and

RStudio

Help Book

by **Todd Partridge**
and
Kady Schneiter, Ph.D.

UtahStateUniversity©

Spring 2020

AggieMathLearningCenter

TABLE OF CONTENTS

DOWNLOADING R, AND RSTUDIO

To download and install R:

- Open your web browser.
- Go to https://www.r-project.org.
- Under "Getting Started", click the link that says "download R".
- From the list of mirrors, scroll down to USA, and click on the first link.
- Choose the link for your corresponding computer operating system.
    - If you clicked Windows, click the link labeled "install R for the first time", then click the top link on the page.
    - If you clicked Mac, click on the first link found under "Latest release:".
- Once the file is downloaded, open the file, and go through the installation instructions.

This booklet was written under the premise that users are also using RStudio, which is a platform that makes using R more straightforward for new users.

To download and install RStudio:

- Open your web browser.
- Go to https://www.rstudio.com/products/rstudio/download.
- Click on the "Download" button found under "RStudio Desktop".
- Under "Installers for Supported Platforms", choose the link for your corresponding computer operating system.
- Once the file is downloaded, open the file, and go through the installation instructions.

Now that you have R and RStudio downloaded on your computer, any time you wish to use R, just click on the RStudio program, and it will open R for you in the more straightforward platform.

BASIC ARITHMETIC

Here is a basic list of arithmetical operations that can be used in R:

" + " is used for addition.

```
> 4+5
[1] 9
```

" - " is used for subtraction.

```
> 5-4
[1] 1
```

" * " is used for multiplication.

```
> 4*5
[1] 20
```

" / " is used for division.

```
> 5/4
[1] 1.25
```

" ^ " creates exponents.

```
> 4^5
[1] 1024
```

" ( ) " are used in the normal fashion. HOWEVER, parentheses are not used as a multiplication sign. If characters are found directly before an open parenthesis, R interprets those characters as the name of a function.

```
> 4*(5+1)
[1] 24
> 4(5+1)
Error: attempt to apply non-function
```

" sqrt( ) " is used to find the square root of a number. For other custom roots, write it out like an exponent.

```
> sqrt(16)
[1] 4
> 16^(1/4)
[1] 2
```

" pi " can be used to call the value of the irrational number π.

```
> 2*pi
[1] 6.283185
```

" exp( ) " is used to exponentiate numbers. It raises the irrational number *e* to whatever is found in the parentheses.

```
> exp(2)
[1] 7.389056
```

" log( ) " is used to take the natural log of numbers. So, this is, in fact, the *ln*, and not log_{10}. For base 2 or base 10, you can use " log2( ) " or " log10( ) " respectively. For other bases, you will input " log(x, base=y) ", where x is the value and y is the new base.

```
> log(10)
[1] 2.302585
> log(exp(1))
[1] 1
> log10(10)
[1] 1
> log(25, base=5)
[1] 2
```

" sin( ) ", " cos( ) ", and " tan( ) " are the sine, cosine, and tangent functions, respectively. Angles input are to be in radians, not degrees.

```
> sin(pi/4)
[1] 0.7071068
> cos(pi/4)
[1] 0.7071068
> tan(pi/4)
[1] 1
```

" asin( ) ", " acos( ) ", and " atan( ) " are the arcsine, arccosine, and arctangent, respectively. Angles output are in radians, not degrees.

```
> asin(sqrt(2)/2)
[1] 0.7853982
> acos(sqrt(2)/2)
[1] 0.7853982
> atan(1)
[1] 0.7853982
```

VARIABLES

Creating Variables

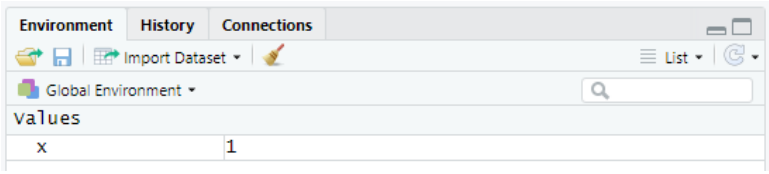
You can create variables to hold all sorts of values, characters, lists, matrices, functions, or even entire data sets. There are other sections in this booklet that go into these in more depth. Here, we will discuss basic creation.

" = " or " <- " are used as assignment operators in R. " <- " is more common, as it avoids confusion with how " = " is often perceived in other circles. The assignment operator assigns the value on the right of the operator to the variable on the left of the operator. For instance, " x = 1 " assigns the value 1 to the variable x.

When you create a variable, the console will not return anything, even if it worked:

```
> x <- 1
> x = 1
>
```

However, you will see in the top right section of RStudio that a variable has appeared:



This shows that there is a variable x with value 1.

Naming Variables

Variables can have just about any name you can think of, as long as they follow these few rules:

1. The variable name is any combination of letters, digits, periods, and underscores.
2. The variable name begins with a letter or a period.
3. If the variable name begins with a period, it cannot be followed by a digit.
4. The variable name is not already being used by R for something else. (In many cases, this rule can be broken, but can confuse your code later.)

Using Variables

Once you've created a variable, using it is as simple as typing the name of the variable. That variable name now calls forth whatever information you put into it. For instance, with our "x=1" example, we see the following:

```
> 5+x
[1] 6
```

Altering Variables

A variable's value can be changed at any time (this includes many pre-programmed variables, such as "pi", so be careful) using the same assignment operators used for creating variables. Remember, the assignment operator does not denote equality. Thus, we can write something like "x=x+1", which might not make sense mathematically, but in R, it simply assigns "x" a value that is 1 greater than whatever it was previously:

```
> x<-1
> x=x+1
> x
[1] 2
```

Deleting Variables

If you want to delete a variable, simply use the " rm( ) " function.

```
> rm(x)
> x
Error: object 'x' not found
```

VECTORS AND MATRICES

Vectors and matrices are objects that hold more than one piece of information. These can be saved as variables with some dynamic properties.

Creating Vectors

Vectors are also often referred to as "lists." The simplest way to create a vector is to use the concatenate function " c( ) ".

```
> vector1 <- c(1,2,3,4,5)
```

The variable "vector1" now represents a vector of five numbers, 1 through 5. A simpler way to evoke a list of consecutive numbers is with a colon:

```
> vector2 <- c(1:5)
```

The sequence function " seq( ) " can also be useful to create a more specific sequence of numbers. For instance, you can define where to begin, where to end, and how large the steps should be:

```
> vector3 <- seq(from=1,to=5,by=1)
```

You can also use " seq( ) " to define where to begin, where to end, and how many steps you want to take to get there:

```
> vector4 <- seq(from=1,to=5,length.out=5)
```

In any case, you would now see these vectors listed in the top right window of the RStudio application like so:

Values	
vector1	num [1:5] 1 2 3 4 5
vector2	int [1:5] 1 2 3 4 5
vector3	num [1:5] 1 2 3 4 5
vector4	num [1:5] 1 2 3 4 5

As you can see, all four vectors are one-by-five tables of numbers, 1 through 5. The only difference between them is that vector1, vector3, and vector4 are vectors of numbers, and vector2 is a vector of integers. The reason for this is because when we used the colon feature, R recognized that everything coming into the vector was going to have an integer value. However, the other vectors simply received different values that were fed individually into the vector, so R made no such assumption about those values.

Creating Matrices

A basic matrix is fundamentally a "list of lists" in R. We can create a basic matrix by creating several lists and binding them together. We can use the row-bind function " rbind( ) " or the column-bind function " cbind( ) " to do this:

```
> matrix1 <- rbind(c(1,2,3),c(4,5,6),c(7,8,9))
> matrix2 <- cbind(c(1,2,3),c(4,5,6),c(7,8,9))
> matrix1
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> matrix2
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

As you can see, the row-bind function took each list and made it a row in the matrix. The column-bind function took each list and made it a column in the matrix. We can also do this with vectors we've saved as variables:

```
> matrix3 <- cbind(vector1,vector2,vector3,vector4)
> matrix3
     vector1 vector2 vector3 vector4
[1,]       1       1       1       1
[2,]       2       2       2       2
[3,]       3       3       3       3
[4,]       4       4       4       4
[5,]       5       5       5       5
```

In this case, since each column already had a name when input, the matrix shows those names in the output.

Searching Vectors and Matrices

The most straightforward way to look at different parts of a vector or matrix are with square brackets. Typing the name of a vector or matrix followed by square brackets lets R know that you are looking at a particular place in the list. For instance, typing " vector1[2] " will show whatever is the second item in vector1.

```
> vector1[2]
[1] 2
```

You can use a colon to look at several consecutive items:

```
> vector1[2:4]
[1] 2 3 4
```

In a matrix, you need to define both the row and the column you wish to look at. For instance, in this matrix,

```
> matrix2
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

R should return " 8 " if we look at the 2nd row, 3rd column.

```
> matrix2[2,3]
[1] 8
```

If you leave the row or column space empty, it will return all rows or all columns, respectively. So, matrix2[,2] should show all the rows, but only column 2:

```
> matrix2[,2]
[1] 4 5 6
```

We can also use the " which( ) " function to search out rows or columns that fit certain criteria. For instance, if you wanted to see every row in matrix2 where the first number was greater than 1, you would put " which(matrix2[,1]>1) " in the row space (thus picking out every row in which the first column is a number greater than 1), and leave the column space blank (thus showing every column in that row, ultimately displaying entire rows that fit the criteria):

```
> matrix2[which(matrix2[,1]>1),]
     [,1] [,2] [,3]
[1,]    2    5    8
[2,]    3    6    9
```

You can use the following comparison operators in the " which( ) " function:

- " x > y " x is greater than y.
- " x < y " x is less than y.
- " x >= y " x is greater than or equal to y.
- " x <= y " x is less than or equal to y.
- " x == y " x is equal to y.
- " x != y " x is not equal to y.

Editing Vectors and Matrices

Each individual value in a vector or matrix can be edited as if it were its own variable. For instance, consider matrix33, a 3x3 matrix with a "3" in each cell:

```
> matrix33
     [,1] [,2] [,3]
[1,]    3    3    3
[2,]    3    3    3
[3,]    3    3    3
```

If you wanted to change the middle cell of this matrix to a 9, you would simply input the following:

```
> matrix33[2,2] <- 9
> matrix33
     [,1] [,2] [,3]
[1,]    3    3    3
[2,]    3    9    3
[3,]    3    3    3
```

You can edit an entire list of numbers simultaneously through basic arithmetic operations:

```
> vector1
[1] 1 2 3 4 5
> vector1+2
[1] 3 4 5 6 7
> vector1^2
[1]  1  4  9 16 25
```

You can even use the which function as shown in the previous section to edit entries that fit certain criteria. For instance, we can change any value in vector1 which is less than 3 to a 0 instead:

```
> vector1[which(vector1<3)] <- 0
> vector1
[1] 0 0 3 4 5
```

Sorting Vectors and Matrices

To sort elements in a vector, use the " sort( ) " function. This works on numeric variables,

```
> unsortedVector
[1] 7 3 1 5 2 6 4
> sortedVector <- sort(unsortedVector)
> sortedVector
[1] 1 2 3 4 5 6 7
```

as well as character variables:

```
> unsortedList
[1] "Georgia"  "Delaware" "Utah"     "Colorado"
> sortedList <- sort(unsortedList)
> sortedList
[1] "Colorado" "Delaware" "Georgia"  "Utah"
```

To sort elements in a matrix, use the " order( ) " function. As an example, if you wanted to sort the rows of a matrix by the values in the first column, you would input "matrixName[order(matrixName[,1]),]":

```
> unsortedMatrix
     [,1] [,2] [,3]
[1,]    4    2    1
[2,]    8    7    3
[3,]    2    6    5
[4,]    3    5    7
[5,]    6    1    2
[6,]    1    4    4
[7,]    7    3    6
[8,]    5    8    8
> sortedMatrix <- unsortedMatrix[order(unsortedMatrix[,1]),]
> sortedMatrix
     [,1] [,2] [,3]
[1,]    1    4    4
[2,]    2    6    5
[3,]    3    5    7
[4,]    4    2    1
[5,]    5    8    8
[6,]    6    1    2
[7,]    7    3    6
[8,]    8    7    3
```

BASIC FUNCTIONS

Arithmetic Functions

Many of the functions discussed previously can be applied to vectors and matrices as well as single numbers. These include " sqrt( ) ", " log( ) ", " exp( ) ", and the trigonometric functions. For example:

```
> squaresVector
[1]  1  4  9 16 25
> sqrt(squaresVector)
[1] 1 2 3 4 5
> log(squaresVector)
[1] 0.000000 1.386294 2.197225 2.772589 3.218876
```

There are many other functions that will arithmetically manipulate values as well. A few are:

" abs(x) "	Absolute value of x.
" ceiling(x) "	The integer found just after a decimal number x.
" floor(x) "	The integer found just before a decimal number x.
" round(x, digits=n) "	Rounds x to n digits after the decimal.
" signif(x, digits=n) "	Keeps n significant digits of x.

For example:

```
> decimalvector
[1] 1.234 5.678 9.012 3.579 2.222
> ceiling(decimalvector)
[1]  2  6 10  4  3
> floor(decimalvector)
[1] 1 5 9 3 2
> round(decimalvector, digits=1)
[1] 1.2 5.7 9.0 3.6 2.2
```

There are also many functions that will perform arithmetic operations on a list of values and return one result value. Two of the most common are " sum( ) " and " mean( ) ":

```
> oddvector
[1] 1 3 5 7 9
> sum(oddvector)
[1] 25
> mean(oddvector)
[1] 5
```

You can also nest functions within other functions with relative ease:

```
> sqrt(sum(oddvector))
[1] 5
```

Probability Distribution Functions

There are many functions you can use in R that return values of interest regarding different probability distributions. Here are the most frequently used functions:

Distribution	Function	What It Does
Binomial Distribution	dbinom(x,n,p)	Gives the probability of x successes in a binomial distribution with n trials and probability of success p.
	pbinom(x,n,p)	Gives the probability of x or fewer successes in a binomial distribution with n trials and probability of success p.
	qbinom(q,n,p)	Gives the number of successes at the q^{th} percentile in a binomial distribution with n trials and probability of success p.
Chi-Square Distribution	dchisq(x,n)	Gives the probability of getting the chi-square statistic x in a chi-square distribution with n degrees of freedom.
	pchisq(x,n)	Gives the probability of getting a chi-square statistic of x or less in a chi-square distribution with n degrees of freedom.
	qchisq(q,n)	Gives the chi-square statistic at the q^{th} percentile in a chi-square distribution with n degrees of freedom.
F Distribution	df(x,n,m)	Gives the probability of getting the F statistic x in an F distribution with n and m degrees of freedom.
	pf(x,n,m)	Gives the probability of getting an F statistic of x or less in an F distribution with n and m degrees of freedom.
	qf(q,n,m)	Gives the F statistic at the q^{th} percentile in an F distribution with n and m degrees of freedom.

Geometric Distribution	dgeom(x,p)	Gives the probability of getting x successes in a geometric distribution with probability of success p.
	pgeom(x,p)	Gives the probability of getting x or fewer successes in a geometric distribution with probability of success p.
	qgeom(q,p)	Gives the number of success at the q^{th} percentile in a geometric distribution with probability of success p.
Normal Distribution	dnorm(x,μ,σ)	Gives the probability of getting the value x in a normal distribution with mean μ and standard deviation σ.
	pnorm(x,μ,σ)	Gives the probability of getting a value of x or less in a normal distribution with mean μ and standard deviation σ.
	qnorm(q,μ,σ)	Gives the value at the q^{th} percentile in a normal distribution with mean μ and standard deviation σ.
Poisson Distribution	dpois(x,μ)	Gives the probability of x occurrences in a Poisson distribution with mean μ.
	ppois(x,μ)	Gives the probability of x or fewer occurrences in a Poisson distribution with mean μ.
	qpois(q,μ)	Gives the number of occurrences at the q^{th} percentile in a Poisson distribution with mean μ.
t Distribution	dt(x,n)	Gives the probability of getting the t-statistic x in a t-distribution with n degrees of freedom.
	pt(x,n)	Gives the probability of getting a t-statistic of x or less in a t-distribution with n degrees of freedom.
	qt(q,n)	Gives the t-statistic at the q^{th} percentile in a t-distribution with n degrees of freedom.

Creating Functions

You may want to create your own function to quickly perform actions you will be having R do regularly. You can use the " function( ) { } " function to create a function. In the parentheses, you will input names for variables the user will need to use when they call the function. In the curly brackets, you will input the code you want the function to perform. In the code within the curly brackets, if you want the function to return a value, you must have the last line before the end curly bracket be " return( ) " with whatever you want returned in the parentheses.

For example, perhaps you will be changing a lot of temperatures from Fahrenheit to Celsius. You can create a function that does this with the following:

```
> FtoC <- function(Ftemp) {
+     Ctemp <- (Ftemp-32)*(5/9)
+     return(Ctemp)
+ }
```

Note: When inputting code to R, you can create new lines without having R evaluate the code by pressing Shift+Enter.

Note: R will automatically create a new line (denoted with a " + " symbol) if the current line is clearly not complete when you press Enter. This will often happen if you miss an end-parenthesis, an end-bracket, etc.

Now, with our new " FtoC( ) " function, you can easily convert temperatures from Fahrenheit to Celsius:

```
> FtoC(72)
[1] 22.22222
```

As another example, suppose you wanted to create a function that calculated the n^{th} root of any number. You could do the following:

```
> nthroot = function(val,root) {
+     solution = val^(1/root)
+     return(solution)
+ }
```

Now, to find the 4^{th} root of 16, you can use your newly created function:

```
> nthroot(16,4)
[1] 2
```

DATASETS

Datasets have very similar qualities to matrices, with some extra properties. R has many data sets built in when you download the program. We will use the " iris " dataset to explore those extra properties.

Dataset Variable Names

The columns of a dataset often will have names for easier referencing. Use the " names( ) " function to show the names of a dataset:

```
> names(iris)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
[5] "Species"
```

While the columns of a dataset can be called in the way you would call the column of a matrix, it can also be called via the " $ " operator followed by the variable name:

```
> iris[,2]
  [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4 3.9
 [18] 3.5 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2
 [35] 3.1 3.2 3.5 3.6 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3.2 3.7 3.3 3.2
 [52] 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7 2.0 3.0 2.2 2.9 2.9 3.1 3.0 2.7
 [69] 2.2 2.5 3.2 2.8 2.5 2.8 2.9 3.0 2.8 3.0 2.9 2.6 2.4 2.4 2.7 2.7 3.0
 [86] 3.4 3.1 2.3 3.0 2.5 2.6 3.0 2.6 2.3 2.7 3.0 2.9 2.9 2.5 2.8 3.3 2.7
[103] 3.0 2.9 3.0 3.0 2.5 2.9 2.5 3.6 3.2 2.7 3.0 2.5 2.8 3.2 3.0 3.8 2.6
[120] 2.2 3.2 2.8 2.8 2.7 3.3 3.2 2.8 3.0 2.8 3.0 2.8 3.8 2.8 2.8 2.6 3.0
[137] 3.4 3.1 3.0 3.1 3.1 3.1 2.7 3.2 3.3 3.0 2.5 3.0 3.4 3.0
> iris$Sepal.Width
  [1] 3.5 3.0 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 3.7 3.4 3.0 3.0 4.0 4.4 3.9
 [18] 3.5 3.8 3.8 3.4 3.7 3.6 3.3 3.4 3.0 3.4 3.5 3.4 3.2 3.1 3.4 4.1 4.2
 [35] 3.1 3.2 3.5 3.6 3.0 3.4 3.5 2.3 3.2 3.5 3.8 3.0 3.8 3.2 3.7 3.3 3.2
 [52] 3.2 3.1 2.3 2.8 2.8 3.3 2.4 2.9 2.7 2.0 3.0 2.2 2.9 2.9 3.1 3.0 2.7
 [69] 2.2 2.5 3.2 2.8 2.5 2.8 2.9 3.0 2.8 3.0 2.9 2.6 2.4 2.4 2.7 2.7 3.0
 [86] 3.4 3.1 2.3 3.0 2.5 2.6 3.0 2.6 2.3 2.7 3.0 2.9 2.9 2.5 2.8 3.3 2.7
[103] 3.0 2.9 3.0 3.0 2.5 2.9 2.5 3.6 3.2 2.7 3.0 2.5 2.8 3.2 3.0 3.8 2.6
[120] 2.2 3.2 2.8 2.8 2.7 3.3 3.2 2.8 3.0 2.8 3.0 2.8 3.8 2.8 2.8 2.6 3.0
[137] 3.4 3.1 3.0 3.1 3.1 3.1 2.7 3.2 3.3 3.0 2.5 3.0 3.4 3.0
```

So, if you wanted to see the information on all the irises with a sepal width of 3.0 or greater, you could use the " which( ) " function in the following manner:

```
> iris[which(iris$Sepal.Width>=4), ]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
15          5.8         4.0          1.2         0.2  setosa
16          5.7         4.4          1.5         0.4  setosa
33          5.2         4.1          1.5         0.1  setosa
34          5.5         4.2          1.4         0.2  setosa
```

You can use the " attach( ) " function to save all the variable names in a data set as vector names. If you did this with the iris dataset, you could then refer to sepal widths by simply typing the name " Sepal.Width " rather than typing out " iris$Sepal.Width ". This can save a lot of time in the long run if you will be referring to different parts of the data set often:

```
> attach(iris)
> iris[which(Sepal.Width>=4), ]
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
15          5.8         4.0          1.2         0.2  setosa
16          5.7         4.4          1.5         0.4  setosa
33          5.2         4.1          1.5         0.1  setosa
34          5.5         4.2          1.4         0.2  setosa
```

NUMERICAL DATA SUMMARY

The most basic numerical summaries often used in R are finding the mean, median, standard deviation, or variance of a certain list of numbers, or two find the correlation between two lists of numbers. For our examples, we will use the " iris " dataset as we have before:

```
> mean(iris$Sepal.Width)
[1] 3.057333
> median(iris$Sepal.Width)
[1] 3
> sd(iris$Sepal.Width)
[1] 0.4358663
> var(iris$Sepal.Width)
[1] 0.1899794
> cor(iris$Sepal.Width,iris$Sepal.Length)
[1] -0.1175698
```

You can also use the " summary( ) " function for the "five-number summary" of a list of numbers, as well as the mean, all at once:

```
> summary(iris$Sepal.Width)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   2.800   3.000   3.057   3.300   4.400
```

However, if you use the " summary( ) " function on a qualitative/categorical variable, it will return a list of the categories, and how many of each category there are:

```
> summary(iris$Species)
    setosa versicolor  virginica
        50         50         50
```

GRAPHICAL DATA SUMMARY

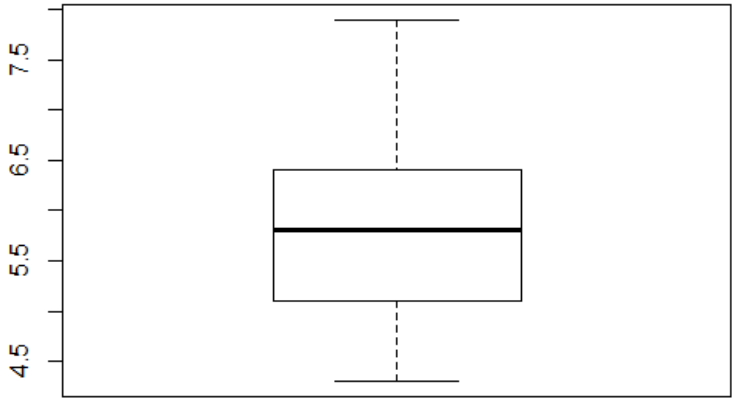
There is no limit to the ways you can summarize data graphically, and R can make it happen with relative ease. This guide will go over *very basic* visualizations that can organize data in a meaningful way. For our examples, we will use the " iris " dataset as we have before.

Numerical Data

Boxplots

Feeding a list of numbers into the " boxplot( ) " function will create a boxplot by graphically displaying the five-number summary of the data, along with any lower or upper outliers, if applicable.

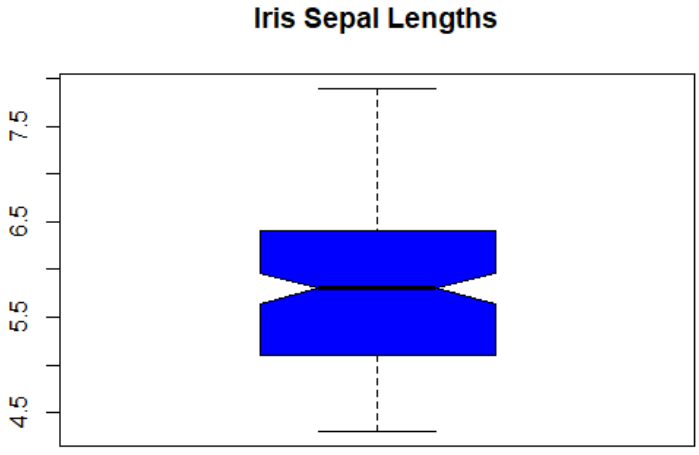
```
> boxplot(iris$Sepal.Length)
```



Additional Variables Used in " boxplot( ) ":

- notch — Creates a notch at the median if set to " TRUE ".
- varwidth — Sets width of boxplot proportional to sample size if set to " TRUE ".
- names — Using " c( ) ", create a list of names for each boxplot in the graphic.
- main — The name of the overall graphic.
- xlab — The name of the x axis.
- ylab — The name of the y axis.
- col — Using " c( ) ", create the list of colors to be used for the boxplots.

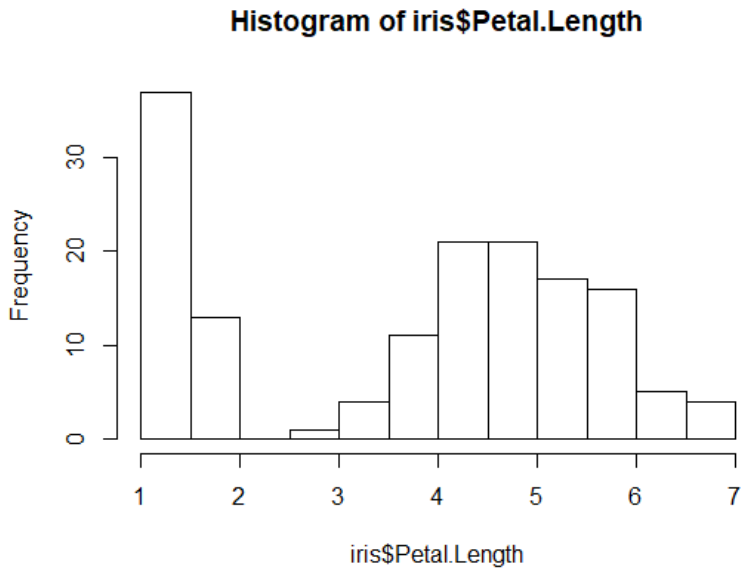
```
> boxplot(iris$Sepal.Length, notch=TRUE, main="Iris Sepal Lengths", col="blue")
```



Histograms

With the " hist( ) " function, you can create a histogram from a list of numbers.

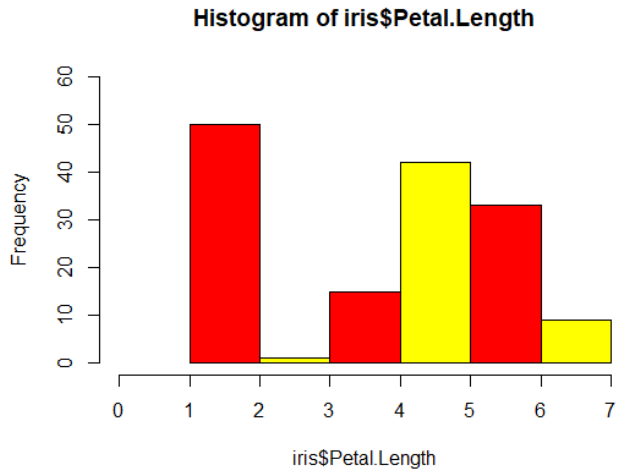
```
> hist(iris$Petal.Length)
```



Additional Variables Used in " hist( ) ":

- breaks — A number can be used here to specify the number of desired bars.
- xlim — Using " c( ) ", insert the first and last value for the x axis.
- ylim — Using " c( ) ", insert the first and last value for the y axis.
- main — The name of the overall graphic.
- xlab — The name of the x axis.
- ylab — The name of the y axis.
- col — Using " c( ) ", create a sequence of colors to be used for the bars.

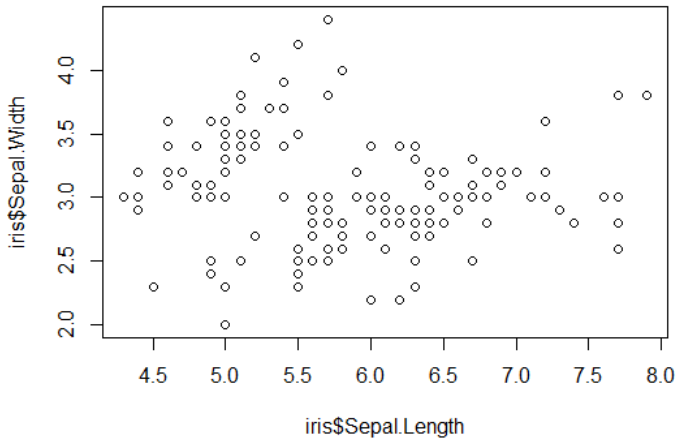
```
> hist(iris$Petal.Length, breaks=8, xlim=c(0,7), ylim=c(0,60), col=c("red","yellow"))
```



Scatterplots

Inputting two lists of numbers (of the same length) into " plot( ) " will create a scatterplot of ordered pairs created as the program moves down both lists together.

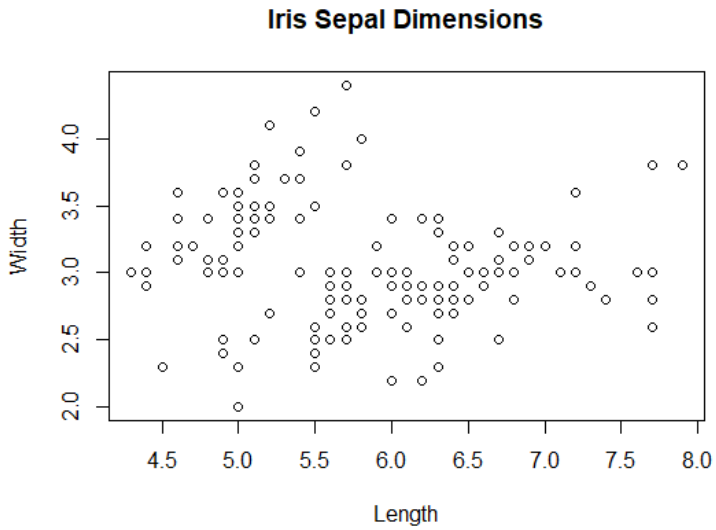
```
> plot(iris$Sepal.Length,iris$Sepal.Width)
```



Additional Variables Used in " plot( ) ":

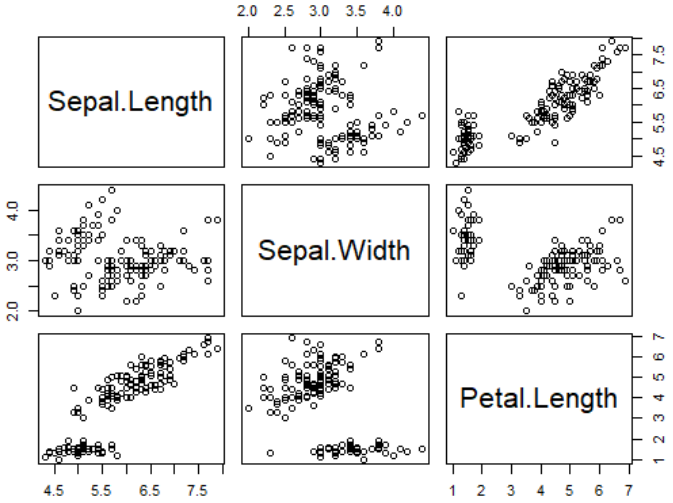
- axes — This will remove the axes if set to " FALSE ".
- xlim — Using " c( ) ", insert the first and last value for the x axis.
- ylim — Using " c( ) ", insert the first and last value for the y axis.
- main — The name of the overall graphic.
- xlab — The name of the x axis.
- ylab — The name of the y axis.
- col — Input a color for the points on the graph to be colored.

```
> plot(iris$Sepal.Length,iris$Sepal.Width, main="Iris Sepal Dimensions", xlab="Width", ylab="Length")
```



You can also create a scatterplot matrix using the " pairs( ) " function. First, enter the regression formula being used with the tilda " ~ " operation. Then, input the name of the dataset where the variables can be found:

```
> pairs(~Sepal.Length+Sepal.Width+Petal.Length, data = iris)
```

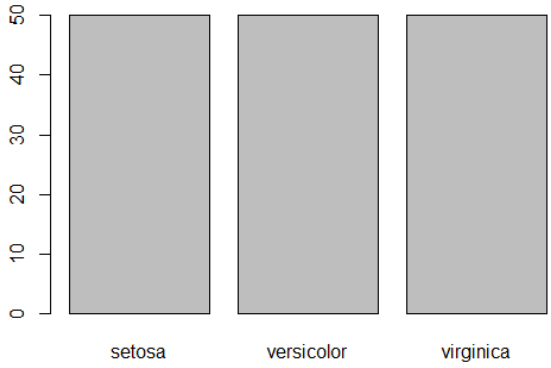


Categorical Data

Bar Charts

Bar charts are perhaps the simplest and most effective way to depict how many subjects belong to each of several classifications. The " barplot( ) " function does not just take a list of category names, however. This function needs a concise *summary* of the categories and how many subjects are within each, which can be accomplished with the " summary( ) " function:

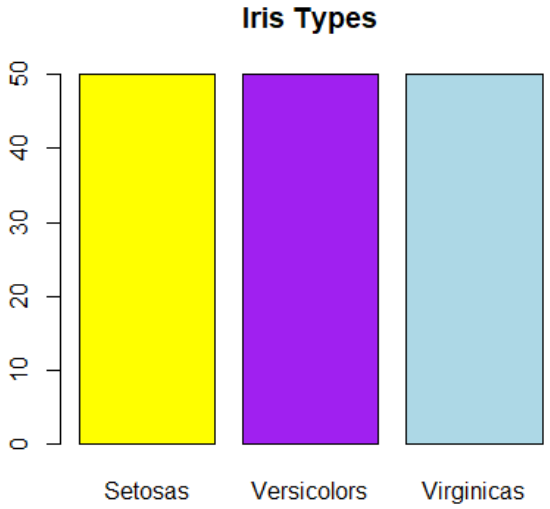
```
> barplot(summary(iris$Species))
```



Additional Variables Used in " barplot( ) ":

- names — Using " c( ) ", create a list of names for each bar in the graphic.
- main — The name of the overall graphic.
- xlab — The name of the x axis.
- ylab — The name of the y axis.
- col — Using " c( ) ", create a sequence of colors to be used for the bars.

```
> barplot(summary(iris$Species), names=c("Setosas","Versicolors","Virginicas"),
+ main="Iris Types", col=c("yellow","purple","lightblue"))
```

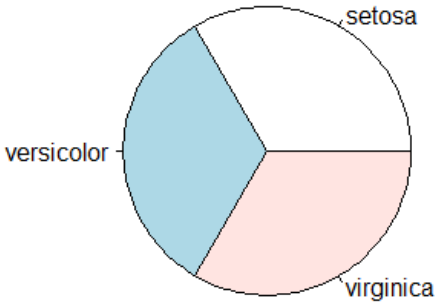


Note: If you have categorical data that has been recorded numerically (for instance, "yes" and "no" recorded as "1" and "0", you will need to let R know to treat the numbers as category names rather than numerical data. You can do this with the " as.factor( ) " function, which will tell R to treat the numbers like factors, or categories. So, if there was a variable in the " iris " dataset called " fullBloom " with 0s and 1s, you could create a bar chart of the data by simply typing " barplot(summary(as.factor(iris$fullBloom))) ".

Pie Charts

The " pie( ) " function also uses summarized data of a categorical variable, as follows:

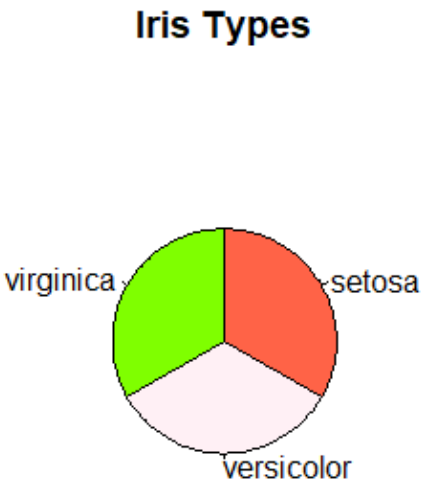
```
> pie(summary(iris$Species))
```



Additional Variables Used in " pie( ) ":

- radius — Input a number between -1 and 1 to change the size of the chart's radius.
- clockwise — If set to " TRUE ", the categories will be listed clockwise on the chart.
- labels — Using " c( ) ", create a list of names for each slice of the pie chart graphic.
- main — The name of the overall graphic.
- col — Using " c( ) ", create a sequence of colors to be used for the slices.

```
> pie(summary(iris$Species), radius=0.5, clockwise=TRUE,
+ main="Iris Types", col=c("tomato","lavenderblush","chartreuse"))
```

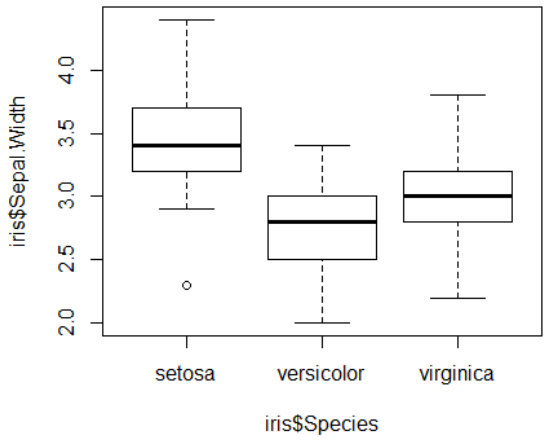


Numerical and Categorical Data

Boxplots

Of the simple graphs shown above, boxplots are the most effective way to compare numerical data across different categories. For instance, with the " iris " dataset, it can be enlightening to create a boxplot for the sepal widths of each kind of iris, and look at them side by side. You can do this by using the " ~ " operation, where " x~y " essentially tells R to look at variable x, but separated into groups according to the y variable.

```
> boxplot(iris$Sepal.Width~iris$Species)
```



INFERENCE

R has many impressive capabilities when it comes to performing statistical inference on datasets. This booklet will go over a couple of the most basic types of inference.

One-Sample t-Tests

To perform a one-sample t-test, you simply input a set of numbers into the " t.test( ) " function. This function will perform a hypothesis test, as well as create a confidence interval, for the population average. The defaults for this function are a two-sided test, with a null hypothesis of $\mu=0$, and a confidence level of 0.95.

```
> t.test(iris$Sepal.Width)

        One Sample t-test

data:  iris$Sepal.Width
t = 85.908, df = 149, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 2.987010 3.127656
sample estimates:
mean of x
 3.057333
```

Additional Variables Used in " t.test( ) ":

- mu — This is the population average, according to the null hypothesis.
- alternative — The alternative hypothesis. Can be " two.sided ", " greater ", or " less ".
- conf.level — A number between 0 and 1 to indicate the confidence level.

```
> t.test(iris$Sepal.Width, mu=3, alternative="greater", conf.level=0.9)

        One Sample t-test

data:  iris$Sepal.Width
t = 1.611, df = 149, p-value = 0.05465
alternative hypothesis: true mean is greater than 3
90 percent confidence interval:
 3.011522      Inf
sample estimates:
mean of x
 3.057333
```

Two-Sample t-Tests

Two-sample t-tests are conducted with the " t.test( ) " function as well, but with two lists of numbers entered instead of one. The defaults for this function are independent samples with assumed different population variances, with the null hypothesis being that there is no difference between the two population means.

```
> vers_Sepal.Width = iris$Sepal.Width[which(iris$Species=="versicolor")]
> virg_Sepal.Width = iris$Sepal.Width[which(iris$Species=="virginica")]
> t.test(vers_Sepal.Width, virg_Sepal.Width)

        Welch Two Sample t-test

data:  vers_Sepal.Width and virg_Sepal.Width
t = -3.2058, df = 97.927, p-value = 0.001819
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.33028364 -0.07771636
sample estimates:
mean of x mean of y
    2.770     2.974
```

Additional Variables Used in " t.test( ) ":

- paired Set this variable to " TRUE " to conduct a paired-sample t-test.
- var.equal Set this variable to " TRUE " to use the pooled-variance procedure.

```
> t.test(vers_Sepal.Width, virg_Sepal.Width, mu=-0.4, var.equal=TRUE)

        Two Sample t-test

data:  vers_Sepal.Width and virg_Sepal.Width
t = 3.08, df = 98, p-value = 0.002686
alternative hypothesis: true difference in means is not equal to -0.4
95 percent confidence interval:
 -0.33028246 -0.07771754
sample estimates:
mean of x mean of y
    2.770     2.974
```

Regression

To calculate the regression line for two possibly correlated variables, use the " lm( ) " function *(which stands for "linear model")*. As it is customary to regress y on x *(defining y as the dependent variable and x as the explanatory variable)*, equations are input into the " lm( ) " function in the form " lm(y~x) ". This function will return the intercept and the slope for the line of best fit.

```
> vers_Sepal.Width = iris$Sepal.Width[which(iris$Species=="versicolor")]
> vers_Petal.Length = iris$Petal.Length[which(iris$Species=="versicolor")]
> lm(vers_Petal.Length~vers_Sepal.Width)

Call:
lm(formula = vers_Petal.Length ~ vers_Sepal.Width)

Coefficients:
     (Intercept)  vers_Sepal.Width
          1.9349            0.8394
```

For more detailed information on the linear model, you can use the " summary( ) " function on the linear model function to get p-values for each of these values *(with the null hypothesis being that the slope and the intercept are 0)*.

```
> summary(lm(vers_Petal.Length~vers_Sepal.Width))

Call:
lm(formula = vers_Petal.Length ~ vers_Sepal.Width)

Residuals:
     Min       1Q   Median       3Q      Max
-1.03337 -0.23337 -0.04321  0.21754  0.89876

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)         1.9349     0.4989   3.878  0.00032 ***
vers_Sepal.Width    0.8394     0.1790   4.689  2.3e-05 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3932 on 48 degrees of freedom
Multiple R-squared:  0.3142,    Adjusted R-squared:  0.2999
F-statistic: 21.99 on 1 and 48 DF,  p-value: 2.302e-05
```

Chi-Square Tests

The " chisq.test( ) " function can perform both the Goodness of Fit Test and the Test of Independence. The function differentiates between the two tests depending on whether it receives a simple list of numbers, or a matrix.

Goodness of Fit

Suppose there is a luxury cruise ship with 424 passengers: 178 men, 155 women, and 91 children. You can put this list of numbers into the " chisq.test( ) " function to perform a Goodness of Fit test.

```
> Men.Women.Children = c(178,155,91)

> chisq.test(Men.Women.Children)

        Chi-squared test for given probabilities

data:  Men.Women.Children
X-squared = 28.759, df = 2, p-value = 5.688e-07
```

The default for the Goodness of Fit Test for the null hypothesis is that all counts should be equal. However, you can change the variable " p " in the function to define a different set of proportions as the null hypothesis distribution.

```
> chisq.test(Men.Women.Children, p=c(0.4,0.4,0.2))

        Chi-squared test for given probabilities

data:  Men.Women.Children
X-squared = 2.1262, df = 2, p-value = 0.3454
```

Test of Independence

Suppose there is a small family cruise ship with 271 passengers: 88 men, 89 women, and 94 children. You can put a matrix containing the types of passengers, found on this boat alongside the types of passengers in the previous example, into the " chisq.test( ) " function to perform a Test of Independence of whether, among these two ships, the type of ship is independent from the distribution of passenger types.

```
> TwoBoats.Men.Women.Children = cbind(c(178,155,91),c(88,89,94))
> chisq.test(TwoBoats.Men.Women.Children)

        Pearson's Chi-squared test

data:  TwoBoats.Men.Women.Children
X-squared = 15.417, df = 2, p-value = 0.0004489
```

ANOVA

To perform a basic one-way ANOVA test in R, use the " aov( ) " function. This function needs a list of numbers, as well as a categorical variable to separate them by, as you would use the " ~ " operator to create side-by-side boxplots.

```
> aov(iris$Petal.Width~iris$Species)
Call:
   aov(formula = iris$Petal.Width ~ iris$Species)

Terms:
                iris$Species Residuals
Sum of Squares       80.41333   6.15660
Deg. of Freedom             2       147

Residual standard error: 0.20465
Estimated effects may be unbalanced
```

For more detailed information on the ANOVA test, you can use the " summary( ) " function on the " aov( ) " function to get a p-value for the corresponding F test.

```
> summary(aov(iris$Petal.Width~iris$Species))
              Df Sum Sq Mean Sq F value Pr(>F)
iris$Species   2  80.41   40.21     960 <2e-16 ***
Residuals    147   6.16    0.04
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

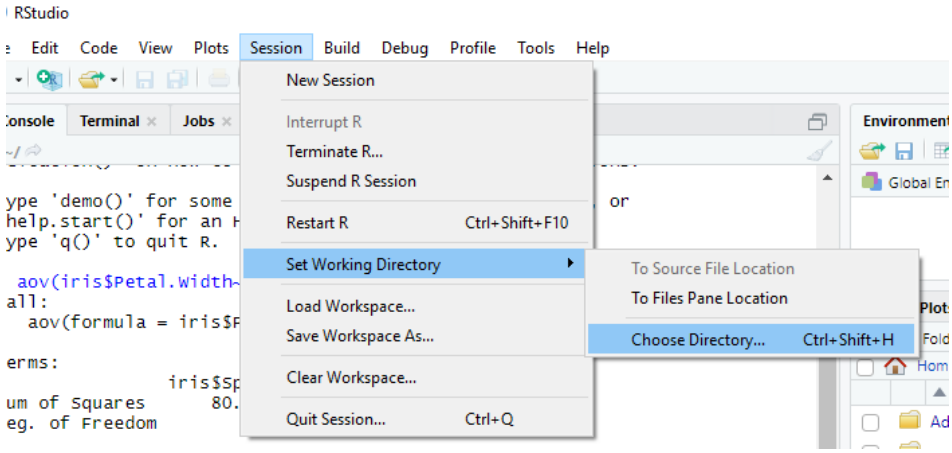
If you've determined you have significant results from your ANOVA test, you can conduct pairwise comparisons by using the " TukeyHSD( ) " function rather than " summary( ) ":

```
> TukeyHSD(aov(iris$Petal.Width~iris$Species))
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = iris$Petal.Width ~ iris$Species)

$`iris$Species`
                       diff       lwr       upr p adj
versicolor-setosa    1.08 0.9830903 1.1769097     0
virginica-setosa     1.78 1.6830903 1.8769097     0
virginica-versicolor 0.70 0.6030903 0.7969097     0
```

READING IN DATA FILES

In order to summarize or analyze data, you need to read it into the R program. There are many ways to do this, but perhaps the simplest are the " read.table( ) " and " read.csv( ) ". These functions have equal purposes, though " read.table( ) " is for .txt files, and " read.csv( ) " is for .csv files.

However, before you can use these functions, you will need to set you working directory to the folder on your computer where the data file can be found. Go to "Session", "Set Working Directory", then "Choose Directory…".



When the file explorer has opened, find the folder containing the .txt or .csv file you want to read into R. When the folder has been selected, click "Open".

Now that the working directory has been set, R will reference this folder when you use either the " read.table( ) " or " read.csv( ) " function.

These functions require you to input the file name, followed by a " TRUE " or " FALSE " statement for the variable " header ". The " header " variable indicates whether or not the first line in the data matrix is a list of variable names or not.

The following are examples of what using these functions might look like:

```
> read.table("airplane_data.txt", header = FALSE)
> read.csv("VolcanoTimes.csv", header = TRUE)
```

HELP IN R

As has been stated, R has many more capabilities than can be explained in this short tutorial. While there are many very helpful places online to find help in coding for R, the R program has a help feature embedded in the software as well.

Function-Specific Help

Use the " ? " operator followed by the name of a function to return the help page on that function:

```
> ?plot
```

Generic X-Y Plotting

Description

Generic function for plotting of R objects. For more details about the graphical parameter arguments, see par.

For simple scatter plots, plot.default will be used. However, there are plot methods for many R objects, including functions, data.frames, density objects, etc. Use methods(plot) and the documentation for these.

Usage

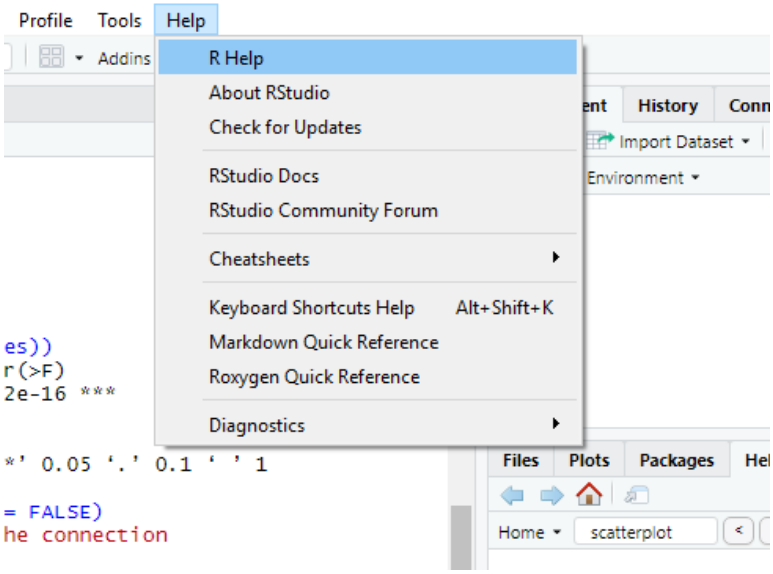
```
plot(x, y, ...)
```

Arguments

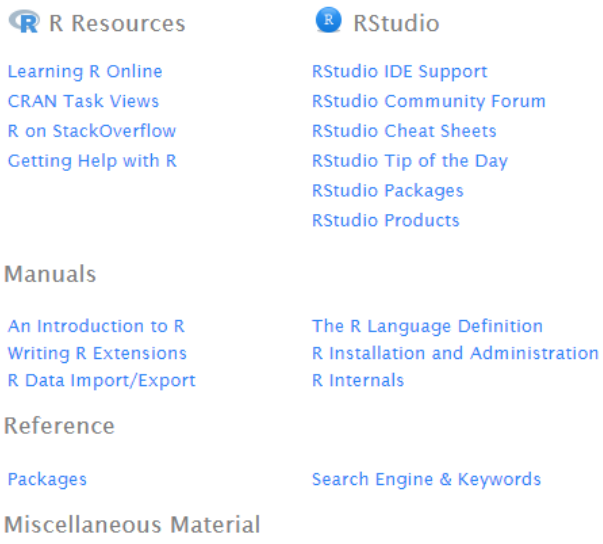
x	the coordinates of points in the plot. Alternatively, a single plotting structure, function or *any R object with a* plot *method* can be provided.
y	the y coordinates of points in the plot, *optional* if x is an appropriate structure.
...	Arguments to be passed to methods, such as graphical parameters (see par). Many methods will accept the following arguments:

General Procedural Help

You can also click " Help " at the top of the RStudio window for more general help.



This will bring up the following window:



There is a lot of useful information here. However, if you need a quick search for how to accomplish a certain task, you can click on " Search Engine & Keywords " and then search for what you are looking for. Also, online sources can also be a great, straightforward help for accomplishing new tasks in R.

REFERENCES: The following are additional references and sources for R Studio

Articles and Books

Adler, J. (2012). *R in a Nutshell: A Desktop Quick Reference.* (2nd edition). O'Reilly Media.

Matloff, N. (2013). *The art of R programming: a tour of statistical software design*. San Francisco: No Starch Press.

Vries, A. de., & Meys, J. (2015). *R For Dummies, 2nd Edition*. John Wiley & Sons.

Wickham, H., & Grolemund, G. (2017). *R for data science*. Beijing: OReilly.

Websites

Modern Dive Website - https://moderndive.netlify.com/1-getting-started.html

R Studios Tutorial - http://web.cs.ucla.edu/~gulzar/rstudio/basic-tutorial.html

R Tutorials Blogsite - https://data-flair.training/blogs/rstudio-tutorial/