Introduction to R

Module 1

Chris Vanlangenberg*& Scott Richter†
University of North Carolina at Greensboro

*cdvanlan@uncg.edu
†sjricht2@uncg.edu

[0]This is a basic introduction to the statistical software R, the purpose of this workshop is to provide guidance with syntax and statistical functions for new R users.

Contents

1 R Programming Basics

From: `http://www.r-project.org/`

"R is an integrated suite of software facilities for data manipulation, calculation and graphical display. It includes

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either on-screen or on hard-copy, and
- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

The term "environment" is intended to characterize it as a fully planned and coherent system, rather than an incremental accretion of very specific and inflexible tools, as is frequently the case with other data analysis software.

Many users think of R as a statistics system. We prefer to think of it of an environment within which statistical techniques are implemented. R can be extended (easily) via packages. There are about eight packages supplied with the R distribution and many more are available through the CRAN family of Internet sites covering a very wide range of modern statistics."

In this workshop, we will learn the basics of using R for statistical analysis, including

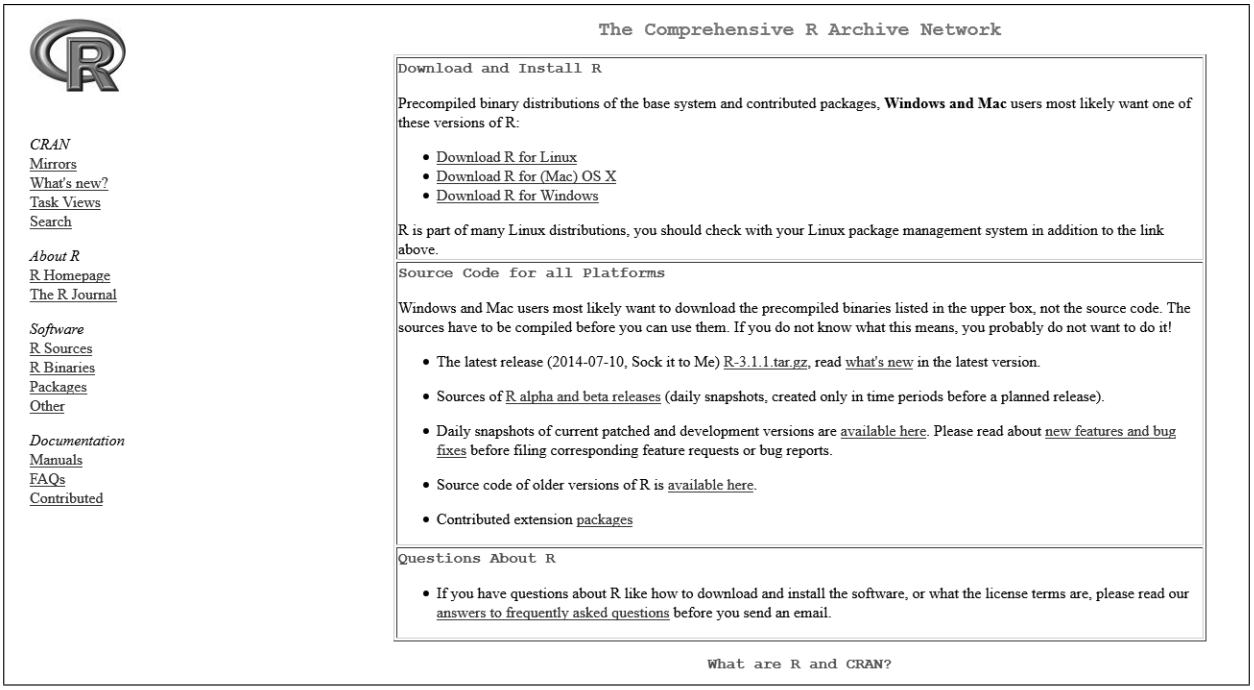
- Data file creation/acquisition
- Data manipulation
- Using supplied functions
- Simple data analyses and graphics

We will only scratch the surface!

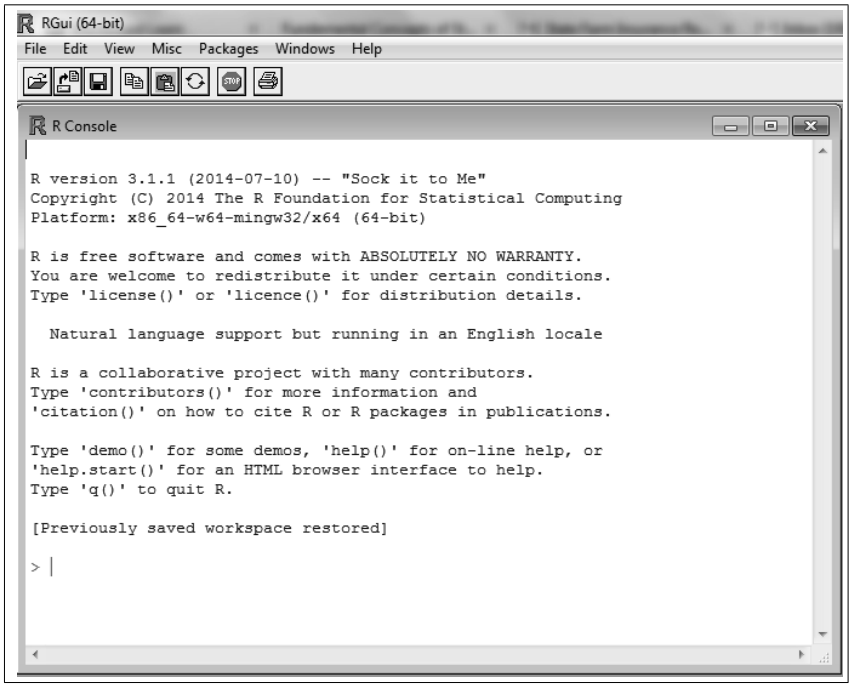
2 Setting up R

2.1 Installing R

The Base R and packages can be downloaded from Comprehensive R Archive Network (CARN:`www.cran.r-project.org`).



When you run the program, you will see the R Console:



2.2 R Coding and Syntax

Commands are entered at the cursor (next to the ">"). Unlike some other software environments that require a complete set of commands (i.e., a "program") be executed to perform a task, R runs interactively, and executes each command as it is entered. (For those used to writing programs in SAS or SPSS, for example, this can take some time getting used to!)

For example, simple calculations can be executed:

```
> 2+5
```

```
[1] 7
```

```
> log(7)
```

```
[1] 1.946
```

If the result of a calculation is to be used in further calculations, then assign the calculation to an object. Notice that the result of the calculation is not shown. However, executing the object name shows the answer.

```
> a<- 2+5
> a
```

```
[1] 7
```

```
> log.a<-log(a)
> log.a
```

```
[1] 1.946
```

Consider the following series of commands (we will discuss these in more detail later).

```
> Data = c(2,5,9,9,10,11)
> list(Data)
> mean(Data)
> sd(Data)
```

If you simply type the first command and hit enter, it seems nothing has happened, as the cursor simply moves to the next line:

```
> Data <- c(2,5,8,9,9,10,11)
```

However, a vector containing the six values inside the parentheses has been created. The next command shows the vector that was created:

```
> Data
```

```
[1]  2  5  8  9  9 10 11
```

or

```
> list(Data)
```

```
[[1]]
[1]  2  5  8  9  9 10 11
```

Next the `mean()` and `sd()` functions to compute the mean and standard deviation , which are displayed:

```
> mean(Data)
```

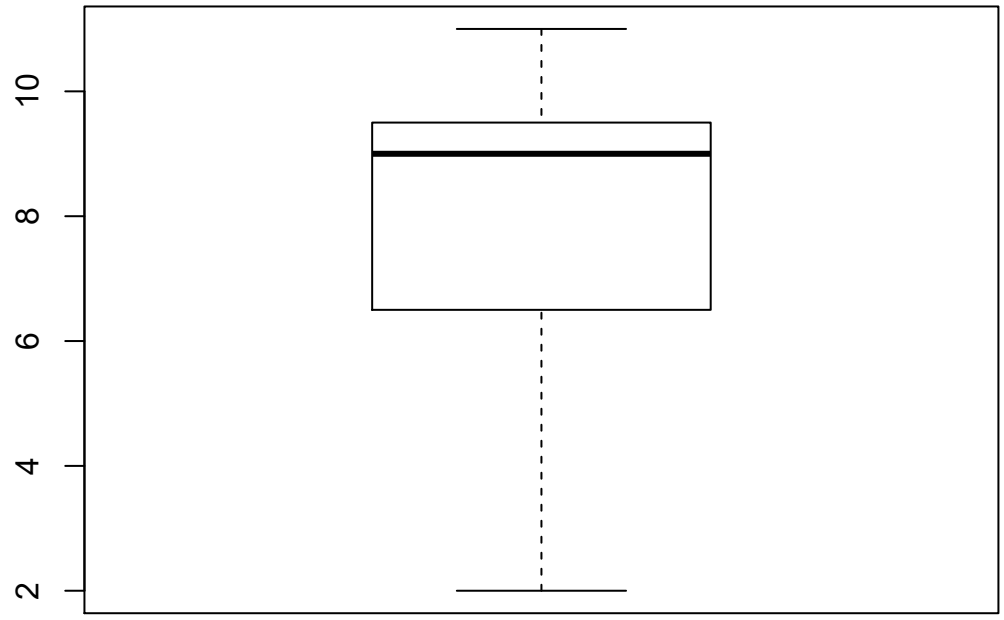
```
[1] 7.714
```

```
> sd(Data)
```

```
[1] 3.147
```

Finally, the `boxplot()` command creates a boxplot of the data, which opens in a new window:

```
> boxplot(Data)
```



Important!

Commands, object and variable names, functions and options are **case sensitive**. For example, recall that we created the object "Data" and executed the mean functions on it. Suppose we "forgot" that we capitalized the object name when we called the mean function:

```
> Data <- c(2,5,8,9,9,10,11)
> mean(data)
```

```
[1] NA
```

Instead of the mean you will get NA, because there is no such object named "data".

Similarly if you type list(data), instead of a list of the contents of the vector, we get a long, hard to decipher bunch of code that looks serious but is not particularly helpful to diagnose the cause of the problem. Thus, it bears repeating: R **is case sensitive**, it is a good idea to check for this first when things are not working as expected.

2.3 R interfaces

Because R is interactive and wants to execute commands each time the return key is hit, many users prefer to write blocks of code outside of the console, and then import or copy and paste the entire block to execute at once in the R Console. There are several options for doing this:

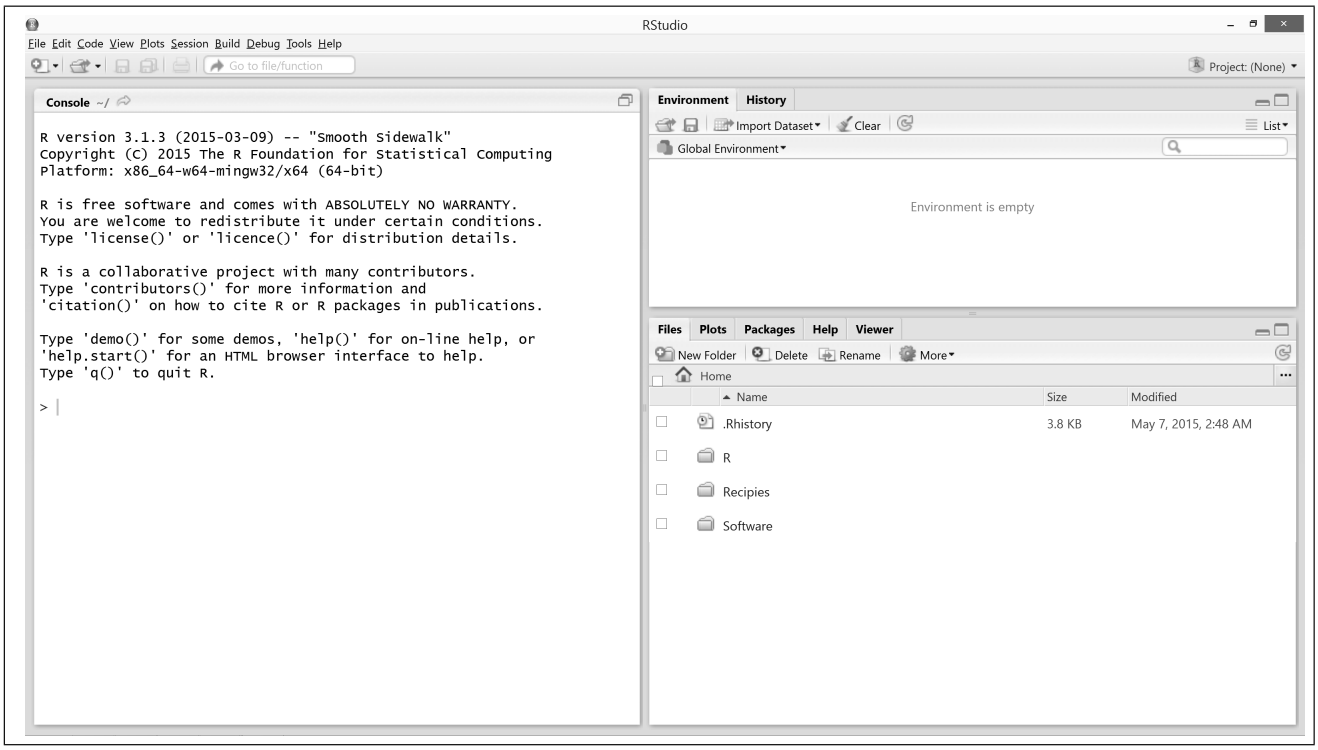
1. Text editor
   The simplest way to compose code is to use a text editor such as Notepad. The code can be saved as a file and then executed from inside the R Console.

2. R editor
   From the File menu inside RGui, chose either New Script to compose new code, or Open Script to open saved code.

3. RStudio
   RStudio is a free workspace that includes a text editor window and the R Console in the same window, and can also show graphics and results of executed commands. This may be the easiest way to use R and we will illustrate it's use in this workshop.
   Download RStudio from : `http://www.rstudio.com/products/rstudio/download/`

   Other interfaces designed for R are available, but we will not cover these in this course.

   When you run the program (RStudio), you will see:



Here the console can be used as before, but with several enhancements:

(a) Environment/History window that shows all the active objects and command history;

(b) A window with tabs that allow you to show all files and folders in your default workspace, see all plots created, list and install packages, and access help topics and documentation.

(c) Editor window from which syntax can be composed and executed (click on the upper right corner of the Console window).

2.4 Reading data into R

In most situations, data will be stored in an external file that will need to be read into R. The `read.table()` function is a general-purpose function for reading delimited text files, with variables in columns. Suppose the data of the previous examples is contained in a text file called "datafile.txt", saved in the folder "$C:\backslash Documents\backslash Rworkshop\backslash MyData$" and arranged as below, with rows corresponding to observations:

```
 2
 5
 8
 9
 9
10
11
```

Then to create the data frame, as before, use the command

```
> Data <- read.table(file = "C:/Documents/Rworkshop/MyData/datafile.txt",
+     sep = "", header = FALSE)
```

Notice that forward slashes (/) are used in R to separate folders, whereas Windows used backward slashes (\).

To view the data file:

```
> Data
```

```
  V1
1  2
2  5
3  8
4  9
5  9
6 10
7 11
```

Now we may use functions to process the data as before. In the `read.table()` function, the first information input is the specification of the file location

file="$C:/Documents/Rworkshop/MyData$"

which is required. Next are two options, separated by commas. The first, **sep**=" ", specifies the delimiter, in this case a space, and the second specifies that the first row of the file does not contain the variable name. If the first row contains the name of the variable, then add the option 'header=TRUE' (also, T≡TRUE and F≡FALSE).

Note that all letters after the equal sign must be capitalized. Many other options can be specified in the `read.table()` function (more on this later).

3 Statistical analysis using R

3.1 Working data set and results

We now consider a space-delimited data file containing several variables measured on students of an introductory statistics class.

Students in an introductory statistics class (MS212 taught by Professor John Eccleston and Dr Richard Wilson at The University of Queensland) participated in a simple experiment. The students took their own pulse rate. They were then asked to flip a coin. If the coin came up heads, they were to run in place for one minute. Otherwise they sat for one minute. Then everyone took their pulse again. The pulse rates and other physiological and lifestyle data are given in the data.

Five class groups between 1993 and 1998 participated in the experiment. The lecturer, Richard Wilson, was concerned that some students would choose the less strenuous option of sitting rather than running even if their coin came up heads, so in the years 1995-1998 a different method of random assignment was used. In these years, data forms were handed out to the class before the experiment. The forms were pre-assigned to either running or non-running and there were an equal number of each. In 1995 and 1998 not all of the forms were returned so the numbers running and sitting was still not entirely controlled.

Variable	Description
Height	Height (cm)
Weight	Weight (kg)
Age	Age (years)
Gender	Sex (1 = M, 2 = F)
Smokes	Regular smoker? (1 = yes, 2 = no)
Alcohol	Regular drinker? (1 = yes, 2 = no)
Exercise	Frequency of exercise (1 = high, 2 = moderate, 3 = low)
Ran	Whether the student ran or sat between the first and second pulse measurements (1 = ran, 2 = sat)
Pulse1	First pulse measurement (rate per minute)
Pulse2	Second pulse measurement (rate per minute)
Year	Year of class (93 - 98)

(complete description available at `http://www.statsci.org/data/oz/ms212.html`)

The `read.table()` function discussed earlier can be used to read the file, and the `head()` function to display first 5 rows of the file:

```
> Pulse <- read.table(file = "C:/Documents/Rworkshop/MyData/data.txt",
+     sep = "", header = T)
> head(Pulse, 5)
```

```
  Height Weight Age Gender Smokes Alcohol Exercise Ran Pulse1 Pulse2 Year
1    173     57  18      2      2       1        2   2     86     88   93
2    179     58  19      2      2       1        2   1     82    150   93
3    167     62  18      2      2       1        1   1     96    176   93
4    195     84  18      1      2       1        1   2     71     73   93
5    173     64  18      2      2       1        3   2     90     88   93
```

Another common type of delimited file is a comma-separated (csv) file. If the previous data file had been saved as a csv file, the `read.table()` function can be modified as

```
> Pulse <- read.table(file = "C:/Documents/Rworkshop/MyData/data.csv",
+     sep = ",", header = T)
```

or you may use the function `read.csv()` to read the data into R as

```
> Pulse<-read.csv(file="C:/Documents/Rworkshop/MyData/data.csv", header=T)
```

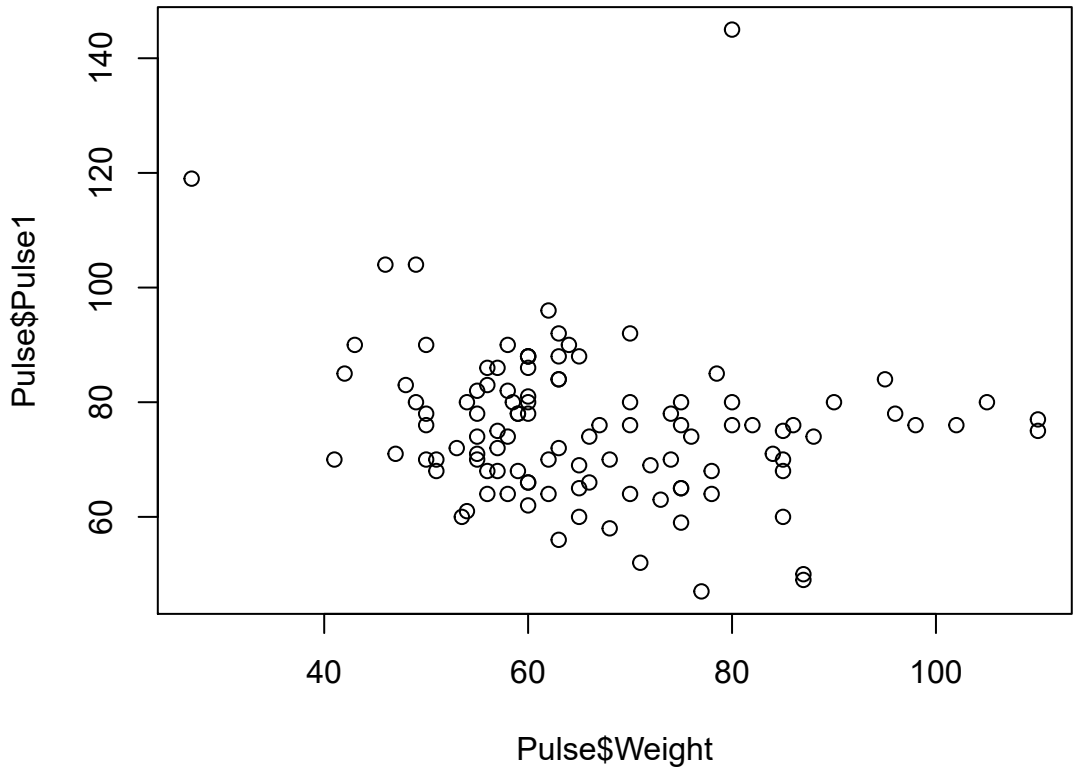
We will use these data to answer several research questions, completing several analysis tasks and illustrating many R concepts along the way.

3.2 Research questions

1. How does Pulse1 (the first pulse measurement) depend on lifestyle and physiological measurements? Are frequent exercisers fitter?

   Explore the relationship between Pulse1 and Weight.

   (a) Scatterplot of Pulse1 by Weight. "The plot suggests a weak negative linear relation between weight and the first pulse reading. There is also an outlying observation that could affect the quantitative assessments of the association."

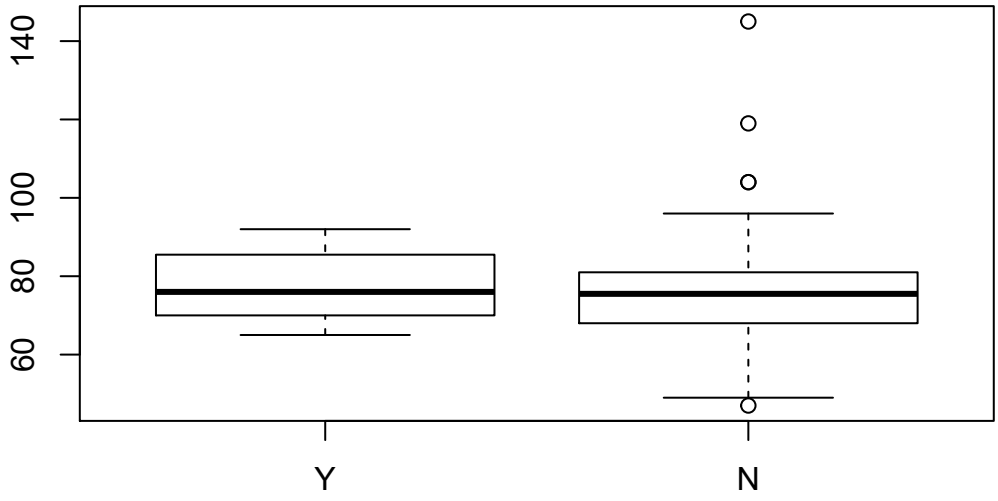


   (b) **Simple linear regression**. "Each additional pound of weight was associated with a 0.17 beat per minute decrease in pulse"

   (c) **Pearson correlation**. "The Pearson correlation between Pulse1 and Weight was r = -0.195, which was statistically significant at the 0.05 level of significance t(df=107) = -2.05, p = 0.043)."

   Explore the relationship between Pulse1 and smoking status.

(a) **Boxplot** of Pulse1 by smoking status: "The boxplots suggest that there is little difference between typical first pulse measurements of smokers and nonsmokers, but that there is more variability among nonsmokers."



(b) **Descriptives**. "The mean pulse rate for smokers was 77.55 bpm and for nonsmokers 75.48 bpm.

(c) **t-test**. The mean difference of 2.07 was not statistically significant t(df=107) = 0.49, p = 0.314)"

2. Is there evidence that some students didn't run even though their coin toss came up heads?

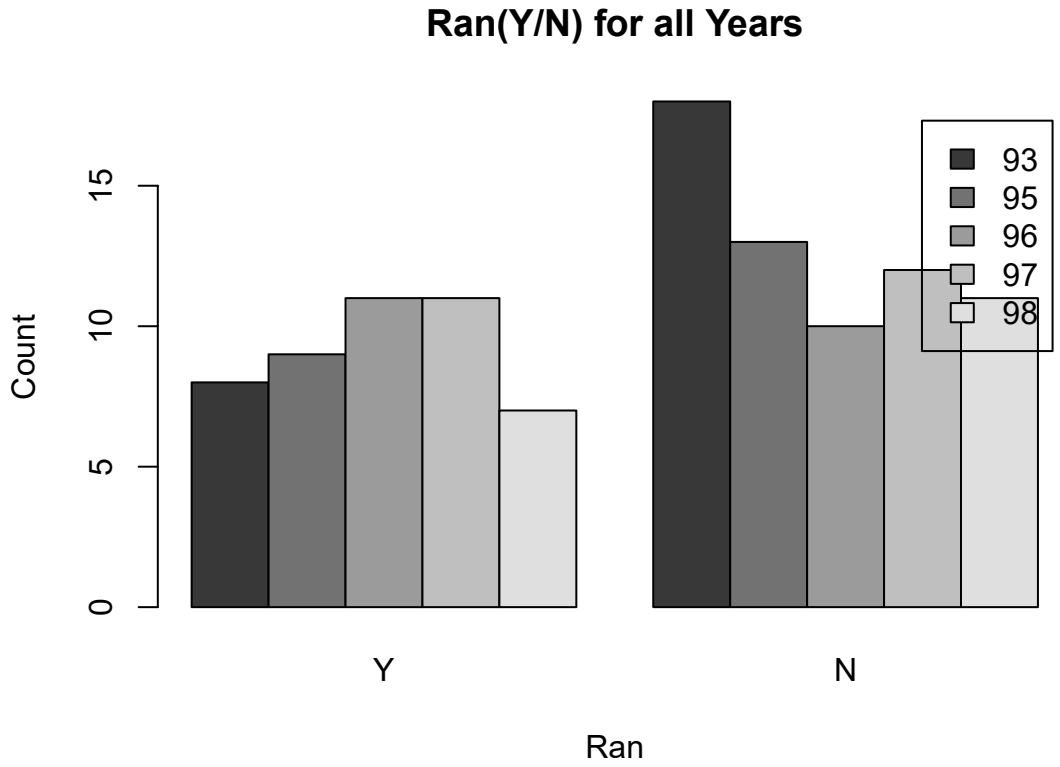
Is there evidence that fewer than 50% would be selected to run?

(a) **Frequencies and proportions**. "Overall, 41.8% of all students ran between pulse readings.

(b) **Test for proportion**. Assuming this groups of students can be considered a random sample from all similar statistics students, this was moderate, but not convincing statistical evidence that fewer than 50% of all students would be selected to run ($\chi_{(1)}$=2.63, p = 0.053), and thus not convincing statistical evidence that students lied about the result of the coin toss."

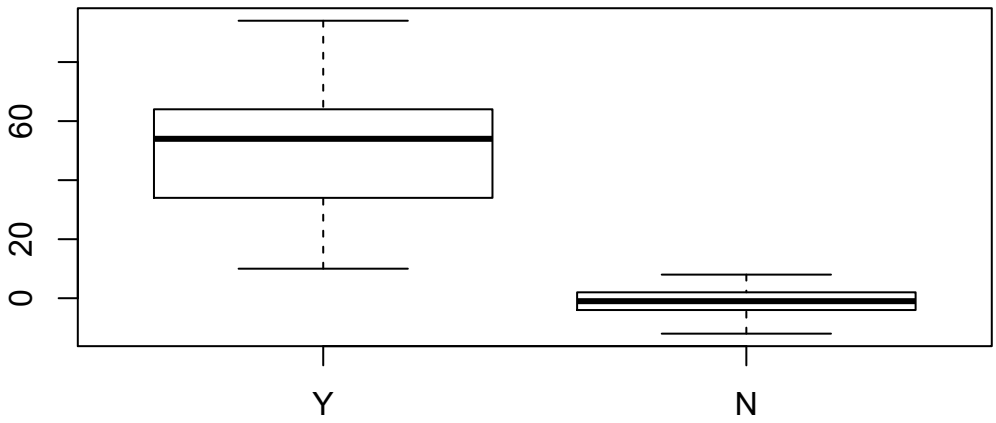
Does the proportion who ran between measurements depend on year?

(c) **Crosstabs, bar charts, chi-squared test**. "The bar charts below show a tendency for fewer students to run (except for 96), but the discrepancy is greatest for 93. When comparing 1993 to 1995-1998, 30.8% of students in 1993 admitted to tossing heads compared to 45.2% in 1995-1998. However, this proportion difference was not statistically significant ($\chi_{(1)}$=1.17 , p = 0.280)."



3. To what extent does the paired comparison (difference) between Pulse1 and Pulse2 depend on Ran?

   (a) **Boxplots**

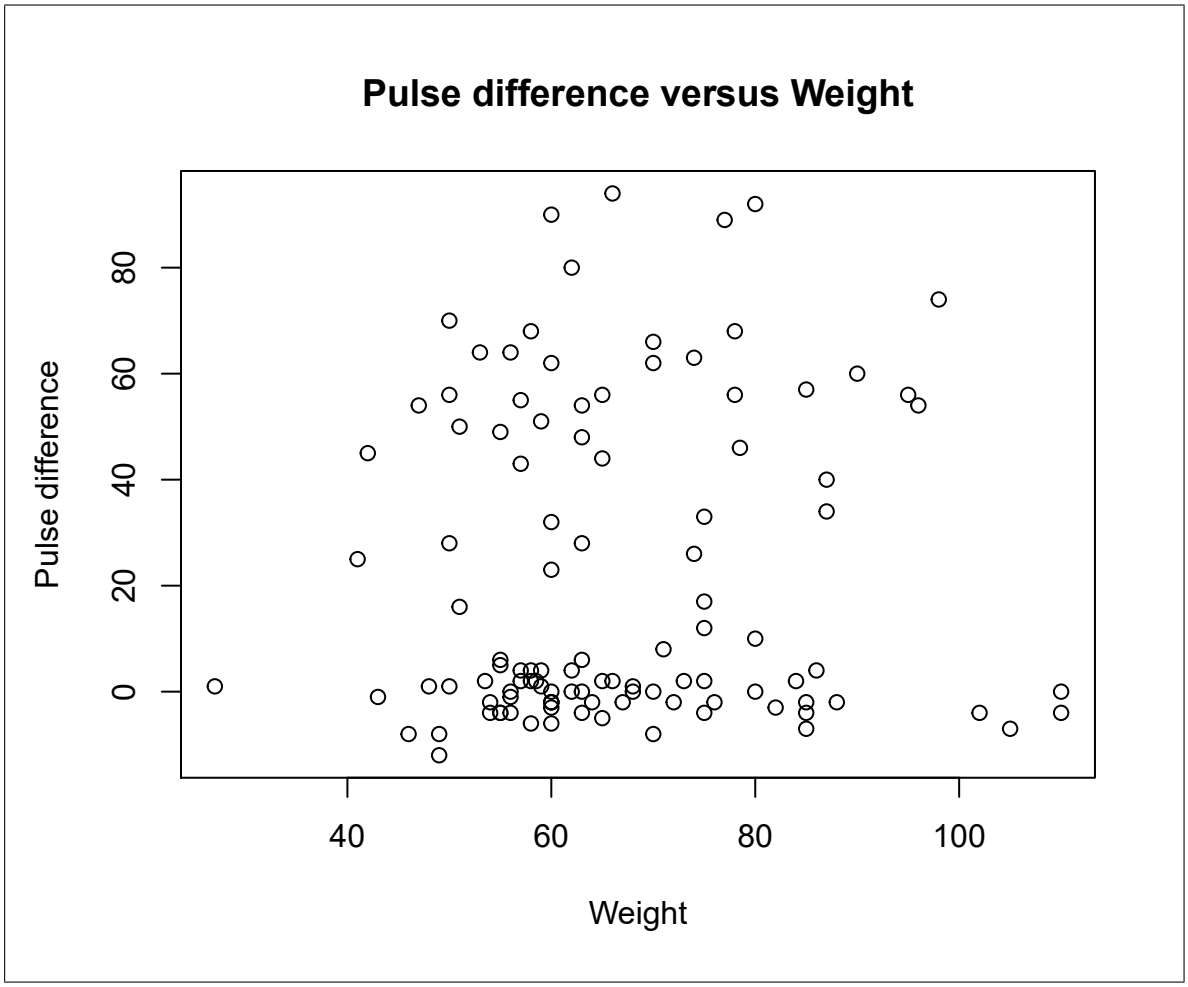


(b) **Means, t-test**. As can be seen from the boxplots, there very little change between pulse readings for those who sat in between (mean = -1.00), while there was a much greater mean difference and greater variability among those who ran (mean = 51.39). This different was statistically significant t(df=47) = 16.64, p < 0.001).

4. How does the Pulse2-Pulse1 difference depend on lifestyle and physiological measurements? Do frequent exercisers run more energetically?

Explore the relationship between Pulse difference and Weight.

(a) **Scatterplot** of Pulse.diff by Weight. "The plot does not suggest a relationship."

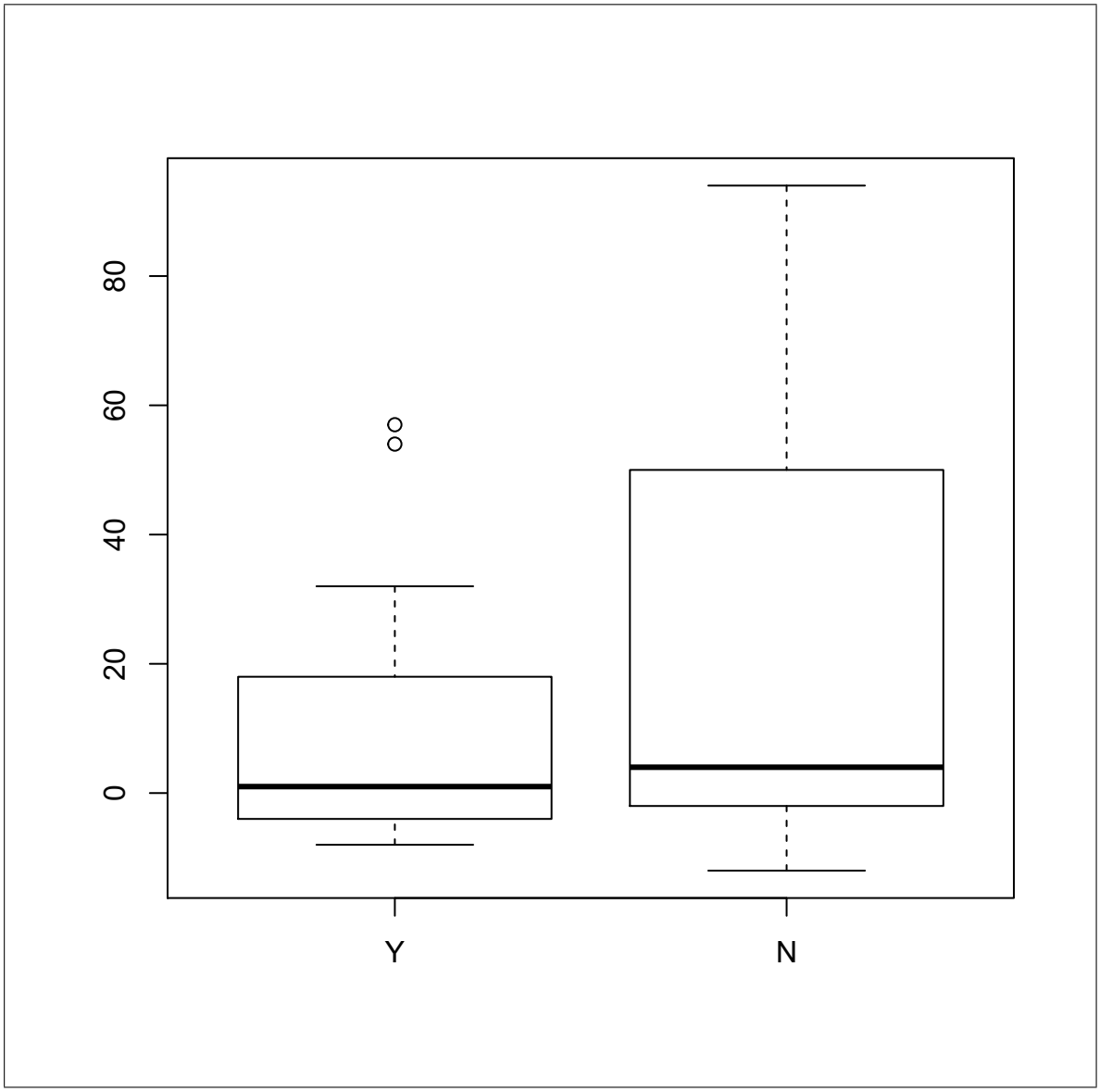


(b) **Pearson correlation**. "The Pearson correlation between Pulse.diff and Weight was r = 0.056, which was not statistically significant at the 0.05 level of significance t(df=107) = 0.59, p = 0.560)."

(c) **Simple linear regression**. "Each additional pound of weight was associated with a 0.11 beat per minute decrease in pulse difference"

Explore the relationship between Pulse difference and smoking status.

(a) **Boxplots** of Pulse difference by smoking status:



(b) **Descriptives**. "The mean difference in pulse rate for smokers was 11.45 bpm and for nonsmokers 22.19 bpm.

(c) **t-test**. The mean difference of 10.74 was not statistically significant t(df=107) = -1.36, p = 0.197)"

3.3 Using R to generate the results

Before we begin addressing the computational details needed to obtain the results, it is very important to check the data types of the variables, as sometimes the data type may be incorrect (e.g., a numerical variable may be read as a character). Thus we start by getting a summary of the variables and their types using the `str()` function:

```
> str(Pulse)
```

```
'data.frame':        110 obs. of  11 variables:
 $ Height  : int  173 179 167 195 173 184 162 169 164 168 ...
 $ Weight  : num  57 58 62 84 64 74 57 55 56 60 ...
 $ Age     : int  18 19 18 18 18 22 20 18 19 23 ...
 $ Gender  : chr  "F" "F" "F" "M" ...
 $ Smokes  : int  2 2 2 2 2 2 2 2 2 2 ...
 $ Alcohol : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Exercise: int  2 2 1 1 3 3 2 2 1 2 ...
 $ Ran     : int  2 1 1 2 2 1 2 2 2 1 ...
 $ Pulse1  : int  86 82 96 71 90 78 68 71 68 88 ...
 $ Pulse2  : int  88 150 176 73 88 141 72 77 68 150 ...
 $ Year    : int  93 93 93 93 93 93 93 93 93 93 ...
```

Notice that the name of each column, along with the type of data and the first 10 observations are given. The variable Gender is listed as a "factor", which is what R calls a nominal level variable. The rest are considered numeric.

The `summary()` function, when applied to a data frame, will compute the mean and 5-number summary for numeric variables and well as frequencies for factors:

```
> summary(Pulse)
```

```
     Height          Weight            Age          Gender               Smokes
 Min.   : 68   Min.   : 27.0   Min.   :18.0   Length:110         Min.   :1.0
 1st Qu.:165   1st Qu.: 56.2   1st Qu.:19.0   Class :character   1st Qu.:2.0
 Median :172   Median : 63.0   Median :20.0   Mode  :character   Median :2.0
 Mean   :172   Mean   : 66.3   Mean   :20.6                      Mean   :1.9
 3rd Qu.:180   3rd Qu.: 75.0   3rd Qu.:21.0                      3rd Qu.:2.0
 Max.   :195   Max.   :110.0   Max.   :45.0                      Max.   :2.0

    Alcohol         Exercise          Ran           Pulse1          Pulse2
 Min.   :1.00   Min.   :1.00   Min.   :1.00   Min.   : 47.0   Min.   : 56.0
 1st Qu.:1.00   1st Qu.:2.00   1st Qu.:1.00   1st Qu.: 68.0   1st Qu.: 72.0
 Median :1.00   Median :2.00   Median :2.00   Median : 76.0   Median : 84.0
 Mean   :1.38   Mean   :2.21   Mean   :1.58   Mean   : 75.7   Mean   : 96.8
 3rd Qu.:2.00   3rd Qu.:3.00   3rd Qu.:2.00   3rd Qu.: 82.0   3rd Qu.:125.0
 Max.   :2.00   Max.   :3.00   Max.   :2.00   Max.   :145.0   Max.   :176.0
                                              NA's   :1       NA's   :1
      Year
 Min.   :93.0
 1st Qu.:95.0
 Median :96.0
 Mean   :95.6
 3rd Qu.:97.0
 Max.   :98.0
```

Notice that there are several variables treated as numeric that are actually categorical variables, such as "Smokes", which is an indicator for whether or not the individual smokes (and value of 1 indicates the person smoked). Since it does not make sense to compute numerical summaries on such a variable, we will need to let R know to treat these as nominal when necessary. We can also select individual variables to summarize. The code below chooses only the variable Weight to summarize:

```
> summary(Pulse$Weight)
```

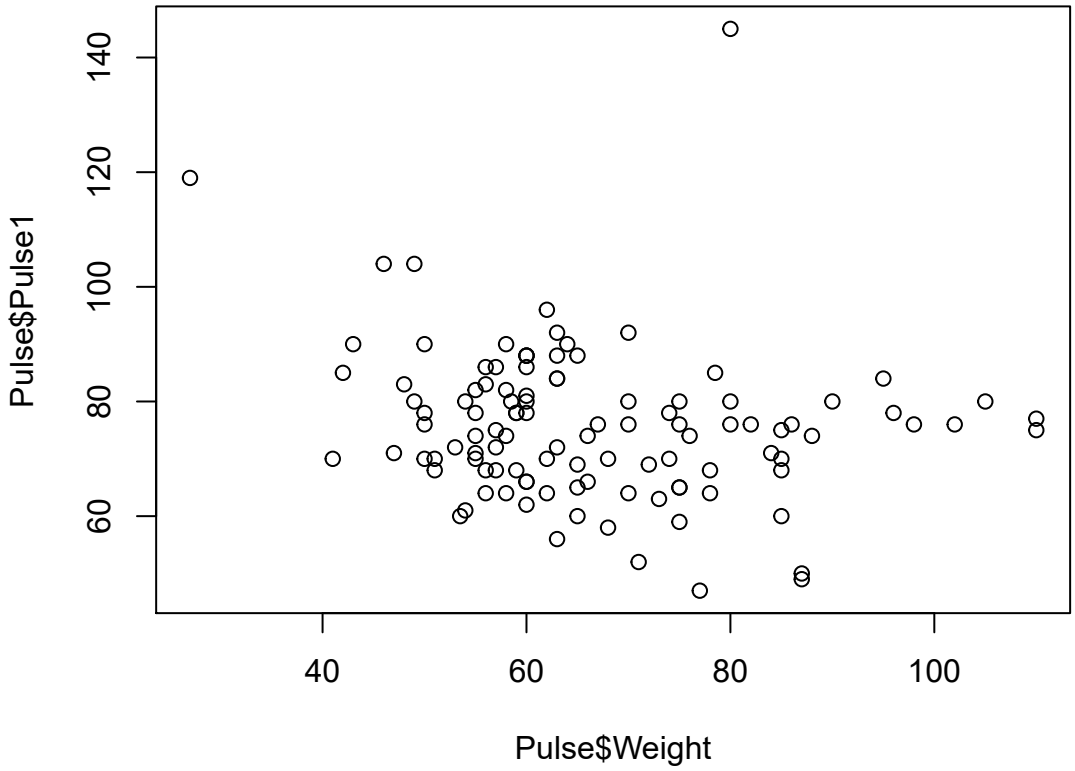
```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   27.0    56.2    63.0    66.3    75.0   110.0
```

1. How does Pulse1 (the first pulse measurement) depend on the lifestyle and physiological measurements? Are frequent exercisers fitter?

   Explore the relationship between Pulse1 and Weight.

   (a) **Scatterplot** of Pulse1 by Weight. The `plot()` function can be used. Two different ways of specifying the variables can be employed: `plot(x,y)` or `plot(y~x)` (The latter is called a function argument). Thus, to obtain the scatterplot, use either:

```
> plot(Pulse$Weight, Pulse$Pulse1)
```

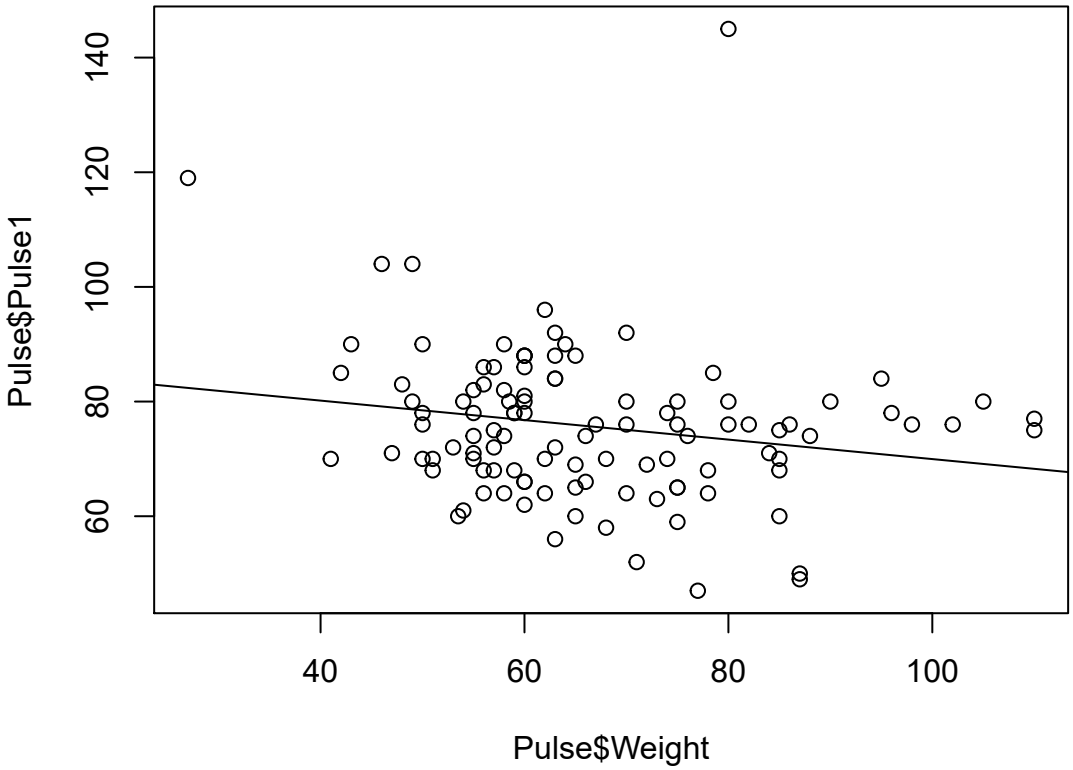


   or

```
> plot(Pulse$Pulse1~Pulse$Weight)
```

   To help determine if a straight line model would be a good approximation, a least squares regression line (See details of the lm function in the next section) can be added to the plot:

```
> abline(lm(Pulse$Pulse1~Pulse$Weight))
```



(b) **Simple linear regression**. The `lm()` function is a general purpose function for fitting linear models. The following code fits a simple linear regression model of Pulse1 as a function of Weight:

```
> lm(Pulse1~Weight, data=Pulse)
```

```
Call:
lm(formula = Pulse1 ~ Weight, data = Pulse)

Coefficients:
(Intercept)       Weight
      86.97        -0.17
```

This will produce as output only the estimates of the regression slope and intercept. However, if we create an object containing the results of the lm function, many additional results can be obtained. The `summary()` and `anova()` functions can then be applied to the object to obtain "typical" results:

```
> reg.fit<-lm(Pulse1~Weight, data=Pulse)
> summary(reg.fit)
```

```
Call:
lm(formula = Pulse1 ~ Weight, data = Pulse)

Residuals:
   Min     1Q Median     3Q    Max
-26.88  -9.22   0.42   6.73  71.63

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  86.9695     5.6368   15.43   <2e-16 ***
Weight       -0.1700     0.0828   -2.05    0.043 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 13.1 on 107 degrees of freedom
  (1 observation deleted due to missingness)
Multiple R-squared:  0.0379,        Adjusted R-squared:  0.0289
F-statistic: 4.21 on 1 and 107 DF,  p-value: 0.0425
```

```
> anova(reg.fit)
```

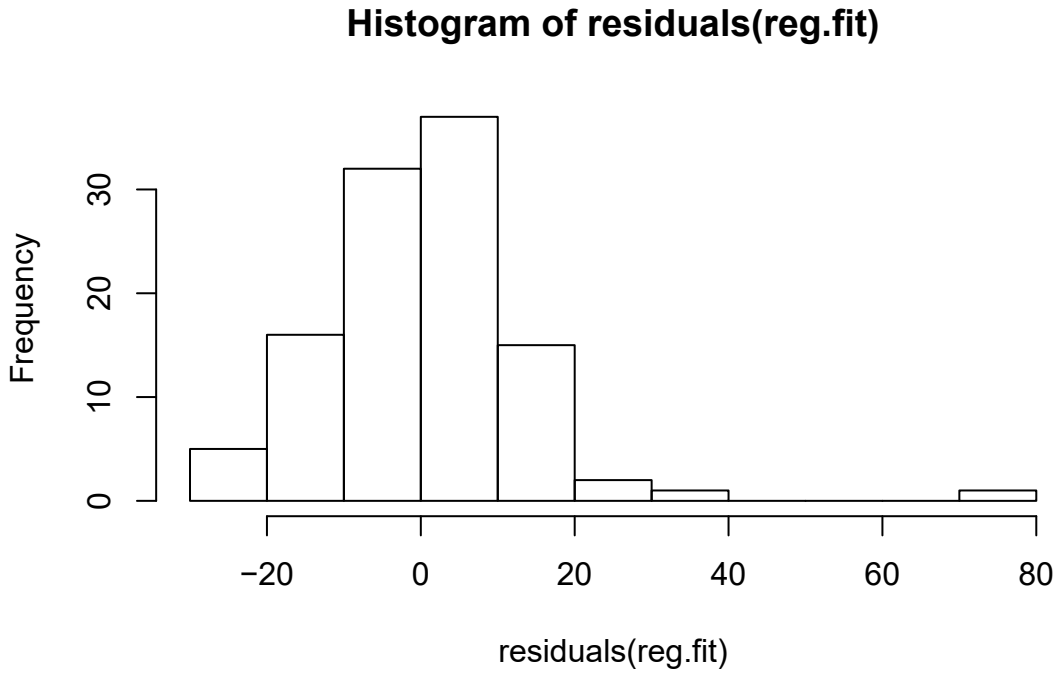
```
Analysis of Variance Table

Response: Pulse1
           Df Sum Sq Mean Sq F value Pr(>F)
Weight      1    724     724    4.21  0.043 *
Residuals 107  18374     172
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

A few other commonly used results such as residuals and fitted values can also be obtained:

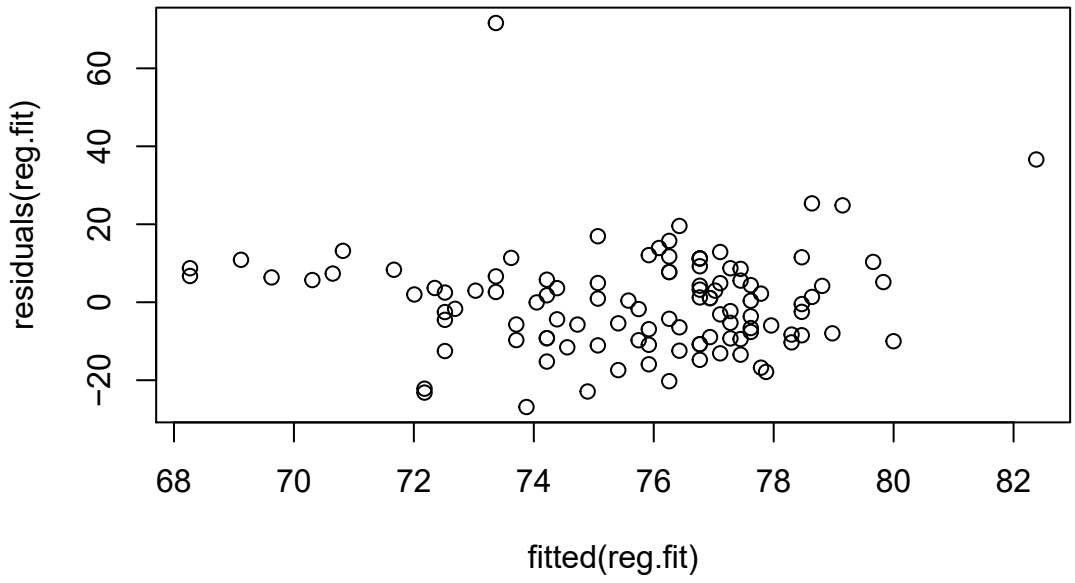
```
> fitted(reg.fit)
> residuals(reg.fit)
```

For anything other than a very small data set, outputting the fitted values and residuals is not very useful. However, plots of the residuals are frequently of interest. This can be done by using the `residuals()` and `fitted()` functions in conjunction with graphics functions. For example, a histogram of residuals and a scatterplot of residuals versus fitted values can be obtained with the `hist()` and `plot()` functions:

```
> hist(residuals(reg.fit))
```




```
> plot(fitted(reg.fit), residuals(reg.fit))
```



Note that the results of the `fitted()` and `residuals()` could be assigned to objects which could be useful especially if these will be used several times, as the functions would need only be called

once:

```
> pred.values<-fitted(reg.fit)
> resid<-residuals(reg.fit)
> hist(resid)
> plot(pred.values,resid)
```

(c) **Pearson correlation**. The `cor()` function will compute the correlation:

```
> cor(Pulse$Weight, Pulse$Pulse1)
```

```
[1] NA
```

Unfortunately, this did not appear to work, since the result is "NA". What is the problem? If we look at the data, observation 76 has a missing value for Pulse1. How does the `cor()` function handle missing data? Let's look at the documentation by submitting one of the following,

```
> help(cor)
> ?cor
```

or by searching for `cor()` function in the help tab in RStudio. The function specification is given below:

```
> cor(x, y = NULL, use = "everything", method = c("pearson", "kendall",
+     "spearman"))
```

What does the 'use= "everything"' argument do?

use	an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs"

If use is "everything", NAs will propagate conceptually, i.e., a resulting value will be NA whenever one of its contributing observations is NA.
If use is "all.obs", then the presence of missing observations will produce an error. If use is "complete.obs" then missing values are handled by casewise deletion (and if there are no complete cases, that gives an error).
"na.or.complete" is the same unless there are no complete cases, that gives NA. Finally, if use has the value "pairwise.complete.obs" then the correlation or covariance between each pair of variables is computed using all complete pairs of observations on those variables. This can result in covariance or correlation matrices which are not positive semi-definite, as well as NA entries if there are no complete pairs for that pair of variables.
For cov and var, "pairwise.complete.obs" only works with the "pearson" method. Note that (the equivalent of) var(double(0), use = *) gives NA for use = "everything" and "na.or.complete", and gives an error in the other cases.

Thus, by default, the `cor()` function will only return a value is all pairs are non missing. In this case, we fix this problem by changing the use argument:

```
> cor(Pulse$Pulse1,Pulse$Weight, use="pairwise.complete.obs")
```

```
[1] -0.1947
```

Finally, the `cor.test()` function will return a confidence interval and a p-value for the test of nonzero correlation:

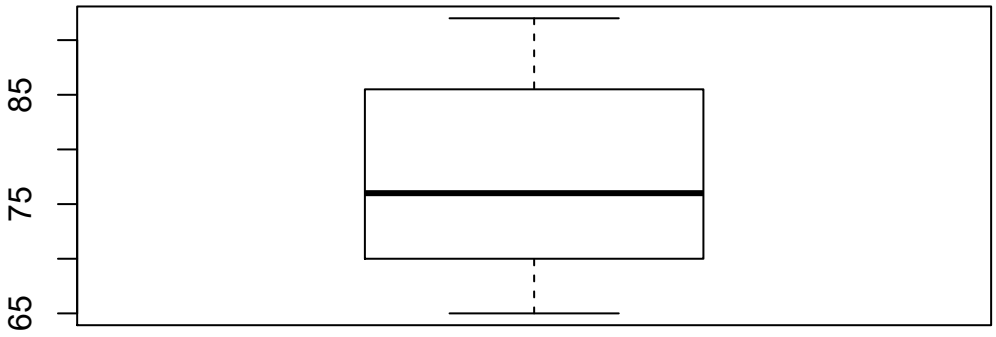
```
> cor.test(Pulse$Pulse1,Pulse$Weight, use="pairwise.complete.obs")
```

```
        Pearson's product-moment correlation

data:  Pulse$Pulse1 and Pulse$Weight
t = -2.1, df = 110, p-value = 0.04
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.369247 -0.006814
sample estimates:
    cor
-0.1947
```

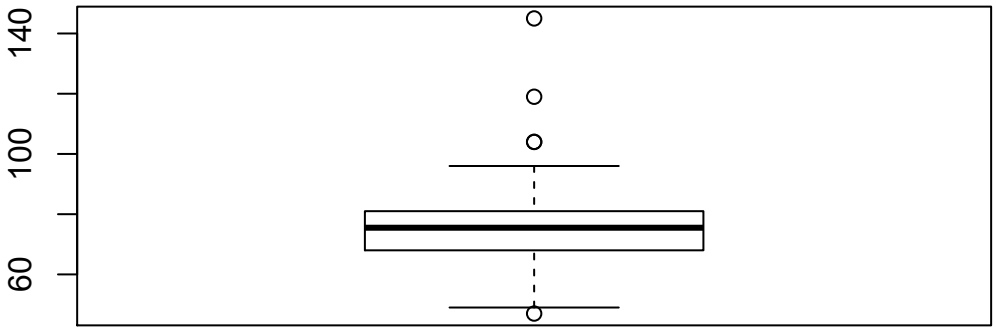
Explore the relationship between Pulse1 and smoking status.

(a) **Boxplot** of Pulse1 by smoking status: We want create two boxplots, one for smokers and one for nonsmokers. There are several ways to do this, and we will illustrate two. First, create individual boxplots for each Smoke category (smokers, nonsmokers)

```
> boxplot(Pulse$Pulse1[Pulse$Smokes==1])
```

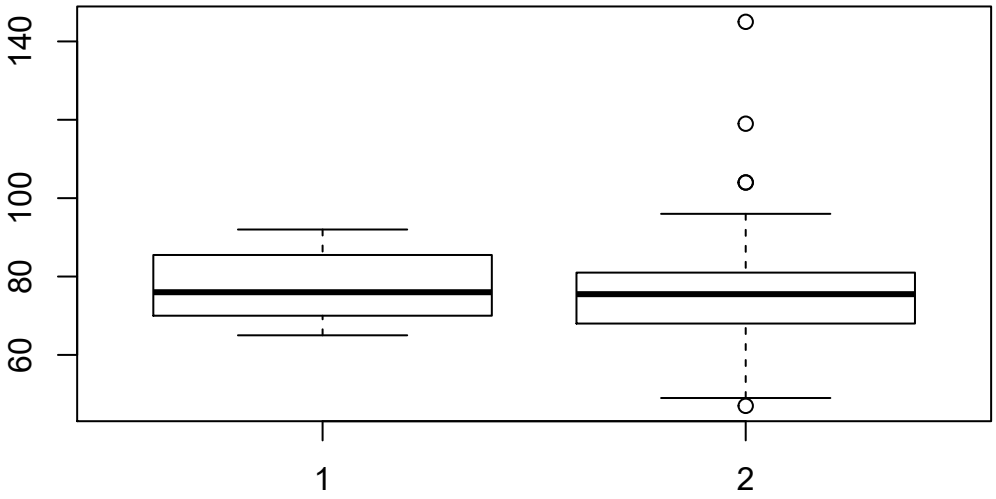



```
> boxplot(Pulse$Pulse1[Pulse$Smokes==2])
```



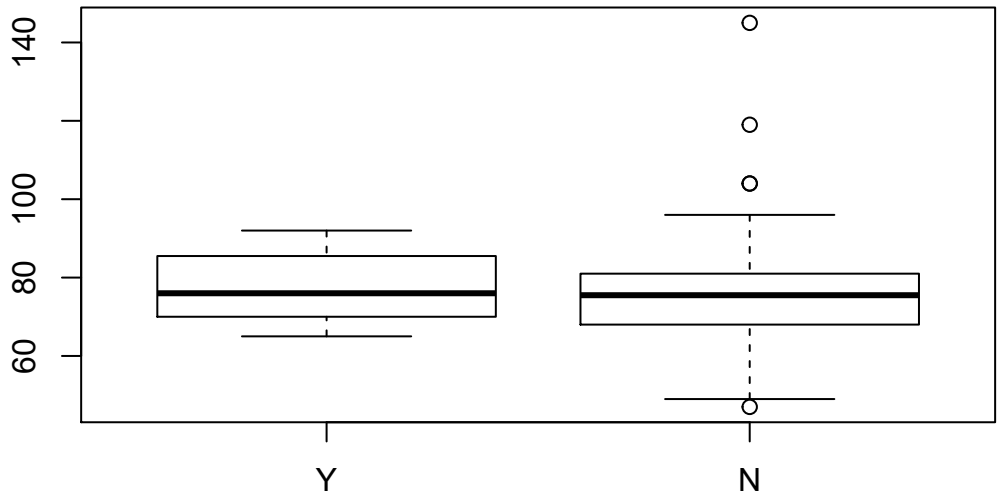
Second, to get side-by-side plots in the same graph, we first use the `factor()` function to create a new variable that transforms Smokes into a nominal variable called Smokes.factor, and then use the function argument in the `boxplot()` function:

```
> Pulse$Smokes.factor <- factor(Pulse$Smokes)
> boxplot(Pulse$Pulse1~Pulse$Smokes.factor)
```



While the factor function creates a new nominal level variable, it still uses the same values. Optional arguments can be added to the previous code to create more descriptive labels

```
> Pulse$Smokes.factor <- factor(Pulse$Smokes, levels = c(1, 2), labels = c("Y",
+     "N"))
> boxplot(Pulse$Pulse1 ~ Pulse$Smokes.factor)
```



(b) **Descriptives**. We will also want to compute descriptive measures, such as the mean and standard deviation of Pulse1 for each group.

```
> mean(Pulse$Pulse1[Pulse$Smokes==1])
```

```
[1] 77.55
```

```
> sd(Pulse$Pulse1[Pulse$Smokes==1])
```

```
[1] 9.575
```

```
> mean(Pulse$Pulse1[Pulse$Smokes==2])
```

```
[1] NA
```

```
> sd(Pulse$Pulse1[Pulse$Smokes==2])
```

```
[1] NA
```

Notice we encounter a problem caused by the missing observation. The documentation for the mean function:

```
> mean(x, trim = 0, na.rm = FALSE, ...)
```

na.rm	a logical value indicating whether NA values should be stripped before the computation proceeds.

This suggests the following fix for the NonSmoker group:

```
> mean(Pulse$Pulse1[Pulse$Smokes==2], na.rm=T)
```

```
[1] 75.48
```

```
> sd(Pulse$Pulse1[Pulse$Smokes==2], na.rm=T)
```

```
[1] 13.67
```

As an alternative to calculating values for each category separately, a single call to the `tapply()` function, using the Smokes.factor variable, will provide results as follows:

Notice we again use the na.rm argument,

```
> tapply(Pulse$Pulse1,Pulse$Smokes.factor, FUN=mean, na.rm=T)
```

```
    Y     N
77.55 75.48
```

(c) **T-test**. The `t.test()` function will compute a confidence interval for the mean difference and a p-value for a test of nonzero difference. We again employ the function argument to specify the variables:

```
> t.test(Pulse$Pulse1~Pulse$Smokes.factor)
```

```
        Welch Two Sample t-test

data:  Pulse$Pulse1 by Pulse$Smokes.factor
t = 0.65, df = 15, p-value = 0.5
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -4.755  8.886
sample estimates:
mean in group Y mean in group N
          77.55           75.48
```

By default, a two-sided p-value is reported, and equal population variances are not assumed.

The code below requests the pooled (equal variances assumed) and also requests a one-sided p-value:

```
> t.test(Pulse$Pulse1 ~ Pulse$Smokes.factor, alternative = "greater",
+     var.equal = T)
```

```
        Two Sample t-test

data:  Pulse$Pulse1 by Pulse$Smokes.factor
t = 0.49, df = 110, p-value = 0.3
alternative hypothesis: true difference in means is greater than 0
95 percent confidence interval:
 -4.975    Inf
sample estimates:
mean in group Y mean in group N
          77.55           75.48
```

2. Is there evidence that some students didn't run even though their coin toss came up heads?

   Is there evidence that fewer than 50% would be selected to run?

   (a) **Frequencies and proportions**. The `table()` function computes frequencies and crosstabs. We first create a new factor variable for the Ran variable. It will be helpful to save the result of the `table()` function as an object:

```
> Pulse$Ran.factor <- factor(Pulse$Ran, levels=c(1,2),
+                            labels=c("Y", "N"))
> ran.counts <- table(Pulse$Ran.factor)
> ran.counts
```

```
 Y  N
46 64
```

   The `prop.table()` function uses the object containing the counts to calculate proportions:

```
> prop.table(ran.counts)
```

```
     Y      N
0.4182 0.5818
```

   (b) **Test for proportion**. The `prop.test()` function, which has arguments similar to `t.test()`, can be used to compute a confidence interval for the proportion of students that ran between measurements. The function takes as arguments the number of successes and the sample size, which are calculated using the `length()` and `sum()` functions:

```
> n <- length(Pulse$Ran.factor)
> success <- sum(Pulse$Ran.factor == "Y")
> prop.test(success, n, alternative="less")
```

```
        1-sample proportions test with continuity correction

data:  success out of n, null probability 0.5
X-squared = 2.6, df = 1, p-value = 0.05
alternative hypothesis: true p is less than 0.5
95 percent confidence interval:
 0.0000 0.5011
sample estimates:
     p
0.4182
```

Does the proportion who ran between measurements depend on year?

(a) **Crosstabs, bar charts, chi-squared test**. The `table()` function can create crosstabulations for two or more variables. Next we calculate a table of Ran by Year:

```
> Year.table<-table(Pulse$Year,Pulse$Ran.factor)
> Year.table
```

```
    Y  N
93  8 18
95  9 13
96 11 10
97 11 12
98  7 11
```

As before, the `prop.table()` function can calculate proportions. For two-way tables, however, there are several different proportions possible for each cell, based on row, column or overall totals. The default is to divide counts by the overall sample size:

```
> prop.table(Year.table)
```

```
          Y       N
93 0.07273 0.16364
95 0.08182 0.11818
96 0.10000 0.09091
97 0.10000 0.10909
98 0.06364 0.10000
```

To obtain marginal proportions, we can specify the "margin=" argument, where 1 requests row marginals (divide counts bt row totals) and 2 column marginals (divide counts by column totals). For the purposes of the research question, dividing counts by Year totals would be most useful. Since Year was listed first in the `table()` function, it will be treated by R as the row variable, as thus the code below requests row marginal proportions:

```
> prop.table(Year.table,margin=1)
```

```
         Y      N
93 0.3077 0.6923
95 0.4091 0.5909
96 0.5238 0.4762
97 0.4783 0.5217
98 0.3889 0.6111
```

The `chisq.test()` function will compute a test for independence of Ran.factor and Year:

```
> chisq.test(Year.table)
```

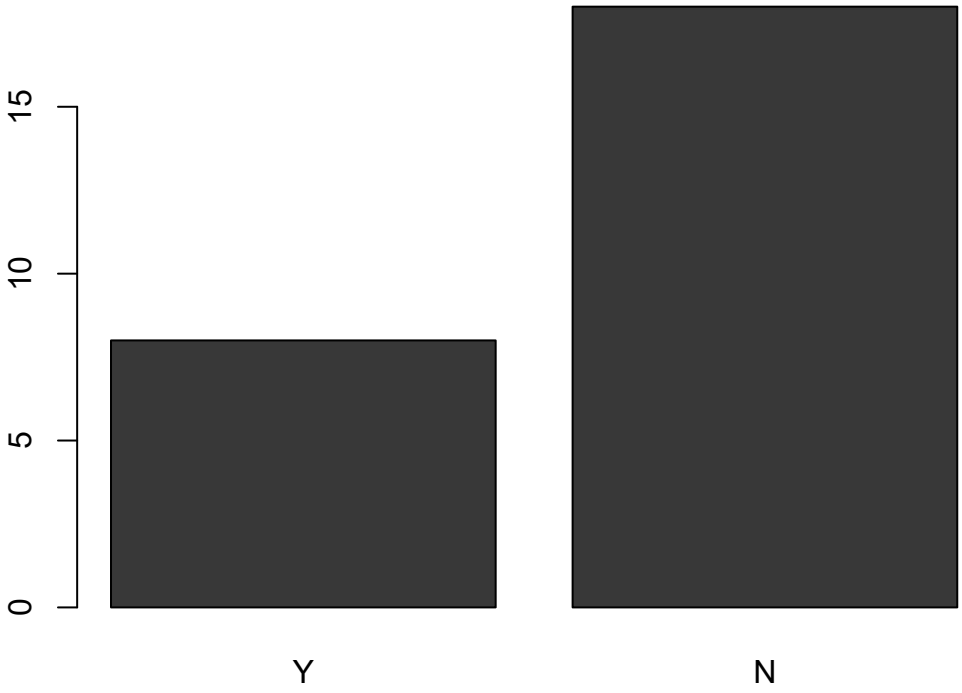
```
        Pearson's Chi-squared test

data:  Year.table
X-squared = 2.7, df = 4, p-value = 0.6
```

Bar charts are useful for displaying categorical data. We first illustrate the `barplot()` function for Ran.factor for just 1993:

```
> Year93.table<-table(Pulse$Year[Pulse$Year==93],
+                     Pulse$Ran.factor[Pulse$Year==93])
> Year93.table
```

```
     Y  N
  93  8 18
```

```
> barplot(Year93.table)
```

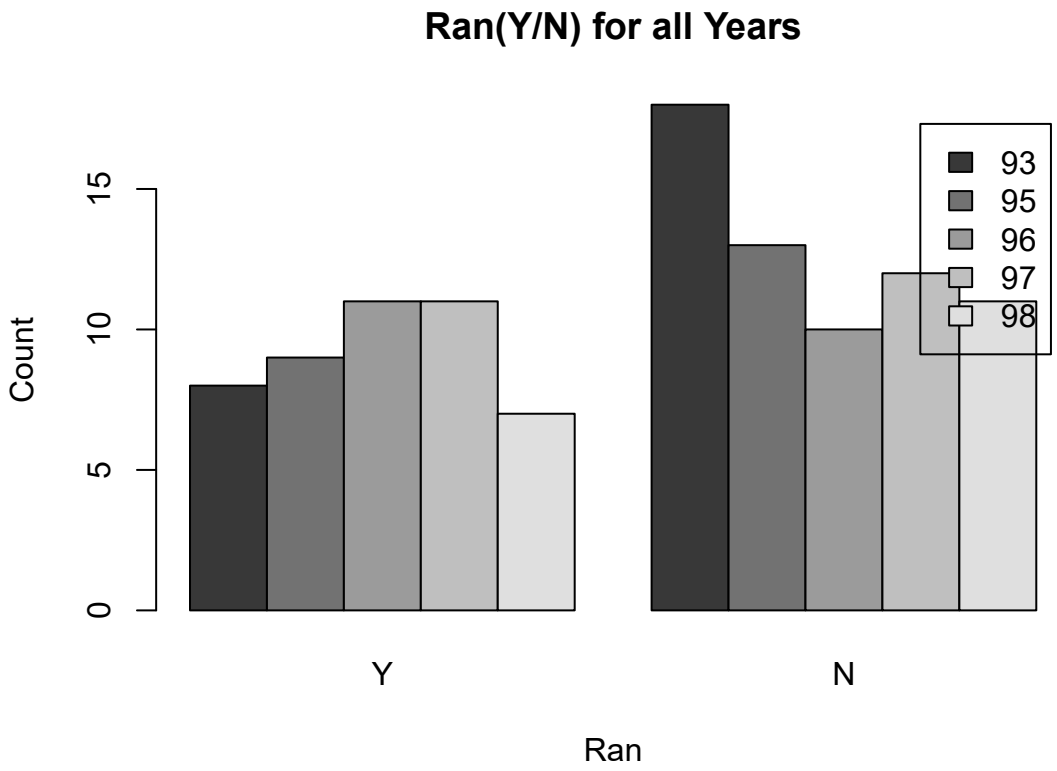


Next, side-by-side plots comparing all years are created:

```
> Year.table<-table(Pulse$Year,Pulse$Ran.factor)
> Year.table
```

```
     Y  N
  93  8 18
  95  9 13
  96 11 10
  97 11 12
  98  7 11
```

```
> barplot(Year.table,xlab="Ran",ylab="Count",
+         main="Ran(Y/N) for all Years",
+         legend=rownames(Year.table),beside=T)
```



We will create a new categorical variable for Year 93 using the function `ifelse()` then will create a new table to compare year 1993 with other years versus ran.factor,

```
> Pulse$Year.93 <- ifelse(Pulse$Year==93, 93, "other")
> Year.table.93 <-table(Pulse$Year.93, Pulse$Ran.factor)
> Year.table.93
```

```
        Y  N
```

```
  93     8 18
  other 38 46
```

Finally, the `chisq.test()` function will compute a test for independence of Ran.factor and Year.93:

```
> chisq.test(Year.table.93)
```

```
        Pearson's Chi-squared test with Yates' continuity correction

data:  Year.table.93
X-squared = 1.2, df = 1, p-value = 0.3
```

While we were able to perform the desired analyses using the packages included in the base install of R, it was somewhat laborious (e.g., separate statements to calculate counts, proportions, marginal proportions, chi-square test). There are many additional packages with functions tailored to specific types of analyses. For instance, the CrossTabs function in the gmodels package has the ability to do all of the analyses in this section (and more) in a single step.

We can use the function `install.packages()` to download the required package from CRAN and install the package in the local computer. Alternatively, choose Install within the Package tab in RStudio. Then we will have to load the package using the functions `library()` or `require()` to use it.

First, install the package gmodels:

```
> install.packages("gmodels")
```

Note: You should do this only one time (the very first time).

Now, use the function `library()` to load the gmodels:

```
> library(gmodels)
```

Searching for help on the `CrossTable()` function reveals the following information.

```
> CrossTable(x, y, digits = 3, max.width = 5, expected = FALSE, prop.r = TRUE,
+     prop.c = TRUE, prop.t = TRUE, prop.chisq = TRUE, chisq = FALSE,
+     fisher = FALSE, mcnemar = FALSE, resid = FALSE, sresid = FALSE,
+     asresid = FALSE, missing.include = FALSE, format = c("SAS",
+         "SPSS"), dnn = NULL, ...)
```

Let's use the `CrossTable()` function to create the previous results:

```
> CrossTable(Pulse$Year, Pulse$Ran, prop.t = T, prop.c = F, prop.chisq = F,
+     chisq = T)
```

```
   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  110


             | Pulse$Ran
  Pulse$Year |         1 |         2 | Row Total |
-------------|-----------|-----------|-----------|
          93 |         8 |        18 |        26 |
             |     0.308 |     0.692 |     0.236 |
             |     0.073 |     0.164 |           |
-------------|-----------|-----------|-----------|
          95 |         9 |        13 |        22 |
             |     0.409 |     0.591 |     0.200 |
             |     0.082 |     0.118 |           |
-------------|-----------|-----------|-----------|
          96 |        11 |        10 |        21 |
             |     0.524 |     0.476 |     0.191 |
             |     0.100 |     0.091 |           |
-------------|-----------|-----------|-----------|
          97 |        11 |        12 |        23 |
             |     0.478 |     0.522 |     0.209 |
             |     0.100 |     0.109 |           |
-------------|-----------|-----------|-----------|
          98 |         7 |        11 |        18 |
             |     0.389 |     0.611 |     0.164 |
             |     0.064 |     0.100 |           |
-------------|-----------|-----------|-----------|
Column Total |        46 |        64 |       110 |
-------------|-----------|-----------|-----------|


Statistics for All Table Factors
```

```
Pearson's Chi-squared test
------------------------------------------------------------
Chi^2 =  2.68     d.f. =  4     p =  0.6128
```

```
> CrossTable(Pulse$Year.93, Pulse$Ran, prop.t = T, prop.c = F, prop.chisq = F,
+     chisq = T)
```

```
   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  110


              | Pulse$Ran
Pulse$Year.93 |         1 |         2 | Row Total |
--------------|-----------|-----------|-----------|
           93 |         8 |        18 |        26 |
              |     0.308 |     0.692 |     0.236 |
              |     0.073 |     0.164 |           |
--------------|-----------|-----------|-----------|
        other |        38 |        46 |        84 |
              |     0.452 |     0.548 |     0.764 |
              |     0.345 |     0.418 |           |
--------------|-----------|-----------|-----------|
 Column Total |        46 |        64 |       110 |
--------------|-----------|-----------|-----------|


Statistics for All Table Factors


Pearson's Chi-squared test
------------------------------------------------------------
Chi^2 =  1.708     d.f. =  1     p =  0.1912

Pearson's Chi-squared test with Yates' continuity correction
------------------------------------------------------------
Chi^2 =  1.165     d.f. =  1     p =  0.2803
```

4 Activity

Data from surveys of customers of 168 Italian restaurants in New York City are available in the file "nyc.csv". The data are in the form of the average of customer views on

Variable	Description
Price	the price (in USD) of dinner (including one drink & a tip)
Food	customer rating of the food (out of 30)
Décor	customer rating of the decor (out of 30)
Service	customer rating of the service (out of 30)
East	1 (0) if the restaurant is east (west) of Fifth Avenue

Use R to help address the following questions:

1. Describe the distribution of Price and identify any outliers. (Hint: Consider plots such as histogram and boxplot and numerical summaries)

2. Is there a difference in Price when comparing restaurants on the east and west side of Fifth Avenue? (Hint: Consider side-by-side boxplots, numerical summaries separately for each area, t-methods).

3. Is Price associated with customer rating of food (Food)? (Hint: Consider a scatterplot, correlation/regression).

Note:
Example code for Activity tasks will be made available soon after the workshop on the QMS website: `http://www.uncg.edu/mat/qms/`