# R and Rstudio: Part III

# Data Analysis and Visualization

Dr. Irene Vrbik

University of British Columbia Okanagan

irene.vrbik@ubc.ca

# Introduction

- In Python and Excel, we have explored several techniques for fitting important statistical models such as:
  - kmeans clustering
  - linear regression
- In addition we saw the ease of creating charts in Excel and the customizable graphics we could create in Python using `matplot.lib`

# Data Analysis in R

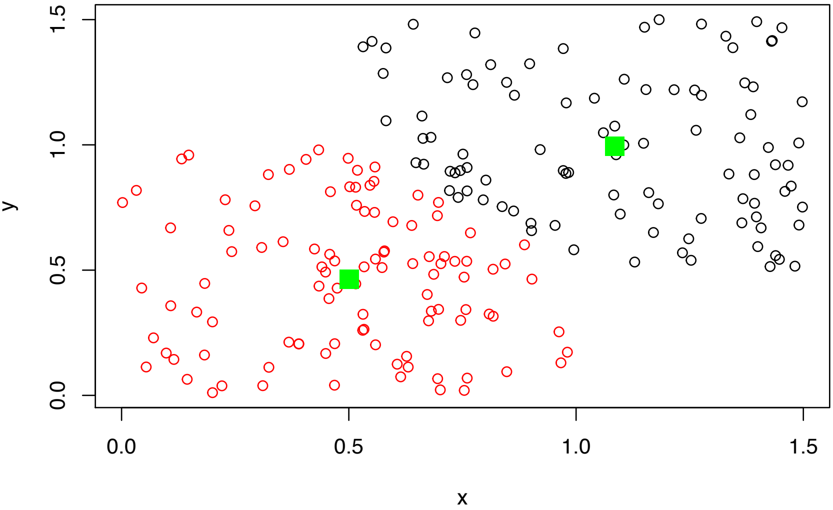
- R is an optimized environment for data analysis.
- We could perform each of these tasks quite readily in base R, ie. without having to import/load any libraries.
- To give you a sneak peak of the `R`'s capabilities, let's reproduce these examples from previous lectures here.
- Don't worry if you don't understand every line of code, we will work our way up to understanding these programs.

# Data Analysis in R

For example, see how we could recreate the *k*-means example from the final Python lecture:

```r
data1 = data.frame(x=runif(100),y=runif(100))
data2 = data.frame(x=runif(100)+0.5,y=runif(100)+0.5)
data = rbind(data1, data2)
km = kmeans(data, centers=2)
plot(data, col=km$cluster)
points(km$centers, col="green", pch= 15, cex = 2)
```

# *k*-means



# Linear Regression

- We can use the `lm` function for fitting a linear regression model:

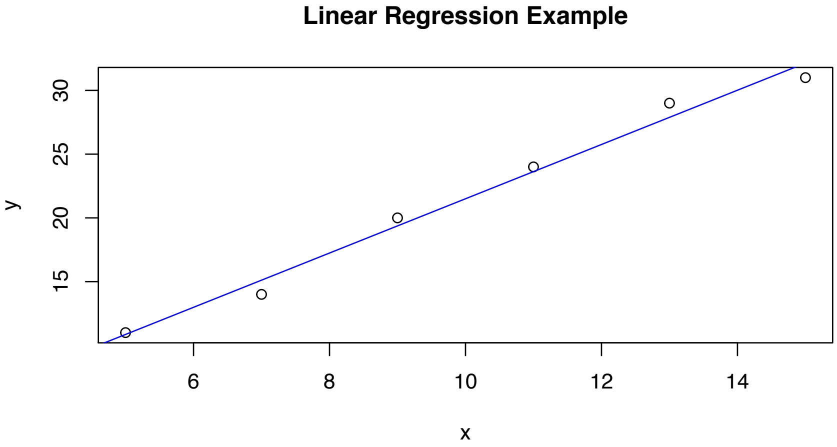
```
x = c(5, 7, 9, 11, 13, 15); y = c(11, 14, 20, 24, 29, 31)
# stores the estimates of slope and y-intercept
fit = lm(y~x); class(fit) # of class "lm"

## [1] "lm"

predict(fit) # calculates the predicted y's (for the x's used in i

##        1        2        3        4        5        6
## 10.85714 15.11429 19.37143 23.62857 27.88571 32.14286

y.int = fit$coefficients[1]
slope = fit$coefficients[2]
```

A number of useful summaries are supplied by taking the `summary()` on an `"lm"` type object:

```
sfit <- summary(fit)
sfit$r.squared # R-squared value

## [1] 0.9864919

sfit$residuals # residuals

##          1          2          3          4          5
##  0.1428571 -1.1142857  0.6285714  0.3714286  1.1142857 -1.142857

sfit$coefficients # p-values/t-tests for parameter estimates

##              Estimate Std. Error    t value     Pr(>|t|)
## (Intercept) 0.2142857   1.316044  0.1628256 8.785506e-01
## x           2.1285714   0.124540 17.0914722 6.873635e-05
```

Plotting is as simple as:

```r
plot(x, y, main="Linear Regression Example")
abline(fit, col="blue")
```



# Background

- One of the main reasons data analysts turn to R is for its strong graphic capabilities.
- R's model for constructing plots strikes a balance between structure and flexibilty.
- Base `R` has a number of functions for basic plots which include:
  - Scatter plots
  - Histograms
  - Boxplots

# Scatterplots

- Plots of this type are produced using `plot(x, y, ...)` in R.
- The first argument supplies the *x* co-ordinate values, while the second number provides the *y* co-ordinate values.
- The `...` denotes optional graphical parameters[1]

  `main` A character string used in the title

  `xlab/ylab` A character string used for the x/y axis labels

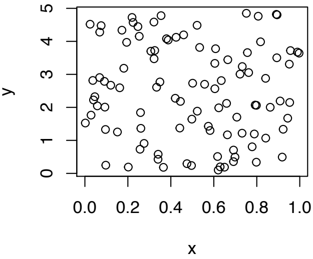
  `xlim/ylim` A vector = `c(xmin, xmax)/c(ymin, ymax)` used for the plotting ranges

  `cex` number indicating plotting text/symbols size. (1=default, 1.5 is 50% larger, 0.5 is 50% smaller, etc.)

[1] see a nice summary of some useful ones here and a cheat sheet

# Scatterplots

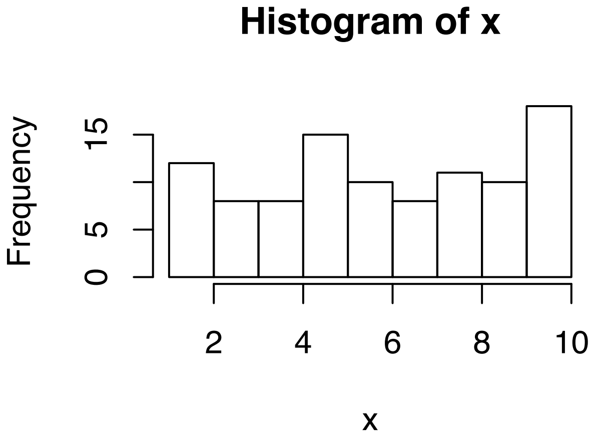
```
x = runif(n=100) # 100 random numbers between 0 and 1
y = runif(n=100, min=0, max=5) # 100 random numbers between 0 and 5
plot(x,y)
```



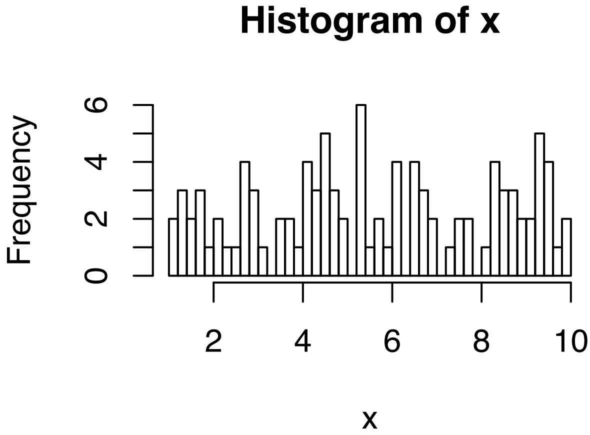
# Histograms

- A histogram is very common visualization that plots the number of observations appearing within certain ranges called “bins".
- Bins (or “buckets") are constructed by dividing the entire range of values into a series of intervals.
- In R histograms are produced using the `hist()` function
- `hist()` tries to calculate reasonable bins automatically; however, we can manually set them ourselves in the `breaks` argument

```r
x = runif(n=100, min=1, max =10)
hist(x)
```



```r
x = runif(n=100, min=1, max =10)
hist(x, breaks = 40)
```



# Boxplots

- A boxplot (AKA box-and-whisker plot) provides a graphical view of the median, quartiles, maximum, and minimum of a data set (i.e. the five number summary).
- When applicable, it will identify outliers and their values.
    - Outliers are defined to be $\geq Q3 + 1.5 * \text{IQR}$ or $\leq Q1 - 1.5 * \text{IQR}$
    - Not all data will have outliers.
- Boxplots provide a useful snapshot of your data and can indicate if data is symmetric or skewed, for example.
- Beware that they can be misleading when there are very few data points

# Boxplot

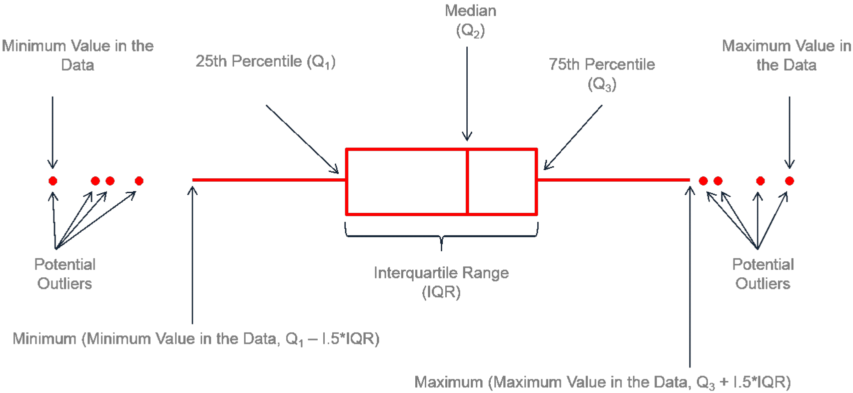
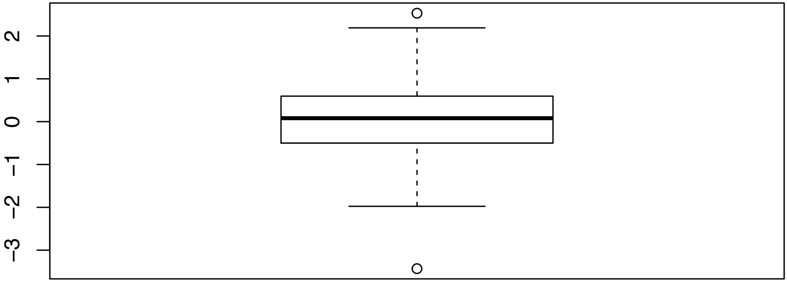

Median
($Q_2$)
Minimum Value in the Data
25th Percentile ($Q_1$)
75th Percentile ($Q_3$)
Maximum Value in the Data
Potential Outliers
Interquartile Range (IQR)
Potential Outliers
Minimum (Minimum Value in the Data, $Q_1$ – I.5*IQR)
Maximum (Maximum Value in the Data, $Q_3$ + I.5*IQR)

Image source

# Boxplots

These plots are available through the `boxplot()` command.

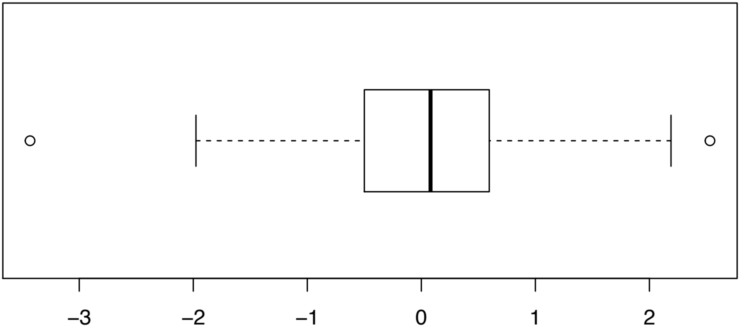
```r
x <- rnorm(100) # generate 100 obs from a standard normal dist
boxplot(x)
```



# Boxplots

We can plot data horizontally by setting `horizontal=TRUE`

```
boxplot(x, horizontal=TRUE)
```



# Boxplots

- The “box" portion represents the IQR = Q3 - Q1.
- The line near the middle represents the median (i.e. Q2).
- The points on the edge of the plot are potential outliers
  - Recall outliers are $\geq Q3 + 1.5*$IQR or $\leq Q1 - 1.5*$IQR
  - Not all data will have outliers.
- The ends of the “whiskers" represent the maximum and minimum observation that are not considered outliers.

# Adding lines to a plot

- To varify that these markings correspond with the five number summary, we can use `abline`.

- `abline` adds a line to the current plot by either specifying:[2] a slope (`a`) and *y*-intercept (`b`); a single *y*-value (`h`) for drawing a horizontal line; a single *x*-value (`v`) for drawing a vertical line

- Related functions:

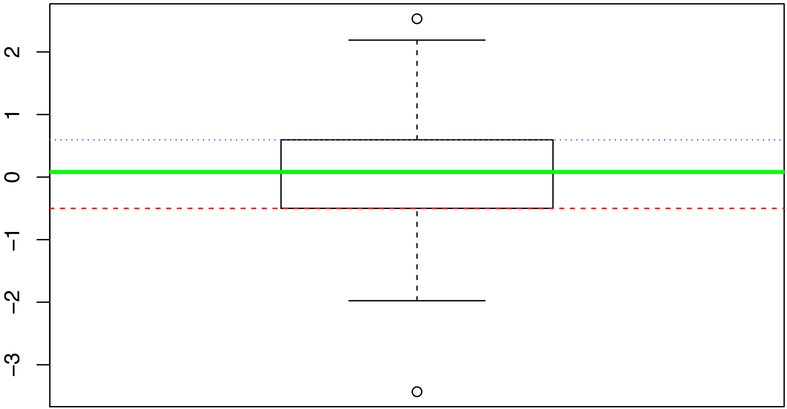
  `points()` for adding points to the current graph

  `text()` for adding text to the current graph

[2] it can also take an `lm` object as an argument

# Adding Lines to a Plot

```r
x5 = fivenum(x); names(x5) = c("min","q1","median","q3","max")
boxplot(x)
# Draw a red dashed line at Q1
abline(h=x5["q1"], col=2, lty=2)
# Draw a green thick line at Q2 (the median)
abline(h=x5["median"], col="green", lwd = 3)
# Draw a purple dotted line at Q3
abline(h=x5["q3"], col="purple", lty = 3)
```


2
1
0
−1
−2
−3

# R colours

We can used colour indeces instead of colour names, eg. 2 = red.

```
palette() # See the entire colour palette

## [1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta"
## [8] "gray"

palette()[2] # colour index 2 = red

## [1] "red"
```

See here for the list of available colour names in `R`

# Boxplot deconstruction

- To determine where our whiskers end, we first need to find our cut-off for potential outliers:

```
IQR = x5["q3"] - x5["q1"]
upper = x5["q3"] + 1.5*IQR
lower = x5["q1"] - 1.5*IQR
(outliers = c(which(x>upper | x<lower)))

## [1] 33 46

x[outliers]

## [1]  2.530865 -3.431558
```

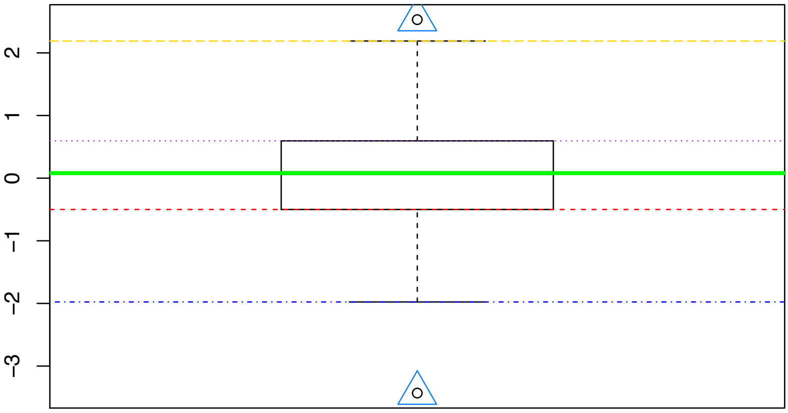
# Boxplot deconstruction

- The whiskers end at the min and max values in the data with the outliers removed.

```
minx = min(x[-outliers])
maxx = max(x[-outliers])
```

- Adding these values to the plot from before we get:

```
abline(h=minx, col="blue")
abline(h=maxx, col="gold")
# plot the outlines in big blue triangles (pch = 2)
points(x[outliers][1], col="dodgerblue2", cex=3, pch =2)
points(x[outliers][2], col="dodgerblue2", cex=3, pch =2)
```



Note that we can only add (not take away) markings in base plotting

# Visualizing Data in R

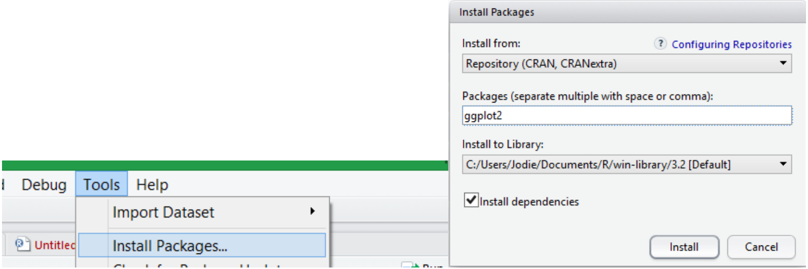
- R supports several graphing libraries (i.e. packages) for producing graphics.
- One of the most popular packages for plotting is `ggplot2` written by Hadley Wickham.
- `ggplot2` implements the Grammar of Graphics and enables us to concisely describe the components of a graphic.
- There is a lot to unpack with this graphic method and it may be helpful to keep a cheatsheet nearby.
- Another great resource is the ggplot2 website and reference.

# What is an R package?

- Recall that an R package is a sharable collection of code/data/functions/documentation.
- While anyone can write a package and share, the Comprehensive R Archive Network, or CRAN is the main repository for vetted R packages that meet a specific criteria.
- There is huge variety of packages available on CRAN ($>$10000) and throughout this course we will be making use of a very small subset of them.

# Installing `ggplot2`

To install `Tools` → `Install Packages...` Then input `ggplot2`.



Install from: Repository (CRAN, CRANextra)

Packages (separate multiple with space or comma): ggplot2

Install to Library: C:/Users/Jodie/Documents/R/win-library/3.2 [Default]

☑ Install dependencies

Alternatively (and this would be my preferred method) type the command:

```r
install.packages("ggplot2")
```

# Installing `ggplot2`

To install this package from CRAN, load the package into R, and access the help files, type the following:

```r
# install.packages("ggplot2") # install (only do once)
library("ggplot2")            # load
package?ggplot2               # help
help(package = "ggplot2")     # another way to get help
```

# A note on installing packages

- Note that this install need only be done once on your personal computer (if you are running this on the lab computers you may need to reinstall every time you log on).

- The `library("ggplot2")` command will need to be executed at each new R session, i.e. every time you close and reopen R Studio

# Grammar of Graphics

Some notation

aes aesthetic attributes, i.e. how data are mapped (eg. colour, shape, size)

geoms geometric objects (eg. points, lines, bars). Also referred to as *layers*. See here for all available geoms.

facets for forming multi-panel plots

stats for statistical transformation (eg. smoothing)

co-ordinate system (eg. $x$ and $y$ axis)

# The basics of ggplot2

- ggplot() is the workhorse function in ggplot2.

- ggplot() works similar to the **base** plotting system, in that we can overlay layers and build-up our plot.

- Rather than specifying graphical features of our plot with arguments in a function, we will add them (literally by using +) to a ggplot object.

# General workflow:

- Identify your data and basic aestheics (identify *x* and *y* variables for example)
  - I'm using the `mtcars` dataset (see `?mtcars`)
  - To produce a simple 2D scatterplot I need to identify which variable will be plotted on the *x*-axis and *y*-axis (note that there are 11 variables in this dataset)
- Save this to an R *object* (which will be `ggplot` *class*).
- Standard convention is to call this ggplot object `g`.

# General workflow:

```
library(ggplot2)

## Warning:  As of rlang 0.4.0, dplyr must be at least version
0.8.0.
## x dplyr 0.7.7 is too old for rlang 0.4.1.
## i Please update dplyr with ‘install.packages("dplyr")‘.

g = ggplot(mtcars, aes(x=mpg, y=disp)) # identify data and x/y var
class(g)

## [1] "gg"     "ggplot"
```

Note that there is nothing plotted when we create this object

# General workflow:

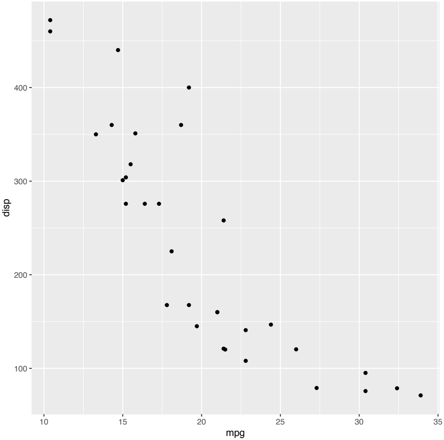
Create desired layers, some examples:

- `geom_point()` creates geometric points
- `geom_smooth()` creates a smoother
  - `geom_smooth(method="lm")` creates a regression line.
- `facet_grid()` for multi-panel plots
- `theme_bw()` changes gray background to black and white theme.
- . . . , many more (see cheatsheet)

# Scatterplot

Like `plot(mpg, disp, data=mtcars)` in **base**

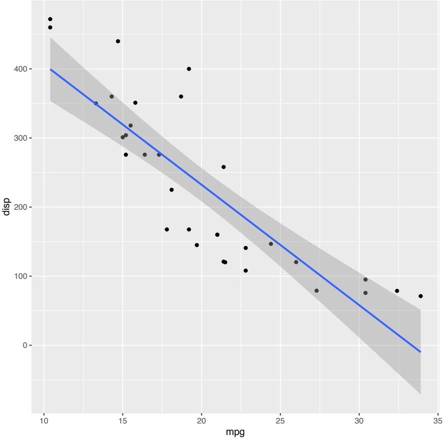
```
g + geom_point()
```



# Adding a line

Add a line overtop our scatterplot (similar to `abline()` in **base**).

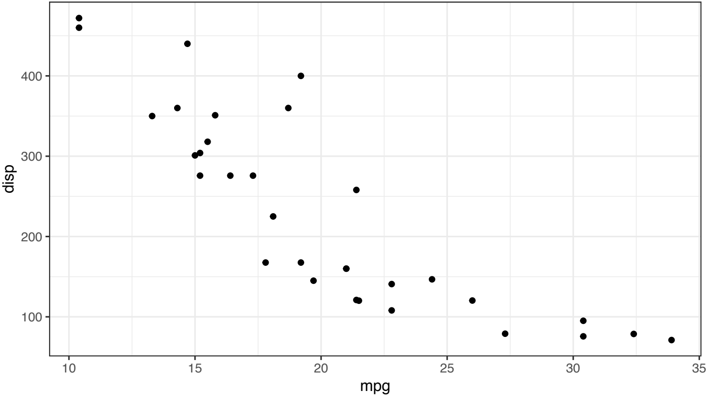
```
g + geom_point() + geom_smooth(method ="lm")
```



# Change Theme

Change the theme from gray to black and white.

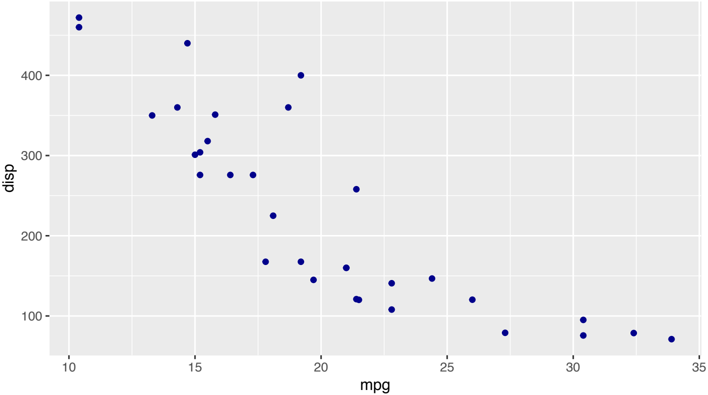
```
g + geom_point() + theme_bw()
```



# Adjusting Graphical Parameters

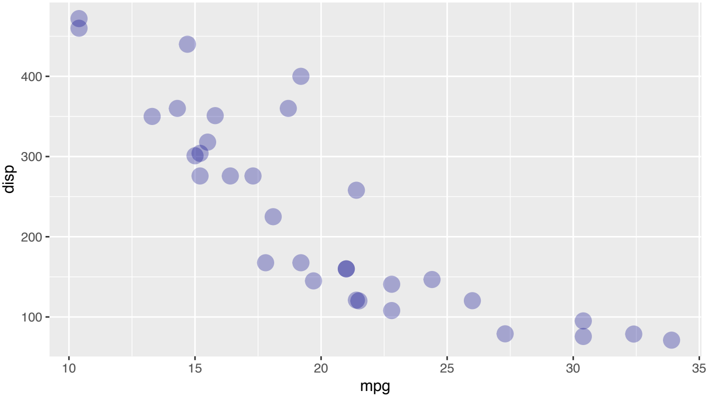
- Annotate with meta-data:
  - Change labels using `xlab()`, `ylab()`, `ggtitle()`, or `labs()` (which is a more general can specify x,y,title)
    - `labs(x=<>, y=<>, title<>)`
- Manage **geom** obects. For example change the default settings in geom_point
  - `geom_point(color=<>, size=<>, alpha =<>)`
  - where `alpha` controls the transparency (0 for completely transparent, 1 for completely opaque)
  - see more options here

```
ggplot(mtcars, aes(x=mpg, y=disp)) + geom_point(color='darkblue')
```



Note here that I am calling `ggplot` directly and not saving a ggplot object

```
g + geom_point(color='darkblue', size = 5, alpha = 0.3)
```



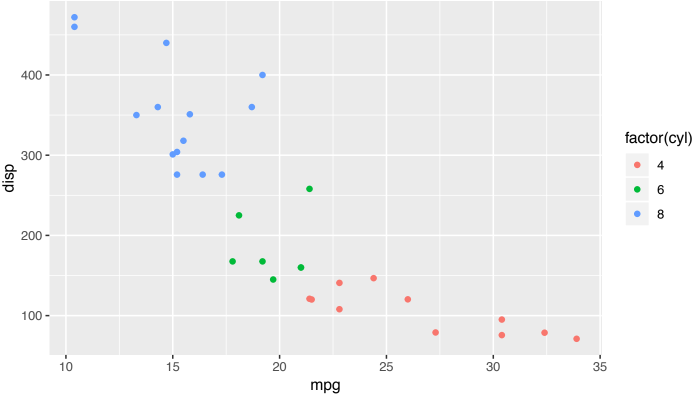
Notice how overlapping points are more obvious when using semi-transparent points.

- We can also colour points according to some category (a factor type variable).

- Notice how we do not need to specify colours and legends in `ggplot()` (although we could change the default settings if we wanted to see here)

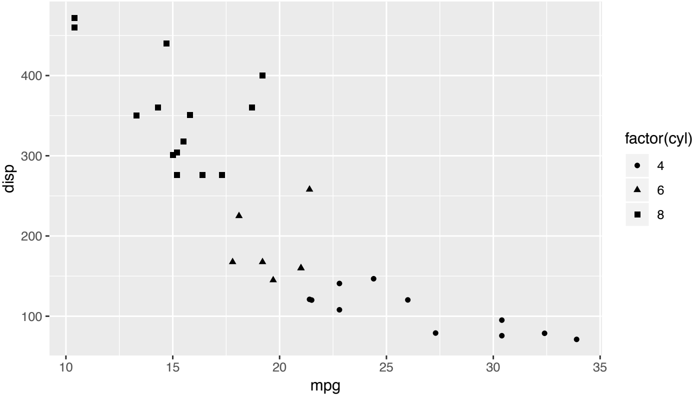
- Notice how we specify this as an aestheic mapping (ie it is wrapped in `aes`) since it describes *how* the variable `cyl` is mapped to the visual property (i.e. aesthetics) of `color`

```
g + geom_point(aes(color=factor(cyl)))
```



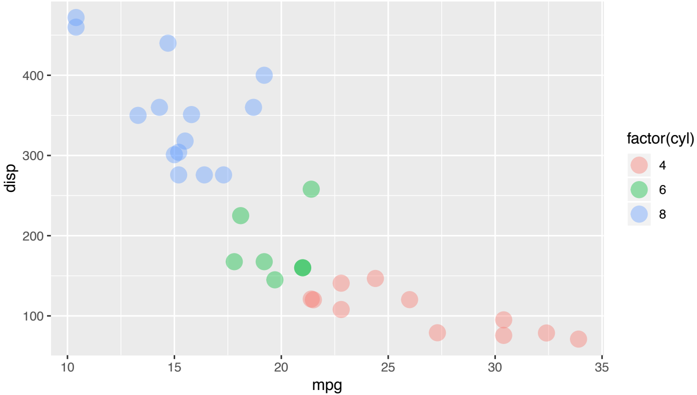
Alternatively, we could have mapped the variable `cyl` to the visual property of `shape`

```
g + geom_point(aes(shape=factor(cyl)))
```



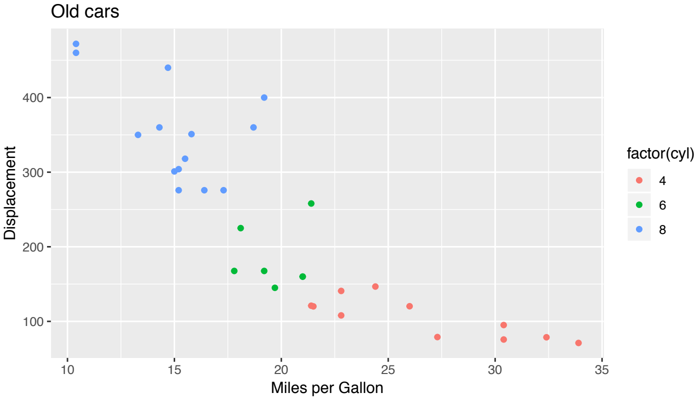
# Change the transparency and size of points

```
g + geom_point(aes(color=factor(cyl)), alpha=0.4, size=5)
```



# Add labels (see slide 49 for an alternative method)

```
p = g + geom_point(aes(color=factor(cyl)))
p + labs(title="Old cars", x="Miles per Gallon", y ="Displacement")
```



Notice how labels were added on a separate line of code.

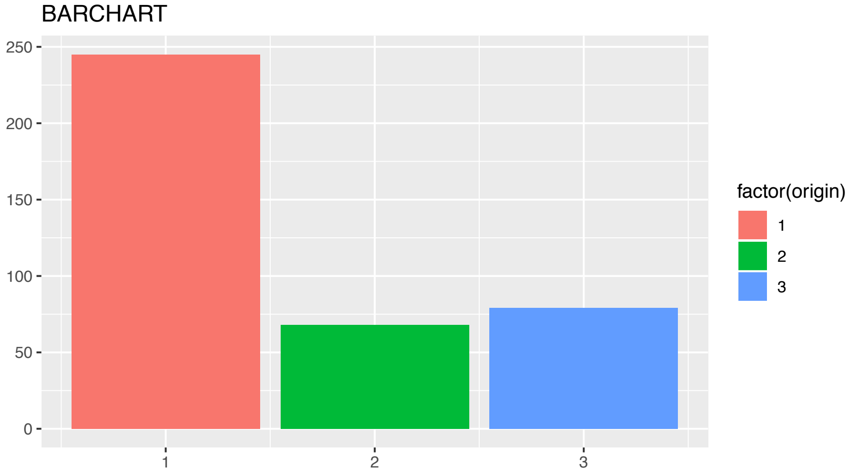
# Graphs for Qualitative Data: Bar Charts

Bar charts have each group along the $x$-axis and a vertical bar with the height representing the number of observations of each group.

See the next slide for an example using the dataset `Auto` in the from the ISLR package:

- A data frame with 392 observations on the following 9 variables.
- Gas mileage, horsepower, and other information (9 variables) for 392 vehicles.
- `orgin` = Origin of car (1. American, 2. European, 3. Japanese)

```r
library("ISLR")
ggplot(Auto, aes(x=origin)) + geom_bar(aes(fill=factor(origin))) +
  xlab("") + ylab("") + ggtitle("BARCHART")
```



# Graphs for Quantitative Data: Histogram

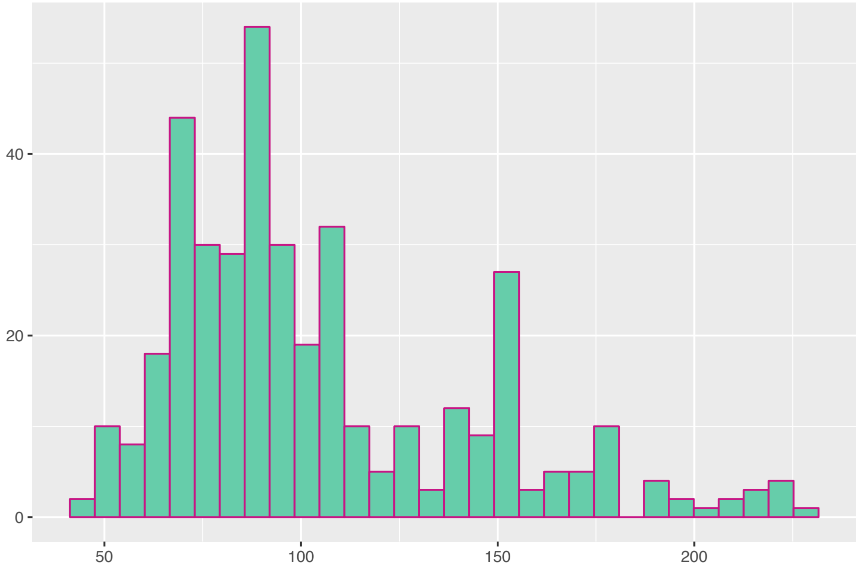
A histogram is similar to a bar chart, but the *x*-axis is divided into bins.

The variable of interest is on the *x*-axis and the *y*-axis represents count of observations within each bin.

Histograms provide a visualization of the data distribution.

```r
ggplot(Auto, aes(x=horsepower)) +
  geom_histogram(color='mediumvioletred', bins=30,
                 fill='mediumaquamarine') +
  xlab("") + ylab("") + ggtitle("HISTOGRAM")
```

# HISTOGRAM



# Graphs for Quantitative Data: Boxplot

A boxplot is a visualization of the five number summary.

1. Groups along the $x$-axis.
2. Data values along the $y$-axis.
3. Lowest and highest points are the min and max of the data respectively (excluding outliers).
4. Bottom of box is Q1 and top is Q3.
5. Median is represented as the bar inside the box.
6. Single points represent outliers.

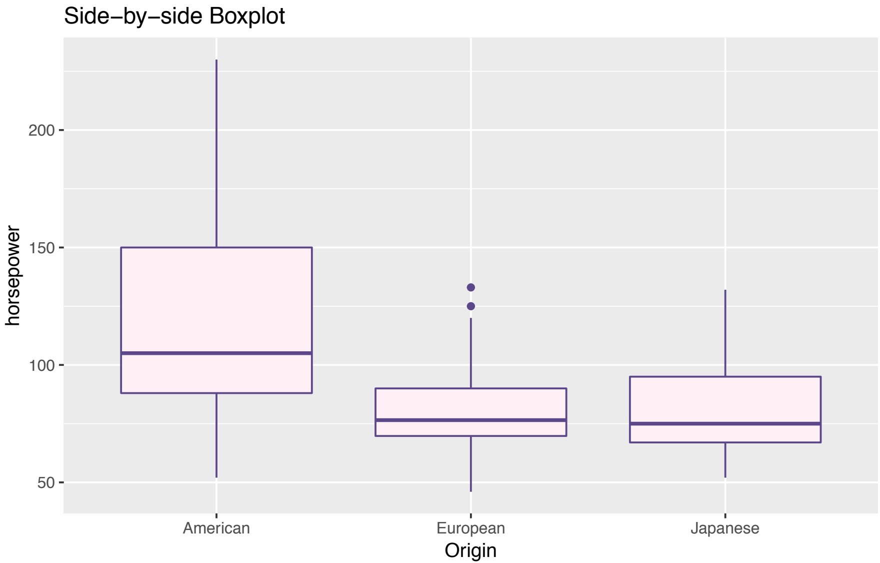
# Boxplot Example Code

We can create a so-called side by side boxplot by adding the `origin` category as an aesthetic mapping for the groupings along the *x*-axis.

```r
ggplot(Auto, aes(x=factor(origin), y=horsepower)) +
  geom_boxplot(color='mediumpurple4', fill='lavenderblush') +
  labs(title = "Side-by-side Boxplot", x="Origin") +
  scale_x_discrete(labels=c("American", "European", "Japanese"))
```

Rather than appearing as `1, 2`, 3, we rename the categoried `"American"`, `"European"`, and `"Japanese"`, respectively.

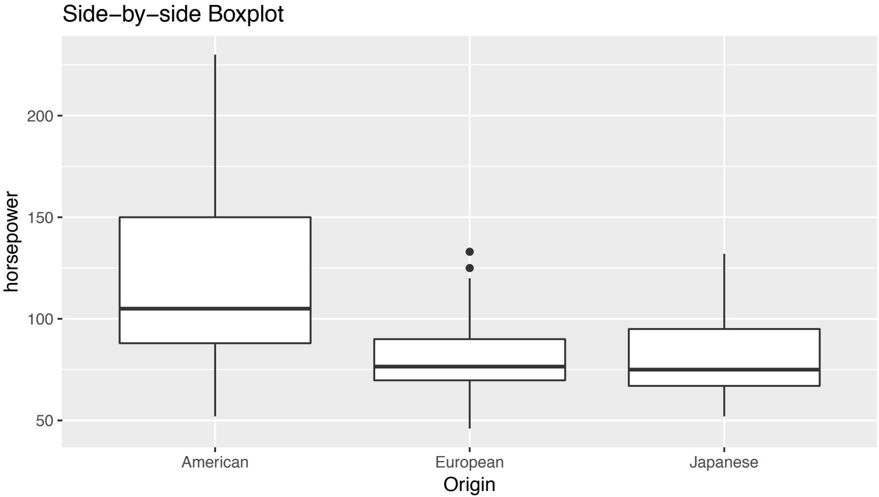
# Boxplot Example Code



- Notice how we rename the *x*-axis groupings using `scale_x_discrete` and force `orgin` to a factor using the `factor()` function.
- An alternative (and arguably better) method would be to update our data frame prior to doing any plotting:

```r
class(Auto$origin)

## [1] "numeric"

Auto$origin = factor(Auto$origin,
          labels = c("American", "European", "Japanese"))
class(Auto$origin)

## [1] "factor"
```

```
ggplot(Auto, aes(x=origin, y=horsepower)) + geom_boxplot() +
  labs(title = "Side-by-side Boxplot", x="Origin")
```



## Question

How many of the are following statements are true?

- Colours within the R palette can be referenced by number
- Boxplots display the five number summary of a data set.
- The `ggplot()` is available in **base** R (i.e. there is no need to load any packages).
- Layers are added to ggplot objects using the + operator.

A) 0 B) 1 C) 2 D) 3 E) 4

## Question

How many of the are following statements are true?

- Colours within the R palette can be referenced by number ✓
- Boxplots display the five number summary of a data set.
- The `ggplot()` is available in **base** R (i.e. there is no need to load any packages).
- Layers are added to ggplot objects using the + operator.

A) 0 B) 1 C) 2 D) 3 E) 4

## Question

How many of the are following statements are true?

- Colours within the R palette can be referenced by number ✓
- Boxplots display the five number summary of a data set. ✓
- The `ggplot()` is available in **base** R (i.e. there is no need to load any packages).
- Layers are added to ggplot objects using the + operator.

A) 0 B) 1 C) 2 D) 3 E) 4

## Question

How many of the are following statements are true?

- Colours within the R palette can be referenced by number ✓
- Boxplots display the five number summary of a data set. ✓
- The `ggplot()` is available in **base** R (i.e. there is no need to load any packages). ✗
- Layers are added to ggplot objects using the + operator.

A) 0 B) 1 C) 2 D) 3 E) 4

## Question

How many of the are following statements are true?

- Colours within the R palette can be referenced by number ✓
- Boxplots display the five number summary of a data set. ✓
- The `ggplot()` is available in **base** R (i.e. there is no need to load any packages). ✗
- Layers are added to ggplot objects using the + operator. ✓

A) 0 B) 1 C) 2 D) *3* E) 4