The graphical system

Plotting figures and graphs with ggplot

- ggplot is the plotting library for tidyverse
  - Powerful
  - Flexible

- Follows the same conventions as the rest of tidyverse
  - Data stored in tibbles
  - Data is arranged in 'tidy' format
  - Tibble is the first argument to each function

Introduction to ggplot2

We will be creating plots using the ggplot2 package.

```
> library(dplyr)
> library(ggplot2)
```

There are also other packages for creating graphics such as **grid** and **lattice**. We chose to use **ggplot2** in this book because it breaks plots into components in a way that permits beginners to create relatively complex and aesthetically pleasing plots using syntax that is intuitive and comparatively easy to remember.

Advantages of ggplot2:

- ✓ Grammar of graphics
- ✓ Default behaviour
- ✓ ggplot2 sheet cheat

Disadvantages of ggplot2:

- ✗ One limitation is that **ggplot2** is designed to work exclusively with data tables in tidy format (where rows are observations and columns are variables).

Code structure of a ggplot graph

- Start with a call to `ggplot()`
  - Pass the tibble of data (normally via a pipe)
  - Say which columns you want to use via a call to `aes()`

- Say which graphical representation (geometry) you want to use
  - Points, lines, barplots etc

- Customise labels, colours annotations etc.

Geometries and Aesthetics

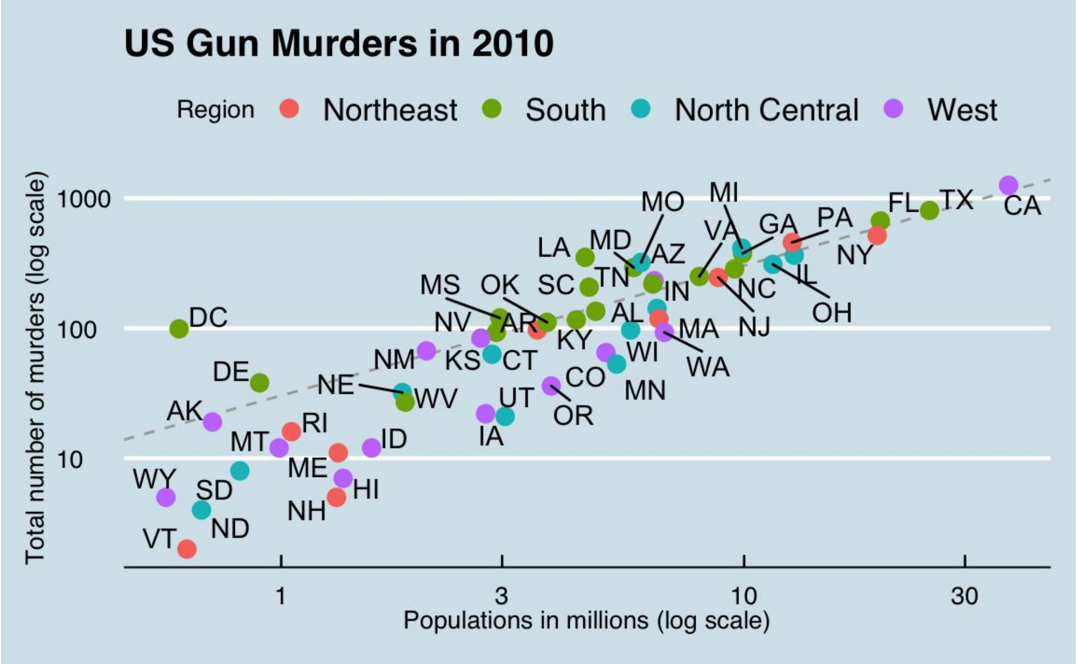
- Geometries are types of plot

`geom_point()`	Point geometry, (x/y plots, stripcharts etc)
`geom_line()`	Line graphs
`geom_boxplot()`	Box plots
`geom_col()`	Barplots
`geom_histogram()`	Histogram plots

- Aesthetics are graphical parameters which can be adjusted in a given geometry

The components of a graph

We will construct a graph that summarizes the US murders dataset that looks like this:



The main three components to note are:

- **Data**: The US murders data table is being summarized.
- **Geometry**: The plot above is a scatterplot. This is referred to as the **geometry** component.
- **Aesthetic mapping**: The plot uses several visual cues to represent the information provided by the dataset.

ggplot objects

`> ggplot(data = murders)` *or* `> murders %>% ggplot()` *or* `> p <- ggplot(data = murders)`

no geometry has been defined!

Geometries

In ggplot2 we create graphs by adding *layers*. Layers can define geometries, compute summary statistics, define what scales to use, or even change styles. To add layers, we use the the symbol +. In general, a line of code will look like this:

DATA %>% ggplot() + LAYER 1 + LAYER 2 + … + LAYER N

Geometry function names follow the pattern: geom_X where X is the name of the geometry. Some examples include geom_point, geom_bar and geom_histogram.

```
> Aesthetics
>
> geom_point understands the following aesthetics (required aesthetics are in bold):
> x
> y
> alpha
> colour
```

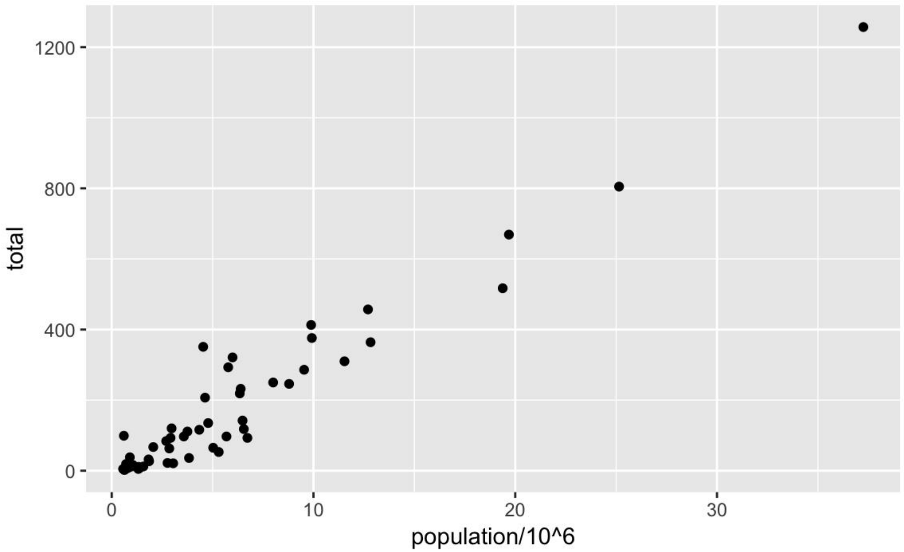
Aesthetic mappings

Aesthetic mappings describe how properties of the data connect with features of the graph, such as distance along an axis, size or color.

murders %>% **ggplot**() + **geom_point**(**aes**(x = population/10^6, y = total))

Or

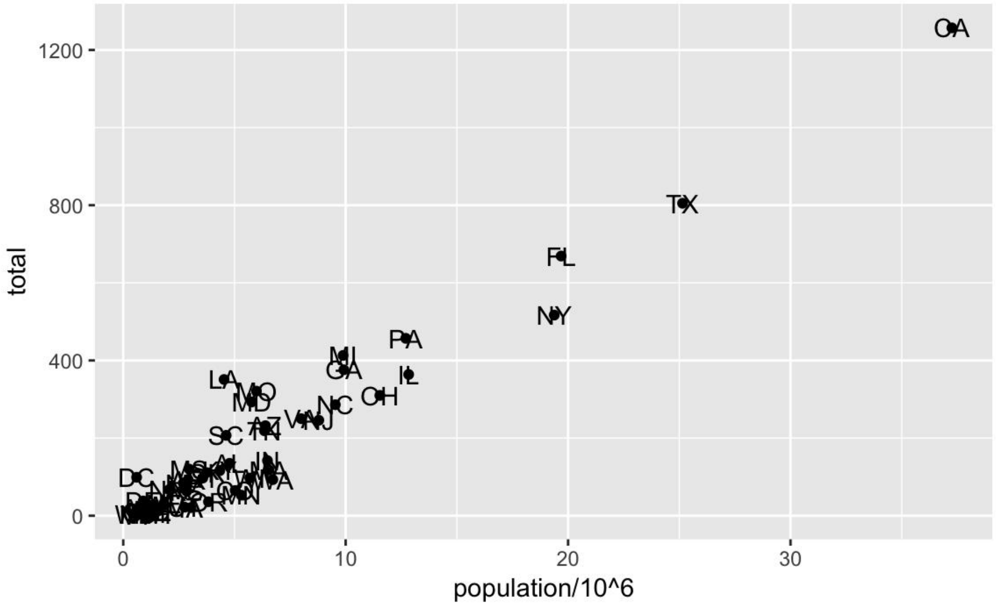
p + **geom_point**(**aes**(population/10^6, total))



Layers

A second layer in the plot we wish to make involves adding a label to each point to identify the state. The **geom_label** and **geom_text** functions permit us to add text to the plot with and without a rectangle behind the text respectively.

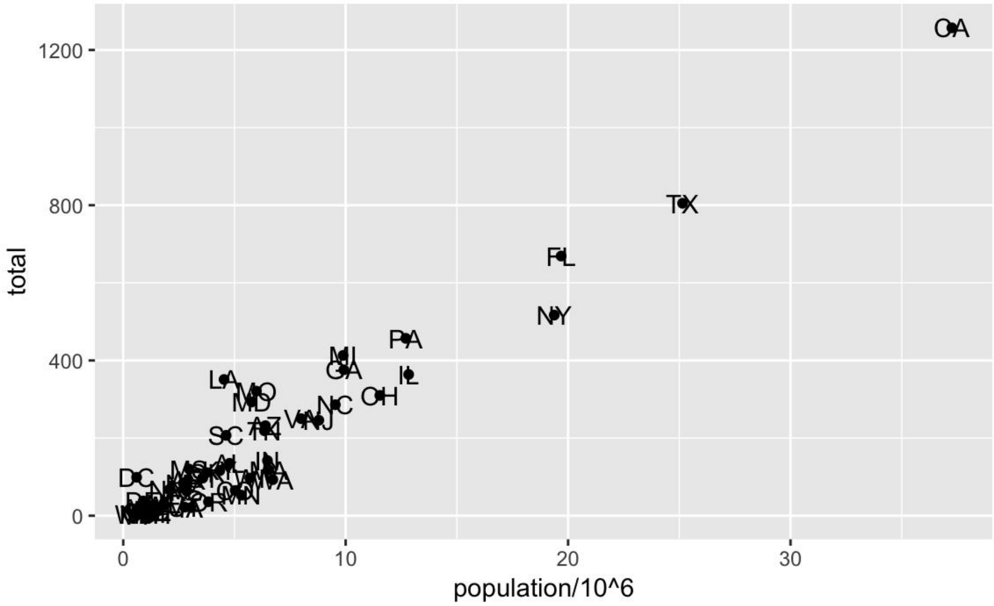
```
p + geom_point(aes(population/10^6, total)) + geom_text(aes(population/10^6, total, label = abb))
```

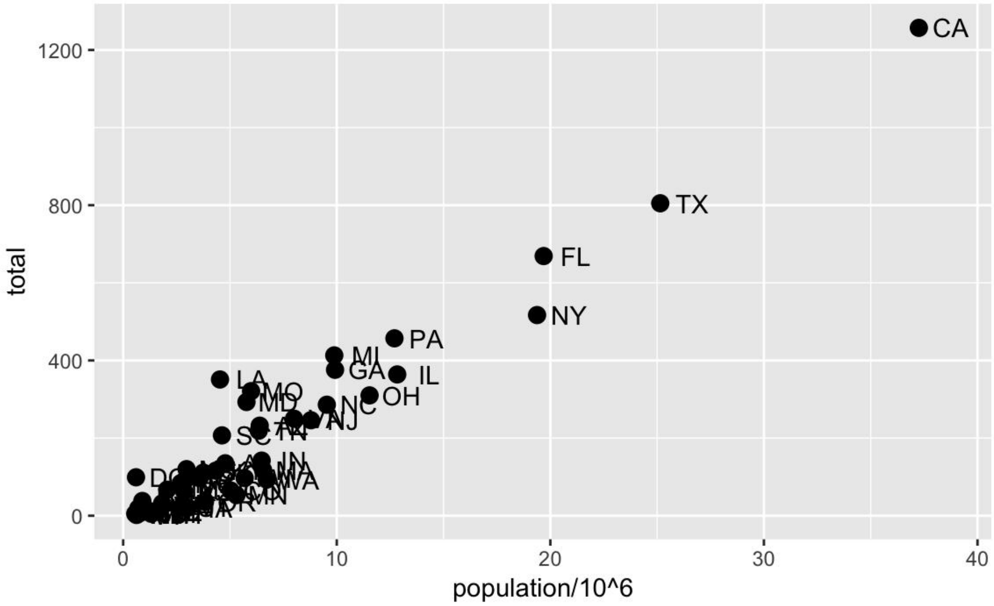


Layers

A second layer in the plot we wish to make involves adding a label to each point to identify the state. The **geom_label** and **geom_text** functions permit us to add text to the plot with and without a rectangle behind the text respectively.

```
p + geom_point(aes(population/10^6, total)) + geom_text(aes(population/10^6, total, label = abb))

p + geom_point(aes(population/10^6, total), size = 3) + geom_text(aes(population/10^6, total, label = abb), nudge_x = 1.5)
```





Global versus local aesthetic mappings

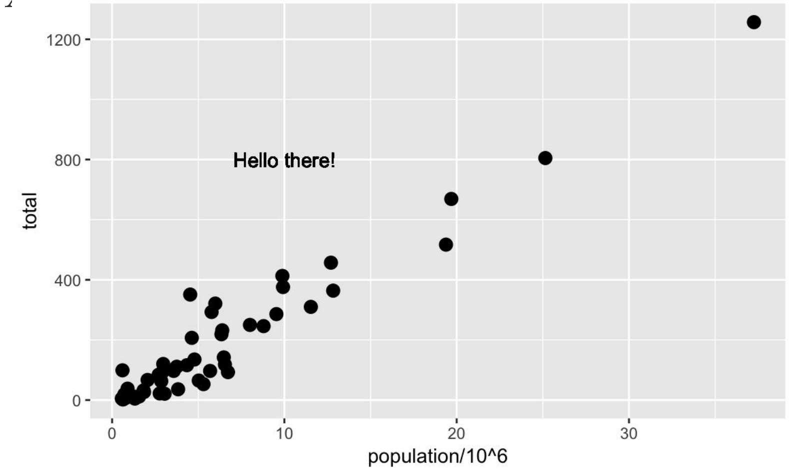
```
p + geom_point(aes(population/10^6, total), size = 3) + geom_text(aes(population/10^6, total, label = abb),
nudge_x = 1.5)
```

Or

```
p <- murders %>% ggplot(aes(population/10^6, total, label = abb))

p + geom_point(size = 3) + geom_text(nudge_x = 1.5)
```

If necessary, we can override the global mapping by defining a new mapping within each layer. These *local* definitions override the *global*. Here is an example:

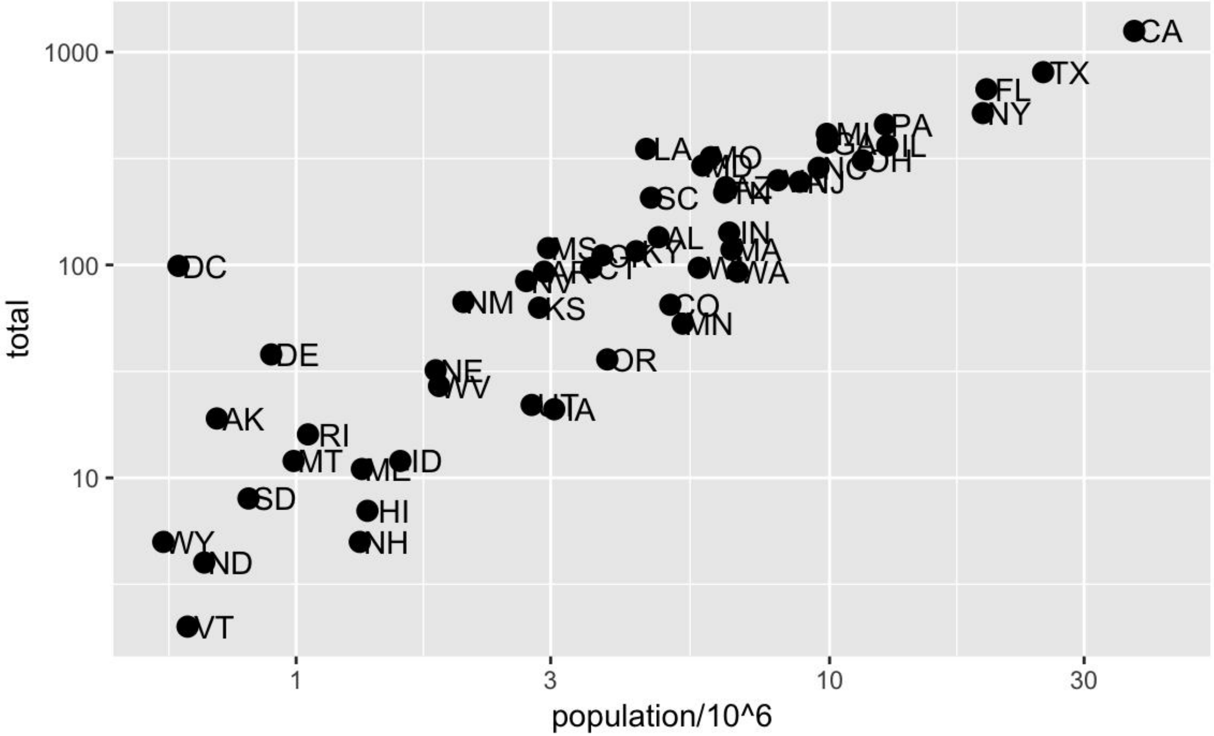
```
p + geom_point(size = 3) +
geom_text(aes(x = 10, y = 800, label =
"Hello there!"))
```



Scales

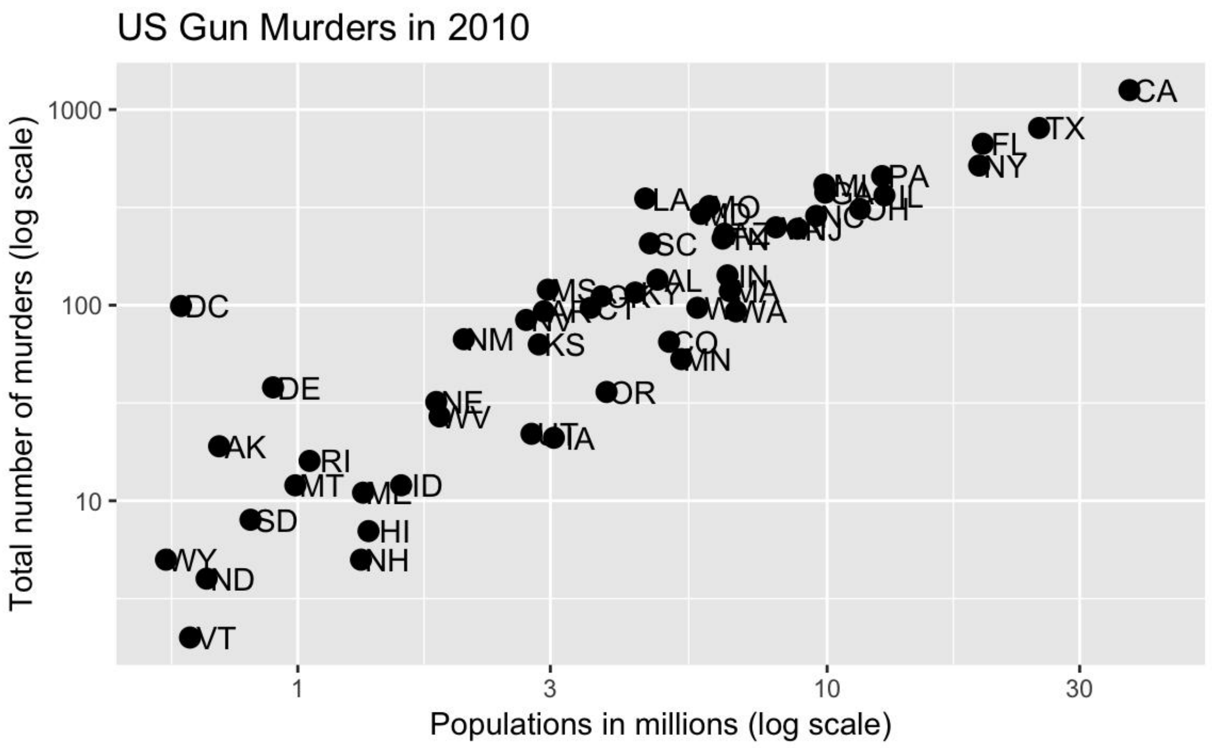
```
p + geom_point(size = 3) +
geom_text(nudge_x = 0.05) +
scale_x_continuous(trans = "log10") +
scale_y_continuous(trans = "log10")
```

```
p + geom_point(size = 3) +
geom_text(nudge_x = 0.05) +
scale_x_log10() + scale_y_log10()
```



Labels and titles

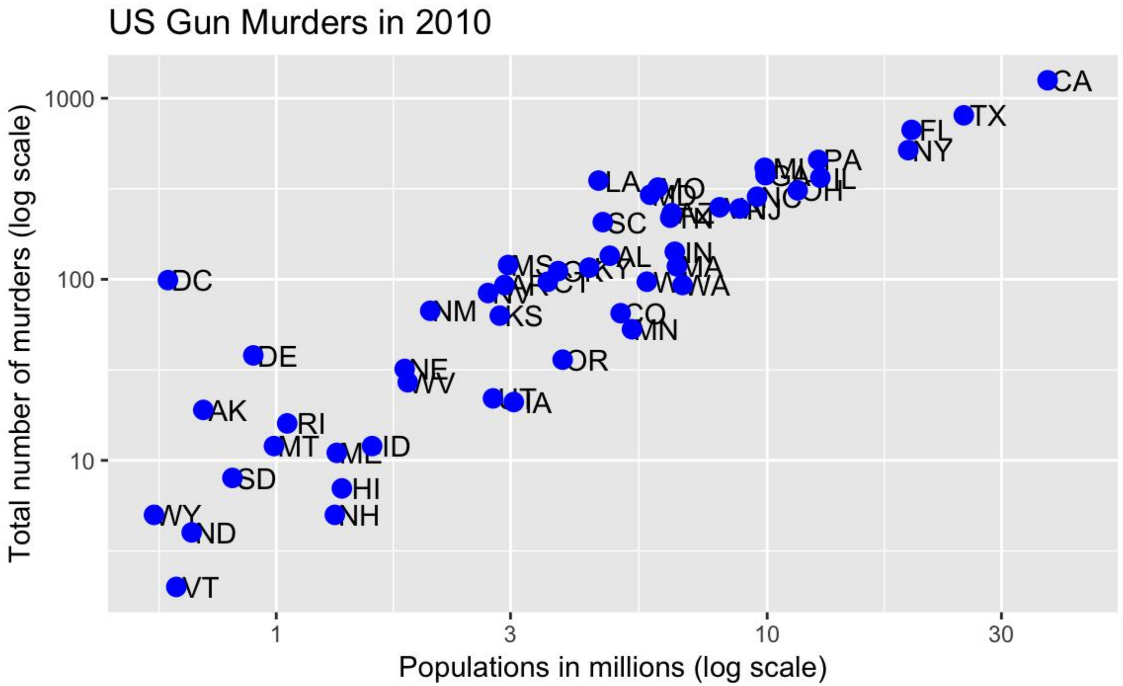
```
p + geom_point(size = 3) +
geom_text(nudge_x = 0.05) +
scale_x_log10() + scale_y_log10() +
xlab("Populations in millions (log scale)") +
ylab("Total number of murders (log scale)")
+ ggtitle("US Gun Murders in 2010")
```



We are almost there! All we have left to do is add color, a legend and optional changes to the style.

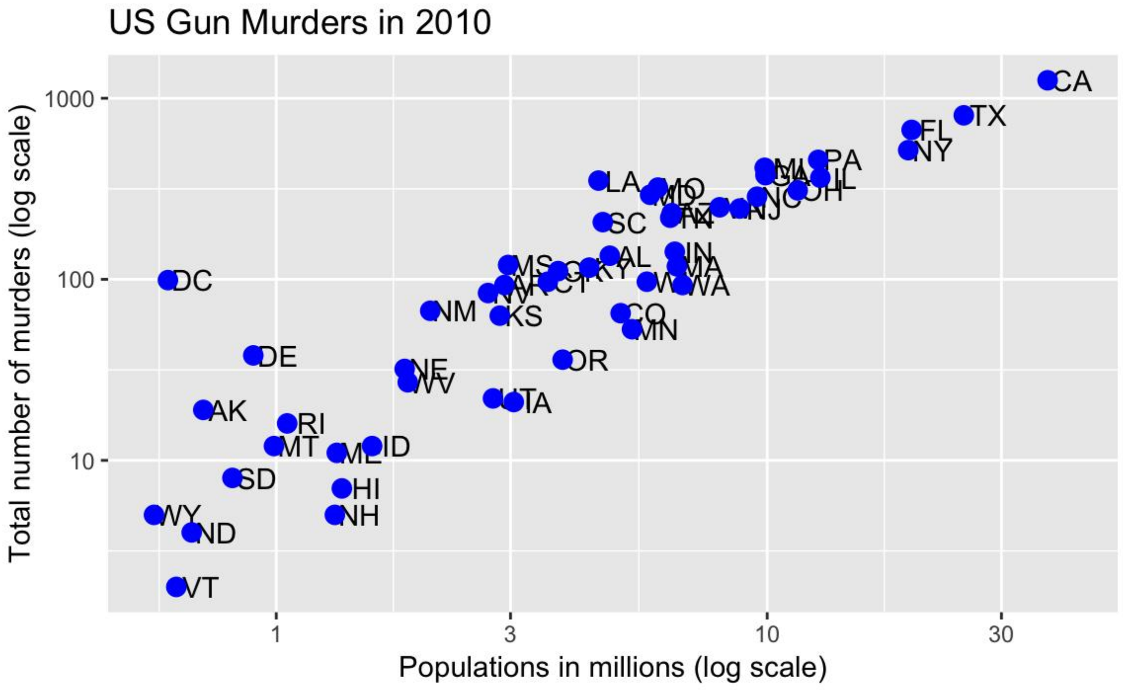
Categories as colors

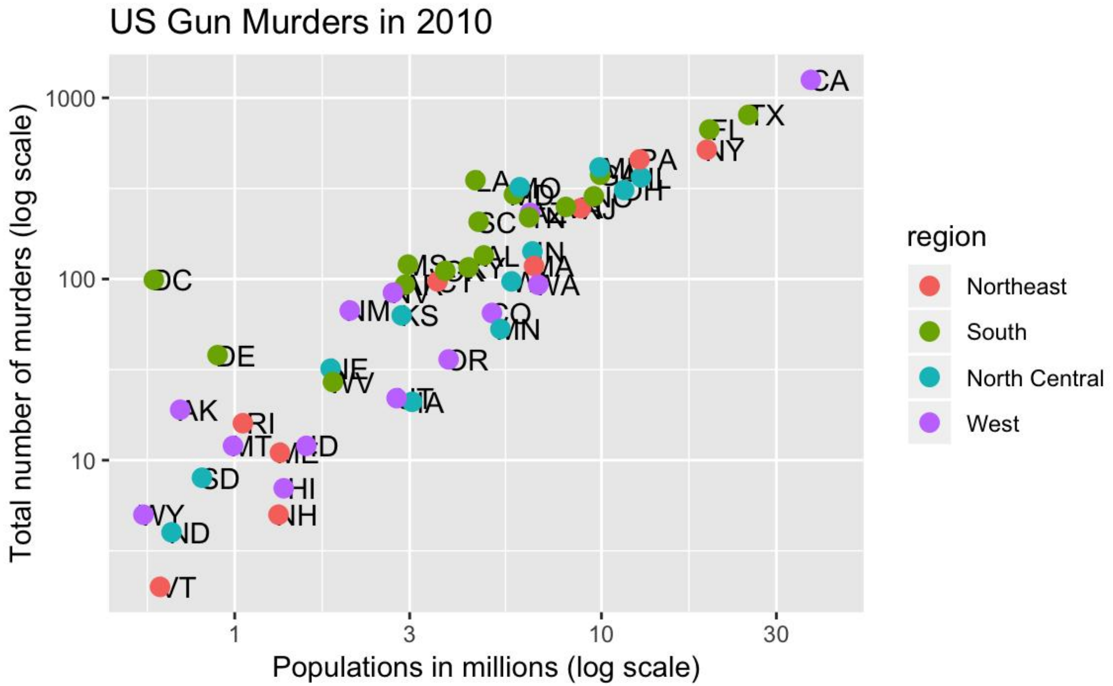
```
p <- murders %>% ggplot(aes(population/10^6, total, label = abb)) + geom_text(nudge_x = 0.05) + scale_x_log10() +
scale_y_log10() + xlab("Populations in millions (log scale)") + ylab("Total number of murders (log scale)") +
ggtitle("US Gun Murders in 2010")

p + geom_point(color ="blue ", size = 3 )
```



Categories as colors

```
p <- murders %>% ggplot(aes(population/10^6, total, label = abb)) + geom_text(nudge_x = 0.05) + scale_x_log10() +
scale_y_log10() + xlab("Populations in millions (log scale)") + ylab("Total number of murders (log scale)") +
ggtitle("US Gun Murders in 2010")

p + geom_point(aes(col=region), size = 3 )
```





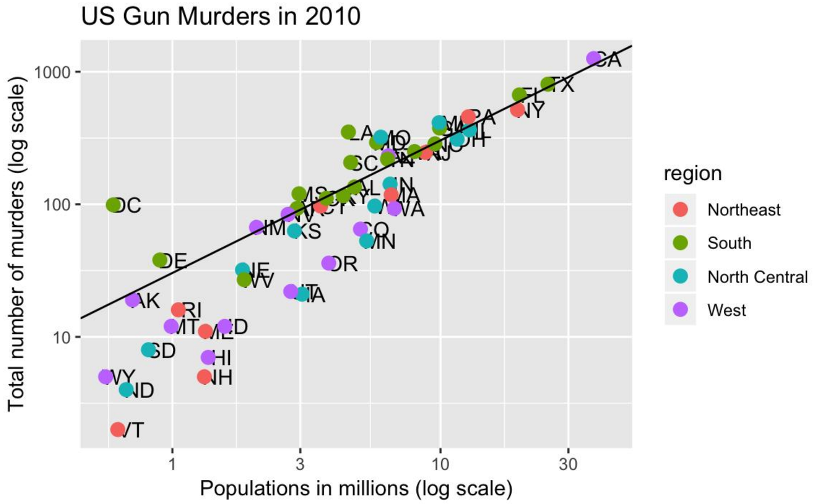
Annotation, shapes, and adjustments

Here we want to add a line that represents the average murder rate for the entire country.

```
r <- murders %>% summarize(rate = sum(total) / sum(population) * 10^6) %>% pull(rate)
```

To add a line we use the geom_abline function. ggplot2 uses ab in the name to remind us we are supplying the intercept (a) and slope (b). The default line has slope 1 and intercept 0 so we only have to define the intercept:

```
p + geom_point(aes(col=region), size = 3) + geom_abline(intercept = log10(r))
```



Add-on packages

The power of **ggplot2** is augmented further due to the availability of add-on packages. The remaining changes needed to put the finishing touches on our plot require the **ggthemes** and **ggrepel** packages.

```
library(ggthemes)
p + theme_economist()
```

The add-on package **ggrepel** includes a geometry that adds labels while ensuring that they don’t fall on top of each other. We simply change geom_text with geom_text_repel.

Putting it all together

Now that we are done testing, we can write one piece of code that produces our desired plot from scratch.

Putting it all together

```
library(ggthemes)
library(ggrepel)

r <- murders %>% summarize(rate = sum(total) / sum(population) * 10^6) %>%
pull(rate)

murders %>% ggplot(aes(population/10^6, total, label = abb)) +
geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
geom_point(aes(col=region), size = 3) +
geom_text_repel() +
scale_x_log10() +
scale_y_log10() +
xlab("Populations in millions (log scale)") +
ylab("Total number of murders (log scale)") +
ggtitle("US Gun Murders in 2010") +
scale_color_discrete(name = "Region") +
theme_economist()
```

Quick plots with qplot

If we have values in two vectors, say:

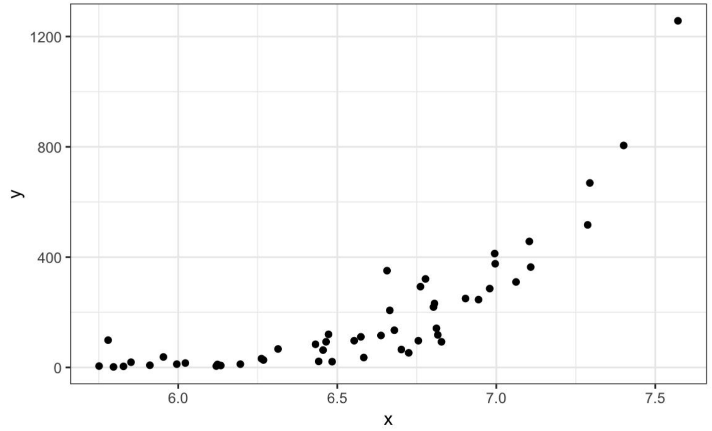
```
data(murders)
x <- log10(murders$population)
y <- murders$total
```

and we want to make a scatterplot with ggplot, we would have to type something like:

```
data.frame(x = x, y = y) %>% ggplot(aes(x, y)) + geom_point()
```

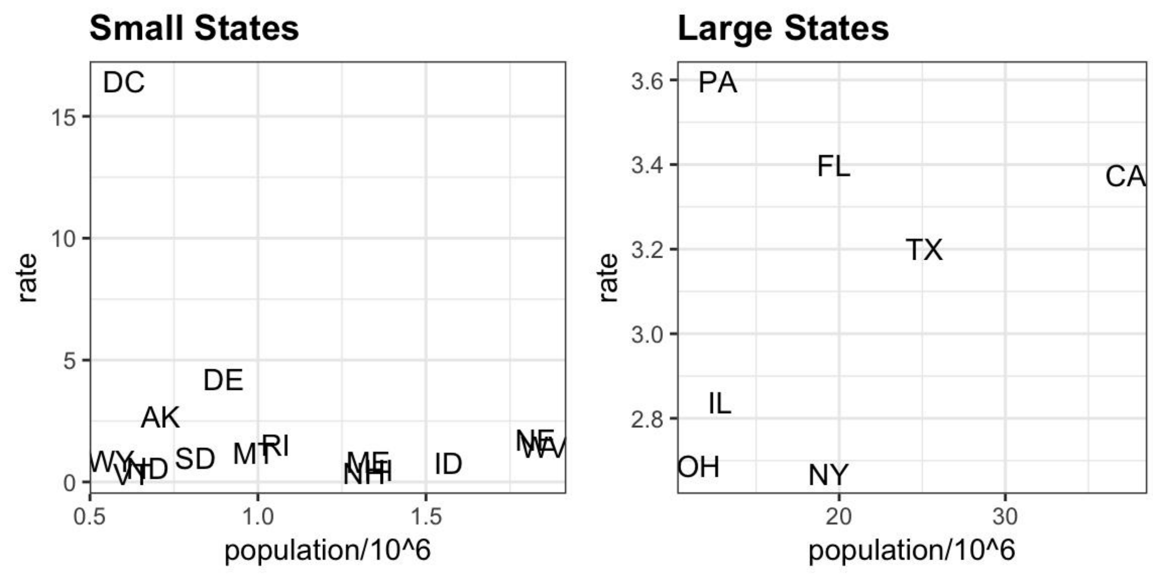
This seems like too much code for such a simple plot. The qplot function sacrifices the flexibility provided by the ggplot approach, but allows us to generate

```
qplot(x, y)
```



Grids of plots

```
library(gridExtra)
#> Attaching package: 'gridExtra’
#> The following object is masked from 'package:dplyr’:
 #> combine

p1 <- murders %>% mutate(rate = total/population*10^5) %>%
filter(population < 2*10^6) %>%
ggplot(aes(population/10^6, rate, label = abb))
+ geom_text()
+ ggtitle("Small States")

p2 <- murders %>% mutate(rate = total/population*10^5) %>%
filter(population > 10*10^6) %>%
ggplot(aes(population/10^6, rate, label = abb)) +
geom_text() +
ggtitle("Large States")

grid.arrange(p1, p2, ncol = 2)
```

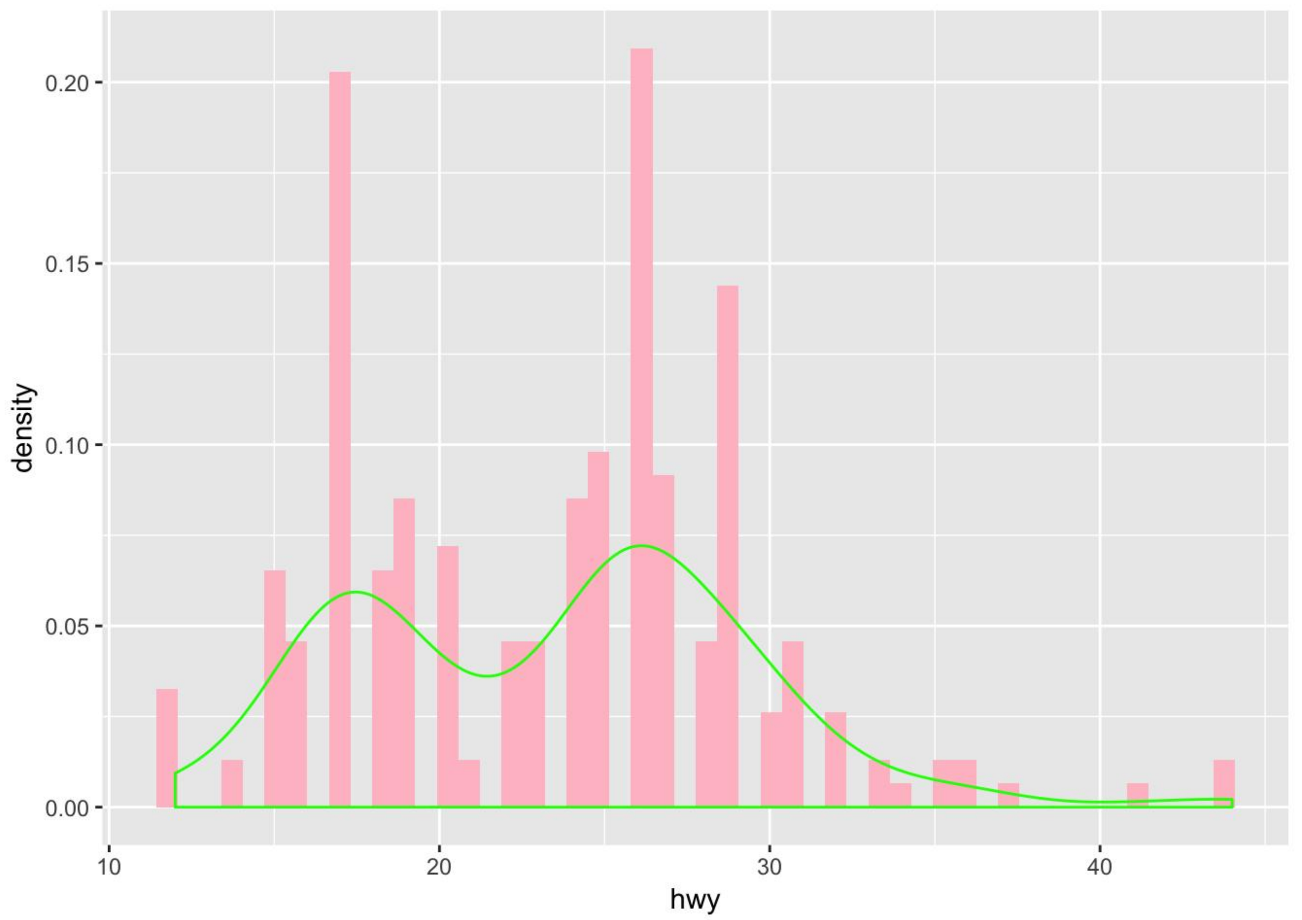


Other Geometries

- Barplots
  - geom_bar
  - geom_col
- Stripcharts
  - geom_jitter
- Distribution Summaries
  - geom_histogram
  - geom_density
  - geom_violin
  - geom_boxplot

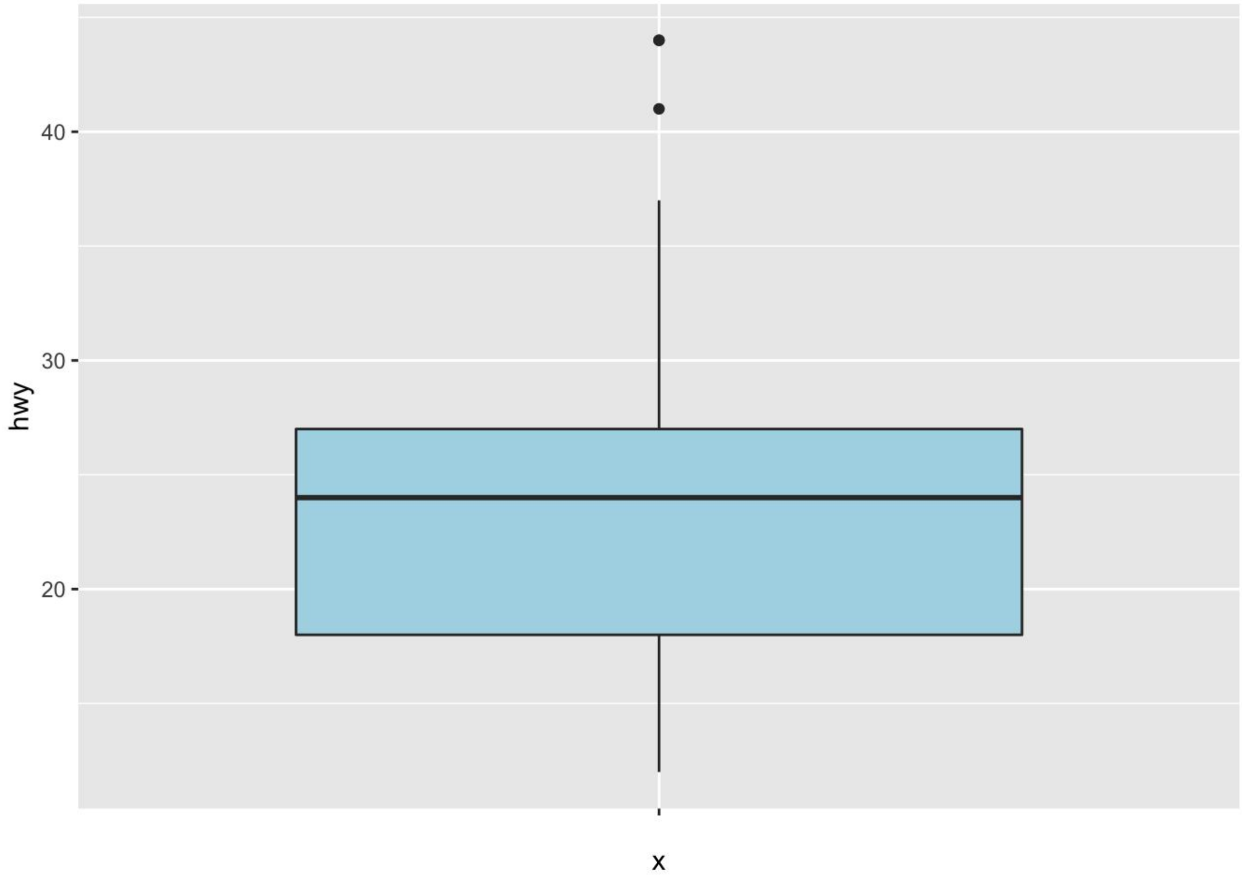
Histograms

```
ggplot(data = mpg) + geom_histogram(aes(x = hwy, y = ..density..), bins=50, fill = 'pink') + geom_density(aes(x = hwy),col = 'green')
```



Boxplots

```
ggplot(data = mpg) + geom_boxplot(aes(x = '',y = hwy), fill = 'lightblue')
```



Let’s practice on boxplots

```
library(dplyr)
library(ggplot2)
# Step 1: Import the data
data_air <- airquality % > %
#Step 2: Drop unnecessary variables
select(-c(Solar.R, Temp)) % > %
#Step 3: Convert Month in factor level
mutate(Month = factor(Month, order = TRUE, labels = c("May", "June", "July", "August", "September")),
#Step 4: Create a new categorical variable dividing the month with three level: begin, middle and end.
day_cat = factor(ifelse(Day < 10, "Begin", ifelse(Day < 20, "Middle", "End"))))

glimpse(data_air)
```

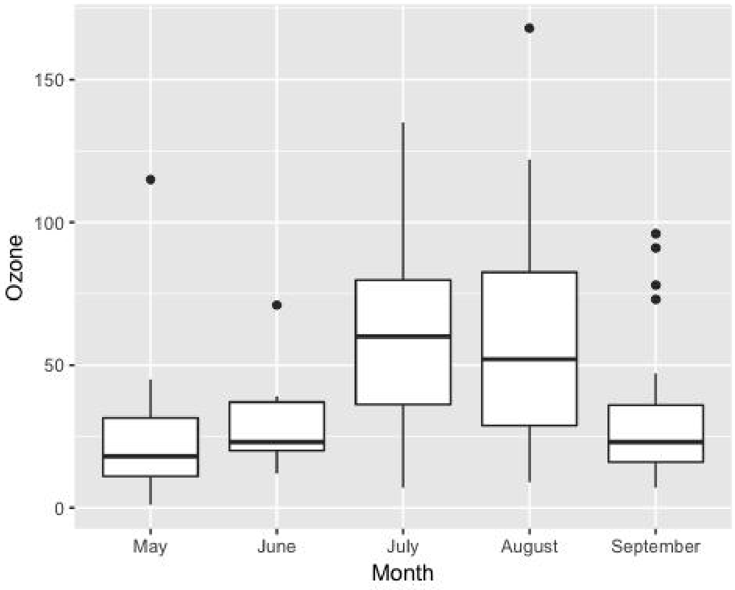
```
## Observations: 153
## Variables: 5
## $ Ozone <int> 41, 36, 12, 18, NA, 28, 23, 19, 8, NA, 7, 16, 11, 14, ...
## $ Wind <dbl> 7.4, 8.0, 12.6, 11.5, 14.3, 14.9, 8.6, 13.8, 20.1, 8.6...
## $ Month <ord> May, May, May, May, May, May, May, May, May, May, May,...
## $ Day <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,...
## $ day_cat <fctr> Begin, Begin, Begin, Begin, Begin, Begin, Begin, Begi...
```

```
# Step 5: Remove missing observations
data_air_nona <-data_air %>% na.omit()
```

Basic box plot

Let's plot the basic R boxplot() with the distribution of ozone by month.

```
# Store the graph
box_plot <- ggplot(data_air_nona, aes(x = Month, y = Ozone))
# Add the geometric object box
plot box_plot + geom_boxplot()
```

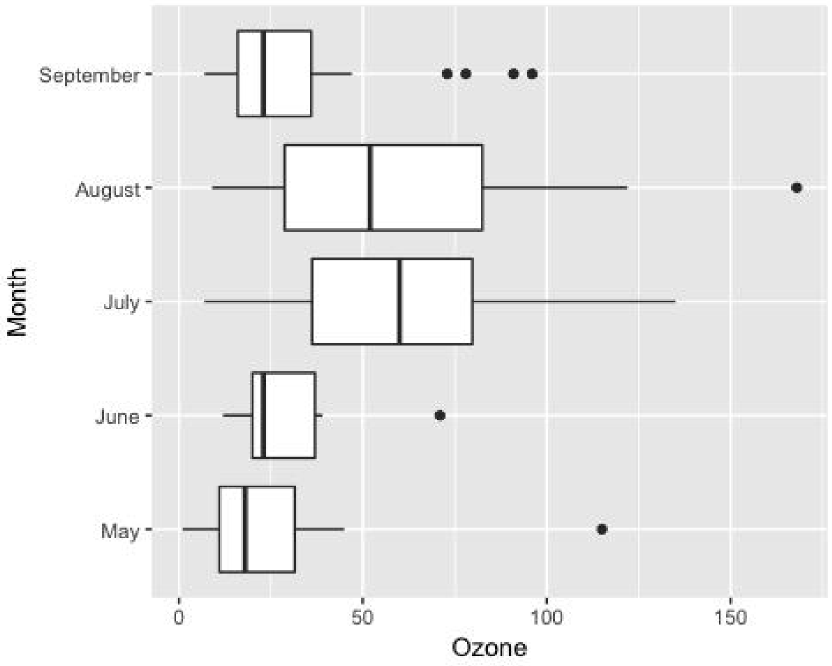


Code Explanation

- Store the graph for further use
  - box_plot: You store the graph into the variable box_plot It is helpful for further use or avoid too complex line of codes
- Add the geometric object of R boxplot()
  - You pass the dataset data_air_nona to ggplot boxplot.
  - Inside the aes() argument, you add the x-axis and y-axis.
  - The + sign means you want R to keep reading the code. It makes the code more readable by breaking it.
  - Use geom_boxplot() to create a box plot

Change side of the graph

```
box_plot + geom_boxplot()+ coord_flip()
```

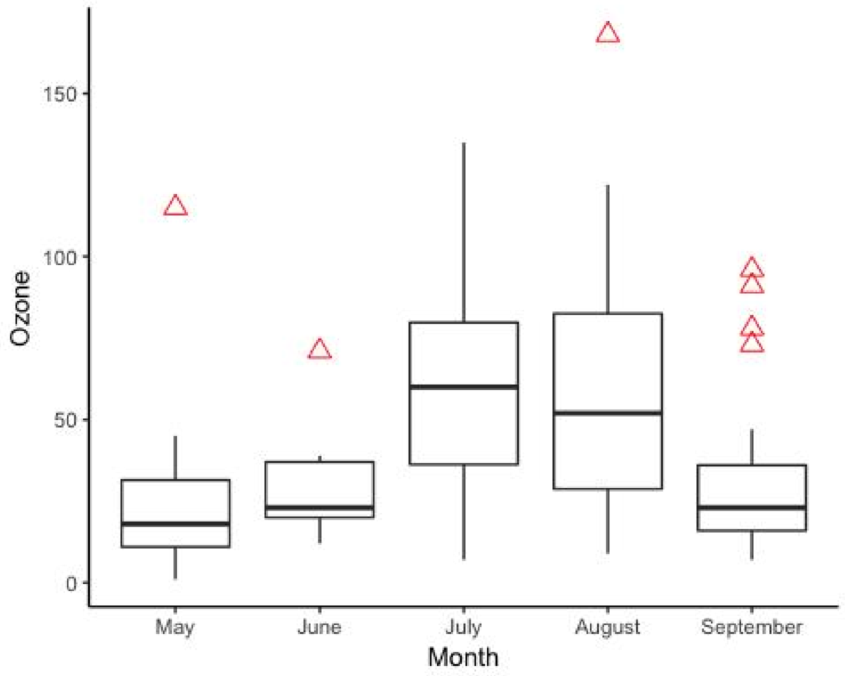


Code Explanation

•box_plot: You use the graph you stored. It avoids rewriting all the codes each time you add new information to the graph.
•geom_boxplot(): Create boxplots() in R
•coord_flip(): Flip the side of the graph

Change colour of outlier

```
box_plot + geom_boxplot(outlier.colour = "red", outlier.shape = 2, outlier.size = 3) + theme_classic()
```

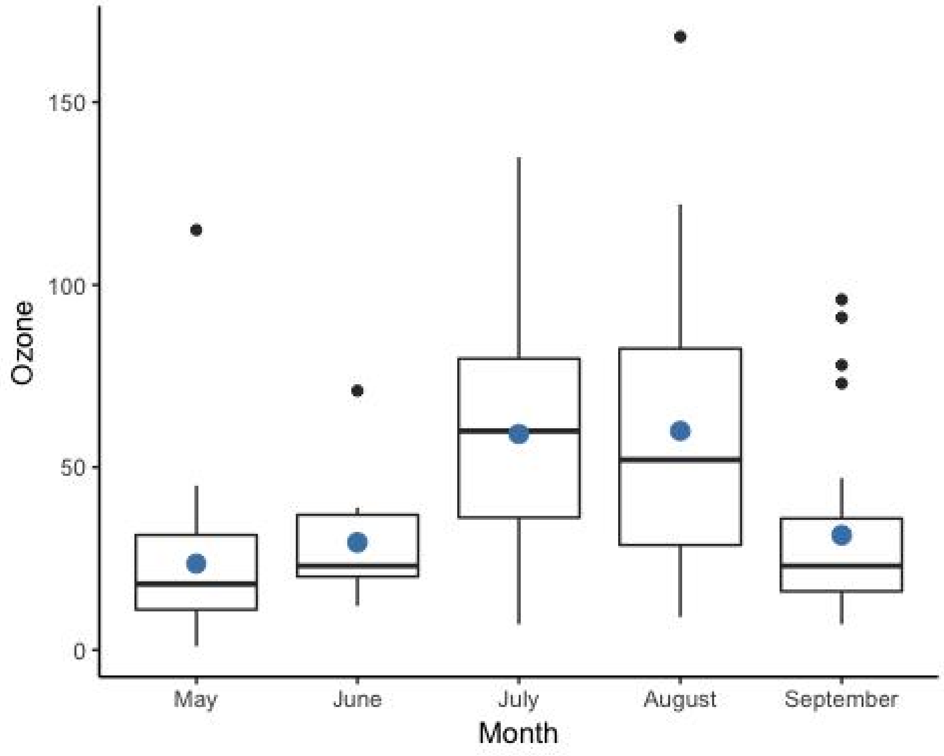


Code Explanation

- outlier.colour="red": Control the color of the outliers
- outlier.shape=2: Change the shape of the outlier. 2 refers to triangle
- outlier.size=3: Change the size of the triangle. The size is proportional to the number.

Add a summary statistic

```
box_plot + geom_boxplot() + stat_summary(fun.y = mean,
geom = "point", size = 3, color = "steelblue") + theme_classic()
```

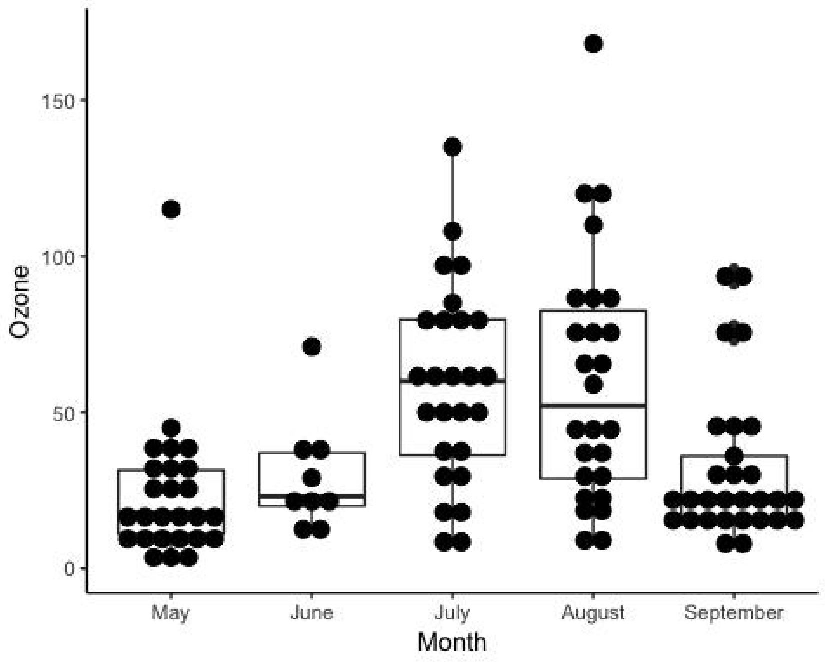


Code Explanation

- stat_summary() allows adding a summary to the horizontal boxplot R
- The argument fun.y controls the statistics returned. You will use mean
- Note: Other statistics are available such as min and max. More than one statistics can be exhibited in the same graph
- geom = “point”: Plot the average with a point
- size=3: Size of the point
- color =”steelblue”: Color of the points

Box Plot with Dots

```
box_plot + geom_boxplot() + geom_dotplot(binaxis = 'y',
dotsize = 1, stackdir = 'center') + theme_classic()
```

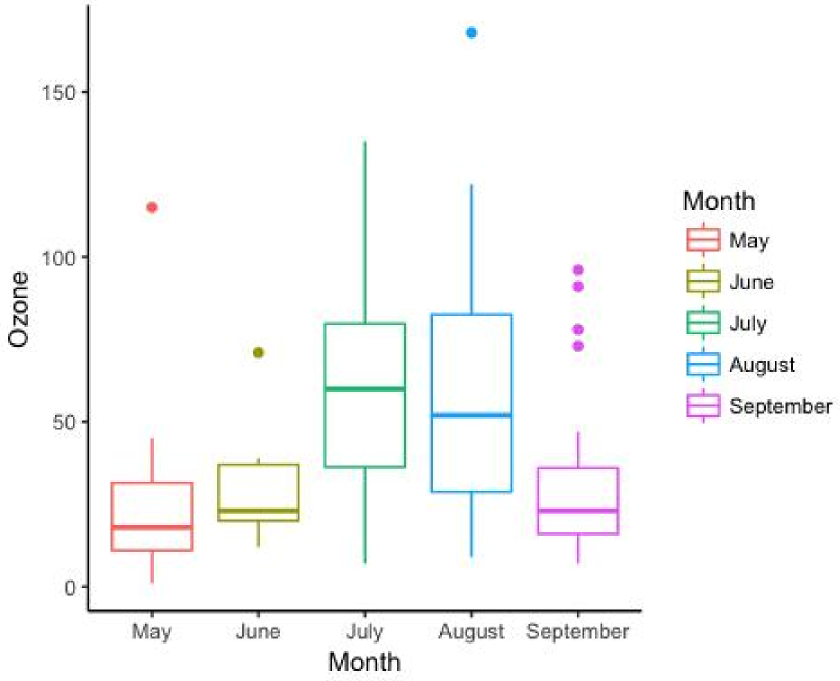


Code Explanation

- geom_dotplot() allows adding dot to the bin width
- binaxis='y': Change the position of the dots along the y-axis. By default, x-axis
- dotsize=1: Size of the dots
- stackdir='center': Way to stack the dots: Four values:
  - "up" (default),
  - "down"
  - "center"
  - "centerwhole"

Change the color of the box

```
ggplot(data_air_nona, aes(x = Month, y = Ozone, color =
Month)) + geom_boxplot() + theme_classic()
```

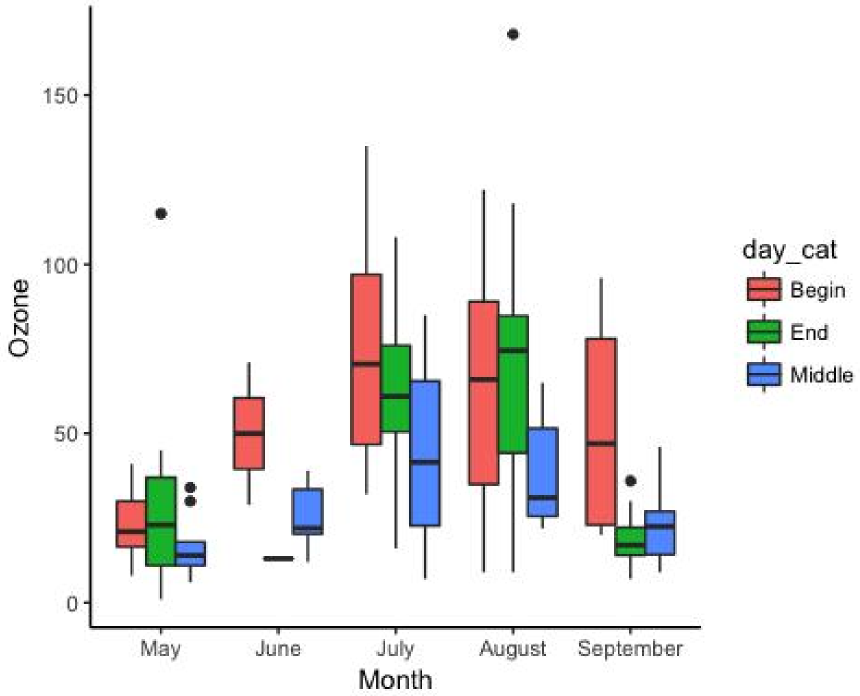


Code Explanation

•The colors of the groups are controlled in the aes() mapping. You can use color= Month to change the color of the box and whisker plot according to the months

Box plot with multiple groups

```
ggplot(data_air_nona, aes(Month, Ozone)) +
geom_boxplot(aes(fill = day_cat)) + theme_classic()
```



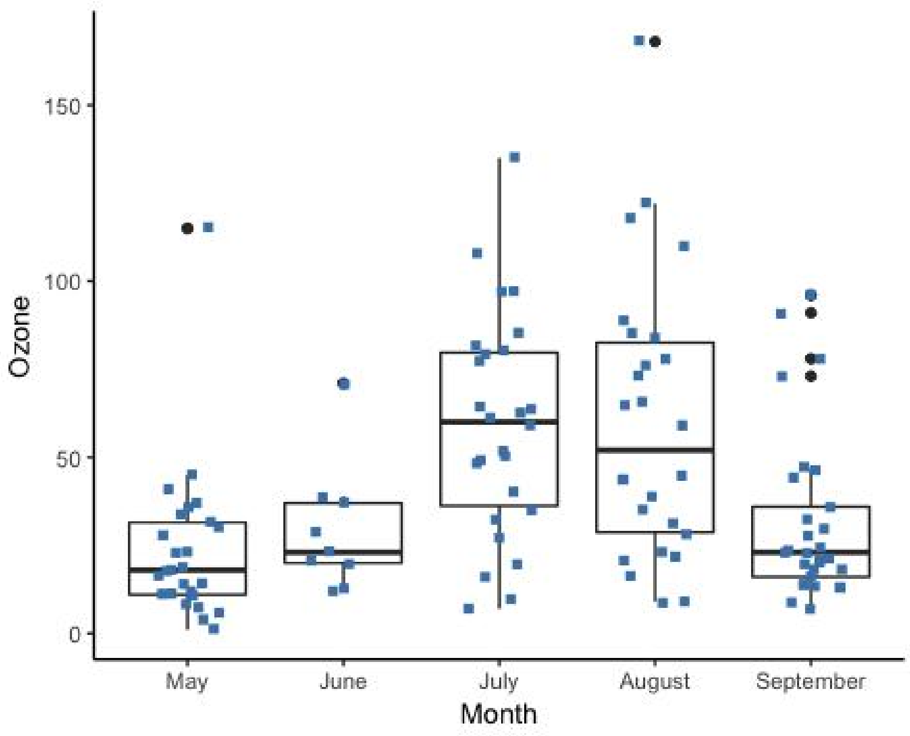
Code Explanation

•The aes() mapping of the geometric object controls the groups to display (this variable has to be a factor)

•aes(fill= day_cat) allows creating three boxes for each month in the x-axis

Box Plot with Jittered Dots

```
box_plot + geom_boxplot() + geom_jitter(shape = 15, color =
"steelblue", position = position_jitter(width = 0.21)) +
theme_classic()
```



Code Explanation

•geom_jitter() adds a little decay to each point.
•shape=15 changes the shape of the points. 15 represents the squares
•color = “steelblue”: Change the color of the point
•position=position_jitter(width = 0.21): Way to place the overlapping points. position_jitter(width = 0.21) means you move the points by 20 percent from the x-axis. By default, 40 percent.

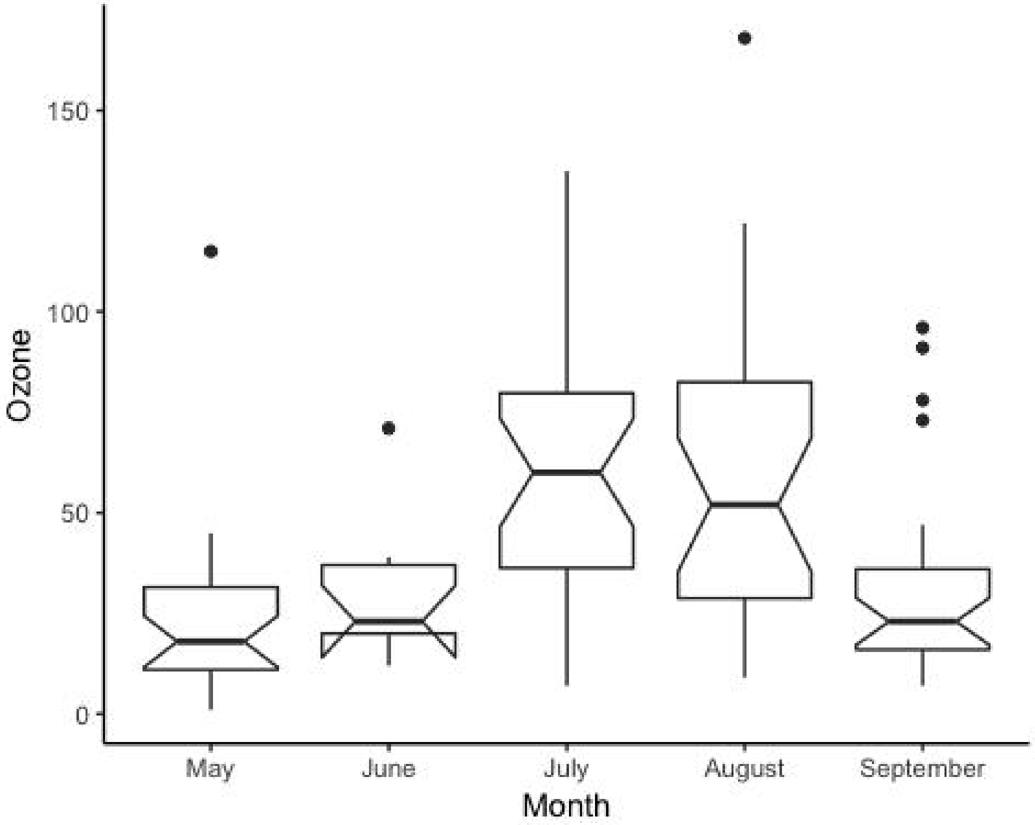
Notched Box Plot

An interesting feature of geom_boxplot(), is a notched boxplot function in R. The notch plot narrows the box around the median. The main purpose of a notched box plot is to compare the significance of the median between groups. There is strong evidence two groups have different medians when the notches do not overlap. A notch is computed as follow:

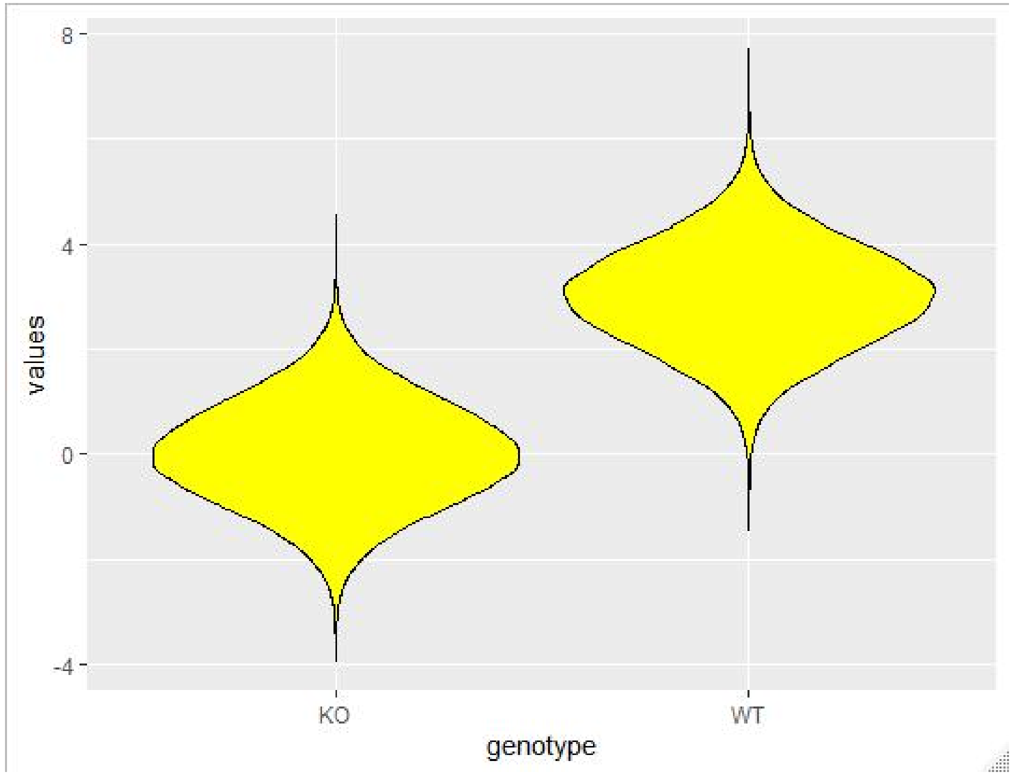
$$median \pm 1.57 * \frac{IQR}{\sqrt{n}}$$

with is the interquartile and number of observations.

```
box_plot + geom_boxplot(notch = TRUE) + theme_classic()
```



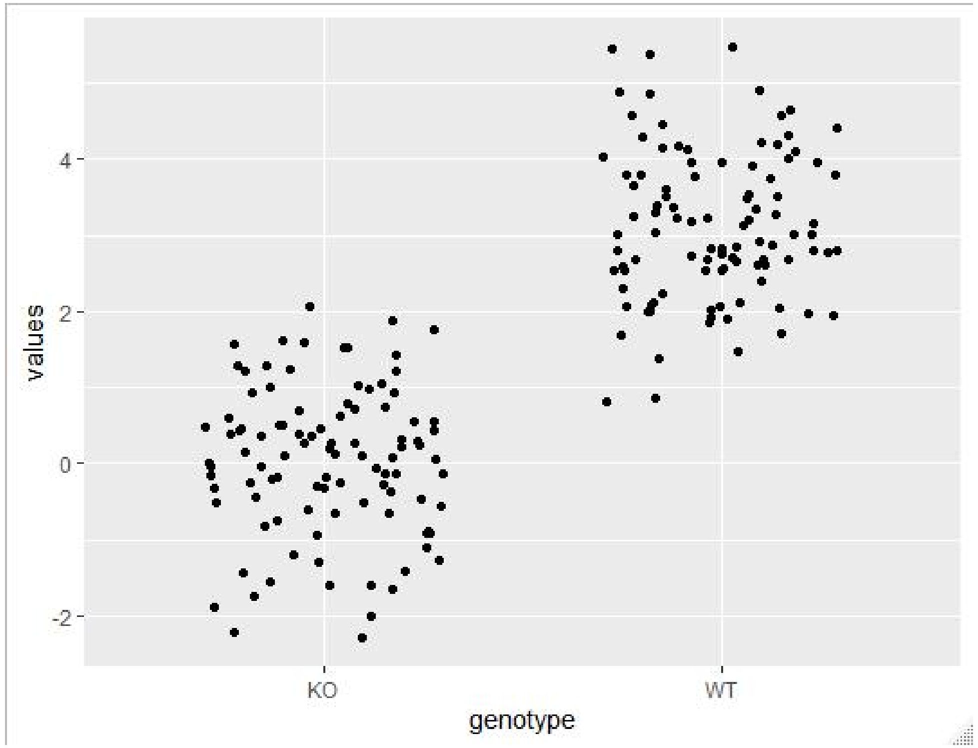
Plotting distributions – violin plots




```
> many.values
# A tibble: 100,000 x 2
   values genotype
    <dbl> <chr>
 1  1.90  KO
 2  2.39  WT
 3  4.32  KO
 4  2.94  KO
 5  0.728 WT
 6 -0.280 WT
 7  0.337 WT
 8 -1.31  WT
 9  1.55  WT
10  1.86  KO
```

```
many.values %>%
  ggplot(aes(x=genotype, y=values)) +
  geom_violin(colour="black", fill="yellow")
```

Plotting distributions – stripcharts




```
> many.values
# A tibble: 100,000 x 2
   values genotype
    <dbl> <chr>
 1  1.90   KO
 2  2.39   WT
 3  4.32   KO
 4  2.94   KO
 5  0.728 WT
 6 -0.280 WT
 7  0.337 WT
 8 -1.31   WT
 9  1.55   WT
10  1.86   KO
```

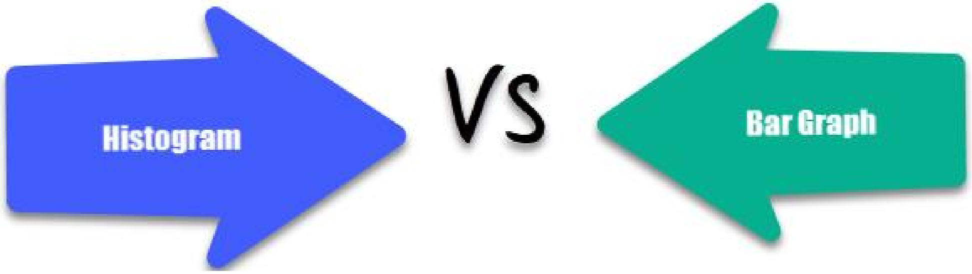
```
many.values %>%
  group_by(genotype) %>%
  sample_n(100) %>%
  ggplot(aes(x=genotype, y=values)) +
  geom_jitter(height=0, width = 0.3)
```

Summary

We can summarize the different types of horizontal boxplot R in the table below:

Objective	Code
Basic box plot	`ggplot(df, aes( x = x1, y =y)) + geom_boxplot()`
flip the side	`ggplot(df, aes( x = x1, y =y)) + geom_boxplot() + coord_flip()`
Notched box plot	`ggplot(df, aes( x = x1, y =y)) + geom_boxplot(notch=TRUE)`
Box plot with jittered dots	`ggplot(df, aes( x = x1, y =y)) + geom_boxplot() + geom_jitter(position = position_jitter(0.21))`

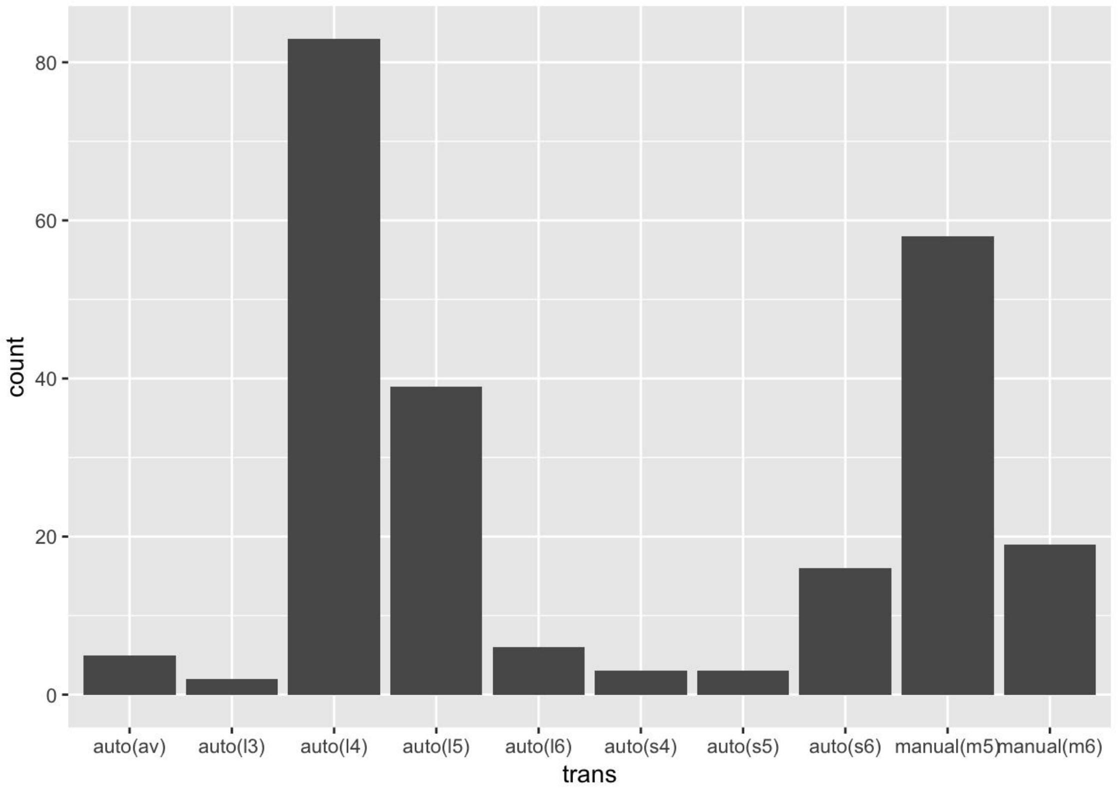
Barplots vs histograms



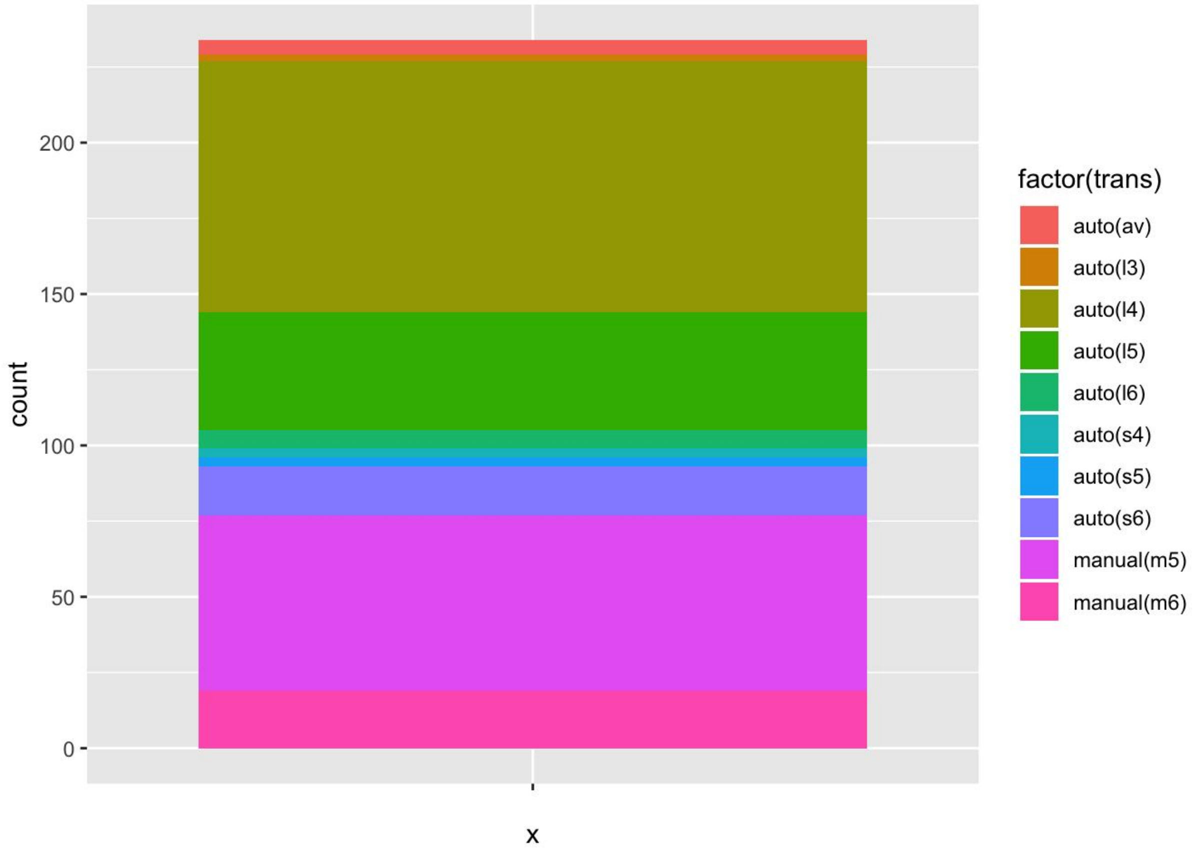
Histogram	Bar Graph
The histogram is a term that refers to a graphical representation that shows data by way of bars to display the frequency of numerical data.	The bar graph is a graphical representation of data that uses bars to compare different categories of data.
Distribution of non-discrete variables.	Comparison of discrete variables.
Bars touch each other, so there are no spaces between bars.	Bars never touch each other, so there are spaces between bars.
In this type of graph, elements are grouped so that they are considered as ranges.	In this type of graph, elements are taken as individual entities.
Histogram width may vary.	The bar chart is mostly of equal width.
To display the frequency of occurrences.	To compare different categories of data.
In Histogram, the data points are grouped and rendered based on its bin value.	In the Bar graph, each data point is rendered as a separate bar.
The items of the Histogram are numbers, which should be categorized to represent data range.	As opposed to the bar graph, items should be considered as individual entities.
In Histogram, we cannot rearrange the blocks.	Bar graph, it is common to rearrange the blocks, from highest to lowest

Barplots

```
ggplot(data = mpg) + geom_bar(aes(x = trans))
```

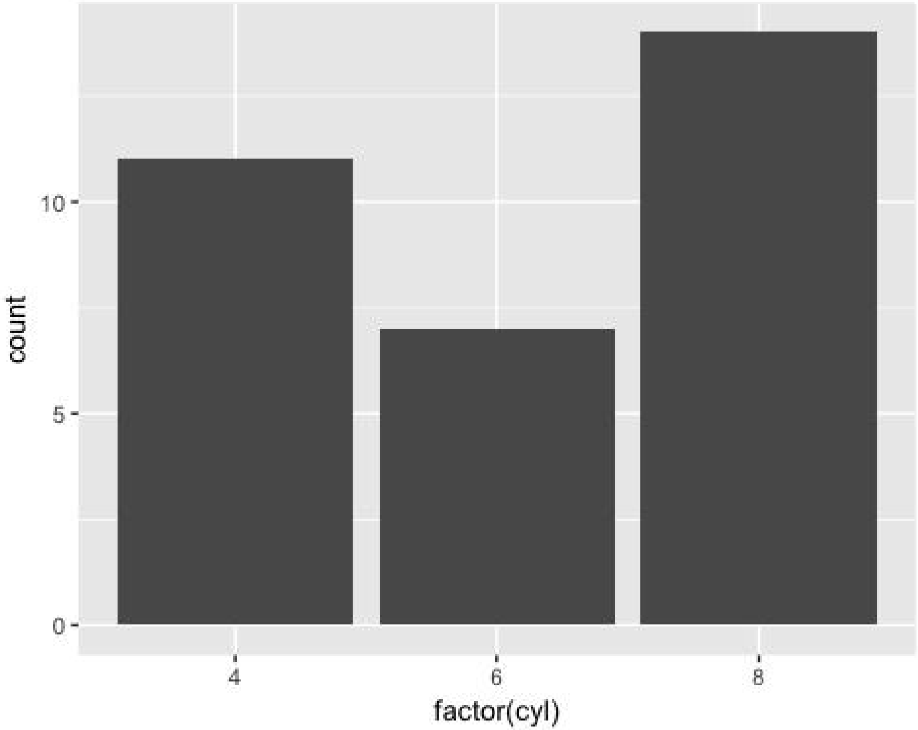



```
ggplot(data = mpg) + geom_bar(aes(x = "", fill = factor(trans)))
```



Bar chart: count

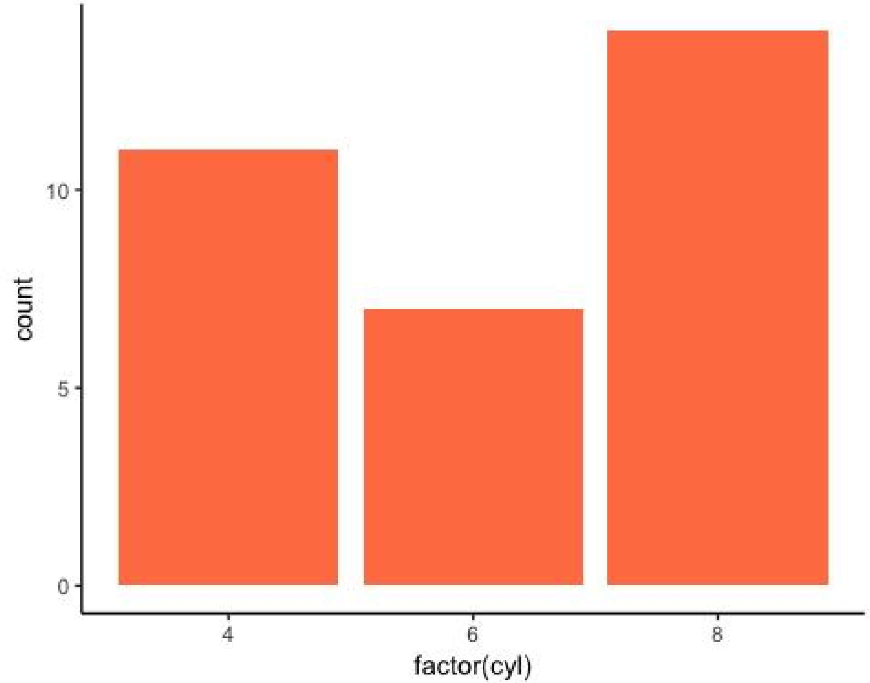
```
library(ggplot2)
# Most basic bar chart
ggplot(mtcars, aes(x = factor(cyl))) + geom_bar()
```



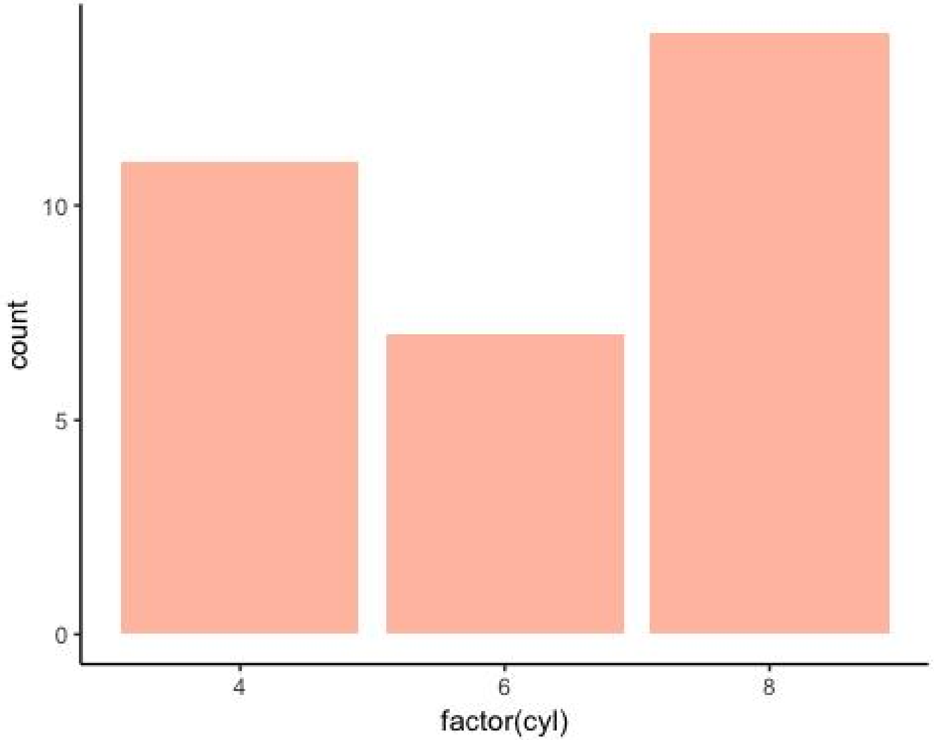
- You pass the dataset mtcars to ggplot.
- Inside the aes() argument, you add the x-axis as a factor variable(cyl)
- The + sign means you want R to keep reading the code. It makes the code more readable by breaking it.
- Use geom_bar() for the geometric object.

Change the color of the bars

```
# Change the color of the bars
ggplot(mtcars, aes(x = factor(cyl))) + geom_bar(fill = "coral") +
theme_classic()
```

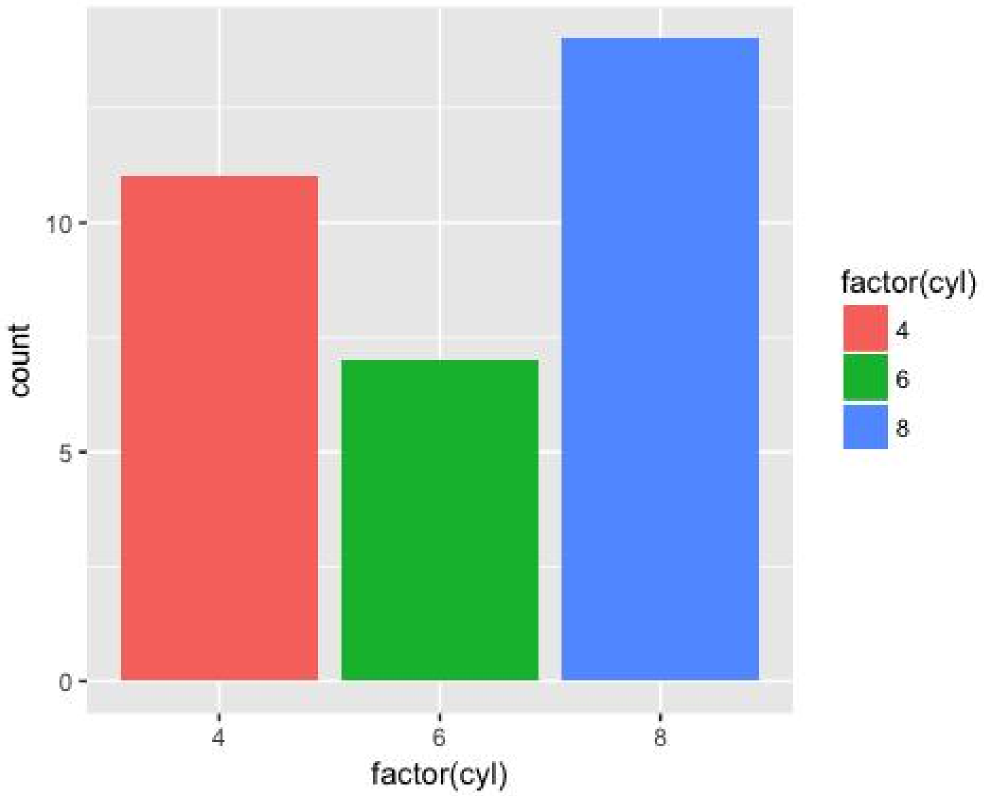



```
# Change intensity
ggplot(mtcars, aes(factor(cyl))) + geom_bar(fill =
"coral", alpha = 0.5) + theme_classic()
```



Color by groups

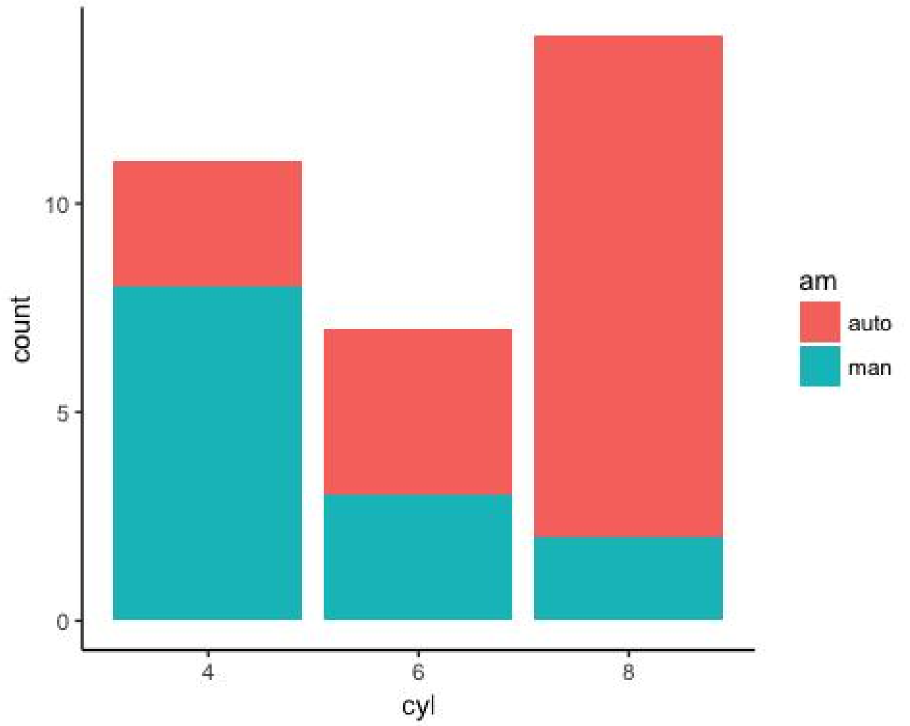
```
# Color by group
ggplot(mtcars, aes(factor(cyl), fill = factor(cyl))) + geom_bar()
```



Add a group in the bars

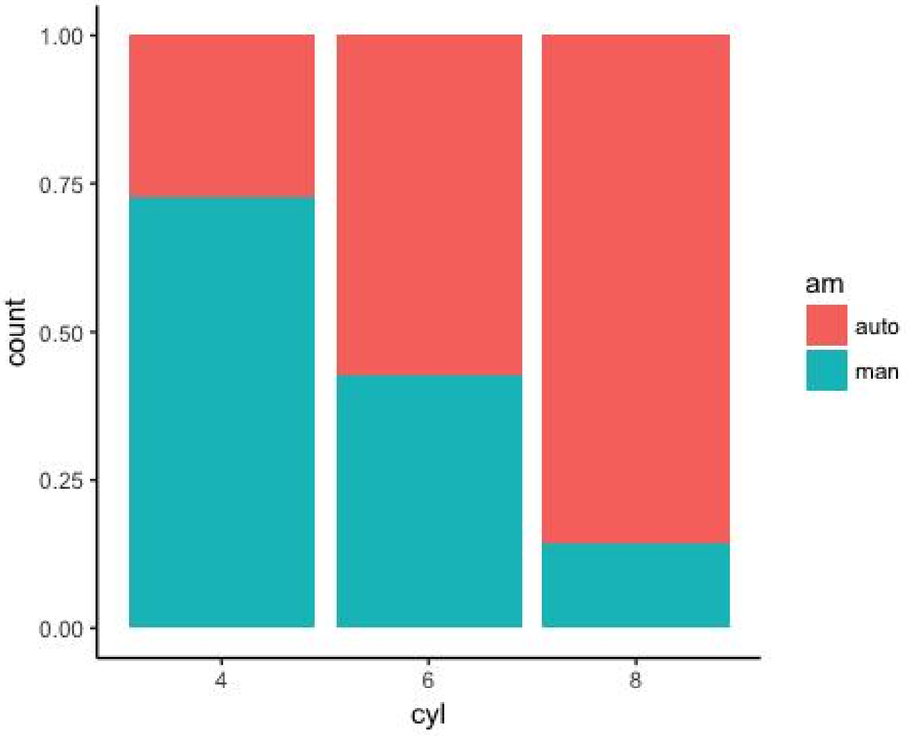
You will proceed as follow:
•Step 1: Create the data frame with mtcars dataset
•Step 2: Label the am variable with auto for automatic transmission and man for manual transmission. Convert am and cyl as a factor so that you don’t need to use factor() in the ggplot() function.
•Step 3: Plot the bar chart to count the number of transmission by cylinder

```
library(dplyr)
# Step 1
data <- mtcars % > %
#Step 2
mutate(am = factor(am, labels = c("auto", "man")), cyl =
factor(cyl))
#Step 3
ggplot(data, aes(x = cyl, fill = am)) + geom_bar() +
theme_classic()
```



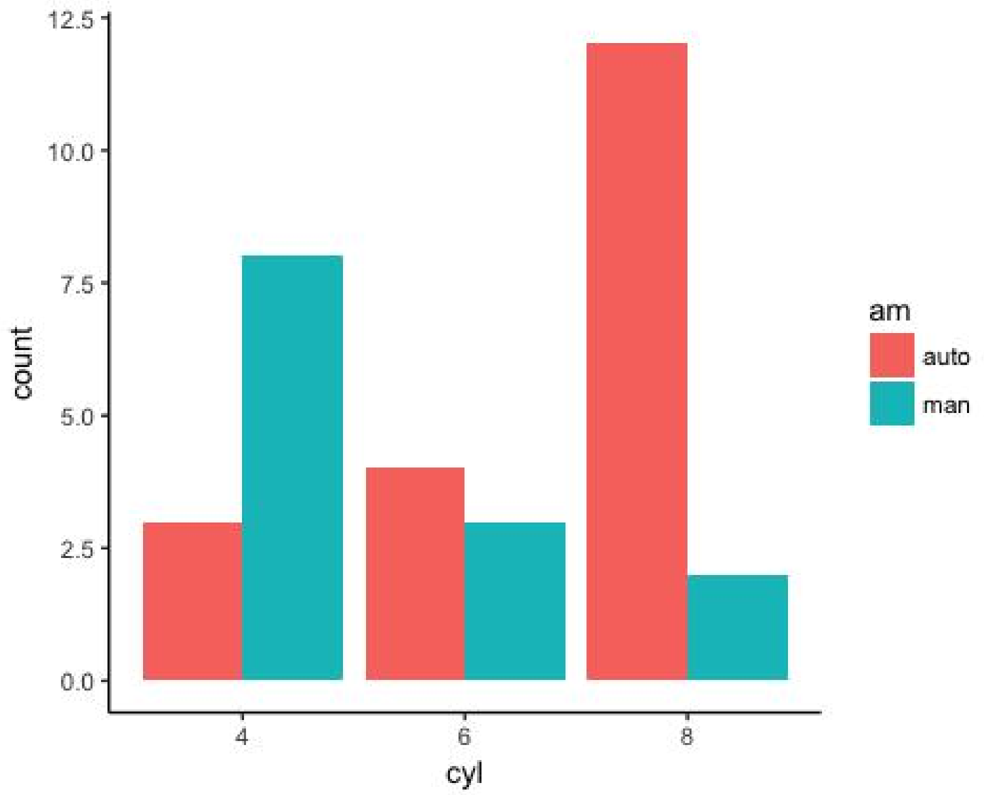
Bar chart in percentage

```
ggplot(data, aes(x = cyl, fill = am)) + geom_bar(position = "fill")
+ theme_classic()
```



Side by side bars

```
# Bar chart side by side
ggplot(data, aes(x = cyl, fill = am)) + geom_bar(position =
position_dodge()) + theme_classic()
```

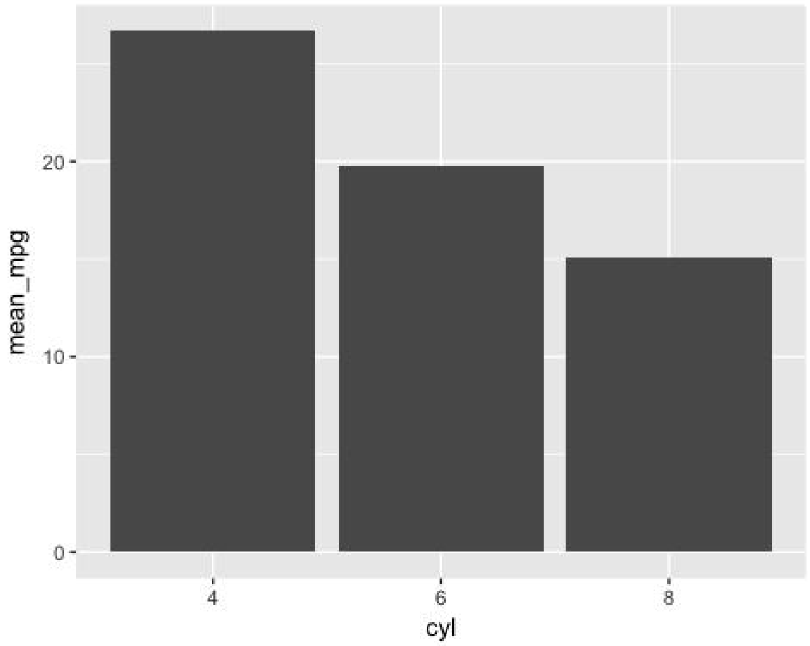


Bar plot

Your objective is to create a graph with the average mile per gallon for each type of cylinder.

- Step 1: Create a new variable with the average mile per gallon by cylinder
- Step 2: Create a basic bp
- Step 3: Change the orientation
- Step 4: Change the color
- Step 5: Change the size
- Step 6: Add labels to the graph

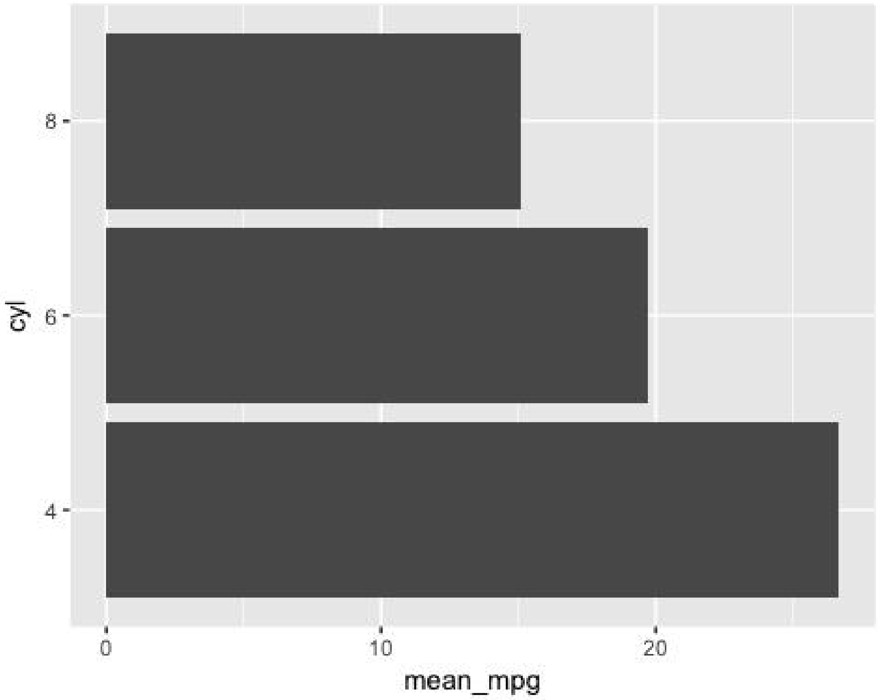
```
# Step 1
data_pb <- mtcars % > % mutate(cyl = factor(cyl)) % > %
group_by(cyl) % > % summarize(mean_mpg =
round(mean(mpg), 2))
# Step 2
ggplot(data_bp, aes(x = cyl, y = mean_mpg)) + geom_bar(stat =
"identity")
```



Bar plot

Step 3) Change the orientation

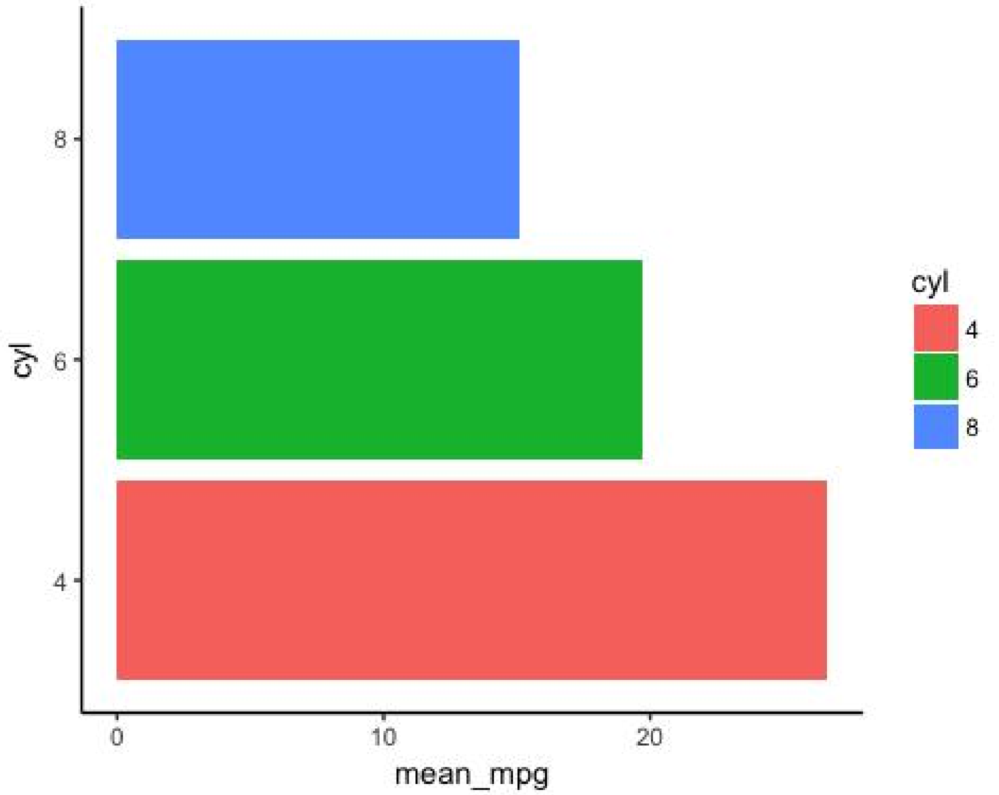
```
ggplot(data_histogram, aes(x = cyl, y = mean_mpg)) +
geom_bar(stat = "identity") + coord_flip()
```



Bar plot

Step 4) Change the color

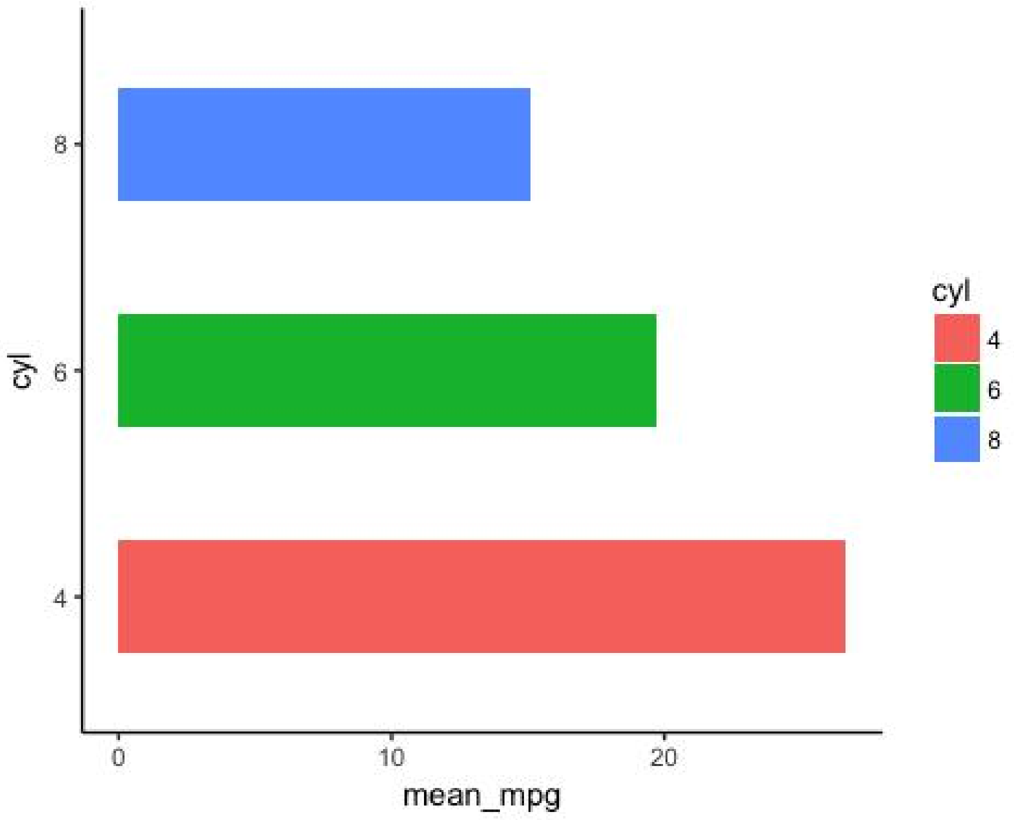
```
ggplot(data_bp, aes(x = cyl, y = mean_mpg, fill = cyl)) +
geom_bar(stat = "identity") + coord_flip() + theme_classic()
```



Bar plot

Step 5) Change the size

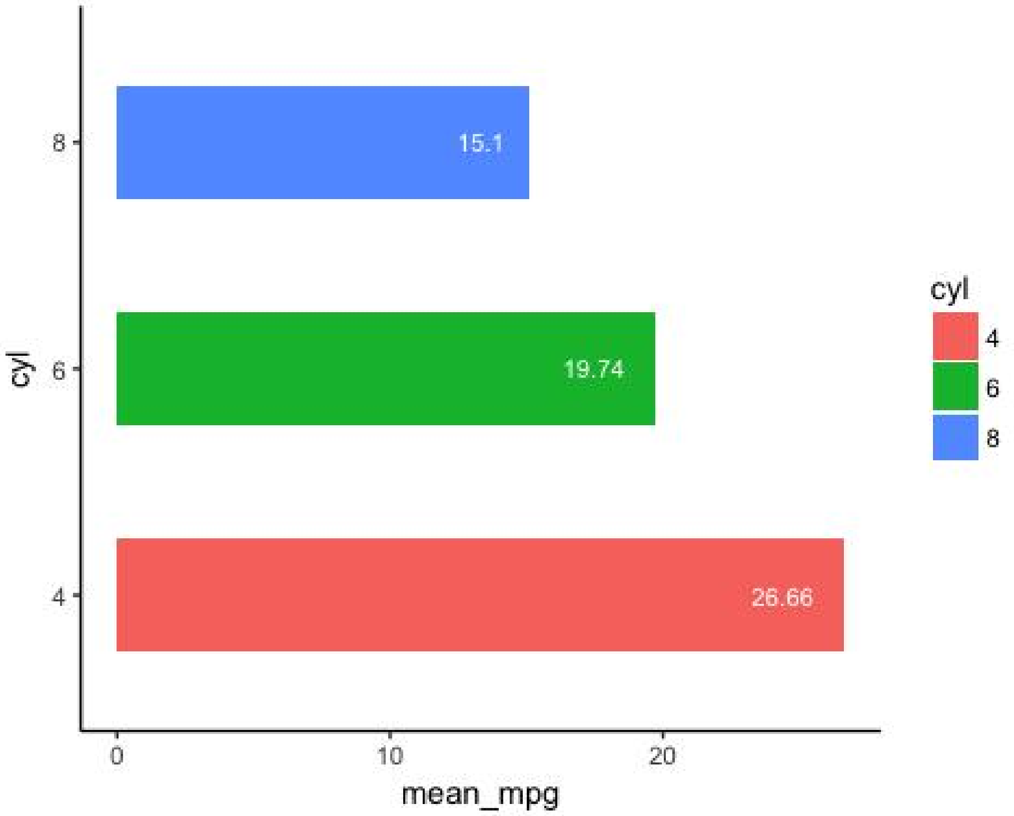
```
graph <- ggplot(data_bp, aes(x = cyl, y = mean_mpg, fill = cyl)) +
geom_bar(stat = "identity", width = 0.5) + coord_flip() +
theme_classic()
```



Bar plot

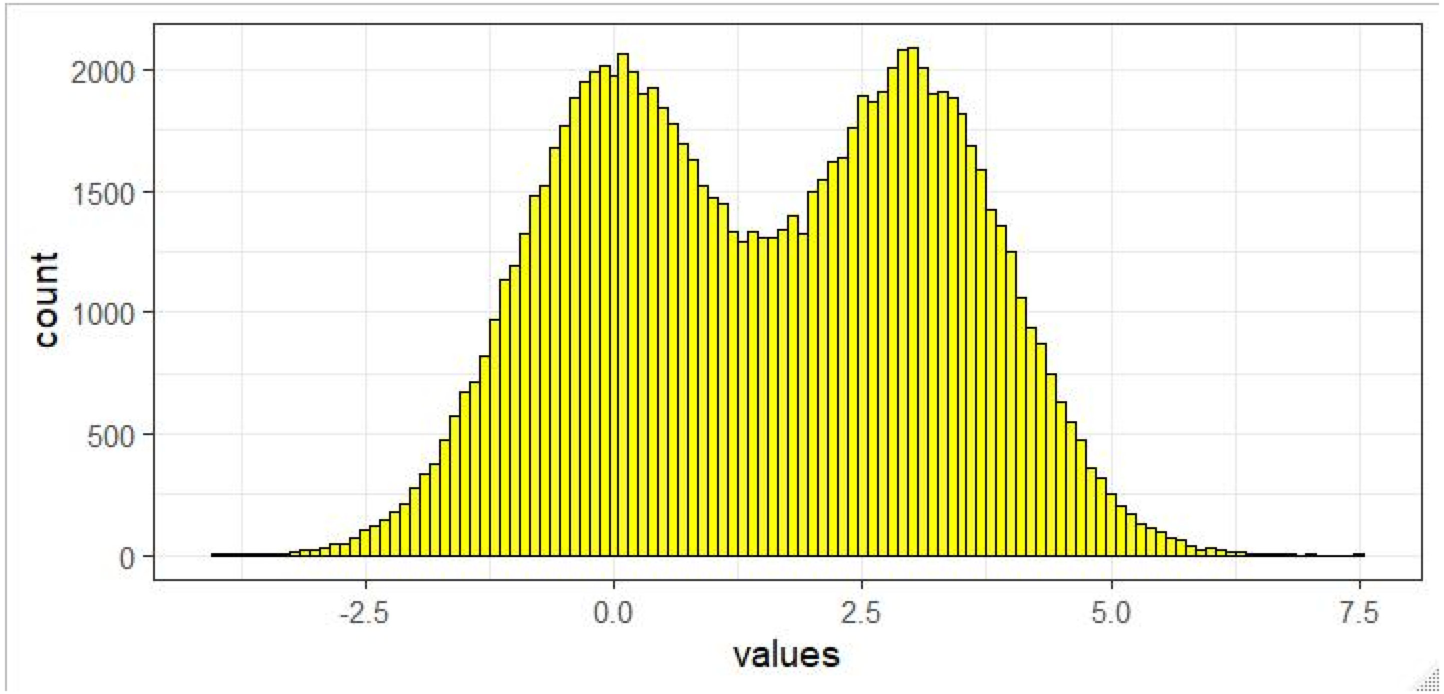
Step 6) Add labels to the graph

graph + geom_text(aes(label = mean_mpg), hjust = 1.5, color = "white", size = 3) + theme_classic()



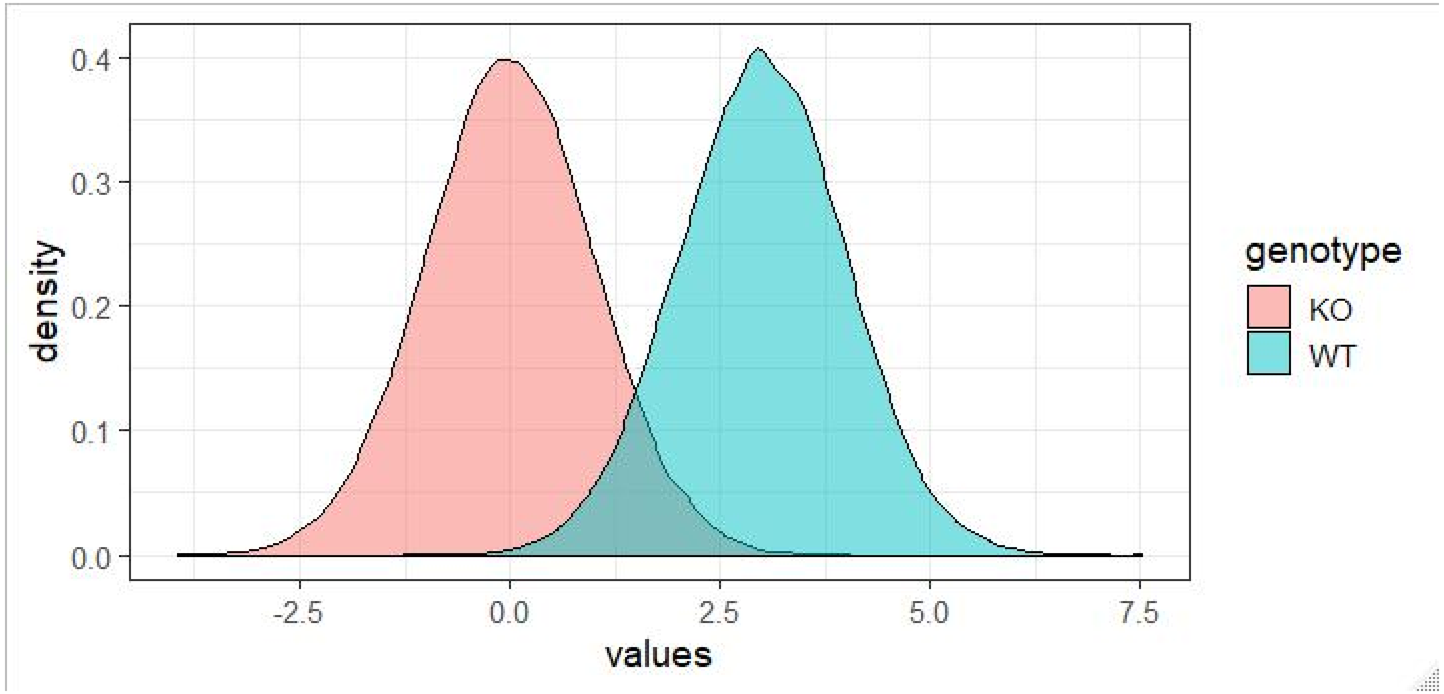
Plotting distributions - histograms

```
> many.values
# A tibble: 100,000 x 2
   values genotype
    <dbl> <chr>
 1  1.90  KO
 2  2.39  WT
 3  4.32  KO
 4  2.94  KO
 5  0.728 WT
 6 -0.280 WT
 7  0.337 WT
 8 -1.31  WT
 9  1.55  WT
10  1.86  KO
```




```
many.values %>%
  ggplot(aes(x=values)) +
  geom_histogram(binwidth = 0.1, fill="yellow", colour="black")
```

Plotting distributions - density




```
> many.values
# A tibble: 100,000 x 2
   values genotype
    <dbl> <chr>
 1  1.90  KO
 2  2.39  WT
 3  4.32  KO
 4  2.94  KO
 5  0.728 WT
 6 -0.280 WT
 7  0.337 WT
 8 -1.31  WT
 9  1.55  WT
10  1.86  KO
```

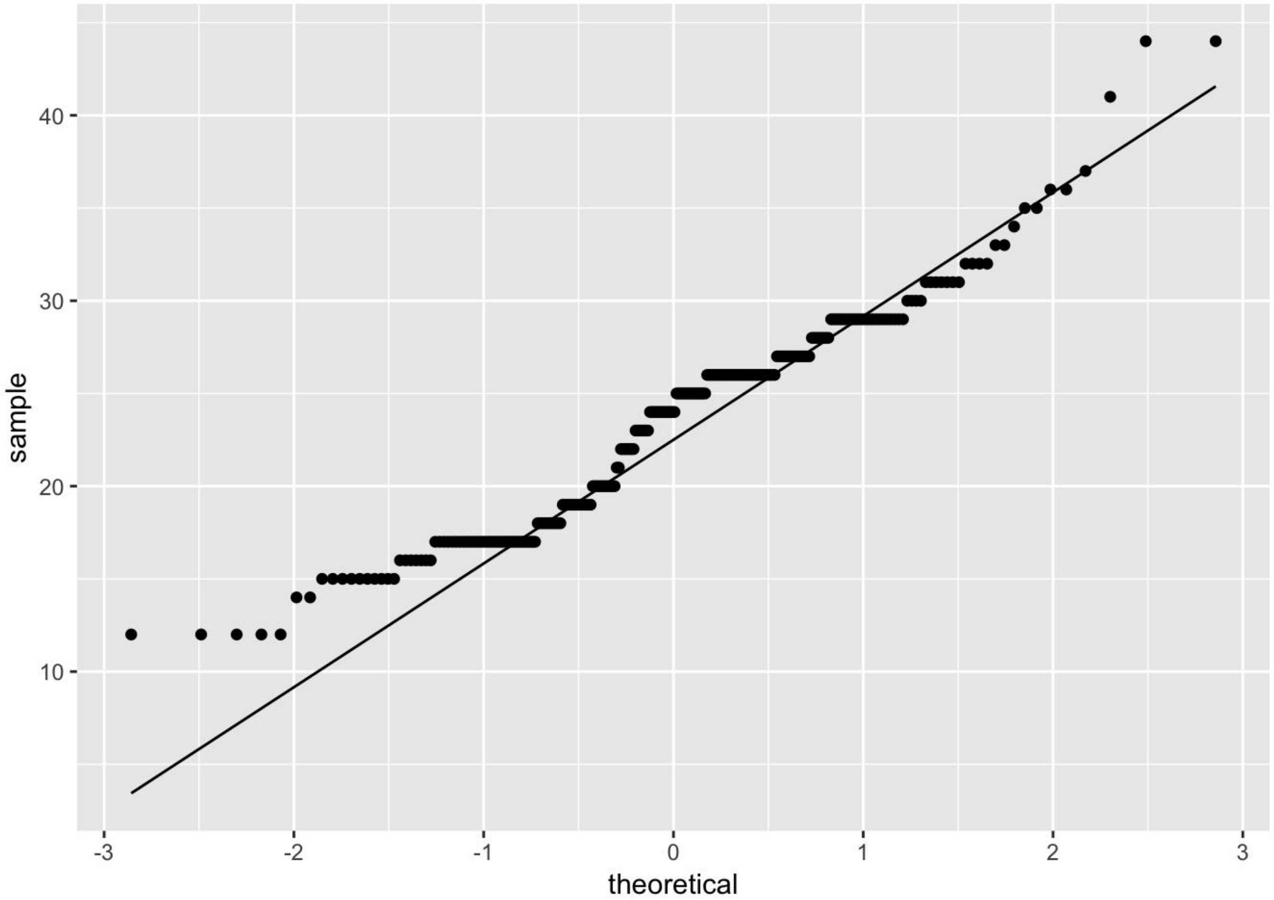
```
many.values %>%
  ggplot(aes(x=values, fill=genotype)) +
  geom_density(colour="black", alpha=0.5)
```

summary

Objective	Code
Count	`ggplot(df, eas(x= factor(x1)) + geom_bar()`
Count with different color of fill	`ggplot(df, eas(x= factor(x1), fill = factor(x1))) + geom_bar()`
Count with groups, stacked	`ggplot(df, eas(x= factor(x1), fill = factor(x2))) + geom_bar(position=position_dodge())`
Count with groups, side by side	`ggplot(df, eas(x= factor(x1), fill = factor(x2))) + geom_bar()`
Count with groups, stacked in %	`ggplot(df, eas(x= factor(x1), fill = factor(x2))) + geom_bar(position=position_dodge())`
Values	`ggplot(df, eas(x= factor(x1)+ y = x2) + geom_bar(stat="identity")`

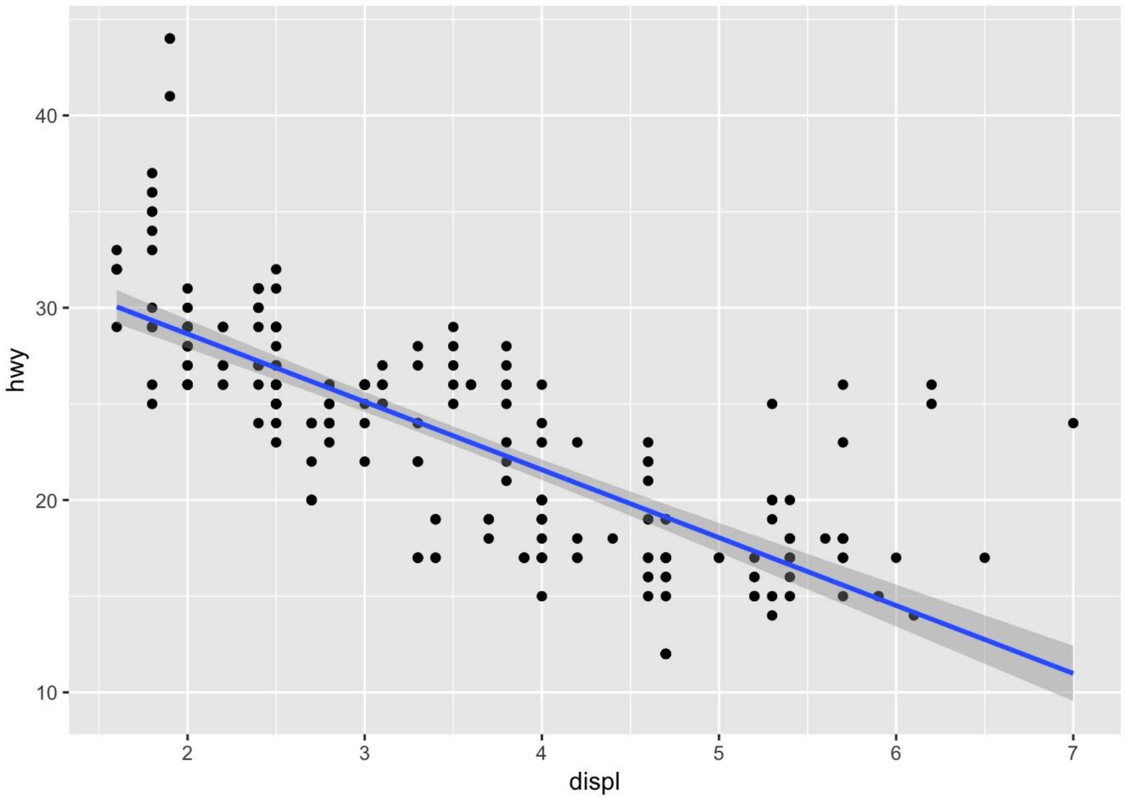
Compare distributions

```
ggplot(data = mpg) + geom_qq(aes(sample = hwy)) + geom_qq_line(aes(sample = hwy))
```



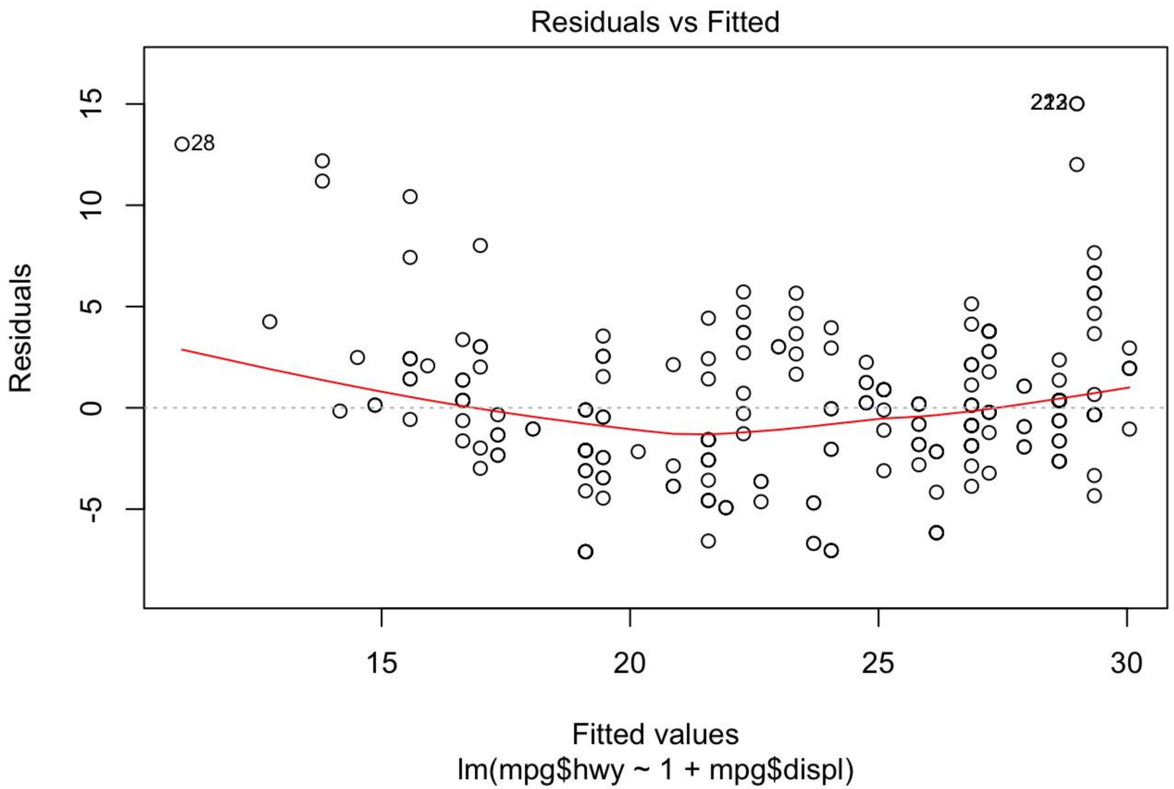
Linear model

gplot(data = mpg) + **geom_point**(mapping = **aes**(x = displ, y = hwy)) + **geom_smooth**(mapping = **aes**(x = displ, y = hwy),method = 'lm')



Linear model

```
out = lm(mpg$hwy ~ 1 + mpg$displ)
plot(out)
```



Titles and axis labels

- Can add calls to functions to set them individually
  - `ggtitle("Main title")`
  - `xlab("X axis")`
  - `ylab("Y axis")`

- Can set them all together with `labs()`
  - `title="Main title"`
  - `x="X axis"`
  - `y="Y axis"`

Changing scaling

- Alter the data before plotting
  - `mutate(value=log(value))`

- Alter the data whilst plotting
  - `ggplot(aes(x=log(value)))`

- Alter the scale of the plot
  - Add an option to adjust the scaling of the axis

Axis scaling options

- Transforming scales
  - `scale_x_log10()`
  - `scale_x_sqrt()`
  - `scale_x_reverse()`

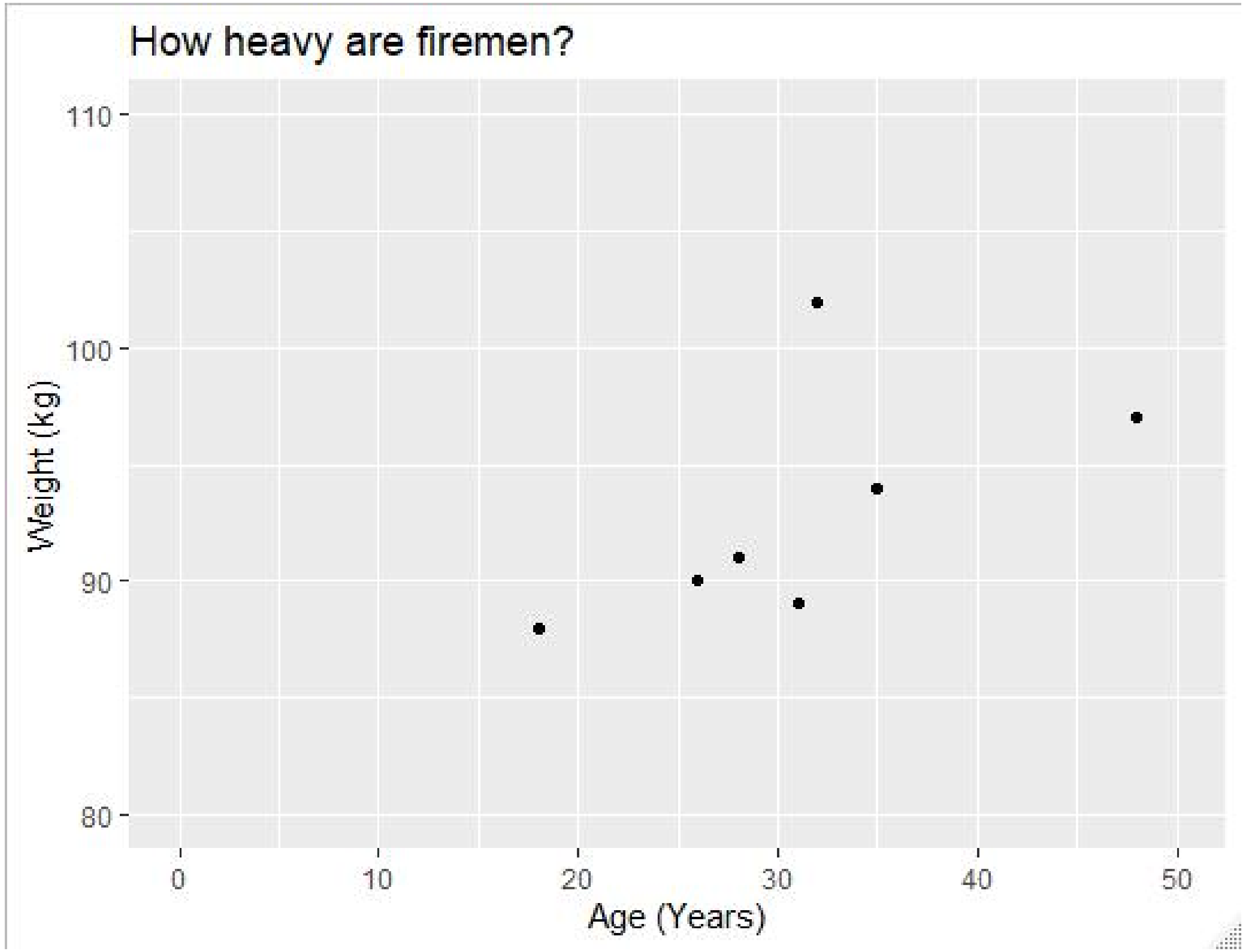
  Equivalent `_y_` versions also exist

- Switching axes
  - `coord_flip()`

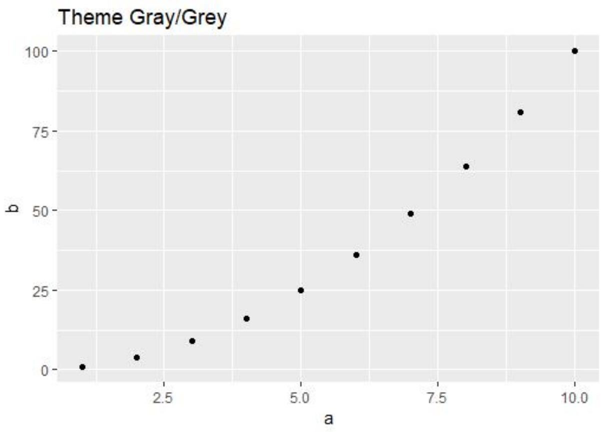
- Adjusting ranges
  - `scale_x_continuous()`
    - `limits=c(-5,5)`
    - `breaks=seq(from=-5,by=2,to=5)`
    - `minor_breaks`
    - `labels`
  - `coord_cartesian()`
    - `xlim=c(-5,5)`
    - `ylim=c(10,20)`

Annotation and scaling example

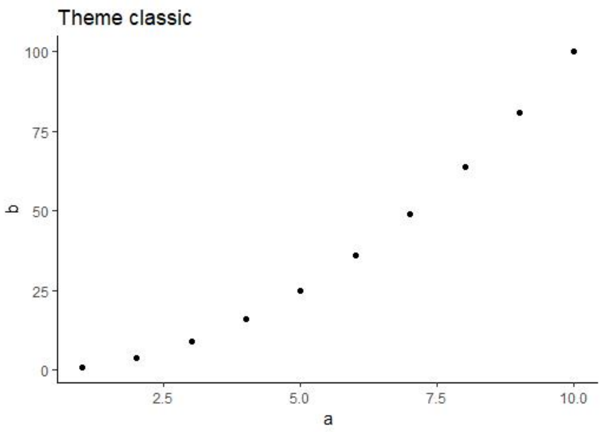
```
trumpton %>%
  ggplot(aes(x=Age, y=Weight))+
  geom_point() +

  xlab("Age (Years)")+
  ylab("Weight (kg)")+
  ggtitle("How heavy are firemen?")+

  coord_cartesian(
       xlim=c(0,50),
       ylim=c(80,110)
  )
```

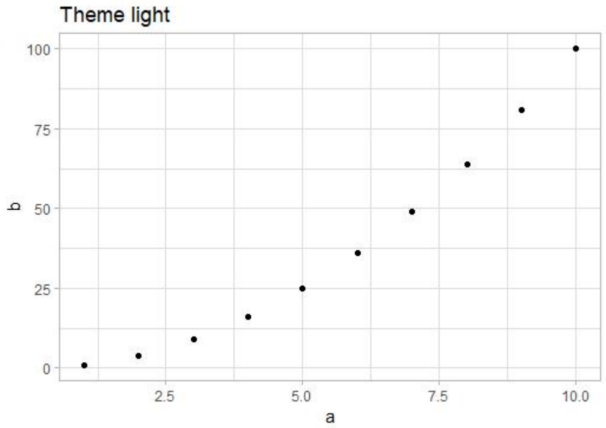


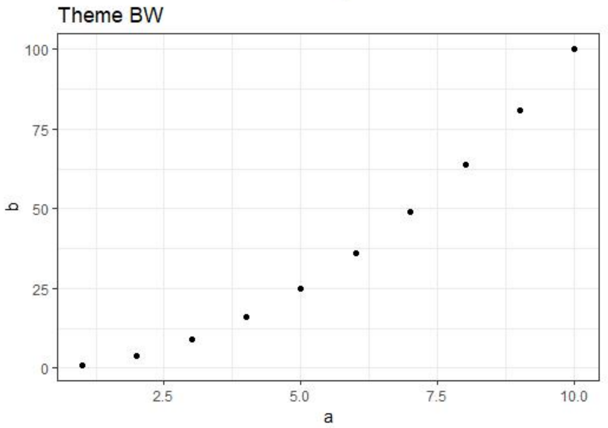
ggPlot Themes

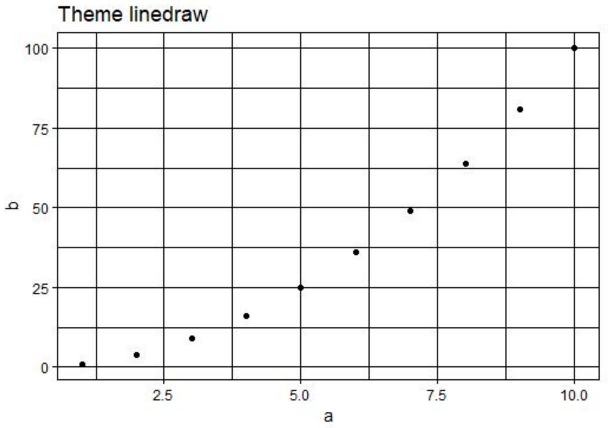


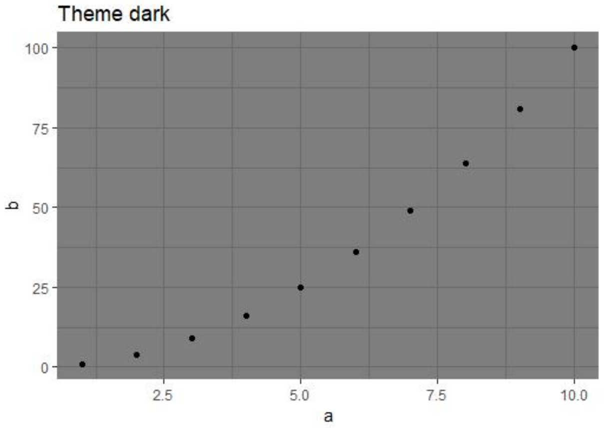
- `theme_grey()`
- `theme_bw()`
- `theme_dark()`
- `theme_light()`
- `theme_minimal()`
- `theme_classic()`
- `theme_linedraw()`

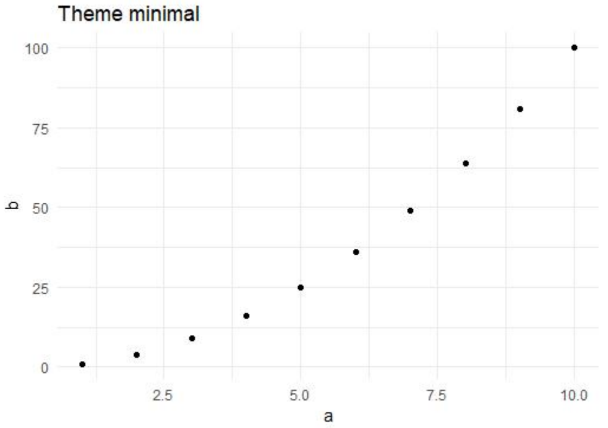












Setting and Customising themes

- Globally
  `theme_set(theme_bw(base_size=14))`

- In a single plot
  `+theme_dark()`

Customising themes

```
theme_update(plot.title = element_text(hjust = 0.5))

plot + theme(plot.title = element_text(hjust = 0.5))
```

```
theme(line, rect, text, title, aspect.ratio, axis.title, axis.title.x, axis.title.x.top, axis.title.x.bottom,
axis.title.y, axis.title.y.left,axis.title.y.right, axis.text, axis.text.x, axis.text.x.top,
axis.text.x.bottom, axis.text.y, axis.text.y.left, axis.text.y.right, axis.ticks, axis.ticks.x,
axis.ticks.x.top, axis.ticks.x.bottom, axis.ticks.y, axis.ticks.y.left, axis.ticks.y.right, axis.ticks.length,
axis.line, axis.line.x, axis.line.x.top, axis.line.x.bottom, axis.line.y, axis.line.y.left, axis.line.y.right,
legend.background, legend.margin, legend.spacing, legend.spacing.x, legend.spacing.y, legend.key,
legend.key.size, legend.key.height, legend.key.width, legend.text, legend.text.align, legend.title,
legend.title.align, legend.position, legend.direction, legend.justification, legend.box, legend.box.just,
legend.box.margin, legend.box.background, legend.box.spacing, panel.background, panel.border, panel.spacing,
panel.spacing.x, panel.spacing.y, panel.grid, panel.grid.major, panel.grid.minor, panel.grid.major.x,
panel.grid.major.y, panel.grid.minor.x, panel.grid.minor.y, panel.ontop, plot.background, plot.title,
plot.subtitle, plot.caption, plot.tag, plot.tag.position, plot.margin, strip.background, strip.background.x,
strip.background.y, strip.placement, strip.text, strip.text.x, strip.text.y, strip.switch.pad.grid,
strip.switch.pad.wrap)
```

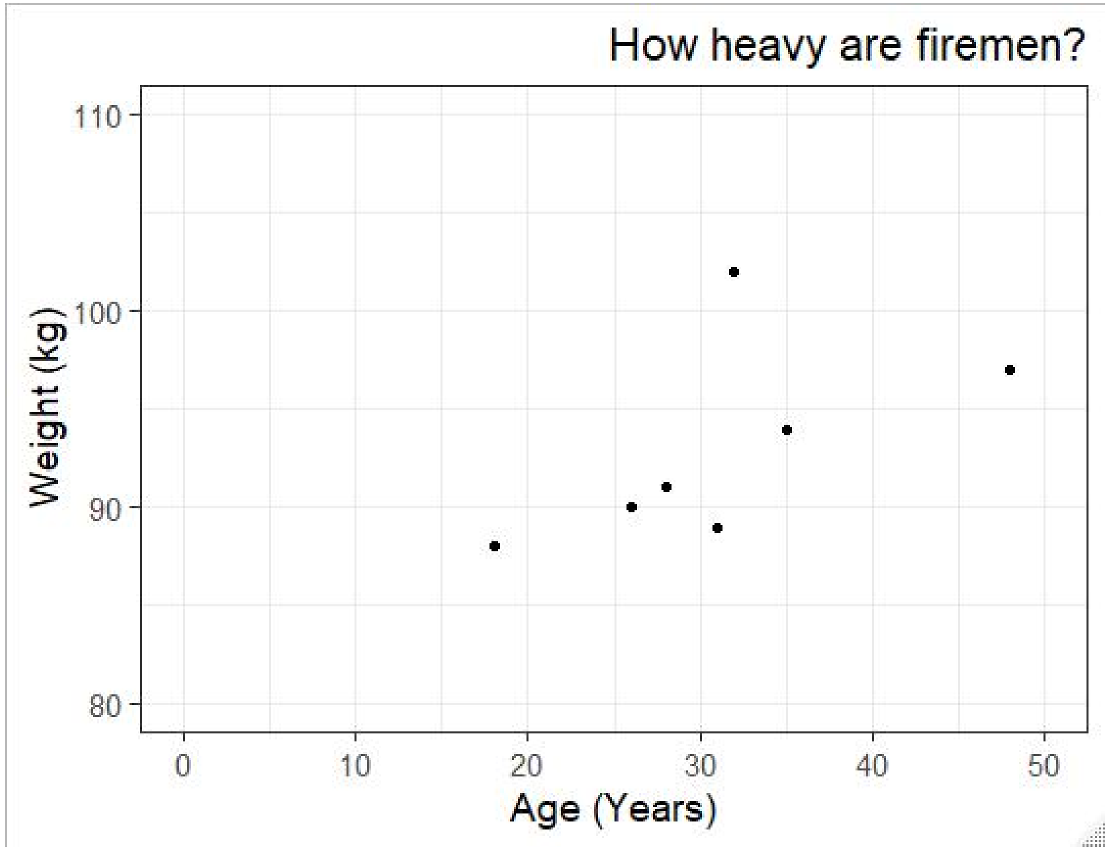
https://ggplot2.tidyverse.org/reference/theme.html

Theme setting example

```
theme_set(theme_bw(base_size = 14))
theme_update(plot.title = element_text(hjust=1))
```

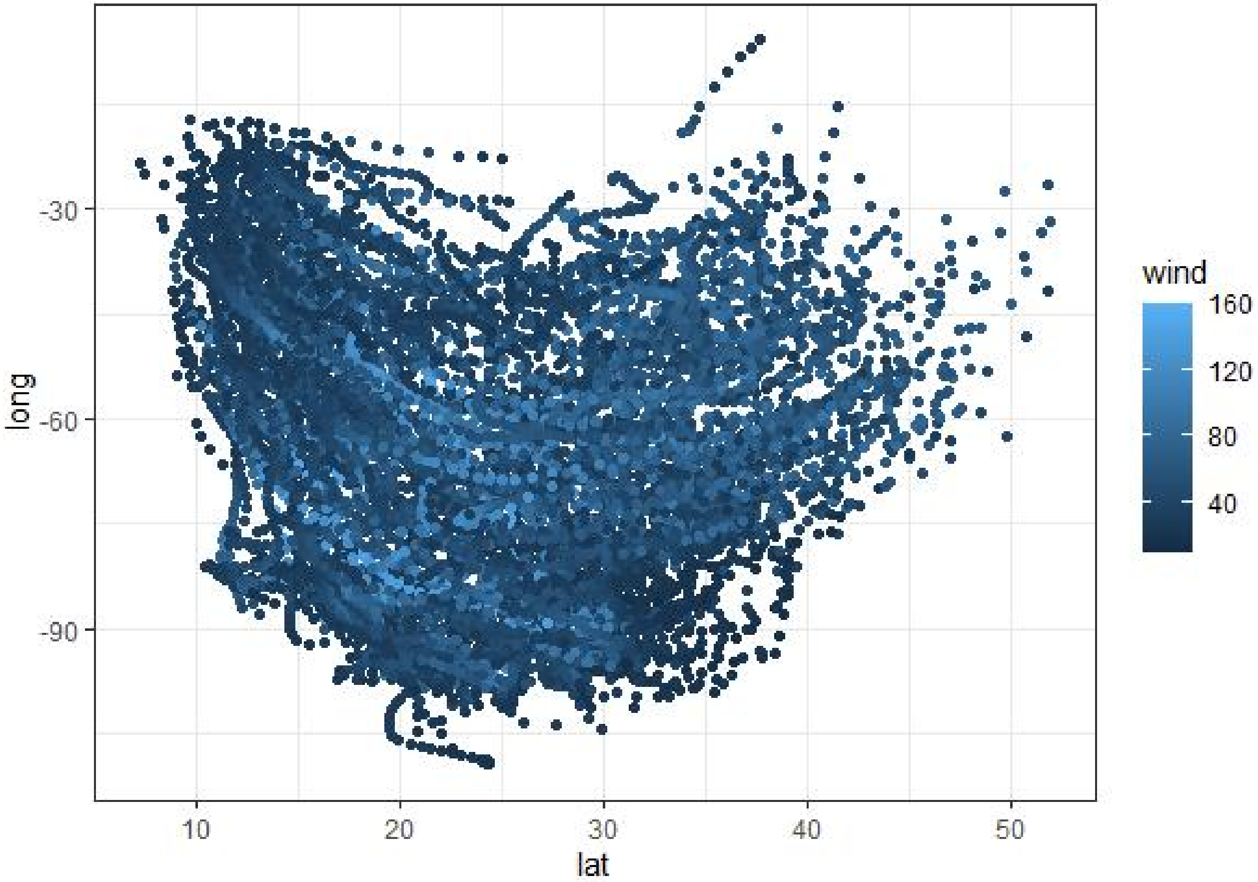
OR

```
my.plot +
theme_bw(base_size = 14) +
theme(plot.title = element_text(hjust=1))
```



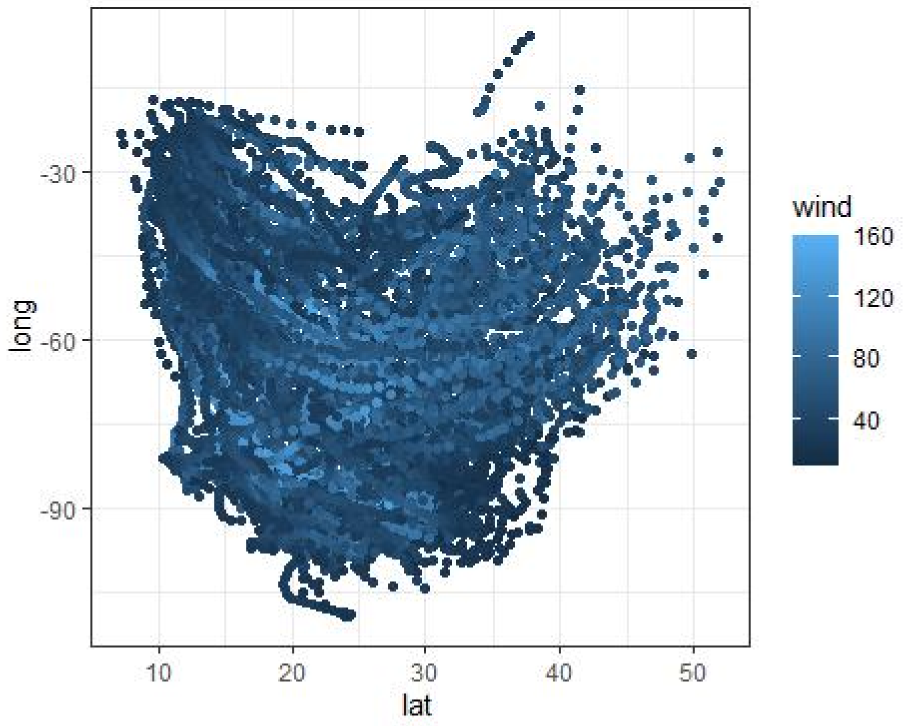
Changing Quantitative Colours

```
storms %>%
  ggplot(aes(x=lat, y=long, colour=wind)) +
  geom_point()
```

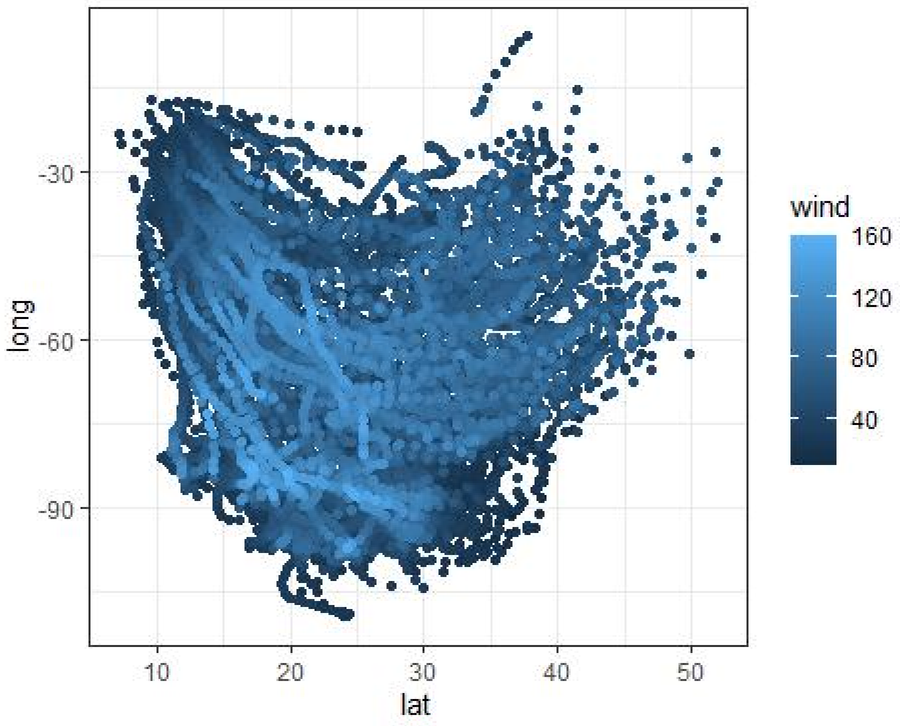


Changing Plotting Order

```
storms %>%
 ggplot(aes(x=lat,y=long,colour=wind))+
 geom_point()
```

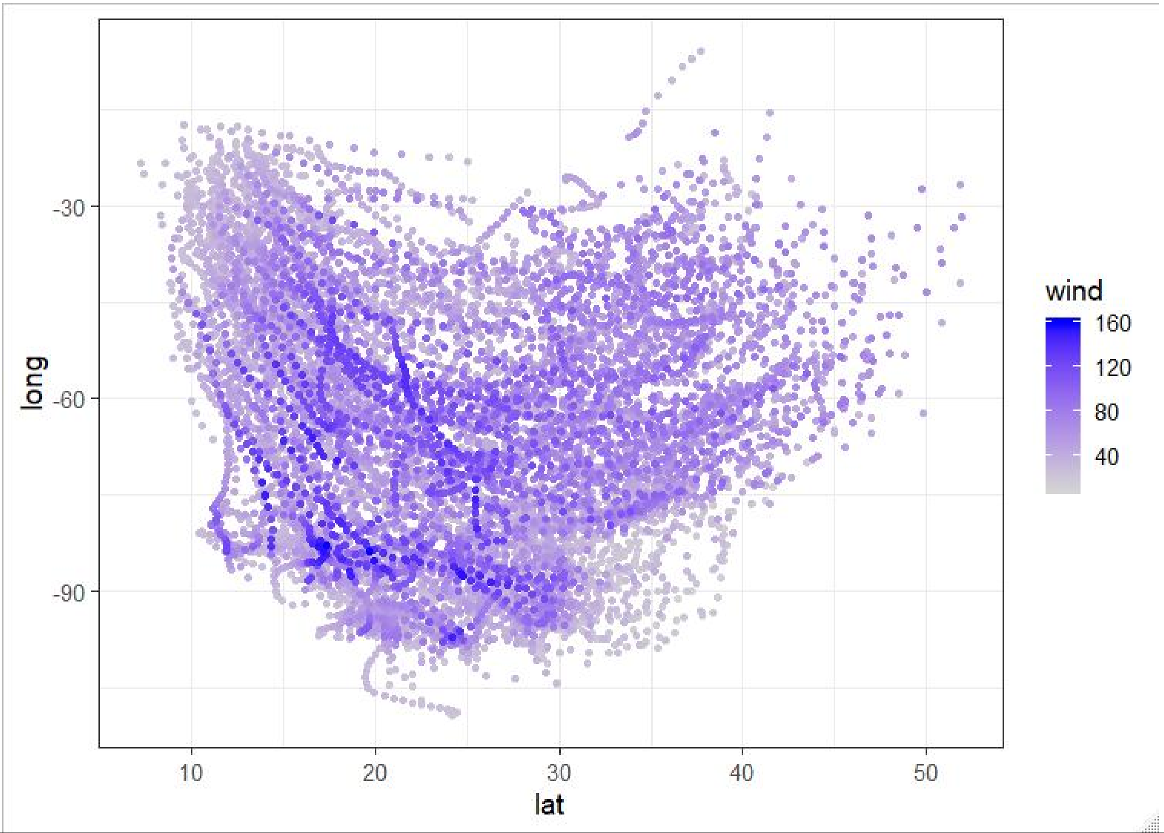



```
storms %>%
 arrange(wind) %>%
 ggplot(aes(x=lat,y=long,colour=wind))+
 geom_point()
```



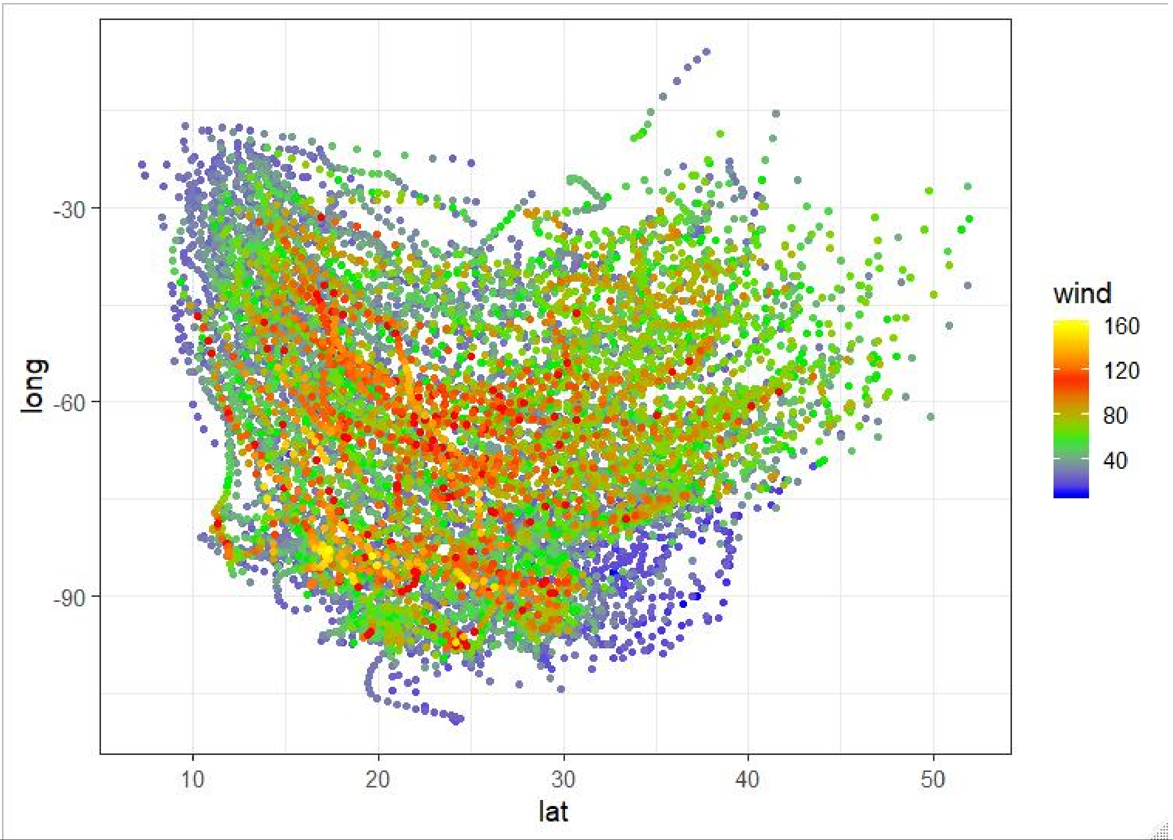
Changing Quantitative Colours

```
storms %>%
  arrange(wind) %>%
  ggplot(aes(x=lat, y=long, colour=wind))+
  geom_point() +
  scale_colour_gradient(low="lightgrey", high="blue")
```

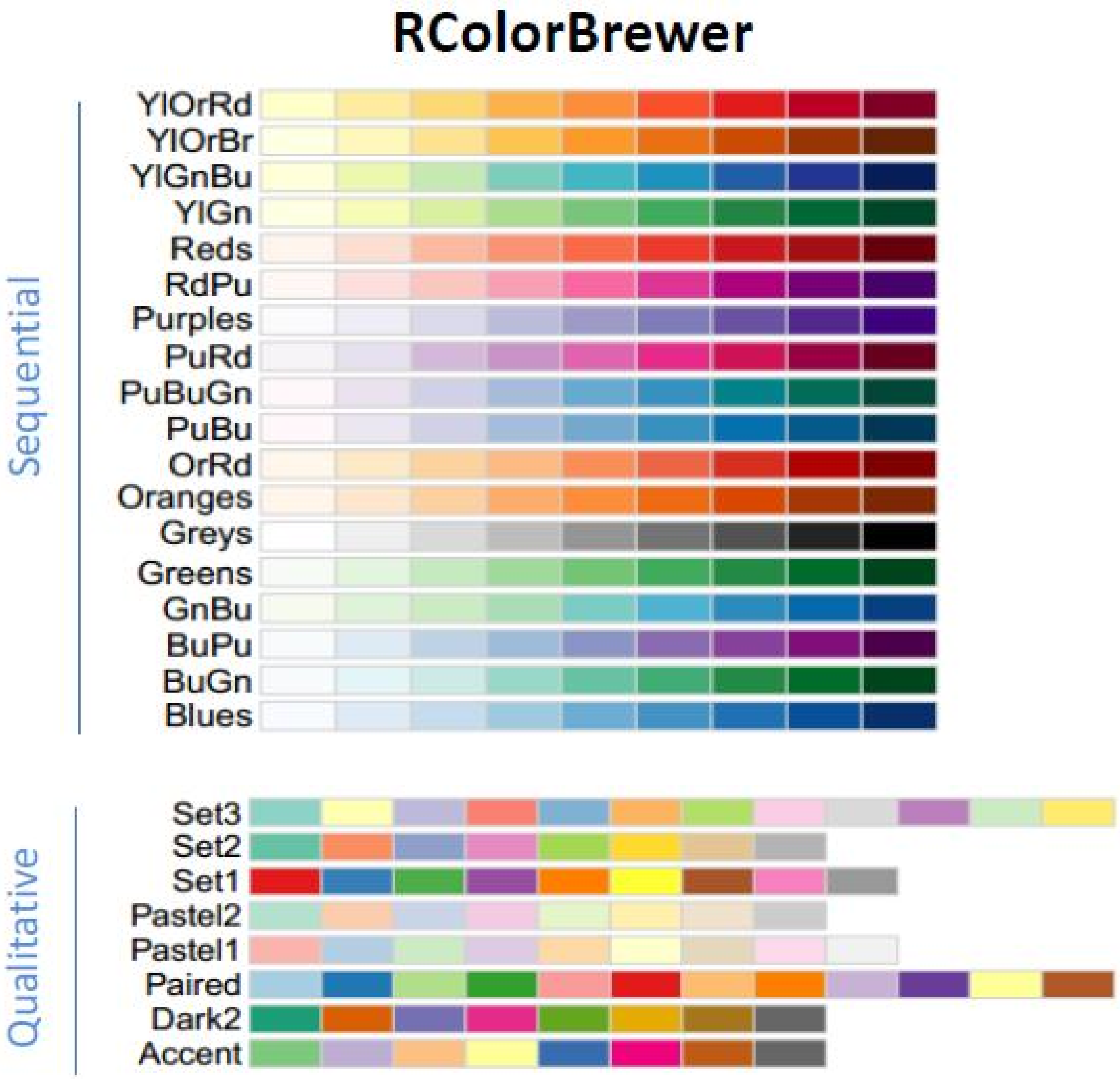


Changing Quantitative Colours

```
storms %>%
  arrange(wind) %>%
  ggplot(aes(x=lat, y=long, colour=wind))+
  geom_point() +
  scale_colour_gradientn(colours=c("blue","green2","red","yellow"))
```



ColorBrewer Scales



Quantitative

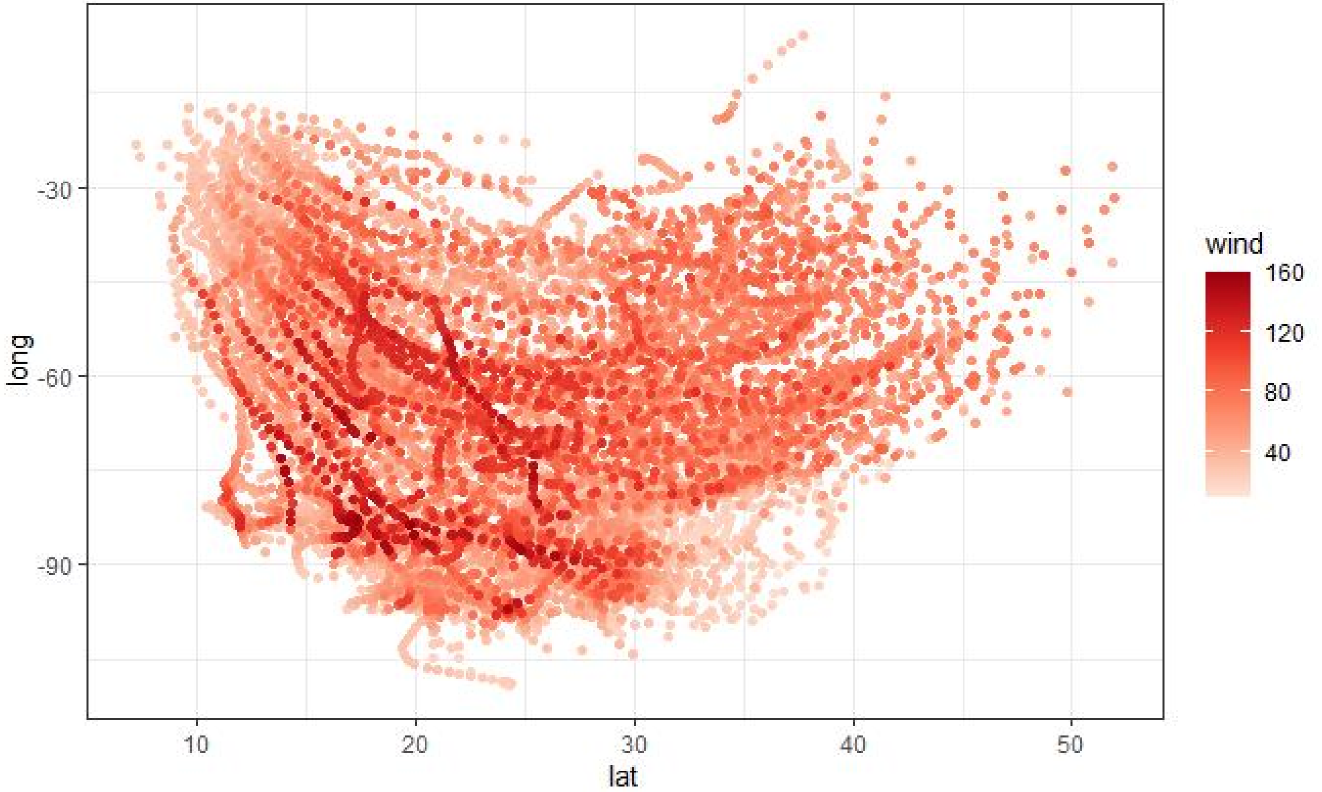
`scale_colour_distiller`

Categorical

`scale_colour_brewer`

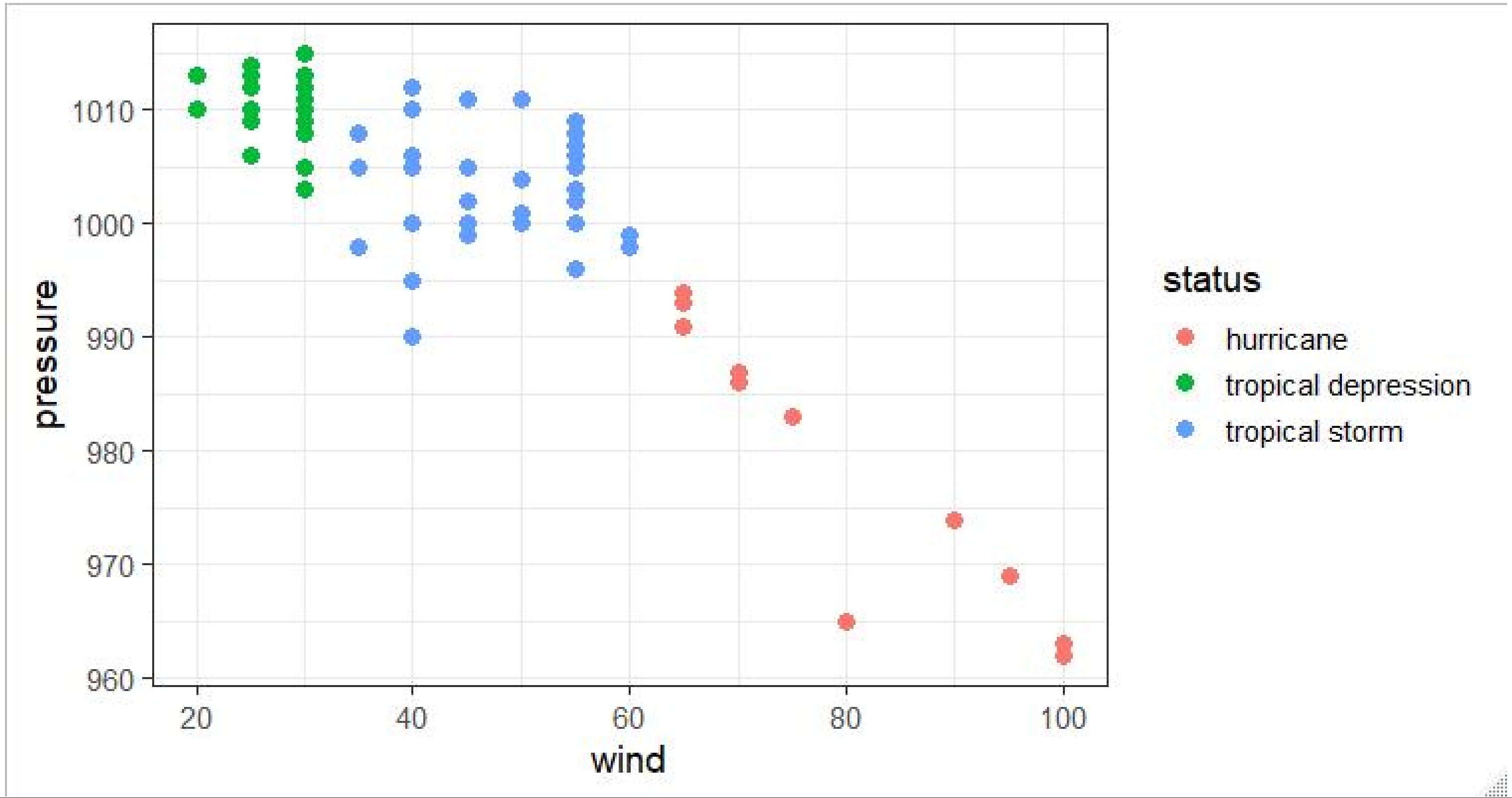
Changing Quantitative Colours

```
storms %>%
  arrange(wind) %>%
  ggplot(aes(x=lat, y=long, color=wind))+
  geom_point() +
  scale_color_distiller(palette="Reds", direction = 1)
```



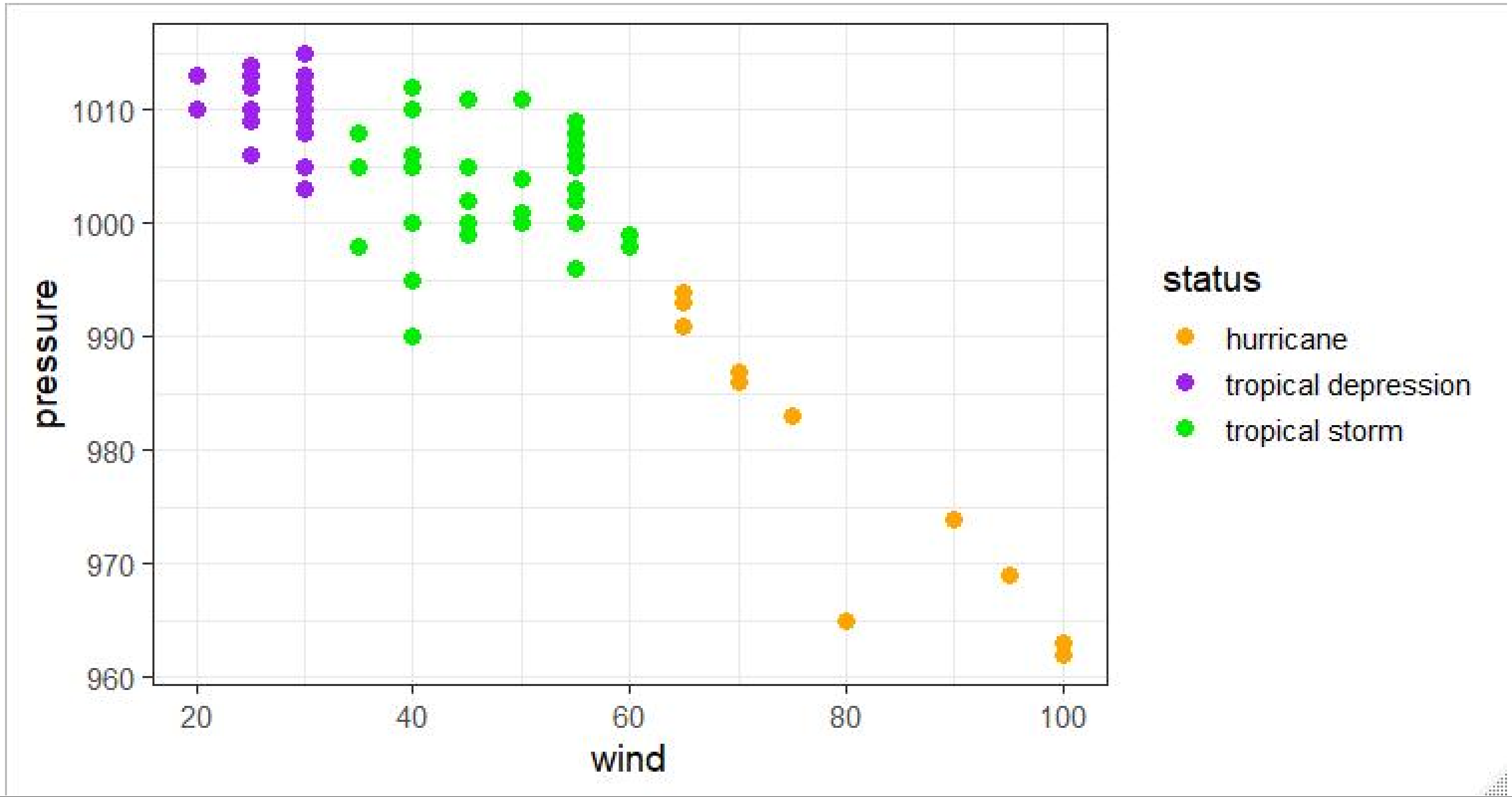
Changing Categorical Colours

```
storms %>%
  filter(year==1983) %>%
  ggplot(aes(x=wind, y=pressure, colour=status)) +
  geom_point(size=3)
```



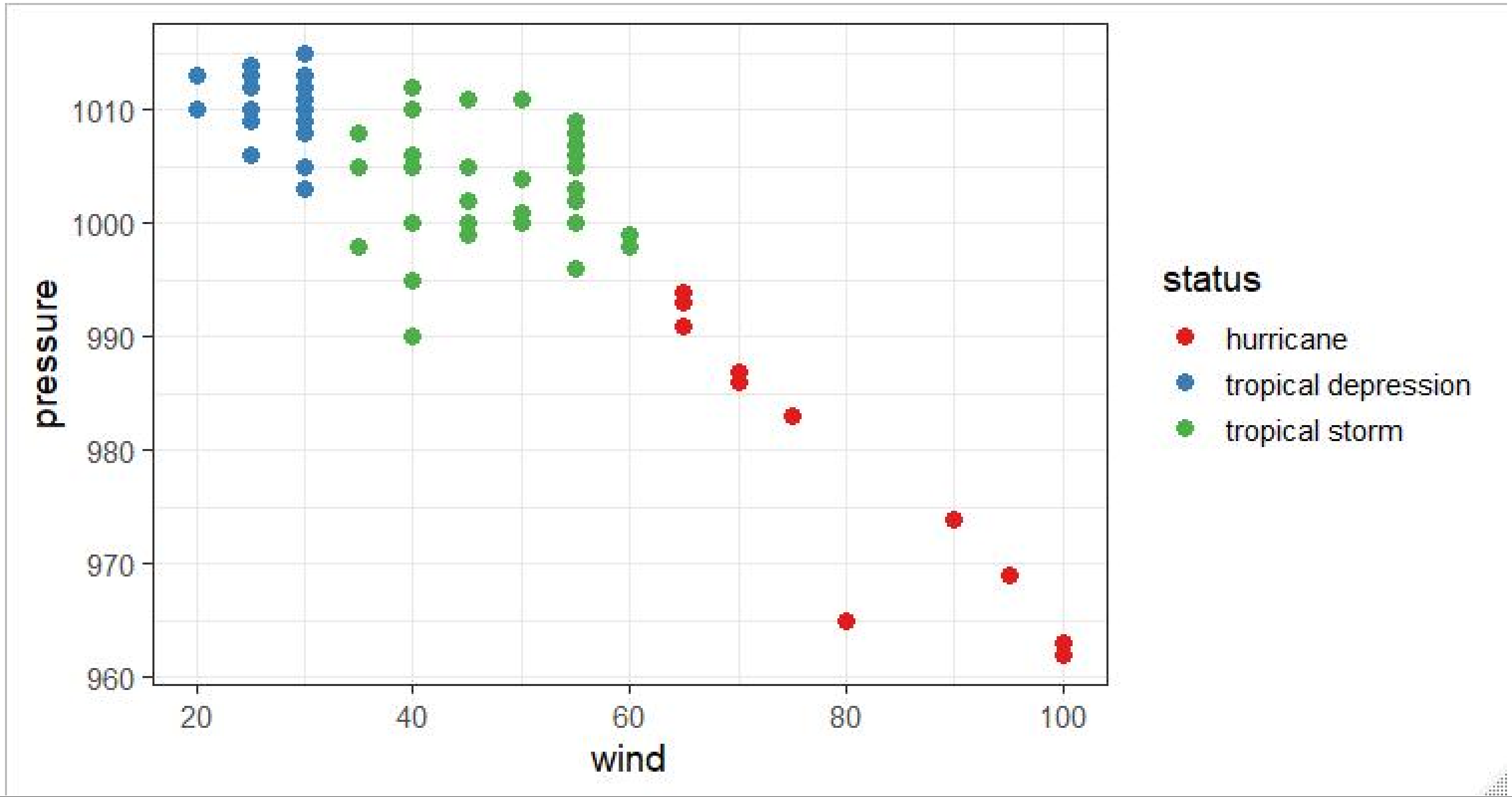
Changing Categorical Colours

```
storms %>%
  filter(year==1983) %>%
  ggplot(aes(x=wind,y=pressure, colour=status)) +
  geom_point(size=3) +
  scale_colour_manual(values =
c("orange","purple","green2"))
```



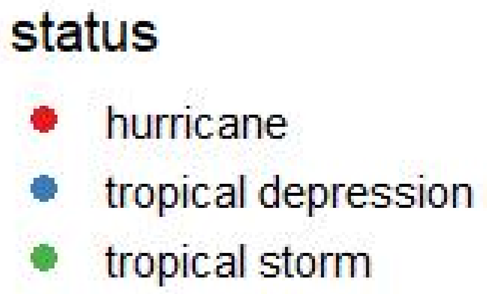
Changing Categorical Colours

```
storms %>%
  filter(year==1983) %>%
  ggplot(aes(x=wind,y=pressure, colour=status)) +
  geom_point(size=3) +
  scale_colour_brewer(palette="Set1")
```



Categorical Colour Ordering

```
# A tibble: 10,010 x 6
     lat  long status              category  wind pressure
   <dbl> <dbl> <chr>               <ord>    <int>    <int>
 1  27.5 -79   tropical depression -1          25     1013
 2  28.5 -79   tropical depression -1          25     1013
 3  29.5 -79   tropical depression -1          25     1013
 4  30.5 -79   tropical depression -1          25     1013
 5  31.5 -78.8 tropical depression -1          25     1012
 6  32.4 -78.7 tropical depression -1          25     1012
 7  33.3 -78   tropical depression -1          25     1011
 8  34   -77   tropical depression -1          30     1006
 9  34.4 -75.8 tropical storm      0           35     1004
10  34   -74.8 tropical storm      0           40     1002
# ... with 10,000 more rows
```



Status is a character vector – ordering is alphabetical

Factors

- Similar to text (character) vectors, but with some differences
  - They have controlled values – you can limit which values can be added
  - The values which can go in are tracked separately to the data
  - The values which can go in have an explicit order

- GGplot respects the ordering of factors, so converting to factors is the simplest way to re-order a plot

Converting character vectors to factors

```
> chr.names
 [1] "simon" "anne"  "laura" "felix" "simon" "anne"  "laura"
 [8] "felix" "simon" "anne"  "laura" "felix" "simon" "anne"
[15] "laura" "felix" "simon" "anne"  "laura" "felix"
```

```
> factor(chr.names)
 [1] simon anne  laura felix simon anne  laura felix simon
[10] anne  laura felix simon anne  laura felix simon anne
[19] laura felix
Levels: anne felix laura simon
```

```
> factor(chr.names, levels=c("simon","anne","laura","felix"))
 [1] simon anne  laura felix simon anne  laura felix simon
[10] anne  laura felix simon anne  laura felix simon anne
[19] laura felix
Levels: simon anne laura felix
```

Categorical Colour Ordering

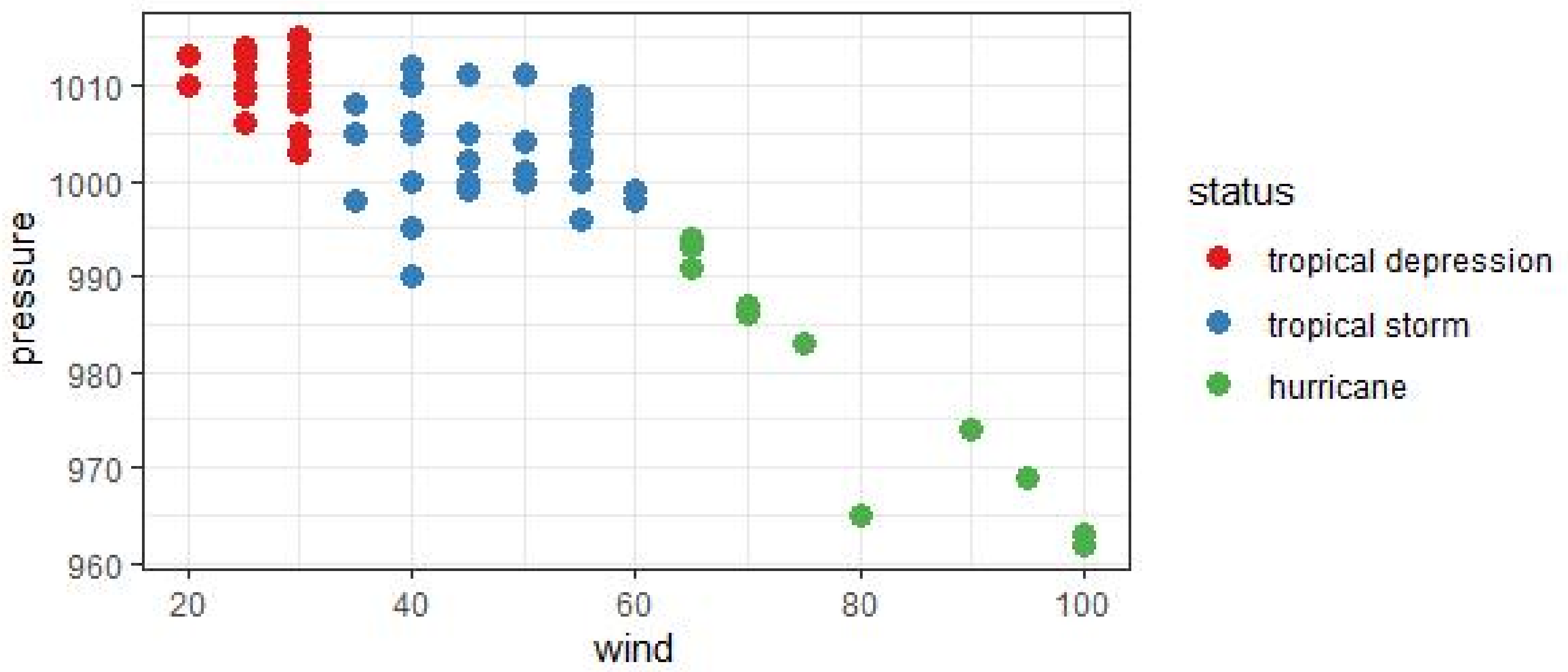
Use factors for explicit ordering

```
storms %>%
  mutate(
    status=factor(
            status,
            levels=c("tropical depression","tropical storm","hurricane")
            )
  )
```

```
# A tibble: 10,010 x 6
     lat  long status              category  wind pressure
   <dbl> <dbl> <fct>               <ord>    <int>    <int>
 1  27.5 -79   tropical depression -1          25     1013
 2  28.5 -79   tropical depression -1          25     1013
 3  29.5 -79   tropical depression -1          25     1013
 4  30.5 -79   tropical depression -1          25     1013
```

Categorical Colour Ordering

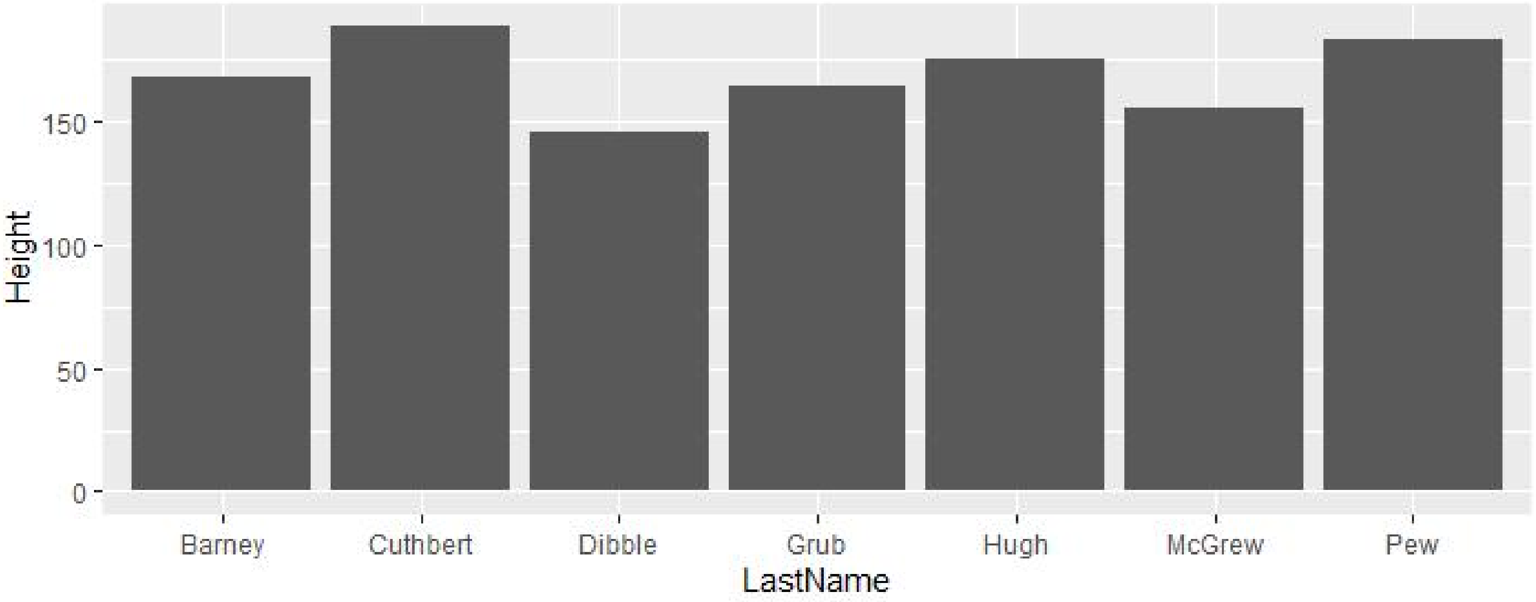
```
storms %>%
  mutate(status=factor(status, levels=c("tropical depression","tropical storm","hurricane"))) %>%
  filter(year==1983) %>%
  ggplot(aes(x=wind,y=pressure, colour=status)) +
  geom_point(size=3)+
  scale_color_brewer(palette="Set1")
```



Reordering example
Keep the original order

```
  LastName FirstName   Age Weight Height
  <chr>    <chr>     <dbl>  <dbl>  <dbl>
1 Hugh     Chris        26     90    175
2 Pew      Adam         32    102    183
3 Barney   Daniel       18     88    168
4 McGrew   Chris        48     97    155
5 Cuthbert Carl         28     91    188
6 Dibble   Liam         35     94    145
7 Grub     Doug         31     89    164
```

```
trumpton %>%
  ggplot(aes(x=LastName, y=Height)) +
  geom_col()
```

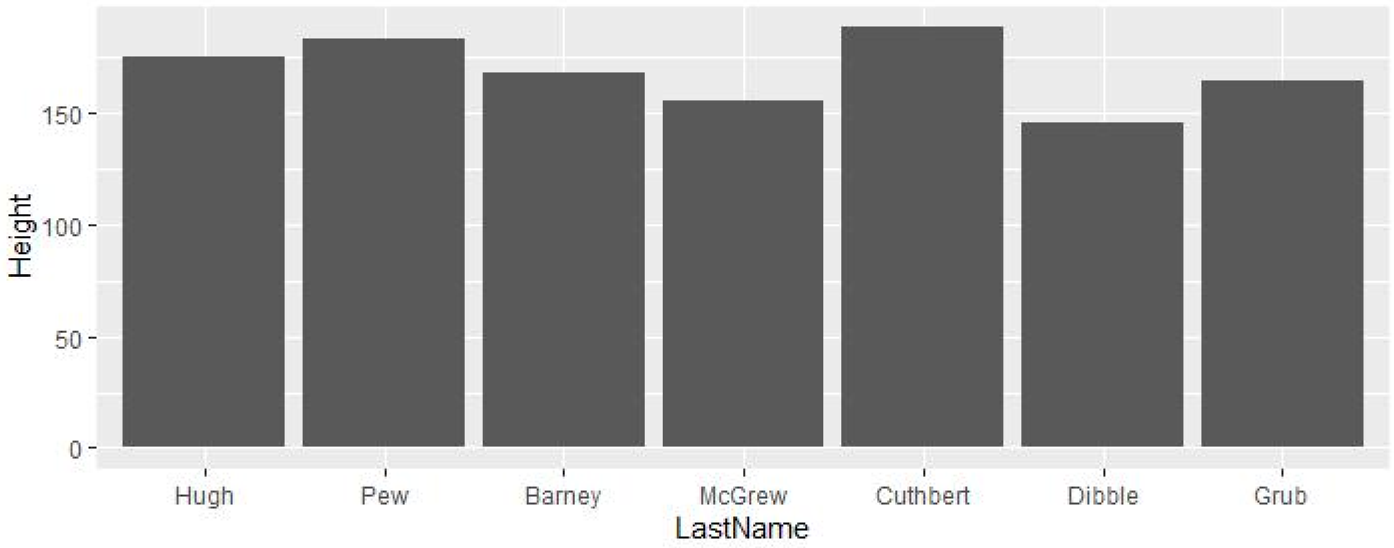


The default is to order alphabetically

Reordering example
Keep the original order

```
  LastName FirstName   Age Weight Height
  <chr>    <chr>     <dbl>  <dbl>  <dbl>
1 Hugh     Chris        26     90    175
2 Pew      Adam         32    102    183
3 Barney   Daniel       18     88    168
4 McGrew   Chris        48     97    155
5 Cuthbert Carl         28     91    188
6 Dibble   Liam         35     94    145
7 Grub     Doug         31     89    164
```

```
trumpton %>%
  mutate(LastName=factor(LastName, levels=LastName)) %>%
  ggplot(aes(x=LastName, y=Height)) +
  geom_col()
```



We can convert to a factor and use `levels` to enforce the same order. If we had just converted to a factor it would have been alphabetical still.

Quantitative ordering with reorder

- The reorder function allows you to order the levels of a factor by a different quantitative variable

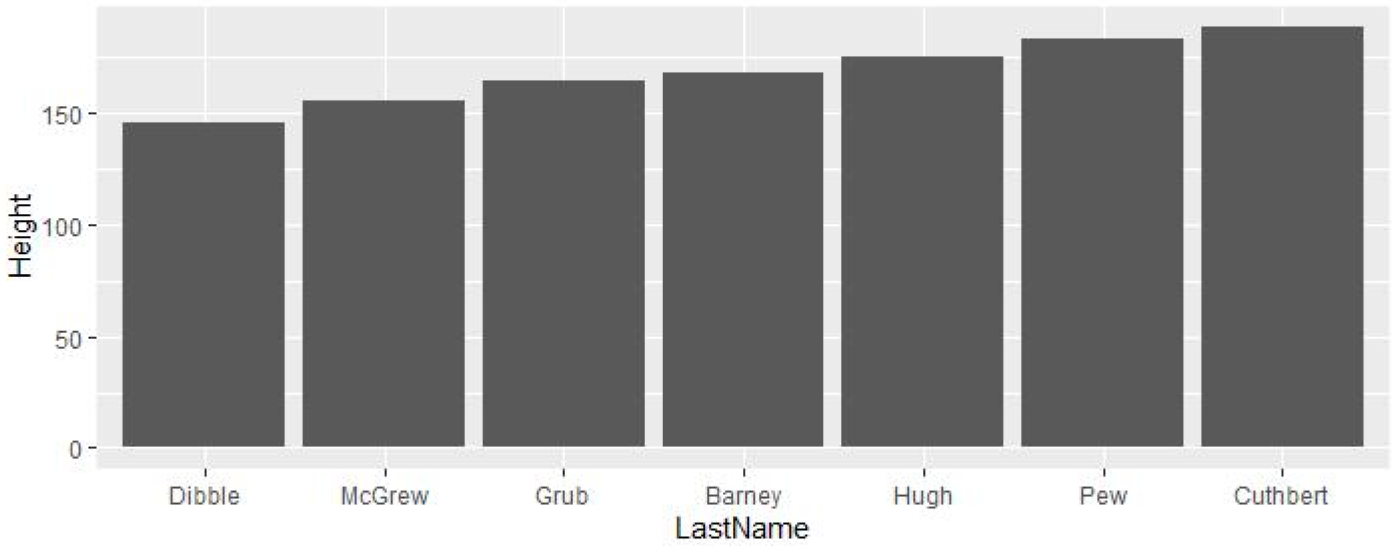
- It allows you to sort a figure by value

- `reorder(categorical, quantitative)`

Reordering examples

```
  LastName FirstName   Age Weight Height
  <chr>    <chr>     <dbl>  <dbl>  <dbl>
1 Hugh     Chris        26     90    175
2 Pew      Adam         32    102    183
3 Barney   Daniel       18     88    168
4 McGrew   Chris        48     97    155
5 Cuthbert Carl         28     91    188
6 Dibble   Liam         35     94    145
7 Grub     Doug         31     89    164
```

```
trumpton %>%
  mutate(LastName=reorder(LastName,Height)) %>%
  ggplot(aes(x=LastName, y=Height)) +
  geom_col()
```

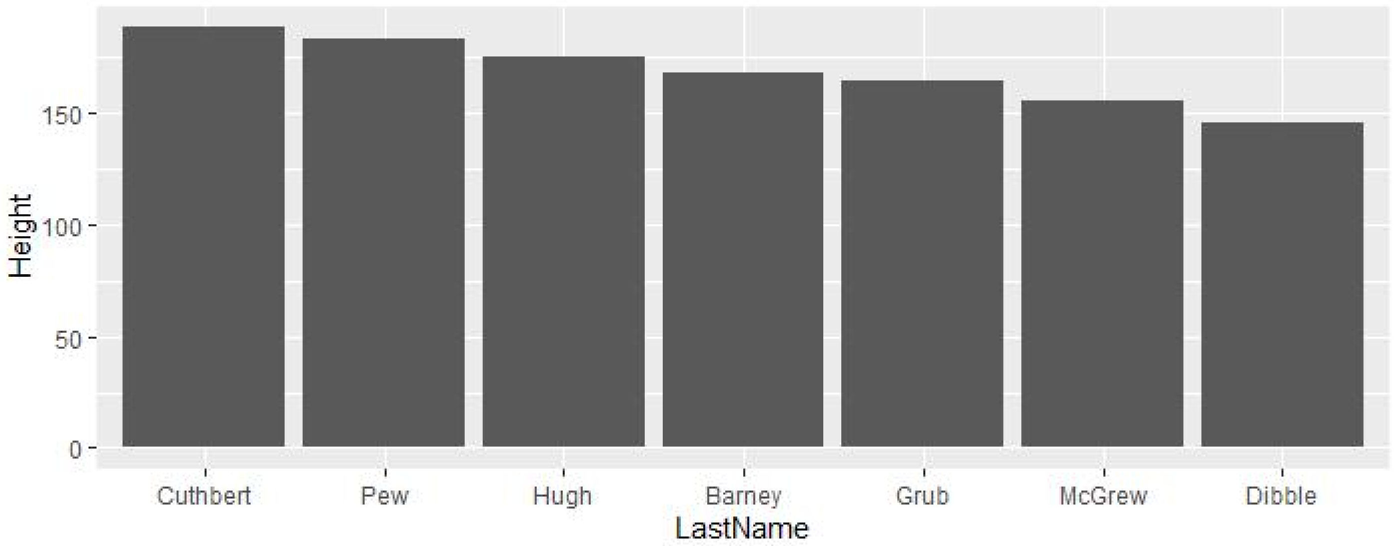


By using `reorder` we can make the levels correspond to a quantitative variable. Here it is the same one we're plotting, but it doesn't have to be.

Reordering examples

```
  LastName FirstName   Age Weight Height
  <chr>    <chr>     <dbl>  <dbl>  <dbl>
1 Hugh     Chris        26     90    175
2 Pew      Adam         32    102    183
3 Barney   Daniel       18     88    168
4 McGrew   Chris        48     97    155
5 Cuthbert Carl         28     91    188
6 Dibble   Liam         35     94    145
7 Grub     Doug         31     89    164
```

```
trumpton %>%
  mutate(LastName=reorder(LastName,-Height)) %>%
  ggplot(aes(x=LastName, y=Height)) +
  geom_col()
```



We can use `-Height` in the reorder to reverse the sorting order

Exercise 1: Simple point and line plots

Load the data from the `weight_chart.txt` file. This is a tab delimited text file. You'll need to use `library(tidyverse)` to load the tidyverse functions, then set the working directory with `Session > Set Working Directory > Choose Directory` in RStudio then use `read_delim()` to load the file and save it to a variable.

This file contains the details of the growth of a baby over the first few months of its life.

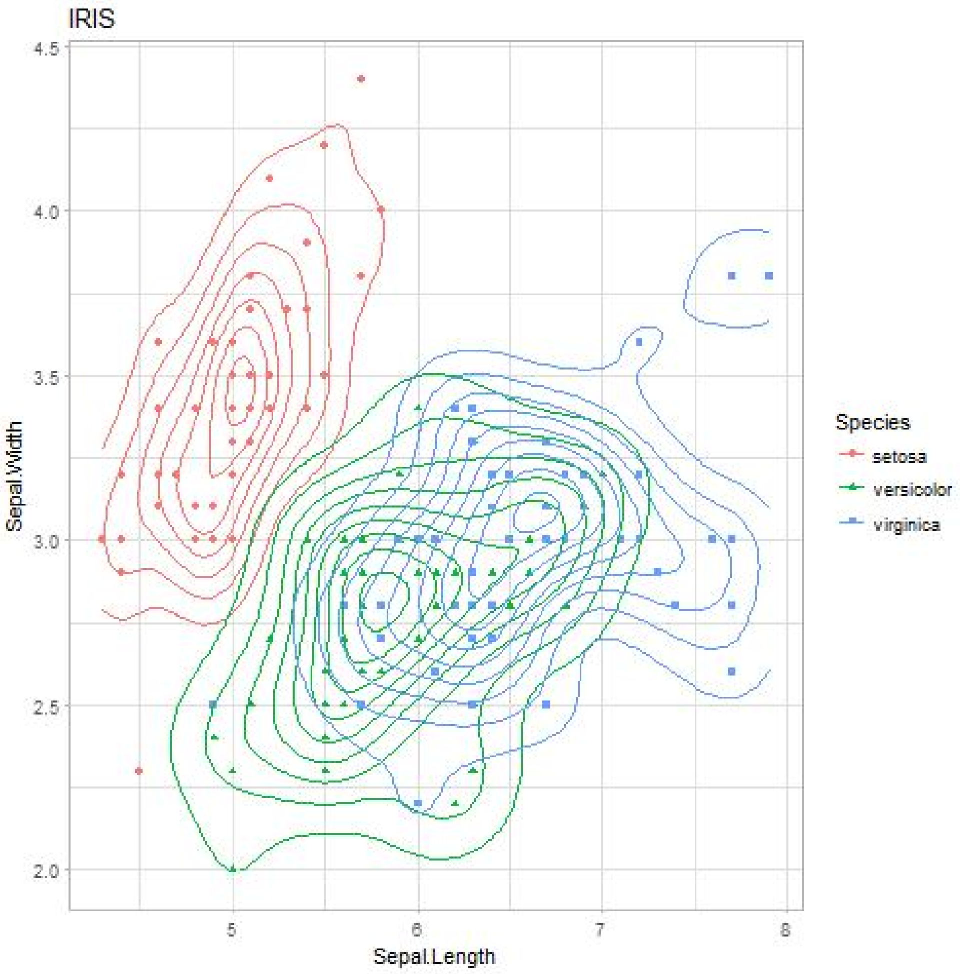
- Draw a scatterplot (using `geom_point`) of the `Age` vs `Weight`. When defining your aesthetics the `Age` will be the `x` and `Weight` will be the `y`.

- Make all of the points filled with `blue2` by putting a fixed aesthetic into `geom_point()` and give them a size of 3

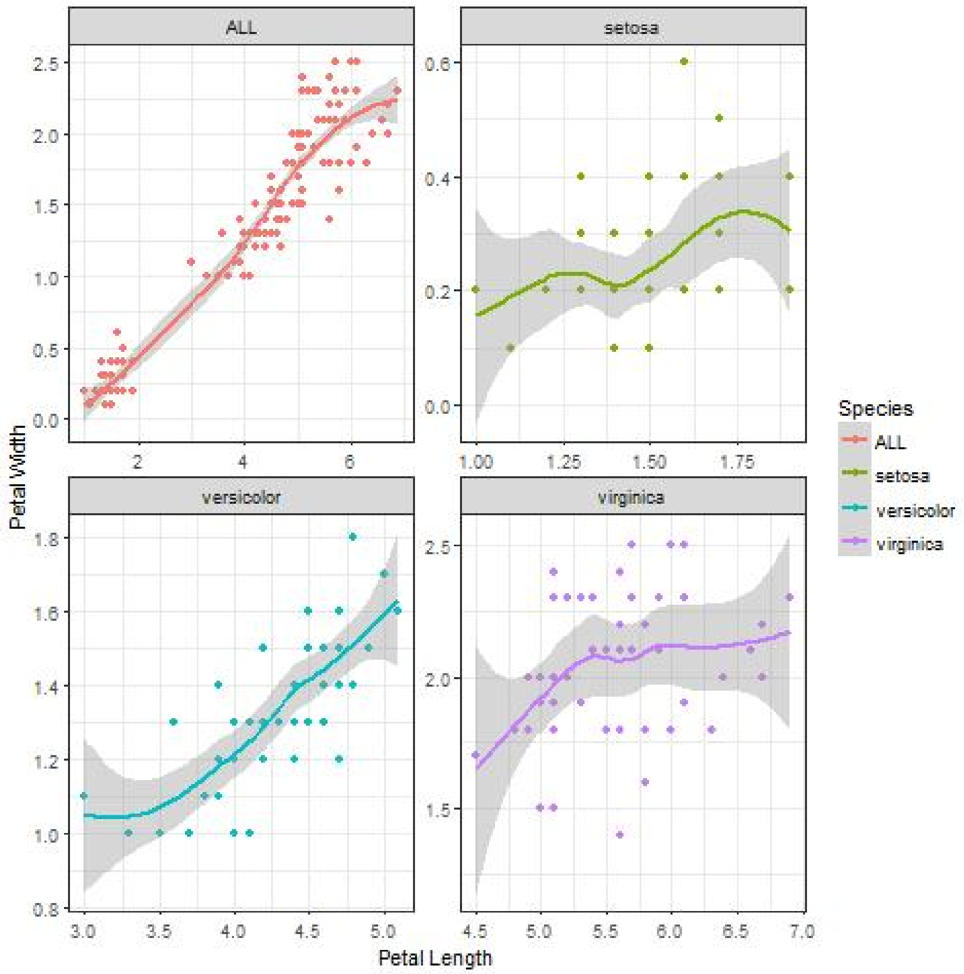
- You will see that an obvious relationship exists between the two variables. Change the geometry to `geom_line` to see another way to represent this plot.

- Combine the two plots by adding both a `geom_line` and a `geom_point` geometry to show both the individual points and the overall trend.

Exercise 2
Fancy the iris dot-plot.

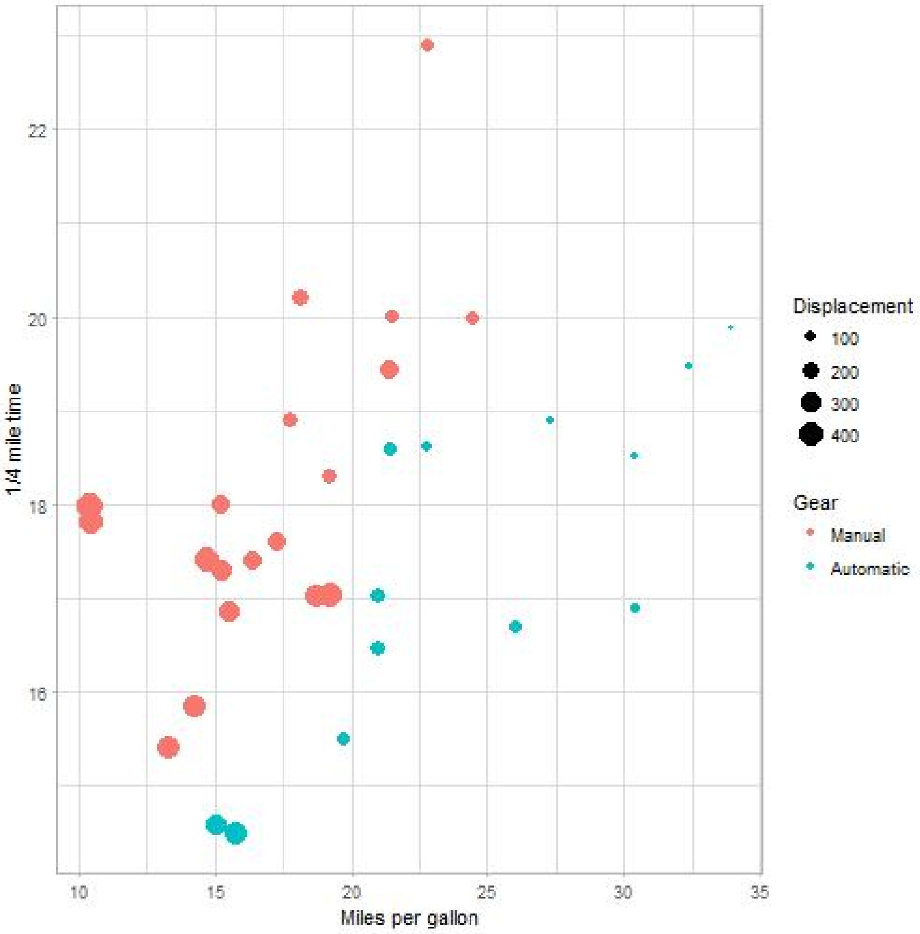


Exercise 3
Faceted smoothing (iris, once again).



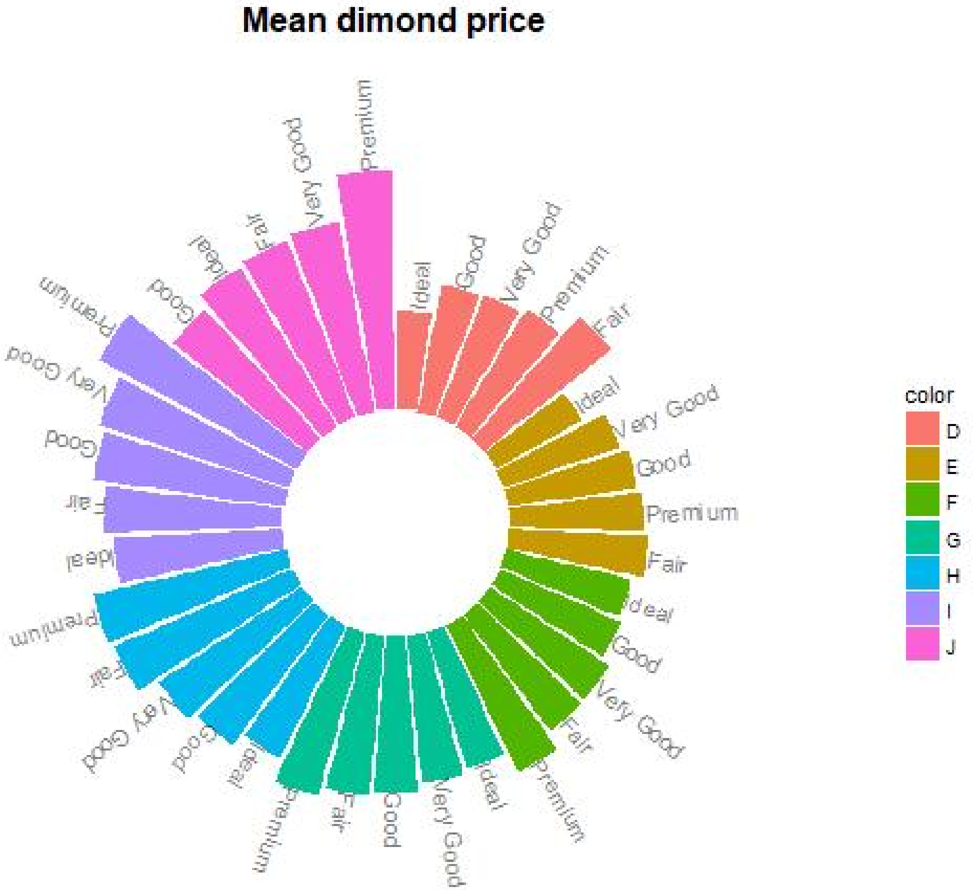
Exercise 4

mtcars bubble-plot.



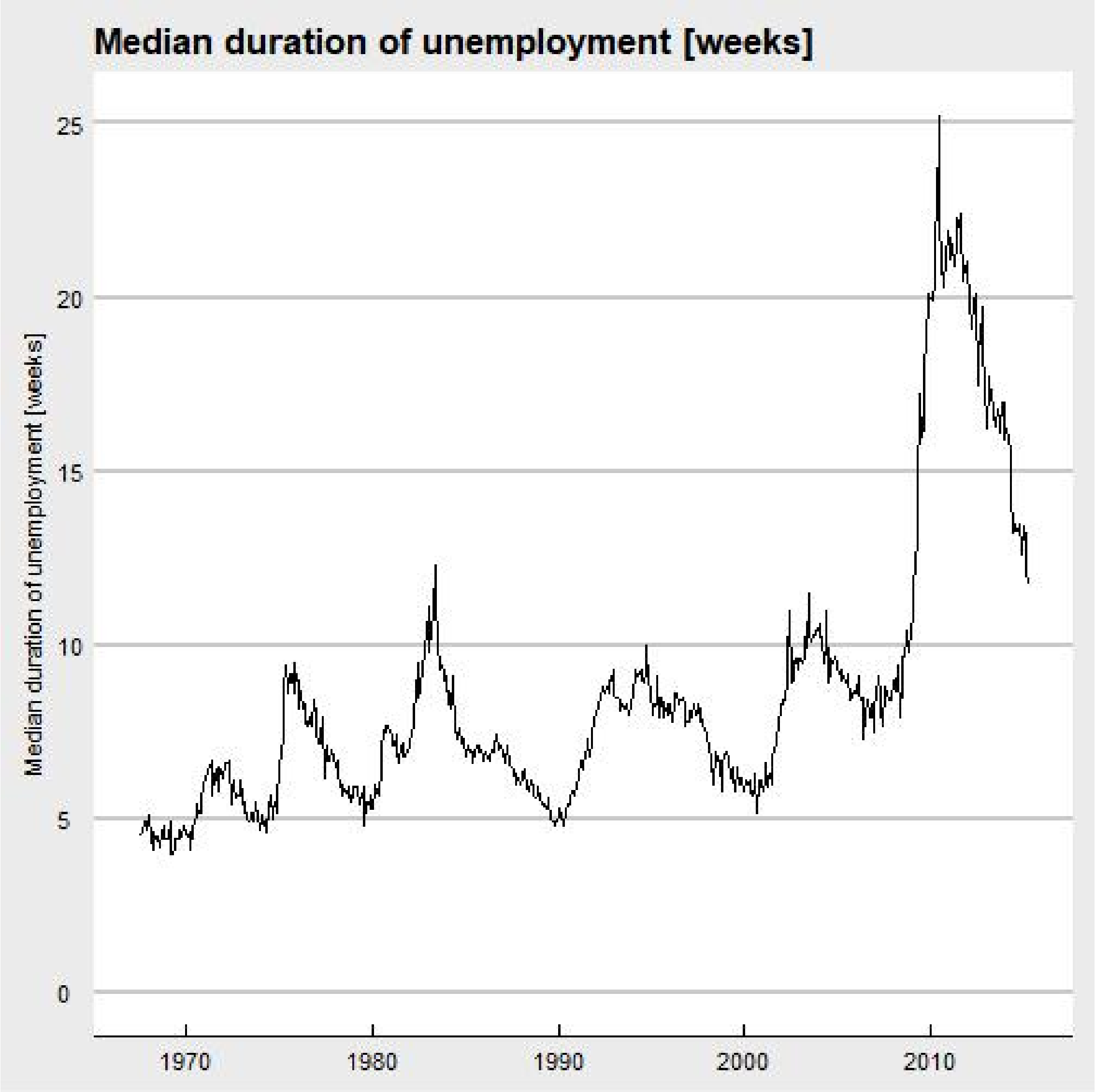
Exercise 5

Polar barplot of the mean diamond price per cut and color.



Exercise 6

Economist style economics time series. (Hint: you will need the ggthemes package.)



Exercise 7
Load the ggplot2, MASS and viridis packages. Combine the three Pima data-sets from (MASS) and make a 2D density (density heat map) plot of bp versus bmi using scale_fill_viridis().
Exercise 8
Using the same data, overlay a histogram of bmi with a normal density curve using the sample mean and standard deviation.
Exercise 9
Using the accdeaths data-set from MASS, make a line plot with time on the x-axis. Mark the maximum and minimum value of accidental deaths in a month with a read and blue dot, respectively. Note that the data does not come in ggplot-friendly format.
Exercise 10
The internet surely loves cats, but most users have little idea how much a cat's organs weigh. Using the cats data from the MASS package, make two 2D density plot of total weight versus hearth weight, side by side; one for each gender. In addition, add a dot for each observation.
Exercise 11
Back to the pima data. Make a boxplot for the glu (glucose concentration), splitting the observations into five age groups with approximately the same number of observations.
Exercise 12
Using ggplot2's inbuilt economics data-set, make a stacked bar plot with proportions of unemployed to employed (employed or not seeking work) with the date in the x-axis.
Exercise 13
Using ggplot2's inbuilt msleep data-set, make a scatter plot (body weight versus total sleep) of all animals of the order artiodactyla. Mark the domesticated animals with a different color (from black) and annotate their names onto the graph.
Exercise 14
Using msleep, make one density plot for the total sleep, colored by vore. Play with the transparency and parameters of the density estimation.