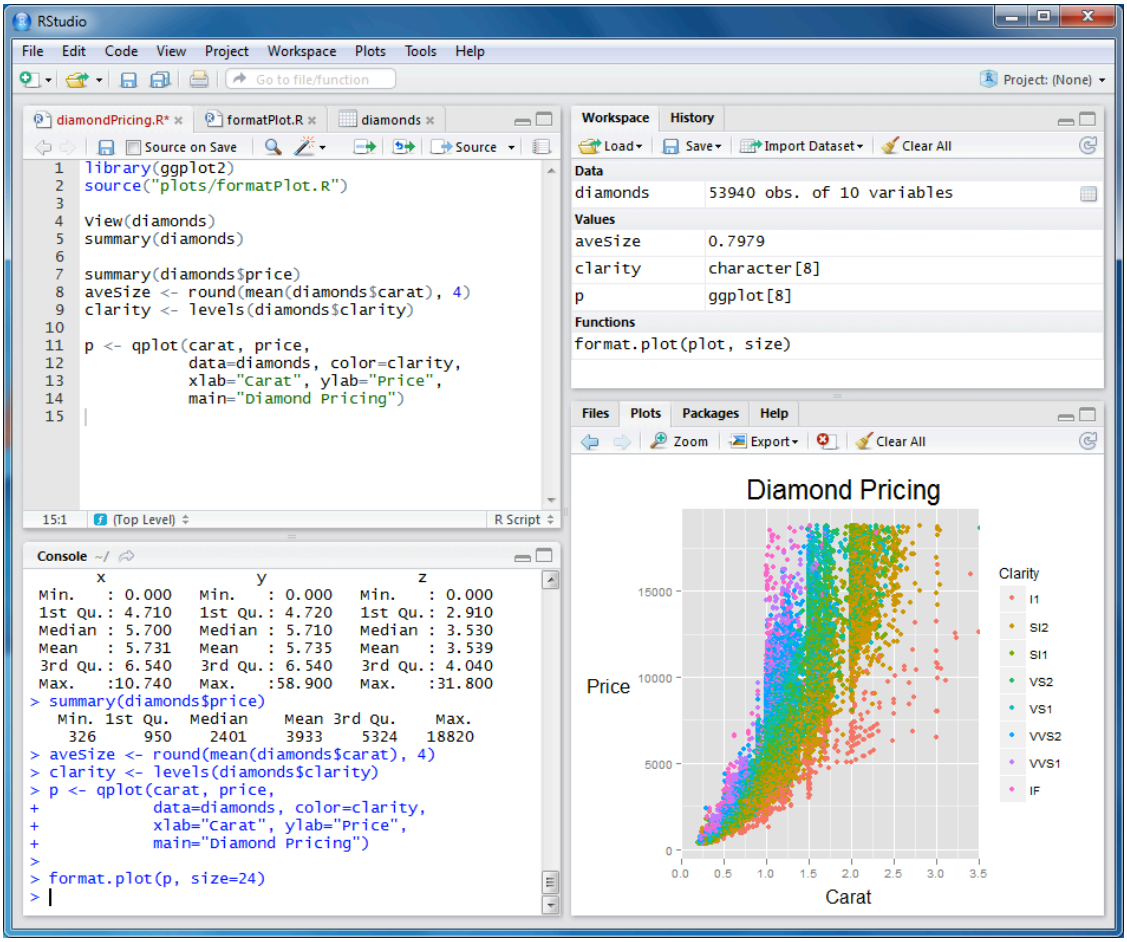
Getting Set Up

RStudio provides a nice environment for working with R, because you edit an R file, run commands and view data all in the same window.



RStudio

RStudio Download: https://www.rstudio.com/ide/

Parts of the RStudio window:

Bottom Left
Manually enter commands in the console on the bottom left. Upper Left

Write R code to a file and run it. Used for long commands you don't want to write by hand and saves work for future use. Upper Right

Toggle between command history and a summary of the variables and datasets you're using. There's an import file button. Bottom Right

Search for files, look at generated images, access help, and manage packages.

R Basics

R functions like a calculator. Pressing "Enter" or "Return" runs (multiple) lines of input.

```
> 1+2
[1] 3
> sqrt(15)
[1] 3.872983
```

> prefaces commands and output lines are labeled `[1]` or `[n]` for the number of output lines. The numbers reset between different lines of input. You get lines of output (`[1]`,`[2]`,`[3]`, etc) and when you enter a new command, the numbers start from `[1]` again. RStudio has blue input and black output.

Use the up arrow key to access previously made commands. Access older commands by tapping the up arrow key repeatedly.

Hit the "Run" button to run an entire file, or hit the command and enter key after highlighting the code that you want to run. The `help()` command is your friend.

```
> help()
```

`help()` pulls up a help file about any command put in the brackets. A question mark prefix also pulls up command's help page.

```
> ?lm
> help(lm)
```

If you put in some input, and when you try to run it, you get a plus sign rather than output, that means you have either an open parenthetical statement or an open bracketed command. Check your input.

One of the great things about R is that there are so many packages already available with commands that you may want to use. For example, "lattice" is a commonly-used package for data visualization. To install a package you can type:

```
> install.packages('lattice')
```

Note that the package name is in single quotation marks. Once this command is used once, you do not need to use it again.

After you do this, you can use the `require` command to make the functions in the package available. Usually you must use this command at the beginning of each session.

Generating Lists and Distributions

Let's look at some of the statistical tools that you may want to use in R. We'll start with `rnorm`:

```
> rnorm(1)
```

This will create a random number drawn from a random distribution centered around 0 with a standard deviation of 1 (the defaults). Now, let's change things around:

```
> rnorm(5, 2, 2.5)
```

This will randomly generate 5 numbers, of mean 2, with a standard deviation of 2.5. Related functions include `dnorm`, `pnorm`, and `qnorm`, which are density functions, distribution functions, and quantile functions. The defaults and syntax appropriate for each can be found by using

```
> help(rnorm)
```

The other modifiers are `log=FALSE` or `log.p = FALSE`, which are the defaults (don't need to be included), which, when modified to true, return the logarithm of probabilities rather than the probabilities themselves (useful in logistic regression). The last modifier is `lower.tail=TRUE`, which takes the probabilities as `P[X<=x]`, as default. Modification reverses it. You can store information in vector variables using:

```
> a = rnorm(5,10,2.5)
```

so that when you call `a`, it will have a vector of 5 randomly generated numbers of mean 10 with a standard deviation of 2.5

In R there are actually 2 ways of assigning value to variables. We could also have written:

```
> a <- rnorm(5,10,2.5)
```

Using an equal sign instead of the arrow may feel more intuitive for those of you who have programmed before, but knowing what the arrow does is important if you want to be able to read code that other people have written.

```
> b1 = c(1,2,3,4,5)
```

Another important command is the `c()` command for creating lists. This collects whatever you place in between the parenthesis, whether that's numbers, more vectors, or commands like rnorm. If you are entering sequential data, like in the case of variable `b`, you can always use this command:

```
> b2 = c(1:5)s
```

This will return the exact same output for `b2` as for `b1`, and if you're using large sequences (like `1:1000` for a for-loop, for example), this will make your life much easier. Call `b1` and `b2` to see if they're the same. A third way of entering this data, and a much more complete way would be to use the seq() command. Follow this example below:

```
> b3 = seq(1,5,1)
```

The first parameter is the starting number for the sequence, the second is the last, and the last parameter is how they count by. In this case, it will create a vector `b3`, counting from 1 to 5 by 1. If your sequence upper bound does not land exactly on the scheduled increase by the last parameter, the sequence will terminate at the highest value reached through sequential counting. For example, try

```
> seq(1,8,3)
```

The last command for putting in manual sequences of numbers is `rep`. Try

```
rep(1:5, 6)
```

This will repeat the numbers 1-5 six times. If you need them to be in order, you can modify it. The repetition quantity can be a vector, not just a number. Try:

```
> rep(1:2, c(5,10))
```

You should get:

```
[1] 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
```

A useful command is `sort()`, which will sort any vector you put into the parenthetical statement. Supposing you want to save the sorted list, you should use:

```
> f = sort(e)
```

Working with Data

First of all, we need to obtain the data that we are going to be working with. R reads delimited text files (csv, tsv, etc.). The most common of these is a csv (comma separated values) file. A csv file is just a text file with fields separated by commas. If you have a data file in Excel that you want to use in R, you can go to "Save as" in Excel and save it as a csv file. Similarly, if you want to bring in a `.dta` file from Stata, you can copy the data, paste it into an Excel file, and then save it as a csv.

RStudio makes it very easy to import datasets. If you go to the upper right hand corner, you will see a button that says "Import dataset". Follow the instructions to import the dataset. One of the questions that it will ask is whether there is a header, meaning whether the first line of file contains variable names or values. Be certain to indicate if you have a header, otherwise R will think that those are values in the dataset.

We're going to use one of the datasets that comes with R, the `tree` dataset, which contains the heights, girths, and volumes of a set of cherry trees. To tell R that that is the dataset we want to use, use the command:

```
> data(trees)
```

Once you do that, you should see "trees" in the upper right hand corner. If you want to look at the data, you can either click on it, or type:

```
> View(trees)
```

For larger datasets where you might want to look at only part of the dataset, you can use `head` function or the `tail` function. `head` shows lines at the beginning and `tail` shows lines at the end.

```
> head(trees)
  Girth Height Volume
1   8.3     70   10.3
2   8.6     65   10.3
3   8.8     63   10.2
4   10.5    72   16.4
5   10.7    81   18.8
6   10.8    83   19.7
```

In working with data, you have two options: you can work with the data *attached* or *unattached*. In attached data, you can refer to variables without specifying what dataset they are in. In unattached data, you have to specify. Attaching data can make things simpler when you are only working with one dataset, but it can get confusing if you want to use more than one. To attach data we can use the command:

```
> attach(trees)
```

Similarly, to detach, you can use:

```
> detach(trees)
```

Let's say you want to find the average of the height variable. With an attached dataset, you could type

```
> mean(Height)
```

For an unattached dataset, you must type:

```
> mean(trees$Height)
```

The `$` indicates that you are accessing the height variable in the `trees` dataset. I'm going to use detached data because it makes your code more understandable once you get used to using the `$`'s.

Now that we know how to work with variables, we can find out more about the data in our dataset. To get a summary of a variable, you can use the summary command:

```
> summary(trees$Height)
   Min.  1st Qu.  Median  Mean  3rd Qu.  Max.
     63       72      76    76       80    87
```

Now let's visualize the data.

```
> plot(trees$Girth, trees$height)
```

At its core, the plot function uses the first variable as the independent variable (x) and the second as the dependent (y). The plot function is one of the most essential functions in R, so lets spice it up a little bit with modifiers:

```
> plot(trees$Girth,
       trees$height,
       main="Girth as a Function of Height",
       xlab = "Height (Inches)",
       ylab = "Girth (Inches)",
       type = "p",
       pch = 2)
```

There are many other parameters that can be varied, including color, point size and shape, fit lines, fonts, tiling multiple graphs in one display (explained below), margin sizes, and pretty much everything else. This is all contained in the `par` command (see `help(par)`). As for the plot command above, it's all pretty self-explanatory, except for `type`, which selects point (`p`), line (`l`), or both (`b`), and `pch`, which selects dot type. If left out, it defaults to dots (try it!), `pch = 2` is triangles, and there are many more. They are conveniently missing from the help file, as well as the colors, because with each incarnation of R, they change. If you're interested in the most up to date list of supported colors and shapes (and their codes and names), check Google.

Another important plot used often in R is the histogram plot. To make a histogram, use:

```
> hist(trees$Girth,
       freq=NULL,
       xlab="Class",
       ylab="Frequency",
       main="Frequency of Girths")
```

Like the `plot` function, there are many other parameters that can modify this command, accessed by `help(hist)` . I have included a few of the important ones here. As you can see, the labels work the same way as in the plot function. The most important parameter in the hist function is the `freq` parameter. This determines whether your data is plotted as frequencies or a density histogram. There is also a parameter to allow you to define your own breaks using a predefined vector. I have not included it here for the sake of simplicity, but if you ever choose to represent data in an even remotely serious setting, you must look into this parameter and appropriately define your breaks. This is particularly true if you're using the density histogram rather than the frequency histogram.

Creating new variables is also relatively easy. Let's say that we want to create a variable that contains the height in feet instead of inches. We can use the command:

```
> trees$Heightfeet <- (trees$Height)/12
```

If we view the dataset, you will see that a new variable has been created:

```
> View(trees)
```

You may find that you only want to work with part of your dataset. Subsetting is an easy way to do this. Subsetting allows you to specify what rows and what columns you want to work with. If we were only interested in working with the first 10 rows of data, we could define a new dataset by writing:

```
> firsttrees <- trees[1:10,]
```

We are creating a new dataset called `firsttrees`. The `1:10` specifies that we want the rows from one to ten. *Notice the comma.* If you do not include a comma R will be confused. After the comma we could put specifications for what columns we want to include. Because I wanted to include all of the columns, that slot was left blank.

We can also subset conditionally. If we only only want trees where the girth is greater than 16 inches and the height is greater than 80 inches, we use the command:

```
> largetrees <-trees[trees$Height > 80 & trees$Girth >16,]
```

The & symbol allows us to put multiple conditions together. Similarly, the | symbol means "or".

Statistical Tests and Regression

The last topic we should probably cover is simple testing and regression. If you want more information on the height variable, you can run a one sample t-test. Let's say we want to test the hypothesis that the true mean value of the trees is equal to 75. The `mu` option allows you to set the value for the null hypothesis. The default value is 0.

```
> t.test(trees$Height, mu=75)

One Sample t-test

data:  trees$Height
t = 0.8738, df = 30, p-value = 0.3892
alternative hypothesis: true mean is not equal to 75
95 percent confidence interval:
 73.6628 78.3372
sample estimates:
mean of x
       76
```

There are many other statistical tests available. Another example is `binom.test`, which performs a binomial test. If we wanted to test the probability of getting 8 heads out of 10 coin tosses, we could use the command:

```
> binom.test(8,10,.5)

Exact binomial test

data:  8 and 10
number of successes = 8, number of trials = 10, p-value = 0.1094
alternative hypothesis: true probability of success is not equal to 0.5
95 percent confidence interval:
 0.4439045 0.9747893
sample estimates:
probability of success
                   0.8
```

Other tests include:

```
> chisq.test
> mantelhaen.test
> poisson.test
> fisher.test
```

These are the tests you will use 99% of the time, but should you need a rare one, use

```
> ??test
```

to access the full list of default tests available in R. Remember that you can also download packages for many more complicated tests.

To do simple linear regression, use the command:

```
> lm(trees$Girth~trees$Height)
```

The ~ means "as described by," so in this case, we would be modeling girth as described by height. This will return an intercept and a slope. If you're interested in more details, use:

```
> summary(lm(trees$Girth~trees$Height))
```

```
Call:
lm(formula = trees$Girth ~ trees$Height)

Residuals:
    Min      1Q  Median      3Q     Max
-4.2386 -1.9205 -0.0714 2.7450 4.5384

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  -6.18839    5.96020  -1.038  0.30772
trees$Height  0.25575    0.07816  3.272   0.00276 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.728 on 29 degrees of freedom
Multiple R-squared: 0.2697,Adjusted R-squared: 0.2445
F-statistic: 10.71 on 1 and 29 DF, p-value: 0.002758
```

This will return residual quartiles, t-values of each estimated parameter and the associated p-value, and also the R-squared value. For a moment, let's go back to our plot. Try this:

```
> plot(trees$Height, trees$Girth)
> abline(lm(trees$Girth~trees$Height))
```

You can of course modify this with the parameters you've learned earlier, but this will create a scatter plot and fit your model to that scatter plot. There are also commands to fit in error bars, make a qq plot, and plot confidence and prediction bands.

If you save your model, it will also automatically generate some of these plots:

```
> model <- lm(trees$Girth~trees$Height)
> plot(model)
```

At the command line you will see:

```
Hit <Return> to see next plot:
```

Hit Enter and you will see a series of plots (residual plot, qqplot, etc.) in the right hand corner of your window.

Notes: You'll notice that the dependent and independent variables are switched in location in the lm commands (the response variable comes before the predictor variable).

If you are really interested in making plots and figures, I recommend a package called `ggplot2`. It produces really dressy plots.

This barely scratches the surface. If you're interested in more complex analysis, R is capable of fitting linear and non-linear models, as well as survival analysis (Kaplan-Meier estimates and curves), multiple regression, Poisson and logistic regression, and non-parametric modeling such as the Wilcoxon rank sum and signed rank tests. In addition, R is capable of running simple programs and executing infinitely nested for loops, while commands, and nested if-then statements.

Further resources:

- Peter Dalgaard's Introductory Statistics with R
- http://www.r-project.org/. This website contains the official R documentation.
- http://cran.r-project.org/doc/manuals/R-intro.pdf. An Introduction to R
- The Internet! Chances are if you want to do something or are having a problem, someone else has written about it extensively. On-line forums can be very helpful.