Introduction to R

A Tutorial Helps

If you have never used R before, please take some time to get familiar with the language and its environment following this free online tutorial, Introduction to R (https://www.datacamp.com/courses/free-introduction-to-r).

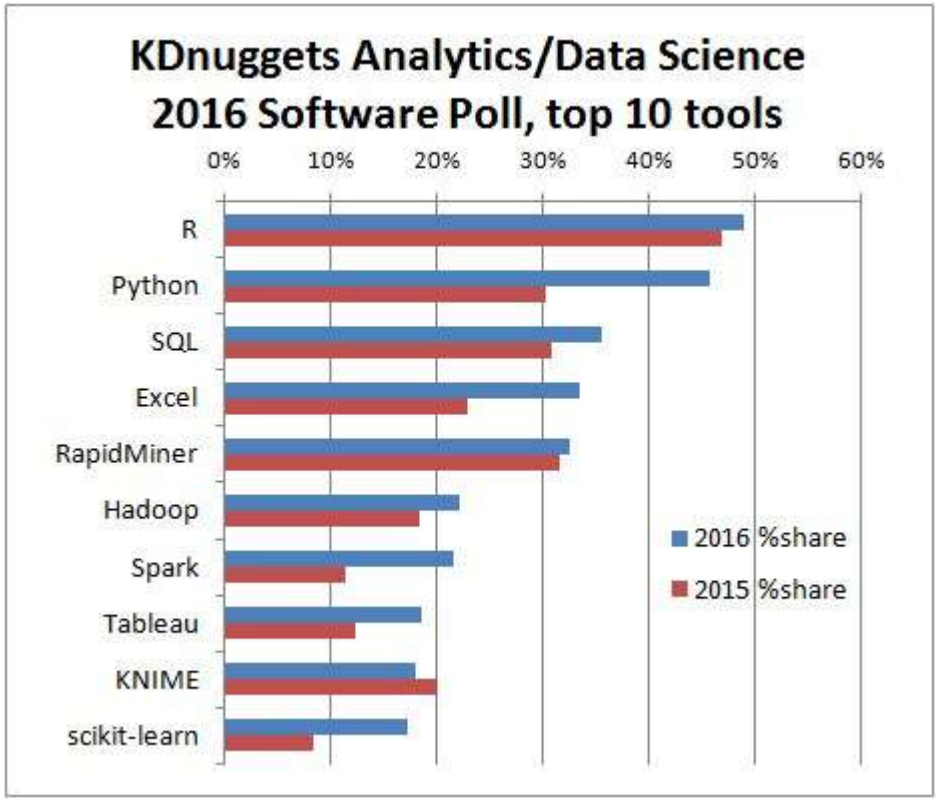
You can download R from here: https://www.r-project.org/ . RStudio is a great open source editor for R. You can download it from here: https://www.rstudio.com/products/rstudio/download2/ .

- R is a language and environment for statistical computing and graphics, providing a wide variety of statistical, graphical, and data mining techniques
- Runs on Mac OS, Windows, Linux and Unix platforms
- Open source, licensed under GPL - http://www.r-project.org/
- Command line based, but highly extensible
- Lots of packages contributed by the open community, available at CRAN repositories
- Get the latest version from http://cran.r-project.org/bin/windows/base/

No R&D budget can compete with nearly ALL the statistics departments of the world and their professors working for free on this project.

A Ohri, author of *R for Business Analytics*

R as a Language for Data Analytics



- The most popular in KDnuggets' poll

http://www.kdnuggets.com/2016/06/r-python-top-analytics-data-mining-data-science-software.html

Main Features

- Extensive capabilities to interact with and pull data from databases (Oracle, MySQL, PostGresSQL, Hadoop-based data, etc)
- Advanced data visualisation through packages like *ggplot2*
- A vast array of statistical and data mining packages covering standard regression, decision trees, association rules, cluster analysis, machine learning, neural networks, etc
- GUI (Rattle) available for data miners using R
- Interfaces from almost all other analytical software including SAS, SPSS, JMP, Oracle Data Mining, RapidMiner, Excel, etc. (vendors viewing R as a complementary language)
- Flexible option for enterprise users from commercial vendors like Revolution Analytics
- Lots and lots of tutorials, codes, books available on web

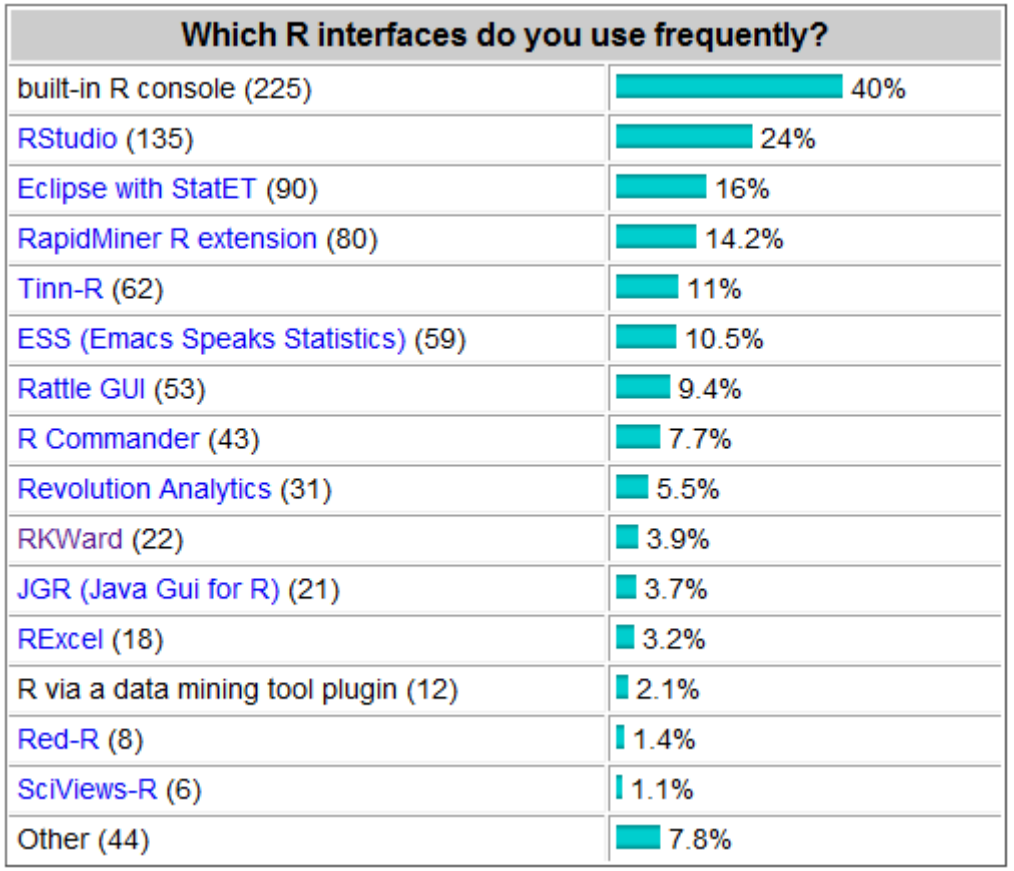
GUIs

- Primarily command-line based
- For Windows users of Basic R, there's a simple GUI (to load package, install package, and set CRAN mirror for downloading packages)
- Other GUIs are available to make the use of R more convenient, such as
  - R Commander: more for statistics, plotting, time series
  - Sciviews-K: flexible GUI, can be used to create other GUIS
  - PKWard: comprehensive GUI with lots of details
  - Red-R: workflow style
  - R Analytic Flow: workflow style
  - Rattle: for data mining
  - PMG: simple interface
  - JGR/Deducer: more for data visualization
  - Grapher: simple graphing

IDEs

- Code editors or Integrated Development Environment (IDE) can make writing R codes easier for developers
  - Enhanced readability with syntax coloring
  - Automatic syntax error checking,
  - Auto code completion
  - Debugging facilities
- Examples
  - RStudio: most popular IDE for R, with code completion, syntax coloring, support for Latex
  - Notepad++: enhanced code editor for a variety of languages
  - TinnR: basic and easy-to-use code editor
  - Eclipse with StatET: R plugin for Eclipse, with support for Latex
  - Other code editors: Gvim, Highlight, etc.

Poll on R Interfaces



Which R interfaces do you use frequently?	
built-in R console (225)	40%
RStudio (135)	24%
Eclipse with StatET (90)	16%
RapidMiner R extension (80)	14.2%
Tinn-R (62)	11%
ESS (Emacs Speaks Statistics) (59)	10.5%
Rattle GUI (53)	9.4%
R Commander (43)	7.7%
Revolution Analytics (31)	5.5%
RKWard (22)	3.9%
JGR (Java Gui for R) (21)	3.7%
RExcel (18)	3.2%
R via a data mining tool plugin (12)	2.1%
Red-R (8)	1.4%
SciViews-R (6)	1.1%
Other (44)	7.8%

Google's R Style Guide

Summary: R Style Rules

1. File Names: end in `.R`
2. Identifiers: `variable.name` (or `variableName`), `FunctionName`, `kConstantName`
3. Line Length: maximum 80 characters
4. Indentation: two spaces, no tabs
5. Spacing
6. Curly Braces: first on same line, last on own line
7. else: Surround else with braces
8. Assignment: use `<-`, not `=`
9. Semicolons: don't use them
10. General Layout and Ordering
11. Commenting Guidelines: all comments begin with `#` followed by a space; inline comments need two spaces before the `#`
12. Function Definitions and Calls
13. Function Documentation
14. Example Function
15. TODO Style: `TODO(username)`

https://google.github.io/styleguide/Rguide.xml

Packages

- Currently >9000 packages from CRAN package repository, almost all free
- CRAN task views by subject areas

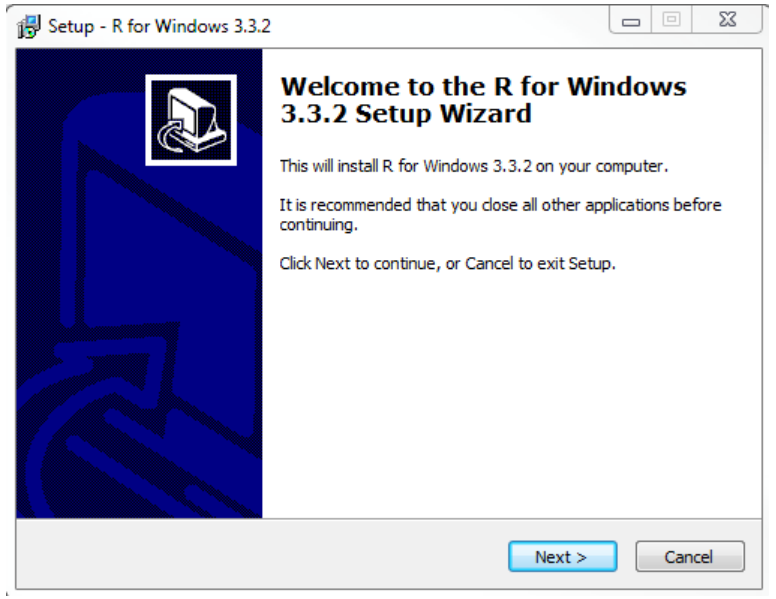
Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Computational Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
Finance	Empirical Finance
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis
MetaAnalysis	Meta-Analysis
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
NumericalMathematics	Numerical Mathematics
OfficialStatistics	Official Statistics & Survey Methodology
Optimization	Optimization and Mathematical Programming
Pharmacokinetics	Analysis of Pharmacokinetic Data
Phylogenetics	Phylogenetics, Especially Comparative Methods
Psychometrics	Psychometric Models and Methods
ReproducibleResearch	Reproducible Research
Robust	Robust Statistical Methods
SocialSciences	Statistics for the Social Sciences
Spatial	Analysis of Spatial Data
SpatioTemporal	Handling and Analyzing Spatio-Temporal Data
Survival	Survival Analysis
TimeSeries	Time Series Analysis
WebTechnologies	Web Technologies and Services
gR	gRaphical Models in R

Getting Started

- The latest R basic can be downloaded from CRAN. Current version is 3.6.x
- Installation
  - Windows
    - Double click on the downloaded installer file, and follow the standard installation steps
    - Choose 32-bit or 64-bit based on your system's OS
  - Linux

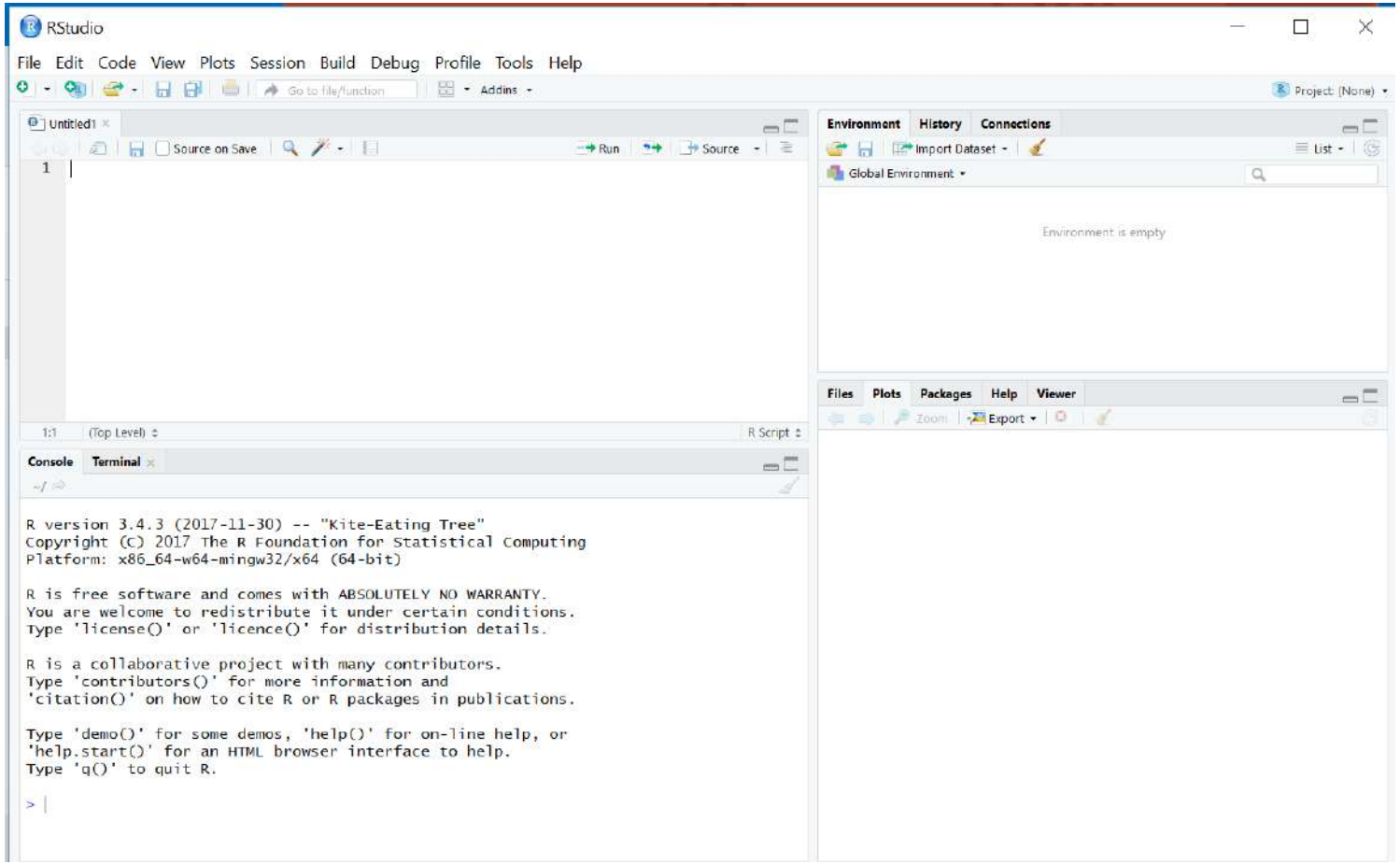
    sudo apt-get update

    sudo apt-get install r-base



RStudio Gui

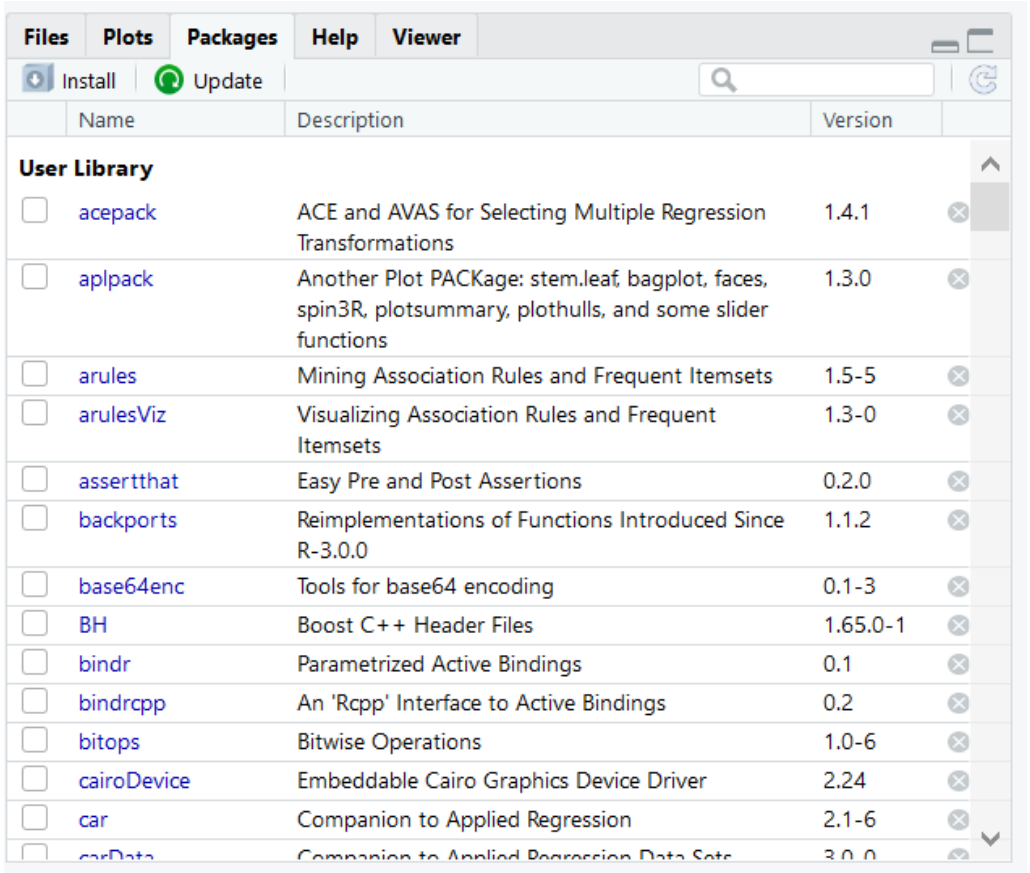
- Console, Script Editor, Graphics windows



Install a package

- Use command

  > install.packages("rattle")



Load a package

- Use command

  > library(rattle)

- *Every time R is started, you need to reload the packages you want to use*.
- To update installed packages later on

  > update.packages()

Documentations on Packages

- Many packages have vignettes, prepared documentations (often in pdf)
- To see available vignettes

  vignette()
- To bring up the vignette about a package (it's fine to use single or double quote)

  vignette('rattle') or *library(help="rattle")*

  The pdf document will be displayed in a viewer like Adobe Acrobat Reader.

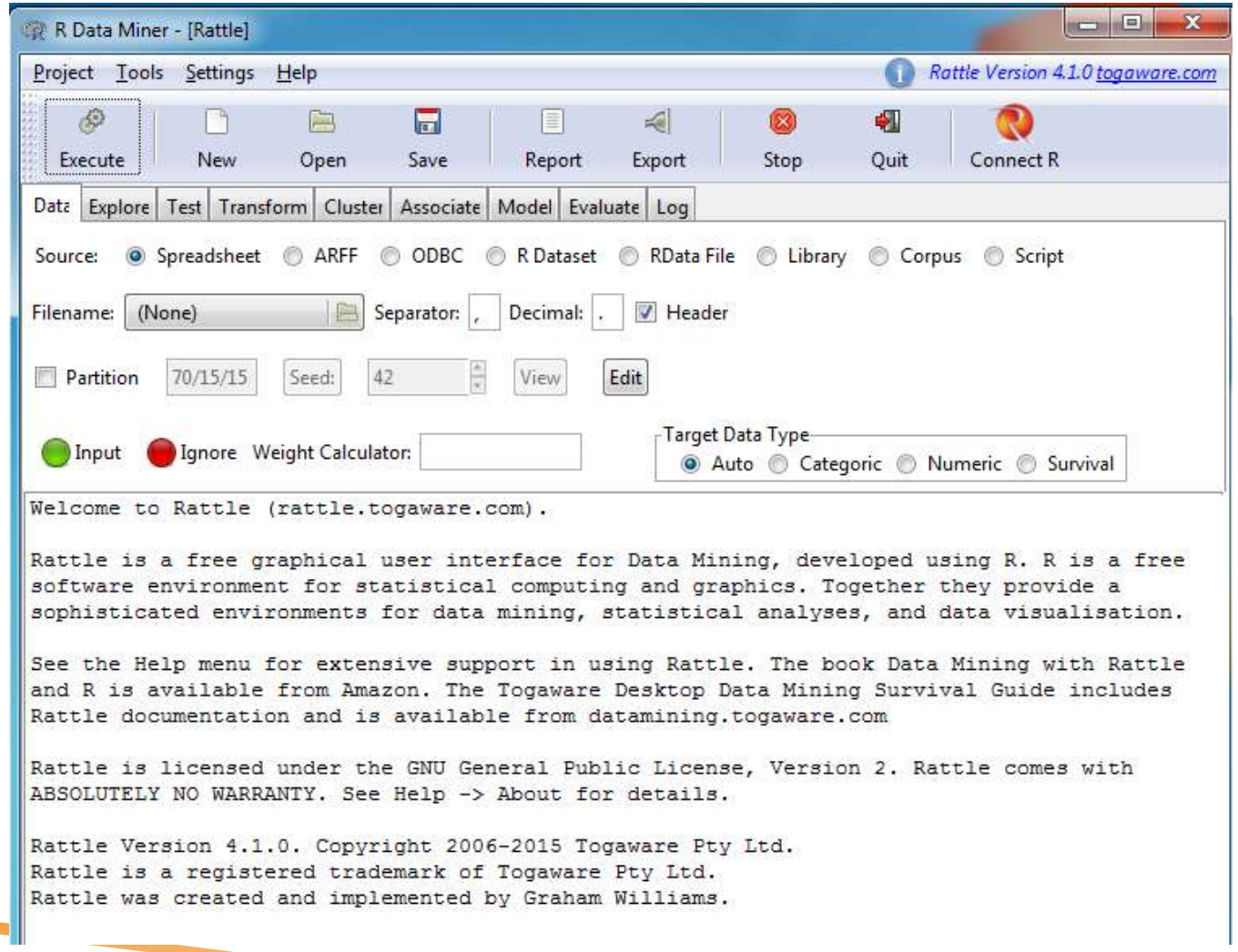
Rattle GUI

- **R A**nalytical **T**ool **T**o **L**earn **E**asily
- Popular GUI for data mining using R, good for R beginners
- Separate tabs for data import, summary, visualization, model building, clustering, association, and evaluation
- Log for R code is auto-generated → good for learning the codes!
- Can fall back to R when functions in Rattle are not sufficient
- An R package itself, therefore, to start Rattle:

  library(rattle)

  rattle()
- You'll be prompted to install many other packages that *rattle* is dependent on. Just agree (click 'yes') to install them all.

Rattle Interface



Rattle DM Tabs

- Tab-based interface
- Order of the tabs from left to right mimicking the typical data mining process
  - **Data** : to load/import data
    - supporting spreadsheets, ARFF, ODBC, R Dataset, R data file, datasets from R libraries, corpus, etc.
  - **Explore**: for data exploration
    - statistical summary, distribution, correlation analysis, principal components, interactive graphs
  - **Test**: for statistical testing
    - Two-sample tests (T-test, F-test, etc.), paired two-sample tests
  - **Transform**: data transformation
    - Scaling, imputation, recoding, cleanup

Rattle DM Tabs

- **Data mining tabs** (continued)
  - **Cluster**
    - K Means, Entropy-weighted K Means, hierarchical clustering, etc.
  - **Associate**
    - Apriori
  - **Model**
    - Decision tree, random forest, boosting, SVM, linear regression, neural network, survival regression
  - **Evaluate**
    - Confusion matrix, lift chart, ROC curve, cost curve, precision/recall, etc.
- **Log** tab – auto generated R codes

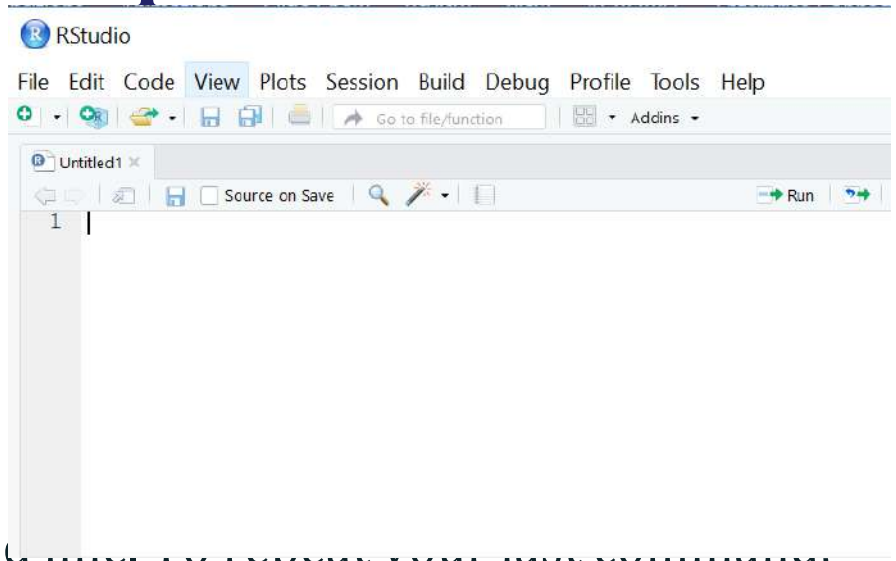
Some Useful Sites

- **Quick R** - lots of samples and short explanations
  - http://www.statmethods.net/
- **An Introduction to R -** accurate, up-to-date information from the R core Team
  - http://cran.r-project.org/doc/manuals/R-intro.pdf
- **Cookbook for R –** solutions to common tasks and problems in data analysis
  - http://www.cookbook-r.com/
- **R-bloggers** - for interesting and latest posts about R
  - http://www.r-bloggers.com
- **OnePageR** – A Survival Guide to Data Science with R
  - http://togaware.com/onepager/
- Find answers to common questions at
  - http://stackoverflow.com
  - http://stats.stackexchange.com/

And a lot more!

Basic R Programming

Useful Tips



- R is **case sensitive**
- **Use "/" in file pathname**

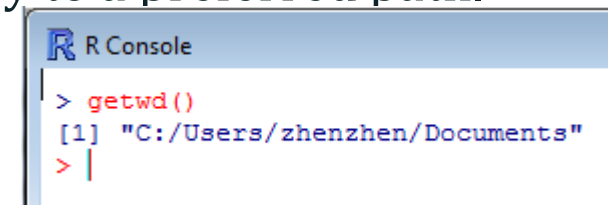
  E.g. *"C:/Users/zhenzhen/Documents"*
- Function calls need **( )**

  E.g. *rattle()*
- Often used interactively at command line. To repeat your last command, press the up arrow button "↑" on your keyboard
- You can also type the codes in a script window (R Editor).
  - Select the codes you want to run
  - Press "**Ctrl+Enter**" to run the line or selection
- The codes in R Editor can be saved as a script file (e.g. "mycode.R")
  - To run the script file: *source("mycode.R")*

R: Basic Stuff

- Useful commands for path checking and setting to make life easier later. Usually you want to set working directory to a preferred path.

  getwd(), setwd()

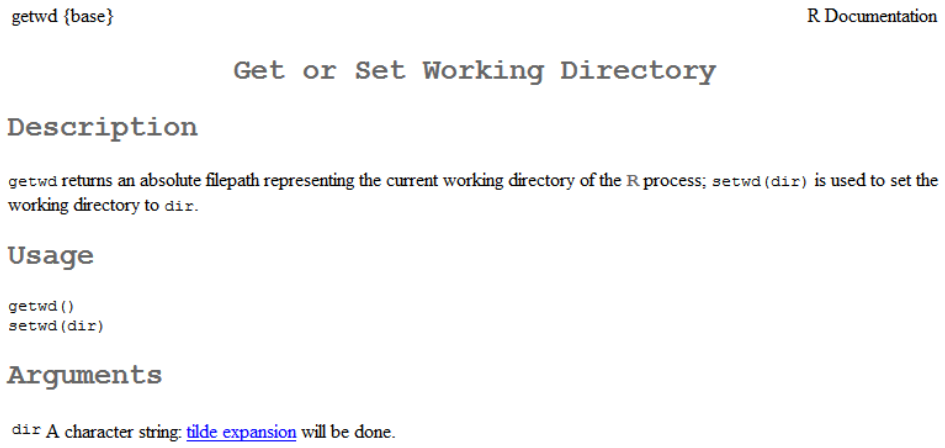


- Use '**?**' before command name to get help. The relevant page in R Documentation will be opened in your browser

  ?setwd

- To quit R:

  q()



getwd {base} R Documentation

Get or Set Working Directory

Description

`getwd` returns an absolute filepath representing the current working directory of the R process; `setwd(dir)` is used to set the working directory to `dir`.

Usage

```
getwd()
setwd(dir)
```

Arguments

`dir` A character string: tilde expansion will be done.

Objects in R

- Data structures in R environment are objects created and stored by name.
- Variables are created by assigning (using "<-")some value to a variable name
- The collection of objects is called the *workspace*.
- To list objects in workspace

  ls() or *objects()*

```
> x <- 3
> x
[1] 3
>
```

- To remove an object: *rm(x)*
- To remove all objects: *rm(list=ls())*
- Objects in an R session can be stored permanently in a file (e.g. *workspace.RData*) for future use

  File -> Save Workspace… or *save.image('workspace.RData')*

  File -> Load Workspace… or *load ('workspace.Rdata')*

Common Operators & Functions

- Arithmetic operators

  *+, -, *, /, ^*

- Logical operators

  <, <=, >, >=, ==, !=, !, &, |

- Common functions

  log, exp, sin, cos, tan, sqrt

  max, min, range, sum, length, mean, var

```
> 3 + 4
[1] 7
> 3 < 4
[1] TRUE
```

Data Structures

- Use class() to find out the type of an object
- The class for simple objects and vectors is a *mode*, which could be ***integer***, ***numeric***, ***character***, ***logical***, ***list***.
- The classes *"**matrix**"*, *"**array**"*, *"**factor**"* and *"**data.frame**"* are composites of simpler objects.
- **Vector** – an ordered collection of items, a one-dimensional array

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> class(x)
[1] "integer"
```

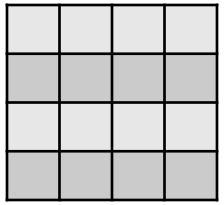
```
> y <- c('a', 'b', 'c', 'd', 'e')
> y
[1] "a" "b" "c" "d" "e"
> class(y)
[1] "character"
```

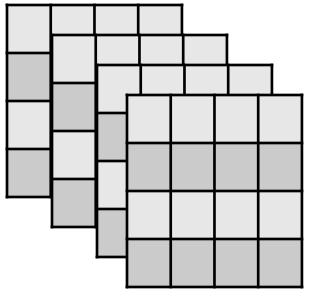
- Use **[ ]** to get subset of a vector

```
> y[3]
[1] "c"
> y[1:3]
[1] "a" "b" "c"
```

Matrix and Array

- **Matrix** – two-dimensional array **Array** – many number of dimensions





- Use *matrix()* to create matrices

```
> m <- matrix(1:24, 4, 6)
> m
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
```

- [] works with matrix too

```
> m[2,5]
[1] 18
```

```
> m[1:3, 2:4]
     [,1] [,2] [,3]
[1,]    5    9   13
[2,]    6   10   14
[3,]    7   11   15
```

- Negative index for removal of column/row

```
> m[-1, -6]
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    6   10   14   18
[2,]    3    7   11   15   19
[3,]    4    8   12   16   20
```

More about Matrix

- Most used in R
- To find out number of rows and columns

  dim() , or *nrow()*, *ncol()*

```
> dim(m)
[1] 4 6
> ncol(m)
[1] 6
> nrow(m)
[1] 4
```

- Columns and rows can be named, e.g.

```
> dimnames(m) = list(c('r1', 'r2', 'r3', 'r4'),
+ c('col1', 'col2', 'col3', 'col4', 'col5', 'col6'))
> m
   col1 col2 col3 col4 col5 col6
r1    1    5    9   13   17   21
r2    2    6   10   14   18   22
r3    3    7   11   15   19   23
r4    4    8   12   16   20   24
```

- Column/row names can be used to access the respective column/row

```
> m['r3',]
col1 col2 col3 col4 col5 col6
   3    7   11   15   19   23
```

Lists

- For vector, matrix, array, their values must all be of the same mode (type).
- For groupings of **different** R objects. Use *summary*() to get list information

```
> mylist <- list(x, y, m)
> mylist
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10

[[2]]
[1] "a" "b" "c" "d" "e"

[[3]]
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
```

```
> summary(mylist)
     Length Class  Mode
[1,] 10     -none- numeric
[2,]  5     -none- character
[3,] 24     -none- numeric
```

- [ ] bracketing is used for subsetting
- And [[ ]] for extracting a list element

```
> mylist[1]
[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
> mylist[[1]]
 [1]  1  2  3  4  5  6  7  8  9 10
```

Data Frames

- Matrix-like structures in which the columns can be of different types.
- Columns and rows can be named

```
> head(mtcars)
                   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
```

- And accessed using "$"

```
> mtcars$gear
 [1] 4 4 4 3 3 3 3 4 4 4 4 3 3 3 3 3 3 4 4 4 3 3 3 3 3 4 5 5 5 5 5 4
```

- Use *attach()* to make the column/row names temporarily visible as variables without using *"mtcars$"*, till *detach()* is called. [Warning: R Style Guide discourages the use of attach().]

```
> attach(mtcars)
> gear
 [1] 4 4 4 3 3 3 3 4 4 4 4 3 3 3 3 3 3 4 4 4 3 3 3 3 3 4 5 5 5 5 5 4
> detach(mtcars)
> gear
Error: object 'gear' not found
```

Factors

- For categorical variables
- Use *as.factor()* when categories are represented as numbers

```
> as.factor(mtcars$cyl)
 [1] 6 6 4 6 8 6 8 4 4 6 6 8 8 8 8 8 8 4 4 4 4 8 8 8 8 4 4 4 8 6 8 4
Levels: 4 6 8
> mtcars$cyl_f <- as.factor(mtcars$cyl)
> class(mtcars$cyl)
[1] "numeric"
> class(mtcars$cyl_f)
[1] "factor"
```

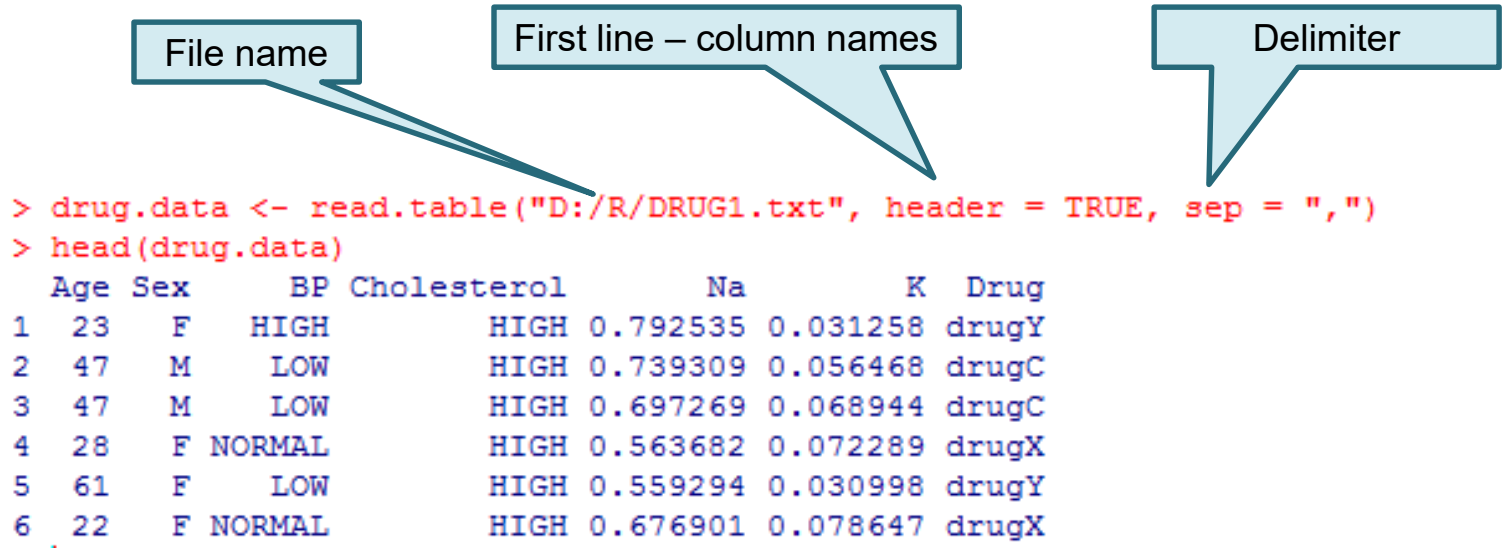
Creating a new column!

- The function *tapply()* can be used to apply a function *mean()* to each group of components' *mpg*, defined by the levels of *cyl_f*

```
> levels(mtcars$cyl_f)
[1] "4" "6" "8"
> mpg_means <- tapply(mtcars$mpg, mtcars$cyl_f, mean)
> mpg_means
       4        6        8
26.66364 19.74286 15.10000
```

Reading in Data

- *scan()* is the low level, more efficient function for reading data into R environment
- Function read.table() is more commonly used, returning a data frame



Read Data from Files

- Use *read.csv()* for comma-delimited files
- Use *read.delim()* for tab-delimited files
- Use *read.csv2()* or *read.delim2()* if comma is used to indicate decimal point in numbers (like in Europe)
- Use *read.fwf()* for files with fixed-width columns
- For MS Excel spreadsheets
  - Export them as .csv, then use *read.csv()*
  - Use packages like xlsReadWrite, XLConnect, RODBC, with their respective limitations

  See http://www.r-bloggers.com/read-excel-files-from-r/ for more details
- More functions available to read in data in various formats like ARFF, SAS, SPSS, etc.

To Get Data from Web

- Pass a URL to a suitable function which can handle this type of data
- For example, to get data from Singapore government's data site (install and load package "***jsonlite***" first)

```
> url <- "https://data.gov.sg/api/action/datastore_search?resource_id=76a8852a-093
a-4e8a-ad24-598f5e82c213"
> sgdata <- jsonlite::fromJSON(url)
> sgdata$result$records
    deaths ethnic_group live_births natural_increase year _id
1     8787      Chinese       35608            26821 1971   1
2     1464       Malays        7246             5782 1971   2
3      859      Indians        3090             2231 1971   3
4      219       Others        1144              925 1971   4
5     8905      Chinese       37797            28892 1972   5
6     1478       Malays        7594             6116 1972   6
7      895      Indians        3107             2212 1972   7
8      244       Others        1180              936 1972   8
```

- *Caution!: don't request for data too frequently when you get data from web, or you might be detected and blocked as suspected attack.*

To Get Data from Web

- For financial data, you can use package ***quantmod***, which make stock data scraping really a breeze

  install.packages('quantmod')

  library(quantmod)

  getSymbols("AAPL")

  getSymbols("GOOG")

- The returned is an XTS (Extensible Time Series) object, which can be converted into a data frame.

  GOOG

  class(GOOG)

```
> GOOG.df <- as.data.frame(coredata(GOOG))
> head(GOOG.df)
  GOOG.Open GOOG.High GOOG.Low GOOG.Close GOOG.Volume GOOG.Adjusted
1  231.4944  236.7899 229.0652   232.2842    15513200      232.2842
2  232.9847  240.4114 232.6618   240.0686    15877700      240.0686
3  239.6910  242.1749 237.5102   242.0209    13833500      242.0209
4  242.2693  243.3522 239.5420   240.2276     9570600      240.2276
5  241.1565  242.5475 239.0452   241.1814    10832700      241.1814
6  240.6498  245.1803 239.4625   243.1486    12014600      243.1486
> GOOG.df$Date <- index(GOOG)
> head(GOOG.df)
  GOOG.Open GOOG.High GOOG.Low GOOG.Close GOOG.Volume GOOG.Adjusted       Date
1  231.4944  236.7899 229.0652   232.2842    15513200      232.2842 2007-01-03
2  232.9847  240.4114 232.6618   240.0686    15877700      240.0686 2007-01-04
3  239.6910  242.1749 237.5102   242.0209    13833500      242.0209 2007-01-05
```

Write Data to Files

- Save the data as R object file (.Rdata)
  - E.g. *save(GOOG.df, file="GOOG.RData")*
  - Read the object back using *load()*
- Save the data to files in ASCII format using function *write.table()*

```
write.table(GOOG.df, file = "GOOG.csv", sep = ",", quote = FALSE,
row.names = FALSE, col.names = TRUE)
```

```
GOOG.Open,GOOG.High,GOOG.Low,GOOG.Close,GOOG.Volume,GOOG.Adjusted,Date
231.494354,236.789917,229.065155,232.28421,15513200,232.28421,2007-01-03
232.984665,240.411362,232.661758,240.068588,15877700,240.068588,2007-01-04
239.69104,242.174881,237.510223,242.020889,13833500,242.020889,2007-01-05
242.269272,243.352234,239.542007,240.227554,9570600,240.227554,2007-01-08
241.156509,242.54747,239.045242,241.181351,10832700,241.181351,2007-01-09
240.649811,245.180344,239.462524,243.14856,12014600,243.14856,2007-01-10
246.993546,249.253845,246.486847,248.245407,14510100,248.245407,2007-01-11
```

Summary

- We've learned the basics of R
- There's of course a lot more about R. References and resources are plenty on the web.
- The following modules will contain workshops in which we'll use R with RStudio.