row and then everything
bottom_row <- plot_grid(plots[[2]], plots[[3]], legend, labels = c("B", "C"), rel_widths = c(1, 1, .3), nrow = 1)
plot_grid(plots[[1]], bottom_row, labels = c("A"), ncol = 1)
```

Multiple Graphs



Fine-tuning the theme

```
plt <- ggplot(bl, aes(x = estimate)) + geom_histogram(bins = 50)+#geom_density() +
  facet_grid(Event ~ Race, scales = 'free', switch = 'y', space = 'free_x') +
  geom_vline(xintercept = 1, linetype = 2) +
 geom_segment(data = bl2, aes(x = estimate, xend=estimate, yend = 5, y = hts),
              color='red', size = 1.5, arrow = arrow(length = unit(.2, 'cm')))+
  scale_x_continuous(breaks = c(1, seq(0.7, 1.8, by = 0.2)))+ # Unified the x-axis ticks
  labs(x = 'Adjusted HR, compared to Whites', y = '') +
  theme(strip.text = element_text(size = 14, face = 'bold'),
        strip.text.y = element_text(angle = 180), # Rotate the y-axis labels
        strip.background.x = element_rect(fill = 'white'),
        strip.placement = 'outside', # Move labels outside the borders
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text.x = element_text(size = 8),
        panel.spacing.x = unit(2, 'lines'))
```


Black
Hispanic
Asian
AI/AN
Stroke
Lung cancer
Dementia
Failure to thrive
0.9
1.0
1.1
0.7
1.3
1.5
1.7
Adjusted HR, compared to Whites