R Scripting

Programming in R

Advantage of Scripting

An R script is simply a text file containing (almost) the same commands that you would enter on the command line of R

- Reproducibility
- Easy to alter analysis
- Open source scripts can be made available to collaborators, reviewers and colleagues

R Scripts Elements

- Set of assume a working directory

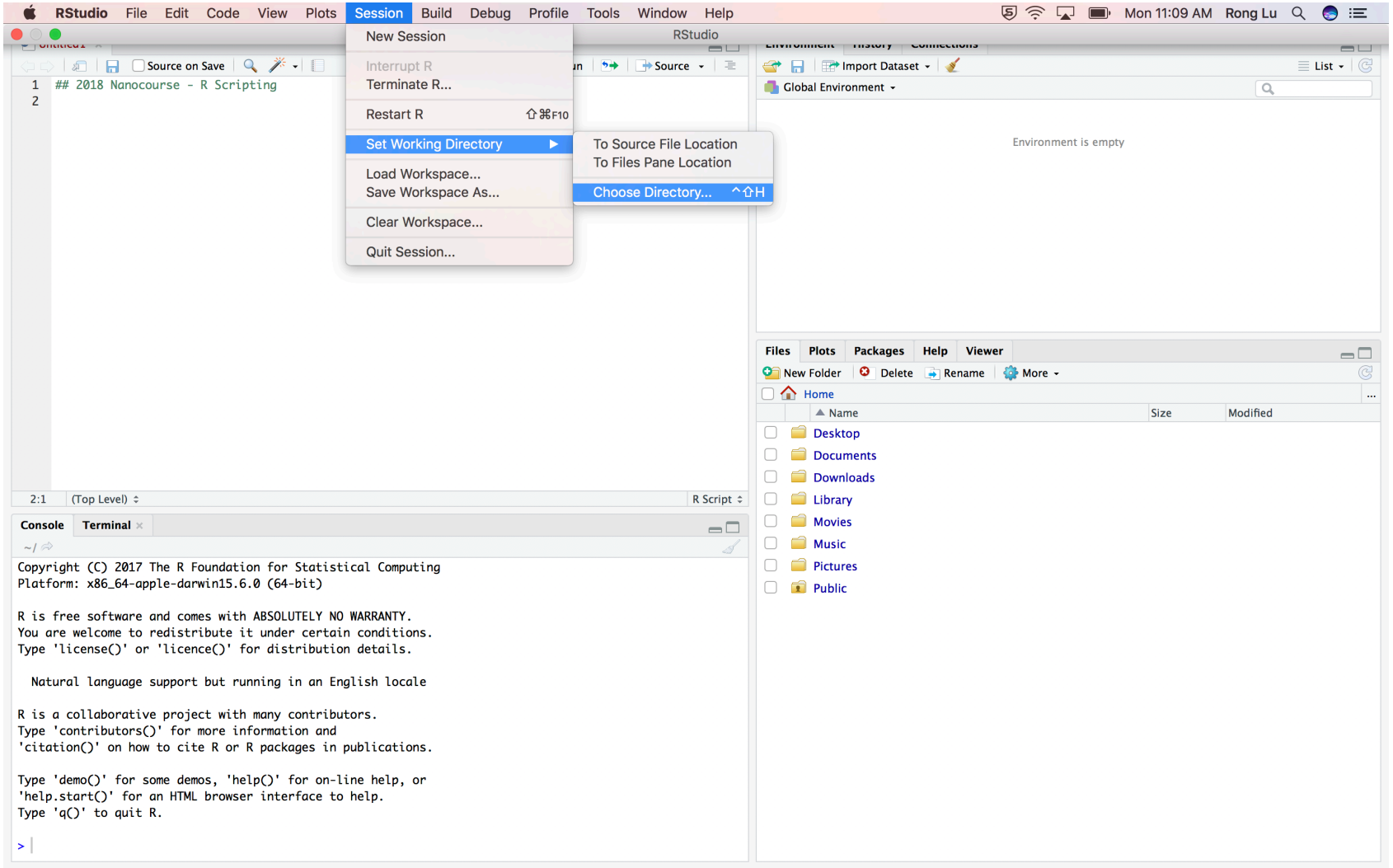
  Where are the input and output files being read and written?

- Input data

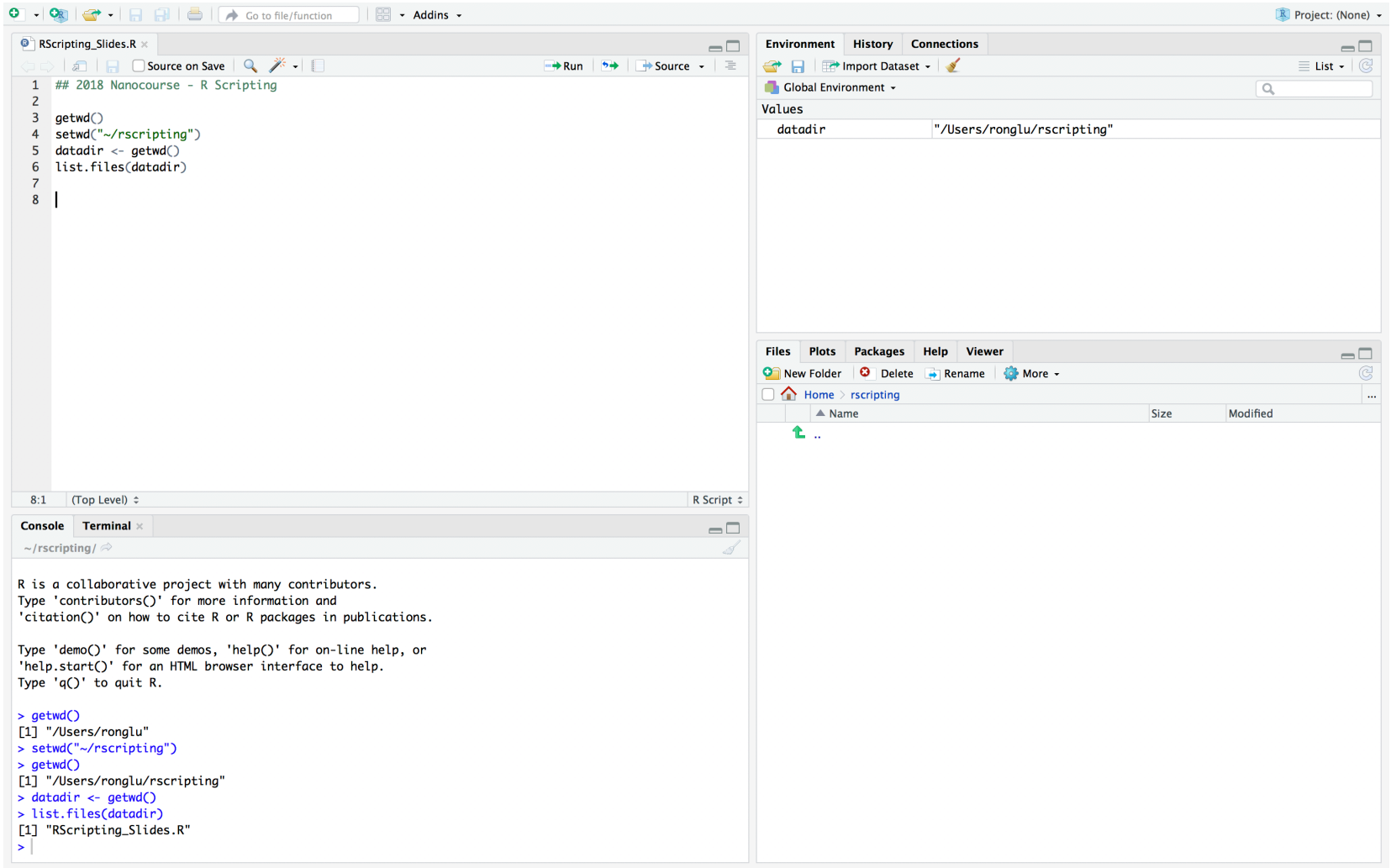
- Processes data, run statistical analysis, or generate plots

- Output figures and tables

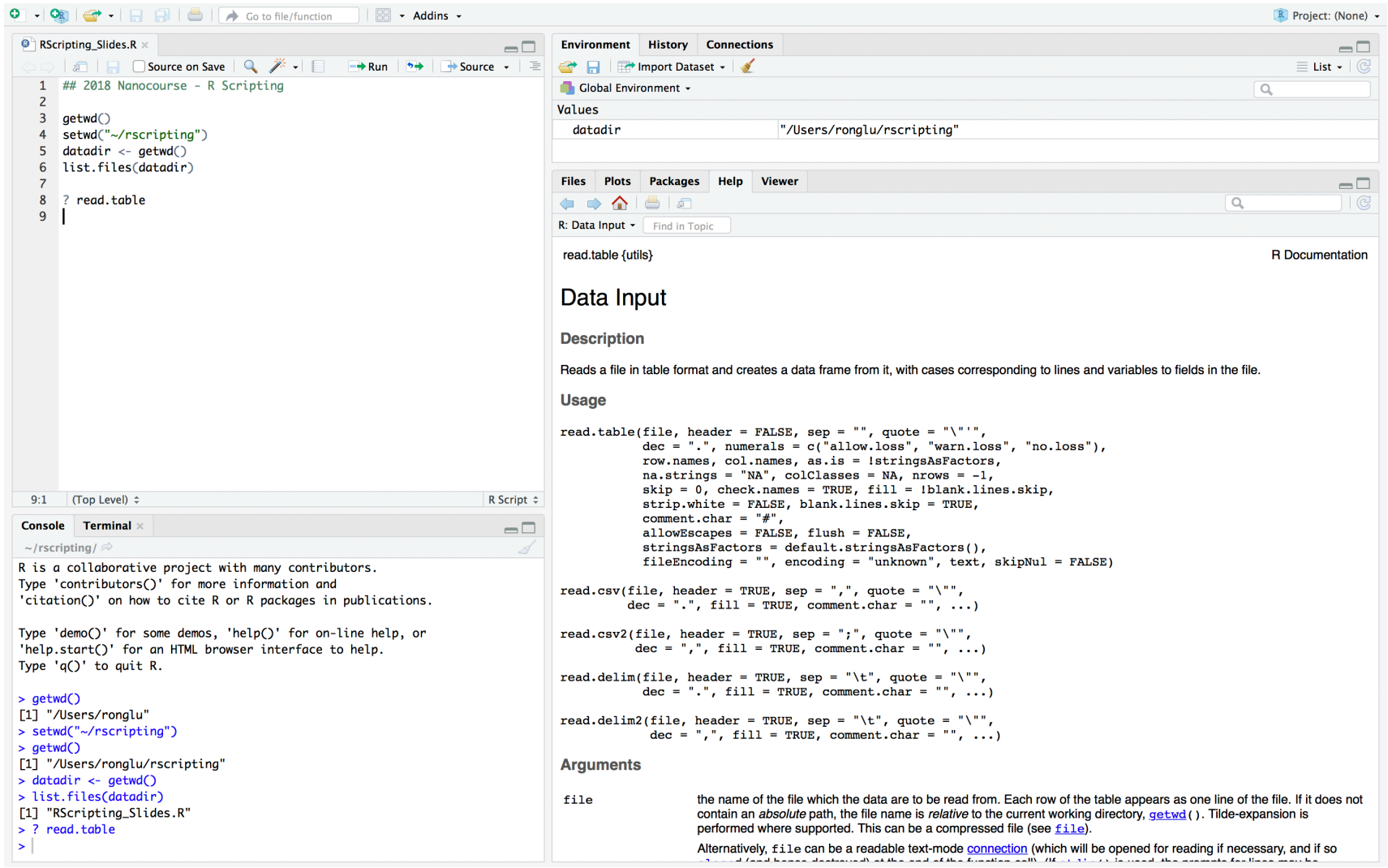
Selecting Working Directory



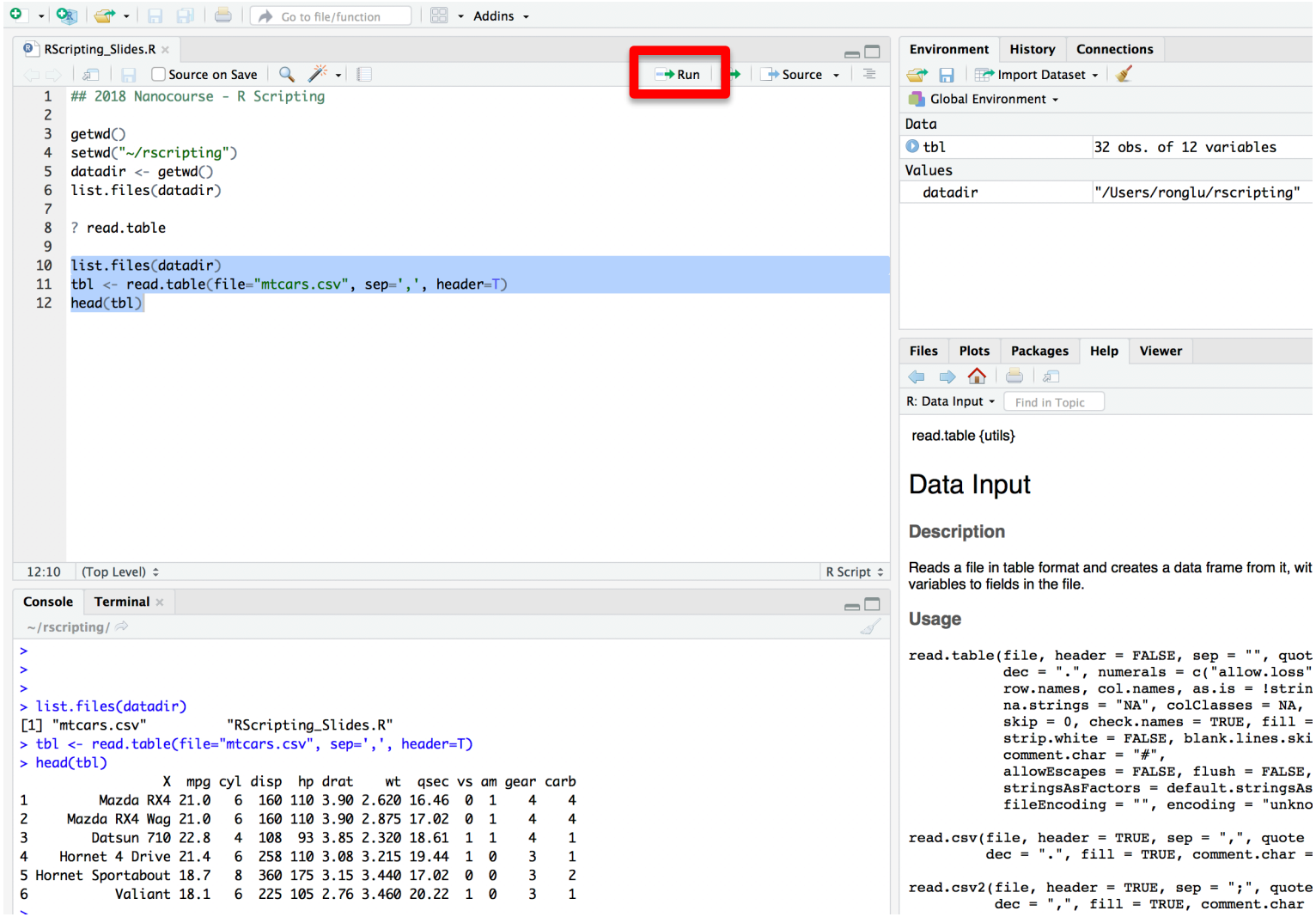
Selecting Working Directory



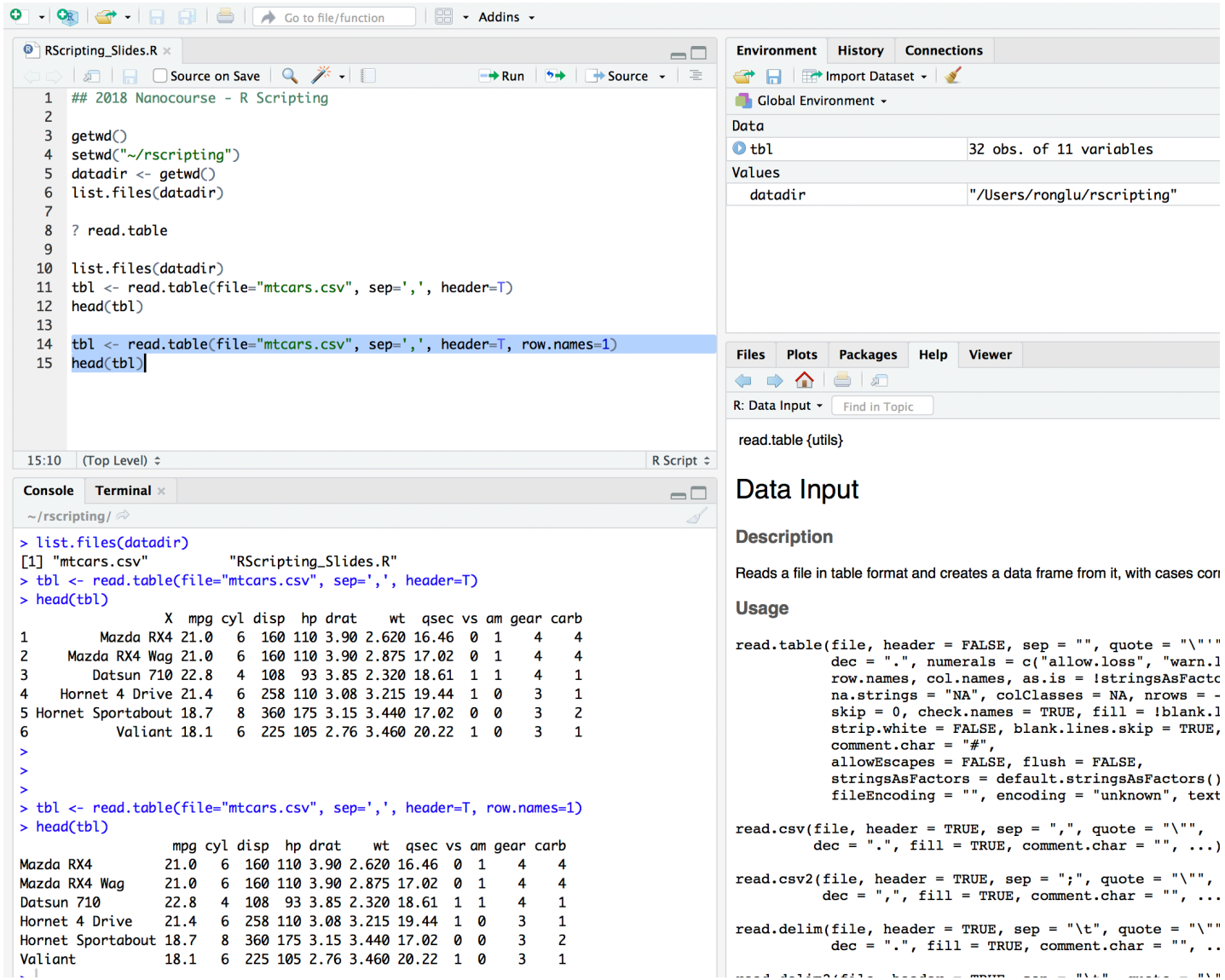
Data Input


RScripting_Slides.R
2018 Nanocourse - R Scripting
getwd()
setwd("~/rscripting")
datadir <- getwd()
list.files(datadir)
? read.table
Console
Terminal
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.
> getwd()
[1] "/Users/ronglu"
> setwd("~/rscripting")
> getwd()
[1] "/Users/ronglu/rscripting"
> datadir <- getwd()
> list.files(datadir)
[1] "RScripting_Slides.R"
> ? read.table
Environment
History
Connections
Import Dataset
Global Environment
Values
datadir
"/Users/ronglu/rscripting"
Files
Plots
Packages
Help
Viewer
R: Data Input
Find in Topic
read.table {utils}
R Documentation
Data Input
Description
Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.
Usage
read.table(file, header = FALSE, sep = "", quote = "\"'",
dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
row.names, col.names, as.is = !stringsAsFactors,
na.strings = "NA", colClasses = NA, nrows = -1,
skip = 0, check.names = TRUE, fill = !blank.lines.skip,
strip.white = FALSE, blank.lines.skip = TRUE,
comment.char = "#",
allowEscapes = FALSE, flush = FALSE,
stringsAsFactors = default.stringsAsFactors(),
fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
read.csv(file, header = TRUE, sep = ",", quote = "\"",
dec = ".", fill = TRUE, comment.char = "", ...)
read.csv2(file, header = TRUE, sep = ";", quote = "\"",
dec = ",", fill = TRUE, comment.char = "", ...)
read.delim(file, header = TRUE, sep = "\t", quote = "\"",
dec = ".", fill = TRUE, comment.char = "", ...)
read.delim2(file, header = TRUE, sep = "\t", quote = "\"",
dec = ",", fill = TRUE, comment.char = "", ...)
Arguments
file
the name of the file which the data are to be read from. Each row of the table appears as one line of the file. If it does not contain an absolute path, the file name is relative to the current working directory, getwd(). Tilde-expansion is performed where supported. This can be a compressed file (see file).
Alternatively, file can be a readable text-mode connection (which will be opened for reading if necessary, and if so

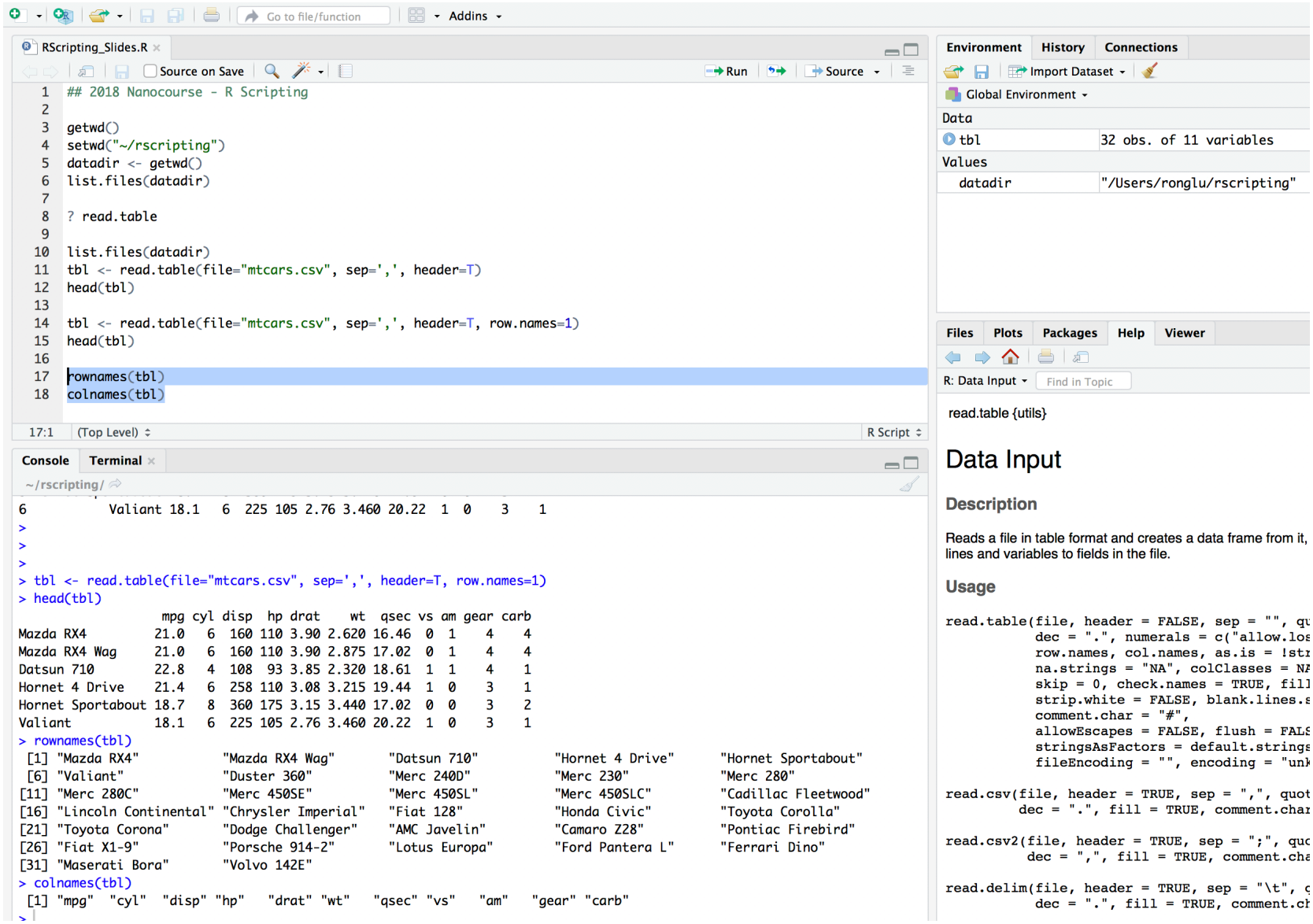
Data Input


Go to file/function
Addins
RScripting_Slides.R
Source on Save
Run
Source
1 ## 2018 Nanocourse - R Scripting
2
3 getwd()
4 setwd("~/rscripting")
5 datadir <- getwd()
6 list.files(datadir)
7
8 ? read.table
9
10 list.files(datadir)
11 tbl <- read.table(file="mtcars.csv", sep=',', header=T)
12 head(tbl)
12:10
(Top Level)
R Script
Console
Terminal
~/rscripting/
> list.files(datadir)
[1] "mtcars.csv" "RScripting_Slides.R"
> tbl <- read.table(file="mtcars.csv", sep=',', header=T)
> head(tbl)
X mpg cyl disp hp drat wt qsec vs am gear carb
1 Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
2 Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
3 Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1
4 Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
5 Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
6 Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1
Environment
History
Connections
Import Dataset
Global Environment
Data
tbl
32 obs. of 12 variables
Values
datadir
"/Users/ronglu/rscripting"
Files
Plots
Packages
Help
Viewer
R: Data Input
Find in Topic
read.table {utils}
Data Input
Description
Reads a file in table format and creates a data frame from it, wit
variables to fields in the file.
Usage
read.table(file, header = FALSE, sep = "", quot
dec = ".", numerals = c("allow.loss"
row.names, col.names, as.is = !strin
na.strings = "NA", colClasses = NA,
skip = 0, check.names = TRUE, fill =
strip.white = FALSE, blank.lines.ski
comment.char = "#",
allowEscapes = FALSE, flush = FALSE,
stringsAsFactors = default.stringsAs
fileEncoding = "", encoding = "unkno
read.csv(file, header = TRUE, sep = ",", quote
dec = ".", fill = TRUE, comment.char =
read.csv2(file, header = TRUE, sep = ";", quote
dec = ",", fill = TRUE, comment.char

Row & Column Names



Row & Column Names



Creating Data Table

Console Terminal ×

~/rscripting/

```
>
> ? rnorm
> ? rep
> mpg = c(23, 34.9, 28, 32.5)
> cyl = rep(1:2, 2)
> drat = rnorm(4, mean=3.5)
> df = data.frame(mpg, cyl, drat)
> head(df)
   mpg cyl     drat
1 23.0   1 3.694824
2 34.9   2 3.838816
3 28.0   1 2.414743
4 32.5   2 3.564814
>
```

Programming Functions

- Conditional Statements

  if/else

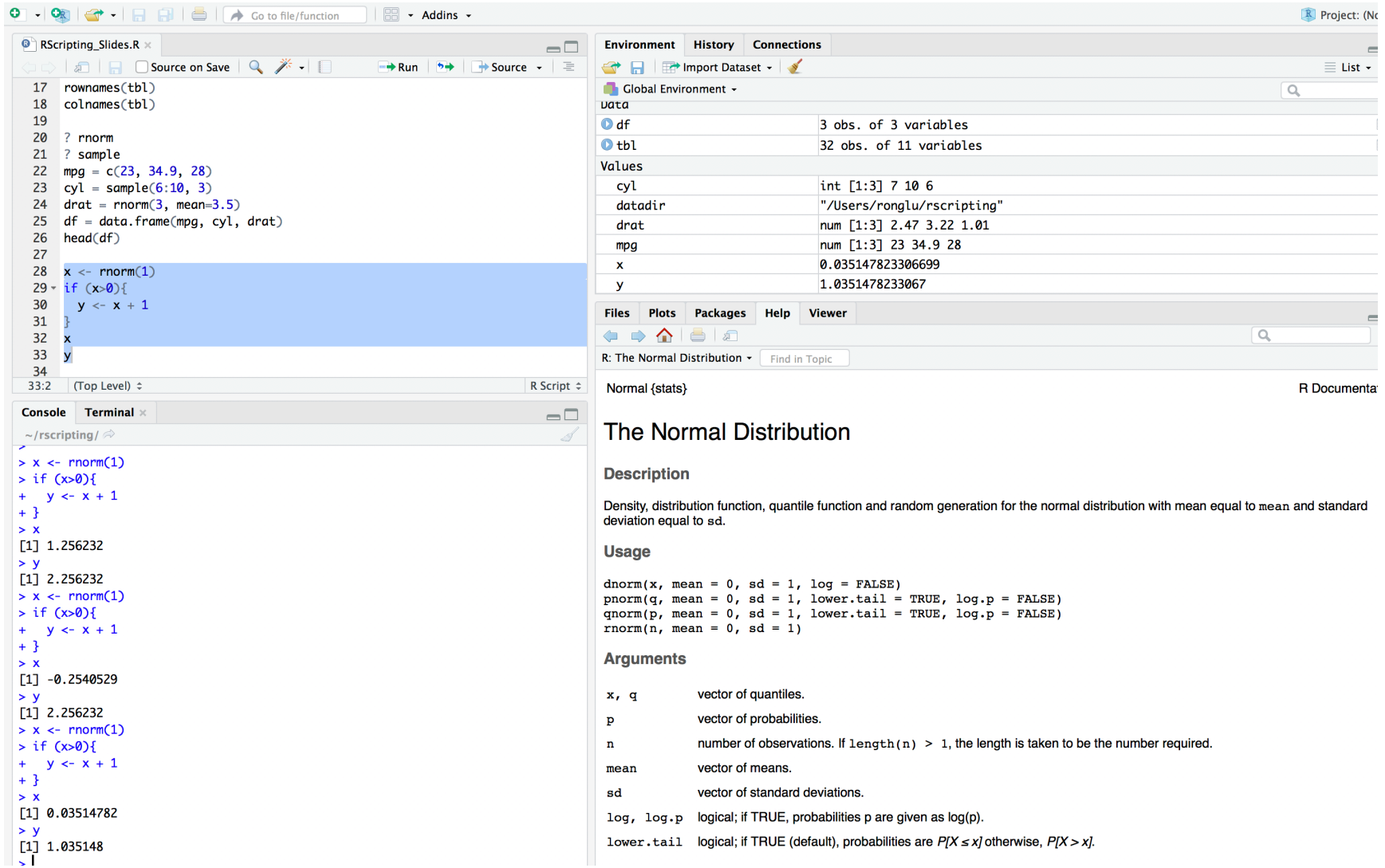
- Loops

  for, while, repeat, apply

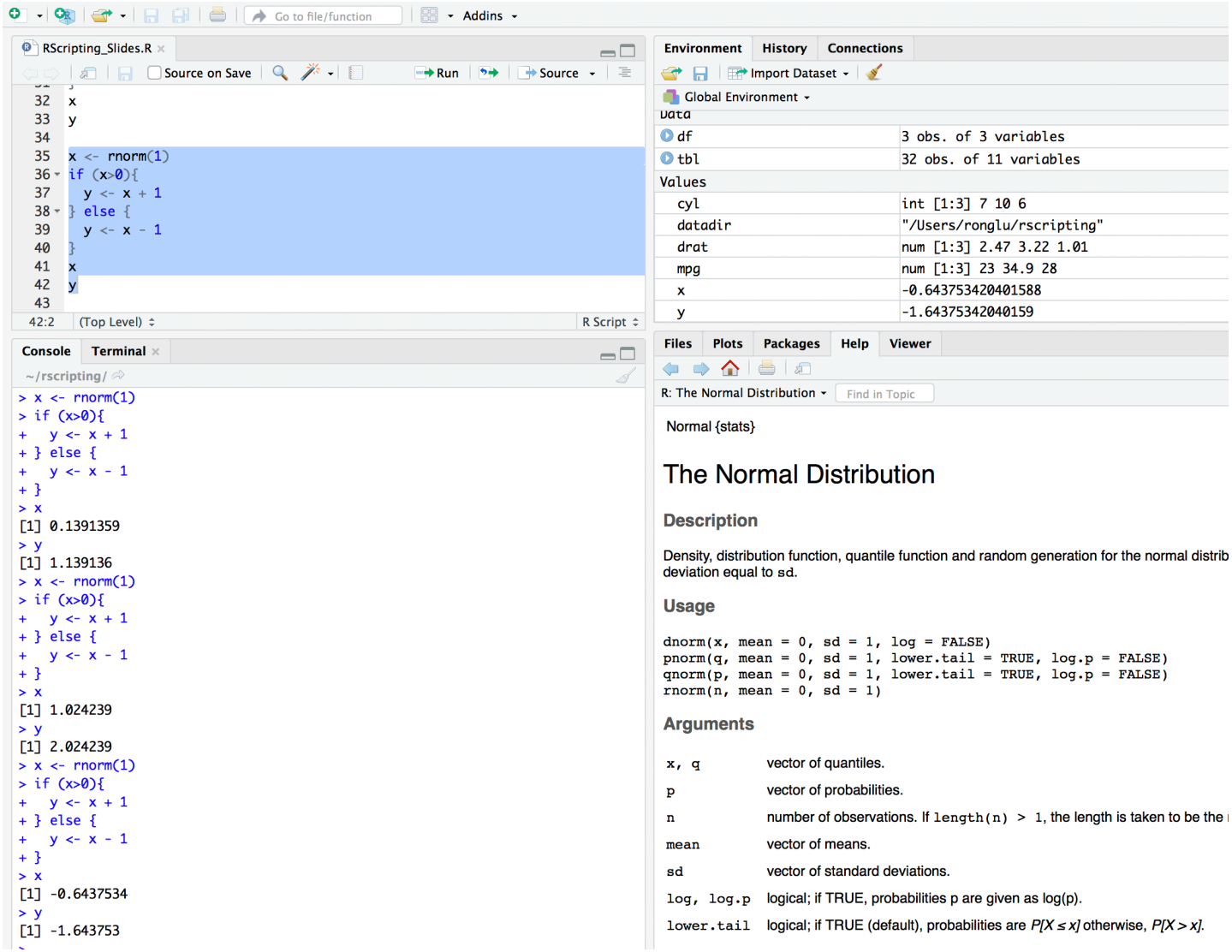
- Functions

  user defined calculations
  calling on 3rd party & built-in functions

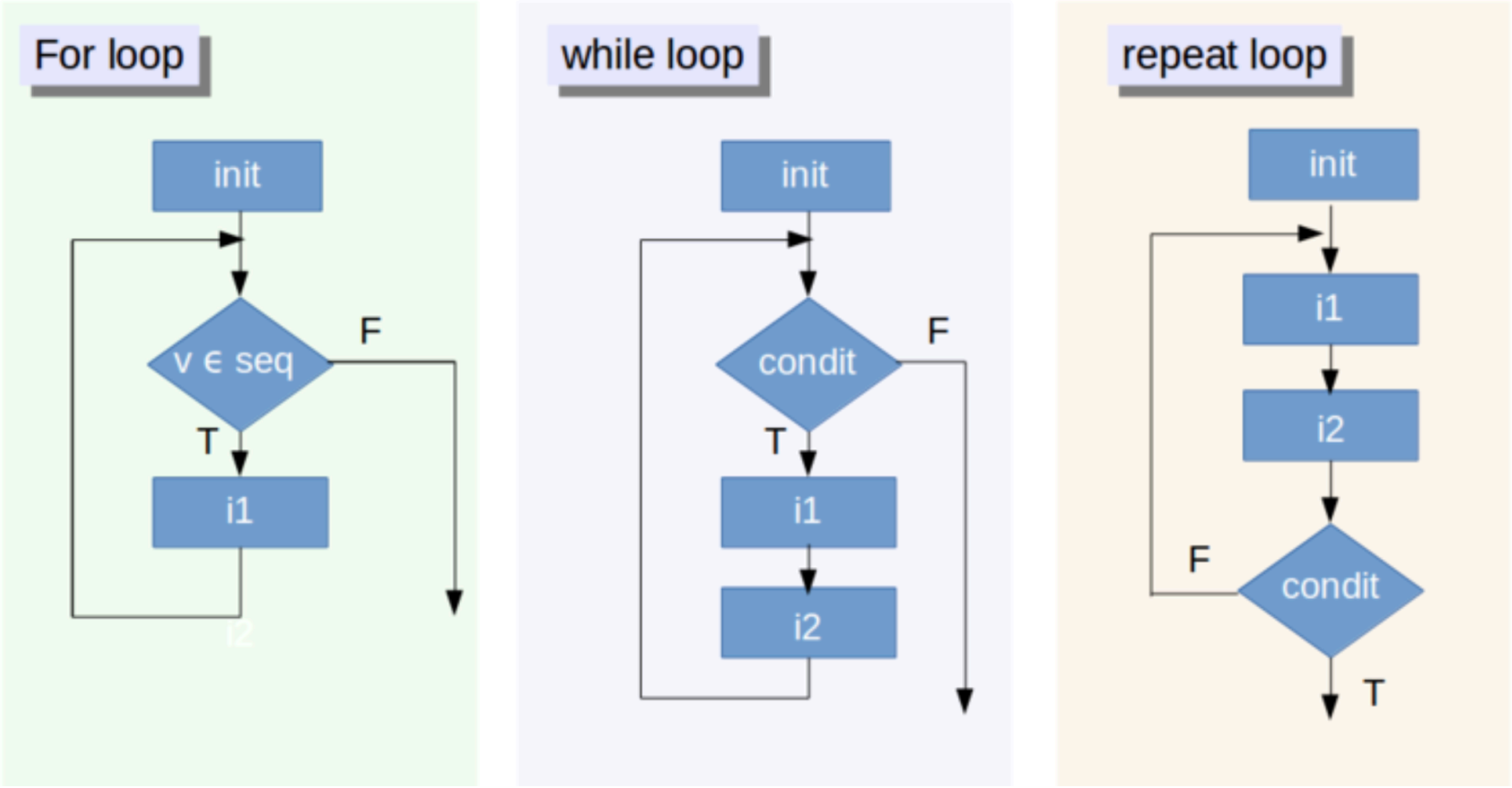
If Statement



If/else Statement



Loop Structure in R



For Loop Example

```
39    y <- x - 1
40  }
41  x
42  y
43
44  names(tbl)
45  Ys <- names(tbl)[3:6]
46  x <- tbl$mpg
47  par(mfrow=c(2,2))
48  for (i in 1:4) {
49    y <- tbl[,Ys[i]]
50    plot(x,y,xlab="mpg", ylab=Ys[i])
51  }
```

51:2 (Top Level) R Script

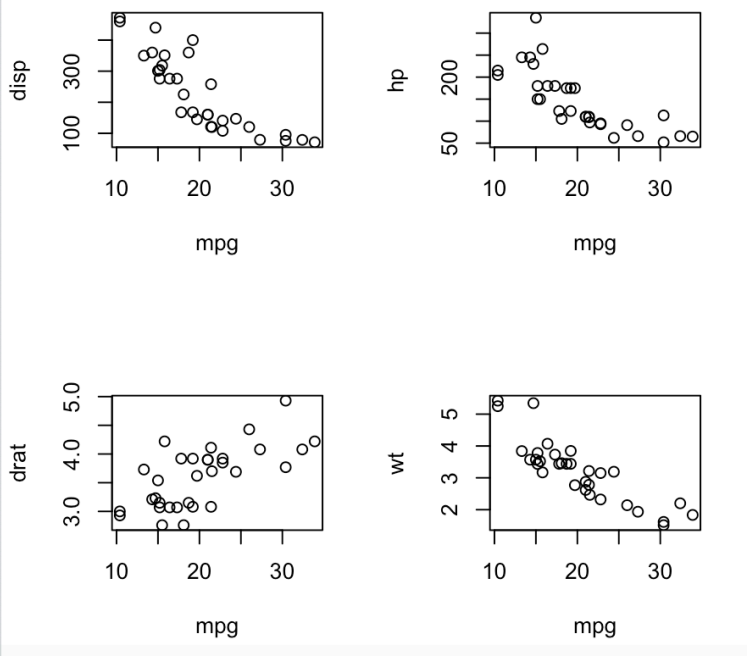
Console Terminal

~/rscripting/

```
>
>
> names(tbl)
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
> Ys <- names(tbl)[3:6]
> x <- tbl$mpg
> par(mfrow=c(2,2))
> for (i in 1:4) {
+   y <- tbl[,Ys[i]]
+   plot(x,y,xlab="mpg", ylab=Ys[i])
+ }
>
```

Files Plots Packages Help Viewer

Zoom Export



Nested For Loop Example

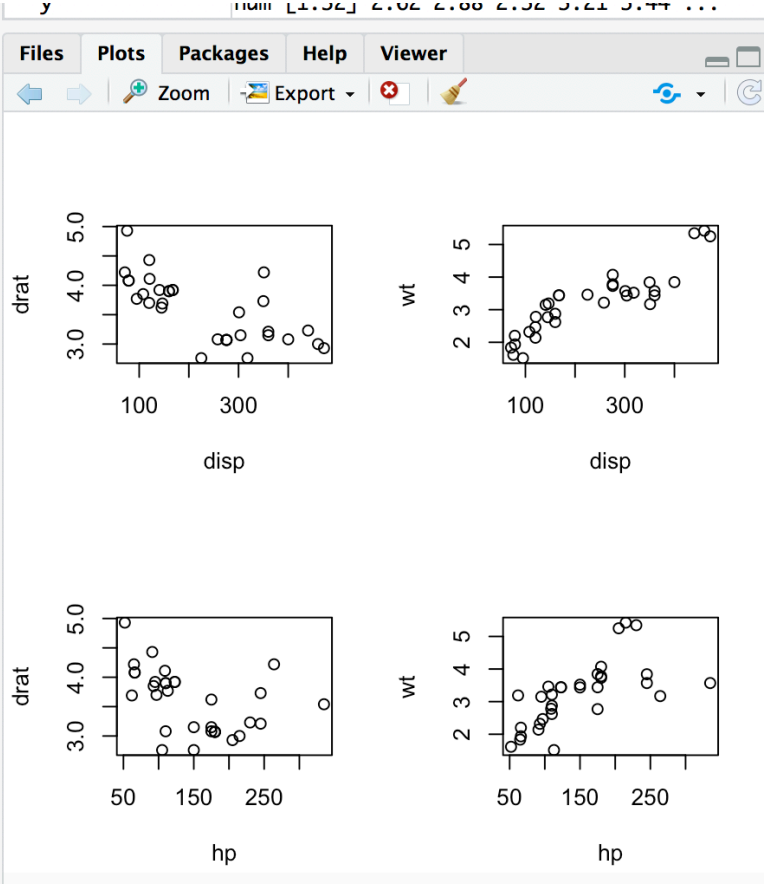
```
53  names(tbl)
54  Xs <- names(tbl)[3:4]
55  Ys <- names(tbl)[5:6]
56  par(mfrow=c(2,2))
57  for (i in 1:2){
58    x <- tbl[,Xs[i]]
59    for (j in 1:2) {
60      y <- tbl[,Ys[j]]
61      plot(x, y, xlab=Xs[i], ylab=Ys[j])
62    }
63  }
```

63:2 (Top Level) R Script

Console Terminal

~/rscripting/

```
> names(tbl)
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"
[10] "gear" "carb"
> Xs <- names(tbl)[3:4]
> Ys <- names(tbl)[5:6]
> par(mfrow=c(2,2))
> for (i in 1:2){
+   x <- tbl[,Xs[i]]
+   for (j in 1:2) {
+     y <- tbl[,Ys[j]]
+     plot(x, y, xlab=Xs[i], ylab=Ys[j])
+   }
+ }
>
```



While Loop Example

```
> i <- 0
> square <- 0
> while (square < 88) {
+   i <- i + 1
+   square <- i^2
+   cat("i=", i, "; square=", square,"; square<88 is", (square < 88), '\n')
+ }
i= 1 ; square= 1 ; square<88 is TRUE
i= 2 ; square= 4 ; square<88 is TRUE
i= 3 ; square= 9 ; square<88 is TRUE
i= 4 ; square= 16 ; square<88 is TRUE
i= 5 ; square= 25 ; square<88 is TRUE
i= 6 ; square= 36 ; square<88 is TRUE
i= 7 ; square= 49 ; square<88 is TRUE
i= 8 ; square= 64 ; square<88 is TRUE
i= 9 ; square= 81 ; square<88 is TRUE
i= 10 ; square= 100 ; square<88 is FALSE
> i - 1
[1] 9
```

Repeat

```
i <- 0
square <- 0
repeat {
 i <- i+1
 square <- i*i
 if (square > 88) {
  break
 }
 }
 i - 1
```


Without a “break” conditional repeats are infinite loops

Repeat Example

```
> i <- 0
> square <- 0
> repeat {
+   i <- i+1
+   square <- i*i
+   cat("i=", i, "; square=", square,"; square<88 is", (square < 88), '\n')
+   if (square > 88) {
+     break }
+ }
i= 1 ; square= 1 ; square<88 is TRUE
i= 2 ; square= 4 ; square<88 is TRUE
i= 3 ; square= 9 ; square<88 is TRUE
i= 4 ; square= 16 ; square<88 is TRUE
i= 5 ; square= 25 ; square<88 is TRUE
i= 6 ; square= 36 ; square<88 is TRUE
i= 7 ; square= 49 ; square<88 is TRUE
i= 8 ; square= 64 ; square<88 is TRUE
i= 9 ; square= 81 ; square<88 is TRUE
i= 10 ; square= 100 ; square<88 is FALSE
> i-1
[1] 9
```

Controlling Loops

- break

  in a conditional statement to stop the loop

- next

  in a conditional statement to skip the analysis for certain rounds of the loop

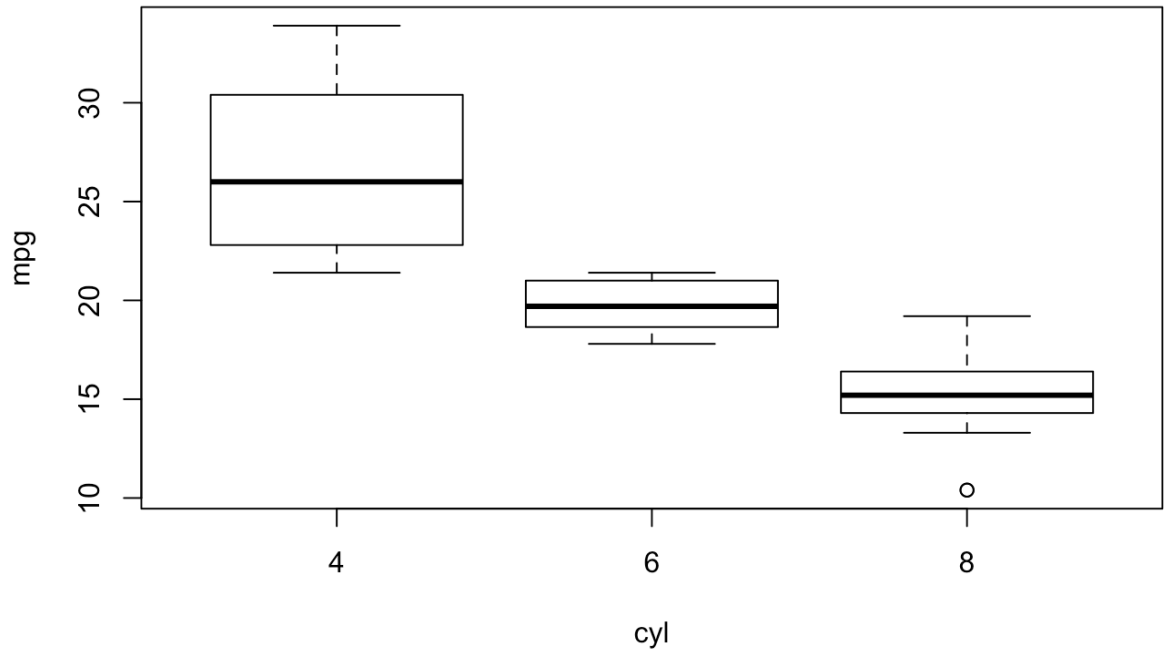
Next Example

```
i <- 0
j <- 0
repeat {
  if (j > 14) { break }
  i <- i+1; j <- j + 1;
  if(i == 4) {next}
  j <- j + 1;
  cat("i=", i, "; j=", j, '\n');
}
i
j
```

```
> i <- 0
> j <- 0
> repeat {
+   if (j > 14) { break }
+   i <- i+1; j <- j + 1;
+   if(i == 4) {next}
+   j <- j + 1;
+   cat("i=", i, "; j=", j, '\n');
+ }
i= 1 ; j= 2
i= 2 ; j= 4
i= 3 ; j= 6
i= 5 ; j= 9
i= 6 ; j= 11
i= 7 ; j= 13
i= 8 ; j= 15
> i
[1] 8
> j
[1] 15
```

Built-in function: aggregate

```
> aggregate(mpg ~ cyl, tbl, summary)
  cyl mpg.Min. mpg.1st Qu. mpg.Median mpg.Mean mpg.3rd Qu. mpg.Max.
1   4 21.40000    22.80000   26.00000 26.66364    30.40000 33.90000
2   6 17.80000    18.65000   19.70000 19.74286    21.00000 21.40000
3   8 10.40000    14.40000   15.20000 15.10000    16.25000 19.20000
> par(mfrow=c(1,1))
> boxplot(tbl$mpg ~ tbl$cyl, xlab="cyl", ylab="mpg")
```



Built-in function: aggregate

```
> table(tbl$cyl)

 4  6  8
11  7 14
> for (i in c(4,6,8)) {
+   cat("mpg mean (when cyl=", i, "): ", mean(tbl$mpg[tbl$cyl==i]), '\n')
+ }
mpg mean (when cyl= 4 ):  26.66364
mpg mean (when cyl= 6 ):  19.74286
mpg mean (when cyl= 8 ):  15.1
```

Other built-In Loop Functions

- `summary(tbl)`
  - gives the min, quantiles, mean, max for a data.frame or matrix of numbers
- `aggregate(mpg ~ cyl, tbl, mean)`
  - will perform a functions on a vector in a matrix or data.frame using a variable to “bin” the data
- `by(tbl, tbl$cyl, colMeans)`
  - will perform a function on all vectors in a matrix or data.frame using a variable to “bin” the data
- `replicate(12,rnorm(10))`
  - will create a matrix using a function that is “repeated”

by: aggregate on a matrix

```
> by(tbl, tbl$cyl, colMeans)
tbl$cyl: 4
       mpg        cyl       disp         hp       drat         wt
26.6636364  4.0000000 105.1363636 82.6363636  4.0709091  2.2857273
      qsec         vs         am       gear       carb
19.1372727  0.9090909  0.7272727  4.0909091  1.5454545
------------------------------------------------------------
tbl$cyl: 6
       mpg        cyl       disp         hp       drat         wt
19.7428571  6.0000000 183.3142857 122.2857143  3.5857143  3.1171429
      qsec         vs         am       gear       carb
17.9771429  0.5714286  0.4285714  3.8571429  3.4285714
------------------------------------------------------------
tbl$cyl: 8
       mpg        cyl       disp         hp       drat         wt
15.1000000  8.0000000 353.1000000 209.2142857  3.2292857  3.9992143
      qsec         vs         am       gear       carb
16.7721429  0.0000000  0.1428571  3.2857143  3.5000000
```

replicate Example

```
> replicate(12, sample(1:50, 4))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
[1,]   21   48   34   47   45    4    5   30    2    32    33    10
[2,]   27   38    4   21   17   15   28   25   25    36    44     2
[3,]   40   43    9   43   15   35   21   41   50     7    48    45
[4,]   34    4   38    4   40   38   46   24    4    17    47    46
```

User Defined Functions

There are lots of built-in functions in R. But sometimes, you need some code that isn't.

Functions are just a sets of instructions that we want to use repeatedly or that, because of their complexity, are better self-contained in a sub program and called when needed.

Basic Function Elements

```
function.name <- function(arguments)
{
  computations on the arguments
  some other code
}
```

User Defined Function Example

```
> my.fun.1 <- function(x, y, n) {
+   return(x+y^n)
+ }
> my.fun.1(1,2,1)
[1] 3
> my.fun.1(2,3,2)
[1] 11
```

Setting default argument value:

```
> my.fun.1 <- function(x, y, n=2) {
+   return(x+y^n)
+ }
> my.fun.1(1,2)
[1] 5
> my.fun.1(2,3,1)
[1] 5
```

Calling Function 1 in Function 2

```
> my.fun.2 <- function(x, y) {
+   return(my.fun.1(x,y,1) + my.fun.1(x,y))
+ }
> my.fun.2(1,2)
[1] 8
> my.fun.2(2,3)
[1] 16
> (1+2^1) + (1+2^2)
[1] 8
> (2+3^1) + (2+3^2)
[1] 16
```

Return a list in function

```
> my.fun <- function(x, y) {
+   z <- my.fun.1(x,y,1) + my.fun.1(x,y)
+   return(list(value=z, vector=rep(z,z)))
+ }
> my.fun(1,2)
$value
[1] 8

$vector
[1] 8 8 8 8 8 8 8 8

> my.fun(2,3)
$value
[1] 16

$vector
 [1] 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
```

Function Best Practices

- Keep your functions short.
- If things start to get very long, you can probably split up your function into more manageable chunks that call other functions. This makes your code cleaner and easily testable.
- Functions makes your code easy to update. You only have to change one function and every other function that uses that function will also be automatically updated.
- Put in comments on what are the inputs to the function, what the function does, and what is the output.
- Check for errors along the way.
- Try out your function with simple examples to make sure it's working properly

apply Functions

- The apply family can be used to perform functions to manipulate slices of data from matrices, arrays, lists and data frames in a repetitive way.
- apply — operates on array or matrix
- lapply and sapply— traversing over a set of data like a list or vector, and calling the specified function for each item. sapply return a vector and lapply returns a list
- mapply —'multivariate' apply.
- tapply — applies a function to each cell of an array

apply Examples

- `(N)apply(X, MARGIN, FUN, …)`
- Apply
  - `apply(tbl,2,sum) #Sum of each column in tbl`
  - `ColMax <- function(x) apply(x,2,max)`
- Sapply/Lappy
  - `sapply(1:3, function(x) x^2)`
- Mappy
  - `mapply(rep, 1:4, 4:1)`
- Tapply
  - `tapply(tbl$mpg,tbl$cyl,mean)`

mapply Example

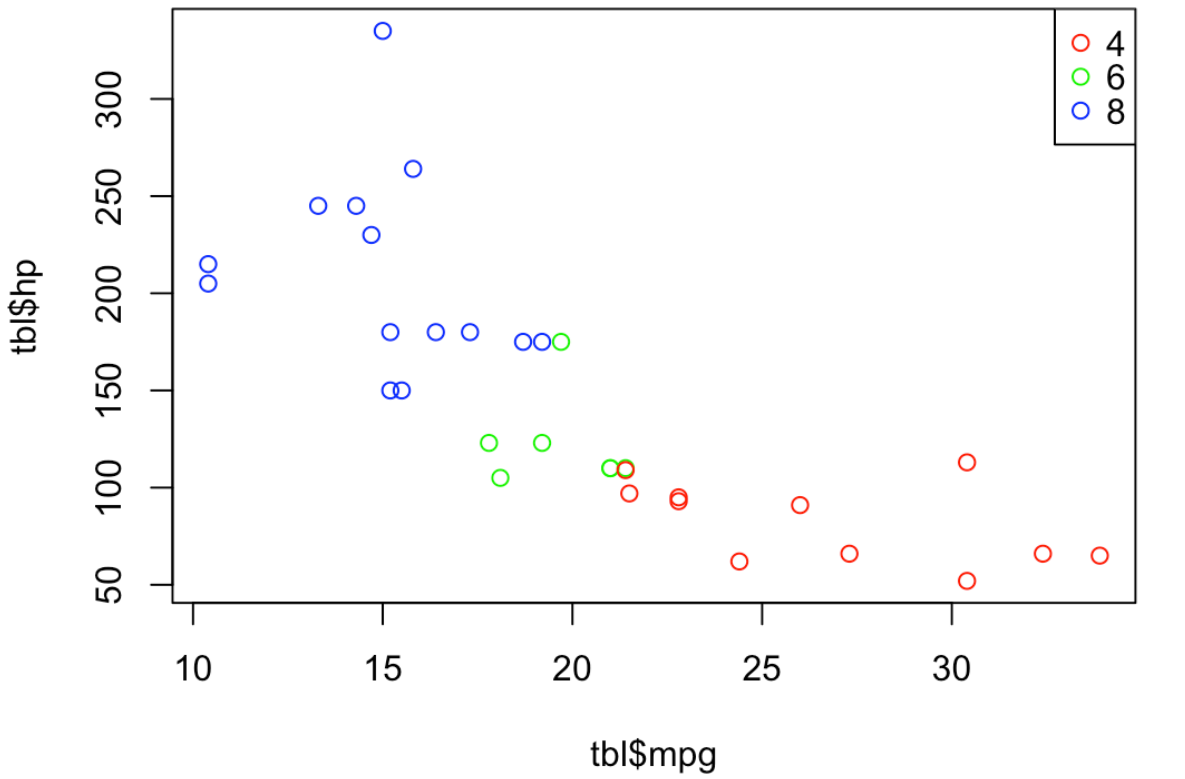
```
> mapply(rep, 1:4, 4:1)
[[1]]
[1] 1 1 1 1

[[2]]
[1] 2 2 2

[[3]]
[1] 3 3

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

sapply Example

```
> choose.col <- function(cyl){
+   if(cyl==4){col <- "red"}
+   if(cyl==6){col <- "green"}
+   if(cyl==8){col <- "blue"}
+   return(col)
+ }
> pch.col <- sapply(tbl$cyl, choose.col)
> cbind(tbl$cyl,pch.col)[1:5,]
         pch.col
[1,] "6" "green"
[2,] "6" "green"
[3,] "4" "red"
[4,] "6" "green"
[5,] "8" "blue"
```

Calling function in plot

```
plot(tbl$mpg, tbl$hp, col=sapply(tbl$cyl, choose.col) )
legend("topright", legend=c(4,6,8), col=c("red","green","blue"), pch=1)
```



Calling Functions

- Functions can be stored in the script
- To use functions in many scripts, they can be saved in their own files or as a function “set”
- Use Source to call functions in another file
  - source(“square_functions.R”)

Save & Load

- R Objects (variables) can be saved into a file
  - save(mut.list,file='mult_list.Rda')
- Saved Objects can be loaded into a new session
  - load('mult_list.Rda')

Export Table

- write.table (tab or comma delimited)
  - write.table(mydata, "mydata.txt", sep="\t",quote=FALSE,row.names=TRUE)
  - write.table(mydata, "mydata.txt", sep=",",quote=TRUE,row.names=TRUE)
- write.xlsx
  - library(xlsx)
  - write.xlsx(mydata, "mydata.xlsx")

Graphical Outputs

- postscript
  - `postscript(file="cool_plot.ps",paper="letter",horizontal=TRUE)`
- png
  - `png(filename = "mpg_by_cyl.png",width = 480, height = 480)`
- tiff
  - `tiff(filename = "mpg_by_cyl.tiff",width = 480, height = 480)`

***Most scientific journals accept eps or tiff for final figures**

Putting it all together

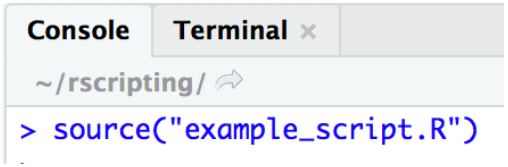
example_script.R

Source on Save | Run | Source

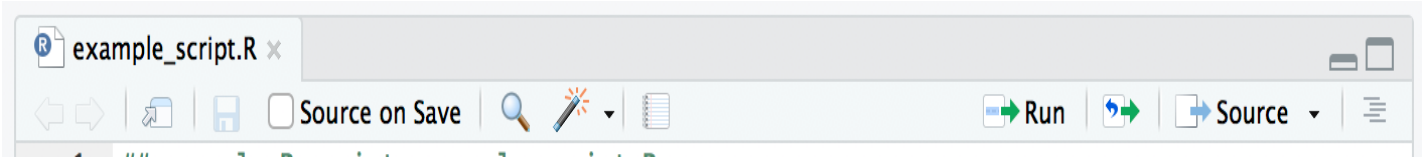
```
1  ## example R script: example_script.R
2
3  choose.col <- function(cyl){
4    if(cyl==4){col <- "red"}
5    if(cyl==6){col <- "green"}
6    if(cyl==8){col <- "blue"}
7    return(col)
8  }
9
10 tbl <- read.table(file="mtcars.csv", sep=',', header=T, row.names=1)
11
12 postscript(file="cool_plot.ps", paper="letter", horizontal=TRUE)
13 plot(tbl$mpg, tbl$hp, col=sapply(tbl$cyl, choose.col) )
14 legend("topright", legend=c(4,6,8), col=c("red","green","blue"), pch=1)
15 dev.off()
16
17 mpg.cyl <- aggregate(mpg ~ cyl, tbl, summary)
18 write.csv(mpg.cyl, file="mpg_summary_cyl.csv", row.names=F)
19
20 png(filename = "mpg_by_cyl.png", width = 480, height = 480)
21 boxplot(tbl$mpg ~ tbl$cyl, xlab="cyl", ylab="mpg")
22 dev.off()
```

Executing R Script

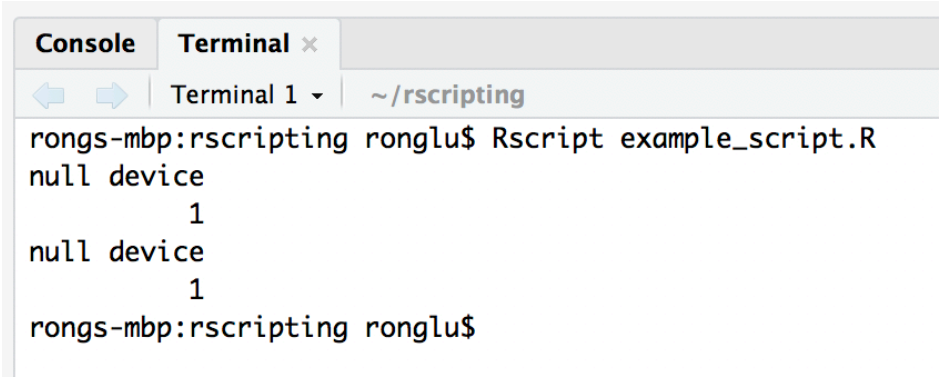
- Source



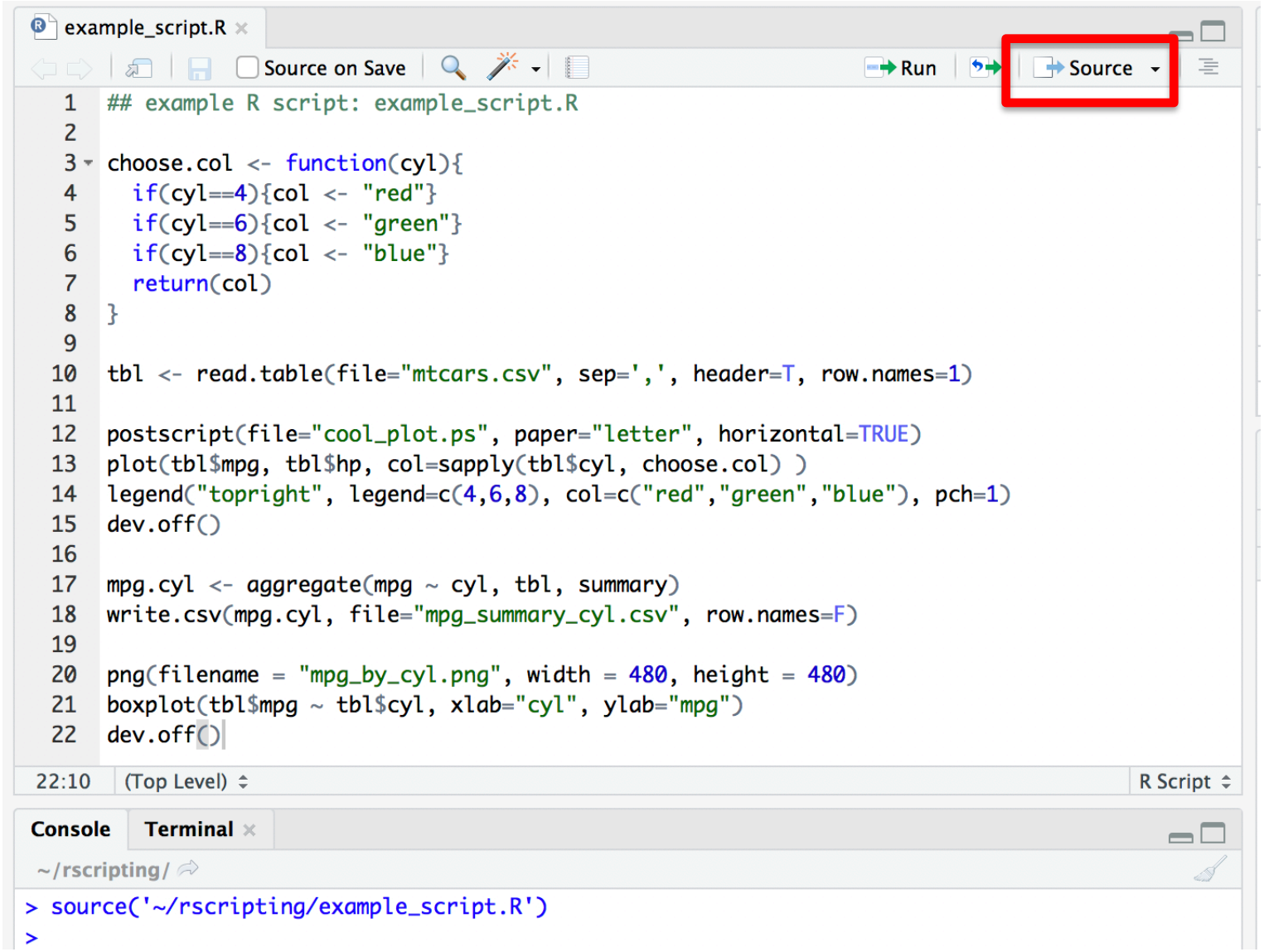
- Run



- Rscript (Terminal Command-line)



Source R Script



Command Line Arguments

- commandArgs
  - accepts values on the command-line and pushes them into an array in the order of the values
- argparse
  - accepts values on the command-line using "command line options"
  - prints out help messages
  - Need to install the Python dependency. Python must be at least version 3.0 Python must have access to the modules: argparse, json | simplejson

commandArgs

exp_plot.R

Source on Save | Run | Source

```
1  ## example R script: exp_plot.R
2  ## On the command-line: Rscript exp_plot.R 2 10
3
4
5  args<-commandArgs(TRUE)   # Get variables from command line
6  num1 <- as.numeric(args[1])
7  num2 <- as.numeric(args[2])
8
9  square <- function(x,n=c(1:num2)) { x^n }
10
11 x <- c(1:num2)
12 y <- square(num1)
13
14 postscript(file="exp_plot.ps", paper="letter", horizontal=TRUE)
15 plot(x, y, ylab=paste(num1,"^x",sep=""))
16 dev.off()
```

16:10 (Top Level) R Script

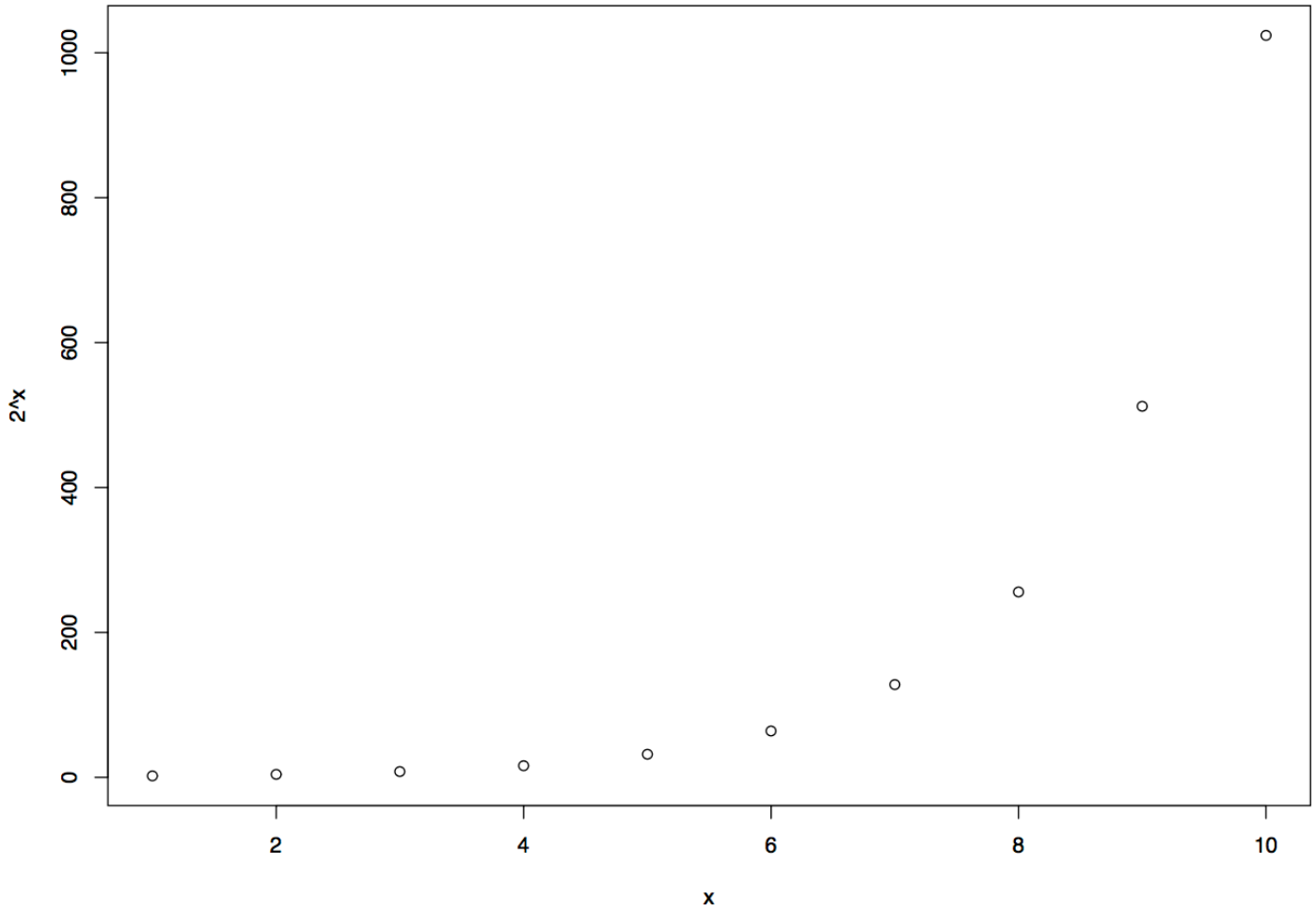
Console Terminal

Terminal 1 ~/rscripting

```
rongs-mbp:rscripting ronglu$ Rscript exp_plot.R 2 10
null device
          1
rongs-mbp:rscripting ronglu$
```

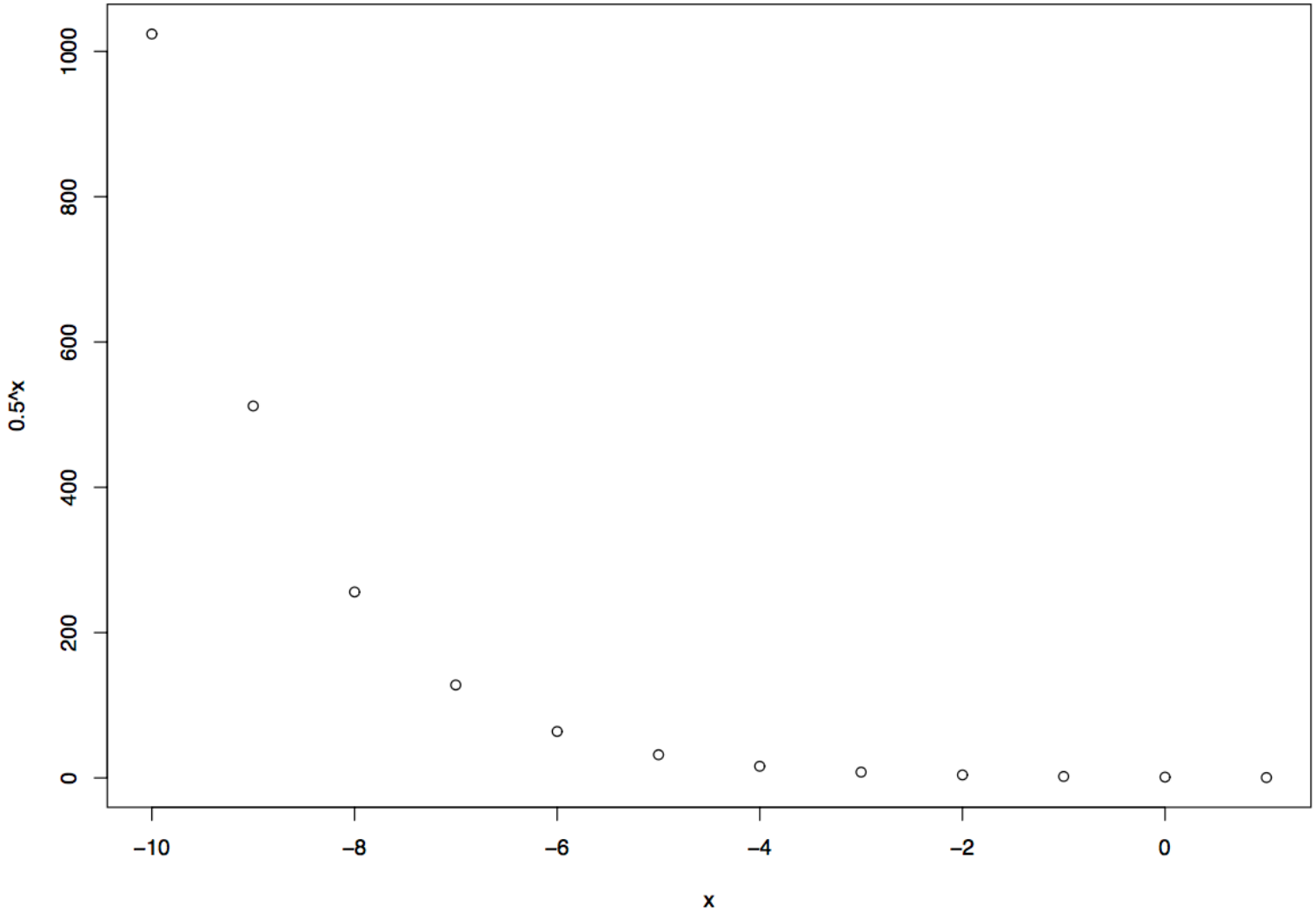
commandArgs

Rscript exp_plot.R 2 10



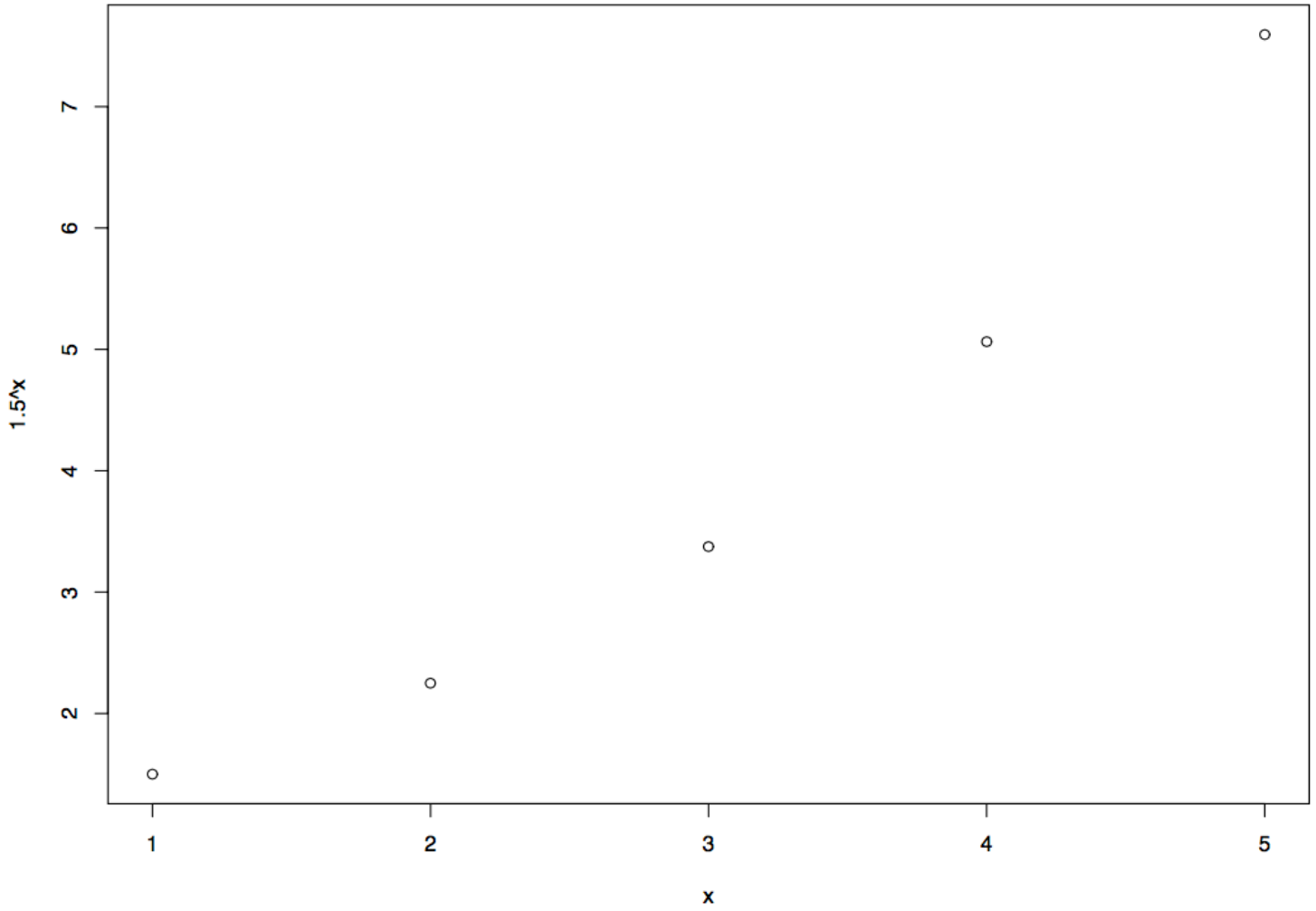
commandArgs

Rscript exp_plot.R 0.5 -10



commandArgs

Rscript exp_plot.R 1.5 5



10-Minute Break

Workshop Starts in 10 Minutes