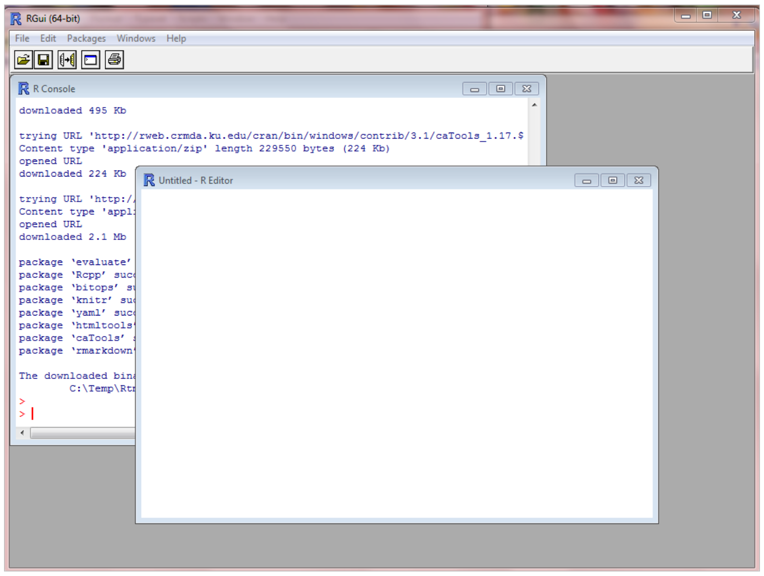
Introduction to R coding

1 What is R?

R is a free, open-source software package designed to perform statistical analysis and generate graphical output. The syntax of R programming is based on S and S-Plus. For windows based machines, you can download R at **https://www.r-project.org/** and click on CRAN on the left. Pick a mirror site to begin the download process. For a more robust version you can dowload Rstudio which is done after installing R. You can download R studio at **https://www.rstudio.com/products/rstudio2/** and download the desktop version.

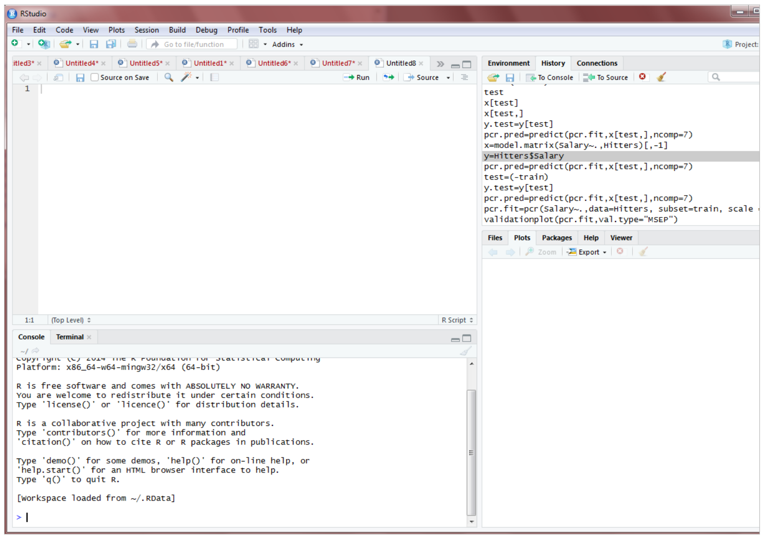
2 Getting Data into R

Let's begin our exploration of R by opening a new script window. In R you can click on file and select a new script to get a scriptwindow that looks like



```
downloaded 495 Kb

trying URL 'http://rweb.crmda.ku.edu/cran/bin/windows/contrib/3.1/caTools_1.17.$
Content type 'application/zip' length 229550 bytes (224 Kb)
opened URL
downloaded 224 Kb

trying URL 'http:/
Content type 'appl
opened URL
downloaded 2.1 Mb

package 'evaluate'
package 'Rcpp' suc
package 'bitops' s
package 'knitr' su
package 'yaml' suc
package 'htmltools'
package 'caTools' s
package 'rmarkdown'

The downloaded bin
        C:\Temp\Rtr
>
> 
```

In R studio your script window looks like



```
test
x[test]
x[test,]
y.test=y[test]
pcr.pred=predict(pcr.fit,x[test,],ncomp=7)
x=model.matrix(Salary~.,Hitters)[,-1]
y=Hitters$Salary
pcr.pred=predict(pcr.fit,x[test,],ncomp=7)
test=(-train)
y.test=y[test]
pcr.pred=predict(pcr.fit,x[test,],ncomp=7)
pcr.fit=pcr(Salary~.,data=Hitters, subset=train, scale
validationplot(pcr.fit,val.type="MSEP")
```

```
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> |
```

2.1 Data by hand

Suppose we have data on the weight of a mountain bike and it's related price.

Brand	Weight (lb)	Price ($)
FRX Raod	37	240
TX 120	35	260
RIDER A10	30	420
FRX X60	29	390
TX 480	29	440
TX 1000	28	570

2.1.1 Single Variables or Objects

Let's start with the numerical data. To enter the data for the weight of each bike we will create a vector that contains the data values. We create an object or variable name that stores the data like this

weight = c(37, 35, 30, 29, 29, 28)

In this example we create an object named "weight" that contains all of the data values of the weights of our bikes. The letter c in the command is used to indicate we are concatenating these data points. In fact when we use c() in R we can concatenate more than data points. We can concatenate two objects or variables. We can even add more data values to weight as we will see in a moment. Some R experts prefer to use the notation $< -$ in stead of $=$ to indicate we are pointing to the object name or

weight$< -$ c(37, 35, 30, 29, 29, 28)

We can also create an object or variable name for the prices. It is important to input them in the same order as their corresponding weights so we can use this later.

```
price = c(240, 260, 420, 390, 440, 570)
```

To compile this and store the data in R you can select the code you wisht to run and type ”CTRL + r” to run. In R studio there is a ”RUN” button that you can click after highlighting the code. You will notice that you get nothing on the screen. But, if you type price and then run the code, it will print out the data.

```
weight = c(37, 35, 30, 29, 29, 28)
price = c(240, 260, 420, 390, 440, 570)
price
```

To input the brand we still use the concatenate command. However, the character type variables need to be written in quotes like this

```
brand = c(”FRX Road”, ”TX 120”, ”RIDER A10”, ”FRX X60”, ”TX 480”, ”TX 1000”)
```

Suppose another data point were collected and the price of a 32 lb FRX 590 is $325. We can add this into our data like this

```
brand = c(brand,”FRX 590”)
weight = c(weight, 32)
price = c(price, 325)
```

We are concatenating the old object with the new value. If you print out brand, weight and price, you will see that this data value is added to the end of the vector.

If we wish to view a specific element of the variable we can type the variable name and in brackets type the element number like this

```
price[3]
```

This gives the price of the third bike listed in our data.

2.1.2 Data Frames

Because these three variables or objects are related, we may want to combine them into one big object called a data frame. Data frames makes a table from vectors of the same length. We invoke the command data.frame(a, b, ...,n) to combine variables a through n together. For our mountain bikes we can combine them as

```
mbike = data.frame(brand, weight,price)
```

If your print out mbike, you will see a table of our data. Again we can refer to specific elements like mbike[2,3] which refers to row two and column three of the data frame or the price of the TX 120 or mbike[2,3] = 260.

If you want to print out or use one of the variables from the data frame, you call the data frame and select a column (the variable). To select the column, you use a $ like this

```
onlyprice = mbike$price
```

2.2 Data From a File

Many times we will import data from a file. These files can come in a variety of formats like text files (.txt) that are space delimited, tab delimited or any character that is a delimiter, comma separated values (.csv), Excel files (.xlsx or .xls), SAS files (.sas7bdat), dbase files (.dat) or any other database system. We will look at the first two types .txt and .csv.

First you will need to download the file to your computer. Make sure you know the directory and folder where you saved the file. Generally files are saved to your download folder.

2.2.1 Space Delimited Text Files

If the file is a text file, you can create a data frame and use the read.table( ) command. Inside the command parentheses you will need a few argruments. The first and most important argurment is the name of the file and where it is located. Two other helpful arguments are the ”header = ” command and the ”sep = ” command. If the file has the name of the variables in the first row of the file, you want ”header = TRUE”. If not, there is no header row and ”header = FALSE”. Be sure to capitalize TRUE and FALSE. The ”sep=” command is needed if the data values on each row are separated by a special character. The default for a text file is space delimited. If the file is space delimited, then you don’t need the ”sep” argument.

Here is an example. The file lead.txt is a space delimited text file with a header row. There are many ways to open this file depending on your operating system.

Easiest method for finding file The easiest way to download a file is to include the file.chose() command in the argument of the read.table command. This will open a window where you can point and click on the file you would like to open. Let’s look a the file lead.txt which is a space delimited text file with headers. We can create a data frame and import the data into the data frame using read.table.

```
leadpoison = read.table(file.chose(),header =TRUE)
```

A file dialog box should pop up and you can navigate to the file. Click on it and R will import the data into the data frame leadpoison.

There is a lot of data in this file so rather than look at all of it, we can print out the first 6 lines of data using the head() command. If you type

```
head(leadpoision)
```

You will see the names of the columns imported using the header = TRUE command and the first six rows of data.
While this method is the easiest, it has it's drawbacks from a technical stand point.

File Path method The better method for importing data from a file is to point directly to the file by knowing the directory and path for the file. If yuo search for the file on your computer, you can find the file. If you click on properties, you find the path. Let's say my file can be found at " C:\Users\personal\download" . This is what the path would look like in Windows. Notice that backslashes are used to point to each subfolder. In UNIX or LINUX it would look like "C:/Users/personal/download" with forward slashes. I point this out because R prefers the UNIX method. So if you are using windows OS you will need to replace the backslashes with forward slashes. The syntax to create the data frame leadpoison would be

```
leadpoison = read.table("C:/Users/personal/download/lead.txt", header = TRUE)
```

We could print out the column ageyrs like this

```
leadpoison$ageyrs
```

2.2.2 Other files

If the text file is delimited by other characters than a space, you can use the following syntax

```
a = read.table("C:/file",header = TRUE, sep = " ")
```

Comma Delimited

a = read.table(”C:/file”,header = TRUE, sep = ”, ”)

Tab Delimited

a = read.table(”C:/file”,header = TRUE, sep = ” \t ”)

Comma Separted Files (.csv) a = read.csv(”C:/file”,header = TRUE)

2.2.3 Finding what is in the file

Once you have the data imported, you can explore the data. To find the names of the variables use the *name* command or

name(leadpoison)

The *head* command will give you the names of the variables and the first 6 data values

head(leadpoison)

3 Graphs

3.1 Bar Charts

With caterogical data we can create bar charts showing frequency distribution. The *barplot* command will produce a bar graph. In the lead.txt file is a variable lead_grp which uses 1 for patients with lead levels in the blood less the 40 mg $ ml in 1972 and 1973, 2 for patients who had lead levels greater than 40 mg $ ml in 1973 and 3 for patients who had lead levels greater than 40 mg $ ml only in 1972. We can see the number of patients in each category using

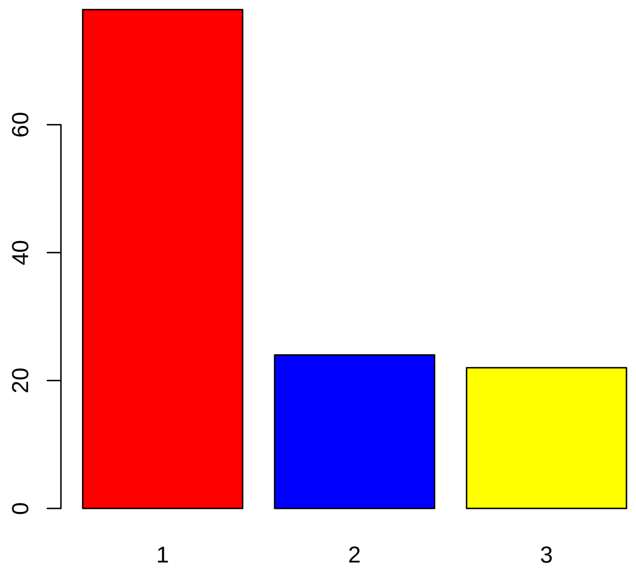
barplot(table(leadpoison$lead_grp))

The table command creates a frequency table of the categories before plotting. If you have a variable that contains the frequency. As you see, you get a gray scale graph. If you would like to add color, you can create a vector colors like this

```
barcolors = c("red","blue","yellow")
```

and create the bar graph with a the argument *col* like this

```
barplot(table(leadpoison$lead_grp),col = barcolors)
```



If the frequencies are already given, the arguments are different. Say you are studying socio-economic levels (SES) and have the following information

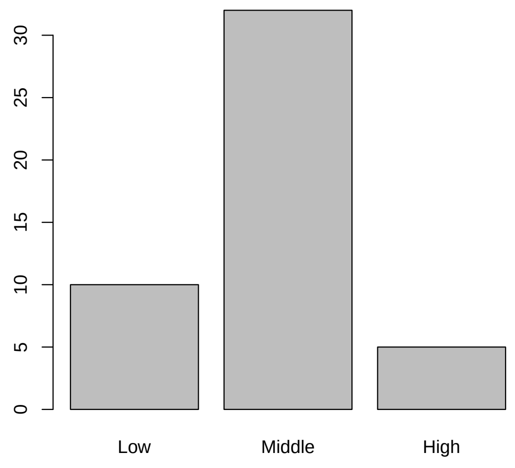
SES	Number
Low	10
Middle	32
High	5

you would first need to create two vectors for the data

```
ses = c("Low", "Middle","High")
freq = c(10,32,5)
```

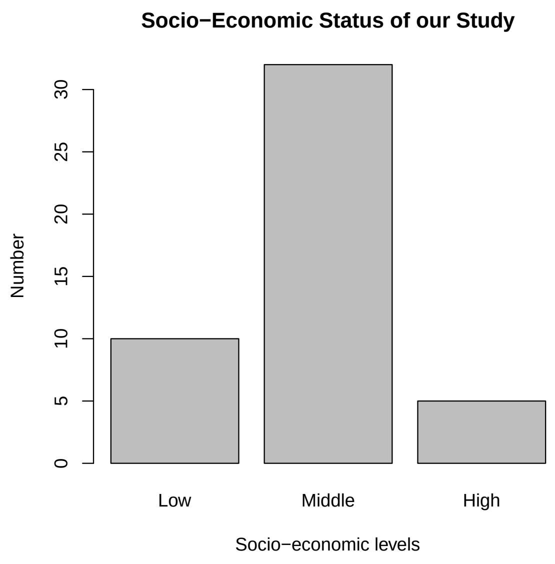
Now we make a bar chart of freq with an argument to point to the three SES levels like this

```
barplot(freq,names.arg = ses)
```



It is also helpful to include a title and label the x and y axis. We can add these to our graph like this

```
barplot(freq,names.arg = ses,main=”Socio-Economic Status of our Study”,xlab=”Socio-economic levels”,ylab=”Number”)
```

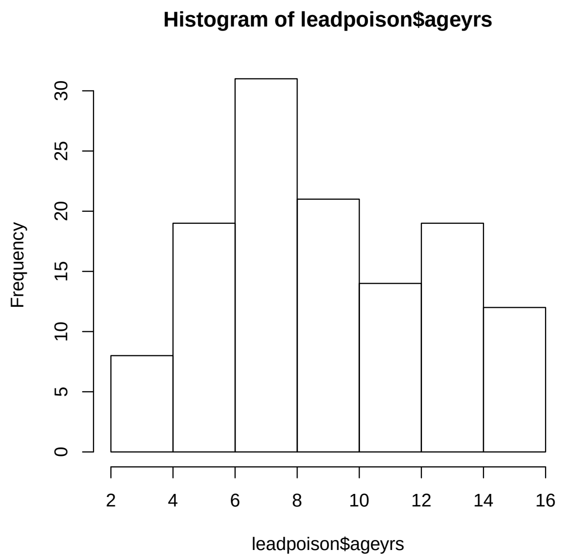


where main=”Socio-Economic Status of our Study” is the title, xlab=”Socio-economic levels” labels the x-axis and ,ylab="Number" labels the y-axis.

3.2 Histograms

The *hist* command will generate a histogram of numerical data. If we use the age in years variable in the lead poison data frame we can make a histogram

```
hist(leadpoison$ageyrs)
```

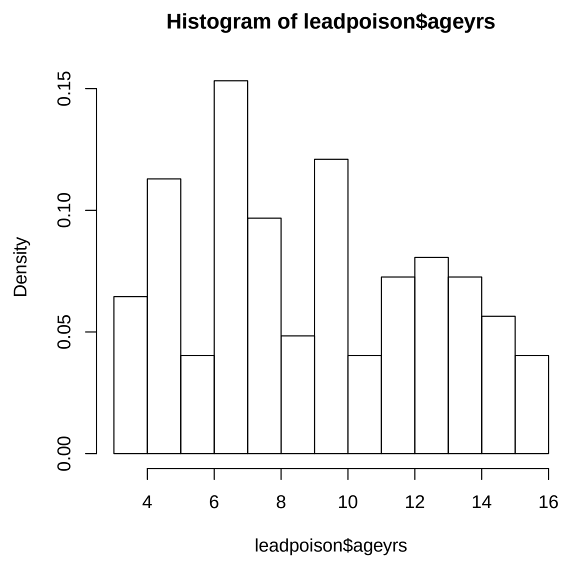


If you want the probability instead of the frequency then use

```
hist(leadpoison$ageyrs,probability = TRUE)
```

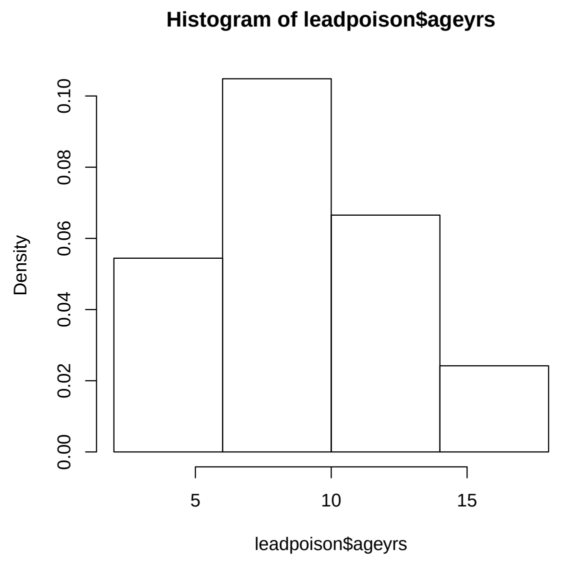
If you want to control the number of bars, you can use the *breaks* command.

```
hist(leadpoison$ageyrs,probability = TRUE,breaks = 10)
```



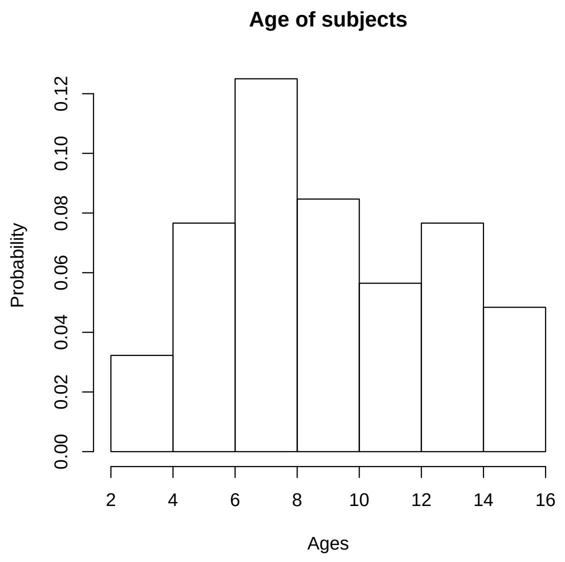
You define the breaks or bin sizes rather than R choosing the endpoints of the bins like this

hist(leadpoison$ageyrs,probability = TRUE,breaks = c(2,6,10,14,18))



And as before, you can include titles and label the x and y axis

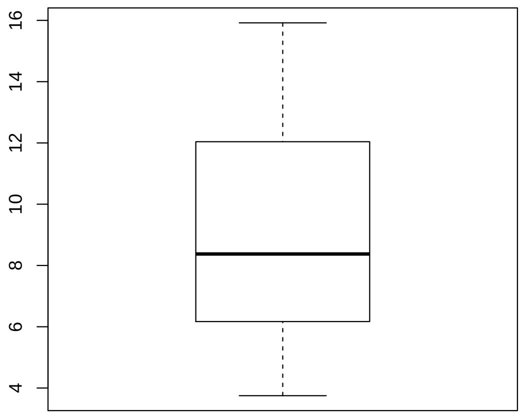
hist(leadpoison$ageyrs,probability = TRUE,main = ”Age of subjects”, xlab = ”Ages”, ylab = ”Probability”)



3.3 Box Plots

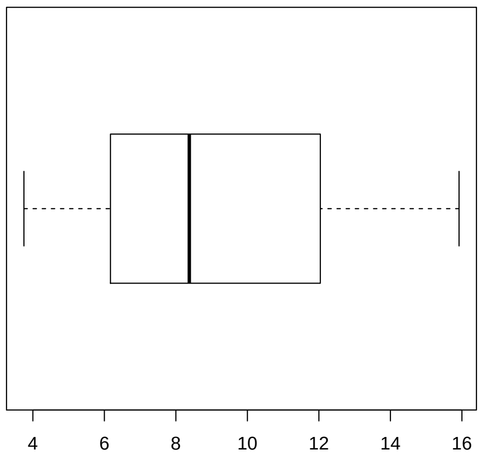
The *boxplot* command will generate a boxplot. The default in R is to make a modified box plot where outliers are accented. Again, we can make a boxplot of age in years from a lead poison data like this

```
boxplot((leadpoison$ageyrs)
```



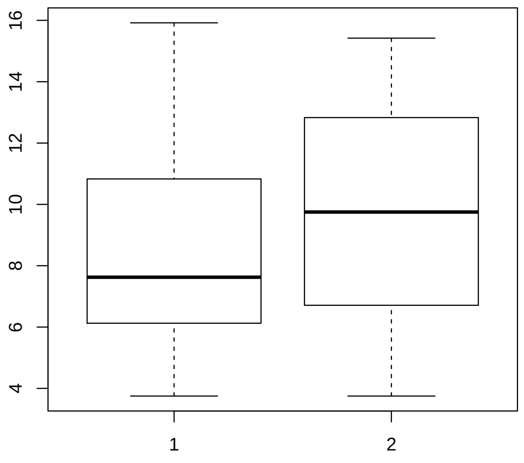
If we want the box plot to be horizontal we use

```
boxplot((leadpoison$ageyrs,horizontal = TRUE )
```



We can make side-by-side box plots of two groups as well. If we separate the ages by sex we can make two box plots on the same scale. We can separate the data frame using the *subset* command and create two new data frames.

```
agefemales = subset(leadpoison,leadpoison$sex==1)
agemales = subset(leadpoison,leadpoison$sex==2)
boxplot(agefemales$ageyrs,agemales$ageyrs)
```

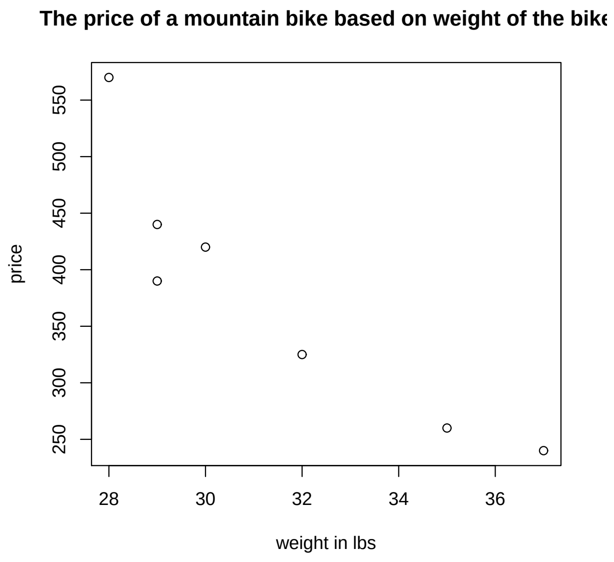


We can add titles and labels just like we did with bar graphs and histograms.

3.4 Scatter Plots

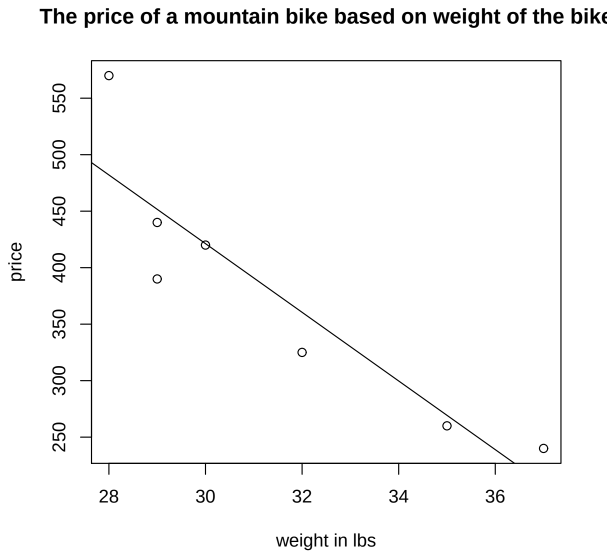
The *plot* command will make a scatter plot of two numerical data values. If we return to our mountain bike example, we can make a scatterplot of the price as a function of weight. The code below also includes titles and labels

plot(mbike$weight,mbike$price,main=”The price of a mountain bike based on weight of the bike”,xlab=”weight in lbs”,ylab= ”price”)



Notice, the independent variable (x) comes first in the command line followed by the dependent variable (y). To add a least squares regression line to the plot, we make the following command right after the scatterplot command

```
abline(lm(mbike$price~mbike$weight))
```



Notice this time the dependent variable is listed first and the independent variable is second.

4 Basic Descriptive Statistics

Now that we have some data in R, we can perform some simple statistics.

4.1 Summary Statistics

We could find the mean weight of our mountainbikes like this

```
mean(mbike$weight)
```

and the standard deviation

```
sd(mbike$weight)
```

If we want all the summary statistics we can call
summary(mbike$weight)

4.2 Probability Distributions

We can study random variables from a known distribution. We will primarily look at the normal distribution, t distribution, χ^2 distribution and the F distribution. There are four ideas we may want to investigate about the distribution. Let's look at the normal curve first. The commands for this work similarly for the other distributions. We may want to know the actual value of the density function at a point (not a typical use for us). We would type command
dnorm(x, mean = 0, sd = 1)

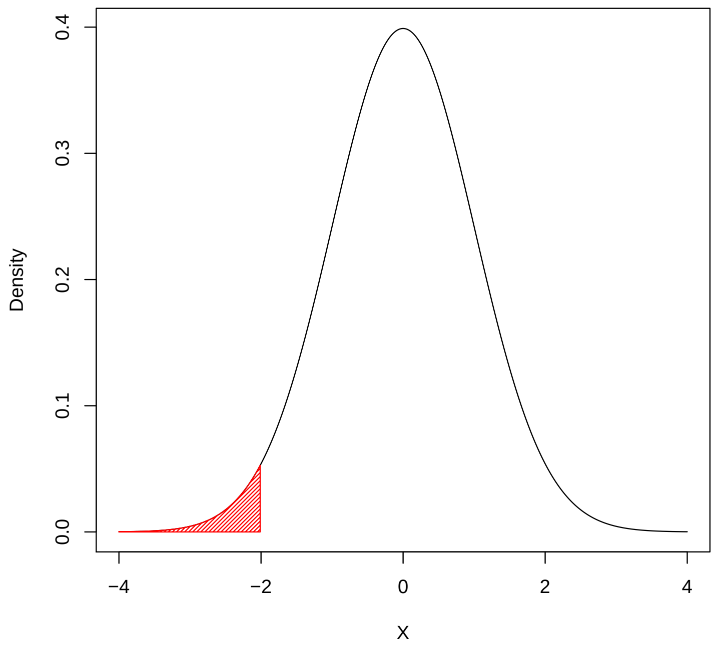
for some x value on the distribution and this will return the height of the function. We can change the mean and sd for any normal curve centered at any mean with any sd. For example

dnorm(2,mean = 2, sd = 0.5)

returns a value of 0.7979. This would be the highest value since the normal curve peaks at 2, the mean. For the other distribution and a value of x, you would type

t-distribution— dt(x,df) where df means the degrees of freedom
chi square distribution — dchisq(x,df)
F-distribution —- df(x,df1,df2)

We may want to find $P(Y < x)$ or find the area to the left of x of our distribution. This would be like finding the probablity for some z-score in a table. For a standard normal curve, the $P(z < -2)$



could be found

```
> pnorm(-2,mean = 0, sd = 1)

[1]  0.0227.
```

Of course we can recenter this for any mean and standard deviation.

We can find the same probability for the other distributions

t-distribution—$P(t < x)$ = pt(x,df)
Chi-square distribution —$P(\chi^2 < x)$ = pchisq(x,df)
F-distribution —-$P(F < x)$ = pf(x,df1,df2)

To find the inverse of this or to find the z value (or x value when not in standard normal) given a probability, we use the qnorm function. The arguments are the probability or percentile we are investigating, the mean and standard deviation or

```
qnorm(0.25, mean = 2, sd = 0.5)
```

which gives us the value for the 25th percentile of a normal curve with mean 2 and standard deviation 0.5.

The corresponding quantile command for our other distributions are

t-distribution— qt(x,df) Chi-square distribution — qchisq(x,df)
F-distribution —- qf(x,df1,df2)

The last command of interest is to randomly select n values from one of these distributions. We simply use r in from instead of d,p or q with the argument being the number of values you want to create. So the commands are

normal distribuion — rnorm(n, mean = a, sd = b)
t-distirbution — rt(n,df)
Chi-Square distribution — rchisq(n,df)
F-distribution — rf(n,df1,df2)

The syntax to create a vector of 100 values from a normal distribution with mean 10 and standard deviation of 2 is

```
rand100 = rnorm(100,mean =10, sd = 2)
```

And if want to create a histogram of these values the syntax would be

```
hist(rand100)
```

which should produce a relatively normal histogram. Increase the number of data values and you will see a really normal distribution.

4.3 Confidence Intervals

If we want to find a confidence interval, we first need the standard error. We know the average weight of our mountain bikes is 31.43. Let's look at the 95% confidence level for the weight of a mountain bike. The standard error is

```
se = sd(mbike$weight)/sqrt(length(mbike$weight))
```

where the *length* command finds the number of observations for the variable.

Now we can find the lower bound of the confidence interval using a t distribution

```
lowerbound = mean(mbike$weight) +qt(0.025, df = length(mbike$weight) - 1) * se
lowerbound
```

and the upper bound is

```
upperbound = mean(mbike$weight) +qt(0.975, df = length(mbike$weight) - 1) * se
upperbound
```

4.4 Hypothesis Testing

Recall in a hypothesis test, we have two hypothesis, the null hypothesis and the alternative hypothesis. We always assume the null hypothesis true and try to build evidence against the null hypothesis. The p-value of a hypothesis indicates the probability of getting a test statistic as larger or larger than the one derived from the data. If the probability is small, then we may think our assumption that the null hypothesis is true is incorrect. We set a threshhold value called the significance level indicated by α as the probability we are comparing against. A p-value smaller than the significance level means we reject the null hypothesis or we have sufficient evidence to believe the null hypothesis is false. A p-value greater than the significance level does not mean the null hypothesis is true or that the alternative is false. It just means we don't have enough evidence to suggest the null is false.

The null hypothesis is always compared to a null distribution. The common distributions we assume are the t distribution, the Chi-square distribution and the F distribution. Though we can compare to any distribution.

4.4.1 t test

To perform a t test in R, you can either compare the sample mean to a known mean, compare the mean of two independent samples or compare the mean of two dependent samples (paired t test).
Let's first consider a one sample t-test where a population mean is known. In our lead poison data frame, let's compare the IQ performance level to the average IQ of 100.

```
t.test(leadpoison$iqp,mu=100)
```

In your output you should get the test statistic t, the degrees of freedom and the p-value. You also get the 95% confidence interval. In this example, $t = -0.1235, df = 123, p = 0.9019$

If we want to compare two groups, we first need to separate categories. If we separate the data in the lead poison data frame by the lead level groups of those with lead levels less than 40 and those who registered a lead level greater than 40, we can compare the IQ performance of the two groups. So subset the data as we did before

```
nolead = subset(leadpoison,leadpoison$lead_grp==1)
lead = subset(leadpoison,leadpoison$lead_grp!= 1)
```

Now we can perform a t-test to compare the mean iq performance level between subjects without and with lead poison

```
t.test(nolead$iqp,lead$iqp)
```

Finally, we can do a paired t-test using the finger-wrist tapping of each hand by the subjects. The syntax to perform the t-test is

```
t.test(leadpoison$fwt_r,leadpoison$fwt_l,paired =TRUE)
```

4.4.2 ANOVA

We can perform an ANOVA on three distinct groups. Since the lead groups have three levels we can use this as our example to perform an ANOVA and again compare the iqp of each group. It is best to save the output as an object and use the *summary* command to generate the ANOVA table. In the command, the first argument is the numerical variable being compared and the second argument following the $\sim$ symbol is the categorical variable.

```
fit = aov(leadpoison$iqp ~ leadpoison$lead_grp)
summary(fit)
```

4.4.3 Chi-square test

The Chi square test is used to determine if a relationship exists between two categorical type data. In order to perform a chi-square test we first must create a table of our data. Let's say we have gender of a group and whether or not they wear corrective lenses. Suppose the data collected is

	Female	Male
Wears corrective lenses	12	17
Does not wear corrective lenses	27	23

First we input the data values into a matrix like this

```
eyes = matrix( c(12, 17 ,27, 23), ncol = 2,byrow = TRUE)
```

We could perform our chi-square test from this matrix, but it helps to label the rows and columns as the table above so we insert the names of the columns and rows

```
colnames(eyes) = c(”Female” , ”Male”)
rownames(eyes) = c(”Wears corrective lenses” , ”Does not wear corrective lenses”)
```

Now print out the table

```
eyes
```

You will see that the labels are not attached to the rows and columns. The syntax for the chi-square test of independence is

```
chisq.test(eyes)
```

4.5 Linear Regression

To compare two numerical variables, we can determine the correlation coefficient and/or perform a linear regression.

The correlation coefficient is found by the command *cor*. We can find the correlation coefficient of the price of a mountain bike and it's weight like this

```
cor(mbike$price , mbike$weight)
```

To go one step further and find the relationship between two numerical variables, we can do a linear regression. The syntax of a linear regression is very similar to the ANOVA and in fact we saw the syntax when we plotted the regression line on the scatterplot. Finding the regression line of the price of our mountain bike as a function of weight is coded as

```
fit = lm(mbike$price~mbike$weight)
summary(fit)
```

It is important to note that the dependent variable comes first in the argument line. It is like y = x or $y \sim x$. So be careful of the order.

5 Conclusion

This is an introductory guide for using R. You now have a basic understanding of R syntax and how to use predefined functions in R. The more you practice the language the better you will get at using it. Have fun and good luck!