

# Software Skills Primer

for



&



## BL Introductory Statistics

Bill Nelson



## Table of Contents

Welcome	4
Primer 1: Getting Started with Excel & R	4
Creating a working directory	5
Let's get started with Excel!	6
Let's get started with R!	8
Primer 2: Working with Data	14
How to structure a dataset	14
Creating the dataset in Excel	15
Loading the dataset into R	16
Primer 3: Calculating descriptive statistics	21
Mean, median, variance & standard deviation	21
Quartiles & interquartile range	23
Primer 4: Graphing	26
Contingency tables	26
Histogram	27
Bar plots	28
Box plots	31
Scatter plots	33
Primer 5: Calculating probabilities	34
Event ranges and probabilities	34
Primer 6: Chi-square tests, T-tests & confidence intervals	36
Confidence intervals for single samples	36
Chi-square tests	37
Single-sample t-tests	40
Paired-sample t-tests	42

Independent two-sample t-tests	45
Primer 7: Correlation & Linear Regression	48
Correlation	48
Linear regression	49
Evaluating assumptions of linear regression	53
Primer 8: Single-factor ANOVA	56
Data frames for ANOVA	56
Fitting the ANOVA model	58
Evaluating the assumptions of ANOVA	60
Group tests using TukeyHSD	62
Group test using contrasts	62
Primer 9: Two-factor ANOVA	64
Creating data frames for ANOVA	64
Fitting the ANOVA model	67
Evaluating the assumptions of ANOVA	70
Group test using contrasts	71
Reference Cards for R	73
Some useful R tips	73
Data functions	74
Arithmetic, indexing and Logic Operators	76
Statistical functions	77
Plotting functions	79
Plotting Options	80

## Welcome

Welcome to the software skills side of the Introductory Statistics course. Here you will learn how to do the statistical analyses and create the plots that you've been learning about in the videos and lectures. The software primers are designed to work hand-in-hand with the videos and assignments, and it is important to keep these elements well connected throughout the course.

We will be using two software programs—Microsoft Excel and R. This is a natural pairing because Excel is an excellent spreadsheet program that is good for working with data, and R is a versatile and powerful software program for statistics and graphing. Both are easy to learn, will serve you well after graduation, and give you software skills for your Resume that are in demand. R and Excel are both available for free, and are required for the in-lab quizzes.

## Primer 1: Getting Started with Excel & R

In this first section you will install Excel & R on your computer, and do some simple calculations.



Excel is a spreadsheet program that is well suited for data management and basic calculations. Since Excel is operating system and year dependent, it is important that you have the most recent version installed on your computer that your operating system will allow (2011 or 2016 for Mac, 2013 for Windows). These versions are free to download from Queen's IT Services at <http://www.queensu.ca/its/Office365-ProPlus>.



R is a computational environment that is used for a wide range of statistical and mathematical analyses. For example, it can be used as a calculator; for creating stunning graphs; run basic and advanced statistics; and for more specialized analyses such as found in bioinformatics. In this course you will learn how to do basic statistics, which will give you a foundation from which to develop more specialized skills in other courses or after graduation. R is supported and developed by academics, has a large number of references and help resources available on-line and in print, and is

free (even after graduation). A big advantage to using R in a statistics course is that it uses a command-line interface, which is ideal for teaching because students and instructors can easily look at the written commands to find errors and to reproduce analyses exactly.

To install R, go to [cran.r-project.org](http://cran.r-project.org) (CRAN stands for the Comprehensive R Archive Network) and click on one of the MacOS X or Windows links in the “Download and Install R” box. Follow the links for your operating system (choose ‘base files’ for Windows users) and download the latest version (‘R-3.3.1-win32.exe’ for Windows users and ‘R-3.3.1.dmg’ for MacOSX users). The downloaded file has an installer in it, so double clicking the file and following the prompts is all you need to do to install R.

## Creating a working directory

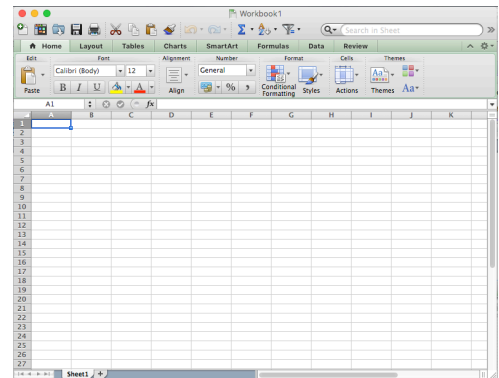
You will need to create a number of files over the course of the term, and you will want to keep these well organized on your computer. We will begin by setting up a dedicated folder on your computer for these files. This will save many hours of retyping your code once we get into the analysis phase of the course. This folder will be used to store data, as well as Excel and R files.

For MacOSX computers, click on *Finder* and navigate to where you would like the folder to be. A good place is under your home directory. Once there, select *File* from the pulldown menu and select *New Folder*. Give that folder a name such as *Statistics Course*. Open the folder by double clicking it. You can add more folders here if you want to organize your files by tutorial week. If you are having trouble creating the folder, Google “OS X Yosemite: Folder basics” to find online tutorials, or ask your TA during tutorial time.

For Microsoft computers, navigate to the location where you want to store your files and right-click. Point to *New* and click on the *Folder* option. Then type the name you want for the folder. You can add more folders inside this one if you want to organize your files by tutorial week. If you are having trouble creating the folder, Google “create a new folder microsoft” to find online tutorials, or ask your TA during tutorial time.

## Let's get started with Excel!

Start Excel as you would any program. The main window that opens on start asks you to select a template. Choose "Excel Workbook", which will create a blank spreadsheet that looks similar to the one shown on the right. The columns are labeled with letters and the rows with numbers. Each cell can be configured to hold text, numbers, or a formula to suit your needs. Let's start by using it as a calculator. In



cells B1 & B2, enter the number 2. In cell B3, enter the formula `=B1+B2`, which adds together the contents of cells B1 & B2. Your spreadsheet should look like the picture shown below as you enter the formula, and give you a value of 4.

	A	B	C
1		2	
2		2	
3		=B1+B2	
4			
5			
6			

You can keep your spreadsheet organized by adding text to adjacent cells. Let's label the first cell 'Apple', the second cell 'Pie' and the result 'Sum'.

	A	B	C
1	Apple	2	
2	Pie	2	
3	Sum	4	
4			
5			
6			

The spreadsheet can be used for a wide range of calculations. Change the 'pie' value to 5, then try dividing (`=B1/B2`) and multiplying (`=B1*B2`) the numbers, and raising one to the power of the other (`=B1^B2`). Your sheet should look something like the following picture.

	A	B	C	D
1	Apple	2		
2	Pie	5		
3	Sum	7		
4	Division	0.4		
5	Multiplication	10		
6	Power	32		
7				

## Working with vectors

A vector is just a list of numbers, and is created in Excel by entering data into a series of cells. We will create our vectors as columns. Here's an example

	A	B	C	D
1		3.2		
2		4.1		
3		5.5		
4		6.2		
5		7.1		
6		8.4		
7		9.5		
8				

We can work with a vector of data using formulas in the adjacent cells. For example, let's subtract 7 from each data point. This is done by entering the formula `=A1-7` into an adjacent cell and copying it down using the square at the bottom right hand side of a highlighted cell. Grab the square with your mouse and drag it down. The following 3 images illustrate the steps.

	A	B
1	3.2	=A1-7
2	4.1	
3	5.5	
4	6.2	
5	7.1	
6	8.4	
7	9.5	
8		

	A	B
1	3.2	-3.8
2	4.1	
3	5.5	
4	6.2	
5	7.1	
6	8.4	
7	9.5	
8		

	A	B
1	3.2	-3.8
2	4.1	-2.9
3	5.5	-1.5
4	6.2	-0.8
5	7.1	0.1
6	8.4	1.4
7	9.5	2.5
8		

We can also compute quantities such as the sum (`=SUM(B1:B7)`) or minimum value (`=MIN(B1:B7)`) of a vector. The round braces indicate that we are using a function, and the 'B1:B7' indicates the range of cells we want the function to be calculated over. See if you get the same answer as shown below for the sum and minimum of the newly calculated vector.

	A	B	C
1	3.2	-3.8	
2	4.1	-2.9	
3	5.5	-1.5	
4	6.2	-0.8	
5	7.1	0.1	
6	8.4	1.4	
7	9.5	2.5	
8			
9	sum	-5	
10	minimum	-3.8	
11			

## Let's get started with R!

Start R as you would any program. You should have a single window open that looks like the image on the right. This is called the console and is the starting window for the R program. When you move your cursor to the console, you should see

```
>
```

This is where commands are entered. Lets try it. Type "2+2" and then press the enter key

```
> 2+2
```

```
R version 3.2.2 (2015-08-14) -- "Fire Safety"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

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

  Natural language support but running in an English locale

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

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

[R.app GUI 1.66 (6996) x86_64-apple-darwin13.4.0]
[Workspace restored from /RData]
>
```

The answer will appear below the line you typed (preceded by "[1]").

```
> 2+2
[1] 4
```

The number in square brackets just indicates that this is the first entry in the vector being returned. Throughout the manual we will use a green background to indicate what you see in the console with green font for text that you enter and black for what R answers back with.

Now let's create some variables. First, create a new variable 'apple' and assign it the value 2

```
> apple=2
```

Then create a new variable 'pie' and assign it the value 5

```
> pie=5
```

Now we can manipulate some of the variables.



```
> apple-pie
[1] -3
> apple/pie
[1] 0.4
> apple*pie
[1] 10
```

If you want to raise something to the exponent, use the '^' symbol

```
> apple^pie
[1] 32
```

A list of common arithmetic operations is found in the 'R Reference Cards' at the end of this manual.

### Working with vectors

A vector is created in R using the *function* `c()`. In R, all functions use round brackets to accept input, with each input entry separated by a comma. To create a vector, all we need to do is to provide the function `c()` with a list of numbers separated by commas.

```
> julie=c(3.2,4.1,5.5,6.2,7.1,8.4,9.5)
```

To see what is a variable, just type its name.

```
> julie
[1] 3.2 4.1 5.5 6.2 7.1 8.4 9.5
```

To access a particular entry in the vector, use square brackets at the end of the name. Here the `[3]` indicates that you want the number in the third spot of the vector.

```
> julie[3]
[1] 5.5
```

Mathematical operations are done directly on vectors—here's some examples:

```
> julie-7
[1] -3.8 -2.9 -1.5 -0.8 0.1 1.4 2.5
> julie-julie
[1] 0 0 0 0 0 0 0
> julie^2
[1] 10.24 16.81 30.25 38.44 50.41 70.56 90.25
```

Notice that the operation is carried out on each entry of the vector independently.

The output from a new operation can also be assigned to a new variable

```
> b=julie-7
```

Type `b` to see what it looks like

```
> b
[1] -3.8 -2.9 -1.5 -0.8  0.1  1.4  2.5
```

## Working with functions

Functions are typed commands that perform a specific task in R. You can pick them out easily because they will be followed by round brackets, such as the function to create a vector shown above `c()`. Some functions that are useful for summarizing information about a vector are `sum()`, `mean()`, `min()` and `max()`. Spaces within a function have no influence, but R is very picky about commas and whether or not letters are capitalized.

Here's an example using the `b` vector

```
> sum(b)
[1] -5
> min(b)
[1] -3.8
```

R commands are case sensitive, so the following code works

```
> mean(julie)
[1] 6.285714
```

but the following do not work

```
> Mean(julie)
Error: could not find function "Mean"
> mean(Julie)
Error in mean(Julie) : object 'Julie' not found
```

There are an enormous number of functions in R, and we will just scratch the surface in this course. One function that's particularly useful when it comes to selecting which objects to sample is the `sample()` function. This function returns a random subset of the numbers you provide. For example, this line will return a random set of 10 numbers between 1 and 100. Try it for yourself.

```
> sample(1:100,10)
[1] 56  8  99 100  85  91  79  20  49  59
```

The `1:100` statement is a shorthand way to get a vector of integers from 1 to 100. Try typing it in the console for yourself.

## Let's get plotting

R can be used to generate some of the nicest graphs of any statistical software. The ease with which R can make plots will allow you to explore your data, check it for errors, visualize the data before analyzing it, and check assumptions of the statistical analyses. In this section, we will create a simple line plot to illustrate what can be done. Let's create a plot that looks at the change in life expectancy in Canada. Our typical approach will be to load the data from a file (covered in Primer 2), but for this first lab we will enter data directly into the console. Start by creating a new 'year' and 'age' vector. To save time, you can highlight the commands below in your PDF reader, then copy and paste them into R.

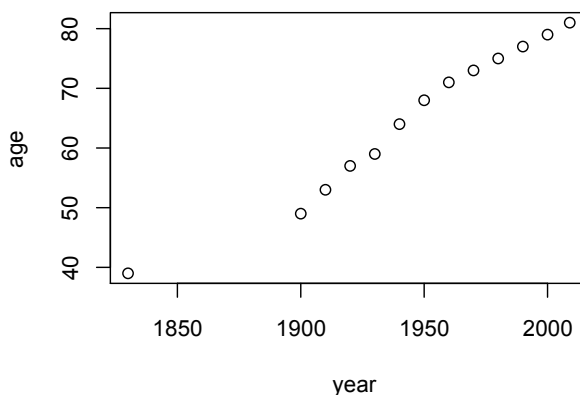
Remember that the `>` symbol is the prompt, so don't include that when copying the text.

```
> year=c(1830, 1900, 1910, 1920, 1930, 1940, 1950, 1960, 1970,1980,
1990, 2000, 2009)
> age=c(39, 49, 53, 57, 59, 64, 68, 71, 73, 75, 77, 79, 81)
```

Line plots are created using the `plot()` function, which requires the arguments X and Y, where X is a vector of all x-axis values you want to plot, and Y is a vector of the y-axis values. The first entry of the X vector corresponds with the first entry of the Y vector, and so on. A basic plot is created by typing

```
> plot(year,age)
```

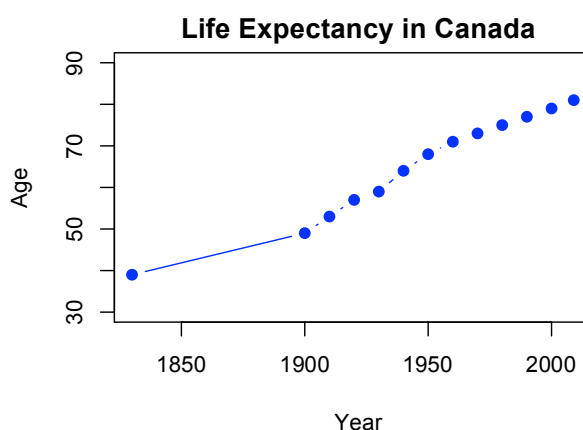
The resulting plot shows an increase in life span through time, but it's not a very nice looking figure.



In Primer 4 we will learn how to work with plotting options. As an illustration, the following command will generate a more complete figure

```
> plot(year, age, type='b',pch=19,xlab="Year", ylab="Age",
col="blue",ylim=c(30,90), main="Life Expectancy in Canada")
```

This looks a bit more complicated, but some of the meanings in the plot command can be figured out from context (e.g. `xlab` and `ylab` let you set the x and y axis labels). Others like `type` and `pch` are less obvious. Don't worry—we will gradually introduce and explain each of these options in the upcoming primers.



## The editor

Now that we have covered some functions and tools in R, we will want to learn to use the editor. The editor allows you to save your instructions for submitting with assignments, and gives you a way to save commands so that you don't have to create them from scratch each time. The editor is also linked to the console, so we can 'submit' our work to the console right from the editor. To start a new editor file:

- MacOSX - click on 'File' from the menu bar and select 'New Document'
- Windows OS, click on 'File' and select 'New Script'

Save this file (it will have a '.r' ending) to the folder where you want to save your work and give it a name to help you remember the content. Now you can type your code, which is called a 'script'. It's a good idea to add comments using the `#` symbol to remind yourself of what each line does. The `#` symbol tells R to ignore the text that follows on the same line, which means you can submit it along with your code without causing an error. Here's an example R script; notice that the prompts are shown in the console but not in the editor.

```
#This file creates a plot of life expectance through time

#Year data
year=c(1830, 1900, 1910, 1920, 1930, 1940, 1950, 1960, 1970, 1980,
1990, 2000, 2009)

#Life expectancy data
age=c(39, 49, 53, 57, 59, 64, 68, 71, 73, 75, 77, 79, 81)

#Plot
plot(year,age,type='b',pch=19,xlab="Year",ylab="Life
Expectancy",col="blue",ylim=c(30,90))
```

To submit the script in MacOSX, select the text with your mouse (or place your cursor anywhere inside the text), hold the command key and press return; in Windows OS select the text with your mouse, hold the control key and press the 'R' key.

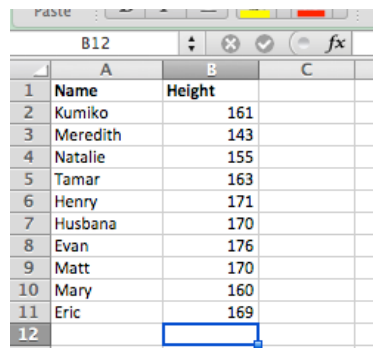
*You should always use the editor for the tutorials rather than the console!*

## Primer 2: Working with Data

The last primer was an introduction to the two software programs we are using in the course. In this section, we are going to learn how to structure our data, enter it into a spreadsheet in Excel, and load it into R.

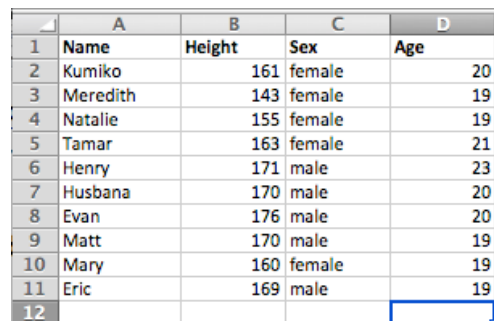
### How to structure a dataset

Throughout the course we will be entering a wide range of data types. As you will learn over the next week or so, this includes both numerical and categorical types of information. To make it easy to work with different types of data and different analyses, all of our datasets will be structured in what is called *long-form* format. In long-form format, each row is a different observation unit, and each column tells us something different about that single unit. For example, if you measure the height (cm) of 10 classmates the dataset would look like this image.



	A	B	C
1	Name	Height	
2	Kumiko	161	
3	Meredith	143	
4	Natalie	155	
5	Tamar	163	
6	Henry	171	
7	Husbana	170	
8	Evan	176	
9	Matt	170	
10	Mary	160	
11	Eric	169	
12			

If you measured additional attributes for each person, then each new attribute becomes a new column. For example, if we also had data on sex (male, female) and age (years), the dataset would look like this image.



	A	B	C	D
1	Name	Height	Sex	Age
2	Kumiko	161	female	20
3	Meredith	143	female	19
4	Natalie	155	female	19
5	Tamar	163	female	21
6	Henry	171	male	23
7	Husbana	170	male	20
8	Evan	176	male	20
9	Matt	170	male	19
10	Mary	160	female	19
11	Eric	169	male	19
12				

Or, if your classmates were part of a study on the effect of exercise level (low, moderate, high) on resting metabolic rate (calories per day), the long-form format would look like this image.

	A	B	C	D	E	F
1	Name	Height	Sex	Age	ExerciseLevel	MetabolicRate
2	Kumiko	161	female	20	low	1520
3	Meredith	143	female	19	moderate	1704
4	Natalie	155	female	19	high	1942
5	Tamar	163	female	21	high	1832
6	Henry	171	male	23	high	1709
7	Husbana	170	male	20	low	1508
8	Evan	176	male	20	low	1499
9	Matt	170	male	19	moderate	1791
10	Mary	160	female	19	moderate	1732
11	Eric	169	male	19	moderate	1563

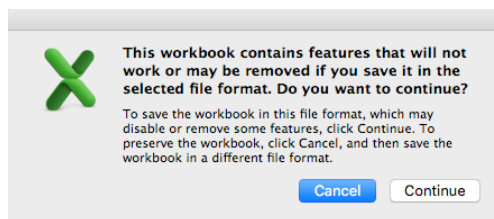
Notice that some columns contain text (categorical variables) and some columns contain numbers (numerical variables); numerical and categorical variables are easily combined in the long-form format. The key here is that each row is a different person, or more generally a different observation unit.

## Creating the dataset in Excel

To create the dataset in Excel, start by opening a new workbook and entering data in a blank sheet. Here are several things to keep in mind:

1. The first row of the sheet is for column names.
2. Column names should start with a letter, not have spaces and contain only letters and numbers. For a more descriptive title, use 'camel case' where each word starts with a capital letter without any spaces (e.g., 'CamelCase').
3. Indicate the units of measurement (e.g., Finger length (cm)) in the text but not in the data sheet.
4. Data start on the second row, and each line is for a different observation unit.
5. If you have missing data for an observation unit, enter the letters NA.

Once you have entered the data in the proper long-form format, then save it in a convenient working folder as a comma separated file (.csv). This type of file contains just the data with the formatting details in an Excel file removed. Comma separated files are straightforward to upload into R, and they avoid any issues with the version of Excel that you are using. To save your file as a comma separated file, click on the "File" menu and select "Save as...". Choose a location to save the file, and then select "Comma Separated Values (.csv)" from the "Format" pull down menu. Then click "Save". You will see the following announcement



click “Continue” and the file will be saved. Choose a name that will help you keep things organized; here I have used the file name ‘metabolic’. If you have entered data on more than one Excel sheet, only the active sheet will be saved in the .csv file. Create a .csv file of the dataset shown above that we can use to load into R.

## Loading the dataset into R

Now that the dataset has been formatted in long-form and saved as a .csv file, it can be loaded into R. Open R and create a new script (make sure you are using the editor and not the console for entering R commands!). To load the dataset that you just created into R, we first need to setup a working directory. This is a directory where you can save your R-scripts, as well as any figures that you create for the tutorial activities. Follow these steps:

1. Create a new folder on your computer as shown in Primer 1 and give it a name. You can call it whatever you like; I’ve called mine ‘Statistics Course 2016’. It is easiest to put the folder in your *user* (OSX) or *home* directory (Windows).
2. In R, manually change the working directory to your ‘Statistics Course 2016’ folder using the `setwd()` function. This function tells R where to load data from and needs to be added to your R script. For example

```
setwd("~/Statistics Course 2016")
```

will tell R to load data from the ‘Statistics Course 2016’ folder. The tilde (~) in the file pathway works differently depending on your operating system. For Apple computers, it represents the *user* directory. For Windows, it represents the home directory. If you created your file somewhere else, then just put the full path name here. Remember to put quotes around the ‘path’ to the folder you have created. You can confirm that the working directory is set properly using the `getwd()` function. Give it a try!

3. Load the .csv file into R using the `read.csv()` function. The `read.csv('MyFileName')` command reads in a .csv file with the name



'MyFileName'. Here's what the start of a typical script will look like

```
#Load a data set
setwd("~/Statistics Course 2016")
MyData=read.csv("metabolic.csv") #load the file metabolic.csv
```

You can name the dataset anything you like; I've used 'MyData' for simplicity. These files are called *data frames* in R.

**Viewing your data** The next step is to look at your dataset in R. This is a very important step, and you can catch a lot of typographical errors simply by looking at the data. There are a number of ways to look at the data. If the dataset is small, just type the name into the console to see the entire thing. Remember that the console has the `>` symbol and runs commands instantly, whereas the script is saved and needs to be 'submitted' to run.

```
> MyData
```

	Name	Height	Gender	Age	ExerciseLevel	MetabolicRate
1	Kumiko	161	female	20	low	1520
2	Meredith	143	female	19	moderate	1704
3	Natalie	155	female	19	high	1942
4	Tamar	163	female	21	high	1832
5	Henry	171	male	23	high	1709
6	Husbana	170	male	20	low	1508
7	Evan	176	male	20	low	1499
8	Matt	170	male	19	moderate	1791
9	Mary	160	female	19	moderate	1732
10	Eric	169	male	19	moderate	1563

If the dataset is large, you can just look at the first six rows of data using the `head()` function, or the last six rows using the `tail()` function. Looking at the first six rows is a good check that the data are lined up properly in the columns, and looking at the last six rows is a good check for blank lines that may have snuck into the dataset. Here's an example of what the `head()` function looks like

```
> head(MyData)
```

	Name	Height	Gender	Age	ExerciseLevel	MetabolicRate
1	Kumiko	161	female	20	low	1520
2	Meredith	143	female	19	moderate	1704
3	Natalie	155	female	19	high	1942
4	Tamar	163	female	21	high	1832
5	Henry	171	male	23	high	1709
6	Husbana	170	male	20	low	1508

Another way to view large datasets is to look at each column individually. To access the columns, use the `$` symbol and the name of the column that wrote in the .csv file. If you have forgotten, you can see the column names by using the `names()` command

```
> names(MyData)
[1] "Name"           "Height"         "Gender"         "Age"
"ExerciseLevel" "MetabolicRate"
```

then access a particular column using

```
> MyData$MetabolicRate
[1] 1520 1704 1942 1832 1709 1508 1499 1791 1732 1563
```

**Ensuring correct column type** The final issue we need to be aware of is how R interpreted the dataset. The file can have a mix of categorical and numerical data, but only one type is allowed per column. If you have mixed data types in a single column, then the dataset is not set up correctly in Excel. R is generally good at figuring out if you intend a column to be numerical versus categorical, but sometimes mistakes happen. For example, if you are looking at three drug treatments and have called them treatment 1, 2 & 3 in your file, R will interpret these as numerical. You can check how R interpreted the file using the `str()` command, which indicates the type of data in each column.

```
> str(MyData)
'data.frame': 10 obs. of 6 variables:
 $ Name      : Factor w/ 10 levels "Eric","Evan",...
 $ Height    : int
 $ Sex       : Factor w/ 2 levels "female","male"
 $ Age       : int
 $ ExerciseLevel: Factor w/ 3 levels "high","low","moderate"
 $ MetabolicRate: int
```

In this example, 'Name', 'Sex' and 'ExerciseLevel' are factors and 'Height', 'Age' and 'MetabolicRate' are integers. If the columns were interpreted incorrectly, we can force the columns into specific data types using the `colClasses` option in the `read.csv()` function. It works by giving R a vector indicating whether each column is numeric or categorical. Here's an example using the dataset on metabolic rates

```
#Load a data set and define column types
MyData=read.csv("metabolic.csv",colClasses=c('factor','numeric',
'factor','numeric','factor','numeric'))
```

With the dataset successfully loaded into R, we can graph the data and run statistical analyses. For example, to calculate the mean you type

```
> mean(MyData$MetabolicRate)
[1] 1680
```

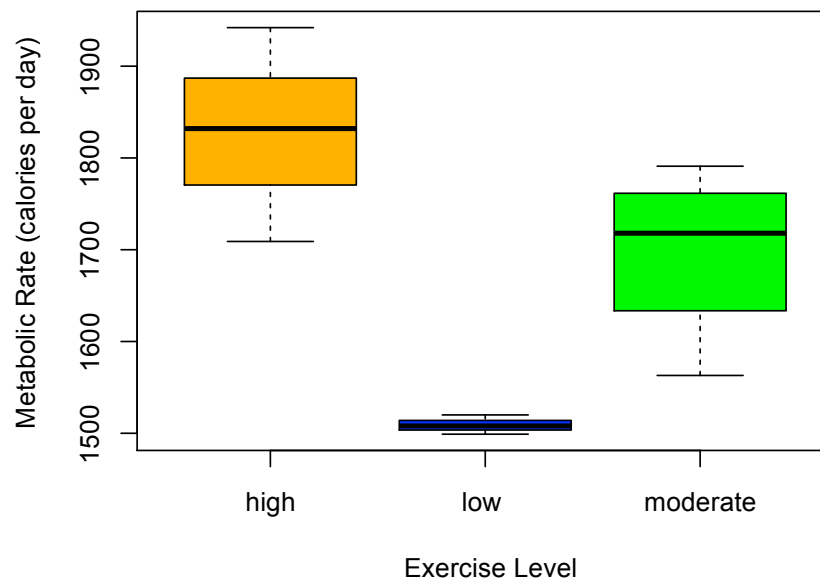
The following short script loads the dataset and creates a box plot of the metabolic rate for each exercise level. We will cover how to create a box plot in Primer 4 (including what on earth the ~ symbol is used for!), but for now we can run the script to see what the data look like

```
#Load the file metabolic.csv and define column types
MyData=read.csv("metabolic.csv",colClasses=c('factor','numeric',
'factor','numeric','factor','numeric'))

#create a boxplot
x=MyData$ExerciseLevel
y=MyData$MetabolicRate

boxplot(y~x,col=c("orange","blue","green"),ylab="Metabolic Rate
(calories per day)", xlab="Exercise Level")
```

The graph should look like the figure below, which shows that basal metabolic rate is higher for people with higher levels of exercise.



**Getting just the data that you want** The data frames contain all the data for a particular problem. However, sometimes we want to work with just a subset of the data. For example, we may only want to know the mean metabolic rate for individuals with 'high' exercise levels. This is easy to do in R by creating a new data frame that is a subset of the original one using the `subset()` function. The function `subset(data, condition)` takes two arguments: *data* and *condition*. The first argument is your original data frame and the second indicates which levels of a categorical variable to use. For example, if we just want individuals with 'high' exercise levels, we would type

```
> SubMyData=subset(MyData, ExerciseLevel=='high')
```

Here 'ExerciseLevel' is the column name in the dataframe 'MyData' and 'high' is the level that you want to select. Notice that level selection is done using a double equals sign '==', which in R stands for evaluating a condition. A single equals sign won't work. Have a look at your new subset data frame to see what it looks like.

```
> SubMyData
  Name Height Gender Age ExerciseLevel MetabolicRate
1 Natalie  155  female  19           high           1942
2  Tamar   163  female  21           high           1832
3  Henry   171   male   23           high           1709
```

Now you can work with the new data frame just like the old one, but you are only working with a select part of the data. Compare the mean metabolic rate for the subset data with the mean you calculated above from the original dataset.

```
> mean(SubMyData$MetabolicRate)
[1] 1827.667
```

## Primer 3: Calculating descriptive statistics

Descriptive statistics are the first step in any data analysis, and this section will show you how to calculate descriptive statistics in both Excel and R. You will learn how to do them in both software platforms because you will often want to whip up some descriptive stats in short order, and it will save you time to be able to do it in whatever program you are currently working in.

### Mean, median, variance & standard deviation

For this section, we will use the metabolic.csv data file from Primer 2. Open the file in Excel, and also load it into R. As a reminder, here are the commands you need to use in R to load the file and then check the contents.

```
#Load a data set
setwd("~/Statistics Course 2016")
MyData=read.csv("metabolic.csv") #load the file metabolic.csv

MyData #short dataset, so we can look at the entire thing
```

Calculating the mean, median, variance and standard deviation in Excel is done using the functions `AVERAGE()`, `MEDIAN()`, `VAR()` and `STDEV()` over your data range. For example, the formula to calculate the average metabolic rate in the following worksheet is `=AVERAGE(F2:F11)`

	A	B	C	D	E	F	G
1	Name	Height	Sex	Age	ExerciseLevel	MetabolicRate	
2	Kumiko	161	female	20	low	1520	
3	Meredith	143	female	19	moderate	1704	
4	Natalie	155	female	19	high	1942	
5	Tamar	163	female	21	high	1832	
6	Henry	171	male	23	high	1709	
7	Husbana	170	male	20	low	1508	
8	Evan	176	male	20	low	1499	
9	Matt	170	male	19	moderate	1791	
10	Mary	160	female	19	moderate	1732	
11	Eric	169	male	19	moderate	1563	
12							
13						=AVERAGE(F2:F11)	
14							

The blue box indicates the data range selected in the function. If you want to change the range, just grab any corner with your mouse and drag it. We can add some text to the worksheet to help identify the newly calculated descriptive statistics as shown below

	A	B	C	D	E	F
1	Name	Height	Sex	Age	ExerciseLevel	MetabolicRate
2	Kumiko	161	female	20	low	1520
3	Meredith	143	female	19	moderate	1704
4	Natalie	155	female	19	high	1942
5	Tamar	163	female	21	high	1832
6	Henry	171	male	23	high	1709
7	Husbana	170	male	20	low	1508
8	Evan	176	male	20	low	1499
9	Matt	170	male	19	moderate	1791
10	Mary	160	female	19	moderate	1732
11	Eric	169	male	19	moderate	1563
12						
13					Mean	1680
14					Median	1706.5
15					Variance	23313.8
16					Standard Deviation	152.7
17						

Some other useful functions are the `MIN()` and `MAX()` functions, which give you the minimum and maximum value respectively of a data range.

Calculating the mean, median, variance and standard deviation in R is done using the functions `mean()`, `median()`, `var()` and `sd()` applied to a data vector. For example, the mean of vector `a` is found using the command `mean(a)`. Here's what an R script to calculate the descriptive statistics for metabolic rates would look like (make sure to write this as a script rather than using the console!).

```
setwd("~/Statistics Course 2016") #set the working directory

MyData=read.csv("metabolic.csv") #load the file metabolic.csv
MyData #short dataset, so we can look at the entire thing

mean(MyData$MetabolicRate) #calculates the mean
median(MyData$MetabolicRate) #calculates the median
var(MyData$MetabolicRate) #calculates the variance
sd(MyData$MetabolicRate) #calculates the stdev
```

Make sure you understand what each line is doing. The first line sets the working directory, the second line reads in the dataset, the third line displays the full dataset on the screen so that you can check for errors, and the remaining four lines calculate the descriptive statistics. The output for the last 4 lines is

```
> mean(MyData$MetabolicRate) #calculates the mean
[1] 1680
> median(MyData$MetabolicRate) #calculates the median
[1] 1706.5
> var(MyData$MetabolicRate) #calculates the variance
[1] 23313.78
> sd(MyData$MetabolicRate) #calculates the stdev
[1] 152.6885
```

Other useful functions are `min()` and `max()`, which give you the minimum and maximum value respectively of a vector.

## Quartiles & interquartile range

Calculating quartiles and interquartile ranges is relatively straightforward, but requires using functions with 2 arguments. In Excel, the function is `QUARTILE(array, quart)`, where the first argument *array* contains the range of the data, and the second argument *quart* indicates the quartile of interest. For example, if we wanted the first quartile (25%) of metabolic rates, we would type the equation as shown below

	A	B	C	D	E	F
1	Name	Height	Sex	Age	ExerciseLevel	MetabolicRate
2	Kumiko	161	female	20	low	1520
3	Meredith	143	female	19	moderate	1704
4	Natalie	155	female	19	high	1942
5	Tamar	163	female	21	high	1832
6	Henry	171	male	23	high	1709
7	Husbana	170	male	20	low	1508
8	Evan	176	male	20	low	1499
9	Matt	170	male	19	moderate	1791
10	Mary	160	female	19	moderate	1732
11	Eric	169	male	19	moderate	1563
12						
13						=QUARTILE(F2:F11,1)
14						

The *quart* argument can take a value of 0, 1, 2, 3 or 4 depending on what quartile you are interested in. The following table shows what each value represents

If quart equals	QUARTILE returns
0	Minimum value
1	First quartile (25th percentile)
2	Median value (50th percentile)
3	Third quartile (75th percentile)
4	Maximum value

Notice that *quart*=0 will give you the minimum, and *quart*=4 will give you the maximum, which is an alternative way to get these values. The second quartile is 50% of the data, so is just the median value. Here's what an Excel worksheet looks like with the three middle quartiles added

	A	B	C	D	E	F
1	Name	Height	Sex	Age	ExerciseLevel	MetabolicRate
2	Kumiko	161	female	20	low	1520
3	Meredith	143	female	19	moderate	1704
4	Natalie	155	female	19	high	1942
5	Tamar	163	female	21	high	1832
6	Henry	171	male	23	high	1709
7	Husbana	170	male	20	low	1508
8	Evan	176	male	20	low	1499
9	Matt	170	male	19	moderate	1791
10	Mary	160	female	19	moderate	1732
11	Eric	169	male	19	moderate	1563
12						
13					First quartile	1530.75
14					Median (2nd quartile)	1706.5
15					Thrid quartile	1776.25
16						

The interquartile range is then calculated in a separate cell as the difference between the third and first quartiles. In this example, you would calculate the interquartile range as `=F15-F13`.

	A	B	C	D	E	F
1	Name	Height	Sex	Age	ExerciseLevel	MetabolicRate
2	Kumiko	161	female	20	low	1520
3	Meredith	143	female	19	moderate	1704
4	Natalie	155	female	19	high	1942
5	Tamar	163	female	21	high	1832
6	Henry	171	male	23	high	1709
7	Husbana	170	male	20	low	1508
8	Evan	176	male	20	low	1499
9	Matt	170	male	19	moderate	1791
10	Mary	160	female	19	moderate	1732
11	Eric	169	male	19	moderate	1563
12						
13					First quartile	1530.75
14					Median (2nd quartile)	1706.5
15					Thrid quartile	1776.25
16					Interquartile range	245.5
17						

Calculating quartiles and the interquartile range in R is a bit easier. The quartile function is `quantile(x, probs)`, where `x` is the data vector, and `probs` is a number or short vector that indicates the proportion of the data for the quartile. For example, if we wanted the first quartile (i.e., first 25% of the data) of vector `a`, we would use the command `quantile(a, 0.25)`. If we wanted the 1st, 2nd and 3rd quartiles, we would type `quantile(a, c(0.25, 0.5, 0.75))`. Notice that we are using the short vector `c(0.25, 0.5, 0.75)` to ask for all three quartiles at once. Here's what it would look like as an R script.

```
#Load a data set
setwd("~/Statistics Course 2016")
MyData=read.csv("metabolic.csv") #load the file metabolic.csv
```



```
MyData #short dataset, so we can look at the entire thing
quantile(MyData$MetabolicRate, 0.25) #First quartile
quantile(MyData$MetabolicRate, c(0.25,0.5,0.75)) #3 quartiles
```

The output for the last 2 lines is

```
> quantile(MyData$MetabolicRate, 0.25) #First quartile
25%
1530.75
> quantile(MyData$MetabolicRate, c(0.25,0.5,0.75)) #quartiles
 25%      50%      75%
1530.75 1706.50 1776.25
```

A nice shortcut in R is the `summary()` function, which calculates, the minimum, maximum, mean and quartiles in one step.

```
> summary(MyData$MetabolicRate) #a bunch of descriptive stats
Min.  1st Qu.  Median    Mean  3rd Qu.   Max.
 1499    1531    1706    1680    1776    1942
```

The interquartile range can be calculated by taking the difference between the 3rd and 1st quartiles, or by using the `IQR()` function.

```
> IQR(MyData$MetabolicRate)
[1] 245.5
```

Compare these values to the calculations in your Excel worksheet.

## Primer 4: Graphing

Just as descriptive statistics are the first step in any data analysis, visualizing data is a key step to understanding your data and communicating results to your readers. Seeing the data helps catch errors, and helps check that the statistical analysis makes sense for the kind of data you have. Hence the mantra: *Always plot your data!*

R creates gorgeous publication-quality figures that—for many types of figures—are much easier to generate than Excel. This primer will show you how to create each of the graphs covered in the course. The examples will use data on mercury concentrations in sediment samples for two lakes (OddBall lake and Lucky lake). For the first couple of graphs, you can enter the data directly into R by copying the following lines into a new script.

```
OddBall=c(122.17, 100.29, 79.54, 86.07, 78.24, 77.25, 69.89,
66.08, 91.26, 73.68, 67.58, 73.08, 102.54, 83.73, 88.86,
106.63, 67.72, 82.56, 93.73, 71.41)

Lucky=c(64.07, 55.36, 61.17, 72.51, 87.68, 72.31, 76.67,
63.05, 68.33, 59.87,86.48, 80.71, 58.85, 45.02, 63.17, 72.06,
71.51,69.88, 63.55, 87.78)
```

## Contingency tables

It might seem odd to start a primer on graphing with contingency tables, but tables are a very useful way to visualize categorical data. Contingency tables summarize the frequency of observations that fall into one or more categories. For example, the following table shows the numbers of sediment samples from two lakes with mercury concentrations that are acceptable (below 100 ppb) or at risk to causing harm to aquatic life (above 100 ppb).

Sediment Status	Lucky Lake	OddBall Lake	Row Total
At Risk	0	4	4
Acceptable	20	16	36
Column Total	20	20	40

To create a contingency table in R, we will use the long-form data structure from Primer 2. This can either be read into R using a .csv file, or entered directly using a script. For this example, we will enter the data directly. We need two vectors. The first will indicate the source lake ('O' is for OddBall lake, and 'L' is for Lucky lake), and the second will indicate risk ('Y' is for at risk, 'N' is for not at risk).

```
lake=c('O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O',
'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'L', 'L', 'L', 'L', 'L',
'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'L',
'L', 'L')
risk=c('Y', 'Y', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
'Y', 'N', 'N', 'Y', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
'N', 'N')
```

The contingency table is then created using the `table()` function.

```
> table(lake,risk)
```

	risk	
lake	N	Y
L	20	0
O	16	4

The table shows that no sediment layers in Lucky lake are of concern, but four layers in OddBall lake are. The data could then be entered as a table in a report.

## Histogram

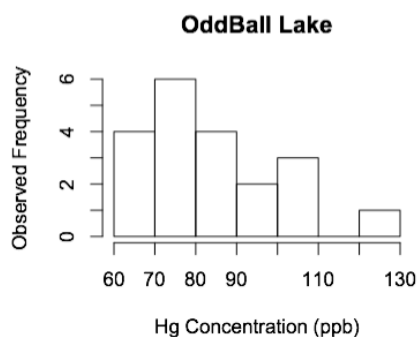
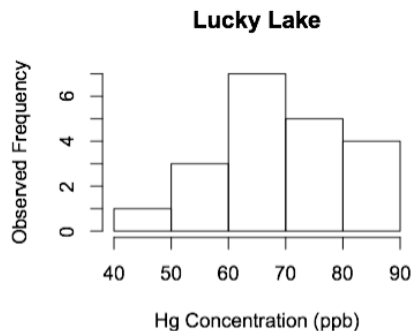
A histogram is a plot of the observed frequency of an event, and is created in R using the `hist()` function. A basic histogram is created as follows:

```
hist(Lucky)
```

To create custom text for the figure, use the `xlab=""`, `ylab=""`, and `main=""` options, which add the desired text (entered in place of ...) to the x-axis, y-axis and plot title respectively.

```
hist(Lucky,main="Lucky Lake",xlab="Hg Concentration
(ppb)",ylab="Observed Frequency")
```

These options work with most of the plotting functions used in R. The histogram for each lake is shown below.



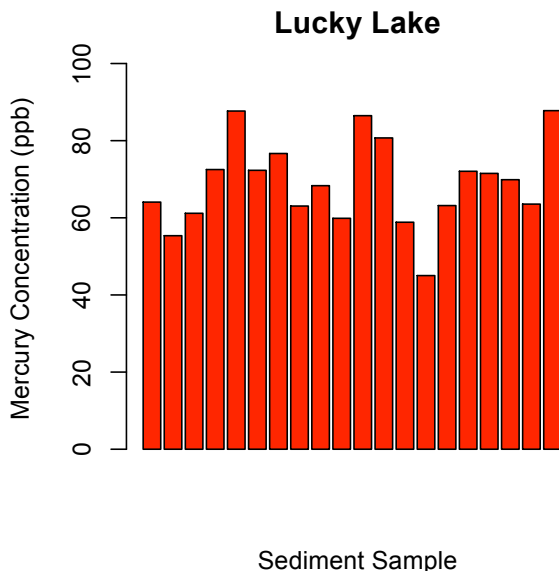
## Bar plots

The `barplot()` function is used to create bar plots of your data. The number of bars in the figure is equal to the number of observations in the data set, and the y axis shows the value of the observations within a variable. The x-axis is the row the data were encountered in the dataset. A labelled bar plot of the Lucky lake data is created using

```
barplot(Lucky,main='Mercury Concentration in Lucky Lake')
```

With the default values, the function created a bar plot with a y-axis that ranges from 0 to 80, which leaves some bars to over hang the observed axis range. To tidy this up, we can use the `ylim` option, which has the form `ylim=c(a,b)`, where the desired minimum and maximum are entered in 'a' and 'b' respectively. We can also add colour to the plot using the `col` option, which has the form `col='...'` (e.g., `col='blue'`), where the color is entered as text (type `colors()` into R to see a list of common colors). A more polished barplot is created using

```
barplot(Lucky,ylim=c(0,100),col='red',main='Lucky Lake',
xlab='Sediment Sample', ylab='Mercury Concentration(ppb)')
```



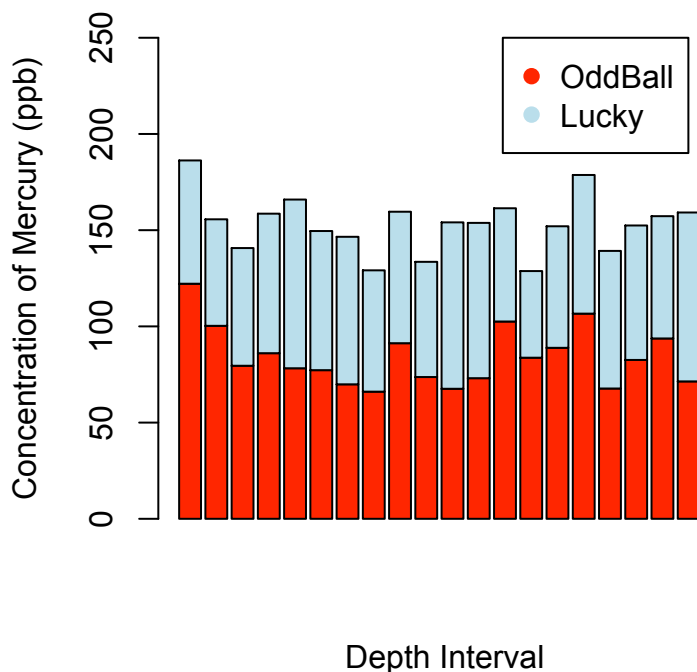
To visualize more than one category, we use stacked or grouped bar plots. To do this, start by combining the two vectors into a single data frame.

```
BothLakes=data.frame('OddBall'=OddBall, 'Lucky'=Lucky)
```

The data frame *BothLakes* has two columns with each row a different depth. The text in quotations will be the column name for the vector in the new data frame. Type `BothLakes` into the R console to see the structure of the data for yourself. If you imported the data from a .csv file, it will already be in a data frame format. Since we want to plot the different depths across the x-axis, the `barplot()` function requires us to switch the rows and columns so that the data frame has two rows (OddBall and Lucky), and 20 columns that represent the depths.

This is done using the `t()` function, which transposes the data set (i.e., switches the rows and columns). This is done by first converting the data frame to a matrix using the `as.matrix()` function.

```
BothLakes=as.matrix(BothLakes)
BothLakesTransform=t(BothLakes)
```



Have a look at the new data frame `BothLakesTransform` to see what the transpose function has done. The stacked bar plot is then created using

```
barplot(BothLakesTransform, xlab="Depth Interval",
        ylab='Concentration of Mercury (ppb)', ylim=c(0,250), col=c('red',
        'lightblue'))
```

All of the plot functions have options that let you get the feel and look you want. Take some time to explore how changing these settings affects the graph, and make sure you know what each option does.

The barplot is starting to look nice, but it's missing a legend to tell us what colour represents each lake. We can add a legend using the `legend()` function. The legend function has a number of arguments that tell R where to put the legend, what text to add and the colours. The function is `legend(x, y, legend, col, pch)` where `x` and `y` give the location of the upper left corner of the legend box, `legend` is a vector of names you want in the legend, `col` is the colour to use, and `pch` indicates the type of symbol (see Plotting Options in the last chapter for different symbols). Here's the code for the above example that places the legend at `x=15` (15th observation in this case) and `y=250`.

```
legend(15, 250, c('OddBall', 'Lucky'), col=c('red', 'lightblue'), pch=19)
```

A grouped bar is created using the `beside` option, which has the form `beside=TRUE`. This script will create a grouped bar graph with a legend.

```
barplot(BothLakes,main='', xlab="Depth Interval",
ylab='Concentration of Mercury (ppb)', ylim=c(0,150),col=c('red',
'lightblue'),beside=TRUE)
legend(40,150,c('OddBall','Lucky'),col=c('red',
'lightblue'),pch=19)
```

## Box plots

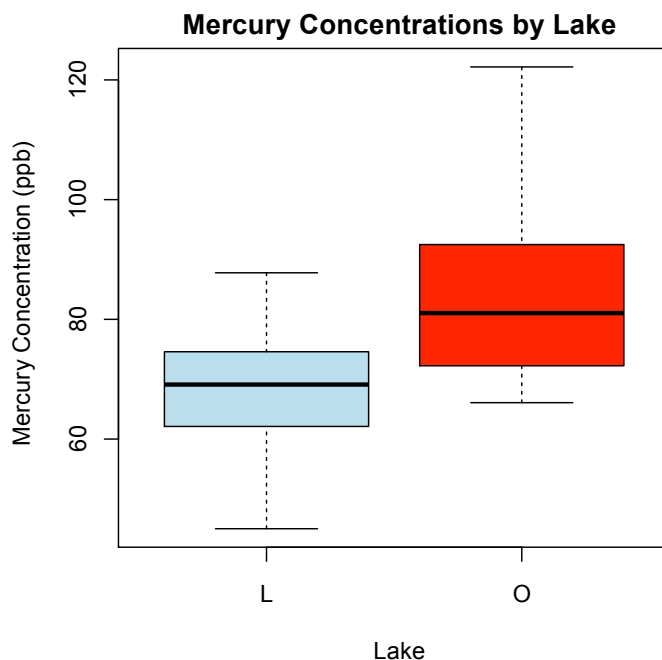
A box and whisker plot illustrates the median and quantile levels. The function `boxplot()` indicates the median by a dark band within a 'box' which is bound by the 1st and 3rd quantiles. 'Whiskers' in this plot represent the most extreme datapoint within 1.5 times the interquartile range. Values outside the whiskers are plotted as points. To visualize the mercury concentrations in each lake, we need to create long-form data similar to what was used for the contingency tables. The first vector will be the mercury concentrations for both lakes, and the second vector will indicate which lake the observation is from. This is the format discussed in Primer 2. At this point it is easier to enter the data in Excel, and import it as a .csv file. The data can be found in the file *mercury.csv*. Copy this file to your working directory, load the data into R, and have a look at the contents so that you see the data structure.

```
MyData=read.csv("mercury.csv")
```

The box plot is created using

```
boxplot(mercury~lake,main="Mercury Concentrations by Lake",
ylab='Mercury Concentration (ppb)',xlab='Lake',col=
c('lightblue','red'),data=MyData)
```

where the term `mercury~lake` is a formula that indicates to the `boxplot()` function that the mercury quantiles should be calculated for each level in lake. We will see more about writing formulas when we cover statistical tests. Here's what the box plot looks like.



Sometimes we want more control over how the categories are presented on the x-axis. In R, this order is determined when the data are first loaded. It's an attribute of the data frame. To see the order, use the `levels()` function, which shows the levels in a categorical vector. For example

```
levels(MyData$lake)
[1] "L" "O"
```

shows us that the data are ordered by Lucky lake (L) then OddBall lake (O). To change the order, we need to use the `factor(x, order)` function where `x` is the vector you want to reorder and `order` is the order you want the categories to appear on your figure. For the column of lake data, we can type

```
MyData$lake=factor(MyData$lake, c("O", "L"))
```

The factor function does not change the data, it only changes the order of how the levels are displayed. Have a look at the new levels.

```
levels(MyData$lake)
[1] "O" "L"
```

Now recreate the box plot and see what effect it had!



## Scatter plots

A scatterplot displays quantitative data for two variables. We encountered the `plot()` function in Primer 1, and it provides a great way to create scatterplots. For the example of mercury concentrations in two lakes, we can compare the mercury at each sediment depth as follows

```
plot(OddBall,Lucky,main='Comparing Lake Mercury
Concentrations',xlab='OddBall Lake Concentration', ylab='Lucky
Lake Concentration',xlim=c(60,130),ylim=c(40,90))
```

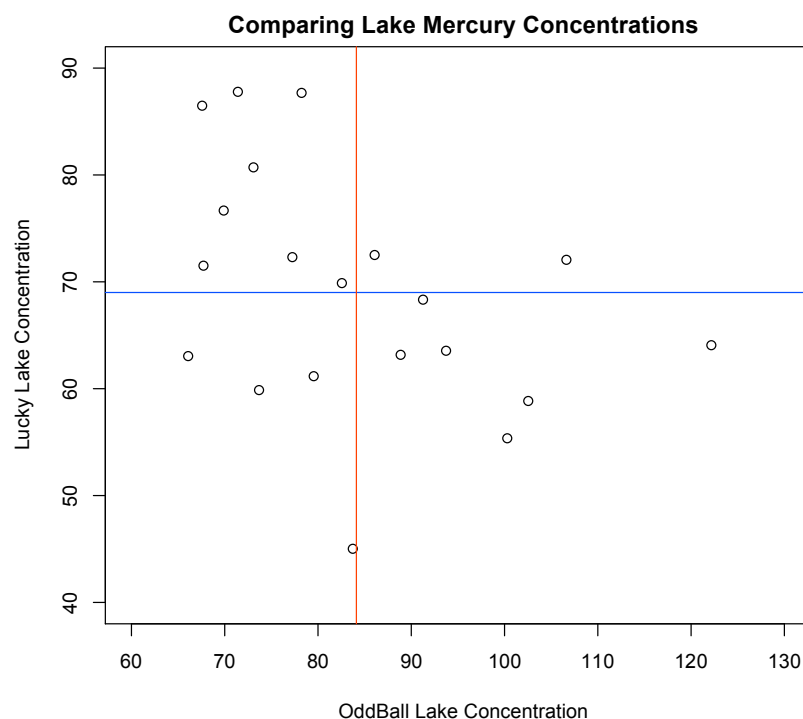
We can highlight changes relative to the means by adding two lines that represent the mean for each lake. The `abline()` function is used to create straight lines on a plot. For a vertical line, the `abline()` function has the form `abline(v)`, where “v” indicates where you want the line drawn.

```
> abline(v=84.1155,col= 'blue')
```

For a horizontal line the `abline()` function has the form `abline(h)` as follows:

```
> abline(h=69.0015,col= 'red')
```

The final graph looks like this image.

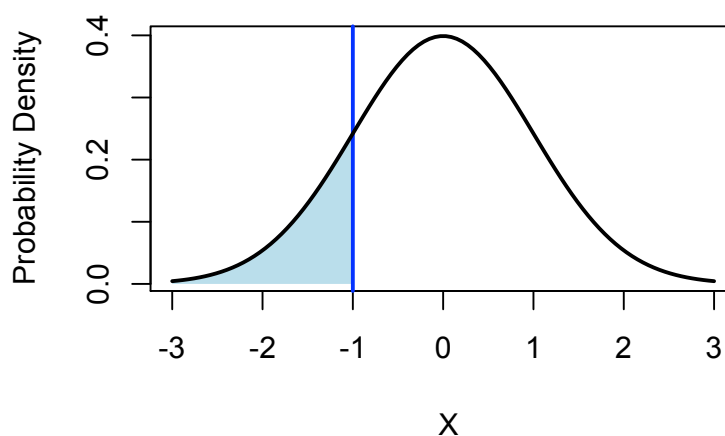


## Primer 5: Calculating probabilities

To help prepare for hypothesis testing, this Primer 5 looks at calculating probabilities and event ranges from distributions. Calculating these quantities is important for being able to find critical test scores.

### Event ranges and probabilities

We are interested in computing two quantities from probability distributions. The first is the probability over a range of events, and the second is reverse, which is to find the range of events for a given probability. For example, consider the following figure that shows the probability density (black line) over a range of event values (x-axis). The probability of observing an event that is less than -1 (vertical line) is given by the area under the curve for that range of events (shown in blue). This is the quantity that we are most familiar with calculating from a probability distribution. To go in the opposite direction, we start with a specified probability and find the range that would give that probability. In this primer we will learn to calculate probabilities and ranges for both the Normal and Binomial distributions.



To calculate probabilities over a range of events from a Normal distribution in R, we use the `pnorm(x, mean, sd)` function, where `x` is the upper limit of the event range, and the options `mean` and `sd` are the parameters of the Normal distribution. The function returns the probability of observing a value `x` or smaller. Be careful of the direction for the events—the probability is the area to the left of the upper limit. To calculate the probability shown in the above figure, we would write

```
> pnorm(-1, 0, 1)
```

```
[1] 0.1586553
```

In words, the code asks for the probability for all values smaller and equal to -1 (event range) for a Normal distribution with mean of 0 and standard deviation of 1. Make sure you understand what each component of the function does.

To go in the other direction, we use a related function called `qnorm(p, mean, sd)`, where  $p$  is the probability and *mean* and *sd* are the parameters for the Normal distribution. The probability is assumed to be the area to the left of the event in these functions. The function then returns the upper threshold that corresponds to the requested probability. For example, to locate the event range in our Normal distribution that corresponds to  $p=0.025$ , we would write

```
> qnorm(0.025, 0, 1)
[1] -1.959964
```

The threshold -1.96 is the upper range corresponding to a probability of  $p=0.025$  for a Normal distribution with a mean of zero and standard deviation of one.

The functions for the Binomial distribution are similar. The `pbinom(x, n, p)` is used to calculate the probability of observing  $x$  successes out of  $n$  trials if the probability of success is given by  $p$ . Asking the reverse question is similar to the Normal distribution, except that R will return an integer number of successes because the events are discrete by definition in the Binomial distribution. That means a range of probabilities are associated with observing a particular event. The following code shows an example. The first line finds the probability of observing 2 successes over 10 trials with a success rate of 0.5, and the remaining lines ask for the event ranges that correspond to three different probabilities.

```
> pbinom(2, 10, 0.5)
[1] 0.0546875
> qbinom(0.11, 10, 0.5)
[1] 3
> qbinom(0.15, 10, 0.5)
[1] 3
> qbinom(0.18, 10, 0.5)
[1] 4
```

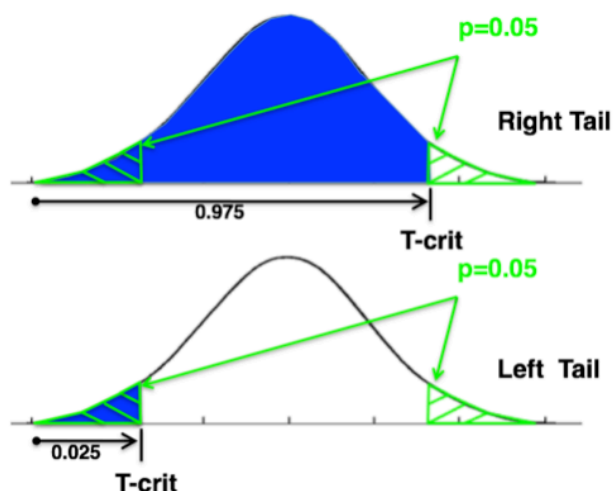
## Primer 6: Chi-square tests, T-tests & confidence intervals

In this primer, you will develop the skills to calculate confidence intervals, and to do hypothesis testing for one-sample t-tests, two-sample t-tests, and Chi-square tests.

### Confidence intervals for single samples

Confidence intervals (CI) for single samples are based on the standard error (SE) and critical t-score ( $t_c$ ), which are both straightforward to calculate in R. The formula is  $CI = \text{mean} \pm t_c SE$ . The standard error is calculated as  $SE = s/\sqrt{n}$ , where  $s$  is the sample standard deviation and  $n$  is the sample size. The standard deviation is calculated using the `sd()` function as shown in Primer 3. The sample size is found using the `length()` function, which reports the number of observations in a vector.

To find the critical t-score, we use the approach shown in Primer 5, but for the t-distribution. Specifically, this is done using the `qt(p, df)` function, where  $p$  is the probability of interest, and  $df$  is degrees of freedom. The `qt()` function is analogous to the `qnorm()` function discussed in the previous section, but we need to take care to calculate the lower and upper interval correctly. The reason is that the `qt()` function assumes that the probability of interest is always to the left of the critical  $t_c$  value (blue in figure below). Since confidence intervals are two-sided, the probability in each tail is  $\alpha/2$ . As shown in the following figure, the left tail threshold is found at  $p = \alpha/2$ , and the right tail threshold at  $p = 1 - \alpha/2$ .



To illustrate the calculation of 95% confidence intervals, consider the following vector of data

```
Mydata=c(3.89,5.85,5.97,6.10,4.44,6.12,4.24,5.63,3.48,4.32)
```

Using the variables UCI and LCI to represent the upper and lower intervals respectively, we calculate confidence intervals using

```
m=mean(Mydata)           #calculate sample mean
n=length(Mydata)         #calculate sample size
SE=sd(Mydata)/sqrt(n)    #calculate standard error
LCI=m+qt(0.025,n-1)*SE   #calculate the lower interval
UCI=m+qt(0.975,n-1)*SE   #calculate the upper interval
LCI
[1] 4.272117
UCI
[1] 5.735883
```

The lower 95% confidence interval is 4.27, and the upper 95% confidence interval is 5.74.

## Chi-square tests

The Chi-squared test is the first hypothesis test that we have encountered. It is used to test for independence among categorical variables. For example, we might be interested in whether colour blindness is independent of gender. If males are more likely to be colour blind than females, then we expect that the relative frequency of people with colour blindness would not be independent of gender. The null hypothesis for the Chi-squared test is that the categorical variables are independent of one another.

$H_0$ : Categorical variables are independent

$H_A$ : Categorical variables are not Independent

For our example of colour blindness, a Chi-squared test hypothesis would be:

$H_0$ : There is no difference in the degree of colour blindness between males and females

$H_A$ : There is a difference in the degree of colour blindness between males and females

For a fully-worked example, let's look at some hypothetical data on an invasive species, *Bythotrephes longimanus*. This invasive animal, commonly called the spiny water flea, entered the Great Lakes region in the mid 1980s, and caused a large

change in the aquatic life of many lakes. One area where *Bythotrephes* invasion has been studied in detail is the 'cottage country' region in southern Ontario. The following table shows the number of lakes with and without cottages, as well as the state of *Bythotrephes* invasion in the lake (not invaded, invaded but not abundant, and invaded and dominant).

	Not invaded	Invaded, but not abundant	Invaded and abundant	Row total
Cottages	25	60	65	150
No Cottages	40	7	3	50
Column total	65	67	68	200

The statistical hypotheses are:

$H_0$ : Presence of cottages and lake invasion status are independent

$H_A$ : Presence of cottages and lake invasion status are not independent

The `chisq.test()` function in R is used to do Chi-square tests. The first step is to get the data into R. This can be done by creating a .csv file and importing the data as was done in Primer 2 or, as shown here, we can enter the data directly in

R. Begin by creating data vectors

```
cottage=c(25, 60, 65)
no.cottage=c(40, 7, 3)
```

and then create the data frame

```
Mydata=data.frame('C'=cottage, 'NC'=no.cottage)
```

Make sure to look at your data frame to check that the variables are in the right order:

```
> Mydata
```

```
  C  NC
1 25 40
2 60  7
3 65  3
```

The Chi-squared test is done by typing

```
> chisq.test(Mydata)
      Pearson's Chi-squared test

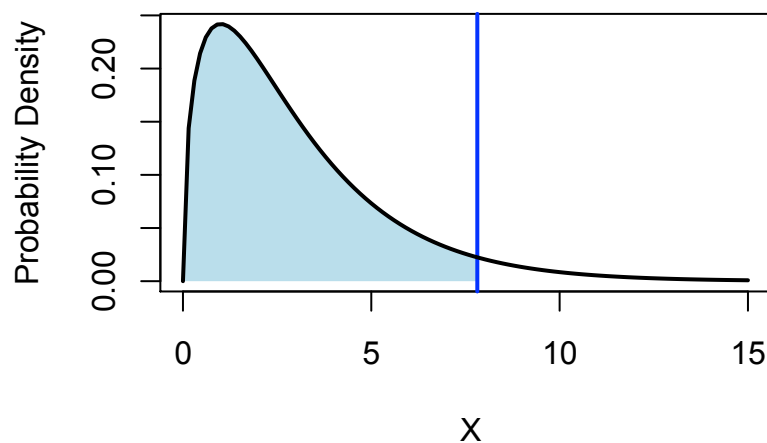
data:  Mydata
X-squared = 69.2218, df = 2, p-value = 9.304e-16
```

The output shows you

- The type of test: Pearson's Chi-squared test
- The Chi-squared value: X-squared = 69.2218
- degrees of freedom: df = 2
- p-value: p-value = 9.304e-16

Since  $p < 0.05$ , we reject our null hypothesis that *Bythotrephes* abundance in Muskoka lakes is independent of the presence or absence of cottages.

Equivalently, we can do the same test by comparing the observed versus critical test statistics. The observed  $\chi^2$  value is given in the output of the `chisq.test()` function, and the critical  $\chi^2$  value can be found using `qchisq()` with a type I error rate of  $\alpha = 0.05$  and degrees of freedom 2. The `qchisq()` function is the analog of the `qnorm()` and `qt()` function for the Chi-square distribution. Recall that Chi-square tests are one-tailed test and the Type I error rate refers to the area in the right tail as shown below.



Since the `qchisq()` function calculates the event threshold that corresponds to the probability to the left, we need to enter  $1 - \alpha$  as the probability value.

```
qchisq(0.95, 2)
```

```
[1] 5.991465
```

Since the observed  $\chi^2$  value (69.2) exceeds the critical  $\chi^2$  value (5.99) the null hypothesis is rejected.

## Single-sample t-tests

A single-sample t-test compares the mean of an observed set of data to a known value. To demonstrate a one-sample t-test in R, lets look at some hypothetical data from a fish farm. As a measure of stress in adult fish, the farm monitors the number of eggs produced per female to ensure that the fish are under optimal conditions for reproduction. The procedure is to randomly select ten fish during each egg harvest, and count the total eggs per fish. Based on previous data, the minimum number of eggs from a non-stressed fish is 1100. The following table shows the egg count from the most recent harvest.

Fish ID	Eggs/fish
F1	778
F2	1367
F3	947
F4	1002
F5	521
F6	656
F7	1082
F8	1144
F9	735
F10	1283

Since the data set is small, it is easy to enter the data directly into R

```
eggs=c(778,1367,947,1002,521,656,1082,1144,735,1283)
```

The t-test function `t.test(data,mu)` is used to compute the observed t-score and p-value for this t-test, where the arguments `data` and `mu` define the test. For a single-sample t-test, `data` is the observed data and `mu` is the constant you want to test the mean against. Here's what it looks like for the fish egg problem:

```
t.test(eggs, mu=1100)
```



Which returns an output that looks like this:

```

One Sample t-test

data:  eggs
t = -1.6974, df = 9, p-value = 0.1238

alternative hypothesis: true mean is not equal to 1100

95 percent confidence interval:
 753.5969 1149.4031

sample estimates:
mean of x
 951.5

```

The key values to look for in this output are the degrees of freedom, observed t-value, and the p-value. In this case the degrees of freedom is 9, t-value is -1.697, and the p-value is 0.1238. The default test for the `t.test()` function is two-tailed, which means that the statistical hypotheses are:

$H_0$ : There is no difference between the mean number of eggs per fish in the sample and the threshold of 1100

$H_A$ : There is a difference between the mean number of eggs per fish in the sample and the minimum threshold of 1100.

Here, we are not concerned with the direction of the difference. Thus, in this case, we would fail to reject the null hypothesis because  $p > 0.05$ . The same result can be obtained using the test scores from the `qt()` function. For a two-tailed Type I error rate of  $\alpha = 0.05$ , the probability we seek is  $p = 1 - \alpha/2$  (see *Confidence Intervals* for explanation). In R, we type

```

> qt(0.025, 9)
[1] -2.262157

```

Thus, for the two-tailed test the left-side value of the critical test statistic is  $t_C = -2.262$ , which is less than the observed value  $t_O = -1.697$ . The purpose of sampling the fish, however, is to evaluate whether the eggs per fish are less than the minimum threshold, so a one-tailed hypothesis would be more appropriate. The statistical hypotheses are:

$H_0$ : The mean number of eggs per fish in the sample is not less than the threshold of 1100

$H_A$ : The mean number of eggs per fish in the sample is less than the threshold of 1100

A one-tailed test is still done using the `t.test()` function, but we must also include the argument, *alternative*, which specifies the alternative hypothesis (either “greater” or “less”). Since we are evaluating whether the mean number of eggs per fish is less than 1100, the new code looks like this:

```
t.test(eggs, mu=1100, alternative="less")
```

and returns this output:

```
One Sample t-test

data:  eggs
t = -1.6974, df = 9, p-value = 0.06192

alternative hypothesis: true mean is less than 1100

95 percent confidence interval:
 -Inf 1111.869

sample estimates:
mean of x
 951.5
```

Here, the  $t$  score and degrees of freedom remain the same, but the one-sided  $p$ -value is  $p=0.062$ , which is greater than 0.05 so we fail to reject the null hypothesis and conclude that the observed eggs per fish are not reflective of stressed fish.

## Paired-sample t-tests

A paired-sample  $t$ -test is a special type of single-sample  $t$ -test that compares whether the difference between pairs of data are different from a mean of zero. The null hypothesis is no difference in the mean between the two groups.

$H_0$ : There is no difference between the two groups

$H_A$ : There is a difference between the two groups

A common application of paired-sample t-tests is to evaluate a response before and after an event. To illustrate the test, we will use an example of coral bleaching in response to a period of warm sea water. In this study, coral density was monitored on a reef before and after the warming event. The following table shows the density of different species at each sampling time.

Species	Density (indv/m <sup>2</sup> ) before event	Density (indv/m <sup>2</sup> ) after event
Porites lutea	18.1	33.1
Porites lobata	21.3	39.4
Leptastrea transversa	16	36.1
Goniastrea aspera	21.6	42
Goniastrea pectinata	21	33.2
Leptastrea purpurea	20.3	43.6
Platrygra ryukuenis	25.2	39.4
Porites rus	22	39.1
Favites halicora	22.9	37.8
Favia fava	23.5	45.1
Millepora intricata	24	1.2
Millepora dichotoma	26.1	0.8
Acropora digitifera	18.1	1.7
Porites attenuata	24.2	12.1
Porites sillimaniani	21	9.7
Stylophora pistillata	19.2	10.1
Porites cylindrica	24	4.8
Montipora aequitub.	21.8	4.1
Porites nigrescens	19.2	10.2
Pocillopora damicornis	19.8	5.2
Millepora platphylla	26.7	6.3
Porites aranetai	23.1	8.1
Porites horizontalata	25	4.1
Seriatopora hystrix	14.8	2.3

You can create a .csv file of the data, or enter it directly into R as follows

```
PreEvent=c(18.1, 21.3, 16, 21.6, 21, 20.3, 25.2, 22, 22.9, 23.5, 24, 26.1, 18.1, 24.2, 21, 19.2, 24, 21.8, 19.2, 19.8,
```

```
26.7, 23.1, 25, 14.8)
PostEvent=c(33.1, 39.4, 36.1, 42, 33.2, 43.6, 39.4, 39.1,
37.8, 45.1, 1.2, 0.8, 1.7, 12.1, 9.7, 10.1, 4.8, 4.1, 10.2,
5.2, 6.3, 8.1, 4.1, 2.3)
```

In this example we assume that the difference between the pairs is zero for the null hypotheses, but this is not always the case.

$H_0$ : The mean difference between pre and post event densities is zero

$H_A$ : The mean difference between pre and post event densities is not zero

As before, the `t.test(data, mu, paired)` function provide the R tools for the test. As in the previous section, we must specify the `data` arguments and the value we are comparing against, `mu`. For a paired-sample t-test, we must include both vectors being compared, and an additional argument, `paired`, which specifies the type of test with a TRUE or FALSE statement.

```
t.test(PreEvent, PostEvent, mu=0, paired=TRUE)
```

The returned output looks like this:

```
Paired t-test
data: PreEvent and PostEvent

t = 0.57225, df = 23, p-value = 0.5727

alternative hypothesis: true difference in means
is not equal to 0
95 percent confidence interval:
 -5.382397  9.499064
sample estimates:
mean of the
differences

2.058333
```

The summary shows  $p > 0.05$ , so we fail to reject the null hypothesis. It is a good habit to do the analysis using both p-values and t-scores so that it becomes second nature. Since the null hypothesis is two-sided, we calculate a two-sided critical t-score as

```
> qt(0.975, length(PreEvent) - 1)
[1] 2.068658
```

where we use the `length()` function to find the number of observations. Since the magnitude of  $t_c > t_0$ , we fail to reject the null hypothesis and conclude that the

data do not provide strong evidence that the difference is something other than zero.

## Independent two-sample t-tests

The independent t-test is our first test that involves more than a single group. These tests are used to evaluate whether two populations have different means, and are an invaluable tool for differentiating between the outcome of two trials or treatments. For example, imagine that you started a new job working for an engineering firm as an aquatic biologist and risk assessment expert. The company is planning to dam a large river, which will cause a decrease in water flow during the trout spawning season. The change in water flow might decrease the amount of oxygen flowing over the eggs, and thereby have an impact on egg hatching rate. As the biologist in the group, they have given you access to a flow tunnel to see if the fish eggs will hatch under the new flow regime. You take many eggs from the same fish stock and used 100 eggs for each trial. After 14 replicate trials at normal and decreased flow speeds, your preliminary data on the number of eggs that hatched is

Trial	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Normal	78	72	88	80	73	81	62	76	73	90	92	76	71	74
Decreased	58	57	45	56	66	60	49	51	52	65	50	48	58	57

To test if the two treatments have a different impact on the number hatching, you use an independent two-sample t-test. The first step is to input the data in the correct form. Specifically, one column must contain the data and the other column a coding variable indicating the treatment for each trial (i.e., in long form, see Primer 1 for details). This can be done by importing a .csv file into R, or by entering the data directly. The data vector is

```
egg.count=c(78, 72, 88, 80, 73, 81, 62, 76, 73, 90, 92, 76, 71, 74,
58, 57, 45, 56, 66, 60, 49, 51, 52, 65, 50, 48, 58, 57)
```

To produce the categorical vector, a code for normal and slow water movement must be used. Let N, and S denote normal and slow respectively

```
trial=c('N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N',
        'N', 'N', 'N', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S', 'S',
        'S', 'S', 'S', 'S', 'S')
```

Then combine these vectors into a single data frame

```
Mydata=data.frame('Hatched'=egg.count, 'Trial'=trial)
```

The data set should look like

```
> Mydata
  Hatched Trial
1      78    N
2      72    N
3      88    N
4      80    N
5      73    N
6      81    N
7      62    N
8      76    N
9      73    N
10     90    N
11     92    N
12     76    N
13     71    N
14     74    N
15     58    S
16     57    S
17     45    S
18     56    S
19     66    S
20     60    S
21     49    S
22     51    S
23     52    S
24     65    S
25     50    S
26     48    S
27     58    S
28     57    S
```

With the data setup in this fashion, the `t.test()` function is again used here, however the arguments are setup slightly differently. Rather than listing the two vectors being compared, as in the paired sample, we use the `~` symbol as follows:

```
t.test(Hatched ~ Trial, data=MyData)
```

The `data` option in `t.test()` indicates where R will look for the data. Now let's look at the results.

```
Welch Two Sample t-test

data: Hatched by Trial
t = 8.1654, df = 24.348, p-value = 1.969e-08

alternative hypothesis: true difference in means is
not equal to 0

95 percent confidence interval:
 16.76380 28.09334

sample estimates:
mean in group N mean in group S
   77.57143      55.14286
```

The output is very similar to the outputs from single and paired-sample t-tests discussed in previous sections, however the alternative hypothesis has changed to suit the different test. Since we are comparing the means of the two populations, null and alternative hypotheses are:

Ho: The difference between the population means is zero.

HA: The difference between the population means is not zero.

Since  $p < 0.05$ , we reject the null hypothesis and conclude that stream flow changes the mean hatching rate of fish eggs.

## Primer 7: Correlation & Linear Regression

Here we will build on our new statistical skills to analyze data with two numerical variables using correlation or linear regression analysis. Correlation is appropriate when the two variables are not causal and you are just interested in the relationship between them. Linear regression is appropriate when you want to use one variable to predict the other variable.

### Correlation

Correlation measures the tendency of two variables to change together. It answers the question: when one variable increases, does the other increase, decrease, or stay the same? The Pearson correlation coefficient is used to estimate this relationship, and is given by

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2(y_i - \bar{y})^2}}$$

The correlation coefficient ( $r$ ) varies from -1 to 1. The absolute value of the coefficient represents the strength of the correlation, and the sign represents the direction. A positive correlation means that when one variable either increases or decreases the other variable does the same. A negative correlation means that when one variable either increases or decreases the other does the opposite. A correlation of zero means that the variables vary independent of each other.

Correlation analysis is done using the `cor.test(x,y)` function in R, where  $x$  and  $y$  denote the two numerical variables. Here is a set of data that compares the quality of canned tuna using two approaches; the  $H$  variable is the 'Hunter L' colour score, and the  $C$  variable is a consumer satisfaction index (1-6).

```
H=c(44.4, 45.9, 41.9, 53.3, 44.7, 44.1, 50.7, 45.2, 60.1)
C=c(2.6, 3.1, 2.5, 5.0, 3.6, 4.0, 5.2, 2.8, 3.8)
```

Plot the data to see the qualitative pattern

```
plot(C,H,xlab="Consumer Index",ylab="Tuna Lightness")
```

The correlation analysis is then done as

```
MyFit=cor.test(H,C) #save correlation test to 'MyFit'
```



which saves the results of the analysis to the variable *MyFit*. The results are displayed by typing the variable name used to store the test

```
> MyFit
1)    Pearson's product-moment correlation

2)    data:  H and C
3)    t = 1.8411, df = 7, p-value = 0.1082
      alternative hypothesis: true correlation is not equal to 0
4)    95 percent confidence interval:
      -0.1497426  0.8955795
sample estimates:
5)    cor
      0.5711816
```

The blue numbers are not part of the R output, but were added to help explain the output. The output indicates:

- 1) The correlation coefficient is a 'Pearson' correlation. There are other types of correlation coefficients, but we are only covering Pearson correlation coefficients in this course.
- 2) The data are given in variables H and C.
- 3) The correlation test is a one-sample t-test, with  $t_0=1.8411$  on 7 degrees of freedom, and a p-value of  $p=0.1082$ . The alternative hypothesis indicates whether the test was one-tailed or two-tailed. In this case it is a two-tailed test.
- 4) The 95% confidence intervals are -0.1497426 and 0.8955795.
- 5) And finally, the correlation value is  $r=0.5711816$ .

## Linear regression

The equation for a linear regression is given by

$$Y = a + bX$$

where  $Y$  is the response variable (dependent variable) and  $X$  is the explanatory variable (independent variable). The coefficients  $a$  and  $b$  are the intercept and slope respectively. We use linear regression to predict the response variable ( $Y$ ) based on the explanatory variable ( $X$ ). The regression coefficients ( $a$ ,  $b$ ) are calculated from your data using the method of least squares. Linear regression is done using the `lm(formula)` function in R. This function can be used for both linear regression and ANOVA with just a change in how the *formula* is specified. For linear

regression, the formula is given as  $y \sim x$ , where both  $y$  and  $x$  are numerical variables. To illustrate linear regression, we will use an example data set already in R called 'cars'. To see the data type

```
> cars
```

You should see two columns *speed* and *dist*, which indicate the speed the car is traveling at the time of breaking and the distance the car travels before coming to a stop. To reduce clutter in the code, we assign these columns to new vector names

```
speed=cars$speed dist=cars$dist
```

At this point it is important to make sure that your data are numerical variables and not accidentally entered as categorical. Use the `str()` function to check the data type. If the variable is numerical, R reports 'numerical'. If the variable is categorical, R reports 'factor'. If you find one is categorical, then force it to be a numerical variable by using the `colClasses` option (Primer 2) when you load the data. Let's plot the figure to see the qualitative trends and catch any errors.

```
plot(speed,dist, xlab="Speed (mph)", ylab="Breaking Distance (feet)")
```

The next step is to fit the linear regression

```
MyFit=lm(dist~speed)
```

To see the results, use the `summary()` function

```
summary(MyFit)
1) Call:      lm(formula = dist ~ speed)

2) Residuals:
   Min       1Q   Median       3Q      Max
-29.069  -9.525  -2.272   9.215  43.201

3) Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -17.5791     6.7584  -2.601  0.0123 *
speed        3.9324     0.4155   9.464 1.49e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

4) Residual standard error: 15.38 on 48 degrees of freedom
Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
F-statistic: 89.57 on 1 and 48 DF, p-value: 1.49e-12
```

Note that the blue numbers are not part of the R output, but were added to help explain the output. There is a lot of information, so let's discuss each piece individually:

- 1) The first lines indicate the formula and variables used in the `lm()` function call, which is handy when you save the output and come back to it at a later time.
- 2) The next section shows the distribution of residuals in the form of quantiles, which gives a sense of the residual characteristics. The residuals are the difference between your data and the predicted regression line.
- 3) The next section shows the estimated linear regression coefficients and two hypothesis tests.

```

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -17.5791     6.7584  -2.601  0.0123 *
              3.9324     0.4155   9.464 1.49e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

The first column of the table shows each parameter in the model (intercept, slope). The name of the slope parameter will vary depending on what terms were in the original model. The next column gives the estimate for each parameter. In this example, the estimated intercept is  $a = -17.58$ , and the estimated slope for the speed covariate is  $b = 3.93$ . We are often interested in whether the slope of the regression line ( $b$ ) or intercept ( $a$ ) is significantly different from zero, which can be evaluated using a one-sample t-test. The next two columns show the standard error of the estimate for each coefficient, as well as the observed t-score. The t-test can be done by comparing the observed t-score against the critical t-score (using the `qt()` function). For this example, the critical t-score for the slope parameter is  $t_c = 2.01$  ( $df = 48$ , two-tailed,  $\alpha = 0.05$ ), which is less than the observed absolute value of  $t_o = 9.464$ , so we reject the null hypothesis that the slope is equal to zero. Equivalently, we can conduct the test using the p-value using the `pt()` function, which is shown in the last column for a two-tailed test. Since the p-value is less than  $\alpha = 0.05$ , we come to the same conclusion. R includes a series of graphical significance codes next to the p-values to give you a quick assessment of the significance of your regression parameters.

4) The last section of the output provides the Analysis of Variance information. The top line in this section gives residual standard error, which is a measure of the variation in the observations around the fitted line. The smaller this number the closer the observed values are to the fitted line. The next line is the  $R^2$  value of the regression, which you can think of as the percent variance explained. One problem with R-squared values is they tend to be artificially inflated with greater numbers of explanatory variables. If we want to compare  $R^2$  values between regression models with differing number of parameters, we need to take this effect into account and the adjusted  $R^2$  value provides a more accurate estimate of the percent variance explained. The final line shows the F-statistic for the test of whether the ratio of the explained variance over the unexplained variance is different from one. The F-test in a linear regression is the same as a t-test of whether the slope is different than zero, but this is not general to other types of statistical model. Using the `qf()` function, the critical F-score is  $F_C=4.04$ . Since the observed F-score is greater than the critical F-score, we reject the null hypothesis that the ratio is equal to one. Equivalently, we could perform the test using the p-value provided using the `pf()` function.

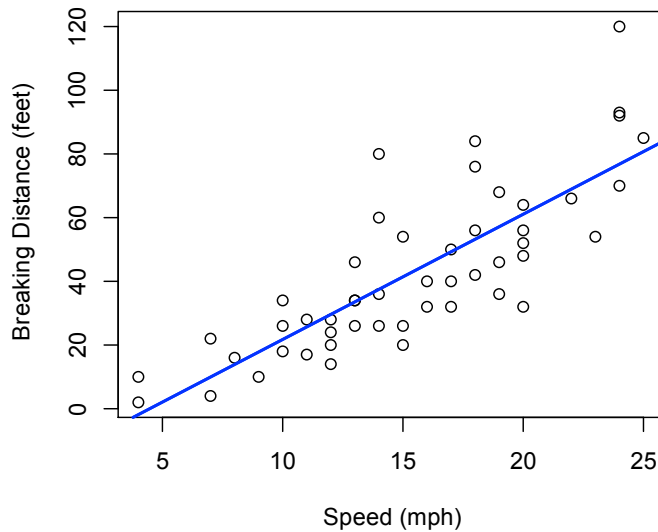
After fitting the linear regression, plot both the raw data and fit line to see if the fit statistical model makes sense. Begin by plotting the raw data

```
plot(speed,dist, xlab="Speed (mph)", ylab="Breaking Distance  
(feet)")
```

The fit linear regression can be added using the `abline(object)` function, which plots a line with the intercept (a) and slope (b) from the fit linear model given in *object*.

```
abline(MyFit, col='blue',lwd=2)
```

The plot should look like the figure below.



## Evaluating assumptions of linear regression

There are four assumptions that need to be met before the results of a linear regression can be trusted.

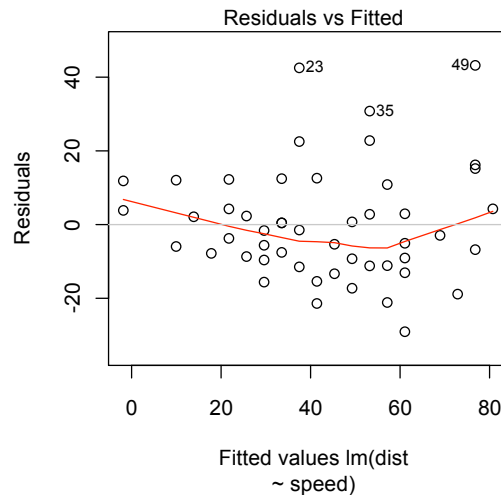
- 1) The relationship between the independent and dependent variable is linear
- 2) The residuals are Normally distributed
- 3) The residuals have equal variance across the range of the independent variable (heteroscedasticity)
- 4) The residuals are independent.

Evaluating the assumptions of linear regression begins with a qualitative assessment using two kinds of plots. The first is a plot of residual by predicted values, and the second is a histogram of residuals. A plot of the residuals (y-axis) by the predicted value (x-axis) allows you to visualize the assumptions of linearity and heteroscedasticity. This is done using the `plot(MyFit, type)` function, where `MyFit` is the fit linear regression object, and `type` is the diagnostic plot (`type=1` for residual plots). We write

```
plot(MyFit,1) #residual plot
```

The residual by predicted plot shows a moving average line (red). While the moving average is helpful for visualizing the qualitative trend, don't over interpret the patterns because we are just looking for large departures. The residual by

predicted plot for the *cars* data suggest that the residual variance increases with higher predicted values suggesting heteroscedasticity, but no strong indication that there is a departure from linearity. Make sure you can pick up these conclusions from the figure.



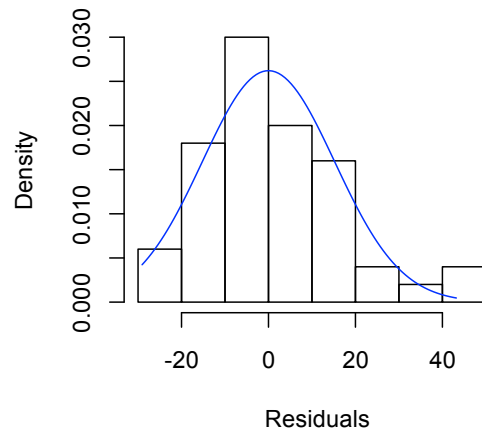
A histogram of the residuals can be used to evaluate qualitatively the assumption of Normality. Start by creating a histogram of residuals

```
hist(residuals(MyFit), main="", xlab="Residuals", freq=FALSE)
```

where the function `residuals(MyFit)` is used to extract the residuals from the `MyFit` object, and the new option `freq` tells the function to plot the proportion rather than counts. Since the plot shows the residuals on a proportional scale, we can add a line that represents what a normal distribution with the same standard deviation as the residuals would look like. Comparing the line against the histogram provides a visual evaluation for the assumption of Normality. To add the line, we create a new vector using the `dnorm()` function.

```
MyRes=residuals(MyFit)
xfit=seq(min(MyRes),max(MyRes),length=100) #new x variable
yfit=dnorm(xfit,0,sd(MyRes)) #predicted Normal lines(xfit,
yfit, col="blue") #add a blue line
```

The following figure shows the residual histogram and predicted Normal distribution for the *cars* data, and suggests that the assumption of Normality is reasonable.



The above two plots give a qualitative evaluation of linear regression assumptions. Quantitative evaluations of assumptions, however, are difficult with the exception of Normality. To test for Normality, we can perform a Shapiro-Wilkes test using the `shapiro.test()` function. The function uses the residuals from your fit model.

```
shapiro.test(residuals(MyFit))
```

```
Shapiro-Wilk normality test  
data:  residuals(MyFit) W = 0.92234, p-value = 0.1099
```

The last line of the output gives the test results. Since  $p > 0.05$ , we fail to reject the null hypothesis and conclude that there is not evidence of a deviation from Normality.

## Primer 8: Single-factor ANOVA

Analysis of variance (ANOVA) is similar to linear regression, but uses categorical rather than numerical variables as independent variables. In fact, there is no difference in the machinery used to fit ANOVA models and linear regressions, so we can use the now familiar `lm()` function. The only difference we need to worry about in R is how the data frame is structured, and some nuances of hypothesis testing. A single-factor ANOVA means that we are interested in one factor (e.g., nutrient concentrations), but there is more than two levels in the factor (e.g., 0 $\mu$ M, 1.2 $\mu$ M, 3.2 $\mu$ M, and 5 $\mu$ M of Nitrogen). Let's start by reminding ourselves of the long form data structure.

### Data frames for ANOVA

Similar to a two-sample t-test (which is just a one factor ANOVA with two levels!), the data must be in long form (also called stacked form) with one column as the response variable (dependent variable) and one column as a coded categorical variable (independent variable). Consider the following example that looks at the influence of substrate type on the growth of benthic algae. The table is presented in short form. Each column is a different substrate and each row is a replicate.

Sand	Silt	Pebbles	Glass
1.45	1.24	2.24	1.18
0.76	1.93	3.71	0.59
1.11	1.96	2.92	0.52
1.71	2.2	3.01	-0.74
0.97	3.93	6.33	-0.99

A long form version of the dataset can be created in Excel and then imported as a .csv file. Alternatively, it can be entered directly into R in long form using data frames. That is what we will do here. To create the long form data set, first create the independent and categorical vectors

```
growth=c(1.45,0.76,1.11,1.71,0.97,1.24,1.93,1.96,2.20,3.93,2.24,3.71,2.92,3.01,6.33,1.18,0.59,0.52,-0.74,-0.99)
```



```
substrate=c('sand','sand','sand','sand','sand','silt','silt',
'silt','silt','silt','pebbles','pebbles','pebbles','pebbles',
'pebbles','glass','glass','glass','glass','glass')
```

then combine these into a data frame

```
MyData=data.frame('growth'=growth, 'substrate'=substrate)
```

Type MyData to see the contents of the data frame, and compare this with the above table.

	growth	substrate
1	1.45	sand
2	0.76	sand
3	1.11	sand
4	1.71	sand
5	0.97	sand
6	1.24	silt
7	1.93	silt
8	1.96	silt
9	2.20	silt
10	3.93	silt
11	2.24	pebbles
12	3.71	pebbles
13	2.92	pebbles
14	3.01	pebbles
15	6.33	pebbles
16	1.18	glass
17	0.59	glass
18	0.52	glass
19	-0.74	glass
20	-0.99	glass

To get an impression of the influence of substrate type on algal growth, begin by plotting the data using the `boxplot()` function (see Primer 4 for more information)

```
boxplot(growth~substrate, ylab='Algal Growth Rate
(/d)',xlab='Substrate Type')
```

The plot suggests that algae have higher growth rates on pebbles, and lower on glass. Let's run the ANOVA model to see whether these trends are statistically significant.

## Fitting the ANOVA model

The ANOVA model is fit using the `lm()` function just as was done for linear regression, but the X variable is now categorical. This difference is dealt with 'behind the scenes' in R, so we do not need to make any changes to the formula.

```
MyFit=lm(growth~substrate, data=MyData)
```

The `data=MyData` options tells R where to look for the variables. The output is

```
summary(MyFit)
  Call:  lm(formula = growth ~ substrate, data
= MyData)

  Residuals:
      Min       1Q   Median       3Q      Max
-1.4020 -0.6545 -0.1600  0.4255  2.6880

  Coefficients:
              Estimate Std. Error t value Pr(>|t|)
1) (Intercept)    0.1120     0.4770   0.235  0.81733
2) substratepebbles  3.5300     0.6746   5.233  8.2e-05 ***
3) substratesand    1.0880     0.6746   1.613  0.12631
4) substratesilt    2.1400     0.6746   3.172  0.00591 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.067 on 16 degrees of freedom
Multiple R-squared:  0.6516, Adjusted R-squared:  0.5862
F-statistic: 9.973 on 3 and 16 DF,  p-value: 0.0006026
```

As with the two-sample t-test, the statistical tests presented are all relative to one factor level. In this example, they are all relative to the 'glass' treatment, which is shown by the fact that it is missing from all of the terms in the first column (e.g., `substratepebbles`). Let's go through the output line by line.

1. The first line is labeled (Intercept), which in this example is the glass substrate (analogous to the intercept in a two-sample t-test). The value under the 'Estimate' heading is the least square means (LS means) estimate from the fit model. The LS mean per-capita algal growth rate is 0.1120 per day. The t-test evaluates the hypothesis that the estimate is different from zero. It is a twotailed test, and may not always be the one you want to evaluate. In this example,

$p > 0.05$ , so we fail to reject the null hypothesis that the growth rate is not different from zero.

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	0.1120	0.4770	0.235	0.81733

- The second line is the first of the treatment levels relative to the glass treatment. The LS mean algal growth rate is 3.53 per day more than glass, which means the actual growth rate estimate is  $3.53 + 0.112 = 3.642$  per day. The t-test evaluates the hypothesis that the difference in algal growth rate relative to the glass is not zero. Since  $p < 0.05$ , we can conclude that the pebbles substrate had an influence on growth relative to glass.

substratepebbles	3.5300	0.6746	5.233	8.2e-05 ***
------------------	--------	--------	-------	-------------

- The third line is the next treatment level relative to the glass treatment. The LS mean algal growth rate is 1.088 per day more than glass, which means the actual growth rate estimate is  $1.088 + 0.112 = 1.2$  per day. Since  $p > 0.05$ , we fail to reject the null hypothesis.

substratesand	1.0880	0.6746	1.613	0.12631
---------------	--------	--------	-------	---------

- The last line is the final treatment level relative to the glass treatment. The LS mean algal growth rate is 2.14 per day more than glass, which means the actual growth rate estimate is  $2.14 + 0.112 = 2.25$  per day. Since  $p < 0.05$ , we can conclude that the silt substrate had an influence on growth relative to glass.

substratesilt	2.1400	0.6746	3.172	0.00591 **
---------------	--------	--------	-------	------------

The remainder of the output is interpreted the same as for a linear regression (see Primer 7 for details).

The output from the `summary()` function above provides t-test evaluations of the term against zero for each level. However, it is the F-table that will let you evaluate the general significance of the factor (substrate in this case) in the model. Use the `aov()` function to get the full F-table

```
summary(aov(MyFit))
          Df Sum Sq Mean Sq F value    Pr(>F)
substrate  3 34.033 11.3443   9.9726 0.0006026 ***
Residuals 16 18.201  1.1376
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The line of interest is the one labelled 'substrate', which is the hypothesis test for the substrate factor. Since  $p < 0.05$ , we reject the null hypothesis and conclude that the substrate had an influence on the per-capita algal growth rate.

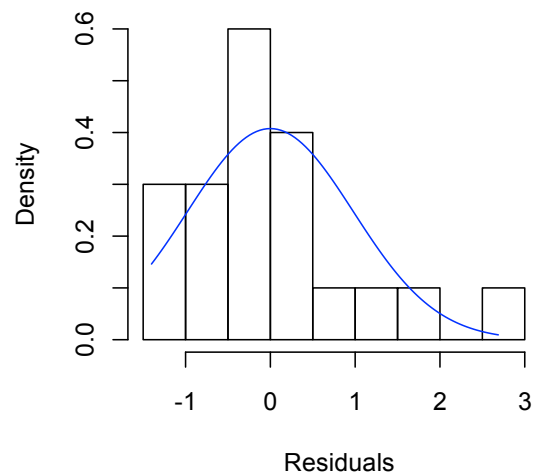
## Evaluating the assumptions of ANOVA

Evaluating the assumptions for one-factor ANOVA is similar to linear regression, but we only need to consider the assumptions of Normality and Homoscedasticity. To evaluate Normality, we use a histogram of the residuals and the Shapiro-Wilkes test for Normality. The residuals are plotted using the `hist()` function (see Primer 4 for details)

```
hist(residuals(MyFit),main="",xlab="Residuals", freq=FALSE)
```

and add the predicted Normal distribution using the same process as introduced in Linear regression (Primer 7).

```
MyRes=residuals(MyFit)
xfit=seq(min(MyRes),max(MyRes),length=100) #new x variable
yfit=dnorm(xfit,0,sd(MyRes)) #predicted Normal lines(xfit, yfit,
col="blue") #add a blue line
```



The residual plot suggests that the residuals are not symmetrical. To quantify this, we perform a Shapiro-Wilkes test for Normality using the `shapiro.test()` function. The function requires the residuals from your fit model, which is extracted using the `residuals()` function.

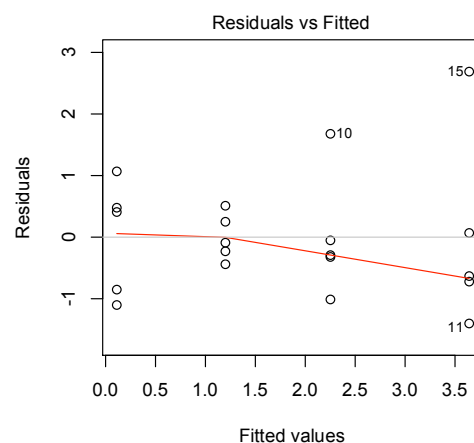
```
shapiro.test(residuals(MyFit))  Shapiro-Wilk normality test
data:  residuals(MyFit) W = 0.92234, p-value = 0.1099
```

The last line of the output gives the test results. Since  $p > 0.05$ , we fail to reject the null hypothesis and conclude that there is not evidence of a deviation from Normality.

To evaluate homoscedasticity, we use a residual plot and the Bartlett test. We can generate a residual plot using the `plot()` function on your fit statistical model.

```
plot(MyFit, 1)
```

Here the number 1 indicates that we want a residual plot. The resulting plot is



The x-axis of the figure is the predicted value shown in the summary output of the model, and the y-axis is the residuals. Looking at each of the four groups, the residuals seem to have different variances, so we might need to be concerned about homoscedasticity. To quantify this, we perform a Bartlett's test for homoscedasticity using the `bartlett.test()` function. This function requires the same formula used in the original call to the `lm()` function.

```
bartlett.test(growth~substrate, data=MyData)
Bartlett test of homogeneity of variances

data:  growth by substrate
Bartlett's K-squared = 5.9996, df = 3, p-value = 0.1116
```

The last line of the output gives the test results. Since  $p > 0.05$ , we fail to reject the null hypothesis and conclude that there is not evidence of a deviation from homoscedasticity.

## Group tests using TukeyHSD

The F-test indicated that the substrates had an impact on algal growth. To identify the groups that are different, we need to do a series of contrast statements. One approach is to use Tukey's Honest Significant Difference test (TukeyHSD), which compares all possible pair-wise contrasts. The TukeyHSD automatically adjusts for the multiple contrasts to maintain the family-wise Type I error rate, which means we can evaluate each test using the family-wise error rate (here it is 5%). The TukeyHSD test is run in R using the `TukeyHSD()` function.

```
TukeyHSD(aov(MyFit))
  Tukey multiple comparisons of means
    95% family-wise confidence level Fit:

aov(formula = my.fit)

$substrate
          diff          lwr          upr      p adj pebbles-
glass  3.530  1.6000921  5.4599079 0.0004306 sand-glass
1.088 -0.8419079  3.0179079 0.3994762 silt-glass      2.140
0.2100921  4.0699079 0.0272350 sand-pebbles -2.442 -4.3719079
-0.5120921 0.0110907 silt-pebbles -1.390 -3.3199079
0.5399079 0.2078944 silt-sand      1.052 -0.8779079  2.9819079
0.4276917
```

The `aov()` function is used to pass the proper elements of the ANOVA to the `TukeyHSD()` function. Each line of the output gives a test between two substrate types. For example, the fourth line tests the hypothesis that the growth rate is not different between the sand and pebble substrates. Since  $p < 0.05$ , we conclude that the per-capita growth rates are different between the two substrates.

## Group test using contrasts

With a large number of groups, the TukeyHSD test becomes problematic because the number of possible contrasts gets too large. A second approach is to use contrast statements and just test the group differences you are interested in. For this approach you adjust the Type I error rate of the contrasts ( $\alpha_C$ ) manually to ensure maintain the overall Type I error rate of the ANOVA ( $\alpha_F$ ). The contrasts are done using the `contrast()` function in R. The contrast function is in the 'contrast' library, so we will need to first install the library and then load the library. The contrast library is installed using

```
install.packages("contrast")
```

Note that you only need to do this once. After the library is installed, all you need to do is load the contrast library each time you want to use it. The library is loaded using the `library()` function

```
library("contrast")
```

To run the contrasts, we need to specify the groups to compare. This is done using `contrast(MyFit, a, b)`, where `a` and `b` are the group names and `MyFit` is your fit `lm` object. The `a` and `b` lists are setup so that `a` represents the list of group names for the left side of the contrast, and `b` represents the list of group names for the right side of the contrast. For example, if we wanted to compare the group pebbles with each of the groups glass, sand and silt, we would write

```
library("contrast")
contrast(MyFit, list(substrate="pebbles"), list(substrate="glass"))
contrast(MyFit, list(substrate="pebbles"), list(substrate="sand"))
contrast(MyFit, list(substrate="pebbles"), list(substrate="silt"))
```

where each line is it's own contrast. The above contrast statements give the following output

Contrast	S.E.	Lower	Upper	t	df	Pr(> t )
3.530	0.6745524	2.1000127	4.959987	5.23	16	0.0001
2.442	0.6745524	1.0120127	3.871987	3.62	16	0.0023
1.390	0.6745524	-0.0399873	2.819987	2.06	16	0.0560

The next step is to calculate the Type I error rate for each contrast.

```
alphaF=0.05
alphaC=1-(1-alphaF)^(1/3)
alphaC
[1] 0.01695243
```

Since there are three contrasts,  $\alpha_C=0.01695$ . With this adjusted criterion, we would reject the null hypothesis for the first two contrasts(pebbles-glass, pebbles-sand), and fail to reject the null hypothesis for the last contrast (pebbles-silt).

## Primer 9: Two-factor ANOVA

The previous primer considered ANOVA's where there was just a single factor. However, it is often the case that you are interested in the influence of two factors (e.g., nutrients and temperature), and whether they interact with each other (e.g., does high temperature amplify the impact of nutrients?). The overall analysis for two factors is similar to a single factor, but they differ in some of the details.

### Creating data frames for ANOVA

Similar to a single-factor ANOVA, the data must be in the long form with one column as the response variable (dependent variable) and two columns as coded categorical variables (independent variables). Consider the following example that looks at the influence of pine tree sap on the growth of wood fungus. Different species of pine trees contain different concentrations of antifungal agents. The following table shows growth rates (mm/day) of wood fungus in agar (a standard medium to evaluate growth) containing sap from two species in isolation, a mix of each, and a control (i.e., no tree sap). Each row is a replicate.

Treatment A <i>Pinus taeda</i>	Treatment B <i>Pinus pinea</i>	Treatment C <i>Pinus taeda &amp; Pinus pinea</i>	Treatment D Control
3.08	3.22	2.31	4.99
3.52	2.42	2.51	5.21
2.61	2.48	2.51	5.03
2.36	2.45	2.15	5.29
2.66	2.58	2.09	5.02
3	2.67	2.3	4.06

Since the data set has observations for all combinations of presence/absence for both species, it is a two-factor experiment as shown in the following table



		<i>Pinus taeda</i>	
		absent	present
<i>Pinus pinea</i>	absent	Treatment D	Treatment A
	present	Treatment B	Treatment C

To create the data set, first create the independent vector and the two categorical vectors.

```
Growth=c(3.08, 3.52, 2.61, 2.36, 2.66, 3.00, 3.22, 2.42, 2.48,
2.45, 2.58, 2.67, 2.31, 2.51, 2.51, 2.15, 2.09, 2.30, 4.99,
5.21, 5.03, 5.29, 5.02, 4.06)
```

```
P.taeda=c('yes', 'yes', 'yes', 'yes', 'yes', 'yes', 'no',
'no', 'no', 'no', 'no', 'no', 'yes', 'yes', 'yes', 'yes',
'yes', 'yes', 'no', 'no', 'no', 'no', 'no', 'no')
```

```
P.pinea=c('no', 'no', 'no', 'no', 'no', 'no', 'yes', 'yes',
'yes', 'yes', 'yes', 'yes', 'yes', 'yes', 'yes', 'yes',
'yes', 'no', 'no', 'no', 'no', 'no', 'no')
```

Here we have used 'yes' to denote that the tree sap was present for a species, and 'no' to denote absent. Even though two of the vectors are categorical, all three can be combined into a single data frame

```
MyData=data.frame('Growth'=Growth, 'P.taeda'=P.taeda,
'P.pinea'=P.pinea)
```

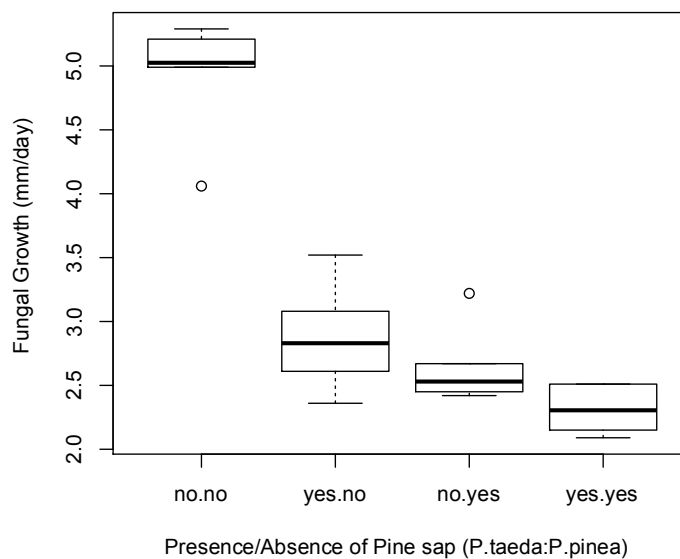
Type 'MyData' to see the contents of the data frame, and compare this with the two-factor table shown above.

```
MyData
  Growth P.taeda P.pinea
1   3.08    yes     no
2   3.52    yes     no
3   2.61    yes     no
4   2.36    yes     no
5   2.66    yes     no
6   3.00    yes     no
```

7	3.22	no	yes
8	2.42	no	yes
9	2.48	no	yes
10	2.45	no	yes
11	2.58	no	yes
12	2.67	no	yes
13	2.31	yes	yes
14	2.51	yes	yes
15	2.51	yes	yes
16	2.15	yes	yes
17	2.09	yes	yes
18	2.30	yes	yes
19	4.99	no	no
20	5.21	no	no
21	5.03	no	no
22	5.29	no	no
23	5.02	no	no
24	4.06	no	no

The new data frame has the same structure as we used for one-factor ANOVA's, but with two categorical variables. Let's begin by plotting the data with the `boxplot()` function as before

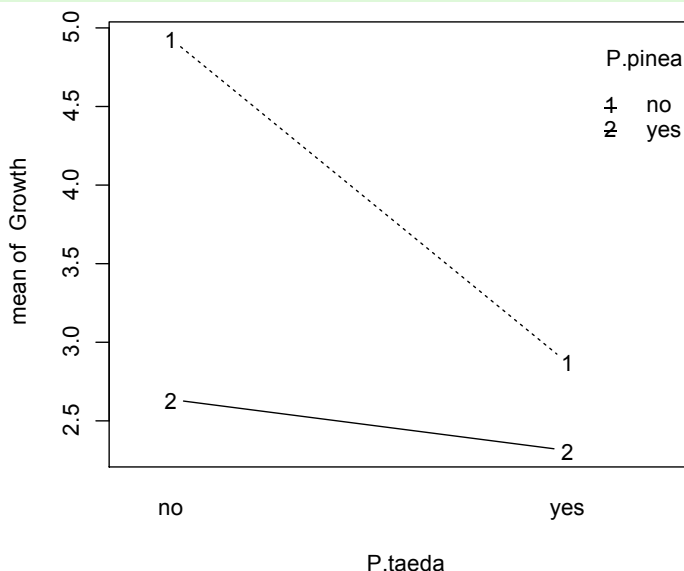
```
boxplot(Growth~P.taeda+P.pinea, ylab='Fungal Growth (mm/day)',
        xlab='Presence/Absence of Pine sap (P.taeda:P.pinea)')
```



It is easy to see that the fungal growth rate decreases with the presence of sap from either species of pine, but it is more difficult to see if there is an interaction

between the species. To see the interaction more clearly, we use the `interaction.plot(X1,X2,Y)` function, where X1 and X2 are the categorical predictor variables, and growth is the quantitative response variable. The interaction plot shows the mean of each treatment cell for each level in the X1 variable on x-axis. Each level of the X2 predictor variable is shown with a different style of line, and a different number.

```
interaction.plot(P.taeda,P.pinea,Growth, type='b')
```



The interaction plot shows that the presence of sap from either species of pine reduces the fungal growth rate, and that there is a negative interaction between the species such that the presence of both is not as effective as you would expect from the reduction in growth caused by the species in isolation.

## Fitting the ANOVA model

The ANOVA model is fit using the `lm()` function as before. For a two-factor ANOVA, we must add both predictor variables to the formula, as well as their interaction. This is done using the '+' and ':' symbols as `y ~ X1+X2+X1:X2`, where the '+' symbol separates the various independent variables. Writing the variable name denotes a main effect, and the ':' symbol denotes an interaction. A short form notation for the formula is to use the '\*' symbol as `y ~ X1*X2`, which represents both the additive and interactive effects. (i.e., `y ~ X1*X2` is the same as `y ~ X1+X2+X1:X2`).

```
MyFit=lm(Growth~P.taeda + P.pinea + P.taeda:P.pinea, data=MyData)
```

Alternatively, you can write

```
MyFit=lm(Growth~P.taeda * P.pinea, data=MyData)
```

The `data=MyData` options tells R where to look for the variables. The output is the same as for a single-factor ANOVA

```
summary(MyFit)
Call: lm(formula = Growth ~ P.taeda * P.pinea, data =
MyData)

Residuals:
    Min       1Q   Median       3Q      Max
-0.87333 -0.19292  0.01583  0.19833  0.64833

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
1) (Intercept)         4.9333     0.1428  34.548 < 2e-16
2) P.taedayes        -2.0617     0.2019 -10.209 2.23e-09
3) P.pineayes        -2.2967     0.2019 -11.373 3.49e-10
4) P.taedayes:P.pineayes  1.7367     0.2856   6.081 6.07e-06
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3498 on 20 degrees of freedom
Multiple R-squared:  0.9118, Adjusted R-squared:  0.8986
F-statistic: 68.96 on 3 and 20 DF, p-value: 1.006e-10
```

The output reveals that the sap from both pine species, as well as the interaction, are significantly different from the control. Let's work through this line by line.

1. The first line is labeled (*Intercept*), which in this example is the control treatment (analogous to the intercept in an independent t-test). The value under the 'Estimate' heading is the least square means (LS means) estimate from the fit model. The LS mean fungal growth rate is 4.93 mm/day. The t-test test is the hypothesis that the estimate is different from zero. It is a two-tailed test, and may not always be the one you want to evaluate. In this example, the growth rate of the control is different from zero ( $p < 0.05$ ).

```
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)         4.9333     0.1428  34.548 < 2e-16 ***
```

2. The main effect of *Pinus taeda* sap on fungal growth *relative* to the control. The LS mean fungal growth rate is 2.06 mm/day less than the control, which means the actual growth rate is 2.87 mm/day (try this for yourself). The t-test tests the

hypothesis that the difference in fungal growth rate relative to the control is not zero. Since  $p < 0.05$ , we can conclude that the *P. taeda* sap had an influence on fungal growth.

```
P.taedayes          -2.0617      0.2019 -10.209 2.23e-09 ***
```

3. The main effect of *Pinus pinea* sap on fungal growth relative to the control. The LS mean fungal growth rate is 2.30 mm/day less than the control, which means the actual growth rate is 2.63 mm/day. Since  $p < 0.05$ , we can conclude that the *P. pinea* sap had an influence on fungal growth.

```
P.pineayes          -2.2967      0.2019 -11.373 3.49e-10 ***
```

4. The final line shows the interaction between the *P. taeda* and *P. pinea* sap on fungal growth relative to the control and relative the main effects of each. If the two tree species were purely additive, then the fungal growth rate would be 4.36 slower than the control (2.06+2.30). The LS mean fungal growth rate is 1.74 mm/day faster than the purely additive case, which means the actual growth rate is 2.31 mm/day. The t-test tests the hypothesis that the interaction, which is the difference between the observed growth and the purely additive case, is zero. Since  $p < 0.05$ , we conclude that there is an interaction between the sap of the two species.

```
P.taedayes:P.pineayes  1.7367      0.2856   6.081 6.07e-06 ***
```

The remainder of the output is interpreted the same as for an ANOVA.

The output from the `summary()` function provides t-test evaluations of the term against zero, which is not always useful. The F-table, however, will let you evaluate the general significance of each factor in the model, rather than just a test against zero. To get the F-table, we can use the `aov()` function:

```
summary(aov(MyFit))
              Df  Sum Sq Mean Sq F value    Pr(>F)
P.taeda      1   8.5443   8.5443   69.839 5.900e-08 ***
P.pinea      1  12.2408  12.2408  100.054 3.149e-09 ***
P.taeda:P.pinea 1   4.5240   4.5240   36.978 6.066e-06 ***
Residuals    20   2.4468   0.1223
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The first line is the F-test for main effects of *Pinus taeda*, the second line is the F-test for the main effects of *Pinus pinea*, and the last line in the F-test for the interaction.

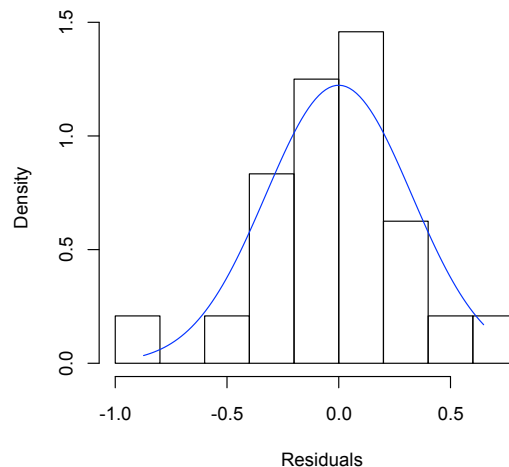
## Evaluating the assumptions of ANOVA

The evaluation of assumptions for two-factor ANOVA's is the same as for onefactor ANOVA's. To evaluate Normality, we use a histogram of the residuals and the Shapiro-Wilkes test for Normality. The residuals are plotted using the `hist()` function (see Primer 4 for details)

```
hist(residuals(MyFit),main="",xlab="Residuals", freq=FALSE)
```

and add the predicted Normal distribution using the same process as introduced in Linear regression (Primer 7).

```
MyRes=residuals(MyFit)
xfit=seq(min(MyRes),max(MyRes),length=100) #new x variable
yfit=dnorm(xfit,0,sd(MyRes)) #predicted Normal
lines(xfit, yfit, col="blue") #add a blue line
```



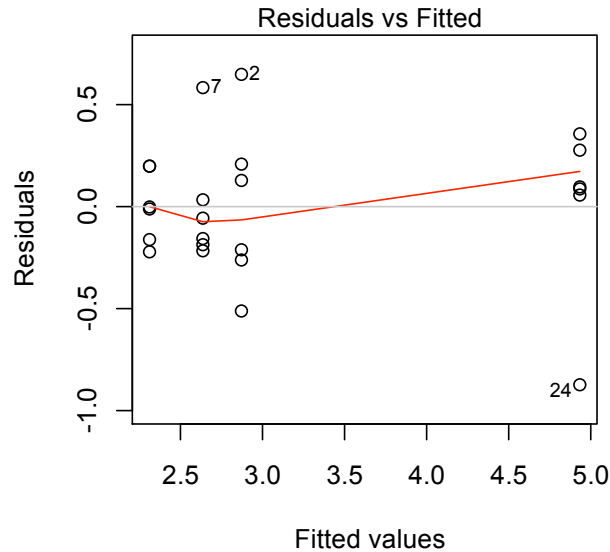
The residual plot suggests that the residuals are symmetrical. To quantify this, we perform a Shapiro-Wilkes test for Normality using the `shapiro.test()` function.

```
shapiro.test(residuals(MyFit)) Shapiro-Wilk normality test
data: residuals(MyFit) W = 0.96099, p-value = 0.4586
```

Since  $p > 0.05$ , we fail to reject the null hypothesis and conclude that there is not evidence of a deviation from Normality.

To evaluate homoscedasticity, we use a residual plot and the Bartlett test.

```
plot(MyFit,1)
```



The x-axis of the figure is the predicted value shown in the summary output of the model, and the y-axis is the residuals. Looking at each of the four groups, the residuals seem to have similar variances. To quantify this, we perform a Bartlett's test for homoscedasticity. Since there are two categorical variables, we need to use a slightly modified version of the function

```
bartlett.test(split(Growth, list(P.taeda, P.pinea)), data=MyData)
      Bartlett test of homogeneity of variances
data:  split(Growth, list(P.taeda, P.pinea))
Bartlett's K-squared = 4.0181, df = 3, p-value = 0.2595
```

The `split()` function is used here to indicate the cells for each combination of the categorical variables. The last line of the output gives the test results. Since  $p > 0.05$ , we fail to reject the null hypothesis and conclude that there is not evidence of a deviation from homoscedasticity.

## Group test using contrasts

The fit ANOVA model shows a significant interaction, which means that the sap from the two species of pine have a non-additive impact on fungal growth rates. It also means that the main effects (e.g., *P. taeda* influences fungal growth rates) are not directly interpretable because it includes treatments with the other species as well. For example, it could be that *P. taeda* in isolation does not influence fungal growth rates, but in the presence of *P. pinea* it does. Contrasts provide a way to disentangle interactions by comparing cells in the ANOVA table (e.g., *P. taeda* in isolation versus control).

Similar to single-factor ANOVA, contrast tests are done using the `contrast()` function. We begin by loading the contrast library

```
library("contrast")
```

The groups are specified by indicating the levels of interest for both categorical variables. In the following example we are testing the contrasts of whether adding the sap from both species together is different than each sap on it's own. This first one tests whether both saps are different from just *P.taeda*

```
#tests whether both saps are different from just P.taeda
a=list(P.taeda=c("yes"), P.pinea=c("yes"))
b=list(P.taeda=c("yes"), P.pinea=c("no"))
contrast(MyFit,a,b)

lm model parameter contrast
  Contrast      S.E.      Lower      Upper      t df Pr(>|t|)
    -0.56 0.201942 -0.9812435 -0.1387565 -2.77 20  0.0117
```

and this second one tests whether both saps are different from just *P.pinea*

```
#tests whether both saps are different from just P.pinea
a=list(P.taeda=c("yes"), P.pinea=c("yes"))
b=list(P.taeda=c("no"), P.pinea=c("yes"))
contrast(MyFit,a,b)

lm model parameter contrast

  Contrast      S.E.      Lower      Upper      t df Pr(>|t|)
    -0.325 0.201942 -0.7462435  0.09624355 -1.61 20  0.1232
```

The next step is to calculate the Type I error rate for each contrast.

```
alphaF=0.05
alphaC=1-(1-alphaF)^(1/2)
alphaC
[1] 0.02532057
```

Since there are two contrasts,  $\alpha_C=0.0253$ . Since  $p < \alpha_C$  for the first test (reject the null hypothesis), and  $p > \alpha_C$  for the second test (fail to reject the null hypothesis), we can conclude that having both saps together slows down fungal growth rates compared to just *P.taeda*, but not compared to just *P.pinea*. Have a look back at the interaction plot to see the effect size for these two contrasts.



## Reference Cards for R

### Some useful R tips

- `>` is the prompt from R on the console, indicating that it is waiting for you to enter a command.
- Each command entered by hitting `<return>` (`<enter>` on a Windows PC) is executed immediately and often generates a response; commands and responses are shown in different colours and this really helps when you are trying to find things on the console after a long session.
- When you hit `<return>` in the console (`<enter>` on a Windows PC) before a command is completely typed a `+` will appear at the beginning of the next line. Continue typing and hit `<return>` (`<enter>` on a Windows PC) when the command is finished to execute it.
- R is case sensitive. For example, `House` is a different variable from `house`.
- R functions are followed by `()`, with or without something inside the brackets depending on the function (e.g., `min(x)` gives the minimum value of the `x` vector).
- Don't worry about spaces around symbols in a typed line; they do not have any effect, so `win=3+4` is the same as `win = 3 + 4`.
- To assign variables, use the `=` assignment operator; `xxx = yyy` assigns `yyy` to a new object `xxx`; some advanced R users and books use `<-` instead of `=` but this is equivalent for our purposes and `=` is easier to remember and type.
- Pressing the up arrow in the console scrolls back through previous commands that you have entered; you can either press `<return>` (`<enter>` on a Windows PC) to execute the command again or edit it to correct a mistake. Very handy.
- If you make a mistake when typing and R returns 'syntax error' or has a '+' sign rather than the prompt, press the escape key. This will give you a fresh prompt.
- There is a Reference Card at the end of this primer that shows a list of common arithmetic operations and functions covered.

## Data functions

Function	Description	Primer
<code>c( x, y, z, ... )</code>	Creates a data vector from the <i>x</i> , <i>y</i> , <i>z</i> , ... entries.	1
<code>data.frame('X'=x , 'Y'=y, ....)</code>	Creates a data frame, which is a special array in R that gives names to columns of data. The text in quotes is the name, and the variable (e.g., <i>x</i> ) contains the data.	2
<code>factor( x , order )</code>	Used to reorder the factor levels in a categorical variable. <i>x</i> is the vector you want to reorder and <i>order</i> is the order you want the categories. Particularly useful for changing the order of factor levels on boxplots.	4
<code>file.choose()</code>	Brings up a menu that allows you to select a file. Don't enter anything in the braces. Use in conjunction with <code>read.csv()</code> .	2
<code>head( data ) , tail( data )</code>	The <code>head()</code> function displays the first 6 rows of the dataset stored in <i>data</i> , and the <code>tail()</code> function displays the last 6 rows.	2
<code>IQR( x )</code>	Calculates the interquartile range of <i>x</i> .	3
<code>library( name )</code>	Some commands are in specialized libraries, which are loaded using the <code>library()</code> function, where <i>name</i> is the title of the library.	6
<code>length( x )</code>	Returns the number of observations in <i>x</i> .	5
<code>levels( x )</code>	Returns the categorical levels in <i>x</i> .	4

Function	Description	Primer
<code>max( x ), min( x )</code>	Returns the maximum/minimum value in the x vector	1, 3
<code>mean( x ), median( x ), variance( x ), sd( x )</code>	Returns the mean, median, variance or standard deviation of the x vector.	1, 3
<code>names( data )</code>	Displays the column names of the dataset stored in <i>data</i> . Use the \$ sign to then access a particular column (e.g., <i>data \$MetabolicRate</i> ).	2
<code>plot( x , y )</code>	Creates a scatter plot of the x and y vectors. See <a href="#">Plotting options</a> to customize your plots.	1, 4
<code>quantile( x , y )</code>	Calculates the quantile(s) of x as described in y. y can be a number or vector, but only contain proportions.	3
<code>read.csv( location )</code>	Loads a .csv file from the source given in <i>location</i> . Use in conjunction with <i>file.choose()</i> .	2
<code>sample( x , size )</code>	Returns a random subset of length <i>size</i> from the entries in the x vector.	1
<code>sum( x )</code>	Returns the sum of all entries in the x vector.	1
<code>str( data )</code>	Displays information about the data stored in each column of the dataset <i>data</i> .	2
<code>subset( data , var==level )</code>	Creates a new data frame that is a subset of an original data frame called <i>data</i> . <i>var</i> is the column name of your variable, and <i>level</i> is a specific level in the variable.	2
<code>summary( x )</code>	Returns a number of descriptive statistics for the x vector.	3

**Arithmetic, indexing and Logic Operators**

Symbol	Description
+	Addition
-	Subtraction
/	Division
*	Multiplication
^	Raise to the power (i.e., $x^y$ is x raised to the power y)
=	Assigns a value
[, ]	Used to access entries of data structures such as vectors, arrays and matrices. Commas are used to separate dimensions.
()	Used for order of operations (i.e., do what is in the brackets first) and for the arguments of functions.

## Statistical functions

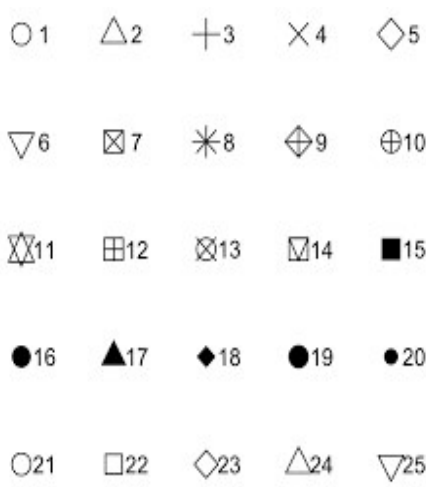
Function	Description	Primer
<code>aov( fit )</code>	Extracts the ANOVA table and F-test from a fit lm object.	7
<code>bartlett.test( y~x )</code>	Performs a Bartlett's test for homoscedasticity. The formula here is the same as for the original lm model.	7
<code>contrast( fit ,a,b)</code>	Performs a series of group contrasts from the lm object <i>fit</i> . The left side of the contrasts are given in <i>a</i> , and the right side are given in <i>b</i> .	7
<code>pbinom( x , n, s )</code> <code>qbinom( p , n, s )</code>	Calculates the probability of $X \leq x$ successes from a Binomial distribution with $n$ trials and a success rate of $s$ . <code>qbinom()</code> is the related function that calculates the event range $X \leq x$ for a given probability $p$ .	5
<code>pnorm( x , m, sd )</code> <code>qnorm( p , m, sd )</code>	Calculates the probability of an event $X \leq x$ from a Normal distribution with mean $m$ and standard deviation $sd$ . <code>qnorm()</code> is the related function that calculates the event range $X \leq x$ for a given probability $p$ .	5
<code>gvlma( MyFit )</code>	Used to assess the assumptions of the linear regression or ANOVA given in <i>MyFit</i> .	6
<code>qt(1-<math>\alpha</math>,df)</code> or <code>qt(1-<math>\alpha</math>/2,df)</code>	Displays the critical value from a $t$ -distribution given a confidence value and the degrees of freedom. For a one-tailed $t$ -test, the confidence value is $1-\alpha$ . The two-tailed test requires $1-\alpha/2$ for the right hand tail and $\alpha/2$ for the left hand tail.	5
<code>qchisq(1-<math>\alpha</math>,df)</code>	Displays the critical chi-squared value for a given confidence value and degrees of freedom.	5

Function	Description	Primer
<code>lm(y~1)</code> (one sample t-test)	Tests the null hypothesis that the data contained in the vector $y$ is significantly different from a population mean $\mu$ .	5
<code>lm(y~1)</code> (two sample paired t-test)	Test the null hypothesis that the difference between two populations, $y$ , is different than zero.	5
<code>lm(y~x)</code> (two-sample independent)	Two-sample t-test: Test the null hypothesis that the mean of two data sets are different.	5
<code>residuals( fit )</code>	Returns the residuals from the <code>lm</code> object in <i>fit</i> .	7
<code>shapiro.test(residuals( fit ))</code>	Performs the Shapiro-Wilkes test for Normality on residuals from the <code>lm</code> object <i>fit</i> . The <code>residuals()</code> function extracts the residuals from <i>fit</i> .	7
<code>summary( fit )</code>	This function extracts additional data from the <code>lm</code> object in <i>fit</i> and displays it on the screen.	5
<code>TukeyHSD(aov( fit ))</code>	Performs a Tukey's HSD test on the <code>lm</code> object <i>fit</i> . The <code>aov()</code> function is used to pass specific parts of the <code>lm</code> object to the <code>TukeyHSD</code> function.	7

## Plotting functions

Function	Description	Primer
<code>abline( h=a )</code> <code>abline( v=a )</code> <code>abline</code> <code>( MyFit )</code>	<p>Draws a horizontal ('h') or vertical ('v') line on a plot that crosses the axis at a.</p> <p>Alternatively, draws the fit linear regression contained in 'MyFit'.</p>	4, 6
<code>barplot( x )</code> <code>barplot( x ,</code> <code>beside=TRUE)</code>	<p>Creates a bar plot of the data in x. If x is a vector, then it is a simple bar plot. If x is a data frame, then each column of x is a separate group. The default is a stacked bar plot, but the option <code>beside=TRUE</code> can be used to create a grouped bar plot. See <a href="#">Plotting options</a> for details.</p>	4
<code>boxplot( x , y )</code>	<p>Creates a box plot of the data in x. If provided, y is a categorical vector describing the grouping of x (i.e., x can be a stacked vector). See <a href="#">Plotting options</a> to customize your plots.</p>	4
<code>hist( x )</code>	<p>Creates a histogram of the x vector. See <a href="#">Plotting options</a> to customize your plots. See <a href="#">Plotting options</a> to customize your plots.</p>	4
<code>legend(x,y,legend,col,pch)</code>	<p>Adds a figure legend. x and y give the location of the legend box (upper left corner), legend is a vector of names, col is for the colours, and pch indicates the type of symbol.</p>	4
<code>plot( x , y )</code>	<p>Creates a scatter plot of the x and y vectors. See <a href="#">Plotting options</a> to customize your plots.</p>	1, 4
<code>plot( fit , i )</code>	<p>Creates a diagnostic plot from the lm object in fit (ANOVA or regression). If <i>i=1</i>, the function returns a plot of the residuals. If <i>i=2</i>, the function returns a qqplot.</p>	7

### Plotting Options

Option	Description	Primer
xlim, ylim = c(a,b)	Specifies the axis limits to be plotted using a vector with two entries. The first entry 'a' is the minimum, the second entry 'b' is the maximum.	4
xlab, ylab = " "	Labels the axes with what is written in the quotes	4
main = " "	Labels the plot title with what is written in the quotes	4
col =	Specifies a color for the line/point/bar. Can be either a color name in quotes (e.g., 'red') or a number. Numbers one to five define colors as 1 black, 2-red, 3-green, 4-blue, 5-cyan, 6-pink, 7yellow and 8-grey. See <code>colors()</code> for a full list of color names.	4
lty =	Specifies the type of line as 1-solid, 2-dashed, 3dotted, 4-dotdash, 5-longdash, 6-twodash.	4
lwd =	Specifies the line width. Takes a positive number.	4
pch =	<p>Specifies the point type. Some useful values are</p> <p style="text-align: center;">  </p> <p style="text-align: center;">!</p>	4