Lecture 2 - Extending R

KM Purcell

September 23, 2015

Review

Analytics Toolbox

A Deepe R dive

Last class

- Built-in functions
- Assingment operator <-
- object naming conventions
- vectors
- basic arithmatic operations
- statistical test output (*from a fxn*)
- simple plotting functions and potential

"Without data you're just another person with an opinion"

- W. Edwards Deming

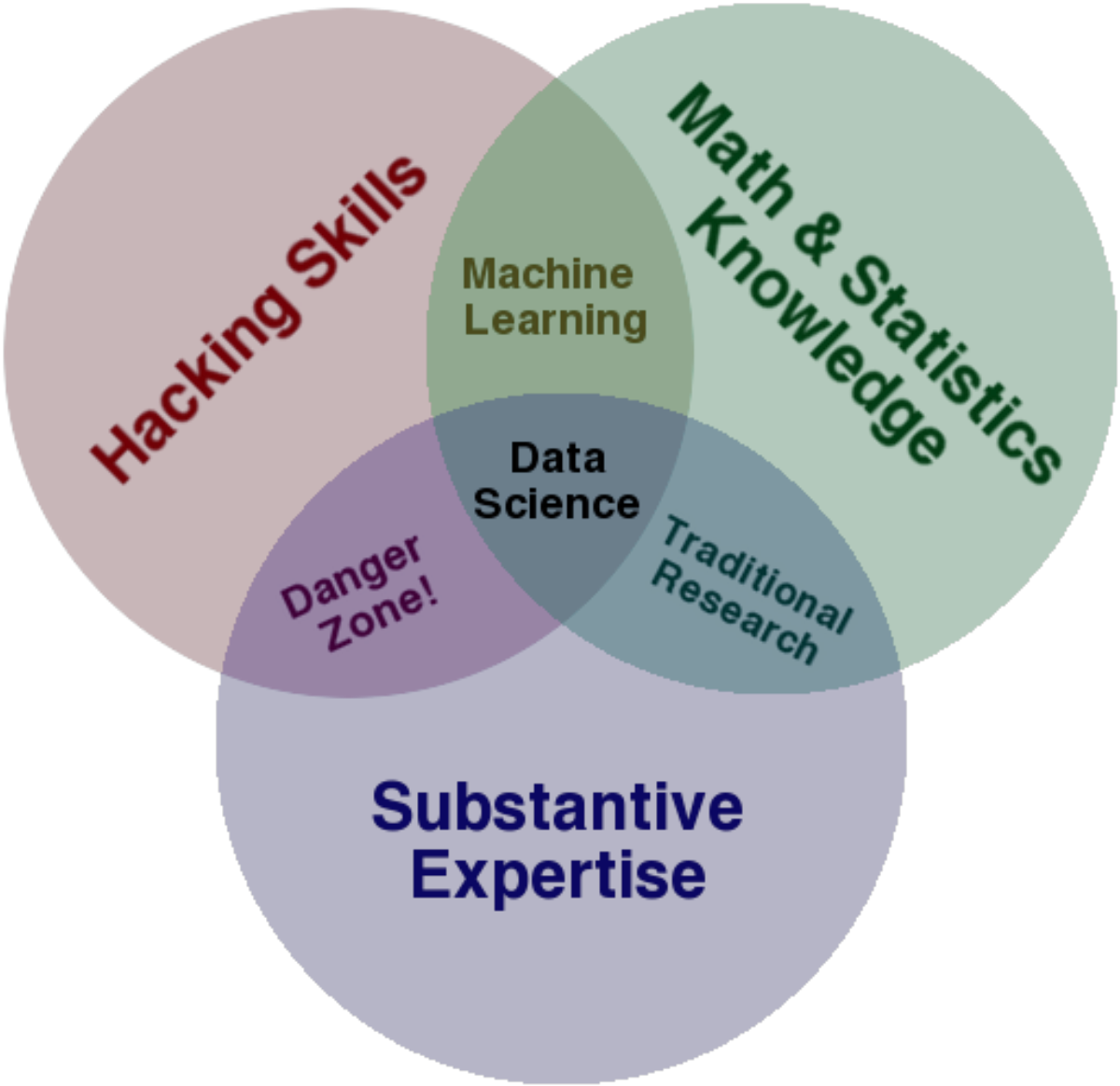
Other business

- the **syllabus** is posted to Moodle
- First class readings are being assigned

R resources

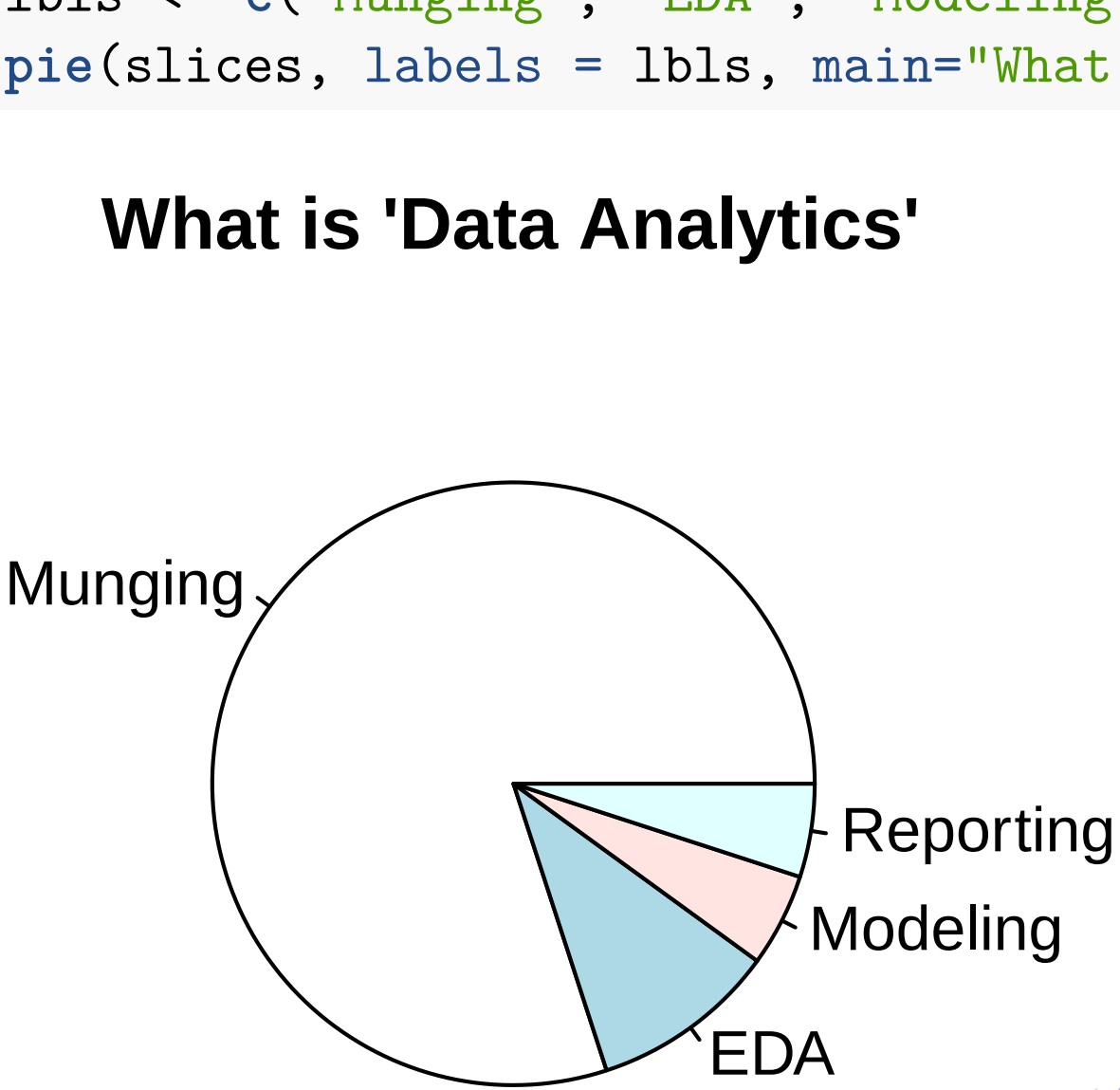
- book list on CRAN (`https://www.r-project.org/doc/bib/R-books.html`)

Data Science (Again!)



Data Analytics

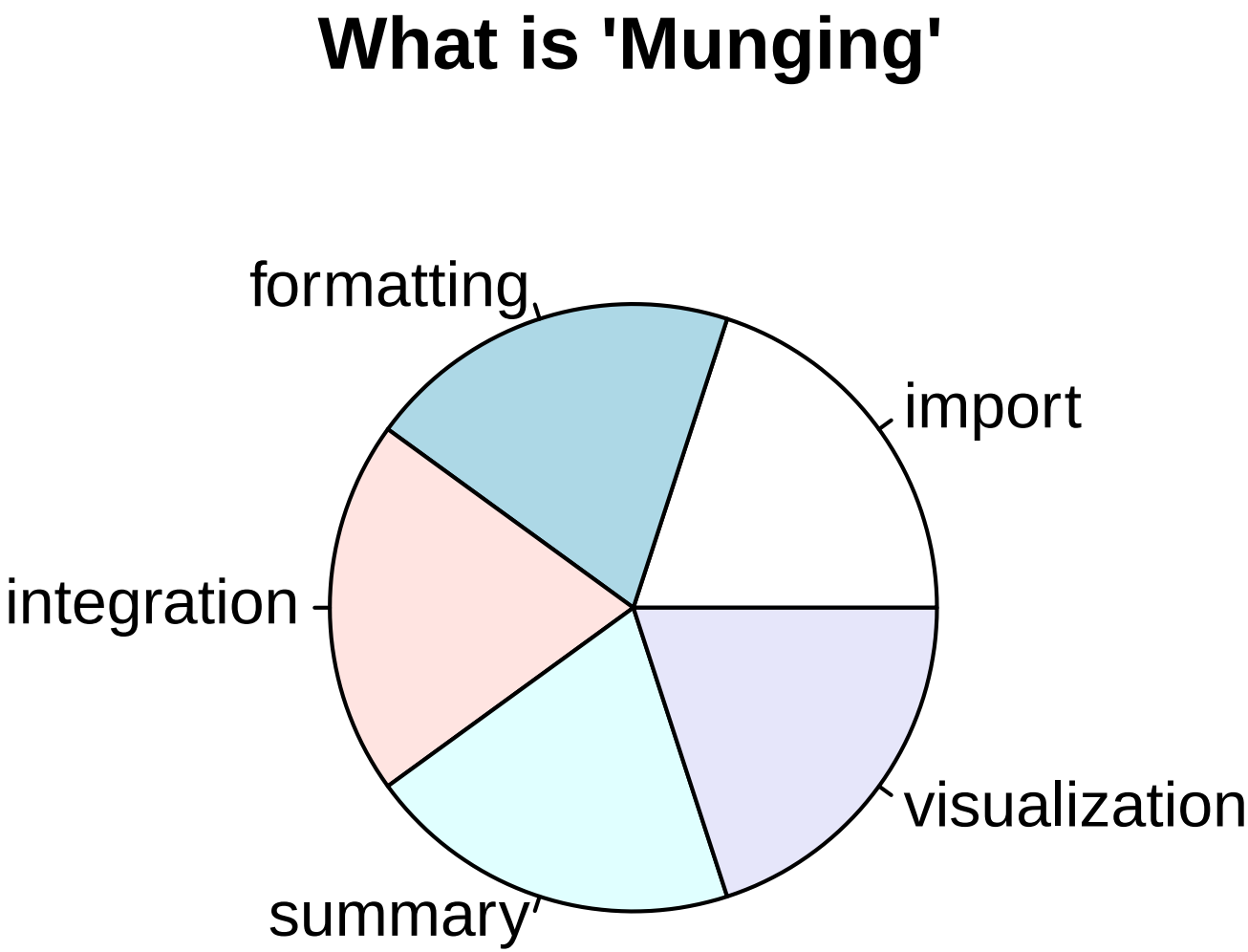
```
slices <- c(80, 10,5,5)
lbls <- c("Munging", "EDA", "Modeling", "Reporting")
pie(slices, labels = lbls, main="What is 'Data Analytics'")
```



Data munging

- The process by which an analyist collects, organizes and maps data for downstream analysis.
- Definitions vary, some include EDA in munging.
- Overall, I deliniate **munging** as the tasks that must be undertake in order to work with the relevant data.
- Important goal is to automate to the greatest extent possible (crucial for all phases of analytical work)
- Hence, `R` or other programitic approaches

Data munging



Munging challenges

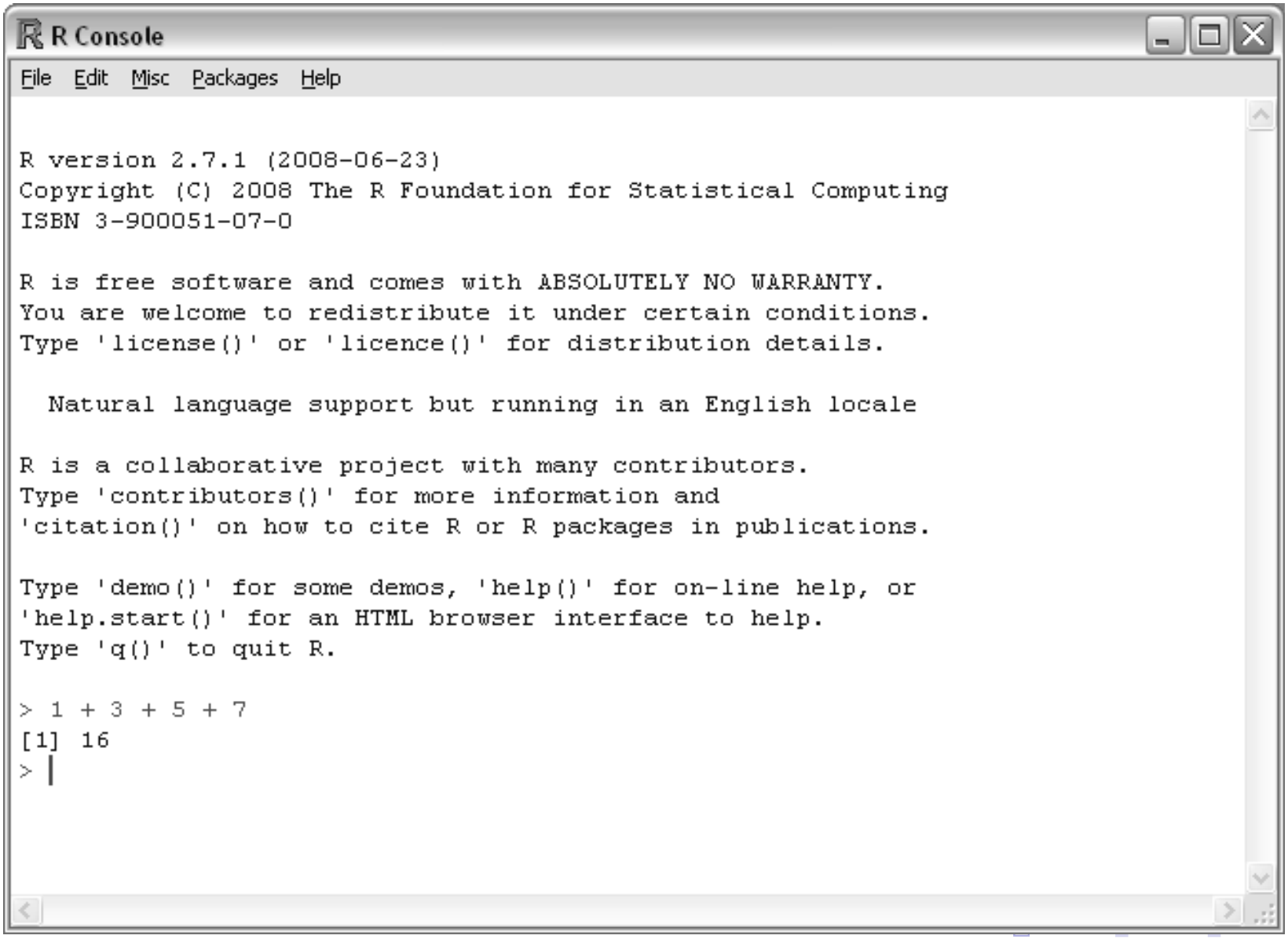
- **import** challenges of importing data from disparate databases and legacy formats
- **formatting** converting data to appropriate formats, converting vectors to be recognized as dates, or categorical variables, or spatial coordinates, etc.
- **integration** often data will be stored/collected differently, ie. different spatio-temporal resolutions that require integration.
- **summary** what is missing? Is there a data dictionary? How was it collected?
- **visualiation** does the data make sense? Often require visual inspections. Big challenge as data sets explode in size!

The R advantage

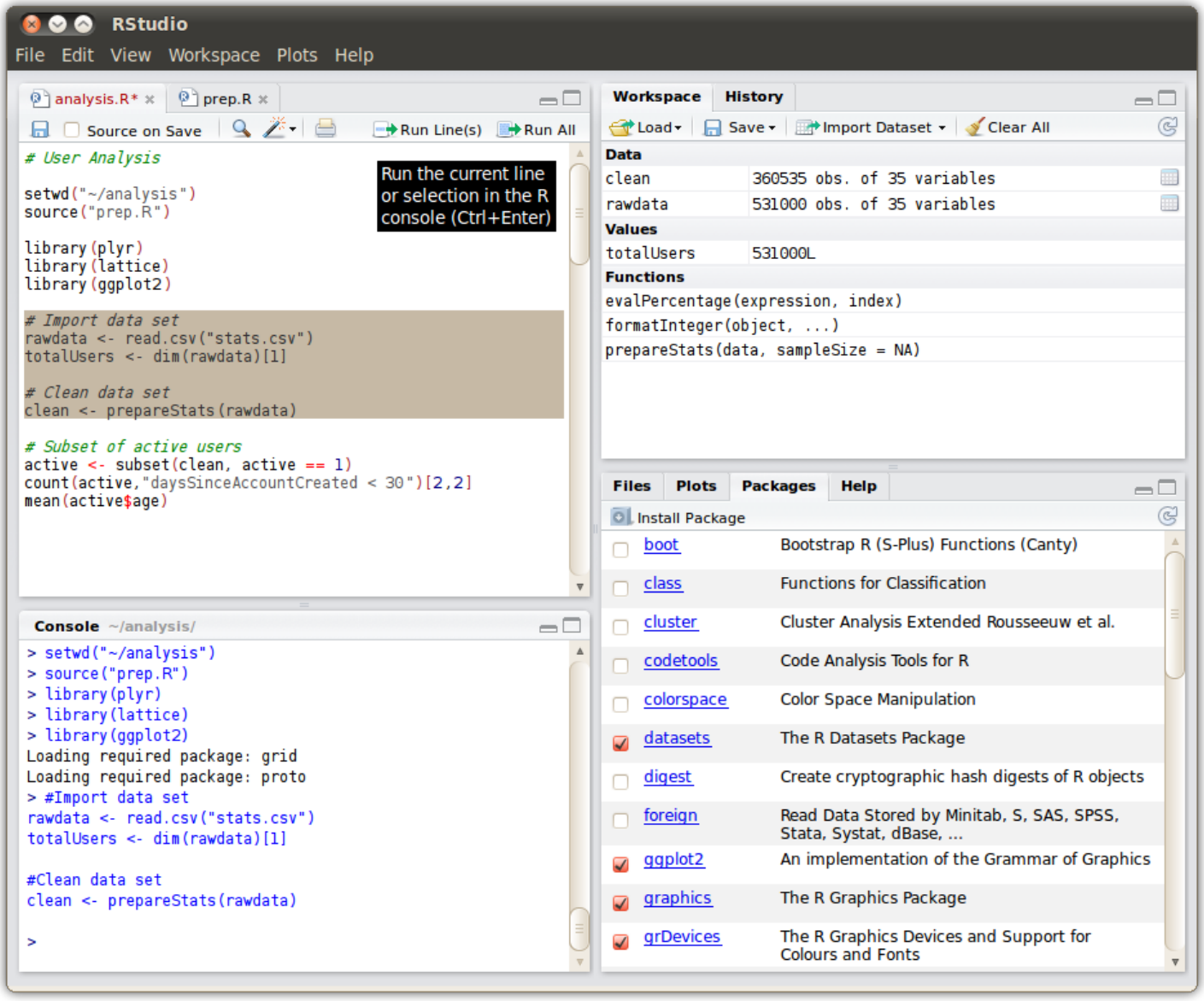
- R language is a powerful tool for data management.
- It has ~6,712 packages adding to base functionality.
- A large community contributing to the code base and extending capabilities.

Interacting with R

- via the console



- via a graphical user interface (GUI)



Interactions with R

- R is built for both *interactive programming* and *batch processing*
- Traditional analytics investigation occurs using and interactive approach

```
X <- 4; Y <- 6
X + Y
```

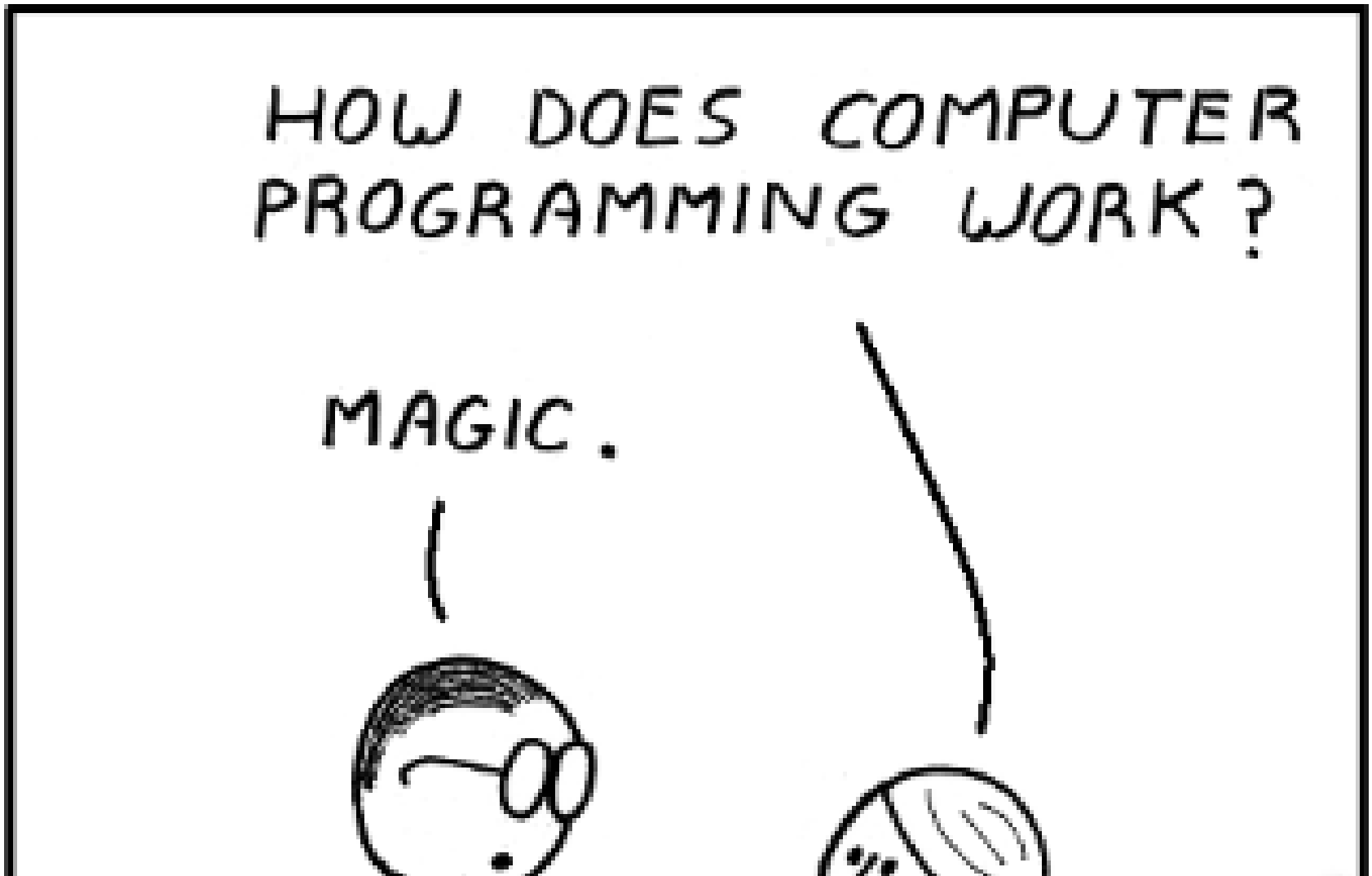
```
## [1] 10
```

- As opposed to batch operations in which to write the script save it as a .R file and then tell the CPU to execute the file. Outputs are observed after the entire analysis is executed.

```
> r xy.R
```

Interactive computation

- a faster developmental model for analytical operations
- common paradigm in scientific programming
- Optional: Semantics of interactive computation



Scientific computing

- core difference is the sequence of input & output cycles
- traditional (batch) development: `INPUT -> COMPUTE -> OUTPUT`
- Scientific computing development: `INPUT -> VIEW -> OUTPUT -> SORT -> OUTPUT etc.`
- **Reading**: Wilson et al. 2014. "Best Practices for Scientific Computing." PLoS Biology 12 (1): e1001745. doi:10.1371/journal.pbio.1001745.

Who to thank for R

```
citation()
```

```
##
## To cite R in publications use:
##
##   R Core Team (2015). R: A language and environment for
##   statistical computing. R Foundation for Statistical Co
##   Vienna, Austria. URL http://www.R-project.org/.
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {R: A Language and Environment for Statistic
##     author = {{R Core Team}},
##     organization = {R Foundation for Statistical Comput
##     address = {Vienna, Austria},
##     year = {2015},
##     url = {http://www.R-project.org/},
```

Expressions

```
# This is a comment,
# all comments in R start with a # symbol

x <- 78          # This is a complete expression
                 # no printed results
y <-             # Incomplete expression
x                # auto-printing
print(x)         # explicit printing
```

```
## [1] 78
```

Expressions

- vector indices

```
x <- 1:40      # Creats a vectors of values from 1 to 40
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
## [24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

- index printed in [brackets]

R has five classes of objects

- numeric
- character
- logical
- integer
- complex

```r
x <- c(1, 2, 3); class(x)
```

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

```r
x <- c("a", "b", "c"); class(x)
```

```
## [1] "character"
```

```r
x <- c(TRUE, TRUE, FALSE); class(x)
```

```
## [1] "logical"
```

```r
x <- 1:30; class(x)
```

```
## [1] "integer"
```

```r
x <- c(1+4i); class(x)
```

```
## [1] "complex"
```

Data structures

- R offers a number of structures to hold data including: vectors, matrices, array, lists, data frames, tables.
- Each structure is defined by dimensional and compositional characterisitics

```
##   Dimension   Homogeneous Heterogeneous
## 1        1D Atomic vector          list
## 2        2D        Matrix    data frame
## 3        nD         array
```

Vectors

- we introduced vectors last class
- vectors are created using the construct c() function

```
x <- c(1,2,3,4,5)  # create a simple vector
str(x)
```

```
##  num [1:5] 1 2 3 4 5
```

```
x <- vector("numeric", length=5)  #create an empty vector
x
```

```
## [1] 0 0 0 0 0
```

Indexing Vectors

- often we need data values from *vectors*

```
v <- rnorm(10, mean=290, sd=2)
v
```

```
##  [1] 290.1711 291.8910 291.7945 291.0570 291.0827 290.91
##  [8] 288.8626 287.6573 289.7331
```

```
v[5]
```

```
## [1] 291.0827
```

```
# Do it again
v2 <- rnorm(10, mean=290, sd=2)
v2
```

```
##  [1] 290.1107 289.5700 293.7135 293.4494 288.3844 291.01
##  [8] 291.9973 285.5148 287.4342
```

Seeds

- making simulations reproducible with `set.seed()`

```
set.seed(4)
v <- rnorm(10, mean=290, sd=2)
v[5]
```

```
## [1] 293.2712
```

```
# Do it again
set.seed(4)
v2 <- rnorm(10, mean=290, sd=2)
v2[5]
```

```
## [1] 293.2712
```

- all simulation or RNG work requires seeds to be reproducible.

Matrices

- *matrices* are vectors with dimensionality.

```
m <- matrix(nrow = 3, ncol = 5)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   NA   NA   NA   NA   NA
## [2,]   NA   NA   NA   NA   NA
## [3,]   NA   NA   NA   NA   NA
```

```
# Examine the matrix
attributes(m)
```

```
## $dim
## [1] 3 5
```

How are matrices built?

- matricies in R are constructed *column-wise*

```
m <- matrix(1:15, nrow = 3, ncol = 5)
m
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7   10   13
## [2,]    2    5    8   11   14
## [3,]    3    6    9   12   15
```

- so when populating a matrix values are placed in cell (1,1) or the upper-left *first* and run down columns.

Populating matrices

- a matrix can be populated from a *vector*

```
m <- 1:15
dim(m) <- c(3,5)
```

- in R matrix references are usually (R, C) references.

```
c(4,5) #specifying a 4 row, 5 column matrix
m[2,4] # Identifying a position in matrix(m) at row 2,
```

Matricies by binding

- matrices can also be constructed by binding data using the `rbind()` and the `cbind()` functions.

```
a <- 1:5
b <- 6:10



m1 <- cbind(a,b)
m1
```

```
##      a  b
## [1,] 1  6
## [2,] 2  7
## [3,] 3  8
## [4,] 4  9
## [5,] 5 10
```

```
m2 <- rbind(a,b)
m2
```

Lists in R

- *lists* are a powerful type of vector in R
- *list* elements may contain different classes
- *lists* can be constructed with `list()` or `c()`

```
x <- list(1:3, c("a", "b", "c"), c(TRUE, FALSE, TRUE), c(3.
str(x)
```

```
## List of 4
##  $ : int [1:3] 1 2 3
##  $ : chr [1:3] "a" "b" "c"
##  $ : logi [1:3] TRUE FALSE TRUE
##  $ : num [1:3] 3.14 2.96 7.9
```

Lists in R

- lists can be *recursive* meaning they contain additional lists
- common data structure when working with more complex analyses

```
x <- list(list(list(list())))
str(x)
```

```
## List of 1
##  $ :List of 1
##   ..$ :List of 1
##   .. ..$ : list()
```

Lists in R

- elements of lists can be referenced specifically

```
x <- list(integers = 1:3,                                    # code co
          alphabet = c("a", "b", "c"),                       # ending
          logicals = c(TRUE, FALSE, TRUE),
          numbers = c(3.14, 2.96, 7.90))
x$logicals
```

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

Data Frame

- most data is stored as data frames
- data frames are structues for storing data tables
- essentially a *data frame* is a list of vectors of equal length

Data Frame

- common example is `mtcars` data set in `R`

```
head(mtcars)  # head fxn shows the top 6 values
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1
```

Viewing data

- up to now we are inputing the data
- that data has been small easily viewable
- Sometimes we need to view a piece of the data
- the `head()` and `tail()` fxns

Viewing data

```
tail(mtcars)
```

```
##                  mpg cyl  disp  hp drat    wt qsec vs am
## Porsche 914-2   26.0   4 120.3  91 4.43 2.140 16.7  0  1
## Lotus Europa    30.4   4  95.1 113 3.77 1.513 16.9  1  1
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1
## Ferrari Dino    19.7   6 145.0 175 3.62 2.770 15.5  0  1
## Maserati Bora   15.0   8 301.0 335 3.54 3.570 14.6  0  1
## Volvo 142E      21.4   4 121.0 109 4.11 2.780 18.6  1  1
```

- `head()` & `tail()` can be used on vectors, matrices, data frames, or functions

Data Import

- once we go behind a small # of values *data import* is essential
- to work with data from outside `R` the information must first be imported into the `R` environment
- all base function are associated with `read.table()`
- read more: `https://stat.ethz.ch/R-manual/R-devel/library/utils/html/read.table.html`

Read Table

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", numerals = c("allow.loss", "warn.loss
           row.names, col.names, as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.line
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text, s
```

Read Table

- `file` is name of the file containing the data

```
read.table(file="mtcars.csv")
read.table(file="C://Users/Desktop//File name")
```

read.table() Example

```
# import the data to an object
dat <- read.table(file='/Users/kpurcell/Desktop/mtcars.csv
dim(dat)   # displays the dimensions of an object
```

```
## [1] 33  2
```

```
# display a summary of object
summary(dat)
```

```
##                      V1
##                       : 1
##  AMC Javelin          : 1
##  Cadillac Fleetwood: 1
##  Camaro Z28           : 1
##  Chrysler Imperial : 1
##  Datsun 710           : 1
##  (Other)              :27
##
```

read.table() Example

```
dat <- read.table(file='/Users/kpurcell/Desktop/mtcars.csv
                  header=TRUE,
                  sep=",")
dim(dat)   # displays the dimensions of an object
```

```
## [1] 32 12
```

```
summary(dat)
```

```
##                    X                 mpg              cyl
##  AMC Javelin       : 1   Min.   :10.40   Min.   :4.000
##  Cadillac Fleetwood: 1   1st Qu.:15.43   1st Qu.:4.000
##  Camaro Z28        : 1   Median :19.20   Median :6.000
##  Chrysler Imperial : 1   Mean   :20.09   Mean   :6.188
##  Datsun 710        : 1   3rd Qu.:22.80   3rd Qu.:8.000
##  Dodge Challenger  : 1   Max.   :33.90   Max.   :8.000
##  (Other)           :26
##        hp             drat             wt             qs
```

`read.table()` options

- lots of options to fit the input file you are working with read.table help

`read.csv()` function

- is a modified version of `read.table()`
- it has specific defaults set

```
read.csv(file, header = TRUE, sep = ",",
      quote = "\"", dec = ".", fill = TRUE,
      comment.char = "", ...)
```

`read.csv2()` function

- slightly different version

```
read.csv2(file, header = TRUE, sep = ";",
      quote = "\"", dec = ",", fill = TRUE,
      comment.char = "", ...)
```

Delimited files

- .csv files are a common type of delimited file
- `read.delim()` 1 & 2 are aimed at other delimited files
- tab (`\t`) or space (`\s`) delimited specifically

So far...

- the `read.XXX()` family of functions can be used on a number of standard text based input files that are common for data entry

Additional packages for Data Import

```
library(xlsx)      # Read Excel spreadsheets.
library(RCurl)     # Scraping web
library(foreign)   # Importing from other software
library(openxlsx) # Importing from OSS
library(readxl)    # The definitive Excel reader.
library(readr)     # Wickham package for data import
```

- We will touch on more throughout the semester

`readr()` Package

- The `readr()` package is an new package
- it builds upon `R` functionality

Read 2 data frame

```
library(readr)

# Read a csv file into a data frame
dat <- read_csv("/Users/kpurcell/Desktop/mtcars.csv")
head(dat)
```

```
##                [EMPTY]  mpg cyl disp  hp drat    wt  qsec v
## 1            Mazda RX4 21.0   6  160 110 3.90 2.620 16.46
## 2        Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02
## 3           Datsun 710 22.8   4  108  93 3.85 2.320 18.61
## 4       Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44
## 5 Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02
## 6              Valiant 18.1   6  225 105 2.76 3.460 20.22
```

Read 2 vector

```
# Read lines into a vector
dat <- read_lines("/Users/kpurcell/Desktop/mtcars.csv")
head(dat)
```

```
## [1] "\"\",\"mpg\",\"cyl\",\"disp\",\"hp\",\"drat\",\"wt\
## [2] "\"Mazda RX4\",21,6,160,110,3.9,2.62,16.46,0,1,4,4"
## [3] "\"Mazda RX4 Wag\",21,6,160,110,3.9,2.875,17.02,0,1
## [4] "\"Datsun 710\",22.8,4,108,93,3.85,2.32,18.61,1,1,4
## [5] "\"Hornet 4 Drive\",21.4,6,258,110,3.08,3.215,19.44
## [6] "\"Hornet Sportabout\",18.7,8,360,175,3.15,3.44,17.0
```

Read 2 string

```
# Read whole file into a single string
dat <- read_file("/Users/kpurcell/Desktop/mtcars.csv")
head(dat)
```

```
## [1] "\"\",\"mpg\",\"cyl\",\"disp\",\"hp\",\"drat\",\"wt\
```

mtcars

```
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 1
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```