Week 2 Cheat Sheet

Statistics and Data Analysis with R

Course Link: https://www.coursera.org/learn/statistics-and-data-analysis-with-r/

Charlie Nuttelman

Here, I provide the functions in R required to perform various calculations in Week 2 of the course. The headings represent the screencasts in which you will find those concepts and examples.

Descriptive Statistics

In R, we can use the following functions to calculate basic statistics:

- **mean** (base): calculates the mean of a vector
- **max/min** (base): calculate the maximum and minimum values in a vector, data frame, or tibble
- **range** (base): outputs a vector of the min and max of a vector, data frame, or tibble
- **median** – see subsequent slides
- **mfv** (modeest): The **mode** function in base R is very different than the statistical mode. Fortunately, we can use the **mfv** function from the **modeest** library.
- **sd/var** (stats): calculate the standard deviation and variance of a vector, data frame, or tibble
- **mad** (stats): calculates the median absolute deviation of a vector, data frame, or tibble
- **summary** (base): also provides the min, max, and mean of a vector or each column of a data frame or tibble

We can also perform summing and averaging over columns and rows of a data frame or tibble using the following functions:

- **colSums** (base): Calculates the sum of all items in each column of a data frame or tibble
- **rowSums** (base): Calculates the sum of all items in each row of a data frame or tibble
- **colMeans** (base): Calculates the mean of all items in each column of a data frame or tibble
- **rowMeans** (base): Calculates the mean of all items in each row of a data frame or tibble

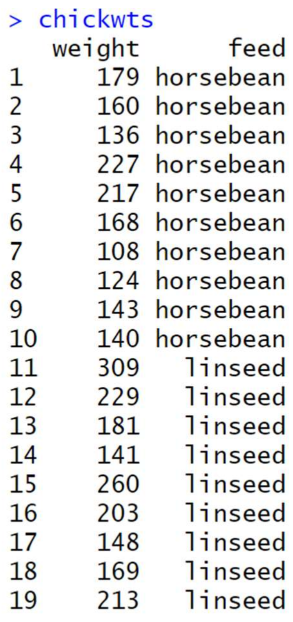
The **quantile** function in R can be used to calculate quartiles and percentiles and the **median** function can be used to determine the median of a set of data:

- **quantile** (stats): calculates and displays the quartiles (0%, 25%, 50%, 75%, and 100%) of a data set if no optional second argument is provided. If a second argument vector is provided, percentiles are output by this function [for example, **quantile(x,c(0.2,0.4,0.6,0.8))** will output the 20th, 40th, 60th, and 80th percentiles of the data in vector x].
- **median** (stats): calculates the median of a vector or column of a data frame or tibble. This is also the 2nd quartile (50% percentile)
- **summary** (base): also provides the quartiles

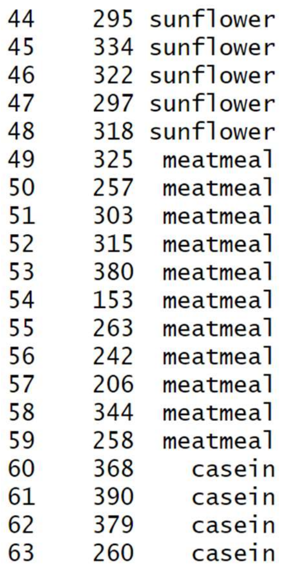
Conditional Statistics

For data sets where we have multiple columns, we oftentimes would like to only sum, count, or perform some other function (mean, standard deviation, max, min, others) on only those observations where a certain condition is met. In R, we can use the following techniques to perform conditional statistical calculations.

As an example, let's consider the **chickwts** data set, which is built into R (here, I'm only showing lines 1-19 and 44-63):



```
> chickwts
   weight      feed
1     179 horsebean
2     160 horsebean
3     136 horsebean
4     227 horsebean
5     217 horsebean
6     168 horsebean
7     108 horsebean
8     124 horsebean
9     143 horsebean
10    140 horsebean
11    309   linseed
12    229   linseed
13    181   linseed
14    141   linseed
15    260   linseed
16    203   linseed
17    148   linseed
18    169   linseed
19    213   linseed
```



```
44    295 sunflower
45    334 sunflower
46    322 sunflower
47    297 sunflower
48    318 sunflower
49    325  meatmeal
50    257  meatmeal
51    303  meatmeal
52    315  meatmeal
53    380  meatmeal
54    153  meatmeal
55    263  meatmeal
56    242  meatmeal
57    206  meatmeal
58    344  meatmeal
59    258  meatmeal
60    368    casein
61    390    casein
62    379    casein
63    260    casein
```

Let's say that we only wanted to perform calculations on the weight of chicks fed meatmeal:

- `mean(chickwts$weight[chickwts$feed=="meatmeal"])` will calculate the mean of chicks that were fed meatmeal
- `with(chickwts,mean(weight[feed=="meatmeal"]))` will also calculate the mean of chicks that were fed meatmeal
- `median(chickwts$weight[chickwts$feed=="meatmeal"])` or `with(chickwts,median(weight[feed=="meatmeal"]))` can be used to calculate the median weight of chicks that were fed meatmeal
- `sd(chickwts$weight[chickwts$feed=="meatmeal"])` or `with(chickwts,sd(weight[feed=="meatmeal"]))` can be used to calculate the standard deviation of the weight of chicks that were fed meatmeal
- `max(chickwts$weight[chickwts$feed=="meatmeal"])` or `with(chickwts,max(weight[feed=="meatmeal"]))` can be used to calculate the maximum weight of chicks that were fed meatmeal
- `length(chickwts$weight[chickwts$feed=="meatmeal"])` or `with(chickwts,length(weight[feed=="meatmeal"]))` can be used to calculate the number of chicks that were fed meatmeal
- Other functions can be used, as well

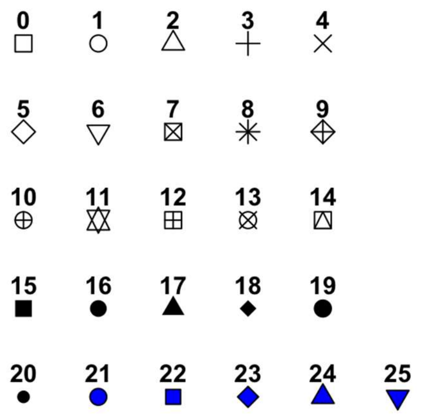
Scatter Plots

In R, we can use the following methods to create and tailor scatter plots:

- **plot** (graphics): creates a basic plot
- **lines** (graphics): good way to add one or more series to an existing plot
- **legend** (graphics): creates a figure legend
- **scatter.smooth** (stats): creates a scatter plot using a single series

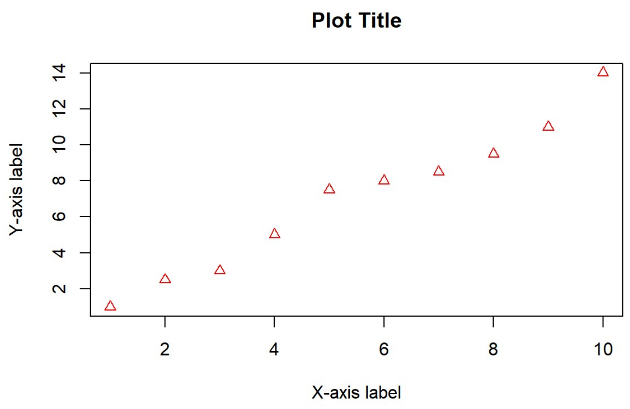
We can add or change many of the attributes of the plot. Common attributes that we might want to add or change include the following:

- **grid**: adds a grid to the plot
- **main**: title of the plot
- **xlab**: x-axis label
- **ylab**: y-axis label
- **cex**: size of the markers
- **col**: color of the markers
  - Common colors are “red”, ”darkred”, “orange”, “darkorange”, “yellow”, “green”, “darkgreen”, “blue”, “darkblue”, “purple”, “cyan”, “magenta”, and “black”.
- **pch**: characteristics of the plot markers (point characteristics)

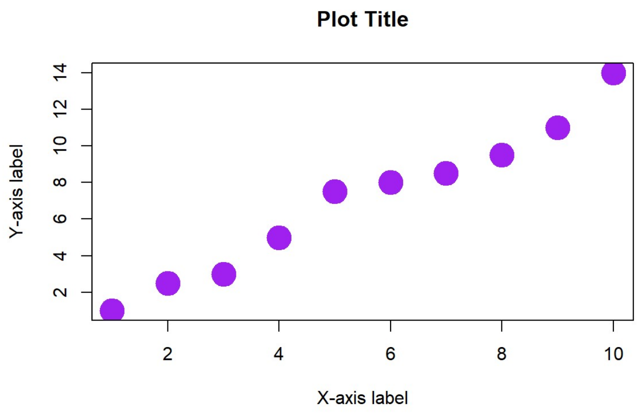


Reference: http://www.sthda.com/english/wiki/r-plot-pch-symbols-the-different-point-shapes-available-in-r

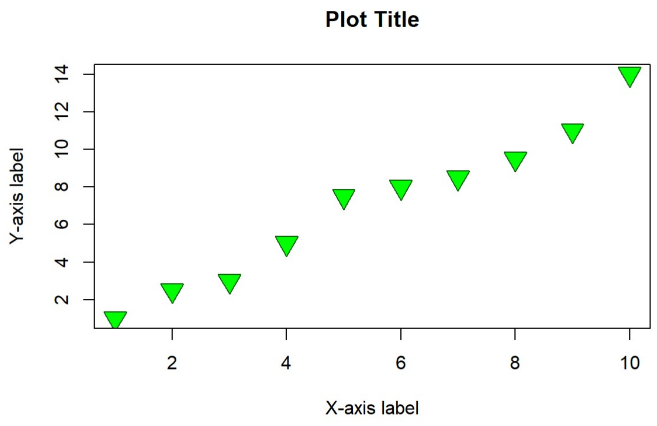
- For **pch** of 0-14, **col** determines the color of the border (fill will be white)
- For **pch** of 15-20, **col** determines the entire marker color
- For **pch** of 21-25, **col** determines the border color of the marker and **bg** determines the fill color
- Three examples:
  - `plot(x,y,main="Plot Title",xlab="X-axis label",ylab="Y-axis label",pch=2,col="red")`



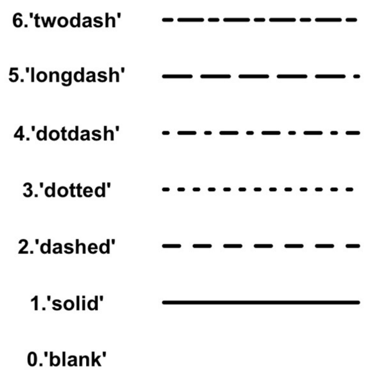
- `plot(x,y,main="Plot Title",xlab="X-axis label",ylab="Y-axis label",pch=19,col="purple",cex=3)`



- `plot(x,y,main="Plot Title",xlab="X-axis label",ylab="Y-axis label",pch=25,col="darkgreen",bg="green",cex=2)`

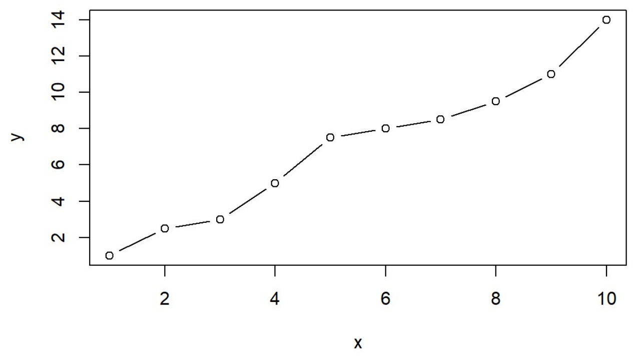


- We can add lines that connect markers of a scatter plot by adding the “**type**” optional argument.
  - `type=”p”` is the default (just points)
  - `type=”l”` is for lines only (no points)
  - `type=”b”` is for both lines and points
  - `type=”c”` is for lines but empty points
  - `type=”o”` is for both lines and points, lines extend into marker interior
  - `type=”s”` is for stair steps
  - `type=”h”` is for histogram-like vertical lines
  - `type=”n”` is for nothing
- **lwd**: line width
- **lty**: line type

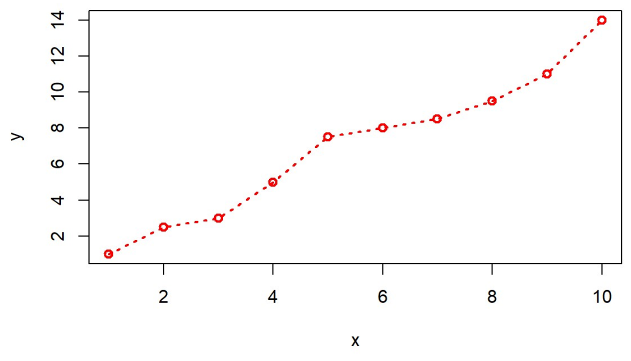


Reference: http://www.sthda.com/english/wiki/line-types-in-r-lty

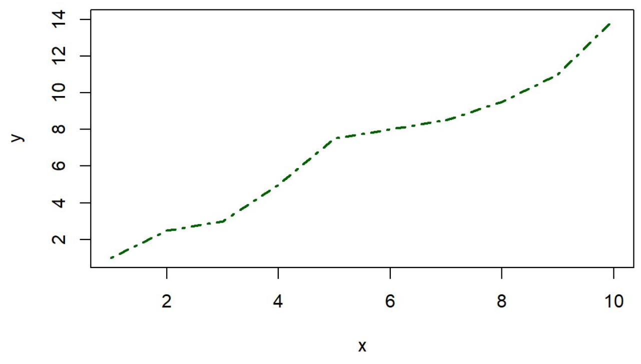
- Three examples:
  - `plot(x,y,type="b")`



- `plot(x,y,type="o",lty=3,lwd=2,col="red")`



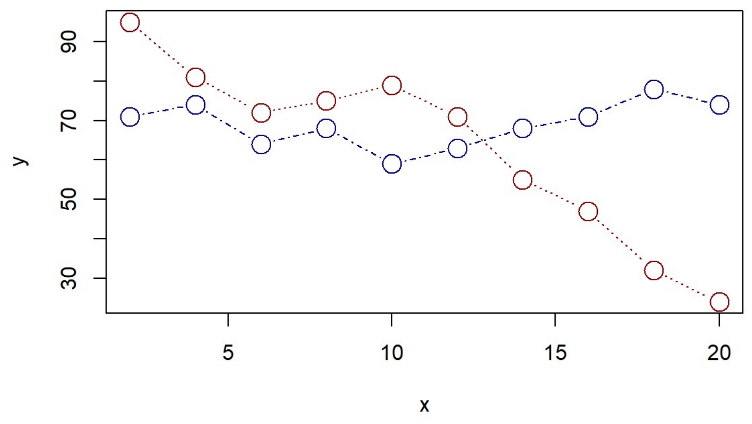
- `plot(x,y,type="l",lty=4,lwd=2,col="darkgreen")`



- To add more than one series to a scatter plot, we can use the **lines** function. Example:

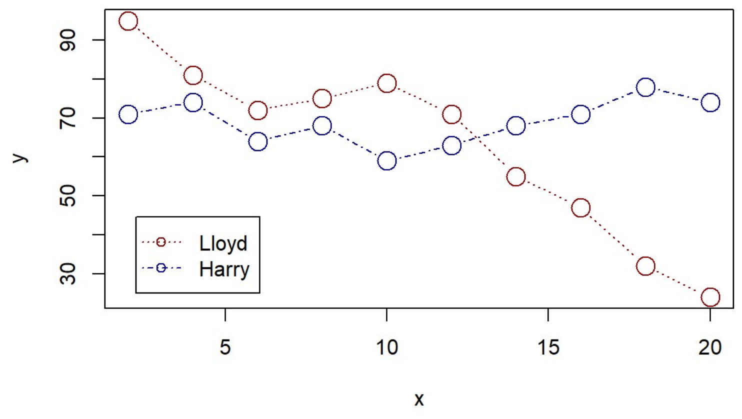
```
x <- seq(2,20,2)
y <- c(95,81,72,75,79,71,55,47,32,24)
z <- c(71,74,64,68,59,63,68,71,78,74)
plot(x,y,"b",cex=2,lty=3,col="darkred")
lines(x,z,"b",cex=2,lty=4,col="darkblue")
```

Result:



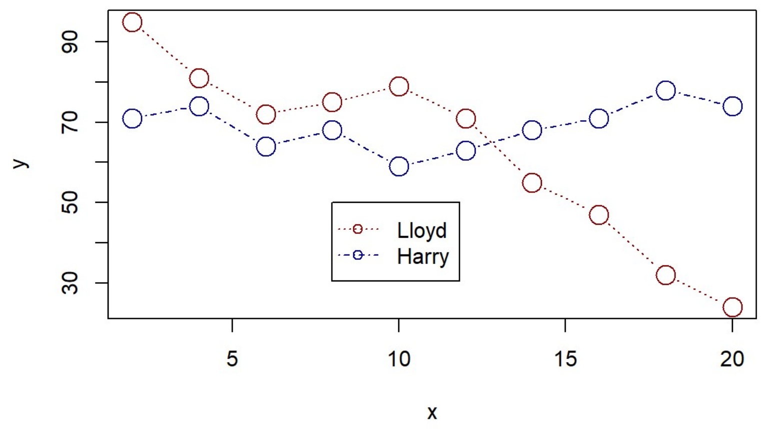
- Legends can be added using the **legend** function.
  - The x- and y-coordinates of the upper left corner of the legend can be specified or "`topleft`", "`left`", "`bottomleft`", "`top`", "`center`", "`bottom`", "`topright`", "`right`", or "`bottomright`" can be used to specify the location of the legend.
  - The legend can be inset using the **inset** optional argument followed by a decimal amount.
  - Example #1 (using the figure above as a starting point):

```
legend("bottomleft",
       inset=0.05,
       legend=c("Lloyd","Harry"),
       pch=c(1,1),
       lty=c(3,4),
       col=c("darkred","darkblue"))
```



- Example #2 (again using the figure at the top of the previous page as a starting point):

```
legend(x=8,
       y=50,
       legend=c("Lloyd","Harry"),
       pch=c(1,1),
       lty=c(3,4),
       col=c("darkred","darkblue"))
```



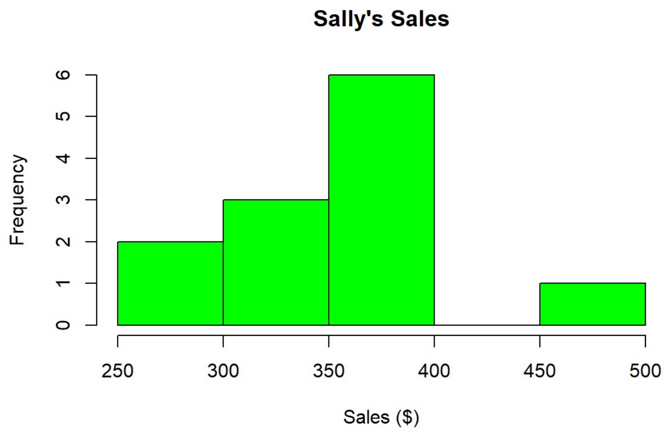
- Much more tailoring of aspects of these scatter plots can be performed. Lots of information on the internet and in various reference books, etc.

Histograms

In R, we can create a basic histogram using the **hist** function. The attributes of the histogram can be tailored using the optional arguments that follow.

- **hist** (graphics): creates a histogram of the data
- Common attributes:
  - **main**: used to set the title of the chart
  - **col**: used to set the color of the bars
  - **border**: used to set the border color of each bar
  - **xlab**: used to label the horizontal axis
  - **xlim**: used to set the x-scale
  - **ylim**: used to set the y-scale
  - **breaks**: used to set the number of bars in the histogram
  - **labels**: used to display values above each bar (TRUE/FALSE)
- As an example:
  - `hist(sally,main="Sally's Sales",xlab="Sales ($)",col="green",border="black")`

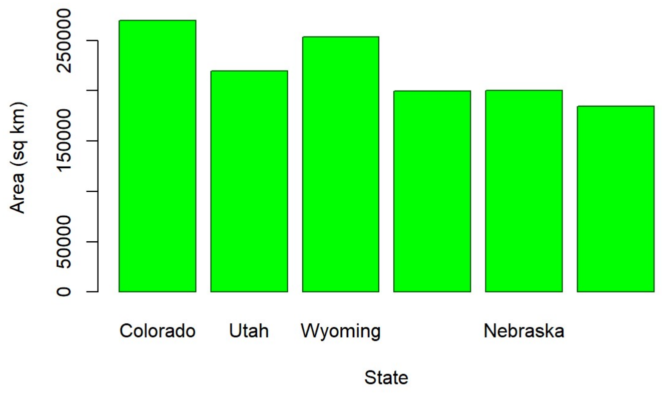
Result:



Column and Pie Plots

We can create column plots (also known as bar plots) in R using the **barplot** function. Attributes can be tailored using the optional arguments summarized below.

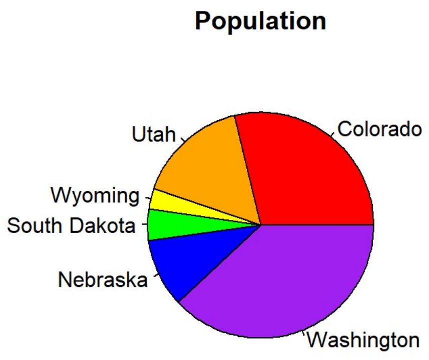
- **barplot** (graphics): creates a bar (column) plot of the data
- Common attributes:
    - **main**: used to set the title of the chart
    - **col**: used to set the color of the bars
    - **border**: used to set the border color of each bar
    - **xlab**: used to label the horizontal axis
    - **ylab**: used to label the vertical axis
    - **ylim**: used to set the y-scale
    - **names.arg**: used to label individual columns
- As an example:
    - `barplot(area,xlab="State",ylab="Area (sq km)",col="green",border="darkgreen", names.arg=states)`



(Note the issues with not all states showing up on the x-axis.)

Similarly, we can create pie plots in R using the **pie** function. Attributes can be tailored using the optional arguments summarized below.

- **pie** (graphics): creates a pie chart of the data
- Common attributes:
    - **labels**: a vector of strings used to label the slices
    - **main**: used to set the title of the chart
    - **col**: used to set the color of the slices (a vector)
    - **border**: used to set the border color of the chart
- As an example:
    - `pie(population,labels=states,main="Population",col=c("red", "orange","yellow","green","blue","purple"))`



Box Plots

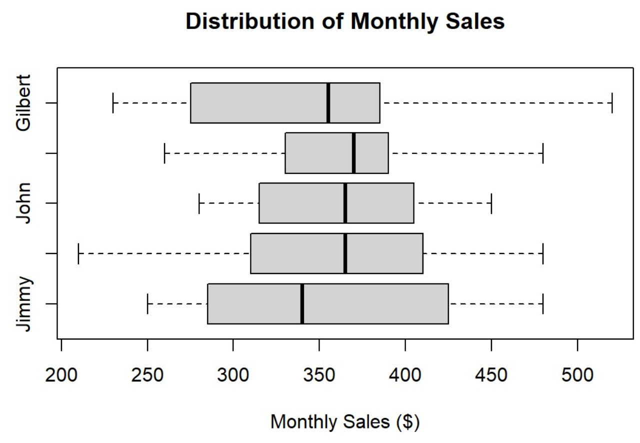
We can use the **boxplot** function in R to create basic box plots, either of a single series or of multiple series.

- **boxplot** (graphics): creates a box plot of the data
- Common attributes:
    - **main**: used to set the title of the chart
    - **xlab**: used to label the horizontal axis
    - **ylab**: used to label the vertical axis
    - **col**: used to set the color of the bars
    - **border**: used to set the border color of each bar
    - **xlim**: used to set the x-scale
    - **ylim**: used to set the y-scale
    - **horizontal**: TRUE for horizontal boxes, FALSE for vertical boxes

- Example #1:
    - Let's create a single figure that shows 5 different box plots, one for each salesperson, using the following **sales** data frame:

```
> sales
# A tibble: 12 × 6
   Month     Jimmy   Sue  John Sally Gilbert
   <chr>     <dbl> <dbl> <dbl> <dbl>   <dbl>
 1 January     290   250   280   260     330
 2 February    310   310   320   390     240
 3 March       420   390   420   380     360
 4 April       280   400   300   350     280
 5 May         370   320   450   390     360
 6 June        440   480   320   360     520
 7 July        480   470   450   390     390
 8 August      430   310   380   480     390
 9 September   300   410   390   290     350
10 October     260   410   310   380     380
11 November    410   340   380   320     270
12 December    250   210   350   340     230
```

    - `boxplot(sales,horizontal=TRUE,xlab="Monthly Sales ($)",main="Distribution of Monthly Sales")`

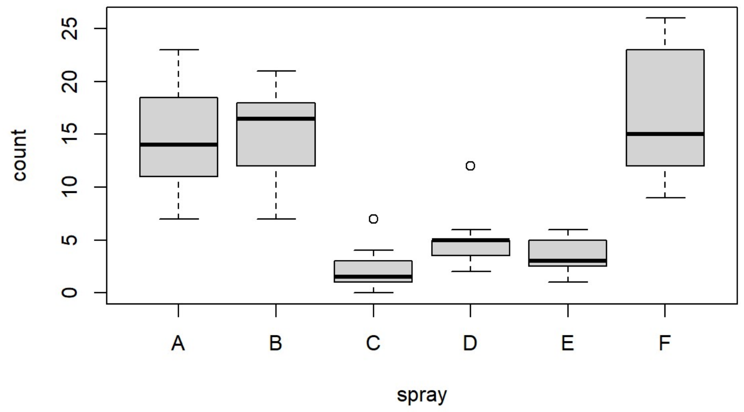


- Example #2:
    - Let's create a single figure that shows 6 different box plots, one for each of the 6 different insect sprays (A through F) of the **InsectSprays** data set. The data in the **InsectSprays** data set are formatted in long (or narrow) format; shown here are the first 20 or so lines:

```
> InsectSprays
   count spray
1     10     A
2      7     A
3     20     A
4     14     A
5     14     A
6     12     A
7     10     A
8     23     A
9     17     A
10    20     A
11    14     A
12    13     A
13    11     B
14    17     B
15    21     B
16    11     B
17    16     B
18    14     B
19    17     B
20    17     B
```

(The data goes all the way through spray = F.)

- To deal with narrow format, we can use the following technique to create the desired boxplot: `boxplot(count~spray,data=InsectSprays)`
  (This can also be tailored using the optional arguments outlined above.)



Probability Plots and the AD Statistic

We can use the **qqnorm** and **qqline** functions to create a quantile-quantile probability plot:

- **qqnorm** (stats): creates a quantile-quantile plot (a type of probability plot)
- **qqline** (stats): adds a line to the **qqnorm** plot (must be executed after the **qqnorm** statement above)

The above plot is more of a subjective technique; if most of the data points lie along a straight line, then we can assume that the data is normally distributed. An objective test can be performed by using the **ad.test** function of the **nortest** library, which calculates a P-value of the Anderson-Darling (AD) statistic:

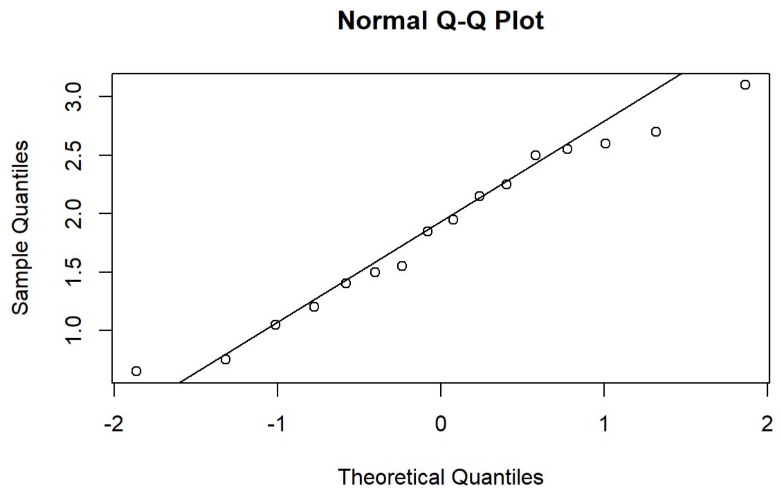
- **ad.test** (nortest): calculates the P-value of the AD statistic (as well as the value of the AD statistic, which is fairly meaningless)
- The **ad.test** function requires a vector of at least 8 items

If the P-value of the AD statistic is greater than 0.05, we can assume that the data are normally distributed.

Example:

```
qqnorm(dataA)

qqline(dataA)
```



```
ad.test(dataA)
```

```
> ad.test(dataA)

        Anderson-Darling normality test

data:  dataA
A = 0.21359, p-value = 0.8202
```

Conclusion: Since the p-value (0.82) of the AD statistic is greater than 0.05, we can assume that the data are normally distributed.

We will use these techniques later on in the course, especially when we start building mathematical regression models.