R Scripting

Programming in R

Advantages of Scripting

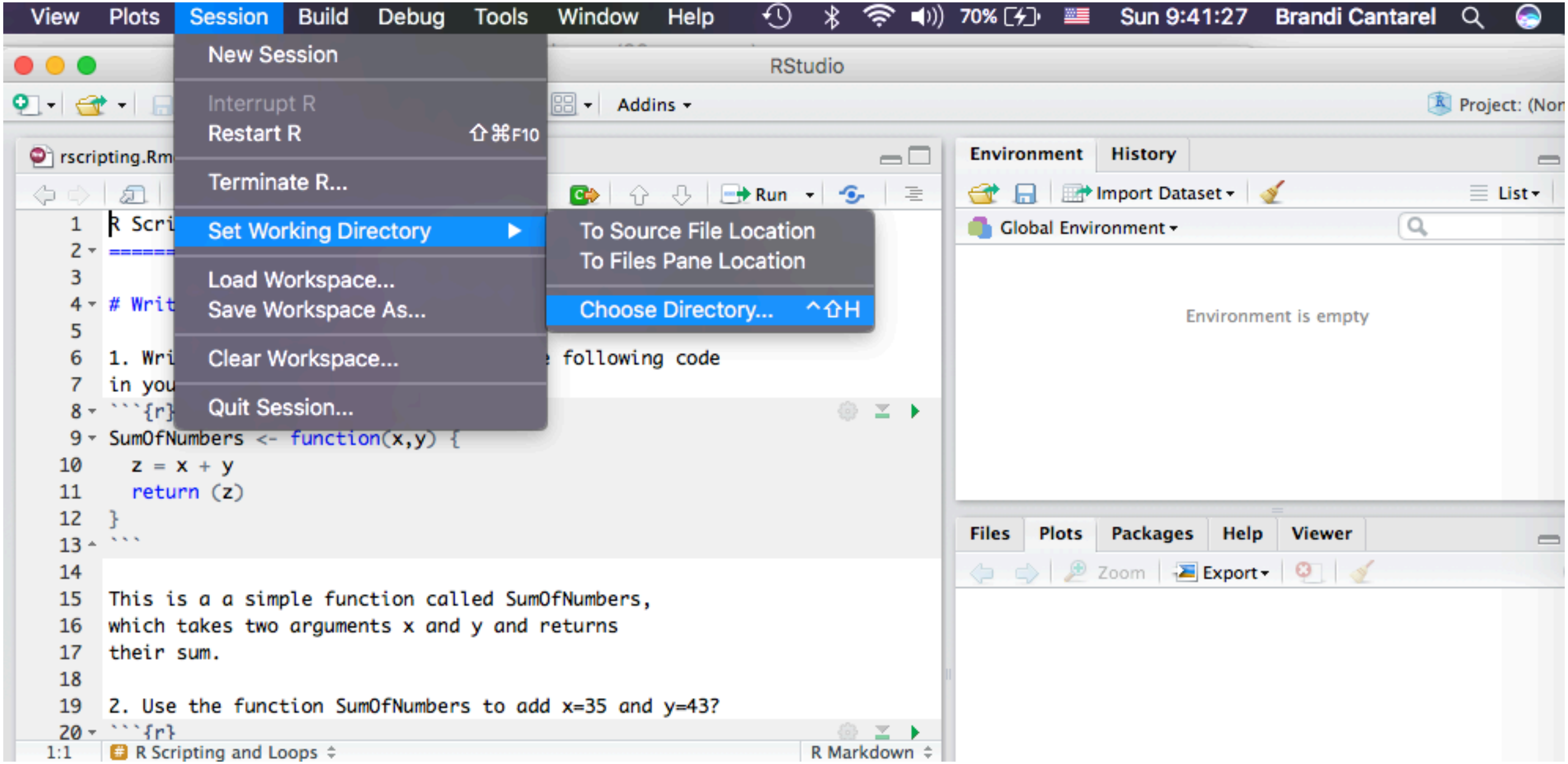
An R script is simply a text file containing (almost) the same commands that you would enter on the command line of R

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

Elements of the R scripts

- Set or 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 a Working Directory

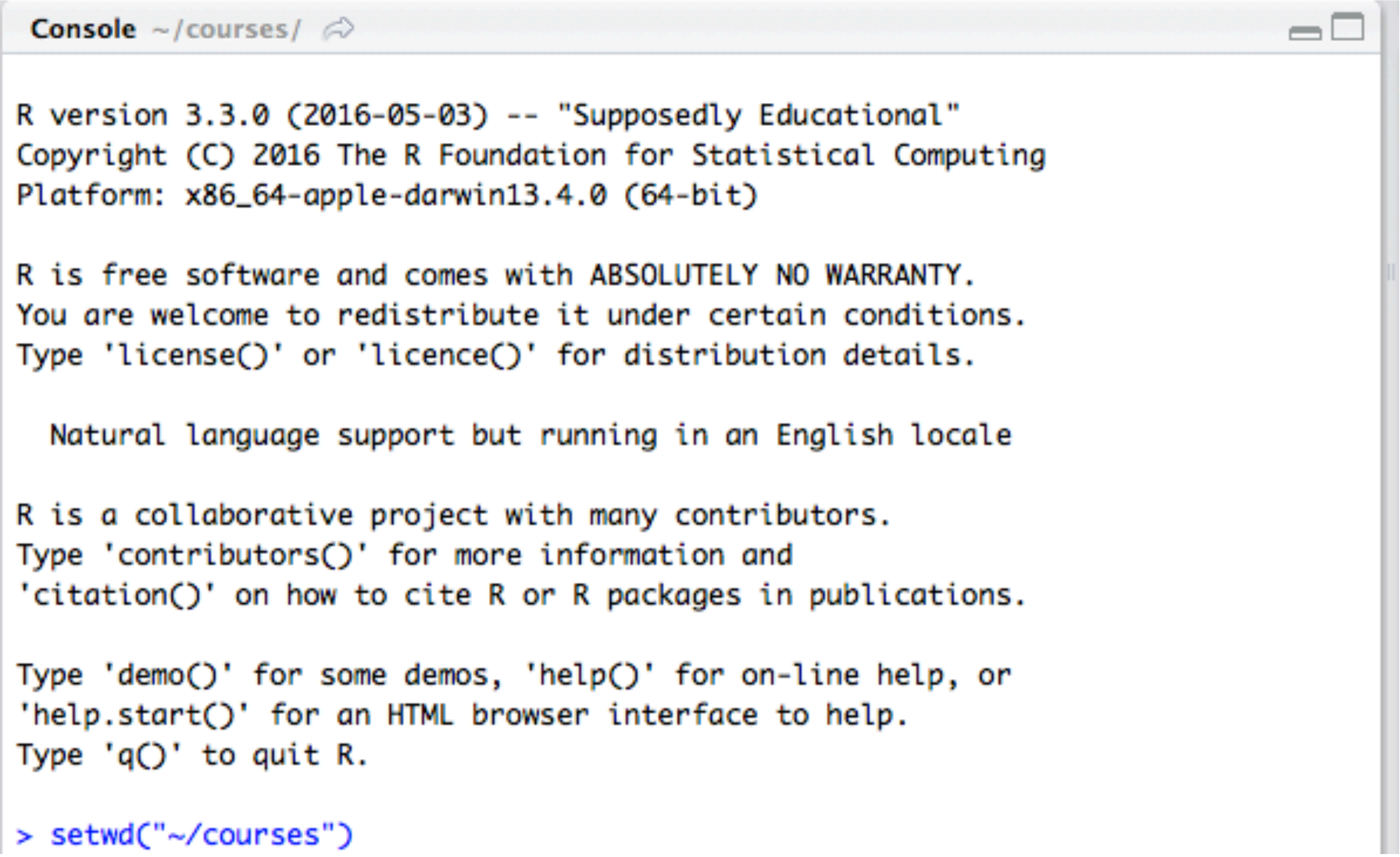


~~~
R Scri
======

# Writ

1. Wri                                        following code
in you
```{r}
SumOfNumbers <- function(x,y) {
  z = x + y
  return (z)
}
~~~

This is a a simple function called SumOfNumbers,
which takes two arguments x and y and returns
their sum.

2. Use the function SumOfNumbers to add x=35 and y=43?
```{r}
```

Selecting a Working Directory



```
R version 3.3.0 (2016-05-03) -- "Supposedly Educational"
Copyright (C) 2016 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.

> setwd("~/courses")
```

```
> datadir <- "~/courses/rscripting"
> setwd(datadir)
> list.files(datadir)
[1] "correlation_plot.R"    "mtcars.csv"               "multi_plots.R"
[4] "rscripting.Rmd"        "rscriptingAnswers.Rmd" "statistical_tests.R"
```

Elements of the R scripts

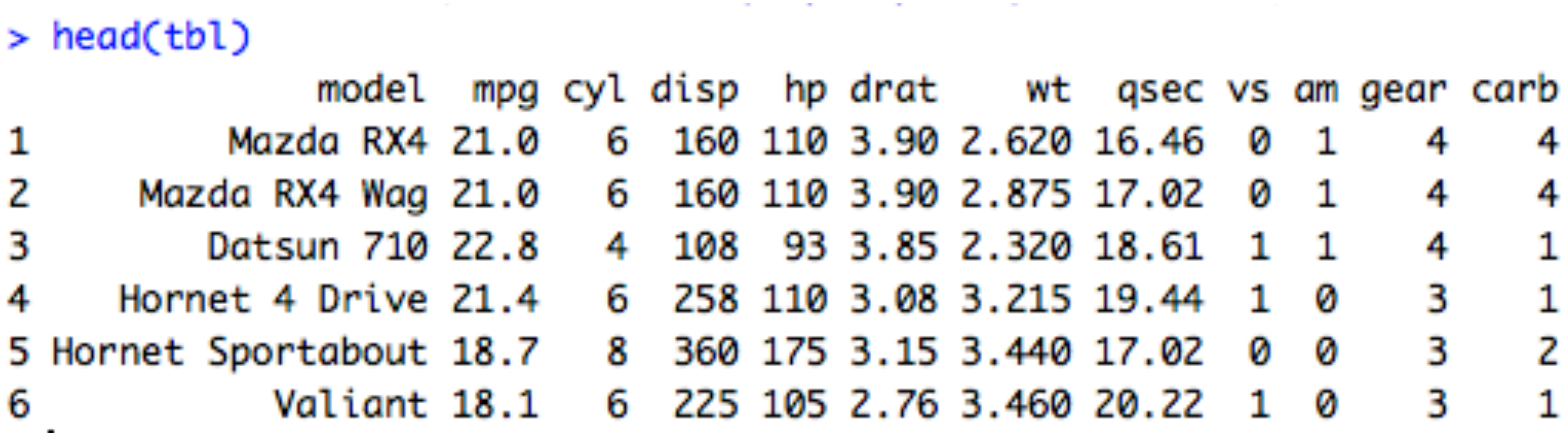
- Set or Assume a Working Directory
  - Where are the input and output files being read and written?
- Input Data
- Processes Data, Run Statistical Analysis or Generate Plots
- Output figures and tables

Reading in Data From A File

```
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)
```

Reading in Data From A File

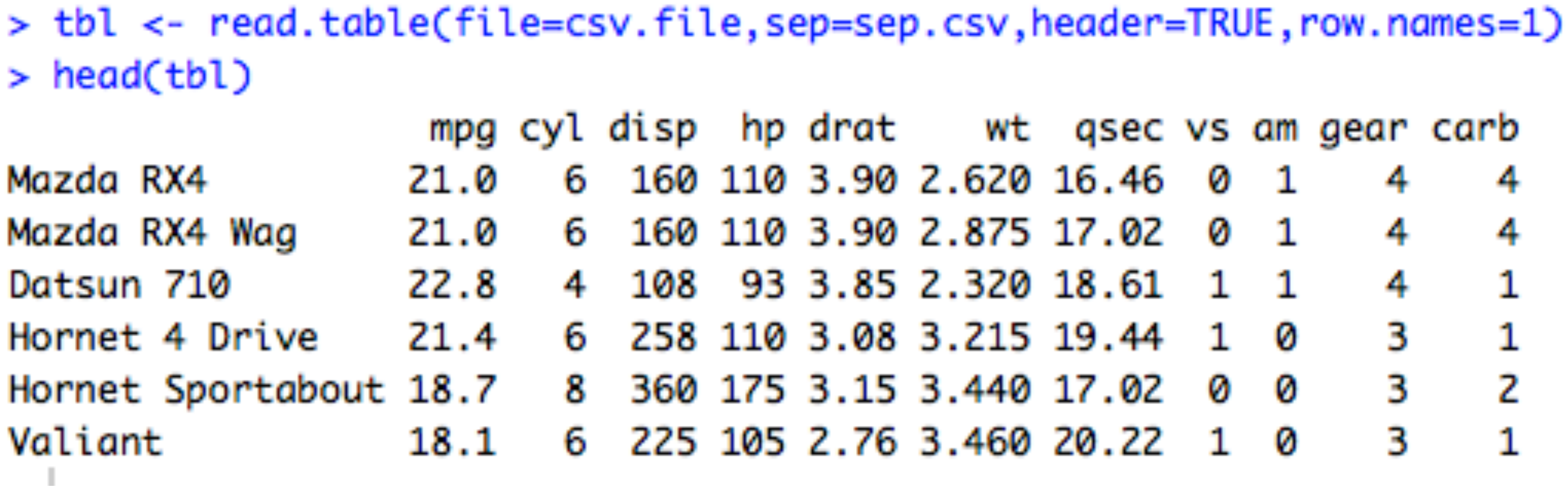
- sep.csv <- ‘,'   ### tab = “\t”
- csv.file <- "mtcars.csv"
- tbl <- read.table(file=csv.file,sep=sep.csv,header=TRUE)



```
> head(tbl)
              model  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
```

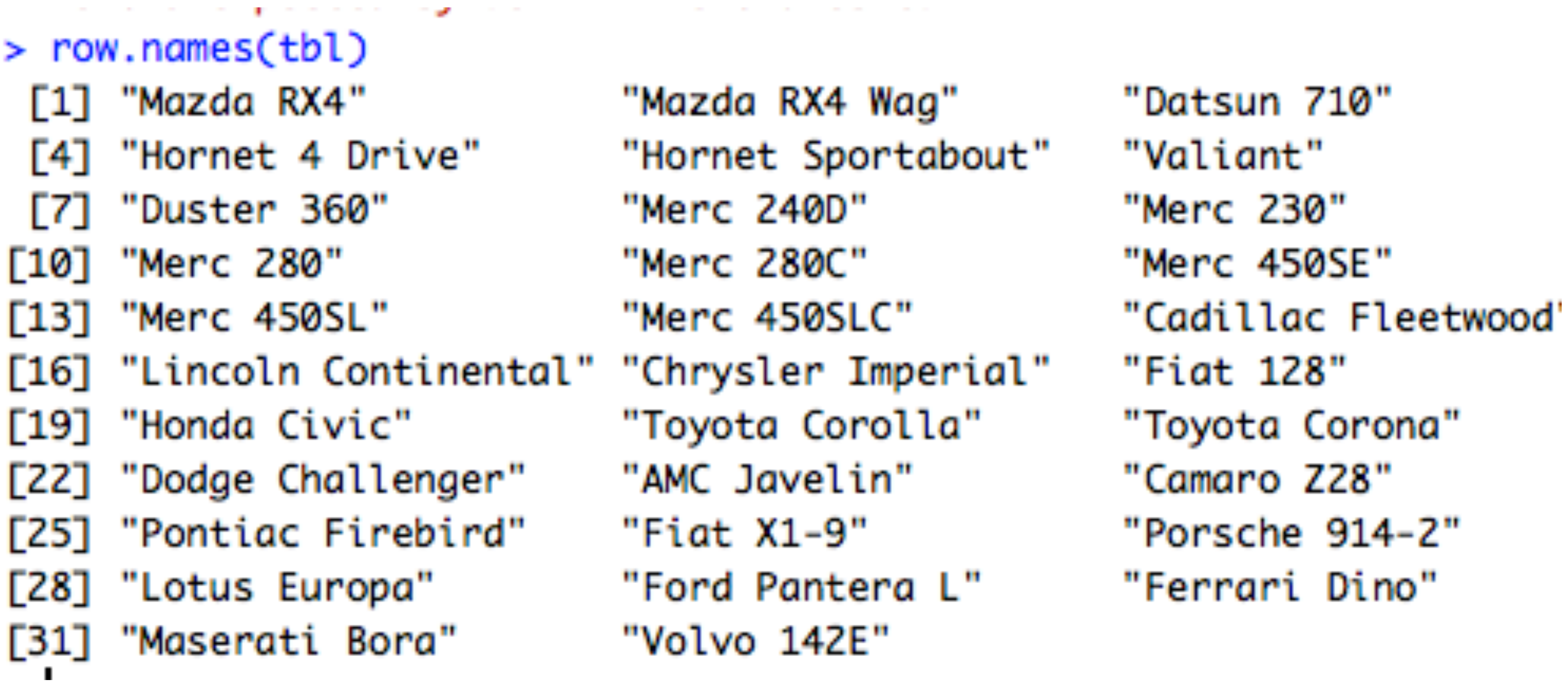
Row and Column Names

- `tbl <- read.table(file=csv.file,sep=sep.csv,header=TRUE,row.names=1)`

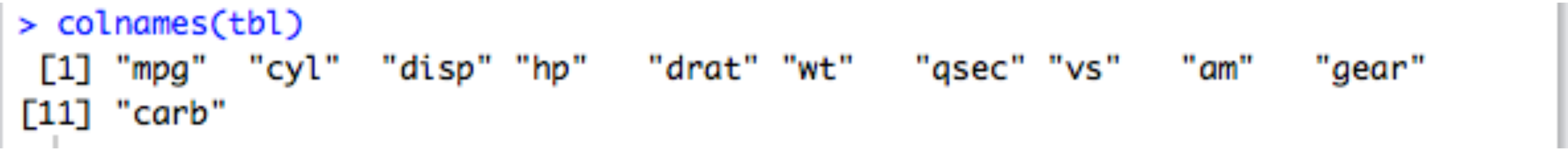


```
> tbl <- read.table(file=csv.file,sep=sep.csv,header=TRUE,row.names=1)
> head(tbl)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

Row and Column Names



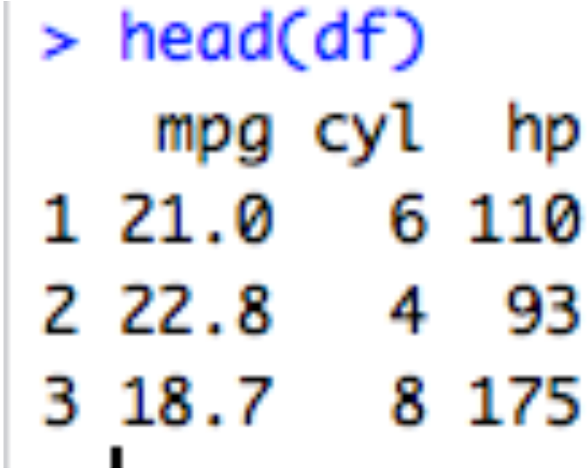
```
> row.names(tbl)
 [1] "Mazda RX4"           "Mazda RX4 Wag"       "Datsun 710"
 [4] "Hornet 4 Drive"      "Hornet Sportabout"   "Valiant"
 [7] "Duster 360"          "Merc 240D"           "Merc 230"
[10] "Merc 280"            "Merc 280C"           "Merc 450SE"
[13] "Merc 450SL"          "Merc 450SLC"         "Cadillac Fleetwood'
[16] "Lincoln Continental" "Chrysler Imperial"   "Fiat 128"
[19] "Honda Civic"         "Toyota Corolla"      "Toyota Corona"
[22] "Dodge Challenger"    "AMC Javelin"         "Camaro Z28"
[25] "Pontiac Firebird"    "Fiat X1-9"           "Porsche 914-2"
[28] "Lotus Europa"        "Ford Pantera L"      "Ferrari Dino"
[31] "Maserati Bora"       "Volvo 142E"
```



```
> colnames(tbl)
 [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
[11] "carb"
```

Creating a Table on the Fly

```
mpg = c(21, 22.8, 18.7)
cyl = c(6, 4, 8)
hp = c(110, 93, 175)
df = data.frame(mpg, cyl, hp)
```



```
> head(df)
   mpg cyl  hp
1 21.0   6 110
2 22.8   4  93
3 18.7   8 175
```

Elements of the R scripts

- Set or 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

Programming Functions

- Conditional Statements
  - If and else
- Loops
  - For and apply
- Functions
  - user defined calculations
  - calling on 3rd party and built-in functions

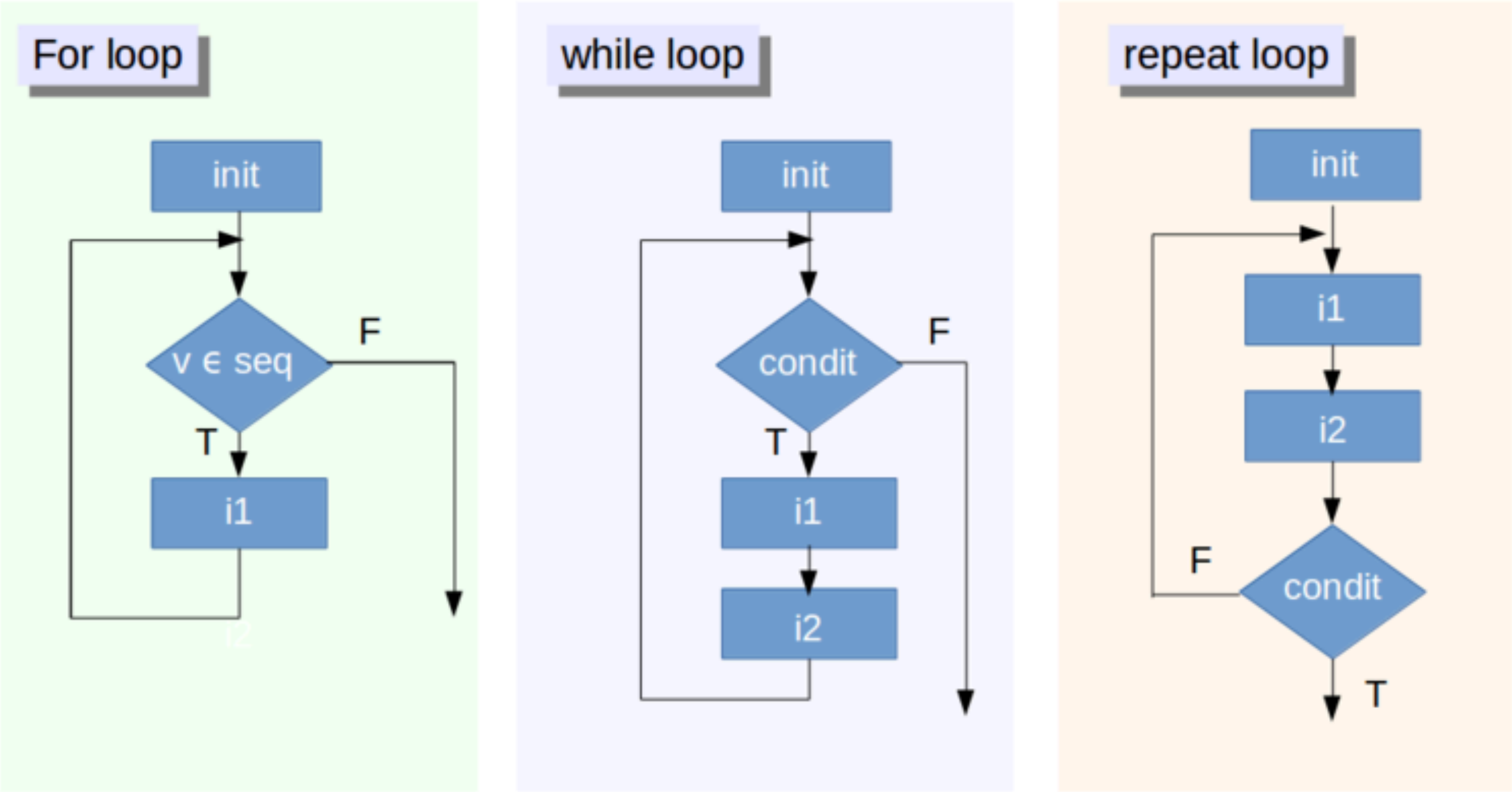
If Statements

```
x <- 2

if (x > 0) {

log.x <- log2(x)

}

log.x
```

If/Else Statements

```
x <- 2
if (x > 0) {
log.x <- log2(x)
}else {
log.x <- 1
}
log.x
```

Loop Structure in R

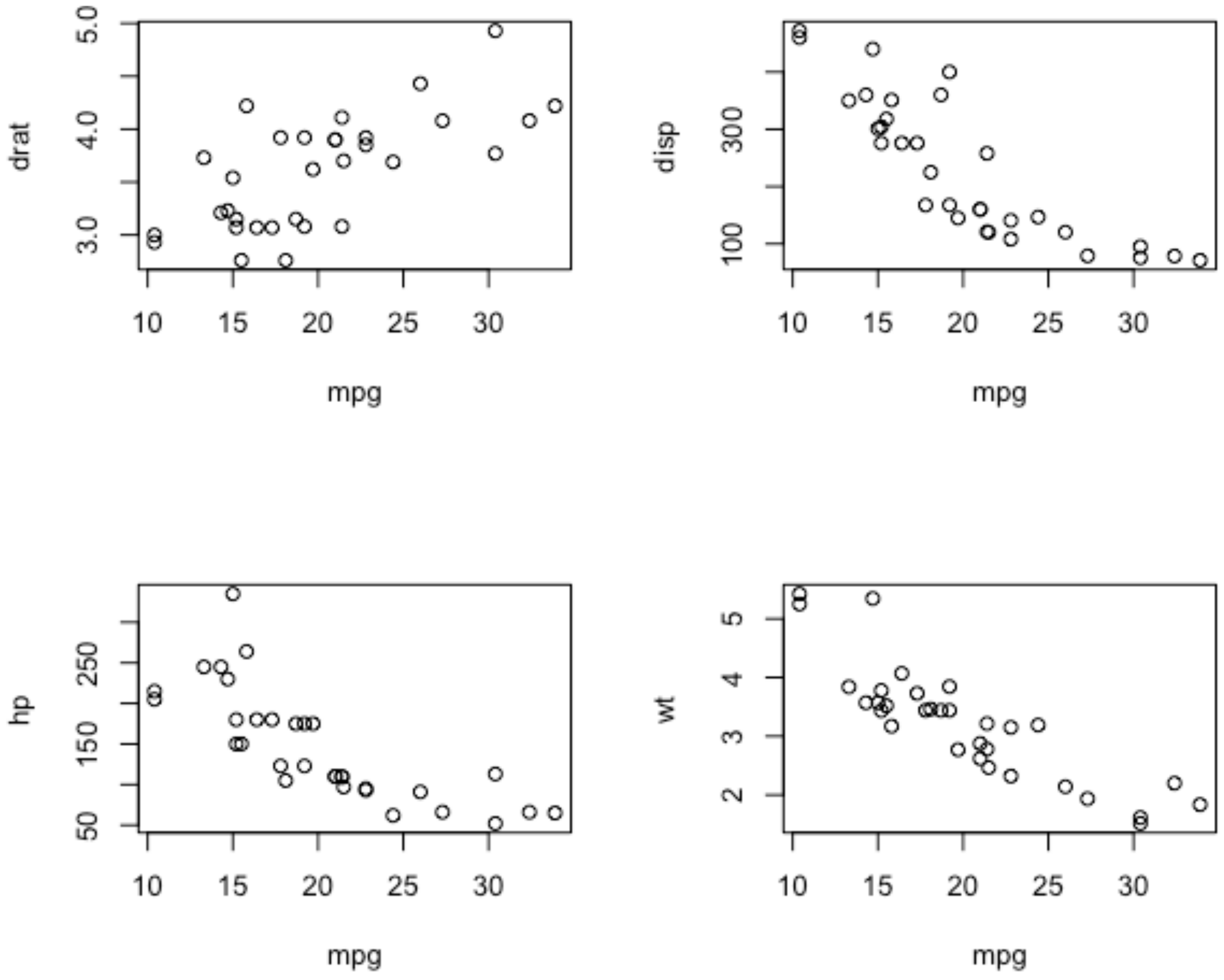


For Loops

```
Ys <- c(“drat”,”disp","hp","wt")
x <- tbl$mpg
par(mfrow=c(2,2))
for (i in 1:4) {
 y <- y <- tbl[,Ys[i]]
 plot(x,y,xlab=“mpg",ylab=Ys[i])
 }
```

For Loop to Create A Plot

```
Ys <- c(“drat”,”disp","hp","wt")
x <- tbl$mpg
par(mfrow=c(2,2))
for (i in 1:4) {
  y <- y <- tbl[,Ys[i]]
  plot(x,y,xlab=“mpg",ylab=Ys[i])
  }
```

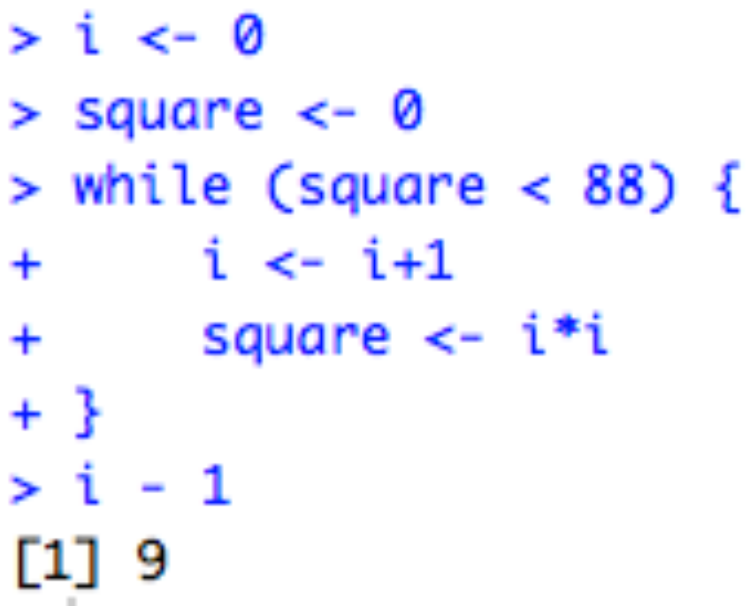


Nested For Loops

```
Ys <- c("drat","disp","hp")
Xs <- c("mpg","gear","carb")
for (i in 1:3) {
    x <- tbl[,Xs[i]]
    for (j in 1:3) {
        y <- y <- tbl[,Ys[j]]
        plot(x,y,xlab=Xs[i],ylab=Ys[j])
    }
}
```

While Loops

```
i <- 0
square <- 0
while (square < 88) {
 i <- i+1
 square <- i*i
 }
 i - 1
```



```
> i <- 0
> square <- 0
> while (square < 88) {
+     i <- i+1
+     square <- i*i
+ }
> i - 1
[1] 9
```

Repeat

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

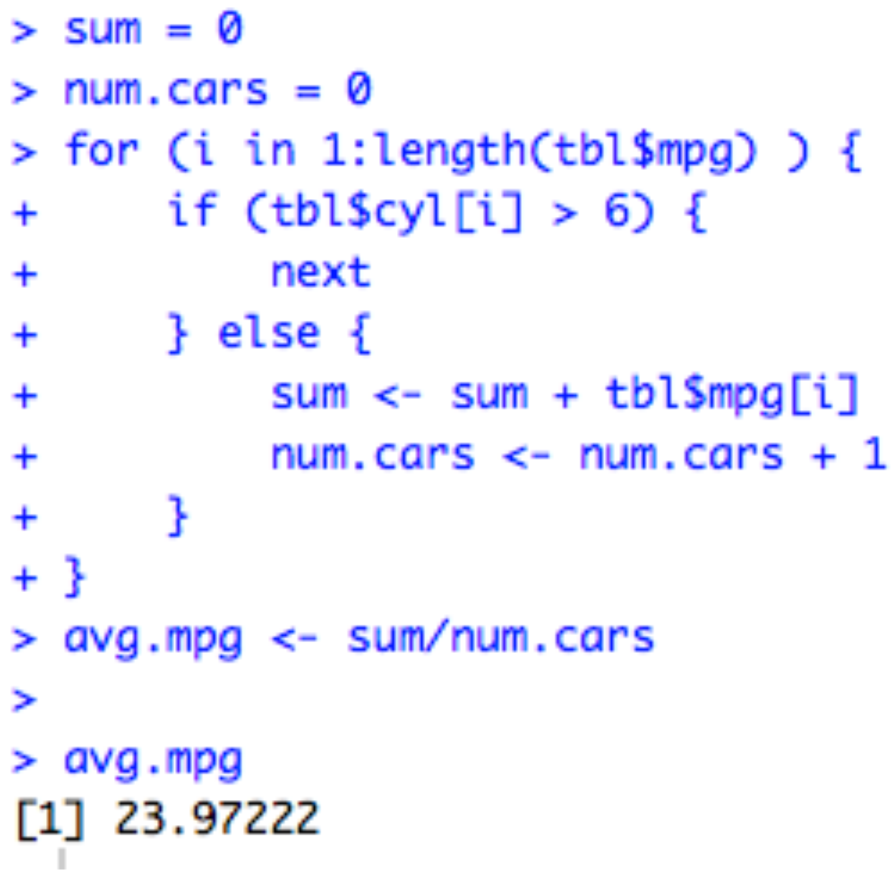
Without a “break” conditional repeats are infinite loops

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

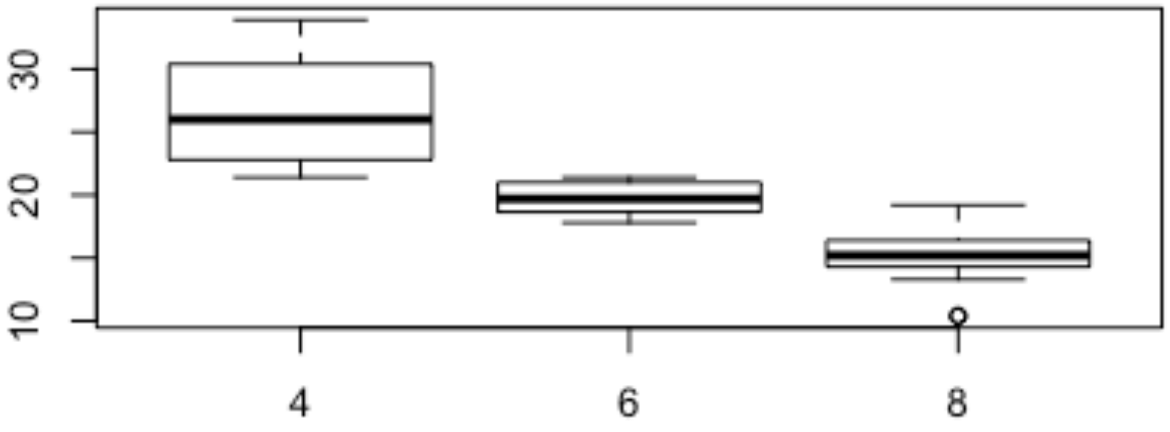
```
sum = 0
num.cars = 0
for (i in 1:length(tbl$mpg) ) {
    if (tbl$cyl[i] > 6) {
        next
    } else {
        sum <- sum + tbl$mpg[i]
        num.cars <- num.cars + 1
    }
}
avg.mpg <- sum/num.cars
```



```
> sum = 0
> num.cars = 0
> for (i in 1:length(tbl$mpg) ) {
+     if (tbl$cyl[i] > 6) {
+         next
+     } else {
+         sum <- sum + tbl$mpg[i]
+         num.cars <- num.cars + 1
+     }
+ }
> avg.mpg <- sum/num.cars
>
> avg.mpg
[1] 23.97222
```

aggregate

```
>aggregate(mpg ~ cyl, tbl, mean)
  cyl      mpg
1   4 26.66364
2   6 19.74286
3   8 15.10000
>boxplot(tbl$mpg ~ tbl$cyl)
```



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

```
> replicate(12,rnorm(10))
            [,1]       [,2]        [,3]        [,4]        [,5]       [,6]
 [1,] -2.4484221 -0.1425849 -0.33104757  0.02131725 -0.51043217  0.6299466
 [2,]  0.1922703  0.1823492 -1.22800137 -1.13147301  0.03581989 -1.2599562
 [3,]  0.5017231  2.0178642  0.03438109  0.98566852 -0.72803750  0.7020802
 [4,]  1.4442887  0.6111521  0.01140353 -0.42082194  1.17832161  1.2882195
 [5,]  1.1524227 -0.9662884  0.06398537  2.13513351 -0.13679937  0.4534989
 [6,]  2.1098891  0.2999618  0.47736012  0.95958254  1.66187993 -1.1621893
 [7,]  1.4262934  0.5653800  0.32676984  1.06038741  0.08110211 -1.2467717
 [8,] -0.8335361 -0.9941313  1.16202298 -0.03162884  0.71558689 -0.5021393
 [9,] -0.5051440 -0.0198518 -2.22389459 -0.76559150 -0.42438225  0.8612826
[10,]  0.1673019 -0.8884450 -1.27388283  0.55279621 -0.74381015 -0.2384732
            [,7]        [,8]       [,9]      [,10]      [,11]       [,12]
 [1,] -1.1074932  1.43061337 -0.4752098  0.1267295 -1.4341393 -0.39218672
 [2,] -0.5775814  0.33127789  0.3512621 -0.2248461  0.1112402  1.05474794
 [3,]  0.7867060  0.52046312  1.8914330  0.3709762 -0.7369606 -0.66392587
 [4,]  0.4120193 -1.79027281  0.1290036  0.4780985  1.3180441 -0.68438674
 [5,] -1.0544244  0.50192616 -0.2211385  0.1509953 -1.3769244 -0.94383187
 [6,] -1.1150998 -0.85835157  1.3853010 -1.1402802  0.9414104 -0.84507143
 [7,] -0.2715535  0.18882009  1.5430503 -0.1789196  0.1631383  0.74051938
 [8,]  0.3062152 -0.04821108  1.0386594  0.6944796 -0.8141440  0.41171604
 [9,] -1.1613990  0.88830458 -1.4306303  0.6175879  0.7409139 -1.42885670
[10,] -1.4743401  0.03121555 -0.8163763  1.5238221 -0.6018527  0.05055171
```

User Defined Functions

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

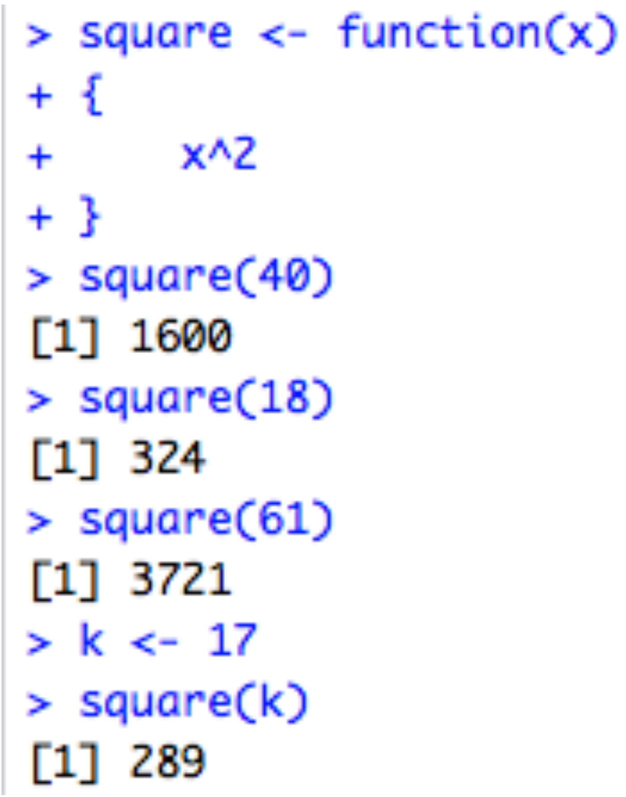
Functions are just a sets of instructions that we want to use repeatedly or that, because of their complexity, are better self-contained in a sub program and called when needed.

Basic Function Elements

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

Basic Function Elements

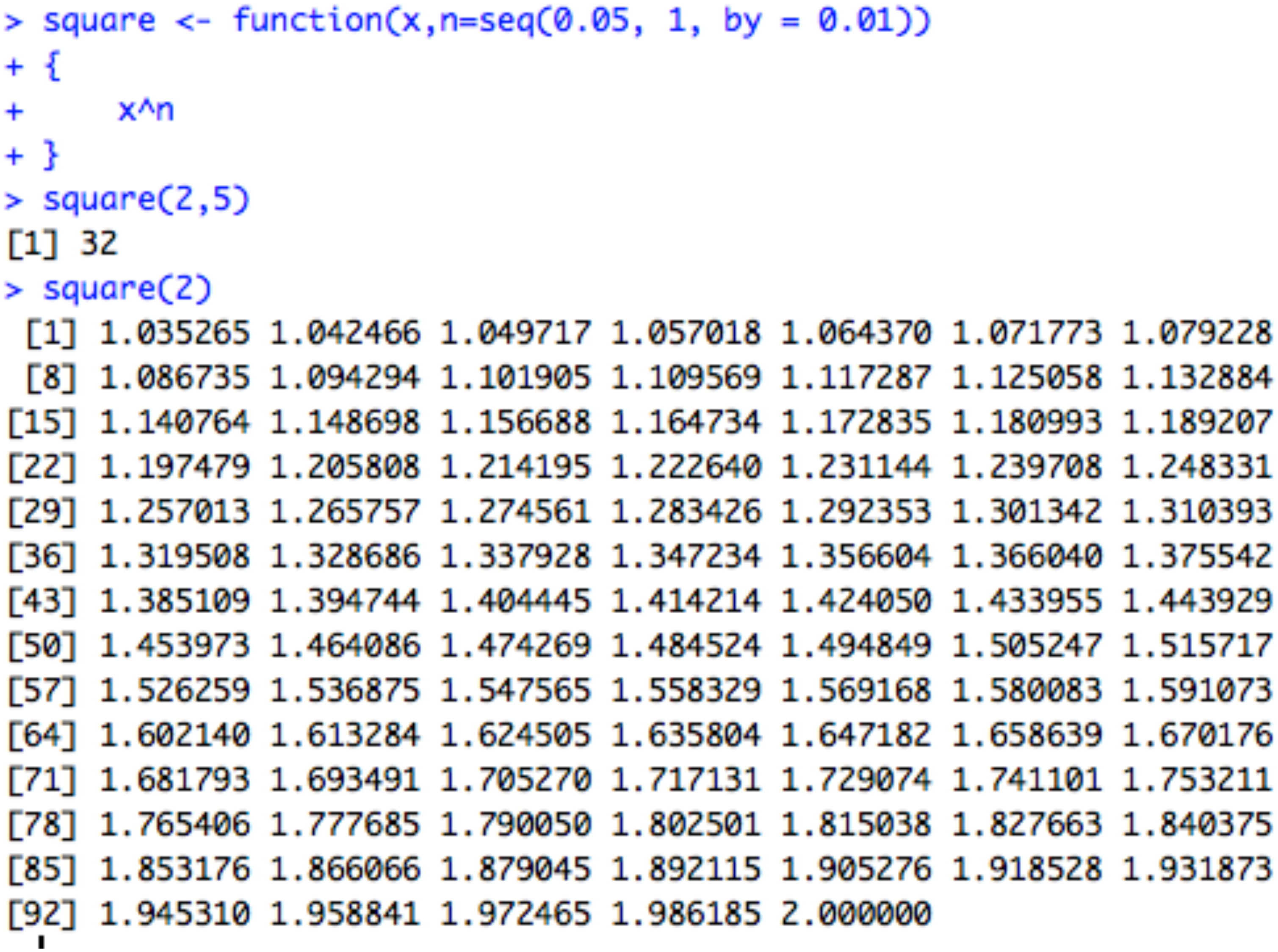
```
square <- function(x)

{

x^2

}
```



```
> square <- function(x)
+ {
+     x^2
+ }
> square(40)
[1] 1600
> square(18)
[1] 324
> square(61)
[1] 3721
> k <- 17
> square(k)
[1] 289
```

Setting a default value range

```
square <- function(x,n=seq(0.05, 1, by =
0.01))

{

x^n

}
```



```
> square <- function(x,n=seq(0.05, 1, by = 0.01))
+ {
+     x^n
+ }
> square(2,5)
[1] 32
> square(2)
 [1] 1.035265 1.042466 1.049717 1.057018 1.064370 1.071773 1.079228
 [8] 1.086735 1.094294 1.101905 1.109569 1.117287 1.125058 1.132884
[15] 1.140764 1.148698 1.156688 1.164734 1.172835 1.180993 1.189207
[22] 1.197479 1.205808 1.214195 1.222640 1.231144 1.239708 1.248331
[29] 1.257013 1.265757 1.274561 1.283426 1.292353 1.301342 1.310393
[36] 1.319508 1.328686 1.337928 1.347234 1.356604 1.366040 1.375542
[43] 1.385109 1.394744 1.404445 1.414214 1.424050 1.433955 1.443929
[50] 1.453973 1.464086 1.474269 1.484524 1.494849 1.505247 1.515717
[57] 1.526259 1.536875 1.547565 1.558329 1.569168 1.580083 1.591073
[64] 1.602140 1.613284 1.624505 1.635804 1.647182 1.658639 1.670176
[71] 1.681793 1.693491 1.705270 1.717131 1.729074 1.741101 1.753211
[78] 1.765406 1.777685 1.790050 1.802501 1.815038 1.827663 1.840375
[85] 1.853176 1.866066 1.879045 1.892115 1.905276 1.918528 1.931873
[92] 1.945310 1.958841 1.972465 1.986185 2.000000
```

Calling Functions in Your Function

```
my.fun <- function(X.matrix, y.vec, z.scalar) {

   sq.scalar <- square(z.scalar,2)
   mult <- X.matrix %*% y.vec
   final <- mult * sq.scalar
   return(final)
}
```

```
> my.fun(my.mat, my.vec, 5)
           [,1]
      [1,]  475
      [2,]  600
      [3,]  625
```

Functions: Returning a List

```
my.fun <- function(X.matrix, y.vec,
z.scalar) {


   sq.scalar <- square(z.scalar,2)

   mult <- X.matrix %*% y.vec

   final <- mult * sq.scalar

   return(list(sq.num=sq.scalar,
matmult=final))

}
```

```
> my.fun(my.mat,
my.vec, 5)
$sq.num
[1] 25

$matmult
     [,1]
[1,]  475
[2,]  600
[3,]  625
```

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

- 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 Functions

- `(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)`

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”

Function with Loop and Conditional

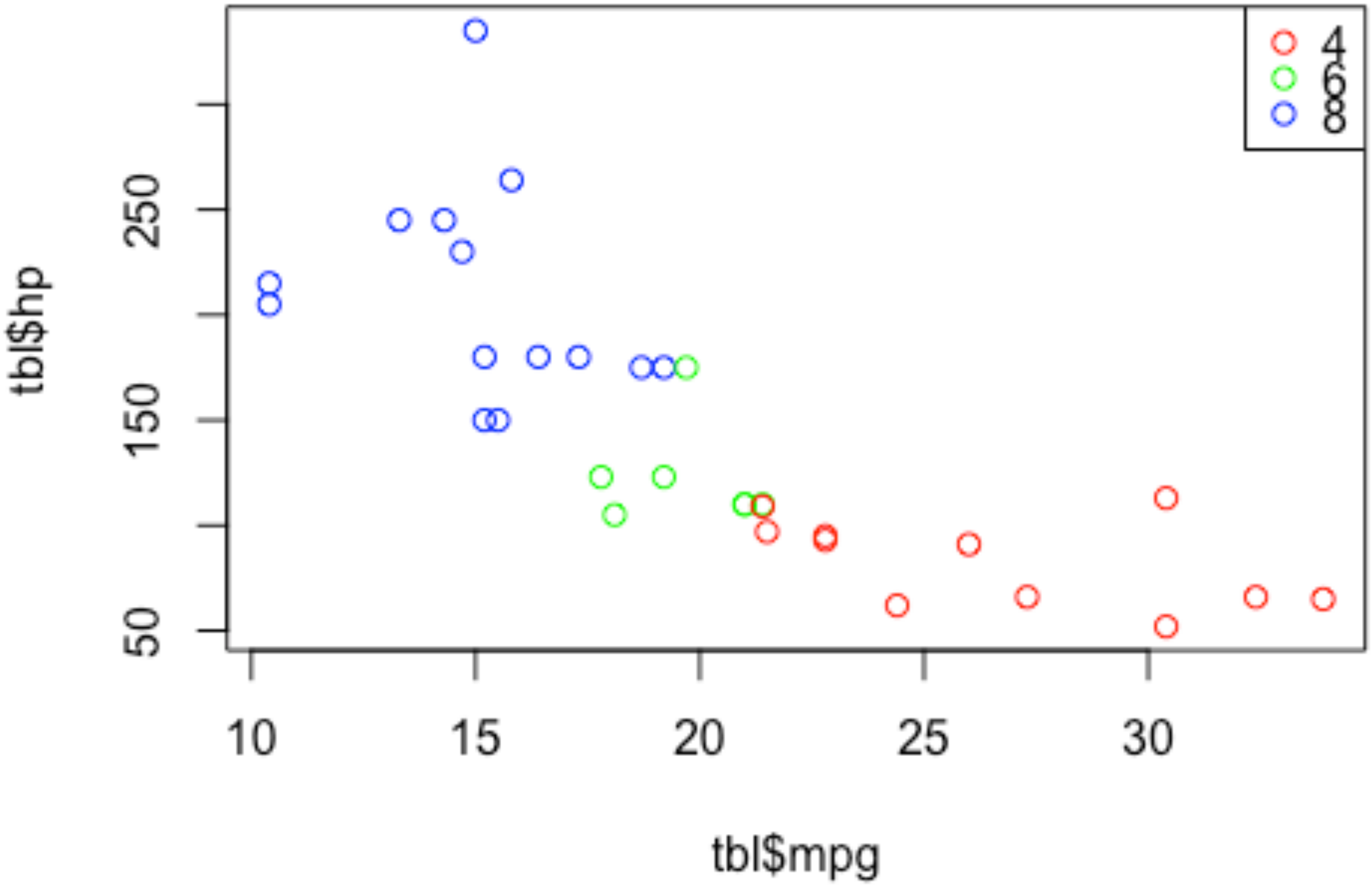
```
r <- rainbow(3)
choose.col <- function(n) {
  colorvec <- vector(mode="character", length=length(n))
  for (i in 1:length(n)) {
    if ( n[i] > 3 ) {
      colorvec[i] = r[1]
    }
    if (n[i] > 5) {
      colorvec[i] = r[2]
    }
    if( n[i] > 7) {
      colorvec[i] = r[3]
    }
  }
  c(colorvec)
}

col.pch <- choose.col(tbl$cyl)
```

Apply and Conditional Function

```
r <- rainbow(3)
choose.col <- function(n) {
  if ( n > 3 ) {
    col.n <- r[1]
  }
  if (n > 5) {
    col.n <- r[2]
  }
  if( n > 7) {
    col.n <- r[3]
  }
 col.n
}

col.pch <- sapply(tbl$cyl,choose.col)
```

Calling Functions in Plot

```
plot(tbl$mpg,tbl$hp,col=sapply(tbl$cyl,choose.col))
legend("topright",legend=c(4,6,8),col=r,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”)

Elements of the R scripts

- Set or 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 objects, figures and tables

Save and 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 except eps or tiff for final figures

Putting it all together

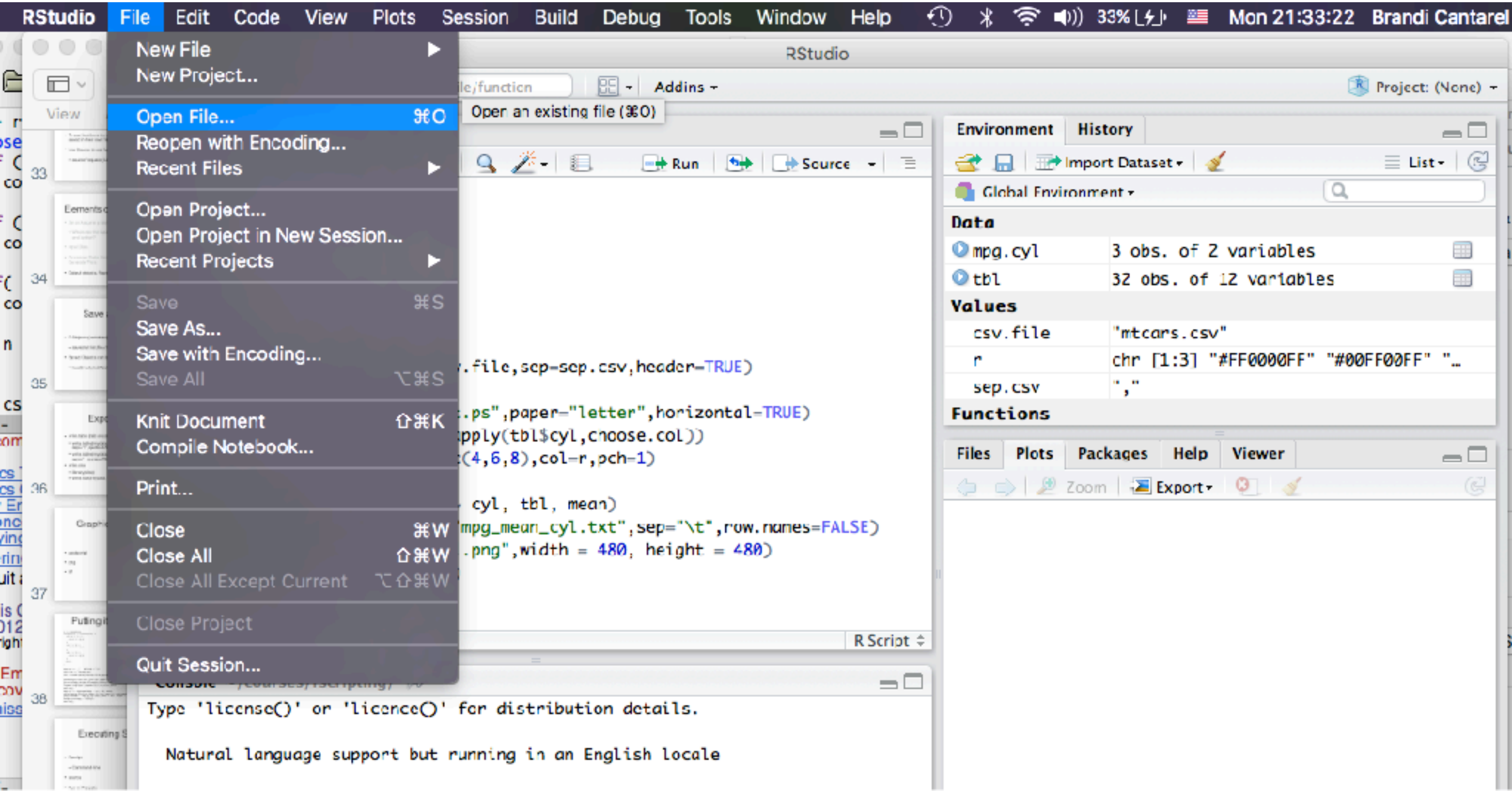
plot_mpg.R

```
r <- rainbow(3)
choose.col <- function(n) {
  if ( n > 3 ) {
    col.n <- r[1]
  }
  if (n > 5) {
    col.n <- r[2]
  }
  if( n > 7) {
    col.n <- r[3]
  }
  col.n
}

sep.csv <- ‘,'   ### tab = “\t”
csv.file <- "mtcars.csv"
tbl <- read.table(file=csv.file,sep=sep.csv,header=TRUE)

postscript(file=“cool_plot.ps”,paper=“letter”,horizontal=TRUE)
plot(tbl$mpg,tbl$hp,col=sapply(tbl$cyl,choose.col))
legend("topright",legend=c(4,6,8),col=r,pch=1)
dev.off()
mpg.cyl <- aggregate(mpg ~ cyl, tbl, mean)
write.table(file=“mpg_mean_cyl.txt”,sep=“\t”,row.names=FALSE)
png(filename = “mpg_by_cyl.png”,width = 480, height = 480)
boxplot(tbl$mpg ~ tbl$cyl)
dev.off()
```

Executing Scripts

- source
- Run in R studio
- Rscript
    - Command-line

source

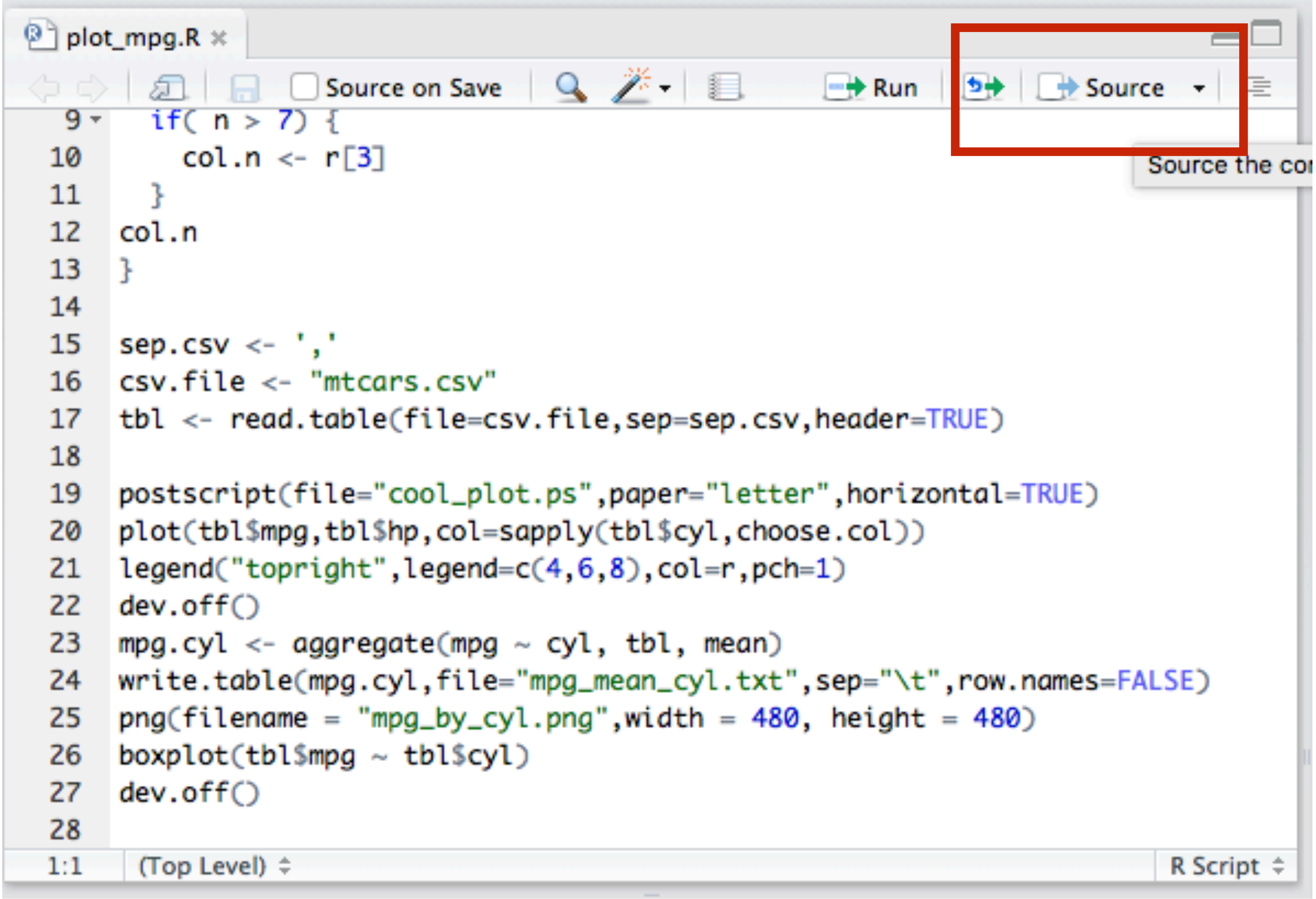


```
.file,sep=sep.csv,header=TRUE)

.ps",paper="letter",horizontal=TRUE)
pply(tbl$cyl,choose.col))
(4,6,8),col=r,pch=1)

 cyl, tbl, mean)
mpg_mean_cyl.txt",sep="\t",row.names=FALSE)
.png",width = 480, height = 480)
```

Data	
mpg.cyl	3 obs. of 2 variables
tbl	32 obs. of 12 variables
Values	
csv.file	"mtcars.csv"
r	chr [1:3] "#FF0000FF" "#00FF00FF" "…
sep.csv	","
Functions	

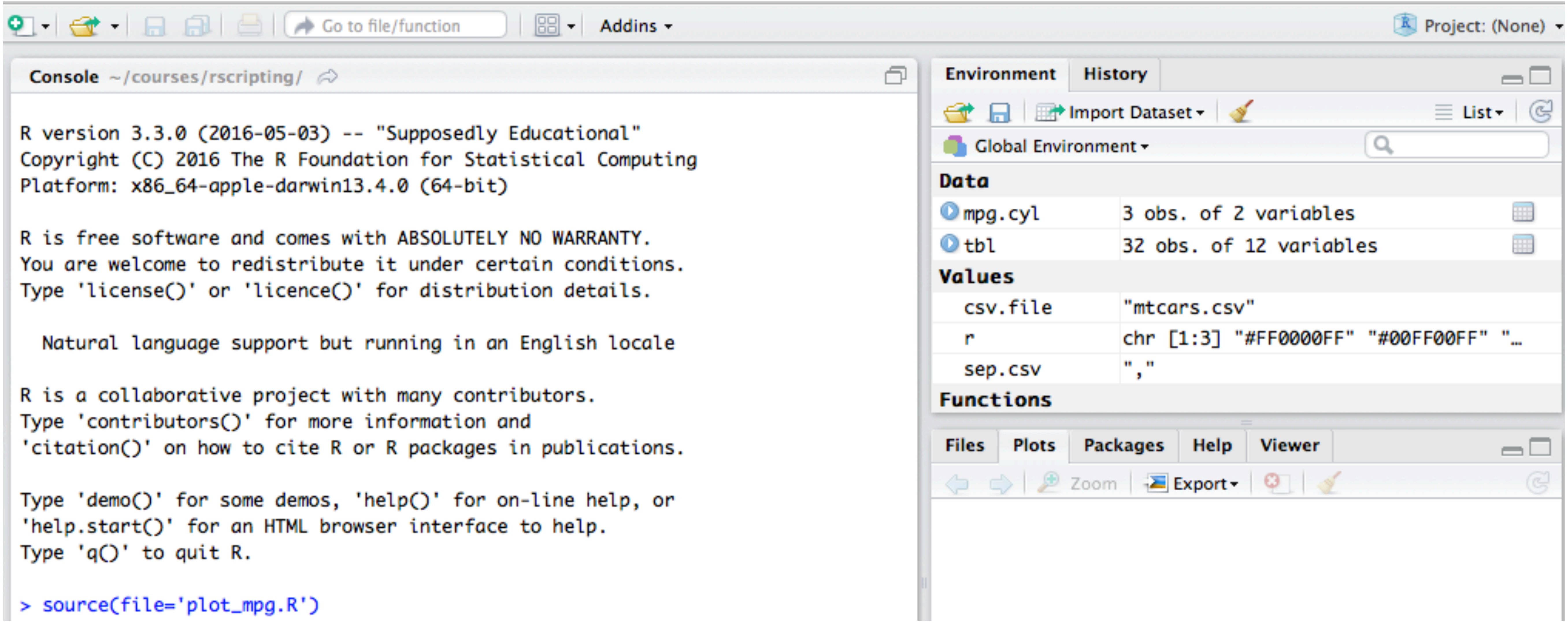
```
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale
```

source



```
  if( n > 7) {
    col.n <- r[3]
  }
col.n
}

sep.csv <- ','
csv.file <- "mtcars.csv"
tbl <- read.table(file=csv.file,sep=sep.csv,header=TRUE)

postscript(file="cool_plot.ps",paper="letter",horizontal=TRUE)
plot(tbl$mpg,tbl$hp,col=sapply(tbl$cyl,choose.col))
legend("topright",legend=c(4,6,8),col=r,pch=1)
dev.off()
mpg.cyl <- aggregate(mpg ~ cyl, tbl, mean)
write.table(mpg.cyl,file="mpg_mean_cyl.txt",sep="\t",row.names=FALSE)
png(filename = "mpg_by_cyl.png",width = 480, height = 480)
boxplot(tbl$mpg ~ tbl$cyl)
dev.off()

```

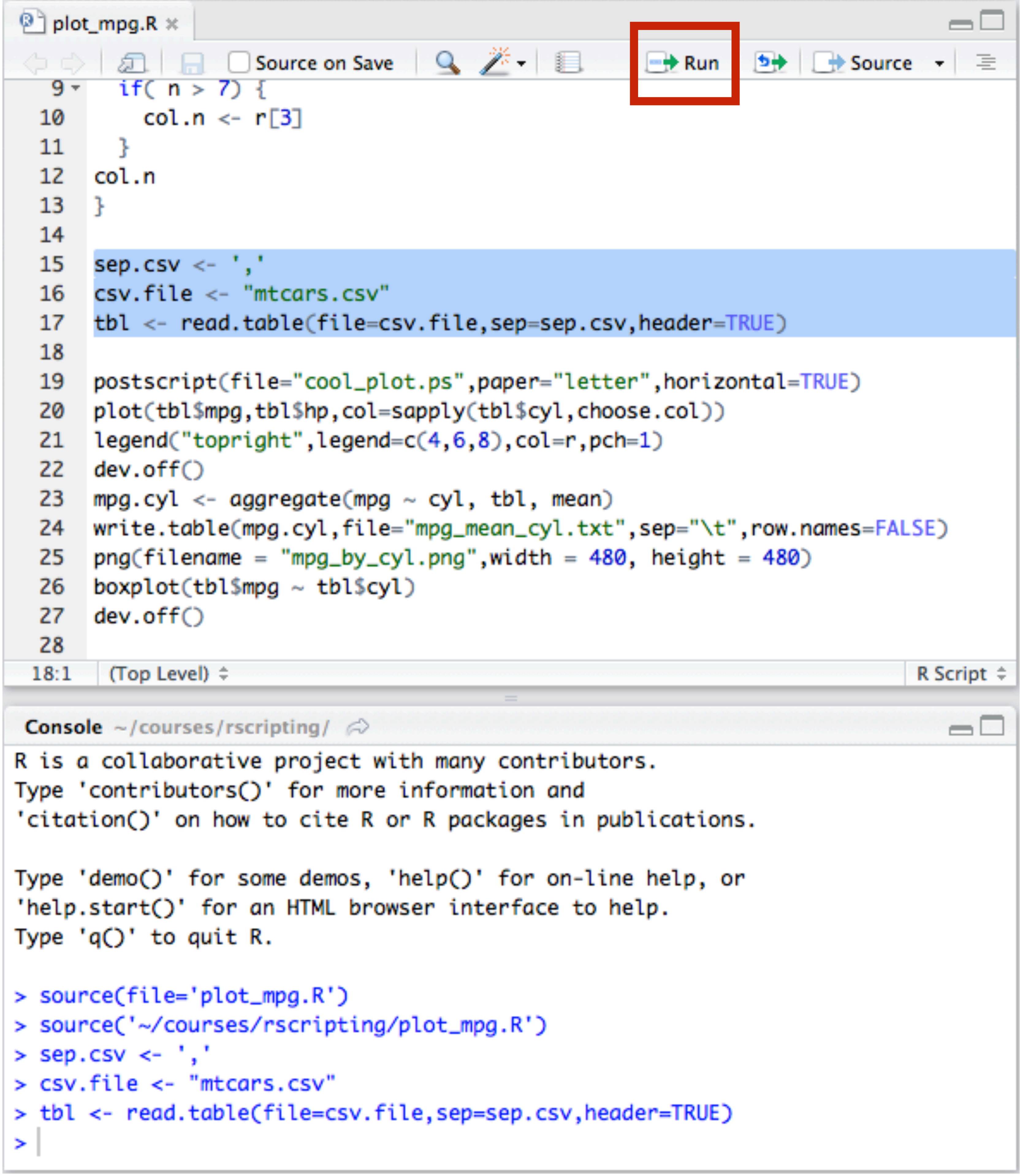
source



```
R version 3.3.0 (2016-05-03) -- "Supposedly Educational"
Copyright (C) 2016 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.

> source(file='plot_mpg.R')
```

Data	
mpg.cyl	3 obs. of 2 variables
tbl	32 obs. of 12 variables
Values	
csv.file	"mtcars.csv"
r	chr [1:3] "#FF0000FF" "#00FF00FF" "…
sep.csv	","
Functions	

Run



```r
  if( n > 7) {
    col.n <- r[3]
  }
col.n
}

sep.csv <- ','
csv.file <- "mtcars.csv"
tbl <- read.table(file=csv.file,sep=sep.csv,header=TRUE)

postscript(file="cool_plot.ps",paper="letter",horizontal=TRUE)
plot(tbl$mpg,tbl$hp,col=sapply(tbl$cyl,choose.col))
legend("topright",legend=c(4,6,8),col=r,pch=1)
dev.off()
mpg.cyl <- aggregate(mpg ~ cyl, tbl, mean)
write.table(mpg.cyl,file="mpg_mean_cyl.txt",sep="\t",row.names=FALSE)
png(filename = "mpg_by_cyl.png",width = 480, height = 480)
boxplot(tbl$mpg ~ tbl$cyl)
dev.off()

```

```
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.

> source(file='plot_mpg.R')
> source('~/courses/rscripting/plot_mpg.R')
> sep.csv <- ','
> csv.file <- "mtcars.csv"
> tbl <- read.table(file=csv.file,sep=sep.csv,header=TRUE)
> 
```

Rscript

```
[bcantarellt-osx:~/courses/rscripting] bcantarel% Rscript plot_mpg.R
null device
          1
null device
          1
[bcantarellt-osx:~/courses/rscripting] bcantarel% ls -ltr
total 136
-rw-r--r--@ 1 bcantarel  staff    944 Jan 21 10:23 statistical_tests.R
-rw-r--r--@ 1 bcantarel  staff   8169 Jan 21 10:23 rscriptingAnswers.Rmd
-rw-r--r--@ 1 bcantarel  staff   6612 Jan 21 10:23 rscripting.Rmd
-rw-r--r--@ 1 bcantarel  staff    900 Jan 21 10:23 correlation_plot.R
-rw-r--r--@ 1 bcantarel  staff    293 Jan 21 10:23 multi_plots.R
-rw-r--r--@ 1 bcantarel  staff   1700 Jan 21 10:23 mtcars.csv
-rw-r--r--  1 bcantarel  staff    141 Jan 23 21:11 mult_list.Rda
-rw-r--r--  1 bcantarel  staff    637 Jan 23 21:27 plot_mpg.R~
-rw-r--r--  1 bcantarel  staff    645 Jan 23 21:27 plot_mpg.R
-rw-r--r--  1 bcantarel  staff     57 Jan 23 21:27 mpg_mean_cyl.txt
-rw-r--r--  1 bcantarel  staff  12129 Jan 23 21:27 mpg_by_cyl.png
-rw-r--r--  1 bcantarel  staff   5548 Jan 23 21:27 cool_plot.ps
```

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

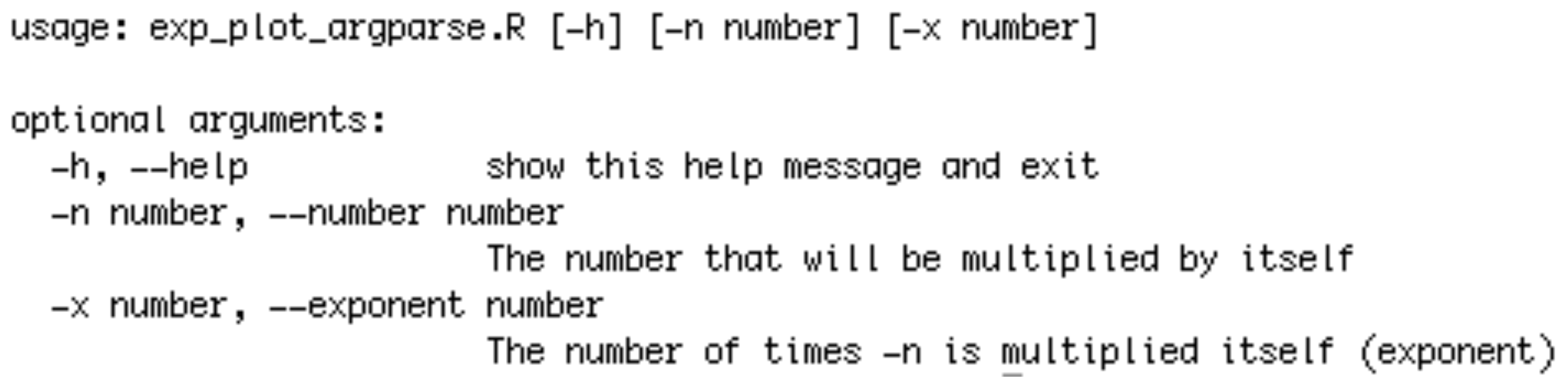
commandArgs

```
args<-commandArgs(TRUE)
 # Get variables from command line
num1 <- as.numeric(args[1])
num2 <- as.numeric(args[2])
square <- function(x,n=seq(1, num2, by
= 1))
{
x^n
}
x <- c(1:num2)
y <- square(num1)
postscript(file=“exp_plot.ps”,paper=“le
tter”,horizontal=TRUE)
plot(x,y,ylab=paste(num1,”^x",sep=""))
dev.off()
```

On the command-line:

```
Rscript exp_plot.R 2 10
```

argparse



```
usage: exp_plot_argparse.R [-h] [-n number] [-x number]

optional arguments:
  -h, --help            show this help message and exit
  -n number, --number number
                        The number that will be multiplied by itself
  -x number, --exponent number
                        The number of times -n is multiplied itself (exponent)
```

```
parser <- ArgumentParser()
# specify our desired options
# by default ArgumentParser will add an help option
parser$add_argument("-n", "--number", type="integer",
default=2,
 help=“The number that will be multiplied by itself",
 metavar="number")
parser$add_argument("-x", "--exponent", type="integer",
default=10,
 help=“The number of times -n is multiplied itself
(exponent“,
 metavar=“number")
```

argparse

```
library(argparse)
parser <- ArgumentParser()

parser$add_argument("-n", "--number",
type="integer", default=2,
  help="The number that will be multiplied by
itself",
  metavar="number")
parser$add_argument("-x", "--exponent",
type="integer", default=10,
  help="The number of times -n is multiplied itself
(exponent)",
  metavar="number")
args <- parser$parse_args()
num1 <- args$number
num2 <- args$exponent
square <- function(x,n=seq(1, num2, by = 1))
{
  x^n
}
x <- c(1:num2)
y <- square(num1)
postscript(file=“exp_plot2.ps",paper="letter",
horizontal=TRUE)
plot(x,y,ylab=paste(num1,"^x",sep=""))
dev.off()
```

On the command-line:

```
Rscript exp_plot.R -n 2 -x 10
```

10-Minute Break

Workshop Starts in 10 Minutes