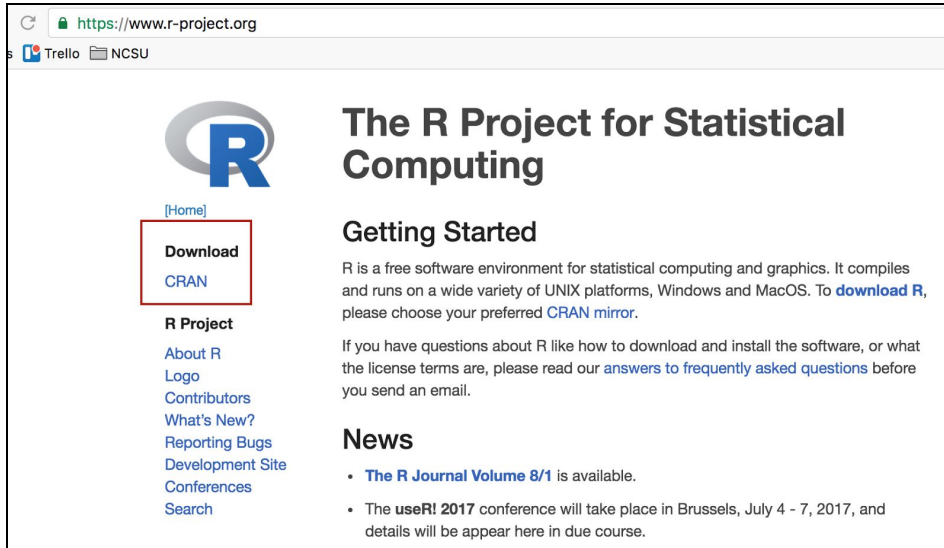


# Workshop Instructions

## Setup: Getting R in Your Machine


### Downloading and Installing R

#### 1. Go to - <https://www.r-project.org/>



The screenshot shows the homepage of The R Project for Statistical Computing. The browser address bar displays <https://www.r-project.org>. The page features the R logo, a navigation menu on the left with a red box around the 'Download CRAN' link, and a 'Getting Started' section. The 'Getting Started' text states: 'R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To **download R**, please choose your preferred [CRAN mirror](#).' Below this, there is a 'News' section with two bullet points: 'The R Journal Volume 8/1 is available.' and 'The useR! 2017 conference will take place in Brussels, July 4 - 7, 2017, and details will be appear here in due course.'

#### 2. Select one of the CRAN mirrors listed under USA -- Duke University is closest

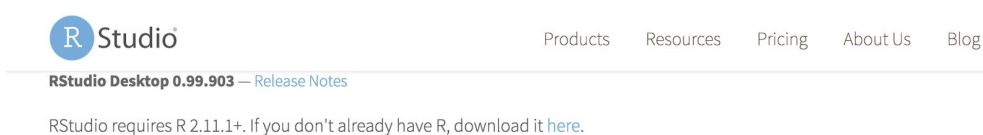


The screenshot shows the CRAN mirrors page at <https://cran.r-project.org/mirrors.html>. The page lists mirrors for various countries, with the USA section highlighted. The mirrors listed for the USA are:

URL	Location
<a href="http://mirror.mdx.ac.uk/R/">http://mirror.mdx.ac.uk/R/</a>	Middlesex University London
<a href="http://star-www.st-andrews.ac.uk/cran/">http://star-www.st-andrews.ac.uk/cran/</a>	St Andrews University
<a href="https://cran.cnr.berkeley.edu/">https://cran.cnr.berkeley.edu/</a>	University of California, Berkeley, CA
<a href="http://cran.cnr.berkeley.edu/">http://cran.cnr.berkeley.edu/</a>	University of California, Berkeley, CA
<a href="http://cran.stat.ucla.edu/">http://cran.stat.ucla.edu/</a>	University of California, Los Angeles, CA
<a href="https://mirror.las.iastate.edu/CRAN/">https://mirror.las.iastate.edu/CRAN/</a>	Iowa State University, Ames, IA
<a href="http://mirror.las.iastate.edu/CRAN/">http://mirror.las.iastate.edu/CRAN/</a>	Iowa State University, Ames, IA

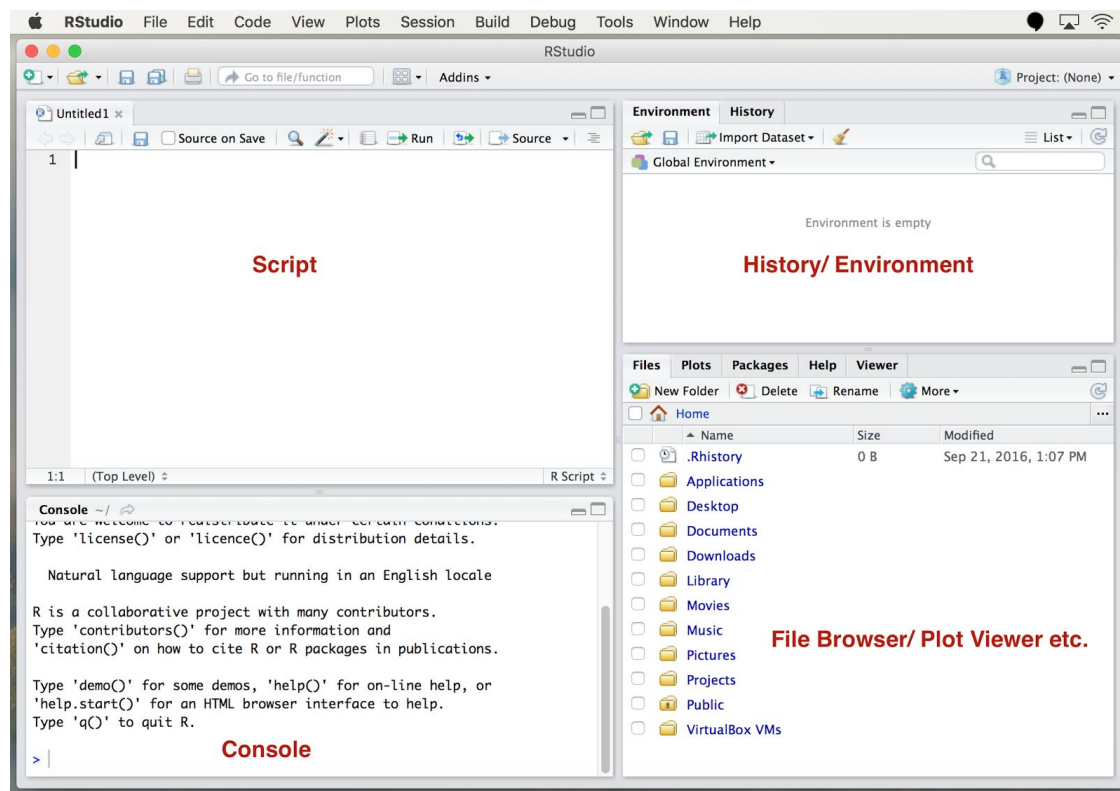
#### 3. Download the R Version for your operating system (Windows, Mac, or Linux)

#### 4. Download R Studio - <https://www.rstudio.com/products/rstudio/download3/>



The screenshot shows the RStudio download page. The header includes the RStudio logo and navigation links: Products, Resources, Pricing, About Us, and Blog. The main content area displays 'RStudio Desktop 0.99.903' with a link to 'Release Notes'. Below this, a note states: 'RStudio requires R 2.11.1+. If you don't already have R, download it [here](#).'

## Understanding the structure of R Studio



## Activity 1

### 1. Understanding prompts:

- In console a new line starts with `>`. This means it is waiting for us to communicate

### Type 2+3 in the Console and hit Enter:

```
> 2+3
```

```
[1] 5
```

- If we give it an incomplete command then it returns `+`. Press `esc` button to return to a new line.

### Type 2+3+ in the Console and hit Enter:

```
> 2+3+
```

```
+
```

Now type 7 and hit Enter:

```
> 2+3+  
+ 7  
[1] 12
```

Hit the up arrow until you see 2+3+ pop up (the up arrow lets you access previous commands). Hit enter.

Now press ESC. This starts a new prompt.

```
> 2+3+  
+  
  
>
```

## 2. Set working directory (Get default directory/Grab path)

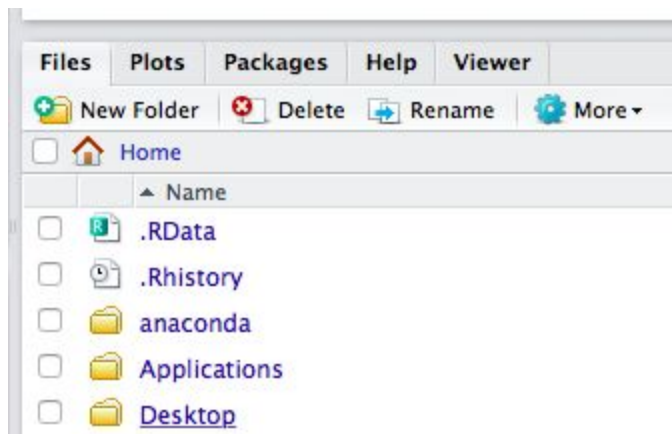
In the console type `getwd()`.

This will return the current working directory, listed as a file path.

```
> getwd()  
[1] "/Users/ablaine"
```

Let's create a new folder and set that as our working directory.

in In the lower-right corner File Browser menu, double Click on the Desktop Folder

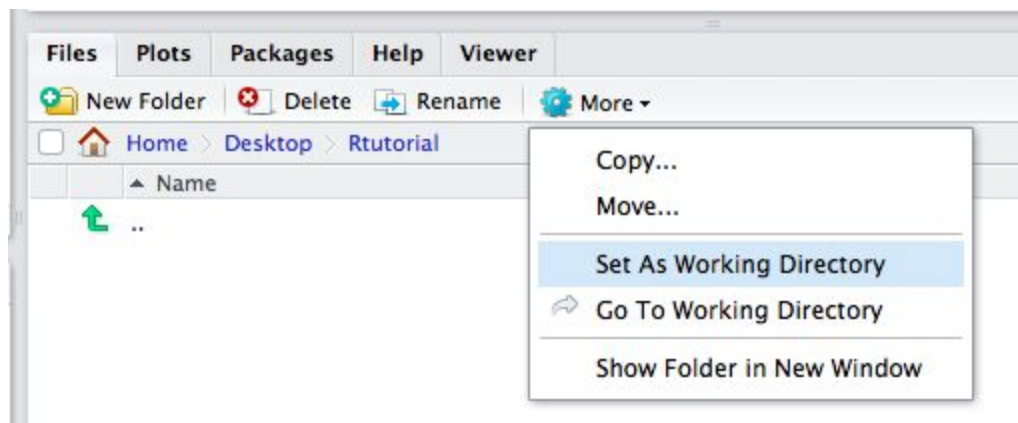


Click on “New Folder” to create a new folder on your computer’s Desktop

Name the folder “Rtutorial”. Click OK  
Double click on Rtutorial folder title to go inside of the folder



**Click More, then Set As Working Directory**



**When you do that, your Console should have this output:**

```
> setwd("~/Desktop/Rtutorial")
```

**\*To set your working directory manually:**

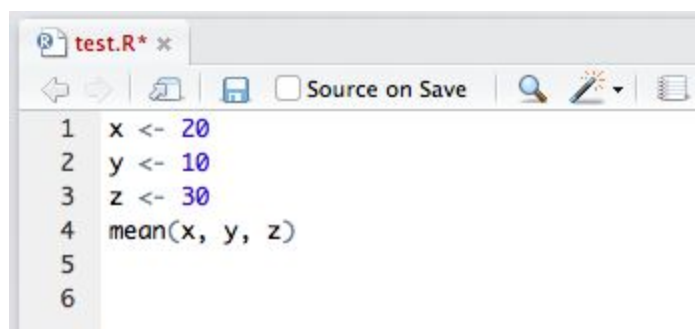
**setwd()** - To assign a new working directory copy the path of the directory where you would like to save your codes (In Mac right click on the folder and it will show the path under “where”). Now type `setwd(“copied_path_to_your_directory”)` in console. This will set the directory. Now if you type `getwd()` again, it will return the directory you pointed to. You can also do the same thing from the menu - [Session > Set Working Directory > Choose Directory](#)

### 3. Writing a script and saving it

Write the following script in the text editor at the top left of your screen:

```
x <- 20
y <- 10
z <- 30
mean(x, y, z)
```

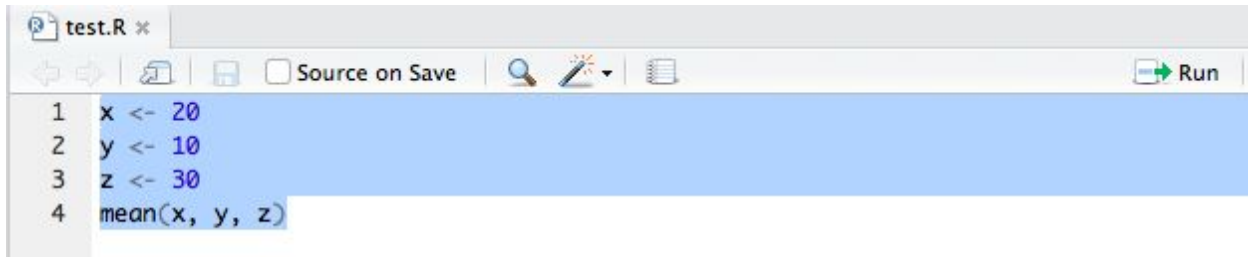
**Click on File > Save As... to save the script. Name it “test.R”**



When you try to save it, you will see it will automatically save in the directory you assigned.

#### 4. Running a Script

Highlight the Script and Click Run:



```
test.R x
1 x <- 20
2 y <- 10
3 z <- 30
4 mean(x, y, z)
```

This runs each line of code, in the order listed. You can run one or more lines.

#### 5. Doing basic calculations on R (Arithmetic/ Built-in functions)

The order of arithmetic operations is (left [done first] to right [done last]) : ^ / \* - +

^ is used for raised to the power of, followed by division, multiplication, subtraction and addition.

**Copy and Paste these built-in Functions into your R script, then run the code!**

pi

exp(3) ## provides the cube of e

log(1.4) ## provides the natural logarithm of the number 1.4

log10(1.4) ## provides the log to the base of 10

sqrt(16) ## provides the square root of 16

Result:

```
> pi
[1] 3.141593
> exp(3) ## provides the cube of e
[1] 20.08554
> log(1.4) ## provides the natural logarithm of the number 1.4
[1] 0.3364722
> log10(1.4) ## provides the log to the base of 10
[1] 0.146128
> sqrt(16) ## provides the square root of 16
[1] 4
> |
```

#### 6. Variables (how to assign, reserved symbols, creating meaningful variables)

Variables are the symbols that store assigned values. We can store a computation under a new variable or change the existing value of an old variable. **Variable names in R are case sensitive** (upper or lower case). It is a good practice to assign meaningful variable names that helps to refer to easily for complex calculations.

To assign a value: variable\_name <- value

**Example:**

```
currentYear <- 2016  
birthYear <- 1990  
age <- currentYear - birthYear
```

**7. Exercise:** You have already assigned values to variables x, y and z in your R script. Try assigning new variables and doing calculations on them.

```
i <- 10  
j <- 20  
i*j - (i + j)
```

**Now try re-assigning different values and trying out some basic arithmetic calculations and built-ins using variables.**

Here are some things you can try out:

Function	Description
<code>abs(x)</code>	absolute value
<code>sqrt(x)</code>	square root
<code>ceiling(x)</code>	<code>ceiling(3.475)</code> is 4
<code>floor(x)</code>	<code>floor(3.475)</code> is 3
<code>trunc(x)</code>	<code>trunc(5.99)</code> is 5
<code>round(x, digits=n)</code>	<code>round(3.475, digits=2)</code> is 3.48
<code>signif(x, digits=n)</code>	<code>signif(3.475, digits=2)</code> is 3.5
<code>cos(x)</code> , <code>sin(x)</code> , <code>tan(x)</code>	also <code>acos(x)</code> , <code>cosh(x)</code> , <code>acosh(x)</code> , etc.
<code>log(x)</code>	natural logarithm
<code>log10(x)</code>	common logarithm
<code>exp(x)</code>	$e^x$

**8. Reserved Symbols**

In all programming languages certain symbols are reserved for specific purposes. The reserved symbols in R are -

`c` `q` `t` `C` `D` `F` `I` `T`

**Type `q()` in the Console and hit Enter to quit R (but don't actually quit the program)**

**9. Functions**

A function is a sub-program that performs a specific task. The built-in math operations we have seen such as `mean()`, `sqrt()` and `cos()` are functions. Functions help to avoid repetition and easy execution in future.

**Run the following code to understand how functions work. Copy and paste it into your script, then run it!**

```
firstFunction <- function(x){x*x}  
firstFunction(3)
```

Result:

```
> firstFunction(3)  
[1] 9
```

This function named firstFunction is supposed to return square of any integer.

**Now write a function that returns an integer multiplied by 10. Test it out!**

## 10. Vectors

Vectors have different meanings in different contexts. In math and physics, a vector is an element with both value and direction. But in R, vector is a sequence of data elements of the same basic type. It can be defined by concatenating the members in a set c().

Example: `x <- c(7, 1, 5, 3)`.

Once we have a vector of numbers we can apply certain built-in functions to them to get useful summaries. For example:

```
sum(x)    ## sums the values in the vector  
length(x) ## produces the number of values in the vector, ie its length  
mean(x)   ## the average (mean)  
var(x)    ## the sample variance of the values in the vector (has n-1 in denominator)  
sd(x)     ## the sample standard deviation of the values in the vector (square root of the  
           sample variance)  
max(x)    ## the largest value in the vector  
min(x)    ## the smallest number in the vector  
median(x) ## the sample median  
y <- sort(x) ## the values arranged in ascending order
```

**Exercise: Create a vector and try out some of the functions.**

```
x <- c(7, 1, 5, 3)  
sum(x)  
length(x)  
mean(x)  
sort(x)
```

## 11. Data Frame

A data frame can be created by defining different variables for each column as vectors and then joining them together.

**Create three vectors listing different fruits with their names, colors and size:**

```
name <- c("apple", "banana", "peach", "watermelon", "grape")
color <- c("red", "yellow", "peach", "green", "red")
size_cm <- c(10, 15, 8, 40, 2)
```

**Add these three columns together to create the data frame names fruits.data**

```
fruits.data <- data.frame(name, color, size_cm)
```

**To see the values of the data frame, type fruits.data in the Console.**

```
fruits.data
```

	name	color	size_cm
1	apple	red	10
2	banana	yellow	15
3	peach	peach	8
4	watermelon	green	40
5	grape	red	2

**Add new variable to the data frame**

```
fruits.data["quantity"] <- c(2, 6, 5, 1, 30)
```

**Select a subset of fruits.data**

```
my_fruits <- subset(fruits.data, quantity < 5, select = c(name, quantity))
```

## Extra time activities

### Working with variables

- Select a variable by adding \$ sign after the data frame's name.
- Exclude variables v1, v2, v3

```
myvars <- names(mydata) %in% c("v1", "v2", "v3")
newdata <- mydata[!myvars]
```
- Exclude 3rd and 5th variable

```
newdata <- mydata[c(-3, -5)]
```
- To delete a variable you can set it to NULL. Example: `newdata$X <- NULL`.
- To delete multiple variables - `newdata[1:2] <- list(NULL)`



## Activity 2

### Sample data sets in R -

- The R Datasets Package - contains the list of all datasets available in R by default and details of the content  
<https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html>
- List of all datasets in R that can also be downloaded in CSV/Doc format -  
<https://vincentarelbundock.github.io/Rdatasets/datasets.html>

### Loading data into R:

Few things to keep in mind:

- Avoid names, values, fields with blank spaces, otherwise each word will be interpreted as a separate variable, resulting into errors in calculating the number of elements per line in the dataset.
- Shorter names are preferred over long names.
- Try to avoid using names that contain symbols such as ?, \$, %, ^, &, \*, (, ), -, #, ?, >, <, /, |, \, [ , ] , { , and }
- Delete any comments in your file to avoid extra column or NA
- Make sure to indicate any missing value with NA.
- If the top row in your dataset has variable names then header = TRUE (default), otherwise set as FALSE.
  - If header = FALSE, R will set column names as V1, V2 ... To define own column names specify them under col.names. Example: col.names = c("X", "Y", "Z", "A", "B")

To load data:

- For text file, `data_frame_name <- read.table("path")`
- For CSV file, `data_frame_name <- read.csv("path")`

**\* You don't have to type the full directory path if you have set your working directory.**

For further information refer to [This R Data Import Tutorial Is Everything You Need](#) from the DataCamp.

**Example of how to load in a dataset (we won't do this, but try it if you'd like!) -- first make sure you have put the file from (go.ncsu.edu/rworkshop) into your working directory:**

```
aq <- read.csv("airquality.csv")
```

**Remove the extra column X added**

```
aq$X <- NULL
```

**Try these examples with the airquality dataset (it's already loaded into RStudio as a sample data set)**

```
summary(airquality),  
mean(airquality$Temp),  
sapply(airquality, mean, na.rm = TRUE)
```

**Install Hmisc package and run this code. COPY AND PASTE THIS INTO YOUR SCRIPT:**

```
install.packages("Hmisc",  
lib="/Library/Frameworks/R.framework/Versions/3.2/Resources/library")  
library(Hmisc)  
describe(airquality)  
warmDays <- subset(airquality, Temp > 80, select = c(Day, Month))
```

**Install dplyr package and run this code. COPY AND PASTE THIS INTO YOUR SCRIPT:**

```
install.packages("dplyr", lib="/Library/Frameworks/R.framework/Versions/3.2/Resources/library")  
library(dplyr)  
arrange(warmDays, Month)
```

## Creating Plots (Basic plotting, ggplot2)

**This uses the mtcars dataset. To see the data, type:**

```
mtcars
```

**Result:**

```
> mtcars
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1

...

**You can load the dataset into R or directly work on the one provided by R**

Plot by default refers to Scatter plot. The basic function is plot(x, y), where x and y are numeric vectors.

**Example: Find out the relationship between car weight and mile per gallon (mpg) use from mtcars dataset.**

```
plot(mtcars$wt, mtcars$mpg, main="Scatterplot Example", xlab="Car Weight ", ylab="Miles Per  
Gallon ", pch=19)
```

```
# Add fit lines
abline(lm(mtcars$mpg~mtcars$wt), col="red")      # regression line (y~x)
lines(lowess(mtcars$mpg~mtcars$wt), col="green") # lowess line (x, y)
```

#The same graph can be created using the ggplot function too.

```
install.packages("ggplot2")
library("ggplot2")
qplot(mtcars$wt, mtcars$mpg)
```

#If the two vectors are already in the same data frame, you can use the following syntax:

```
qplot(wt, mpg, data=mtcars)
#This is equivalent to:
ggplot(mtcars, aes(x=wt, y=mpg)) + geom_point()
```

The path type is "point" by default in Scatter plot. To change it to line add - type = "l", or mix of point and line - type = "o"

Example:

```
plot(pressure$temperature, pressure$pressure, type="l")
```

### Extra time activities

#### Generate a bar plot

```
barplot(mtcars$wt)
# Generate a table of counts
barplot(table(mtcars$cyl))
# Using ggplot2
qplot(mtcars$cyl)
# Treat cyl as discrete
qplot(factor(mtcars$cyl))
```

#### Histogram - view the distribution of one dimensional data

```
hist(mtcars$mpg)
# Specify approximate number of bins with breaks
hist(mtcars$mpg, breaks = 10)
# Use ggplot2
qplot(mtcars$mpg)
qplot(mpg, data = mtcars, binwidth = 4)
```

---

### Want to Go Deeper with R?

To learn more about visualization with R refer to: [Chang, W. \(2012\). R graphics cookbook. "O'Reilly Media, Inc."](#)