R Guide

version 0.1

Last generated: December 10, 2019

Table of Contents

Introduction

This is a list of commands and their contexts for getting started with using R. R is a free programming language for statistical computing and graphics. To download R, visit cran and find the location closest to you. Then, follow the instructions based on your operating system. Many people who use R like the program RStudio, which has a console and interface for using R. That can be downloaded for free under an open source license from the RStudio website.

Using R

R can be loaded from the command line with `R`. Easy enough. If you remember, in Unix there is a `$` to tell you you're at the command line. For R's internal command line, that symbol is `>`. To clear the console in R or RStudio on a Windows or Linux operating system, use `Cntrl+L`. On a Mac, you can use either `Cntrl+L` or `Command+Option+L`.

PDF Download

Getting Help

R has some internal help commands. These include `help(functionname)` and `example(functionname)`. An example is below.

```
> help(sum)
sum                     package:base                    R Docume
ntation

Sum of Vector Elements

Description:

     'sum' returns the sum of all the values present in its arg
uments.

Usage:

     sum(..., na.rm = FALSE)
...
```

Printing and Setting the Working Directory

The folder that you work through the terminal is known as the working directory. To know what your current working directory is, use:

```
> getwd()
```

which will return the path to where you are. To specify a working directory, you need to put the path to that directory in, which is shown in the example below.

```
> setwd("/Users/Path/Through/The/Computer")
```

Similarly, if a directory called Land within Computer needed to be accessed, then use:

```
> setwd("Land")
```

to access it.

Using Unix Commands from R

Another important thing to mention is that anything that needs to be run from the terminal can be done through the R command prompt using `system("")`. It will also print the information in the typical Unix format.

```
> system("pwd")
/Users/Path/Through/The/Computer
```

Loading a Previous Session

Previous R sessions are saved with the `.RData` extension.

```
> load("~/Path/Through/The/Computer/Example_old_session.RData")
```

Using External Data, Files, and Scripts

Files can be listed by using “list.files()”, which will return possible files. The source command will run a script, and is shown below.

```
> source("bottle1.R")
[1] "This be a message in a bottle1.R!"
```

R can read a .csv file, which literally means comma separated values. The example below shows a csv, titled `targets.csv`, being read.

```
read.csv("schools.csv")
University                          Students          Tuition
1 Truman State University         6200               13500
2 University of Iowa                 33300               29000
3 University of Michigan         44700               45000
4 University of North Texas         37900               20000
```

Text files (.txt) can also be read, but if the separator is a tab, the “read.table” command is a better fit. The following example shows the tab separator specification (`sep="\t"`).

```
read.table("population.txt", sep="\t")
        V1        V2
1          City         Population
2 Joliet        148262
3 Kirksville        17519
4 Denton        133808
5 New Lenox        26217
```

This example had V1 and V2 as headers. R isn’t automatically aware that they are headers, so that needs to be specified when reading the table.

```
read.table("population.txt", sep="\t", header=TRUE)
        City        Population
1 Joliet        148262
2 Kirksville        17519
3 Denton        133808
4 New Lenox        26217
```

Directing Output to a File

The `sink()` command is used direct output from an R job to a new file. Having nothing in the parentheses will return the output to the terminal. Using something like

```
> sink("myfile.txt", append=TRUE, split=TRUE)
```

will direct the output to a file titled `myfile.txt`, the append option will allow the output to be added to the existing file as opposed to overwriting it, and the split option will send the data to the screen and the output file.

Installing, Using, and Removing Packages

Packages are basically code bundles that enable certain functions and data sets to be used. If you know the name of the package that you wish to install, (in this example that package is `car`), then use:

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

Once the package is installed, it can be referenced by using `library` to call it.

```
> library(car)
```

The active packages can be checked by using:

```
> (.packages())
```

Removing packages is known as “detaching” them. The following example removes the `car` package.

```
> detach("package:car")
```

Sample Data

R has several preloaded data packages that can be very helpful in learning R. The available packages can be loaded by using the command `data()`, which will then bring up a menu with options. To exit that menu, type `q`. To list what data is currently loaded, use the `ls()` command with empty parentheses. The following example loads the Orange data set (regarding the growth of orange trees).

```
> data(Orange)
```

The commands `head()`, `tail()`, and `rm()` can be used similarly to how they would be in a Unix environment. To show the first line of the Orange dataset, the command would be `head(Orange, n=1)`. If you wanted to unload the Orange dataset , use `rm(Orange)`.

Exiting R

The R console can be exited by using `q()`.

```
> q()
Save workspace image? [y/n/c]:
```

The quit command prompts the ability to save your R session. Respond to the prompt, where `y=yes`, `n=no`, and `c=cancel`.

Math in R

You can use several different commands that are similar to how calculator programming works. The sum function would take the sum of all entered after it.

```
> sum(1, 3, 5)
[1] 9
```

The repeat function is fairly intuitive.

```
> rep("Yo ho!", times = 3)
[1] "Yo ho!" "Yo ho!" "Yo ho!"
```

As is acquiring the square root.

```
> sqrt(16)
[1] 4
```

Vectors

Vectors are a basic data structure that can include logical, integer, double, character, complex, or raw data. It can only hold one type of data at a time. A new vector can be created with the values `4, 7,` and `9`, where `c` stands for combine.

```
> c(4, 7, 9)
[1] 4 7 9
```

If the following command was used, all of the values would be coded as character values.

```
> c(1, TRUE, "three")
[1] "1"         "TRUE"         "three"
```

Sequences and Variables

Sequences of numbers can also be saved as a vector, using a colon (`start:end`) or [`seq(start, end)`]. Using `seq` allows increments other than 1 to be used, when the increment is specified after end [`seq(start, end, increment)`].

```
> a <- 5:9
[1]        5        6        7        8        9
> alpha <- seq(5, 9)
[1]        5        6        7        8        9
> quartersvector <- seq(5, 6, 0.25)
[1]        5.00        5.25        5.50        5.75        6.00
> variablecharlie <- 9:5
[1]        9        8        7        6        5
```

The above example demonstrated saving sequences of numbers as variables. You can access the individual value in the vector through using the variable name and brackets. The following example will show that and include comments (shown by `\#`) about what's happening.

```
> variablecharlie[3] #I'm calling value 3 stored in the vector
[1] 7 #The third value is the number 7
> quartersvector[2]
[1] 5.25
> alpha[2] <-68 #replaces the second position of the vector wit
h 68
> print(alpha) #prints the values in alpha to command line
[1] 5 68 7 8 9
```

Names can be assigned to the variable with `names()`. After assignment, printing will show the names. Either single or double quotes can be used to assign names.

```
> limbs <- c(4, 3, 4, 3, 2, 4, 4, 4)
> names(limbs) <- c('One-Eye', 'Peg-Leg', 'Smitty', 'Hook', 'Sc
ooter', 'Dan', 'Mikey', 'Blackbeard')
> print(limbs)
One-Eye        Peg-Leg        Smitty        Hook        Scoote
r        Dan        Mikey        Blackbeard
4        3        4        3        2        4        4
4
```

To remove a variable (or dataset), use the `rm()` command, where the data you wish to remove is specified in the parentheses. Similarly, to list what data is available, use the `ls()` command with empty parentheses.

Saving R Data

Data can be saved with the `save()` command. Multiple data sets can be saved in one data file (in the example, these are `x`, `y`, and `z`). ```R

> save(x, y, z, file=”xyz.rda”)

Matrices

Filling Matrices

To make a 4x5 matrix filled with zeroes, it would look like:

```
> matrix(0, 4, 5)
        [,1]        [,2]        [,3]        [,4]        [,5]
[1,]        0        0        0        0        0
[2,]        0        0        0        0        0
[3,]        0        0        0        0        0
[4,]        0        0        0        0        0
```

If you wanted to have a matrix with values specified, store those values as a vector (page 15) first. The following example specifies a vector, `a`, as a sequence from 1 to 20, and then stores it as a matrix.

```
> a <- 1:20
> matrix(a, 4, 5)
        [,1]        [,2]        [,3]        [,4]        [,5]
[1,]        1        5        9        13        17
[2,]        2        6        10        14        18
[3,]        3        7        11        15        19
[4,]        4        8        12        16        20
```

Determinants and Eigen-stuff

The determinant can be taken on a matrix. If the variable has been defined, but not yet placed in a matrix, both steps must be done at once.

```
> a<-1:25
> det(matrix(a,5,5))
[1] 0
```

The `eigen` function (I know, I'm laughing too) computes both eigenvalues and eigenvectors. If the matrix is symmetric, then include `symmetric=TRUE` to skip the symmetry check.

```
> eigen(matrix(a,5,5))
eigen() decomposition
$values
[1]  6.864208e+01+0.000000e+00i -3.642081e+00+0.000000e+00i
[3]  4.257350e-15+0.000000e+00i -1.270981e-16+4.588876e-16i
[5] -1.270981e-16-4.588876e-16i

$vectors
             [,1]           [,2]
[,3]                  [,4]
[1,] 0.3800509+0i -0.76703416+0i  0.54621260+0i  0.1175132+0.04
59634i
[2,] 0.4124552+0i -0.48590617+0i -0.27228461+0i  0.4017692+0.00
65072i
[3,] 0.4448594+0i -0.20477817+0i -0.66418830+0i -0.7435988+0.00
00000i
[4,] 0.4772637+0i  0.07634982+0i -0.03961996+0i -0.1881630-0.20
33750i
[5,] 0.5096680+0i  0.35747782+0i  0.42988027+0i  0.4124793+0.15
09044i
                      [,5]
[1,]  0.1175132-0.0459634i
[2,]  0.4017692-0.0065072i
[3,] -0.7435988+0.0000000i
[4,] -0.1881630+0.2033750i
[5,]  0.4124793-0.1509044i
```

Matrix Multiplication

For element-wise multiplication, simply use the `A*B` format. To multiply matrices, you would use the `%*%` symbol. The outer product (AB') can be obtained using the `%o%` symbol.

```
> a <- 1:20
> b <- 21:40
> matrix(a, 4, 5)
        [,1]        [,2]        [,3]        [,4]        [,5]
[1,]       1         5         9        13        17
[2,]       2         6        10         14         18
[3,]       3         7        11         15         19
[4,]       4         8        12         16         20
> matrix(b, 5, 4)
     [,1] [,2] [,3] [,4]
[1,]   21   26   31   36
[2,]   22   27   32   37
[3,]   23   28   33   38
[4,]   24   29   34   39
[5,]   25   30   35   40
> a%*%b
     [,1]
[1,] 7070
```

Other Matrix Operations

Different formats are given, where `A` and `B` are matrices, and `k` is a scalar.

Table: Formats for Matrices

Command	Function
`t(A)`	transpose
`crossprod(A,B)`	A’B
`crossprod(A)`	A’A
`solve(A)`	Inverse of A (if A is a square matrix)
`solve(A, b)`	Returns x vector in b = Ax equation
`diag(A)`	Vector with elements of principal diagonal
`diag(K)`	Creates the k x k identity matrix
`rowMeans(A)`	Vector of row means

Command	Function
`rowSums(A)`	Vector of row sums
`columnMeans(A)`	Vector of column means
`columnSums(A)`	Vector of column sums
`rowbind(A,B,...)`	Combines matrices or vectors vertically, returning a matrix
`cbind(A,B,...)`	Combines matrices or vectors horizontally, returning a matrix

Writing Information to a File

Data, usually matrix data, are written to a file using the `write` command. In it, the original data vector is specified, then the file name, and then the delimiter. Tabs are specified through `\t`, but other separators can be used if placed in the quotes.

```
> write(mydata, "mydata.txt", sep="\t")
```

Files can also be written using `write.csv()` and `write.table()`.

Factors

Factors are a new vector comprised of integer values with a corresponding character value. They are categorical variables. A good way to describe this is when thinking of an Olympic roster. There's hundreds of athlete names, and under "medal type" there would be a factor with four levels: gold, silver, bronze, and none. R can assign numbers to these factors when doing analysis (so, in the case of alleles, AA=3, Aa=2, aa=1 could be an assignment). To store a vector as a variable of a factor, use `variable <- factor(vectorname)`. Their integer assignments can be checked with `as.integer(variable)`, and the levels can be checked with `levels(variable)`.

```
> allele <- c( 'aa', 'Aa', 'AA', 'aa', 'aa', 'aa', 'aa', 'Aa',
'Aa')
> types <- factor(allele)
> print(types)
[1] aa Aa AA aa aa aa aa Aa Aa
Levels: aa Aa AA
> as.integer(types)
[1] 1 2 3 1 1 1 1 2 2
> levels(types)
[1] "aa" "Aa" "AA"
```

Mean

Taking the average of a variable is as easy as using `mean(variable)`.

```
> spidersconsumed <- c(0,0,0,0,0,0,0,0,0,30)
> mean(spidersconsumed)
[1] 3
```

The average person eats 3 spiders per year. The child of Spiders Georg, Spiders Meorg, ate 30 spiders, and kept the average in his class of 10 students. Clearly, this lead to a statistical error, and the data would be better represented another way.

Median

Similarly to mean, simply running `median(variable)` provides the median.

```
> median(spidersconsumed)
[1] 0
```

Thank goodness!

Mode

Unlike the other areas of central tendency, R doesn't have a straightforward way to calculate the mode. Never fear: the following has an example for both numeric and character vectors.

```
# Create the function.
getmode <- function(spidersconsumed) {
   uniq <- unique(spidersconsumed)
   uniq[which.max(tabulate(match(spidersconsumed, uniq)))]
}

# Create the vector with numbers.
spidersconsumed <- c(0,0,0,0,0,0,0,0,0,30)

# Calculate the mode using the user function.
result <- getmode(spidersconsumed)
print(result)
[1] "0"

# Create the vector with characters.
students <- c("Susie", "Jessica", "Sally", "Henry", "George",
"Alex", "Alex", "Olive", "Patrick", "Meorg")

# Calculate the mode using the user function.
result2 <- getmode(students)
print(result2)
[1] "Alex"
```

Standard Deviation

To get the standard deviation, use `sd(variable)`.

```
> sd(spidersconsumed)
[1] 9.486833
```

This is almost worse, because it means people eat 3 ± 9.48 spiders per year!

Inner-Quartile Range

The command `summary(variable)` can be used to get the minimum, 1st IQR, median, mean, 3rd IQR, and maximum values.

```
>summary(spidersconsumed)
        Min.        1st Qu.        Median        Mean        3r
d Qu.        Max.
        0        0        0        3        0        30
```

Stem Plot

One way to visualize your data is with a stem and leaf plot, which gives a visual representation of where your data lie. The command to create a stem plot is `stem(variable)`.

```
> stem(spidersconsumed)

  The decimal point is 1 digit(s) to the right of the |

  0 | 000000000
  1 |
  2 |
  3 | 0
```

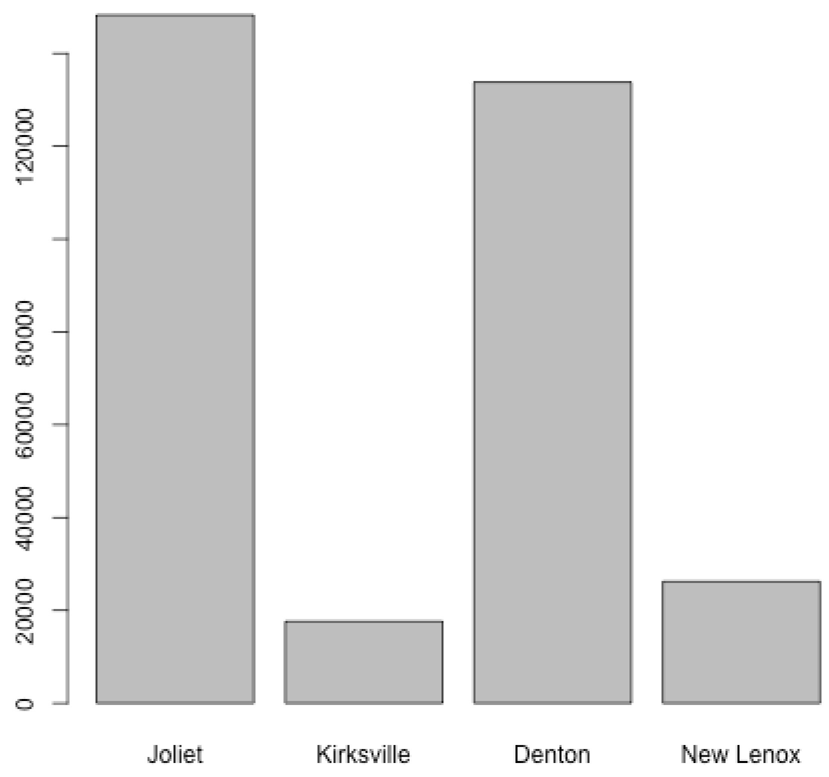
Wow, Spiders Meorg really does look like an outlier now!

Barplots

The following example shows what steps were taken to create a vector named `population` and then have a barplot with those values.

```
> population <- c(148262,17619,133808,26217)
> names(population) <- c("Joliet", "Kirksville", "Denton", "Ne
w Lenox")
> setwd("~/Desktop")
> png(filename="populationbarplot.png")
> barplot(population)
> dev.off()
null device
        1
```

The command `names` assigned names to the values in the vector `population`. The working directory was set to specify where the file save location for a `png` file of the generated image `populationbarplot.png` should be saved (in this case, the Desktop). (Note: you can also select `pdf()` and `jpg()`.) Then, the barplot was made with `barplot`. Finally, because you're no longer using the screen for image generation, you turn off the graphics device with `dev.off()`. The generated barplot is shown in the image below.



Example use of `barplot`, which creates a barplot.

To add a line at the mean (or median, etc.) of the population, the `abline` command can be used, which will update the current plot.

```
> abline(h = mean(population))
```

To show one standard deviation above the mean, use an argument of `h = mean(population) + sd(population)`. The only way to remove a line by mistake is to regenerate the barplot and effectively start over. Multiple lines need to be added with individual `abline` commands.

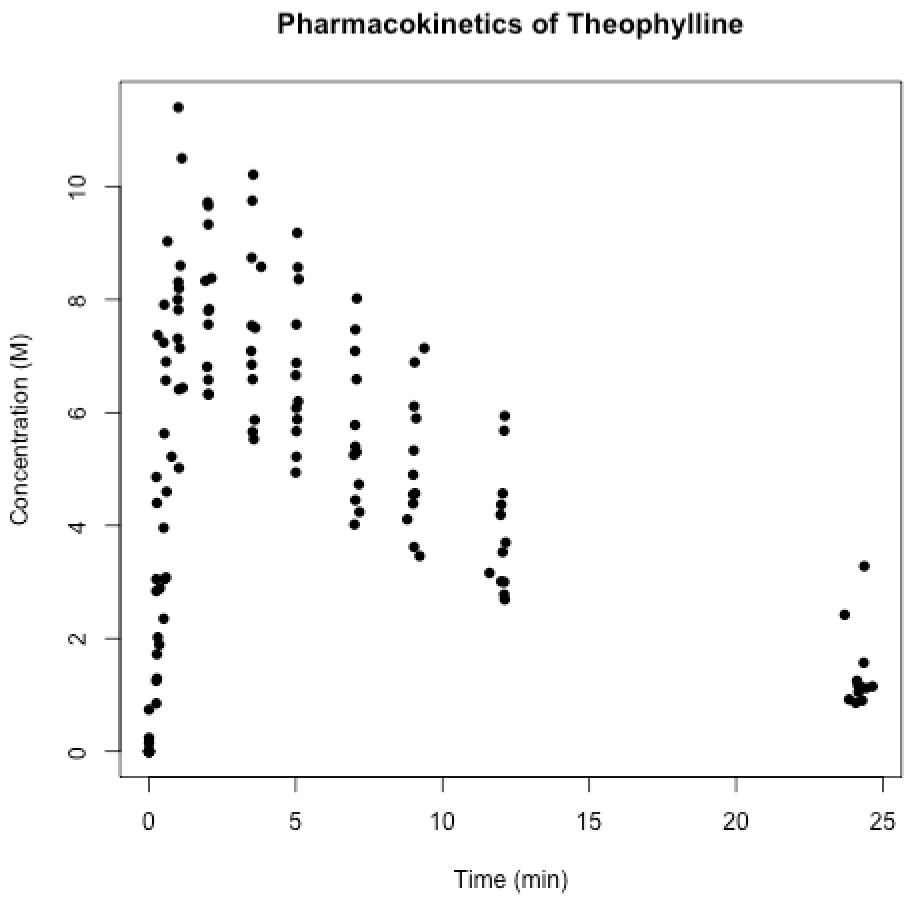
Scatter Plots

This example will use the preloaded dataset Theoph, which has data from an experiment on the pharmacokinetics of the respiratory drug thephylline. By using head on this dataset, we can see what the columns are coded as.

```
> head(Theoph, n = 1)
        Subject        Wt         Dose        Time        conc
1        1        79.6        4.02        0        0.74
```

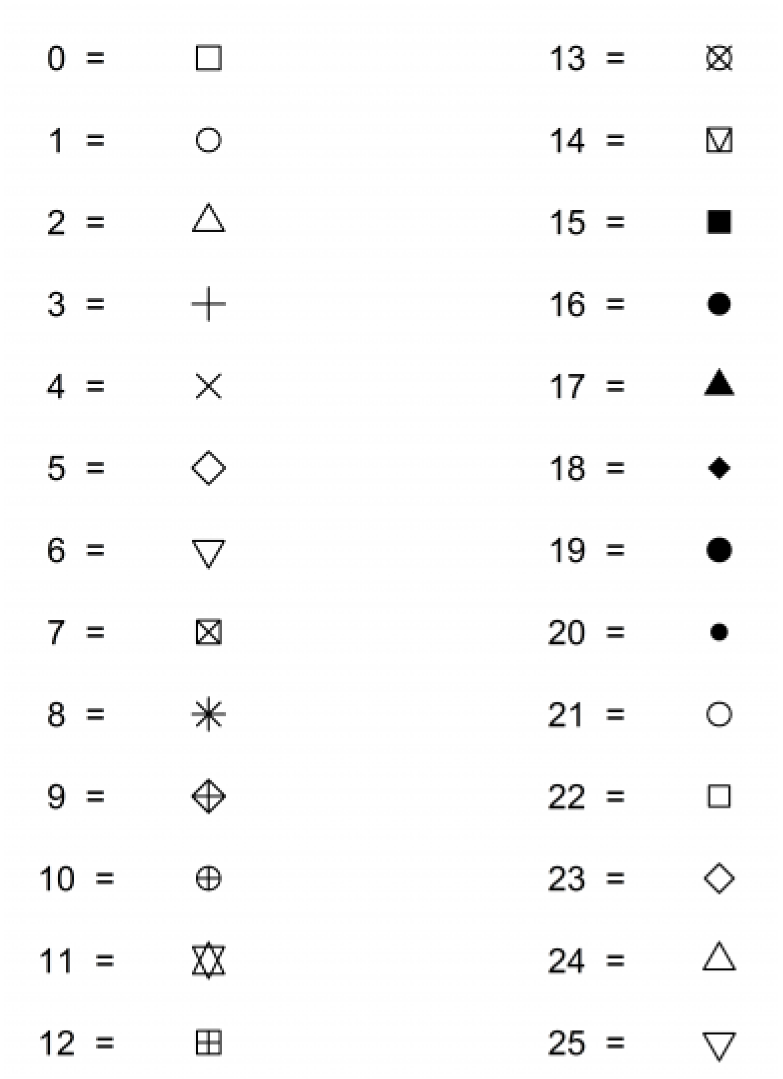
Knowing how the columns are coded allows you to reference certain columns in the data set using `dataset$columnname`. With the plot command, you can designate where the data are (in the following example, the x-axis is `Theoph$Time` and the y-axis is `Theoph$conc`. Using `xlab` and `ylab` sets the axes labels, and `pch` chooses the marker type (seen here (page 32)).

```
> setwd("~/Desktop")
> png(filename="theophscatter.png")
> plot(Theoph$Time, Theoph$conc, xlab="Time (min)", ylab="Conce
ntration (M)", pch=16)
> title("Pharmacokinetics of Theophylline")
> dev.off()
quartz
     2
```



A scatterplot created using the Theoph dataset.

Figure: Marker types

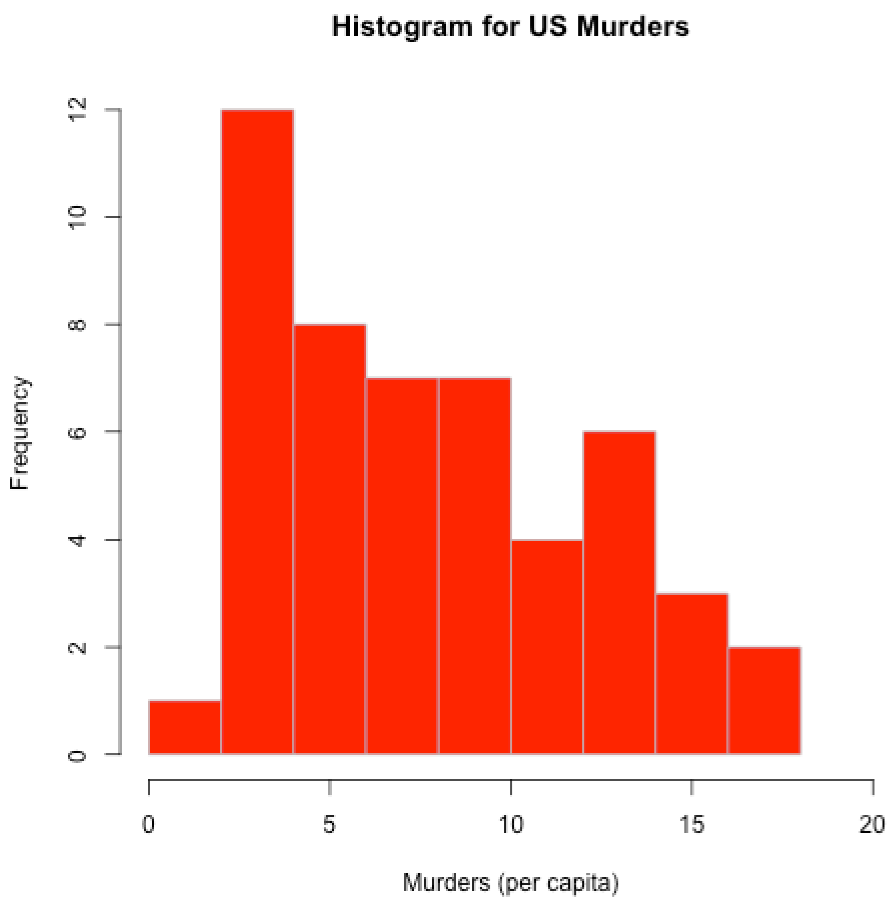


Different marker options available, with the default equal to `pch=1`.

Histograms

A histogram provides a visual depiction of a dataset's distribution. This example uses the dataset USArrests, which contains information on the violent crime rates by US state.

```
> setwd("~/Desktop")
> png(filename="hist_murder.png")
> hist(USArrests$Murder,main="Histogram for US Murders",xlab="M
urders (per capita)",border="gray",col="red",xlim=c(0,20))
> dev.off()
null device
          1
```



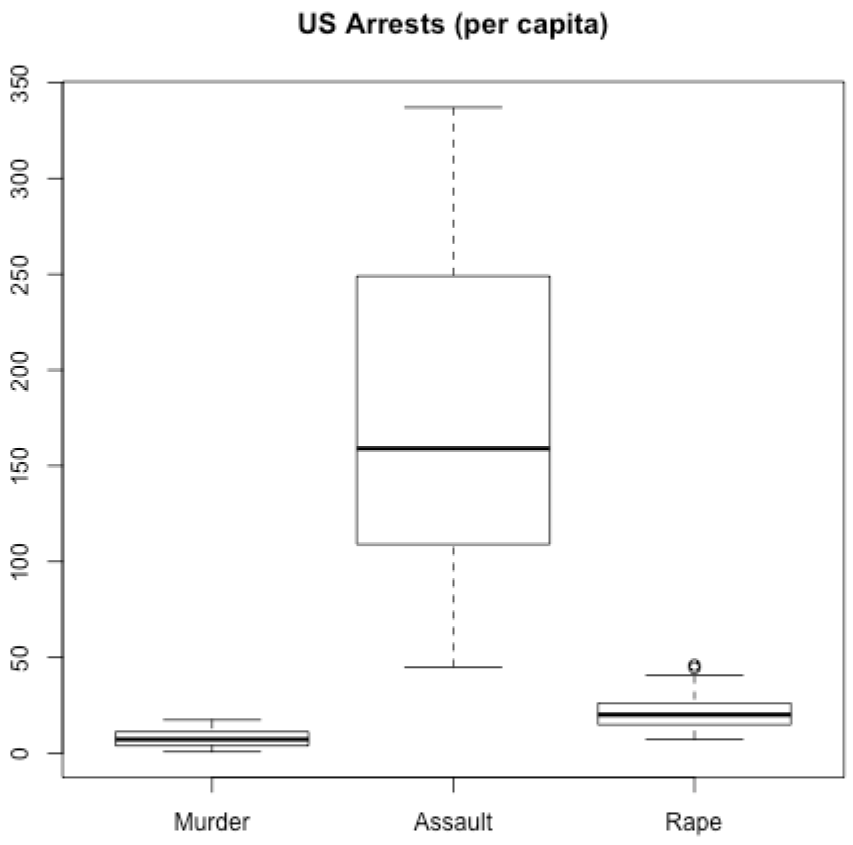
A histogram of the Murder column from the USArrests dataset.

Because there were multiple columns of information, the `Data$Column` option was used to specify which column the histogram should be created from. The title was set using `main=`, colors for the border (`border="gray"`) and columns (`col="red"`) were set, and the x-axis boundries were chosen as 0-20 (`xlim=c(0,20)`). None of the options were necessary, except the initial `Data\$Column` specification. There are also more options for histograms found in the documentation, including adding breaks (which can organize how data is grouped) or density curves.

Boxplots

Boxplots are one way of graphically orienting information from the `summary(variable)` (page 26) command.

```
> setwd("~/Desktop")
> png("arrests_boxplot.png")
> boxplot(USArrests$Murder,USArrests$Assault,USArrests$Rape, na
mes=c("Murder","Assault","Rape"),main="US Arrests (per capit
a)")
> dev.off()
null device
          1
```



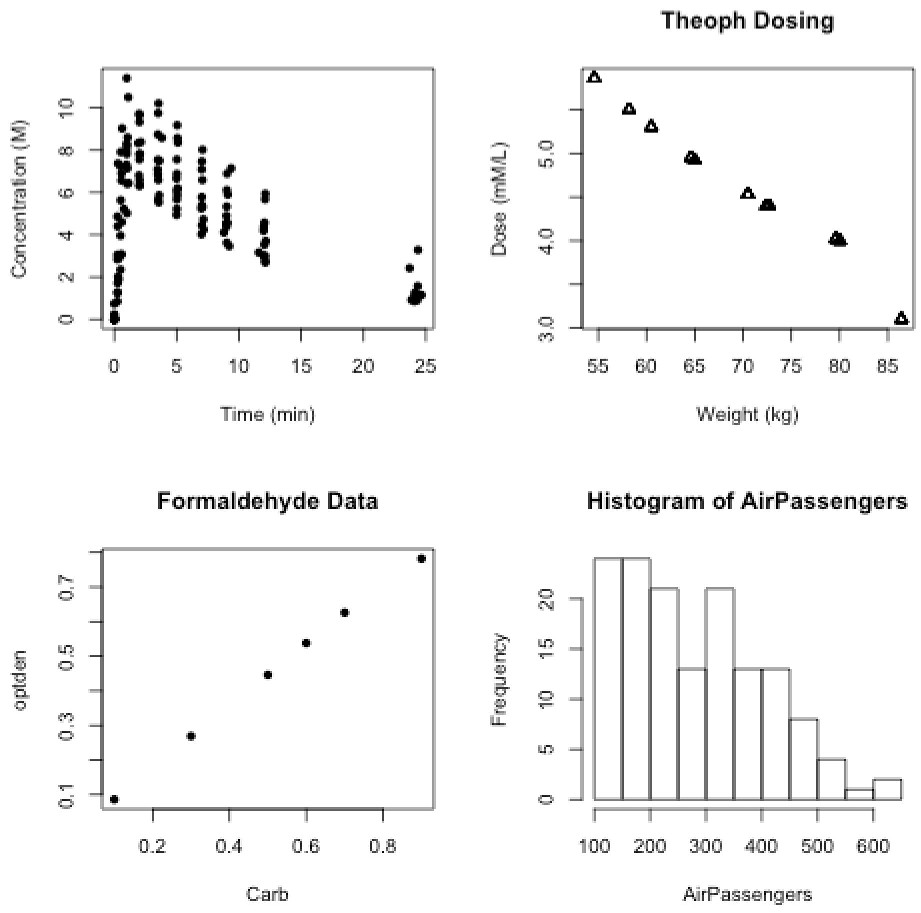
A boxplot depicting the Murder, Assault, and Rape columns from the USArrests dataset.

Because specific columns of data were selected, their individual names had to be specified to create the image above. Otherwise, the default names of 1, 2, and 3, would have been used.

Multiple Graphs

R lets you graph multiple data sets in one composed figure. This is done through the `par` command. Using `par`, you can specify a matrix (in the examples case, 2x2), where images are added from top left to top right, bottom left to bottom right. Each one can be formatted as previously described. The example used three different sets of pre-loaded data, and the very first graph is the only one without a specified title.

```
> setwd("~/Desktop")
> png(filename="random_data_R.png")
> par(mfrow=c(2,2))
> plot(Theoph$Time, Theoph$conc, xlab="Time (min)", ylab="Conce
ntration (M)", pch=16)
> plot(Theoph$Wt, Theoph$Dose, xlab="Weight (kg)", ylab="Dose
(mM/L)", pch=2, main="Theoph Dosing")
> plot(Formaldehyde, xlab="Carb", ylab="optden", pch=16, mai
n="Formaldehyde Data")
> hist(AirPassengers)
> dev.off()
quartz
     2
```



A random dataset, created using `par`.

Hypothesis Testing Overview

What would a statistics software be without hypothesis testing? Answer: Not a statistics software. Anyway, there are several options for hypothesis testing, which are all better explained in a stats class. So let's get on with the examples. First, we'll create our fake data set (which would be something we'd eventually want to test for Hardy-Weinberg equilibrium).

```
>Input =("
Genotype Observed Expected
AA            200          160
Aa            400          360
aa            400          480
")

>Matrix = as.matrix(read.table(textConnection(Input),
                   header=TRUE,
                   row.names=1))
> Matrix
   Observed Expected
AA      200      160
Aa      400      360
aa      400      480
```

Fisher's Exact Test

Using our fake data, we'll run Fisher's exact test. This test is typically used in place of the chi-square on small samples, but it is valid for all sample sizes.

Hypotheses

- H_0 = *in HWE*
- H_a = *not in HWE*

```
>fisher.test(Matrix)

        Fisher's Exact Test for Count Data

data:  Matrix
p-value = 0.0009992
alternative hypothesis: two.sided
```

Since the p-value is less than α (0.05 at the 95% level), we reject the (H_0) null hypothesis in favor of the alternative (H_a). There is enough evidence to say that the fake data is not in HWE.

Chi-Square Test

Using our fake data, we'll run a chi-square test. This test is used to evaluate how likely there is to be an observed difference between sets.

Hypotheses

- H_0 = *in HWE*
- H_a = *not in HWE*

```
> chisq.test(Matrix)

        Pearson's Chi-squared test

data:  Matrix
X-squared = 13.822, df = 2, p-value = 0.0009965
```

Since the p-value is less than α (0.05 at the 95% level), we reject the (H_0) null hypothesis in favor of the alternative (H_a). There is enough evidence to say that the fake data is not in HWE.

Logistic Regression

Logistic regression is a statistical model where the independent variable is continuous and the dependent variable is binary. Some binary variables of interest would be health/sick, pass/fail, win/lose, or alive/dead.

R has a function to use the general linearized model to perform logistic regression. You're not going to need that fake data set here, but you will need a data set with several rows of individual data, specified by columns. Perhaps something like this:

```
StudentID Passing Age Tutor TotalCredits Gender matched_sets
1234       1       32   1       178        0        100
1235       0       12   0       12         1        300
1236       1       25   0       0          2        200
1237       1       24   1       123        0        200
1238           1       34   0       293        0        100
1239       0       15   1       25         1        300
```

In the future example, `Passing` will be referred to as `STATUS` (where 1 = yes, 0 = no), and the columns to the right of `Passing` refer to the dummy variables A through D. Similarly, the data is stratified into the `matched_sets` strata. There can be more columns in the data, and the columns can be in any order; any formulas will reference columns by their header.

In the following example, the result of the test for the general linearized model is saved under `logistical`. The syntax for the `glm` command is thoroughly explained through using `help(glm)`. A quick summary is that it tests a formula of specified data columns against a specific distribution model, such as binomial or Gaussian (`family =`).

```
> data = read.table("/path/to/text/file/with/data", header=TRU
E, na.strings = "NA")
> logistical <- glm(data$STATUS ~ data$VariableA + data$Variabl
eB + data$VariableC + data$VariableD, family = binomial)
> summary(logistical)
Call:
glm(formula = data$STATUS ~ data$VariableA + data$VariableB + d
ata$VariableC +
    data$VariableD, family = binomial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6684  -0.8207  -0.5998   1.1268   2.0336

Coefficients:
                 Estimate Std. Error z value Pr(>|z|)
(Intercept)    -2.5341507  0.5566591  -4.552 5.30e-06 ***
data$VariableA  0.0041581  0.0008622   4.823 1.42e-06 ***
data$VariableB -0.0190579  0.0252128  -0.756  0.44972
data$VariableC  0.3291734  0.2707949   1.216  0.22414
data$VariableD  0.8224185  0.2691547   3.056  0.00225 **
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 366.95  on 293  degrees of freedom
Residual deviance: 330.86  on 289  degrees of freedom
  (1 observation deleted due to missingness)
AIC: 340.86

Number of Fisher Scoring iterations: 3
```

The level of significance for the *p*-value is given by the number of asterisks. Three asterisks means that the *p*-value for that result is below 0.001, but larger than 0. Significant results allow the null hypothesis to be rejected, and the significance code specifies whether this is done at the 90% (.), 95% (*) , 99% (**), or 99.9% (***) level.

Conditional Logistic Regression

Conditional logistic regression is like logistic regression, but it can take stratification and matching into account.

One form of conditional logistic regression can be performed by loading the `Epi` package. An additional form can be performed by loading the `survival` package.

This example uses the `Epi` package. The individual components of the `clogistic` command syntax have been shown through the equals signs. The `data$` is not necessary because the data variable is specified in the equation (which was an option for `glm`). Conditional logistic regression requires the use of stratified data, where the column of the strata is specified (here as `matched_sets`). Unlike before, the definition must be entered after defining the test; summary will not work.

```
> library(Epi)
> data = read.table("/path/to/text/file/with/data", header=TRU
E, na.strings = "NA")
> EPI.clogistic <- clogistic(formula = STATUS ~ VariableA + Var
iableB + VariableC + VariableD, strata = matched_sets, data = d
ata)
> EPI.clogistic

Call:
clogistic(formula = STATUS ~ VariableA + VariableB + VariableC
+ VariableD,
    strata = matched_sets, data = data)




                  coef exp(coef) se(coef)      z       p
VariableA     -0.01002      0.99  0.03261 -0.307 7.6e-01
VariableB      0.50098      1.65  0.36442  1.375 1.7e-01
VariableC      0.15458      1.17  0.27884  0.554 5.8e-01
VariableD      0.00486      1.00  0.00112  4.343 1.4e-05

Likelihood ratio test=26.5  on 4 df, p=2.51e-05, n=292
```

This example uses the `survival` package. It has two distinct differences from Epi's `clogit`. Firstly, `strata` are included into the formula, as opposed to being a second parameter. Second, the summary function can be used.

```
> library(survival)
> data = read.table("/path/to/text/file/with/data", header=TRU
E, na.strings = "NA")
> survival.clogit <- clogit(formula = STATUS ~ VariableA + Vari
ableB + VariableC + VariableD + strata(matched_sets) data = dat
a)
> summary(survival.clogit)
Call:
coxph(formula = Surv(rep(1, 295L), STATUS) ~ VariableA + Variab
leB + VariableC +
   VariableD + strata(matched_sets), data = data, method = "exa
ct")

  n= 294, number of events= 93
   (1 observation deleted due to missingness)

                   coef exp(coef)  se(coef)      z Pr(>|z|)
VariableA     -0.010025  0.990025  0.032612 -0.307    0.759
VariableB      0.500983  1.650342  0.364415  1.375    0.169
VariableC      0.154577  1.167164  0.278843  0.554    0.579
VariableD      0.004864  1.004876  0.001120  4.343 1.41e-05 ***
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

          exp(coef) exp(-coef) lower .95 upper .95
VariableA     0.990     1.0101    0.9287     1.055
VariableB     1.650     0.6059    0.8079     3.371
VariableC     1.167     0.8568    0.6757     2.016
VariableD     1.005     0.9951    1.0027     1.007

Rsquare= 0.086   (max possible= 0.494 )
Likelihood ratio test= 26.5   on 4 df,   p=2.511e-05
Wald test            = 21.21  on 4 df,   p=0.0002873
Score (logrank) test = 24.77  on 4 df,   p=5.607e-05
```

The level of significance for the *p*-value is given by the number of asterisks. Three asterisks means that the *p*-value for that result is below 0.001, but larger than 0. Significant results allow the null hypothesis to be rejected, and the significance code specifies whether this is done at the 90% (.), 95% (*) , 99% (**), or 99.9% (***) level.

Testing for Gene-Environment Interaction

Gene-environment interaction is when different genotypes have different responses to variation in the environment. To test for this, the data set needs genotypic and phenotypic information to be combined, which can be done using R's `cbind` function. If you're the type of person that like combining data in Excel, however, then you can get started once your input file looks clean.

Logistic Regression GxE

The only difference when performing logistic regression for gene by environment with the original is the addition of the terms of interest multiplied by the environment variable that you're testing.

The example tests the environmental `VariableD` on variables A through C (sometimes shortened like `VA`).

```
> data = read.table("/path/to/text/file/with/data", header=TRU
E, na.strings = "NA")
> logisticalGE <- glm(data$STATUS ~ data$VariableA + data$Varia
bleB + data$VariableC + data$VariableD + data$VariableA*data$Va
riableD + data$VariableB*data$VariableD + data$VariableC*data$V
ariableD, family = binomial)
> summary(logisticalGE)

Call:
glm(formula = data$STATUS ~ data$VariableA + data$VariableB + d
ata$VariableC +
    data$VariableD + data$VariableA * data$VariableD + data$Var
iableB *
    data$VariableD + data$VariableC * data$VariableD, family =
binomial)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.7205  -0.8537  -0.5189   1.0470   2.5061

Coefficients:
                            Estimate Std. Error z value P
r(>|z|)
(Intercept)               -4.5268619  1.0737534  -4.216 2.49e-0
5 ***
data$VariableA        0.1690201  0.0530564   3.186 0.001444 **
data$VariableB        0.5921538  0.5478683   1.081 0.279772
data$VariableC        0.5843189  0.5787536   1.010 0.312679
data$VariableD        0.0136132  0.0038047   3.578 0.000346 ***
data$VA:data$VD      -0.0007583  0.0001849  -4.101 4.12e-05 ***
data$VB:data$VD      -0.0012611  0.0018368  -0.687 0.492361
data$VC:data$VD      -0.0003966  0.0018996  -0.209 0.834636
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 366.95  on 293  degrees of freedom
Residual deviance: 311.15  on 286  degrees of freedom
  (1 observation deleted due to missingness)
AIC: 327.15

Number of Fisher Scoring iterations: 5
```

The level of significance for the *p*-value is given by the number of asterisks. Three asterisks means that the *p*-value for that result is below 0.001, but larger than 0. Significant results allow the null hypothesis to be rejected, and the significance code specifies whether this is done at the 90% (.), 95% (*) , 99% (**), or 99.9% (***) level.

Conditional Logistic Regression GxE

Again, like logistic regression's gene by environment interaction, the only change for conditional logistic regression is the addition of the terms of interest multiplied by the environment variable being tested.

The example tests the environmental `VariableD` on variables A through C (sometimes shortened like `VA`).

```
> survival.clogitGE <- clogit(formula = STATUS ~ VariableA+ Var
iableB + VariableC + totalanth + VariableA*VariableD + Variable
B*VariableD + VariableC*VariableD + strata(matched_sets),data)
> summary(survival.clogitGE)
Call:
coxph(formula = Surv(rep(1, 295L), STATUS) ~ VariableA + Variab
leB + VariableC +
   VariableD + VariableA * VariableD + VariableB * VariableD +
VariableC *
    VariableD + strata(matched_sets), data = data, method = "ex
act")

  n= 294, number of events= 93
   (1 observation deleted due to missingness)

                      coef  exp(coef)   se(coef)      z P
r(>|z|)
VariableA        1.776e-01  1.194e+00  5.938e-02  2.991 0.00278
1 **
VariableB        2.608e-01  1.298e+00  7.080e-01  0.368 0.71259
8
VariableC        1.764e-01  1.193e+00  5.686e-01  0.310 0.75643
3
VariableD        1.230e-02  1.012e+00  4.091e-03  3.005 0.00265
2 **
VA:VD           -7.187e-04  9.993e-01  1.988e-04 -3.616 0.00029
9 ***
VB:VD           -6.979e-05  9.999e-01  2.076e-03 -0.034 0.97318
1
VC:VD           -7.048e-04  9.993e-01  1.976e-03 -0.357 0.72126
4
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

             exp(coef) exp(-coef) lower .95 upper .95
VariableA      1.1943     0.8373    1.0631    1.3418
VariableB      1.2980     0.7704    0.3240    5.1991
VariableC      1.1929     0.8383    0.3914    3.6356
VariableD      1.0124     0.9878    1.0043    1.0205
VA:VD          0.9993     1.0007    0.9989    0.9997
VB:VD          0.9999     1.0001    0.9959    1.0040
VC:VD          0.9993     1.0007    0.9954    1.0032

Rsquare= 0.133   (max possible= 0.494 )
Likelihood ratio test= 41.97  on 7 df,   p=5e-07
```

```
Wald test            = 27.16  on 7 df,   p=3e-04
Score (logrank) test = 36.31  on 7 df,   p=6e-06
```

The level of significance for the *p*-value is given by the number of asterisks. Three asterisks means that the *p*-value for that result is below 0.001, but larger than 0. Significant results allow the null hypothesis to be rejected, and the significance code specifies whether this is done at the 90% (.), 95% (*) , 99% (**), or 99.9% (***) level.

PLINK Data Files

PLINK is a free computational package that can be used for genome association analyses.

Data for PLINK should be in PED and MAP files. PED files are space or tab delimited files, and need to start with the following 6 columns: `Family ID, Individual ID, Maternal ID, Paternal ID, Sex,` and `Phenotype`. Sex should be coded with 1=male; 2=female; and any other value meaning unknown. MAP files describe markers, and should contain only four columns: `chromosome` (1-22, X, Y, or 0 if unplaced), `rs\# or SNP ID`, `Genetic distance` (in the morgan unit), and `Base-pair position` (bp units).

There is more information about these file types on the PLINK website .

PLINK Run Information

Some of the summary statistics that PLINK can generate (through different commands) include missing genotype rate (missingness), Hardy-Weinberg equilibrium, minor allele frequency, and linkage disequilibrium. There are many more things PLINK can do (including family-based association testing for disease traits), which are all further described on the PLINK website .

PLINK Job Script

The following script would run the PLINK commands found in the `plink_frq` file on a SLURM scheduler.

```
#!/bin/bash
#SBATCH -p public                # partition aka allocation
#SBATCH --qos general            # quality-of-service (priority)

module load  plink/1.07

./plink_frq
```

If you get an error code, try `source plink_frq` instead of `./plink_frq`. The error dependent on where you're sourcing the file from.

Tests

A sample PLINK information file, `plink_frq`, specifying what PLINK needs to run to test for minor allele frequency (MAF) is below.

```
# no web stops PLINK from updating before run;
# path needs to include folder and the name for the ped and map
plink --noweb --file /path/to/PED/and/MAP/files \
      --nonfounders \ # all individuals included
      --allow-no-sex \ # prevents setting phenotypes with "ambi
guous" sex to missing
      --freq # actual test
```

Where the actual test is will be changed for each different test. If the path were `~/home/euid123/R_jobs/` and the `.ped` and `.map` were both named example, then the line would be `~/home/euid123/R_jobs/example \`.

Table: Test specifics for PLINK

Specifier	Test or Function	Test Statistic
`--freq`	Minor Allele Frequency	MAF
`--het`	Heterozygosity	F Value
`--hardy`	Hardy Weinberg Equilibrium	HWE
`--r2`	Linkage Disequilibrium	r^2
`--out snps`	Linkage Disequilibrium	r^2
`--missing --mind 1`	Missingness	F_MISS
`--recodeAD`	Recode	NA

- **Frequency**: test for minor allele frequency.
- **Heterozygosity**: test for inbreeding coefficients.
- **Hardy-Weinberg Equilibrium**: test for Hardy-Weinberg equilibrium.
- **Linkage Disequilibrium**: test for linkage disequilibrium.
- **Missingness**: test for missingness.
- **Recode**: change data coding to additive and dominance components.

Using awk on Data

The `awk` Unix command (page 0) can be used to parse out specified data. The following command would create the `plinkawk.frq` file from the generated `plink.frq` file, preserving the header, for values in column 5 that are less than 0.05.

```
awk 'NR == 1; NR > 1 {if ($5<0.05) print}' plink.frq > plinkaw
k.frq
```