R - Beginners Workshop Handout

Ria Pinjani, Dr. Amar Ahmad

03/02/2020

Course Directors: Dr. Omar El Shahawy, Dr. Raghib Ali

Program Coordinators: Maryse Bibeau, Meghan Durr

The purpose of this handout is to give you a heads up for what you should expect in your R - Beginners Workshop. For the duration of the workshop we will be covering the following topics;

1. Introducing R & RStudio
2. R Basics
3. Importing, viewing and describing and saving datasets in R
4. t-Test, Chi Squared test and Linear Regression in R
5. Basic Visualization

1. Introducing R & RStudio

What is R?

R is a programming language for statistical computing and graphics. It's open source, which means that it is free and open to the public.

R & RStudio

RStudio is a integrated development environment for R. The inferface enables users to view graphs, data tables, R code and output all at the same time. In figure 1 (Page 2); see how the R Studio interface looks like.

Installing R & Rstudio

For the purposes of this workshop, R & Rstudio will already be installed on your PCs. To download it on your personal computers, You can use the links below.

- Download and install R from http://cran.r-project.org
- Download and install RStudio from https://www.rstudio.com/products/rstudio/download/#download

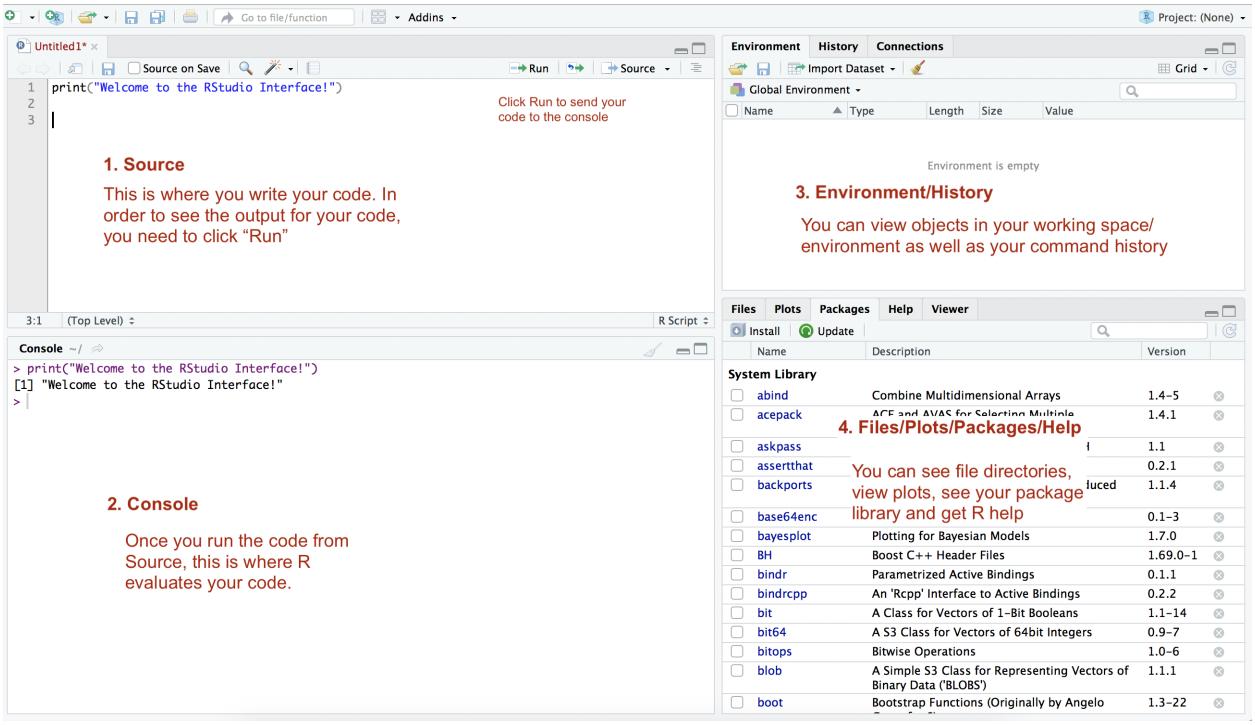


Figure 1: R Studio Interface

(You must install R before you can install RStudio)

What are R packages

R packages are collections of functions and data sets developed by the community. They increase the power of R by improving existing base R functionalities, or by adding new ones. For the purpose of this workshop we will be using functions in base R for the most part. We will make use of the package "readr" to demonstrate importing and saving datasets in R. In order to install and load a package in R; here is what you need to do.

```
install.packages("name_of_package")
library(name_of_package)
```

You can find a list of useful R packages here; https://support.rstudio.com/hc/en-us/articles/201057987-Quick-list-of-useful-R-packages

Getting Started

Once you open RStudio, you can create a new script by performing the following steps;

File –> New File –> R Script

2. R Basics

R as a calculator

Type the following on your R Source; it will give you the sum of $3 + 1 = 4$

```
3 + 1
```

This is what you will get on your Console when you Run the line above;

```
## [1] 4
```

You can take this a step further and perform a more complicated calculation like the one below;

```
((3+1)^2)/(5*0.5)
```

```
## [1] 6.4
```

Creating a variable

In R, we can create a variables and then perform arithmetic calculations on them. Lets say we want to create a variable for the number 10. To create this variable, we can type the below line of code. Notice that the assignment operator "<-", which consists of the two characters "<" ("less than") and "-" (minus) occuring strictly side by side and it "points" to the object recieving the value of the expression. Also keep in mind that we chose to name the variable "ten" here, we could have named it "x", "y", "TEN" etc.

Once you run the below line of code, the variable will be saved to your Environment.

```
ten <- 10
ten/5
```

We can also create this variable by using "="

```
ten = 10
ten/5
```

This is what you will get on your Console when you Run either of the lines above;

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

Suppose you want to create a list for the cups of coffee you had everyday for the past week. You can do this by using the "c" function. The c(...) function combines one or more values into a vector. It is always a good idea to name your vector in a way that represents the information in this vector. Again, the variable coffee_pastweek will be saved to your Environment once you run the line below.

```
coffee_pastweek <- c(2,3,2,1,4,1,1)
```

You can make your code more informative and clear by adding a comment! To add a comment simply type # and insert your text after. Let's add a comment to the above code. Adding the # ensures that the text will not be evaluated and simply printed in the Console as it is.

```
# cups of coffee consumed in the past week (17/02/2020 -23/02/2020) from monday to sunday
coffee_pastweek <- c(2,3,2,1,4,1,1)
```

Adding comments to your code helps you, and anyone reading your code understand it better.

You can view the contents of your vector by simply typing out the name of your vector in Source. The contents will be displayed on your Console.

```
coffee_pastweek
```

```
## [1] 2 3 2 1 4 1 1
```

You can also choose to view a single item in your vector. Let's suppose you want to look at the first element in your vector. You can do this by indexing;

```
coffee_pastweek[1]
```

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

Arithmetic Functions

R has biult in functions that you can use. These functions can take one or more arguments. Multiple arguments are seperated by commas.

The sqrt function allows you to take the square root of number.Let us take a look at the value we get in the console when we take a square roof of our variable ten.

```
sqrt(ten)
```

```
## [1] 3.162278
```

Here are some more examples of the many arithmetic functions one can use.

This provides a sum of all the elements in your vector coffee_pastweek, therefore 14 is the total cups of coffee for the entire week.

```
sum(coffee_pastweek)
```

```
## [1] 14
```

This provides a sum of all the elements in your vector coffee_pastweek and 1.

```
sum(coffee_pastweek, 1)
```

```
## [1] 15
```

This provides an average of all the elements in your vector, therefore 2 is the average cups of coffee per day.

```
mean(coffee_pastweek)
```

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

This gives us the standard deviation;

```
sd(coffee_pastweek)
```

```
## [1] 1.154701
```

This gives us the range (minimum and maximum value in our vector);

```
range(coffee_pastweek)
```

```
## [1] 1 4
```

Besides these, there are numerous other arithmetic and statistical functions in R that one can use.

Logical Operators

The table below presents logical vectors than can be used in R. These can be used along with & (and) / | (or).

Logical Operator	Definition
==	equal to
!=	not equal to
>	more than
>=	more than or equal to
<	less than
<=	less than or equal to

Logical Operator	Definition
%in%	contains

Below are some examples of how they can be put into use;

When did you drink more than 1 cup of coffee in a single day in the past week.

```
coffee_pastweek > 1
```

The ouput in the console is shown below. This indicates that you drank more than one cup of coffee on four out of seven days in the past week. (The result displays TRUE four out of seven times)

```
## [1]  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE
```

When did you drink 2 cups of coffee in a single day in the past week?

```
coffee_pastweek == 2
```

```
## [1]  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE
```

You drank 2 cups of coffee on two out of seven days in the past week. (The result displays TRUE 2 out of 7 times)

Is 9.99 equal to 10?

```
9.99 == 10
```

```
## [1] FALSE
```

9.99 is not equal to 10, and therefore the Console displays FALSE.

Constants

Here are some constants that you will eventually come across once you start using R.

Constant	Definition
pi	3.141593
Inf, -Inf	Positive and Negative infinity
NA	Not Available: Missing values
NaN	Not a Number: e.g. 0/0
NULL	Empty set

3. Importing, viewing and describing and saving datasets in R

Reading your data into R

To read data into R, You can go to File –> Import Dataset –> Choose file format.

Figure 2 on the next page shows the options that you will see on your screen.

Another way to read data into R is by running a line of code. For instance, to import a csv file, we can simply use the function read.csv in R and enter the file path in brackets. You must always remember to put the file path in quotation marks.

```
dat <- read.csv("MyH/MyFolder/dat.csv")
```

We can also use the function "read_csv" from the package readr.

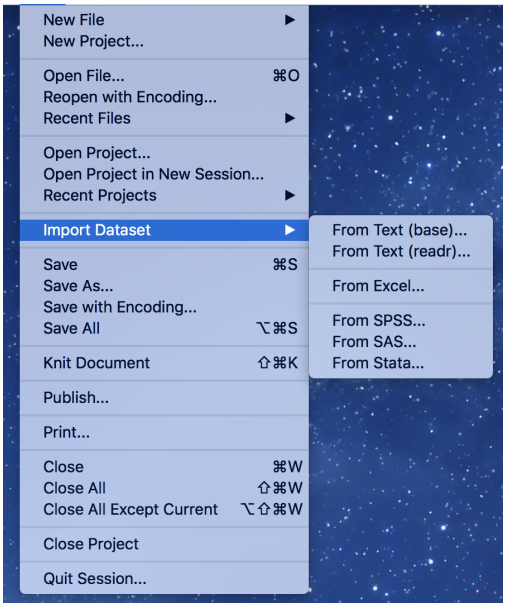


Figure 2: Importing Datasets

We will first install and load the readr package in R.

```
install.packages("readr")
library(readr)
```

Here is the code to import a csv file.

```
dat <- read_csv("MyH/MyFolder/dat.csv")
```

For the purpose of this workshop we will be using data on Smoking, Alcohol and Oesophageal Cancer. The dataset is called esoph. It is from a case-control study of oesophageal cancer in lle-et-Vilaine, France. We can save it in our global environment with the name "dat" for simplicity sake.

```
dat <- esoph
```

Display your data

You can use the function "View" to look at your data on a seperate tab in RStudio. Remember that R is case sensitive. Therefore when you run

```
view(dat)
```

Your console will show you the following error.

```
Error in view(dat) : could not find function "view"
```

You have to instead type;

```
View(dat)
```

OR

You can simply type esoph into your Source and view the data in your Console.

Describe your data

Let's look at the number of rows and columns in our dataset. We can do so by using the functions "nrow" and "ncol"

```
nrow(dat)
```

```
## [1] 88
```

```
ncol(dat)
```

```
## [1] 5
```

Another way of doing this is using the "dim" function in R. This shows us number of rows, followed by the number of columns.

```
dim(dat)
```

```
## [1] 88  5
```

We now know that our data set has 88 rows and 5 columns/variables.

To take a look at what variables we have; we can use the function "colnames"

```
colnames(dat)
```

```
## [1] "agegp"     "alcgp"     "tobgp"     "ncases"    "ncontrols"
```

As often is the case, the variable names do not clearly indicate what information they contain. For the purposes of this dataset, we can simply run ?esoph in our Source. Data description will be displayed in the Help tab on the bottom right corner of RStudio.

Here is a description of the variables;

1. agegp - Age group
2. alcgp - Alcohol consumption
3. tobgp - Tobacco consumption
4. ncases - Number of cases
5. ncontrols - Number of controls

You can use the summary function in base R to summarize the variables in your dataset. You can do so for one or more variables. Here we use it to describe our entire dataset.

```
summary(dat)
```

This is what will be displayed in your console;

```
##    agegp          alcgp         tobgp        ncases         ncontrols
##  25-34:15   0-39g/day:23   0-9g/day:24   Min.   : 0.000   Min.   : 1.00
##  35-44:15   40-79    :23   10-19   :24   1st Qu.: 0.000   1st Qu.: 3.00
##  45-54:16   80-119   :21   20-29   :20   Median : 1.000   Median : 6.00
##  55-64:16   120+     :21   30+     :20   Mean   : 2.273   Mean   :11.08
##  65-74:15                                3rd Qu.: 4.000   3rd Qu.:14.00
##  75+  :11                                Max.   :17.000   Max.   :60.00
```

The output shows that agegp, aclgp and tobgp are ordered factor/categorical variables. It also shows us the frequency for each group/level in all of these variables. for example; 15 individuals in our dataset belong to the age group 25-34; 15 individuals belong to the age group 35-44 and so on.

ncases and ncontrols are numeric variables. The variable ncases has a mean equal to 2.273 whereas the the variable ncontrol has a mean equal to 11.08.

You can also choose to summarize only one variable in your dataset. How do you select one variable from your data set? You can do this by typing out data$variable_name.

Let's try below.

```
summary(dat$agegp)
```

```
## 25-34 35-44 45-54 55-64 65-74   75+
##    15    15    16    16    15    11
```

One can also use the R function table instead of summary as it is the age groups is a categorical variable

```
table(dat$agegp)
```

```
##
## 25-34 35-44 45-54 55-64 65-74   75+
##    15    15    16    16    15    11
```

You can also use the "class" function to see what type of variables you are dealing with.

```
class(dat$agegp)
```

```
## [1] "ordered" "factor"
```

```
class(dat$ncases)
```

```
## [1] "numeric"
```

You might want to ask R a question about your dataset. Here is how you can do that;

Is the variable agegp a factor variable?

```
is.factor(dat$agegp)
```

```
## [1] TRUE
```

Is the variable agegp an ordered factor variable?

```
is.ordered(dat$agegp)
```

```
## [1] TRUE
```

Add a variable to your dataset

We can add a variable to our dataset as well. Suppose we want to add a variable that represents the ratio of cases to controls (we can divide number of cases by number of controls). For our code we will type dat$ followed by the name we want to give this variable; followed by our usual assignment operator; followed by what the variable represents.

```
dat$ratio <- (dat$ncases)/(dat$ncontrols)
```

Once we run the above line of code; our dataset should not have 6 variables. Let's check!

```
dim(dat)
```

```
## [1] 88  6
```

```
colnames(dat)
```

```
## [1] "agegp"     "alcgp"     "tobgp"     "ncases"    "ncontrols" "ratio"
```

The variable ratio has been added to our dataset.

Saving your dataset

You can use the function write_csv from the package "readr" to save your new dataset into your local file. Inside the function write_csv(), you should type out the name of your dataset (dat), followed by a comma, followed by the file path of where you want to save your dataset. (You do not need to install or load a package again if you already have done that in the same R session before)

```
install.packages("readr")
library(readr)
```

```
write_csv(dat,"MyH/MyFolder/mydat.csv")
```

4. t-Test, Chi Squared test and Linear Regression in R

t-Test

The one sample t.test compares the mean of a single sample to a known or hypothesized value to determine if the sample mean is significantly greater or lesser than that value. We can use the "t.test" function in R to see if the mean for ratio is significantly greater or lesser than 0.5.

Below is the line of code you can run; (You can run ?t.test in Source to see other arguments that you can add to your function)

```
t.test(dat$ratio , mu = 0.5)
```

The result is as follows;

```
##
##  One Sample t-test
##
## data:  dat$ratio
## t = -4.0216, df = 87, p-value = 0.0001228
## alternative hypothesis: true mean is not equal to 0.5
## 95 percent confidence interval:
##  0.2710932 0.4225205
## sample estimates:
## mean of x
## 0.3468068
```

In the result above :

- t is the t-test statistic value (t = -4.0216),
- df is the degrees of freedom (df= 87),
- p-value is the significance level of the t-test (p-value = 0.0001228).
- conf.int is the confidence interval of the mean at 95% (conf.int = [0.2710932, 0.4225205]);
- sample estimates is the mean value of the sample (mean = 0.3468068).

The p-value of the test is 0.0001228, which is less than the significance level alpha = 0.05. We can conclude that the mean ratio of cases to controls is significantly different from 0.5 with a p-value = 0.0001228.

Chi-squared tests

The Chi-squared test is a statistical method which can be used to determine if two categorical variables have a significant correlation between them.

```
chisq.test(dat$tobgp , dat$alcgp)
```

```
## Warning in chisq.test(dat$tobgp, dat$alcgp): Chi-squared approximation may be
## incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  dat$tobgp and dat$alcgp
## X-squared = 0.61946, df = 9, p-value = 0.9999
```

In the result above :

- X-squared is the Chi-squared statistic value (X-squared = 0.61946),
- df is the degrees of freedom (df= 9),
- p-value is the significance level of the Chi-squared test (p-value = 0.9999).

As the p-value 0.9999 is greater than the .05 significance level, we do not reject the null hypothesis that the alcohol consumption for an individual is independent of their tobacco consumption.

Simple Linear Regression

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. A linear regression line has an equation of the form Y = a + bX, where X is the explanatory variable and Y is the dependent variable. The slope of the line is b, and a is the intercept (the value of y when x = 0). We can use the function "lm" for linear regression in R.

```
my_linear_reg <- lm(ratio ~ unclass(agegp), data = dat)
```

Let's now look at a summary of our linear regression

```
summary(my_linear_reg)
```

```
##
## Call:
## lm(formula = ratio ~ unclass(agegp), data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.64861 -0.18672 -0.07125  0.20257  0.92875
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -0.04422    0.07434  -0.595    0.554
## unclass(agegp)  0.11547    0.01976   5.845 8.88e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3041 on 86 degrees of freedom
## Multiple R-squared:  0.2843, Adjusted R-squared:  0.276
## F-statistic: 34.16 on 1 and 86 DF,  p-value: 8.884e-08
```

In the summary above (among other statistics);

- Residuals have a median value equal to -0.07125; range from -0.64861 to 0.92875
- Intercept for our model is -0.04422 | p - value equal to 0.554
- Coefficient estimate for agegp is 0.11547 | p - value < 0.001

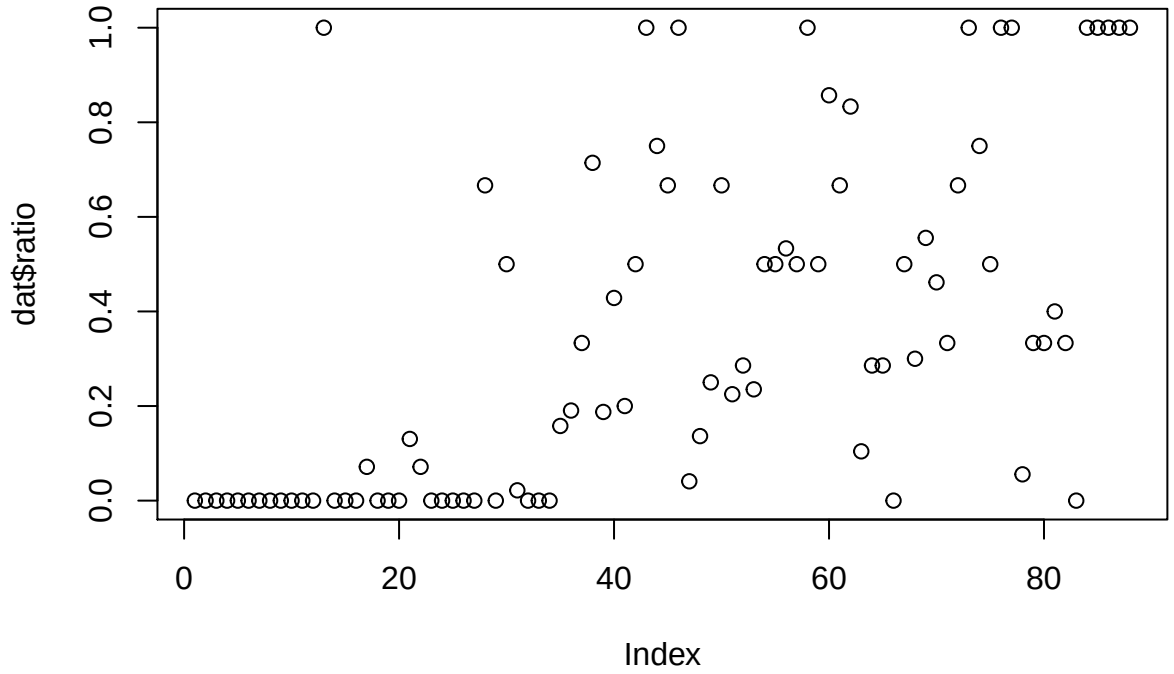
The intercept represents the average ratio of cases to controls when x is 0, since agegp cannot be zero - it does not present any meaningful explanation for us. We use a linear effect for age group with the "unclass" function; as age group changes to one unit or level higher; mean ratio of cases to controls increases by approximately 0.115 units.

5.Basic Visualization

Visualizing your data in base R

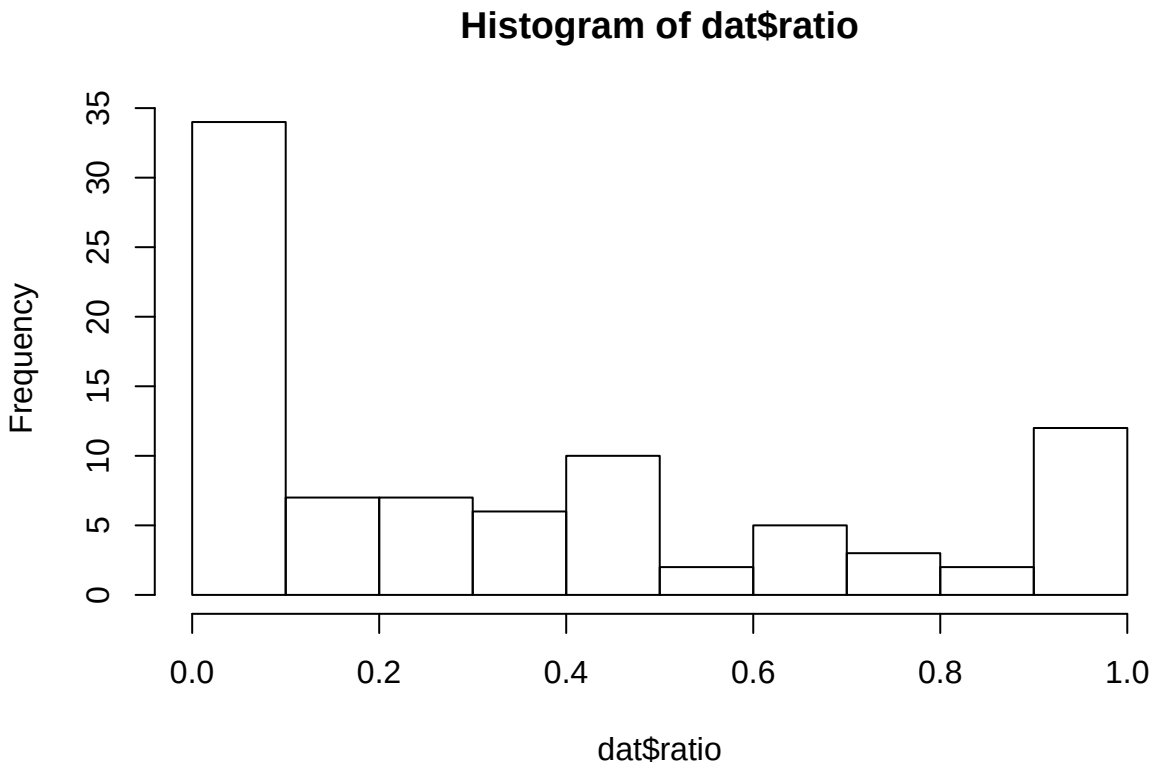
You can plot a single variable in R using the "plot" function. The below line of code gives you a scatter plot of the variable ratio. The plot will be displayed on the Plots tab in the bottom right section of RStudio.

```
plot(dat$ratio)
```



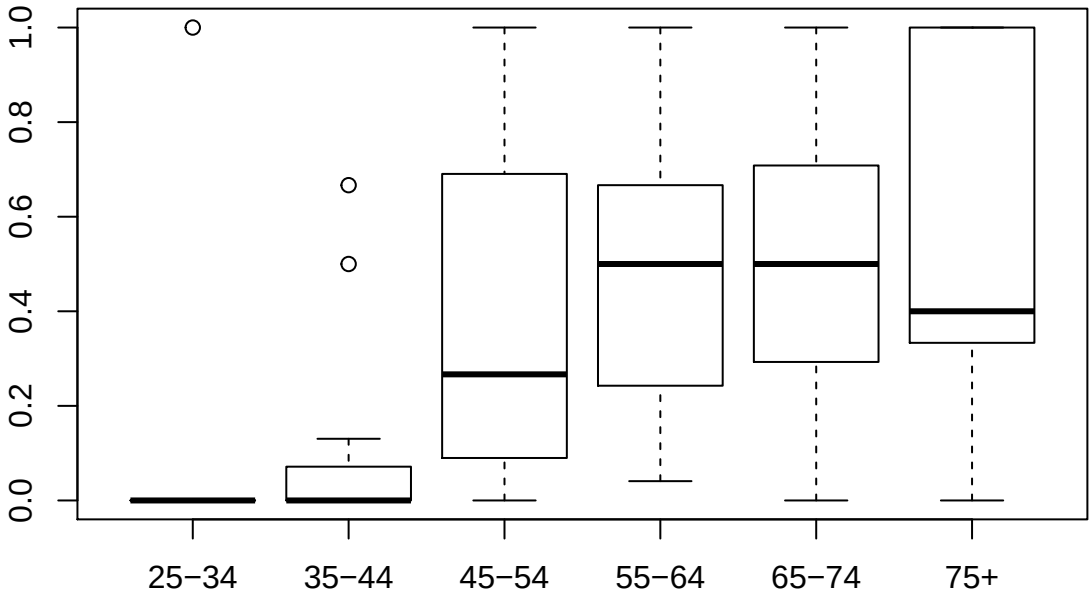
After seeing this plot; you may realize that it does not describe the data very well. Let's look at a histogram instead. You can plot a histogram using the "hist" function.

```
hist(dat$ratio)
```



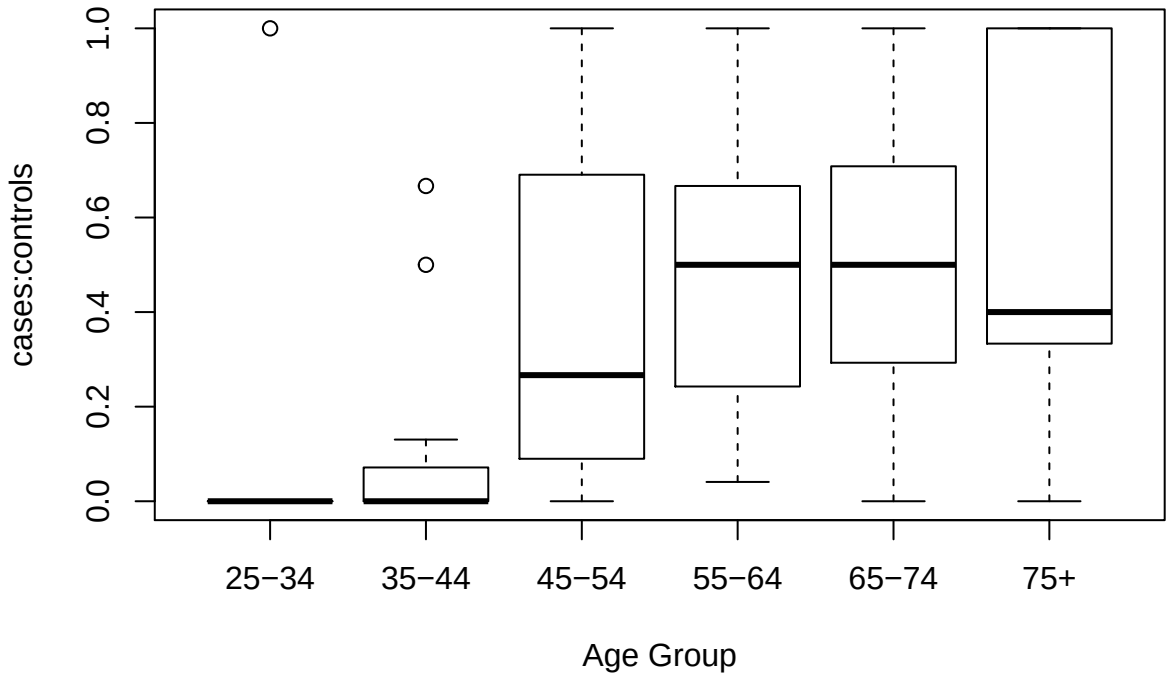
You can also plot two variables in R. The below line of code will give you agegp on the x axis and ratio on the y axis. It shows a box plot by default.

```
plot(dat$agegp, dat$ratio)
```



You can make your plot more informative/clear by adding a heading and labels for the x & y axis.

```
plot(dat$agegp, dat$ratio, main = "Ratio of cases:controls by Age Group",
     xlab = "Age Group", ylab = "cases:controls")
```



Run ?plot to look at the various arguments you can add to your function.

Let's Wrap up

We hope that the above information gets you more aquainted with the RStudio interface and helps you a get started with statistical analysis in R. We are going to wrap things up for the purpose of this handout. For the workshop; we will be running this code live and looking at the output, while using the same dataset.You are expected to run the same code on your own desktop as well - this is simply because you do learn R best by trying yourself! We will be helping you throughout the process.

Other Resources

- An Introduction to Statistical Learning with Applications in R | Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani
- Hands-On Programming with R | Garrett Grolemund
- stackoverflow.com will have the answer to most of your R questions.