Introduction to R

Statistical Consulting Center
University of North Carolina at Greensboro

1. R Programming Basics

From https://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
- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and in-out and out-out 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 as an environment within which statistical techniques are implemented. R can be extended 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

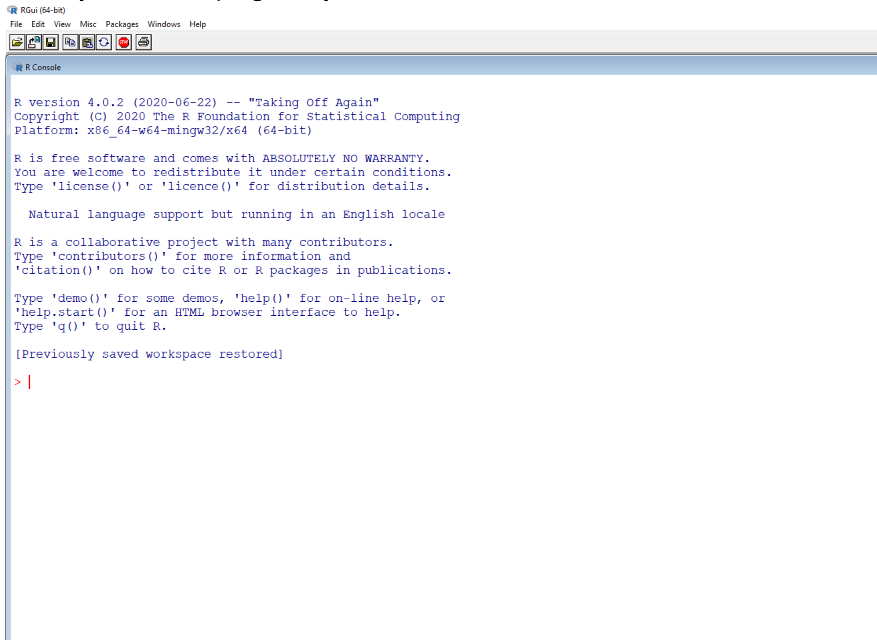
We will only scratch the surface!

2. Setting up R

2.1 Installing R

The Base R and packages can be downloaded from the Comprehensive R Archive Network (CRAN) (https://www.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] 1.94591
```

If the result of a calculation is to be used in further calculations, then assign the result 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] 1.94591
```

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

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

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

```
mean(Data)
```

```
## [1] 7.714286
```

```
sd(Data)
```

```
## [1] 3.147183
```

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
```

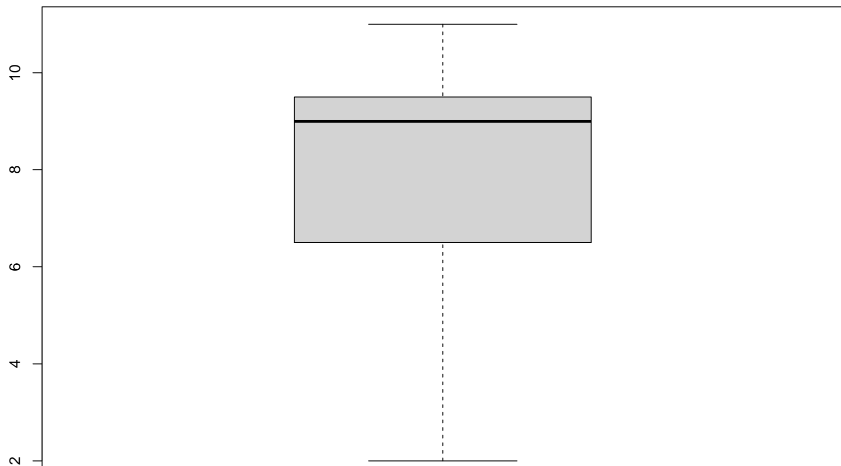
or

```
list(Data)
```

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

Finally, the **boxplot** function creates a boxplot of the data, which opens in a new window:

```
boxplot(Data)
```



Commands, object and variables 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 name "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

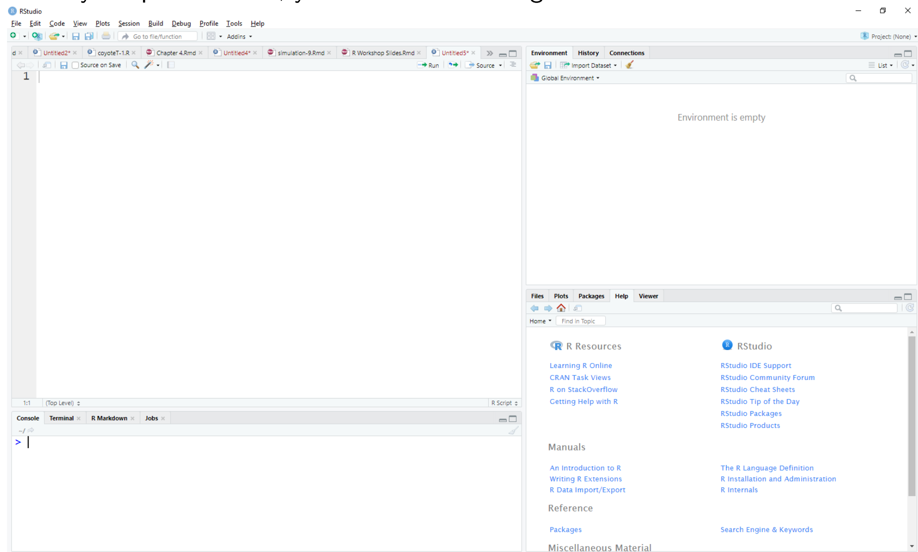
Because R is interactive and wants to execute commands each time the return key is hit, many users prefer to write blocks of code outside of the console, and then import or copy and paste the entire block to excite at once in the R console. There are several options for doing this:

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 R, choose 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: https://www.rstudio.com/products/rstudio/download/.

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

When you open RStudio, you will see something similar to:



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

- Environment/History window that shows all the active objects and command history
- A window with tabs that allows 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
- An Editor window in 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. Suppose the data of the previous examples is contained in a text file called "datafile1.txt", arranged as below, with rows corresponding to observations:

2
5
8
9
9
10

Then to create the data frame, use the command:

```
Data <- read.table(file="https://www.uncg.edu/mat/qms/datafile1.txt", header=F)
Data
```

Notice that forward slashes(/) are used in R to separate folders, whereas Windows uses back-slashes.

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

Now we may use functions to process the data. In the **read.table** function, the first argument is the specification of the location, **file=**, which is required. Next are two options, separated by commas. The first, **sep=""**, specifies the delimiter, which in this case is a space, while the second specifies that the first row of the file does not contain variable name. If the first row contains the name of the variable, then add the option **header=TRUE** (or **header=T**). 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 Our Working data set and results

We now consider a space-delimited data file containing several variables measured on students in an introductory statistics class. Students in a statistics class at The University of Queensland participated in an experiment. First, the students measured their 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. Next, everyone measured their pulse rate again. The pulse rates and other physiological and lifestyle variables are given in the data file. Five class groups between 1993 and 1998 participated in the experiment. After the first year, their arose a concern that some students chose the less strenuous option of sitting, even if their coin came up heads. Thus, in the years 1995-1998 a different method of random assignment was used: an equal number of students were randomly assigned to either running or sitting. 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)

Reading the data file

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="https://www.uncg.edu/mat/qms/data.txt",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="https://www.uncg.edu/mat/qms/data.txt", sep=",", header=T)

or the function **read.csv** may be used

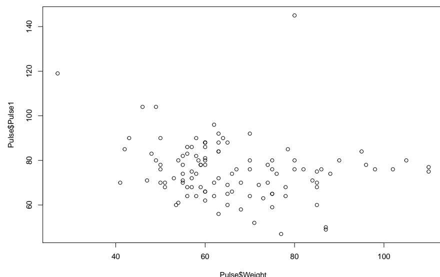
Pulse <- read.csv(file="https://www.uncg.edu/mat/qms/data.txt", header=T)

3.2 Research Questions

1. What is the nature and strength of the association between Pulse1 (the first pulse measurement) and certain 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."

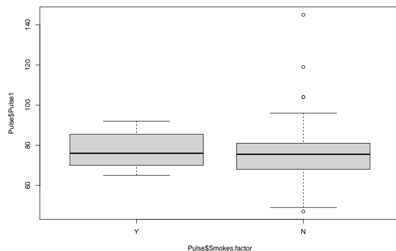


(*b*) **Simple linear regression**. "Each additional pound of weight was associated with a 0.17 beat per minute decrease in pulse"

(*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 non-smokers 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?

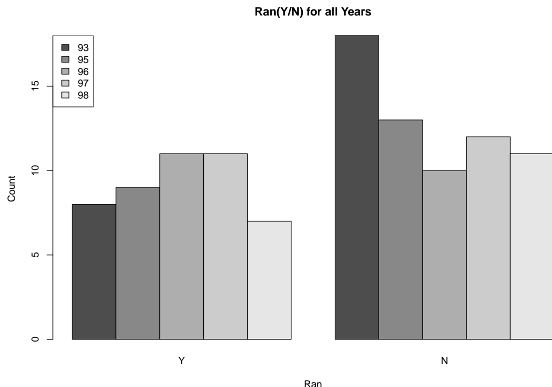
One way to answer this is to ask the question, "Is there evidence that fewer than 50% were selected to run?"

(*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 is moderate, but not convincing statistical evidence that fewer than 50% of all students would be selected to run ($\chi^2(1) = 2.63$, $p = 0.053$). Thus, there is not convincing statistical evidence that students lied about the result of the coin toss.

Is the proportion that ran between measurements dependent on year?

(c) **Crosstabs, bar charts, chi-squared test**. "The bar charts show a tendency for fewer students to run (except for 1996), but the discrepancy is greatest in 1993. When comparing 1993 to 1995-1998, 30.8% of students in 1993 admitted to obtaining heads compared to 45.2% in 1995-1998. However, this difference was not statistically significant ($\chi^2(1) = 1.17$, $p = 0.28$).



3.3 Using R to generate the results

Before we begin, it is important to check the data types of the variables, as sometimes the data type may not be what we expect (e.g., a numerical variable may be interpreted as a character or vice-versa). Thus we start by getting a summary of the variables and their types using the **str** function:

```
## 'data.frame':    110 obs. of  13 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       : int  2 2 2 1 2 1 2 2 2 1 ...
##  $ 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 ...
##  $ Smokes.factor: Factor w/ 2 levels "Y","N": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Ran.factor   : Factor w/ 2 levels "Y","N": 2 1 1 2 2 1 2 2 2 1 ...
```

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. The rest are considered numeric.

The **summary** function, when applied to a data frame, will compute the mean and 5-number summary for all numeric variables and frequencies for factors or character variables:

```
summary(Pulse)
```

```
##      Height          Weight             Age            Gender          Smokes
##  Min.   : 68.0   Min.   : 27.00   Min.   :18.00   Min.   :1.000   Min.   :1.0
##  1st Qu.:165.2   1st Qu.: 56.25   1st Qu.:19.00   1st Qu.:1.000   1st Qu.:2.0
##  Median :172.5   Median : 63.00   Median :20.00   Median :1.000   Median :2.0
##  Mean   :171.6   Mean   : 66.33   Mean   :20.56   Mean   :1.464   Mean   :1.9
##  3rd Qu.:180.0   3rd Qu.: 75.00   3rd Qu.:21.00   3rd Qu.:2.000   3rd Qu.:2.0
##  Max.   :195.0   Max.   :110.00   Max.   :45.00   Max.   :2.000   Max.   :2.0
##
##     Alcohol         Exercise          Ran            Pulse1
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   : 47.00
##  1st Qu.:1.000   1st Qu.:2.000   1st Qu.:1.000   1st Qu.: 68.00
##  Median :1.000   Median :2.000   Median :2.000   Median : 76.00
##  Mean   :1.382   Mean   :2.209   Mean   :1.582   Mean   : 75.69
##  3rd Qu.:2.000   3rd Qu.:3.000   3rd Qu.:2.000   3rd Qu.: 82.00
##  Max.   :2.000   Max.   :3.000   Max.   :2.000   Max.   :145.00
##                                                  NA's   :1
##      Pulse2           Year       Smokes.factor Ran.factor
##  Min.   : 56.0   Min.   :93.00   Y:11          Y:46
##  1st Qu.: 72.0   1st Qu.:95.00   N:99          N:64
##  Median : 84.0   Median :96.00
##  Mean   : 96.8   Mean   :95.63
##  3rd Qu.:125.0   3rd Qu.:97.00
##  Max.   :176.0   Max.   :98.00
##  NA's   :1
```

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 smoked (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 categorical 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.
##   27.00   56.25   63.00   66.33   75.00  110.00
```

How does Pulse1 (the first pulse measurement) depend on the lifestyle and physiological measurements?

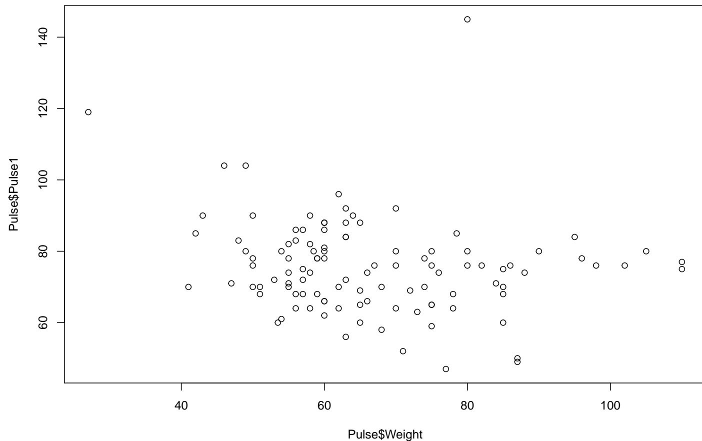
Explore the relationship between Pulse1 and Weight.

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

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

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

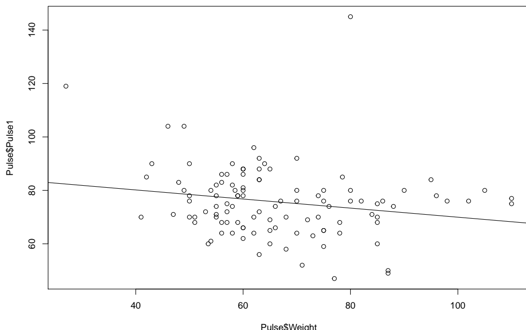
```
plot(Pulse$Weight, Pulse$Pulse1)
```



Explore the relationship between Pulse1 and Weight.

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

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

The previous code produced only the estimates of the slope and intercept of the regression line. However, if we create an object to store the results of the **lm** function, many additional results can be obtained.

The **summary** and **anova** functions are applied here 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
## -26.878  -9.218   0.422   6.733  71.632
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 86.96948    5.63684  15.429   <2e-16 ***
## Weight      -0.17002    0.08282  -2.053   0.0425 *
## ---
## 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:  0.03789,    Adjusted R-squared:  0.0289
## F-statistic: 4.214 on 1 and 107 DF,  p-value: 0.04252
```

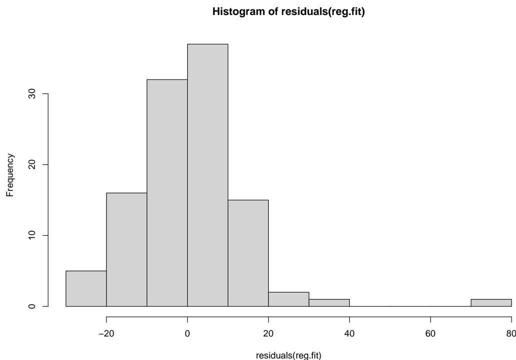
```r
anova(reg.fit)
```

```
## Analysis of Variance Table
##
## Response: Pulse1
##            Df  Sum Sq Mean Sq F value  Pr(>F)
## Weight      1   723.7  723.69  4.2144 0.04252 *
## Residuals 107 18373.7  171.72
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Functions for generating residuals and fitted (or “predicted”) values can also be used.

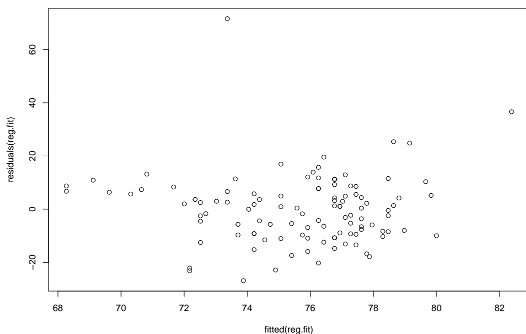
```r
fitted(reg.fit)
residuals(reg.fit)
```

For anything other than a very small data set, outputting fitted values and residuals is not practical. However, plots containing these are sometimes of interest. For example, a histogram of residuals is useful for assessing the normality assumption.



A scatterplot of residuals versus fitted values can be obtained in a similar fashion.

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



It might be helpful to assign the results of the **residual** and **fitted** functions to objects which can then be used in other functions.

```
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.

Executing either of the following commands will bring up the documentation on the **cor** function

help(cor)

cor

or by searching for **cor** in the Help tab in RStudio. The function syntax is given below.

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

What does the **use = “everything”** argument do?
In the “Arguments” section we find

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”

In the "Details" section we find **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 if all pairs are nonmissing. In this case, we can fixed the problem by changing the **use** argument.

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

```
## [1] -0.1946657
```

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

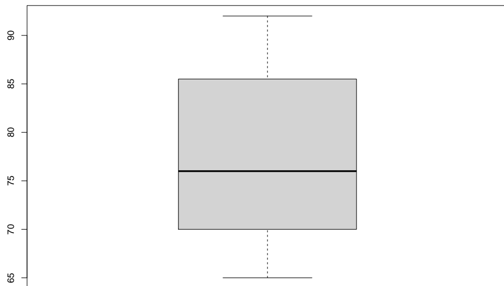
```
cor.test(Pulse$Pulse1, Pulse$Weight, use="pairwise.complete.obs")
```

```
##
##  Pearson's product-moment correlation
##
## data:  Pulse$Pulse1 and Pulse$Weight
## t = -2.0529, df = 107, p-value = 0.04252
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.369246770 -0.006813511
## sample estimates:
##        cor
## -0.1946657
```

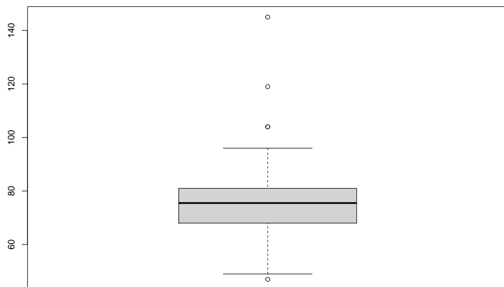
Explore the relationship between the initial pulse measurement and smoking status.

(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 *Smokes* category (smokers, nonsmokers). We do this by evaluating the Pulse1 variable only for observations where *Smokes* is at a single level.

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

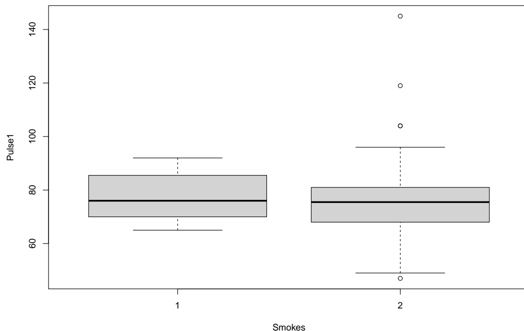


```r
boxplot(Pulse$Pulse1[Pulse$Smokes==2])
```



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

```
Pulse$Smokes.factor <- factor(Pulse$Smokes)
boxplot(Pulse1 ~ Smokes, data=Pulse)
```

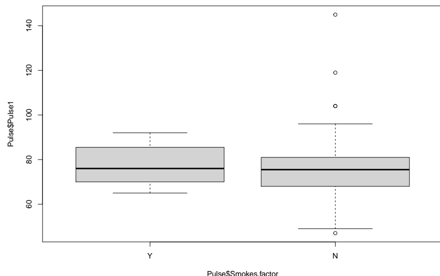


While the factor function creates a new factor variable, it still uses the original values. Optional arguments can be added to create more descriptive labels:

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

The boxplot will now use the newly defined labels.

```
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] 77.54545
```

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

```
## [1] 9.574588
```

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

```
## [1] NA
```

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

```
## [1] NA
```

Notice that we encounter a problem caused by the missing observation. We find in the documentation of 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] 75.47959
```

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

```
## [1] 13.67459
```

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
## 77.54545 75.47959
```

(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 = 0.64552, df = 15.022, p-value = 0.5283
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -4.754562  8.886287
## sample estimates:
## mean in group Y mean in group N
##        77.54545        75.47959
```

By default, a two-sided *p*-value is reported, and equal population variances are not assumed (the Smith/Welch/Satterthwaite method).

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 = 0.48684, df = 107, p-value = 0.3137
## alternative hypothesis: true difference in means is greater than 0
## 95 percent confidence interval:
##  -4.974943       Inf
## sample estimates:
## mean in group Y mean in group N
##        77.54545        75.47959
```

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
## 0.4181818 0.5818182
```

(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 = 2.6273, df = 1, p-value = 0.05252
## alternative hypothesis: true p is less than 0.5
## 95 percent confidence interval:
##  0.0000000 0.5011424
## sample estimates:
##         p
## 0.4181818
```

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. Here we calculate a 2-way 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 0.07272727 0.16363636
##   95 0.08181818 0.11818182
##   96 0.10000000 0.09090909
##   97 0.10000000 0.10909091
##   98 0.06363636 0.10000000
```

To obtain marginal proportions, we can specify the "margin=" argument, where "1" requests row marginals (divide counts by 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 0.3076923 0.6923077
##    95 0.4090909 0.5909091
##    96 0.5238095 0.4761905
##    97 0.4782609 0.5217391
##    98 0.3888889 0.6111111
```

Test for independence

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

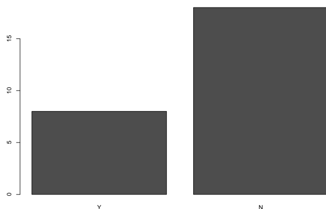
```
chisq.test(Year.table)
```

```
##
##  Pearson's Chi-squared test
##
## data:  Year.table
## X-squared = 2.6797, df = 4, p-value = 0.6128
```

Plots

Bar charts can be useful for displaying categorical data. We first illustrate the **barplot** function for Ran.factor for just 1993.

```
##
##        Y  N
##    93  8 18
```



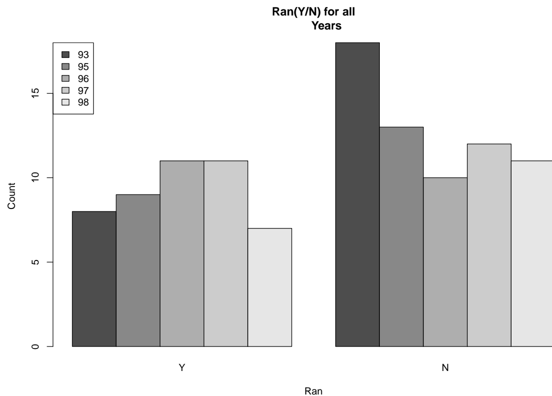
Side-by-side plots

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
```

Side-by-side plots



Side-by-side plots

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
```

Test for independence

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 = 1.1654, df = 1, p-value = 0.2803
```

Installing packages

While we were able to perform the desired analyses using the packages included in the Base install of R, it was somewhat tedious. 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. The **install.packages** function can be used to download the required package from CRAN and install the package locally. (Alternatively, choose "Install" within the "Package" tab in RStudio.) We will then have to make the package available using the **library** function.

First, install *gmodels*.

>install.packages("gmodels")

Note: You only need to do this one time.

Then use the **library** function to make the package available for the current R session.

Once the **library** function is executed, any documentation associated with the *gmodels* package will be available. Searching for help as before 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 **Crosstable** to try to create the previous results.

```
   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 =  2.679713     d.f. =  4     p =  0.6127693
```

CrossTable

```
   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 =  1.708348     d.f. =  1     p =  0.1911998

Pearson's Chi-squared test with Yates' continuity correction
------------------------------------------------------------
Chi^2 =  1.165422     d.f. =  1     p =  0.2803439
```