Introduction R for Stata Users

Day 2

Laura Garcia Montoya

6/26/2019

1. Writing Functions in R

"In programming, you use functions to incorporate sets of instructions that you want to use repeatedly or that, because of their complexity, are better self-contained in a sub program and called when needed. A function is a piece of code written to carry out a specified task; it can or can not accept arguments or parameters and it can or can not return one or more values."

Yesterday, we learned that functions are an important part of using R. By learning how to create simple functions in R you will have more tools to understand the documentation of different R functions and packages.

Example

Let's say we want to build a function that converts temperatures from Fahrenheit to Celsius.If we were to do this by hand, we need to make sure we understand the process required to convert temperatures from Fahrenheit to Celsius. In particular, we need to know that\footnote{This example is from DataCarpentry: https://swcarpentry.github.io/r-novice-inflammation/02-func-R/:

$T(C) = (T(F) - 32) * \frac{5}{9}$

Before we start trying this in R, we need to make sure we know the answers to the following questions:

- What do we want our function to do?

  We want our function to take temperatures measured in Fahrenheit, apply the formula,and return a value in Celsius.

- What is the input - Domain?

  A temperature measured in Fahrenheit.

- What is the output - Codomain?

  The equivalent temperature measured in celsius.

1.1 Structure of Functions in R:

1) We need to decide what the name of our function will be. Let's say $fahrenheit\_to\_celsius$ Then, we need to tell R that $fahrenheit\_to\_celsius$ is an object of the type function.

```
fahrenheit_to_celsius <- function() {

}
```

2) We need to declare the name of the input. We will use $temp\_F$ to refer to the input. Inputs go inside the parenthesis after function.

```
fahrenheit_to_celsius <- function(temp_F){
}
```

3) We will also need to name the output of the function. Let's use *temp_C* for this example.

```
fahrenheit_to_celsius <- function(temp_F) {
  temp_C <-
}
```

4) We are now ready to write the function using the input, output and process.

```
fahrenheit_to_celsius <- function(temp_F) {
  temp_C <- ((temp_F - 32) * (5 / 9))
}
```

5) Finally, we need to specify what we want the function to return. (Not necessary if only one output, but a good practice to when we write functions iwth more than one outcome)

```
fahrenheit_to_celsius <- function(temp_F){
  temp_C <- ((temp_F - 32) * (5 / 9))
  return(temp_C)
}
```

What if instead of Celsius we wanted Kelvin?

```
fahrenheit_to_kelvin <- function(temp_F){
  temp_K <-(temp_F - 32) * (5 / 9) + 273.15
  return(temp_K)
}
```

Alternatively, we can also call a function inside a function. A composite function:

```
celsiustokevin <- function(temp_C){
  temp_K <- temp_C - 273.15
  return(temp_K)
}
```

Now, we are ready to use our functions: Either with single values:

```
fahrenheit_to_celsius(40)
```

```
## [1] 4.444444
```

```
fahrenheit_to_kelvin(40)
```

```
## [1] 277.5944
```

```
celsiustokevin(40)
```

```
## [1] -233.15
```

Or with vectors:

```
temperatures_F<-c(42,40,35,34,39,34,35)

fahrenheit_to_celsius(temperatures_F)
```

```
## [1] 5.555556 4.444444 1.666667 1.111111 3.888889 1.111111 1.666667
```

1.2. Confidence Intervals

"In statistics, a confidence interval (CI) is a type of interval estimate, computed from the statistics of the observed data, that might contain the true value of an unknown population parameter. The interval has an associated confidence level that, loosely speaking, quantifies the level of confidence that the parameter lies in

the interval. More strictly speaking, the confidence level represents the frequency (i.e. the proportion) of possible confidence intervals that contain the true value of the unknown population parameter. In other words, if confidence intervals are constructed using a given confidence level from an infinite number of independent sample statistics, the proportion of those intervals that contain the true value of the parameter will be equal to the confidence level." (Wikipedia)

Let's construct our own function that calculates the bounds of a confidence interval for a given estimate.

```
confidence_z<-function(estimate,alpha,se)
  {
    lower_bound<-estimate-se*qnorm(1-(alpha/2))
    upper_bound<-estimate+se*qnorm(1-(alpha/2))
    return(c(lower_bound, upper_bound))
}
```

Practice:

1. Calculate 95% confidence intervals (alpha=0.05) for the following:
   - Mean 12, variance 5, sample size 150
   - Mean 12, variance 5, sample size 1500
   - Mean 12, standard deviation 5, sample size 1500

```
#Answer 1
confidence_z(12,0.05,sqrt(5/150))
```

```
## [1] 11.64216 12.35784
```

2. Calculate 90% confidence intervals for the following:

- Mean 12, variance 5, sample size 150
- Mean 12, variance 5, sample size 1500
- Mean 12, standard deviation 5, sample size 1500

```
#Answer 1
confidence_z()
```

3. Complete the code below to create a function that calculates a confidence interval using the t distribution. Use the function qt() to get the value from the t distribution. Note 1: for this you will need to input the degrees of freedom too. Note 2: $df = n - 1$

```
confidence_t<-function(estimate,conf_level,n,se){
  lower_bound<-
  upper_bound<-
  return(c(lower_bound,upper_bound))
}
```

4. Analyze the code below and make sure you understand this function.

```
confidence_n<-function(estimate,conf_level,n,se){
  if (n>30) {
  lower_bound<-estimate-se*qnorm(conf_level)
  upper_bound<-estimate+se*qnorm(conf_level)
} else if (n<=30) {
  lower_bound<-estimate-se*qt(conf_level,n-1)
  upper_bound<-estimate+se*qt(conf_level,n-1)
}
```

```
  return(c(lower_bound,upper_bound))
}
```

5. Calculate 95% confidence intervals for the following:

- Mean 12, $SE = 5$, sample size 15
- Mean 12, $SE = 3$, sample size 15
- Mean 12, $SE = 5$, sample size 35

```
confidence(12,0.05,15,5)
```

2. Back to Data Frames and descriptive statistics

Importing the Data Set

1. Download and save the dataset "Data_example_Class.xlsx". KEY: pay attention to where you saved the file.
2. Check your working directory.

```
getwd()
```

```
## [1] "/Users/lauragarciamontoya/Box Sync/PhD/R Workshop/R for Stata Users Workshop"
```

3. If necessary, change your working directory to the folder where your saved "Data_example_Class.xlsx"

```
setwd("/Users/lauragarciamontoya/Box Sync/PhD/R Workshop/R for Stata Users Workshop/")
```

4. Import the dataset "Data_example_Class.xlsx" using code. You will need package "readxl". Make sure you install it first.

```
library(readxl) #install package if necessary
data_example<- read_excel("Data_example_class.xls")
```

Summarizing the dataset:

The function summary() will return descriptive statistics of the object you input.

If the object is a data frame it will return these for all of the variables. You can also use the function for a list of variables or just one variable.

```
summary(data_example)
summary(data_example[,2:3])
summary(data_example$aSETERCUATBAMAZS)
```

Names and Heads:

These two functions are useful to explore what the data set contains: - head() returns the first 6 rows of the dataset - names() returns the names of the variables.

```
head(data_example)
names(data_example)
```

Note on renaming variables:

Some datasets, like the one we are using, have hard variable names that are hard to interpret. There is an easy way to label those.

Let's try rename two variables:

```
names(data_example)[names(data_example) == 'aSIPOVGINI'] <- 'Gini_coef'

names(data_example)[names(data_example) == 'aNYGDPMKTPKD'] <- 'GDP_constant'
```

Simple funtions to describe data

Mean and median:

```
mean_x<-mean(c(6,2,3,9,10))
median_x<-median(c(6,2,3,9,10))
```

Let's estimate the mean and median of the Gini Coefficient and GDP in constant US dollars from our data set.

```
mean(data_example$Gini_coef, na.rm=TRUE) #Add "na.rm=TRUE" if variable has missing values.
```

```
## [1] 37.88181
```

```
mean(data_example$GDP_constant, na.rm=TRUE)
```

```
## [1] 324201270192
```

```
median(data_example$Gini_coef, na.rm=TRUE)
```

```
## [1] 35.75
```

```
median(data_example$GDP_constant, na.rm=TRUE)
```

```
## [1] 20888663889
```

Or the variance and standard deviation:

```
#Variance of Gini Coefficient
var(data_example$Gini_coef, na.rm=TRUE)
#Standard Deviation of Gini Coefficient
sd(data_example$Gini_coef, na.rm=TRUE)
```

Other useful functions: Minimum, Maximum and range.

```
min(data_example$Gini_coef, na.rm=TRUE)
```

```
## [1] 16.2
```

```
max(data_example$Gini_coef, na.rm=TRUE)
```

```
## [1] 64.8
```

```
range(data_example$Gini_coef, na.rm = TRUE)
```

```
## [1] 16.2 64.8
```

Combining indexing with descriptive statistics:

We can use indexing to calculate more detailed descriptive statistics.

```
#Mean gini coefficient for the year 2004.
mean(data_example$Gini_coef[data_example$year==2004], na.rm = TRUE)
```

```
## [1] 37.97164
```

```
#Mean gini coefficient for country Colombia.
mean(data_example$Gini_coef[data_example$CountryName=="Colombia"], na.rm=TRUE)
```

```
## [1] 54.16
```

Or to test logical statements:

```
mean(data_example$Gini_coef[data_example$year==2004], na.rm = TRUE) > mean(data_example$Gini_coef[data_
```

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

Creating variables

We will use all of these skills to create a new variable to measure if a country is above or below the mean on: Educational attainment, at least completed upper secondary, population 25+, total (%) (cumulative) relatively educated. (varname: aSESECCUATUPZS)

```
data_example$above_educ_mean<-data_example$aSESECCUATUPZS>mean(data_example$aSESECCUATUPZS, na.rm = TRU

summary(data_example$above_educ_mean)
```

```
##    Mode   FALSE    TRUE    NA's
## logical     351     382    3173
```

Function aggregate

The function aggregate() is useful if you want to have descriptive statistics that are more detailed. Look at the documentation of this function. (hint: type ?aggregate) (Similar to summarise in tidyverse)

Similar lo collapse in Stata

```
means_vector<-aggregate(data_example$Gini_coef, list(data_example$year), mean, na.rm=TRUE)

sd_vector<-aggregate(data_example$Gini_coef, list(data_example$year), sd, na.rm=TRUE)

min_vector<-aggregate(data_example$Gini_coef, list(data_example$year), min, na.rm=TRUE)

max_vector<-aggregate(data_example$Gini_coef, list(data_example$year), max, na.rm=TRUE)

#Combines vectors
des_all<-cbind(year=means_vector[,1],mean_gini=means_vector[,2],
               sd_gini=sd_vector[,2],min_gini=min_vector[,2],max_gini=max_vector[,2])
```

Practice:

1. Interpret the code above and the resulting table.
2. Use the code above to generate an equivalent table but for the variable GDP.
3. Creating new variable:

Repeat what we did above with variable aSESECCUATUPZS. This time instead of creating a dummy variable, create a variable that takes values 1,2,3,4 depending on the quartile. It will take the values: 1 if observation falls on first quartile (percetiles (0-25]). 2 if it falls between (25,50]. 3 if it belongs to the third quartile (50,75] and 4 if it belong to the fourth quartile (75,100].

There are multiple ways to do this! (hint: quantile(data, c(0.25, 0.5, 0.75), type = 1), summary(), ifelse() might help you).

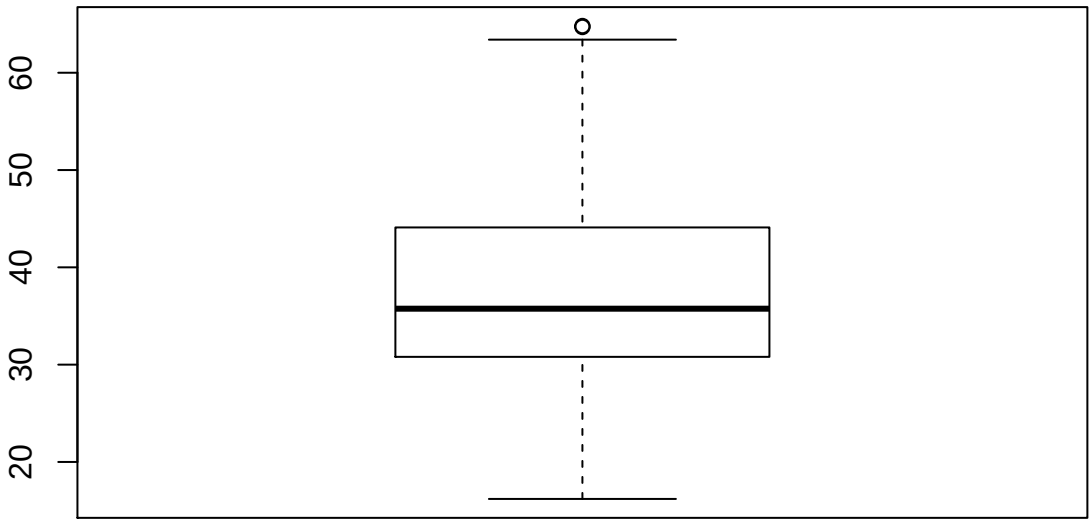
4. What is the mode of the variable that you created above?
5. Check which year has a higher Gini_coef in the United States: 2000 or 2004?

3. Intro to plots in Basic R:

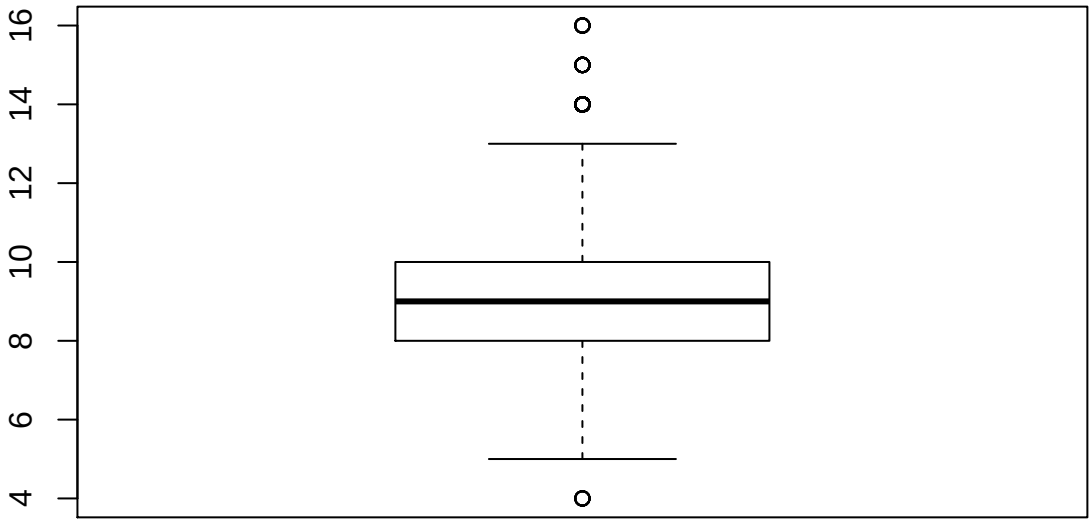
(See Cheat sheet: "BaseGraphicsCheatsheet.pdf")

Example using boxplot

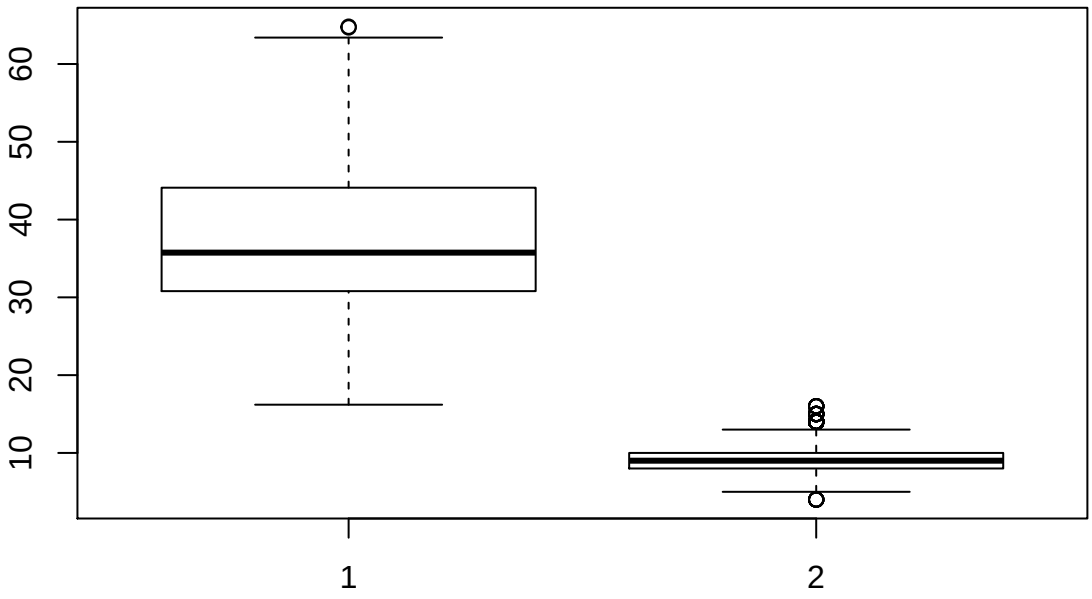
```
boxplot(data_example$Gini_coef)
```




```
boxplot(data_example$aSECOMDURS)
```

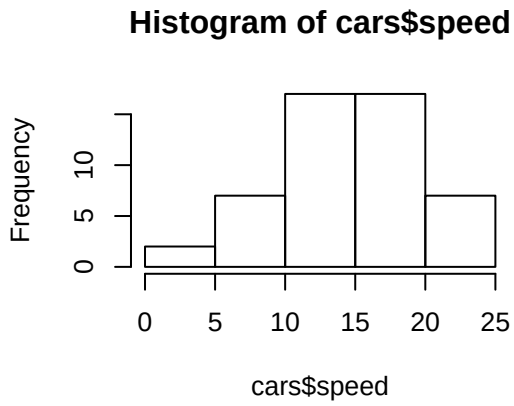


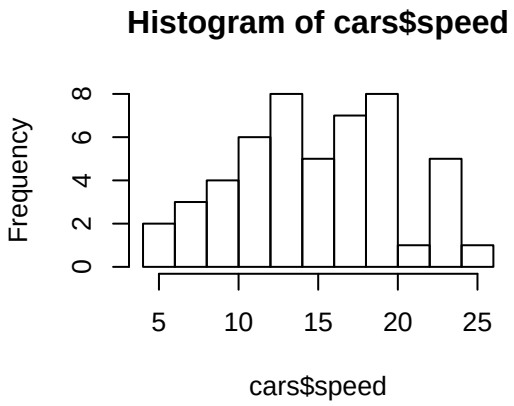

```
boxplot(data_example$Gini_coef, data_example$aSECOMDURS)
```

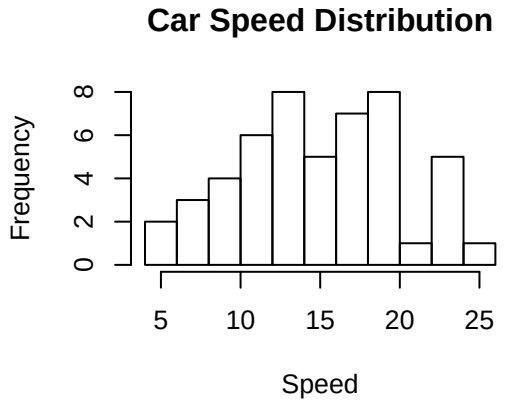


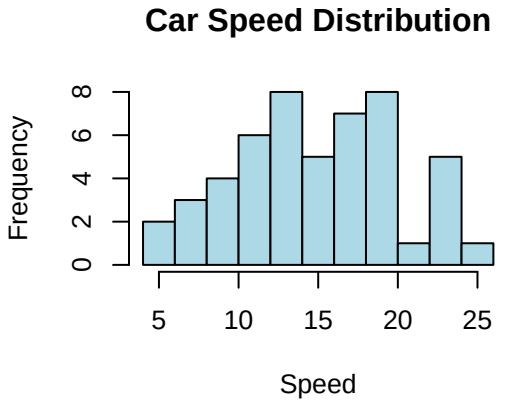
Histograms

```
par(mfrow=c(2,2)) #Allows to have plots side by side
#1
hist(cars$speed)

#2
hist(cars$speed, breaks=10)

#3
hist(cars$speed, breaks=10 , xlab = "Speed", main = "Car Speed Distribution")

#4
hist(cars$speed, breaks=10 , xlab = "Speed", main = "Car Speed Distribution", col="lightblue")
```



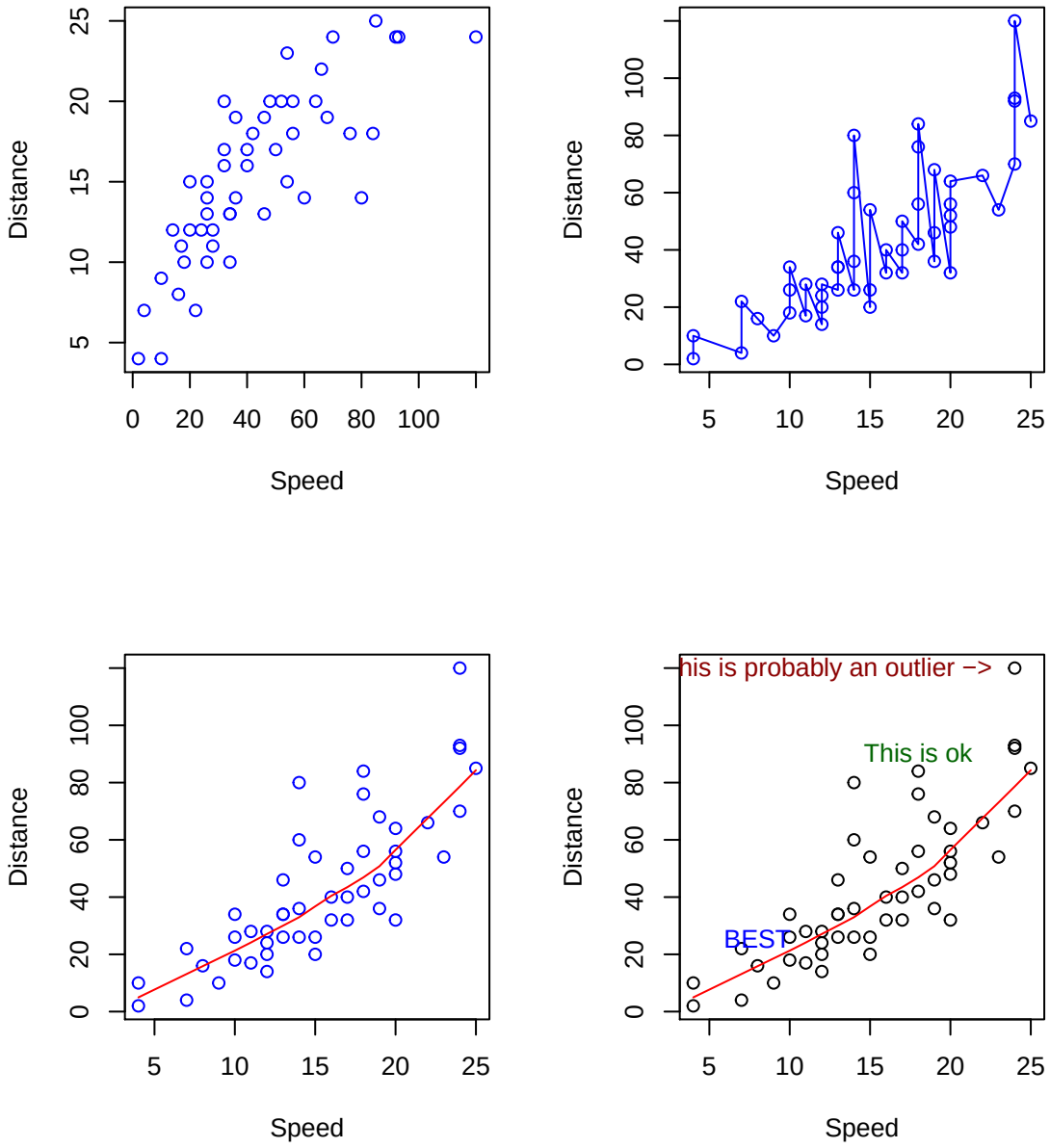






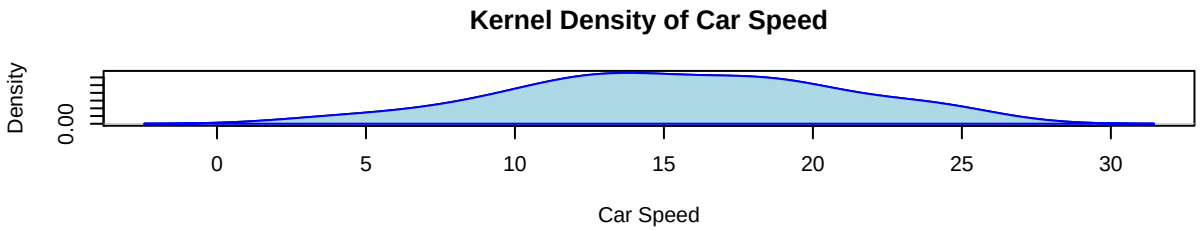
Scatter Plots

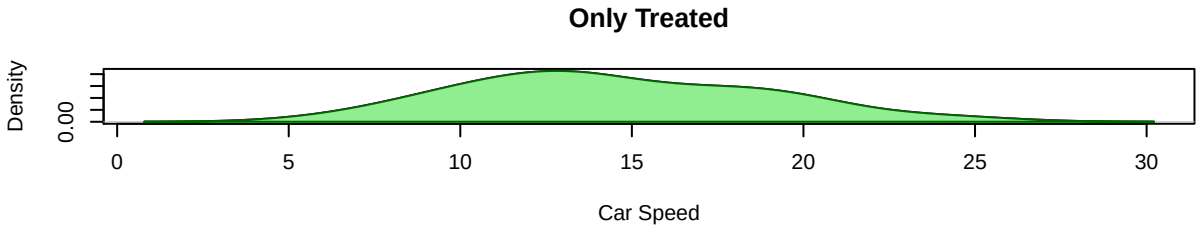
```
par(mfrow=c(2,2))

plot(cars$speed~cars$dist,col="blue", xlab="Speed", ylab="Distance")

plot(cars$speed,cars$dist,col="blue", xlab="Speed", ylab="Distance")
lines(cars$dist~cars$speed, col="blue")

plot(cars$speed,cars$dist,col="blue", xlab="Speed", ylab="Distance")
lines(lowess(cars$dist~cars$speed), col="red")

plot(cars$speed,cars$dist,col="black", xlab="Speed", ylab="Distance")
lines(lowess(cars$dist~cars$speed), col="red")
text(18,82, "This is ok", col="Dark Green", pos=3)
text(8,17, "BEST", col="Blue", pos=3)
text(24,119.5, "This is probably an outlier -> ", col="Dark red", pos=2)
```

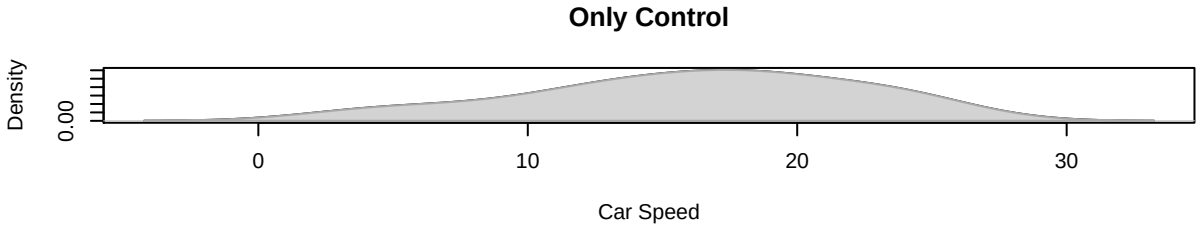


Distribution plots

```
d_speed<-density(cars$speed)
cars$treatment<-rbinom(50,1,0.4)
d_speed_t<-density(cars$speed[cars$treatment==1])
d_speed_c<-density(cars$speed[cars$treatment==0])
par(mfrow=c(3,1))
plot(d_speed , main="Kernel Density of Car Speed", xlab="Car Speed")
polygon(d_speed, col="light blue", border="blue")
plot(d_speed_t , main="Only Treated", xlab="Car Speed")
polygon(d_speed_t, col="light green", border="dark green")
plot(d_speed_c , main="Only Control", xlab="Car Speed")
polygon(d_speed_c, col="light gray", border="dark gray")
```

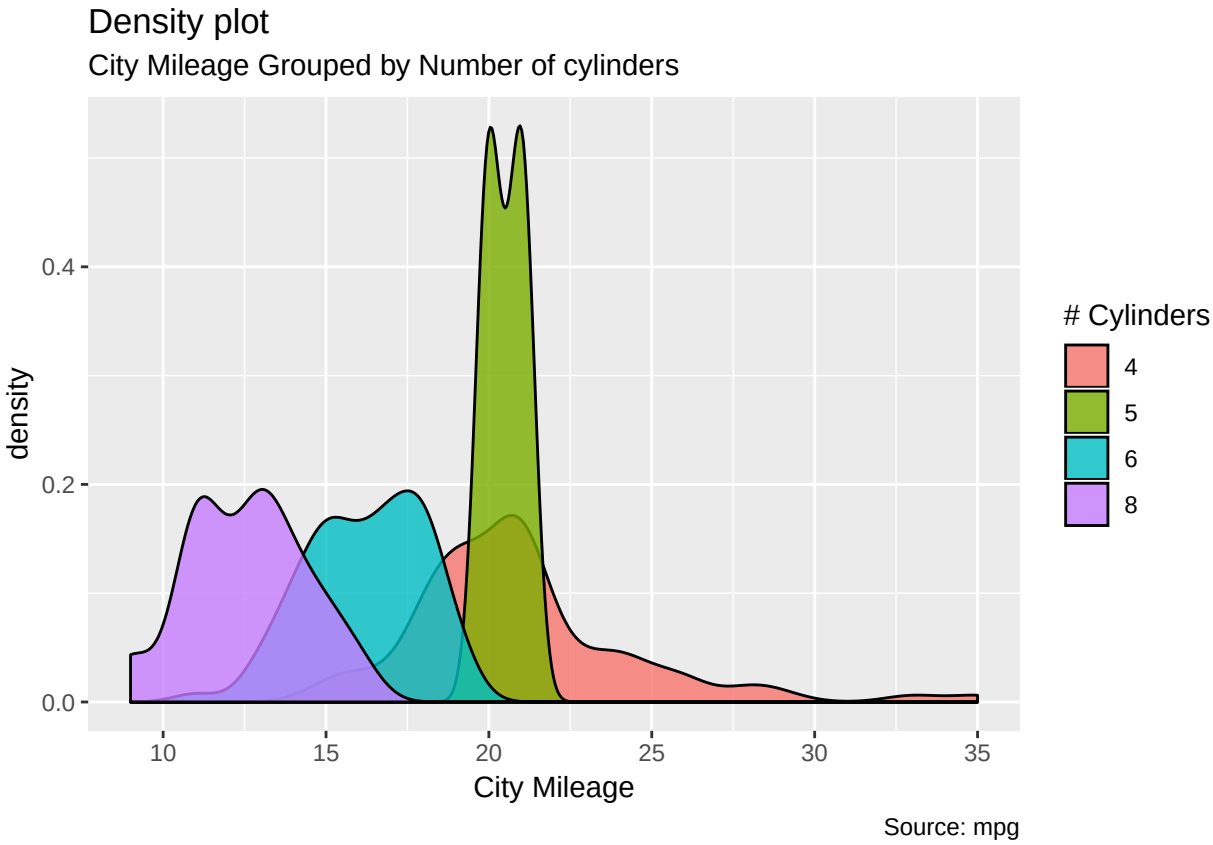

Kernel Density of Car Speed
Density
0.00
0
5
10
15
20
25
30
Car Speed


Only Treated
Density
0.00
0
5
10
15
20
25
30
Car Speed


Only Control
Density
0.00
0
10
20
30
Car Speed

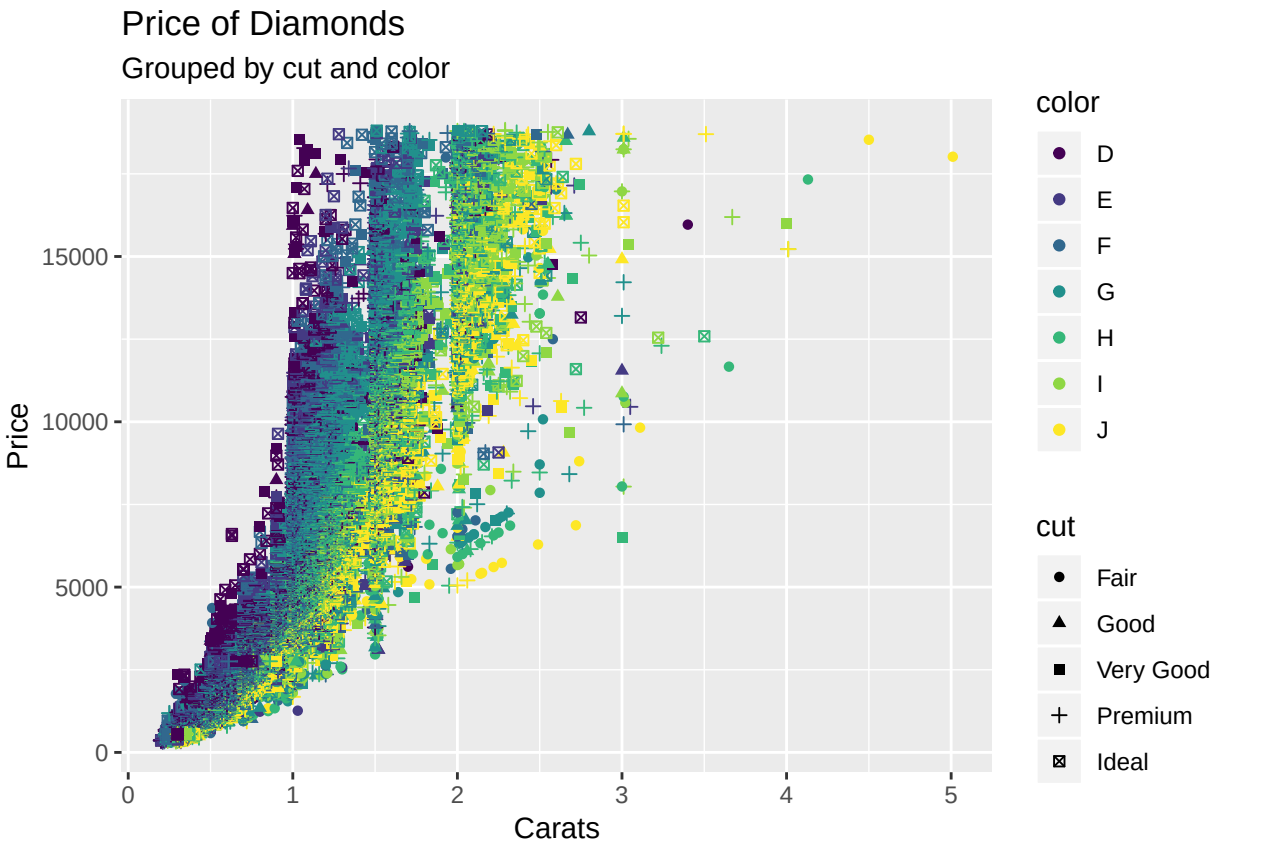
4. Ggplot2

Ggplot2 is a very useful package to graph things in R. It produces extremely cool graphs such as:



And:

```
## Warning: Using shapes for an ordinal variable is not advised
```



I don't want to overwhelm you with all of the details behind the package ggplot2 and I don't expect you to understand everything that is going on. It's hard to sumarize how ggplot2 works because it is complex. T

0. Download and open in R the script titled "R_script_ggplot2.R"

This script has some code and comments that will help you go through this session.

1. Install and Load package:

The name of the package is ggplot2. The first thing we need to do is install and load the package.

2. The data:

We will use a package called "datasets" that contains multiple datasets that we can use to produce plots. You will need to install and load package datasets.

```
## Warning: package 'datasets' is not available (for R version 3.5.0)
```

```
## Warning: package 'datasets' is a base package, and should not be updated
```

3. Main function Ggplot:

The main function of the package ggplot2 is ggplot.

Structure of the function:

```
ggplot( name of dataset,  aes  ( name variable x , name variable y, colour = smt, shape = smt)) + geom_
```

name of dataset: just type the name of the dataset that contains the variables you are interested in graphing.

aes() aesthetic means the things you can see. Examples include: position (i.e., on the x and y axes), color, fill, shape, linetype, size.

name of variable x: name of the variable you want to display in x axxis. Note: this variable needs to exist: type names(nameofdataset) to get list of names name of variable y: name of the variable you want to display in y axxis.

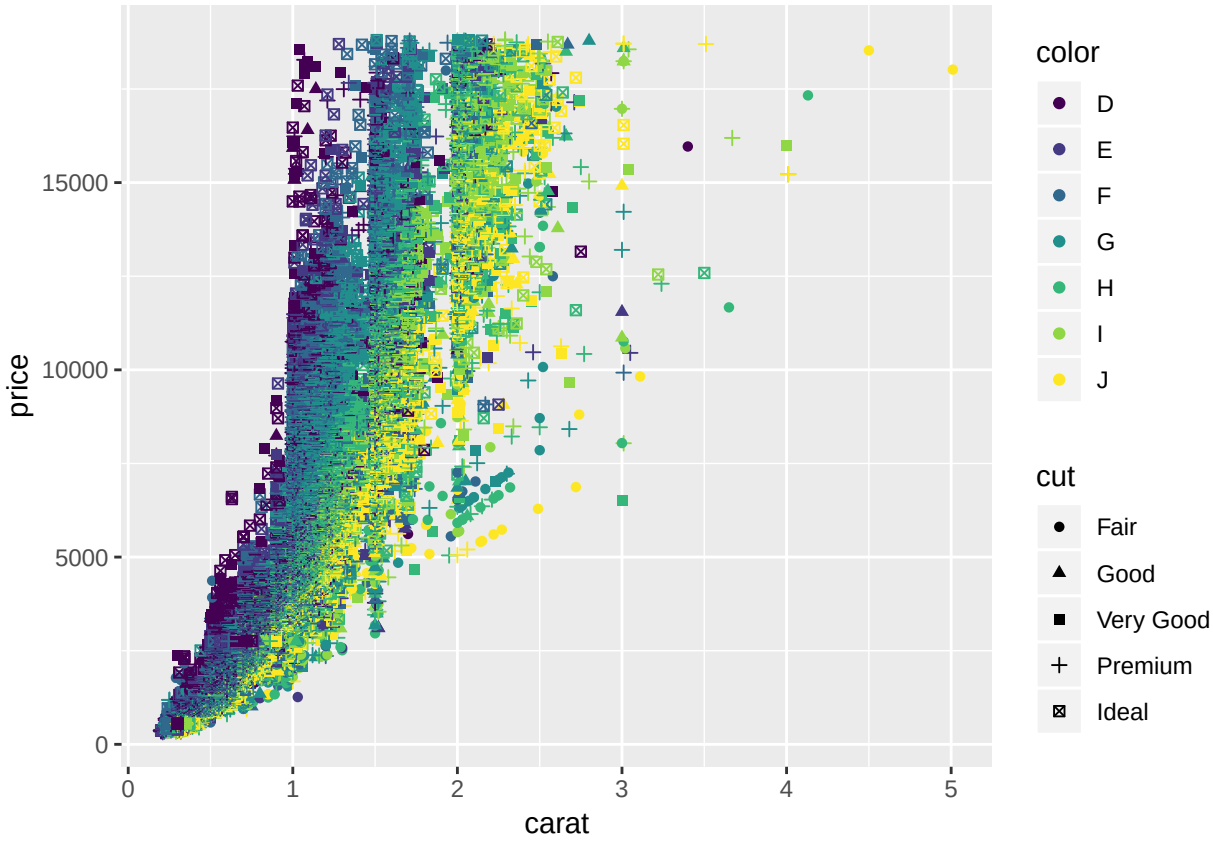
colour: Different coloring options.

geom_something: Geometric objects are the actual marks we put on a plot. Common examples: points (geom_point, for scatter plots, dot plots, etc) lines (geom_line, for time series, trend lines, etc) boxplot (geom_boxplot, for, well, boxplots!)

Example 1 - the structure

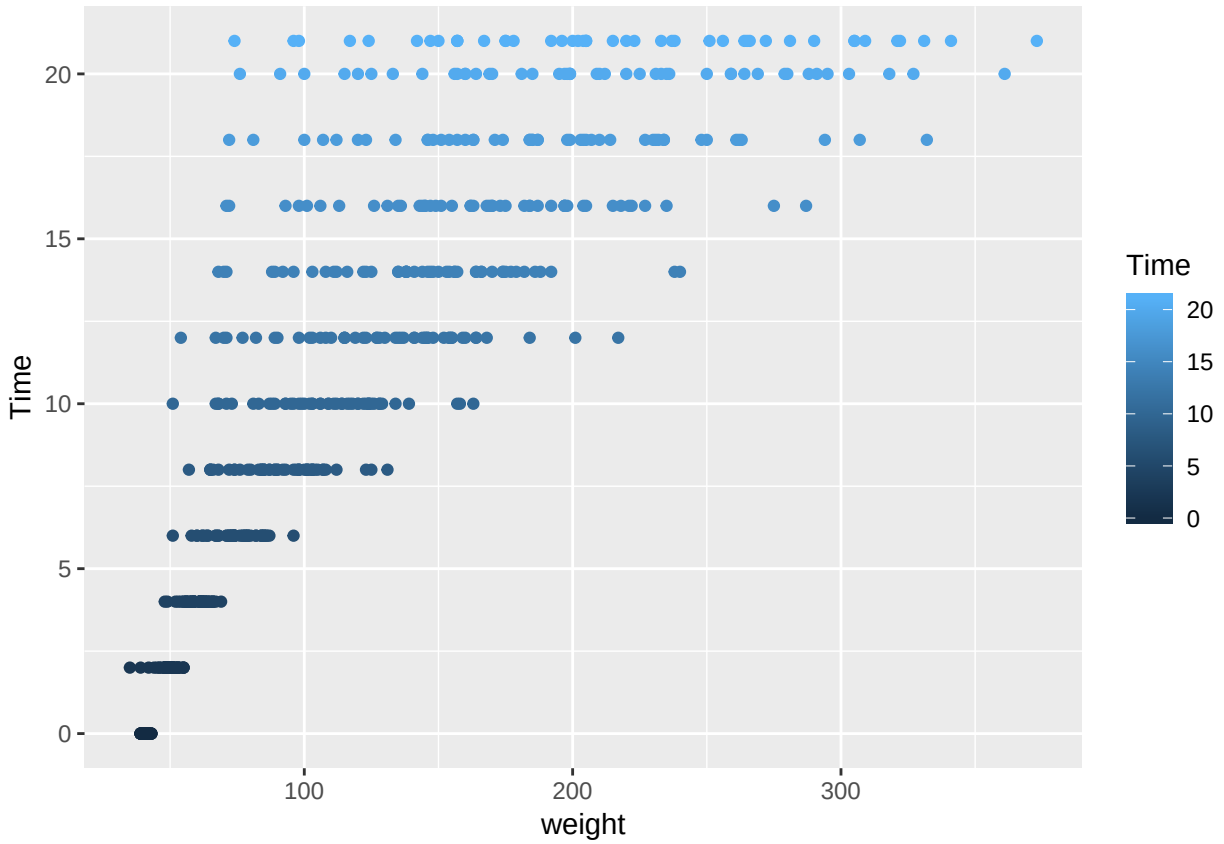
```
ggplot(diamonds, aes(carat, price, colour = color, shape = cut)) +
    geom_point()
```

```
## Warning: Using shapes for an ordinal variable is not advised
```



Example 2 - scatter plot

```
#Points
ggplot(ChickWeight, aes(weight, Time, colour = Time)) + geom_point()
```



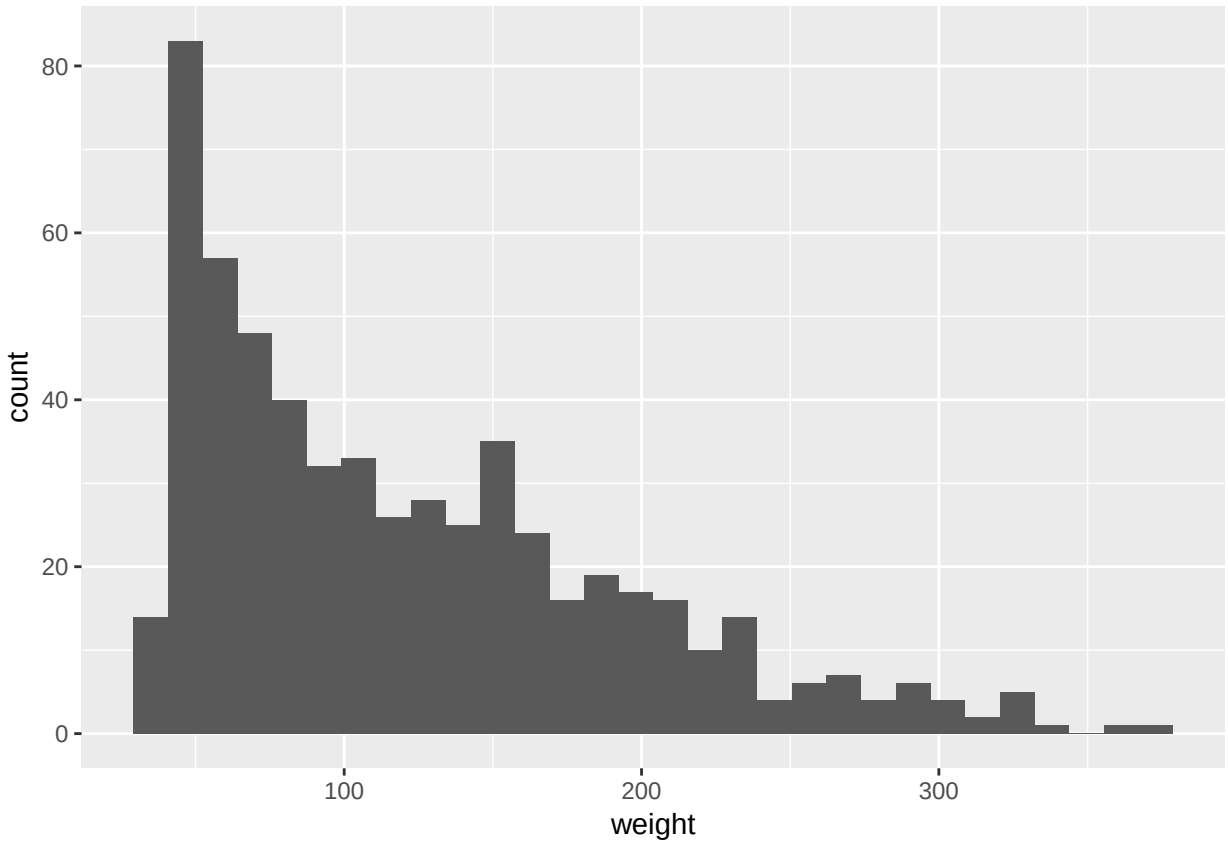
Example 3 - Histogram

```
#Histogram
names(ChickWeight)
```

```
## [1] "weight" "Time"   "Chick"  "Diet"
```

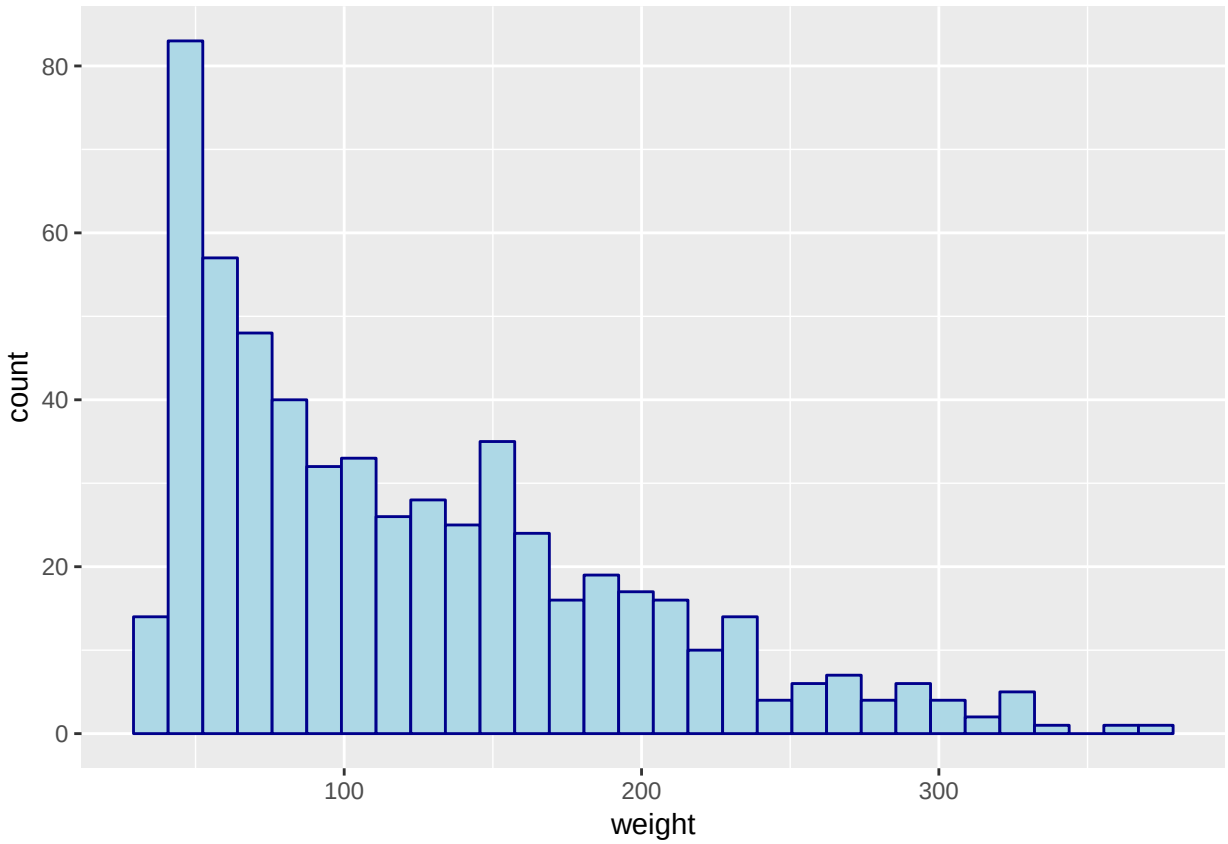
```
#Plot
ggplot(ChickWeight, aes(weight)) + geom_histogram()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
#Change colors
ggplot(ChickWeight, aes(weight)) + geom_histogram(color="darkblue", fill="lightblue")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

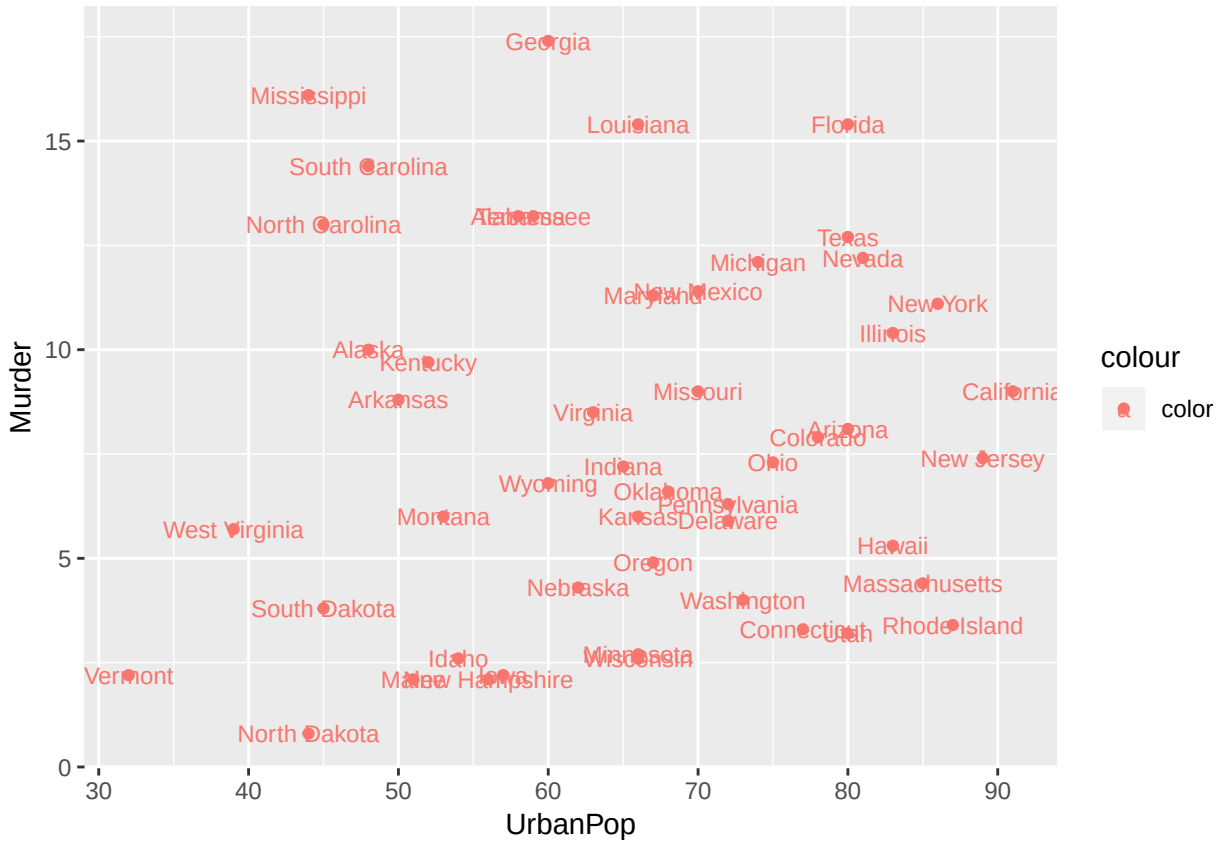


Example 4 - More on plots

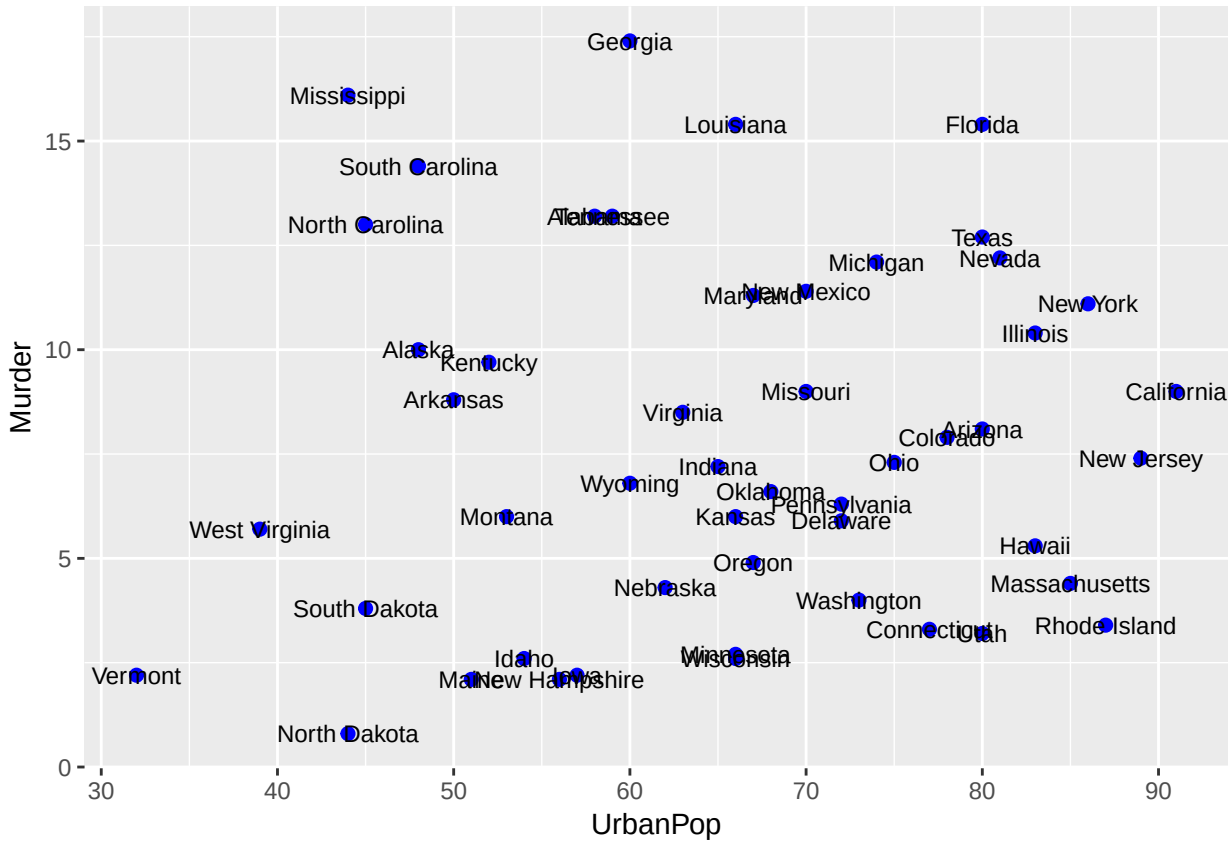
```
names(USArrests)
```

```
## [1] "Murder"   "Assault"  "UrbanPop" "Rape"
```

```
#Create plot
ggplot(USArrests, aes(UrbanPop, Murder, colour="color")) + geom_point()  + geom_text(label=rownames(USA
```

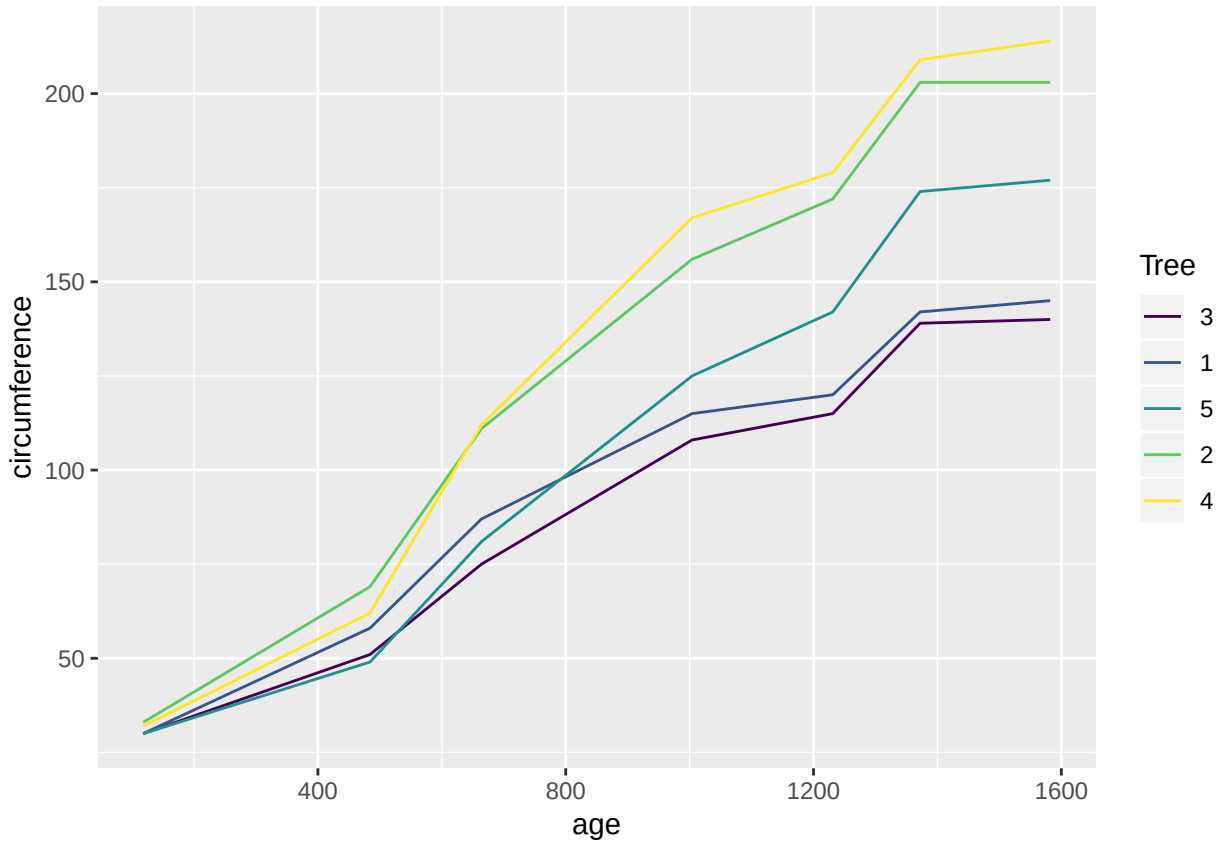



```
#Change color and size
ggplot(USArrests, aes(UrbanPop, Murder)) +
geom_point(colour="blue", size=2) + geom_text(label=rownames(USArrests), color="black", size=3)
```



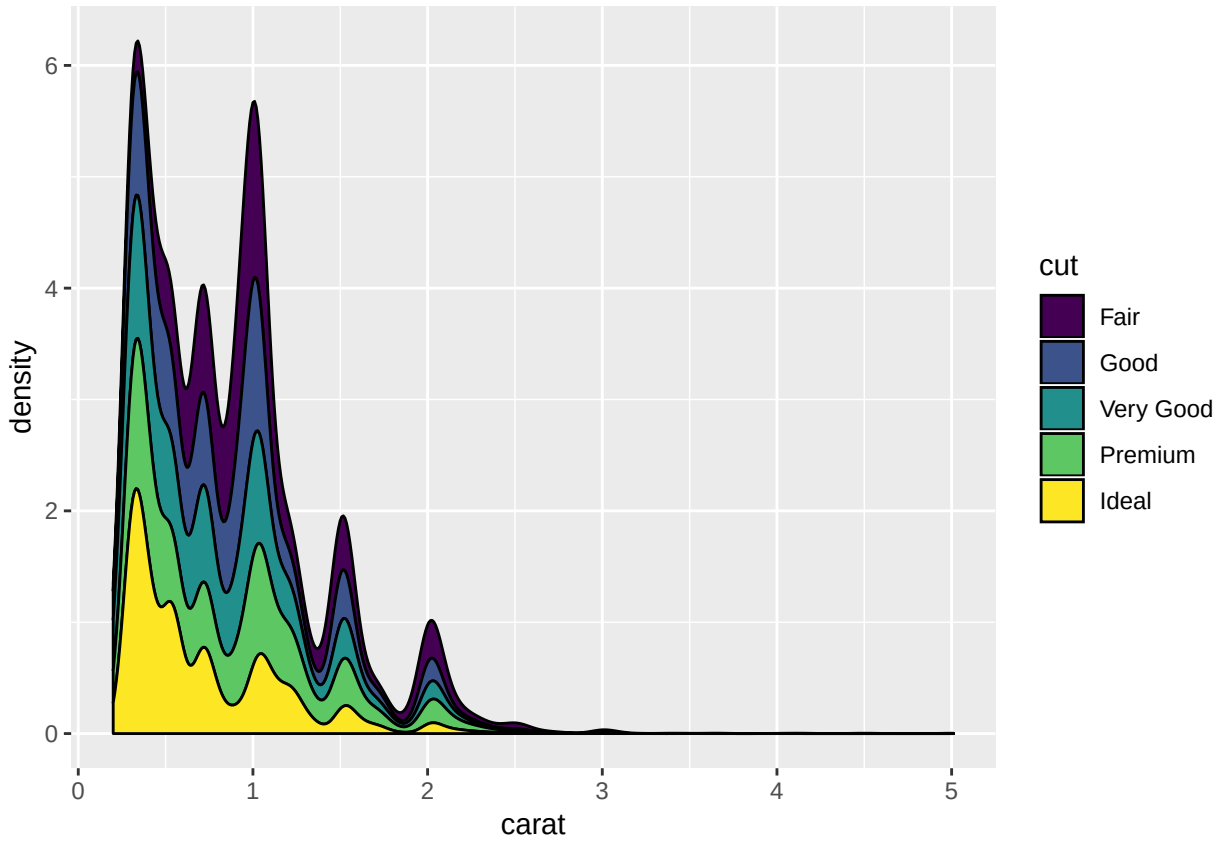
Example 5 - Lines

```
ggplot(data=Orange,
       aes(x=age, y=circumference, colour=Tree)) +
       geom_line()
```



Example 6 - Density functions

```
ggplot(diamonds, aes(carat, fill = cut)) +   geom_density(position = "stack")
```



Practice

Using the dataset "swiss", also in the package "datasets" do the followinf tasks: 1. Scatter plot: a. Plot two variables with default settings (hint: geom_point) Sample code:

```
ggplot(swiss, aes(x=name_varx , y=name_vary)) + geom_something()
```

b. Use options to modify either color or shape of points.

2. Histogram:

a. Choose one variable and make an histogram with default settings.
b. Change both the fill and the lines of the histogram.

3. Construct the following graphs using dataset ToothGrowth: on the effect of Vitamin C on Tooth Growth in Guinea Pigs.

a. A scatter with variable 'dose' on the x axis and 'len' on the y axis.
b. Color the points in the previuos plot blue.
c. Color the points in the previous plot according to 'supp'
d. Make the points bigger by setting size to 2

5. Linear Regression

5.1 Basic structure:

$$y = \beta X + \epsilon$$

Let's use the "mtcars" data set to contruct an example in which the mpg (miles per gallon) by a car is a function of horse power.

$$mpg = \beta_0 + \beta_1 * horsepower + \epsilon$$

```
linear_model1 <- lm(mpg ~ hp, data=mtcars)  # build linear regression model on full data
print(linear_model1)
```

```
##
## Call:
## lm(formula = mpg ~ hp, data = mtcars)
##
## Coefficients:
## (Intercept)           hp
##    30.09886     -0.06823
```

We can explore the estimated model with:

```
summary(linear_model1)
```

```
##
## Call:
## lm(formula = mpg ~ hp, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.7121 -2.1122 -0.8854  1.5819  8.2360
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 30.09886    1.63392  18.421  < 2e-16 ***
## hp          -0.06823    0.01012  -6.742 1.79e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.863 on 30 degrees of freedom
## Multiple R-squared:  0.6024, Adjusted R-squared:  0.5892
## F-statistic: 45.46 on 1 and 30 DF,  p-value: 1.788e-07
```

5.2 Recovering objects from model

Extracting Coefficients:

```
#Names
names(linear_model1)
```

```
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "xlevels"       "call"          "terms"         "model"
```

```r
#Display coefficients
linear_model1$coefficients
```

```
## (Intercept)          hp
## 30.09886054 -0.06822828
```

```r
#Creates object with coefficients
coeficcients_m1<-linear_model1$coefficients
coeficcients_m1
```

```
## (Intercept)          hp
## 30.09886054 -0.06822828
```

```r
#Just the intercept
coeficcients_m1[1]
```

```
## (Intercept)
##    30.09886
```

```r
#Just the hp coefficient
coeficcients_m1[2]
```

```
##          hp
## -0.06822828
```

```r
#Residuals
summary(linear_model1)[3]
```

```
## $residuals
##           Mazda RX4       Mazda RX4 Wag          Datsun 710
##         -1.59374995         -1.59374995         -0.95363068
##      Hornet 4 Drive   Hornet Sportabout             Valiant
##         -1.19374995          0.54108812         -4.83489134
##          Duster 360           Merc 240D            Merc 230
##          0.91706759         -1.46870730         -0.81717412
##            Merc 280           Merc 280C           Merc 450SE
##         -2.50678234         -3.90678234         -1.41777049
##          Merc 450SL         Merc 450SLC  Cadillac Fleetwood
##         -0.51777049         -2.61777049         -5.71206353
## Lincoln Continental   Chrysler Imperial            Fiat 128
##         -5.02978075          0.29364342          6.80420581
##         Honda Civic      Toyota Corolla       Toyota Corona
##          3.84900992          8.23597754         -1.98071757
##    Dodge Challenger         AMC Javelin          Camaro Z28
##         -4.36461883         -4.66461883         -0.08293241
##    Pontiac Firebird           Fiat X1-9       Porsche 914-2
##          1.04108812          1.70420581          2.10991276
##        Lotus Europa      Ford Pantera L        Ferrari Dino
##          8.01093488          3.71340487          1.54108812
##       Maserati Bora          Volvo 142E
##          7.75761261         -1.26197823
```

```r
summary(linear_model1)$residuals
```

```
##           Mazda RX4       Mazda RX4 Wag          Datsun 710
##         -1.59374995         -1.59374995         -0.95363068
##      Hornet 4 Drive   Hornet Sportabout             Valiant
##         -1.19374995          0.54108812         -4.83489134
```

```
##          Duster 360           Merc 240D            Merc 230
##          0.91706759         -1.46870730         -0.81717412
##            Merc 280           Merc 280C           Merc 450SE
##         -2.50678234         -3.90678234         -1.41777049
##          Merc 450SL         Merc 450SLC  Cadillac Fleetwood
##         -0.51777049         -2.61777049         -5.71206353
## Lincoln Continental   Chrysler Imperial            Fiat 128
##         -5.02978075          0.29364342          6.80420581
##         Honda Civic      Toyota Corolla       Toyota Corona
##          3.84900992          8.23597754         -1.98071757
##    Dodge Challenger         AMC Javelin          Camaro Z28
##         -4.36461883         -4.66461883         -0.08293241
##    Pontiac Firebird           Fiat X1-9       Porsche 914-2
##          1.04108812          1.70420581          2.10991276
##        Lotus Europa      Ford Pantera L        Ferrari Dino
##          8.01093488          3.71340487          1.54108812
##       Maserati Bora          Volvo 142E
##          7.75761261         -1.26197823
```

```
#R2
summary(linear_model1)$r.squared
```

```
## [1] 0.6024373
```

```
summary(linear_model1)$adj.r.squared
```

```
## [1] 0.5891853
```

If you want more information from the model:

```
#Display all estimates from model
coef(summary(linear_model1))
```

```
##                Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 30.09886054  1.6339210 18.421246 6.642736e-18
## hp          -0.06822828  0.0101193 -6.742389 1.787835e-07
```

```
#Creates an object with all estimates from model
coef_ds<-coef(summary(linear_model1)) #index this object as dataframe.


#Just the standard errors
coef_se<-coef(summary(linear_model1))[,2]
coef_se
```

```
## (Intercept)          hp
##   1.6339210   0.0101193
```

5.3 Robust Standard errors

To estimate robust standard errors you need to use the sandwich package and the lmtest package.

```
library(lmtest) #Install package before.
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
library(sandwich)  #Install package before.

coeff_rse<-coeftest(linear_model1, vcov = vcovHC(linear_model1, type="HC1")) #This is already a coeffic

#The default type in R is HCW3
coeff_rse[,2]
```

```
## (Intercept)          hp
##   2.0766149   0.0135604
```

And we can use ggplot to graph our results:

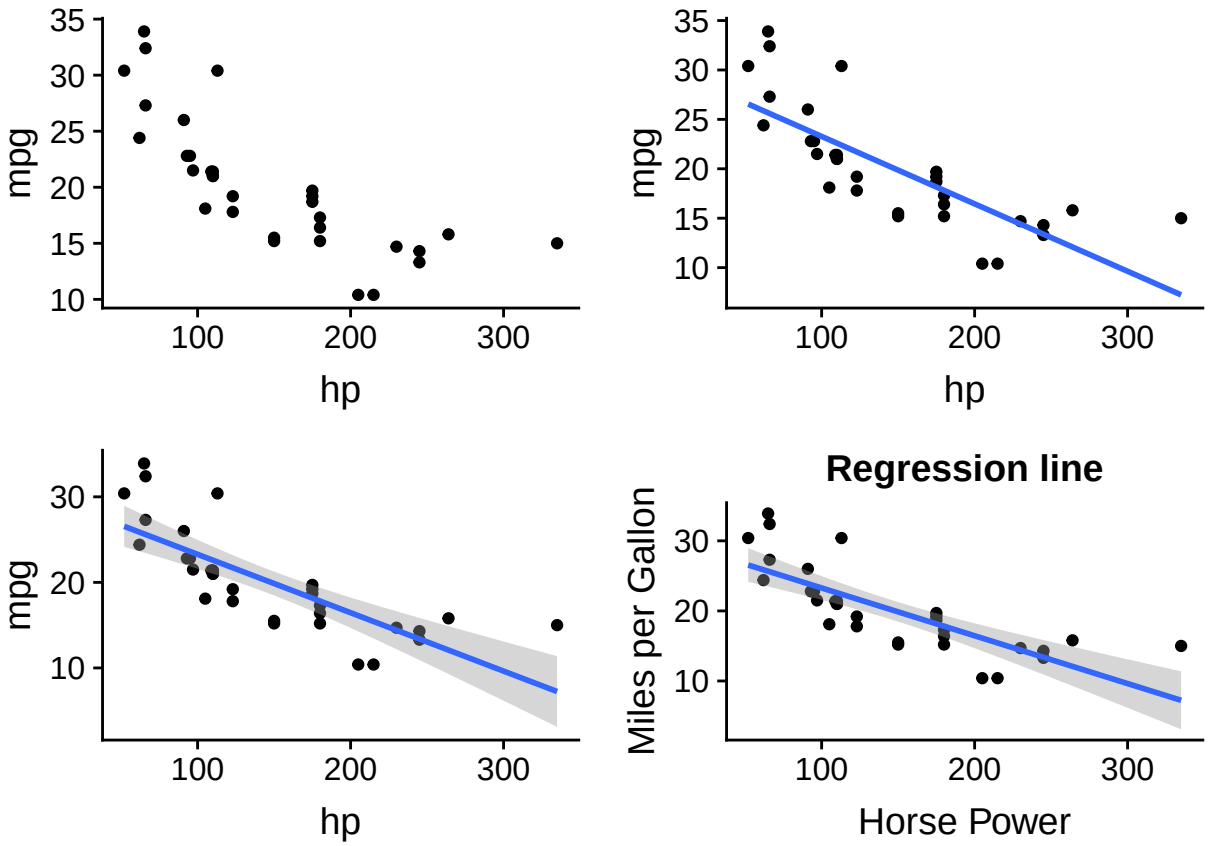
```
library(cowplot)
```

```
## Warning: package 'cowplot' was built under R version 3.5.2
```

```
##
## Attaching package: 'cowplot'
```

```
## The following object is masked from 'package:ggplot2':
##
##     ggsave
```

```
#Simple Scatter Plot
p1<-ggplot(mtcars, aes(x=hp, y=mpg)) + geom_point()

#Add linear model
p2<-ggplot(mtcars, aes(x=hp, y=mpg)) + geom_point() +geom_smooth(method="lm", se=FALSE) #try adding se

#Add linear model
p3<-ggplot(mtcars, aes(x=hp, y=mpg)) + geom_point() +geom_smooth(method="lm", se=TRUE)

p4<-ggplot(mtcars, aes(x=hp, y=mpg)) + geom_point() +geom_smooth(method="lm", se=TRUE) + labs(title="Reg


plot_grid(p1,p2,p3,p4)
```



Another way to do this with package "broom":

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

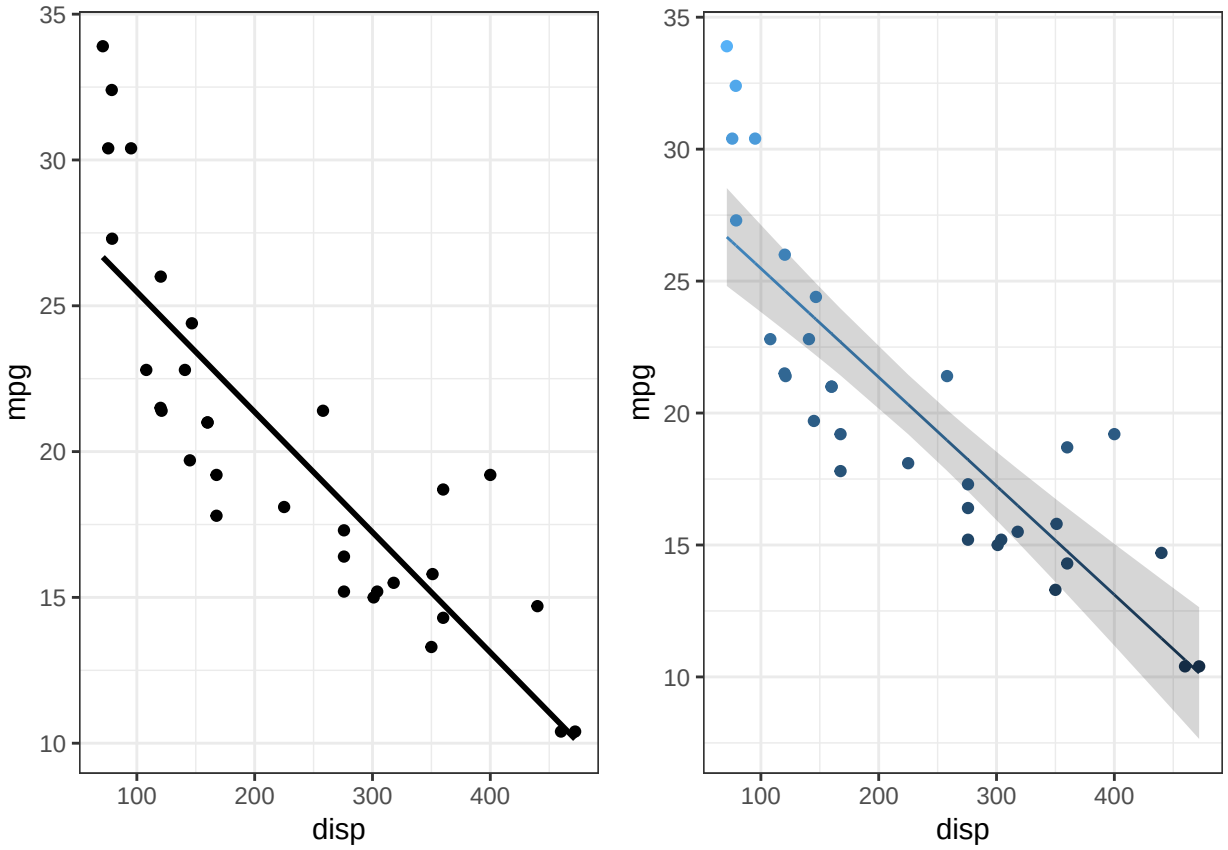
```
library(broom)

mod1<-lm(mpg~disp, data=mtcars)
print(mod1)
```

```
##
## Call:
## lm(formula = mpg ~ disp, data = mtcars)
##
## Coefficients:
## (Intercept)         disp
##    29.59985     -0.04122
```

```r
summary(mod1)
```

```
##
## Call:
## lm(formula = mpg ~ disp, data = mtcars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.8922 -2.2022 -0.9631  1.6272  7.2305
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 29.599855   1.229720  24.070  < 2e-16 ***
## disp        -0.041215   0.004712  -8.747 9.38e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.251 on 30 degrees of freedom
## Multiple R-squared:  0.7183, Adjusted R-squared:  0.709
## F-statistic: 76.51 on 1 and 30 DF,  p-value: 9.38e-10
```

```r
#And plot the fitted lines
lm1_plota<-ggplot(augment(mod1), aes(x=disp, y=mpg))+geom_point()+geom_line(aes(y=.fitted), size=1) + tl

#Using the broom output, you can do something like this for confidence intervals:
lm1_plotb<-ggplot(augment(mod1), aes(x=disp, y=mpg)) +
  geom_ribbon(aes(ymin=.fitted-1.96*.se.fit, ymax=.fitted+1.96*.se.fit), alpha=0.2) +
  geom_point(aes(colour = mpg)) +
  geom_line(aes(disp, .fitted, colour = .fitted)) +
  theme_bw() +theme(legend.position = "none")

plot_grid(lm1_plota, lm1_plotb)
```



Subsetting the data frame to estimate model:

```
library(stargazer) #Cool package to format regression output
```

```
##
## Please cite as:
##  Hlavac, Marek (2018). stargazer: Well-Formatted Regression and Summary Statistics Tables.
##  R package version 5.2.1. https://CRAN.R-project.org/package=stargazer
```

```
mod1_cyl6<-lm(mpg~disp, data=mtcars[mtcars$cyl==6,])
print(mod1_cyl6)
```

```
##
## Call:
## lm(formula = mpg ~ disp, data = mtcars[mtcars$cyl == 6, ])
##
## Coefficients:
## (Intercept)         disp
##   19.081987     0.003605
```

```
summary(mod1_cyl6)
```

```
##
## Call:
## lm(formula = mpg ~ disp, data = mtcars[mtcars$cyl == 6, ])
##
## Residuals:
##      Mazda RX4  Mazda RX4 Wag Hornet 4 Drive        Valiant       Merc 280
```

```
##        1.34119        1.34119        1.38789       -1.79314       -0.48621
##      Merc 280C   Ferrari Dino
##       -1.88621        0.09527
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 19.081987   2.913993   6.548  0.00124 **
## disp         0.003605   0.015557   0.232  0.82593
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.584 on 5 degrees of freedom
## Multiple R-squared:  0.01063,    Adjusted R-squared:  -0.1872
## F-statistic: 0.0537 on 1 and 5 DF,  p-value: 0.8259
```

```
mod1_cyl8<-lm(mpg~disp, data=mtcars[mtcars$cyl==8,])
print(mod1_cyl8)
```

```
##
## Call:
## lm(formula = mpg ~ disp, data = mtcars[mtcars$cyl == 8, ])
##
## Coefficients:
## (Intercept)         disp
##    22.03280     -0.01963
```

```
summary(mod1_cyl8)
```

```
##
## Call:
## lm(formula = mpg ~ disp, data = mtcars[mtcars$cyl == 8, ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.6011 -1.3440 -0.4768  0.6764  5.0208
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 22.032799   3.345241   6.586 2.59e-05 ***
## disp        -0.019634   0.009316  -2.108   0.0568 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.276 on 12 degrees of freedom
## Multiple R-squared:  0.2702, Adjusted R-squared:  0.2093
## F-statistic: 4.442 on 1 and 12 DF,  p-value: 0.05677
```

```
stargazer(mod1_cyl6,mod1_cyl8 )
```

```
##
## % Table created by stargazer v.5.2 by Marek Hlavac, Harvard University. E-mail: hlavac at fas.harvar
## % Date and time: Tue, Jul 02, 2019 - 11:51:03
## \begin{table}[!htbp] \centering
##   \caption{}
##   \label{}
## \begin{tabular}{@{\extracolsep{5pt}}lcc}
```

```
## \\[-1.8ex]\hline
## \hline \\[-1.8ex]
##  & \multicolumn{2}{c}{\textit{Dependent variable:}} \\
## \cline{2-3}
## \\[-1.8ex] & \multicolumn{2}{c}{mpg} \\
## \\[-1.8ex] & (1) & (2)\\
## \hline \\[-1.8ex]
##  disp & 0.004 & $-$0.020$^{*}$ \\
##   & (0.016) & (0.009) \\
##   & & \\
##  Constant & 19.082$^{***}$ & 22.033$^{***}$ \\
##   & (2.914) & (3.345) \\
##   & & \\
## \hline \\[-1.8ex]
## Observations & 7 & 14 \\
## R$^{2}$ & 0.011 & 0.270 \\
## Adjusted R$^{2}$ & $-$0.187 & 0.209 \\
## Residual Std. Error & 1.584 (df = 5) & 2.276 (df = 12) \\
## F Statistic & 0.054 (df = 1; 5) & 4.442$^{*}$ (df = 1; 12) \\
## \hline
## \hline \\[-1.8ex]
## \textit{Note:}  & \multicolumn{2}{r}{$^{*}$p$<$0.1; $^{**}$p$<$0.05; $^{***}$p$<$0.01} \\
## \end{tabular}
## \end{table}
```

And add interaction terms:

```
mtcars$am<-factor(mtcars$am)
mtcars$am<-dplyr::recode(mtcars$am, "0"="Automatic", "1"="Manual")
mod2<-lm(mpg~disp+am, data = mtcars)
mod3<-lm(mpg~disp+am+disp:am, data = mtcars)

#Plots
lm3a<-ggplot(augment(mod2), aes(x=disp, y=mpg, color=am)) +
  geom_ribbon(aes(ymin=.fitted-1.96*.se.fit, ymax=.fitted+1.96*.se.fit), alpha=0.2, linetype=0) +
  geom_point() + geom_line(aes(y=.fitted)) +
  theme_bw() +theme(legend.position = "none") + annotate("text", x=300, y=30, label= "mpg = b0 + b1*Disp


lm3b<-ggplot(augment(mod3), aes(x=disp, y=mpg, color=am)) +
  geom_ribbon(aes(ymin=.fitted-1.96*.se.fit, ymax=.fitted+1.96*.se.fit), alpha=0.2, linetype=0) +
  geom_point() + geom_line(aes(y=.fitted)) + theme_bw() +theme(legend.position = "none") + annotate("te

plot_combined<-plot_grid(lm3a, lm3b)
ggsave("plot.png")
```

```
## Saving 6.5 x 4.5 in image
```

Practice:

1. Install and load the AER and foreign packages. Load the PSID1982 data to your R environment. Furthermore, save a copy of it in .dta format to your hard drive so you can open it in Stata also and compare commands.[1]

[1] Taken from: https://www.r-exercises.com/2018/03/05/basic-r-for-stata-users-exercises/

```
library(AER)
```

```
## Loading required package: car
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
## Loading required package: survival
```

```
data("PSID1982")
names(PSID1982)
```

```
##  [1] "experience" "weeks"      "occupation" "industry"   "south"
##  [6] "smsa"       "married"    "gender"     "union"      "education"
## [11] "ethnicity"  "wage"
```

Export stata dataset

2. Now that the data is loaded in both R and Stata, print summary statistics equivalent to Stata's summarize, describe and list in 1/6.
3. Fit the following linear model and print a summary of the estimated parameters:

$$ln(wage) = \alpha + \beta_1 education + \beta_2 experience + \beta_3 experience2 + \beta_4 female + u$$

4. Add a dummy for African American and test whether the coefficients on the experience variables are jointly statistically significant from zero.
5. Translate to R: twoway scatter lwage experience.
6. Make a histogram of log(wages): hist lwage.
7. drop south from your data (frame) object.
8. Find the equivalent of mean(wage) if married == 1 & gender == 2, that is the mean wage for not married females.
9. Make a two by two frequency table: tabulate occupation union.
10. Estimate a logistic regression with married as an independent variable and education and experience as dependent variables. Estimate the marginal effect of an increase in education at the mean (margins, dydx(education) atmeans).

```
library(margins) #install package "margins" if necessary.
PSID1982$married_n<-ifelse(PSID1982$married=="yes",1,0)
model_logit<-glm(married_n ~ education + experience , data=PSID1982)
margins_logit<-margins(model_logit, atmeans=c(education))
plot(margins_logit)
```


0.020
0.015
0.010
0.005
0.000
−0.005
Average Marginal Effect
education
experience

6. Useful Translation from Stata to R

1. Drop: Example - Drop observations with x greater than 100 drop if x>100

Some options in R:

```
mydata_sub <- subset(mydata,!x>100)

mydata_sub <- mydata[mydata$x<=100]

#Tidyverse
mydata_sub <- filter(mydata, x>100)
```

2. Merge: Merge two data sets together by index variable(s) merge 1:1 index using "mydata2.dta"

Some options in R:

```
merged_data<-merge(mydata,mydata2,index)
```

3. Append: Append mydata2 to mydata

append using "mydata2.dta"

Some options in R:

```
mydata_appended <- rbind(mydata, mydata2)

#Or
library(gtools) #install if don't have it.
mydata_appended <- smartbind(mydata, mydata2)
```

4. Log files: There is not at equivalent in R.

But options:
4.1. Save the objects that you want to keep as text files or excel. 4.2. function sink.

```
# NOT RUN {
sink("sink-examp.txt")
i <- 1:10
outer(i, i, "*")
sink()
# }
# NOT RUN {

# }
# NOT RUN {
## capture all the output to a file.
zz <- file("all.Rout", open = "wt")
sink(zz)
sink(zz, type = "message")
try(log("a"))
2+2
linear_model1
## revert output back to the console -- only then access the file!
sink(type = "message")
sink()
file.show("all.Rout")
# }
```

5. Exporting "nice-looking tables"

Data frames or table-like objects into Excel:

```
write_excel_csv(x, filename)
```

Most objects can be turned into a dataframe which can be easily exported to csv or xlsx files:

```
df_coefficients<-as.data.frame(coef_ds)
write_excel_csv(df_coefficients, "Output/model1.csv") #part of package readr.
```

Other useful packages:

- xtable
- stargazer
- pander
- tables

Very useful and well explained document on package on how to create nice tables to output into Latex: https://cran.r-project.org/web/packages/tables/vignettes/tables.pdf

6. Organize Workflow and Files:
   - Projects: https://support.rstudio.com/hc/en-us/articles/200526207-Using-Projects
   - File organization: https://swcarpentry.github.io/r-novice-gapminder/02-project-intro/ https://towardsdatascience.com/how-to-keep-your-research-projects-organized-part-1-folder-structure-10bd56034d3a https://nicercode.github.io/blog/2013-05-17-organising-my-project/
7. Output graphs:
   1. Manually
   2. Script: pdf('filename.pdf') plot(yourdata) dev.off() # to complete
   3. If using ggplot: ggsave()

7. Other resources

DataCamp:

https://www.datacamp.com/courses/r-for-sas-spss-and-stata-users-r-tutorial