

BIOS 545 Motivations, History, Importing Data

Department of Biostatistics and Bioinformatics

Steve Pittard wsp@emory.edu

January 8, 2018

Why R ?

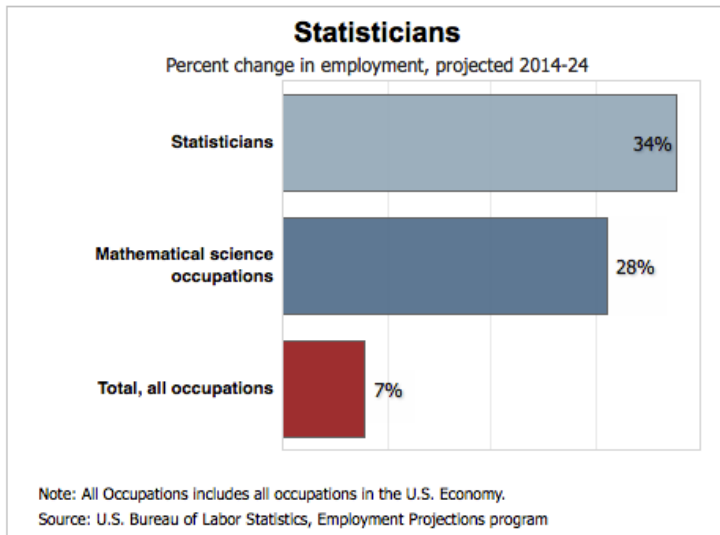
Occupational Outlook Handbook 2016

Employment of computer and information research scientists is projected to grow 11 percent from 2014 to 2024, faster than the average for all occupations. Computer research scientists are likely to enjoy excellent job prospects, because many companies report difficulties finding these highly skilled workers.

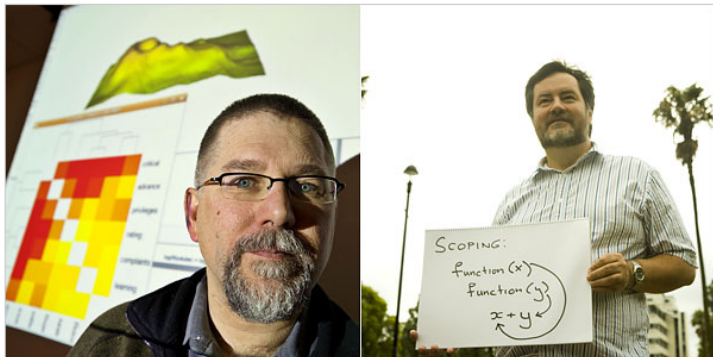
The median annual wage for computer and information research scientists was \$110,620 in May 2015.

Employment of statisticians is projected to grow 34 percent from 2014 to 2024, much faster than the average for all occupations. Growth is expected to result from more widespread use of statistical analysis to make informed business and healthcare decisions.

Why R ?



Data Analysts Captivated by R's Power



Left, Stuart Isett for The New York Times; right, Kieran Scott for The New York Times

R first appeared in 1996, when the statistics professors Robert Gentleman, left, and Ross Ihaka released the code as a free software package.

By **ASHLEE VANCE**

Published: January 6, 2009

<http://tinyurl.com/cxa774n>

Who Uses R ?

Company	How R is Used
Bank of America	Modeling and visualization
Facebook	User analysis and interaction
FDA	Used in parallel with SAS
Ford Motor Company	Decision support
Google	Calculate ROI on advertising
John Deere	Time series modeling and geospatial analysis
National Weather Service	Visualization for flood forecasting
New York Times Newspaper	Data visualization
Nordstrom	Recommendation systems
Orbitz Travel	Search result optimization
Twitter	User experience analysis
Trulia Real Estate	Housing cost predictions
OK Cupid Online Dating	Trend analysis
Lloyd's of London Insurance	Investment recommendation

<http://www.revolutionanalytics.com/companies-using-r>

Why R ?

- R is an interactive framework for data and statistical analysis that also happens to have a builtin programming language.
- Compare this to languages such as Python, Perl, and Java that have data analysis addons
- Which language to use ? Use them all if necessary but if data analysis is a large part of the work then R is the “go to” language
- R can reference or call code written in C, C++, Perl, Python, Java, and FORTRAN.
- Most of the effort in using R relates to shaping data for analysis and understanding the available functions and packages.
- To be a good *programmer* in R one must first be a knowledgeable *user* of R.

Why R ?

Differences between R and other statistical packages

“When talking about user friendliness of computer software I like to the analogy of cars vs. busses. Using this analogy programs like SPSS are busses, easy to use for the standard things, but very frustrating if you want to do something that is not already preprogrammed.”

“R is a 4-wheel drive SUV with a bike on the back, a kayak on the top, good walking and running shoes in the passenger seat, and a mountain climbing and spelunking gear in the back.”

“R can take you anywhere you want to go if you take the time to learn how to use the equipment, but that is going to take longer than learning where the bus stops are in SPSS.”

Greg Snow, R-help (May 2006)

Why R ?

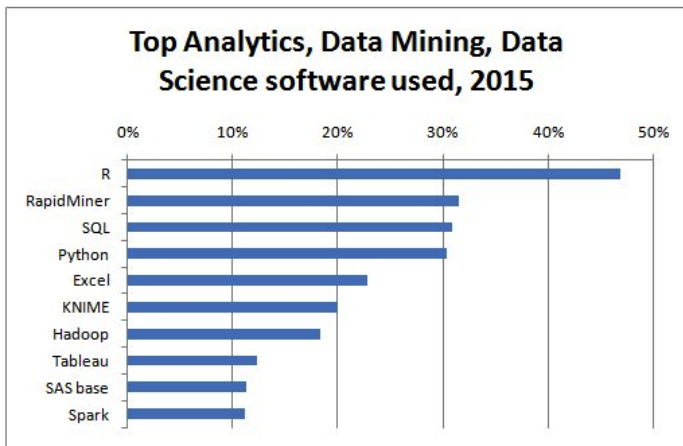
Cool things about R

- Vast capabilities with a wide range of statistical and graphics techniques
- Written primarily by statisticians
- Free of cost
- Collaborative development with over 6,092 user contributed packages
- Excellent community support with mailing lists, blogs, and tutorials
- Excellent “google” support
- Wildly popular in Academia and increasingly so in the business world

`www.slideshare.net/izahn/rintro`

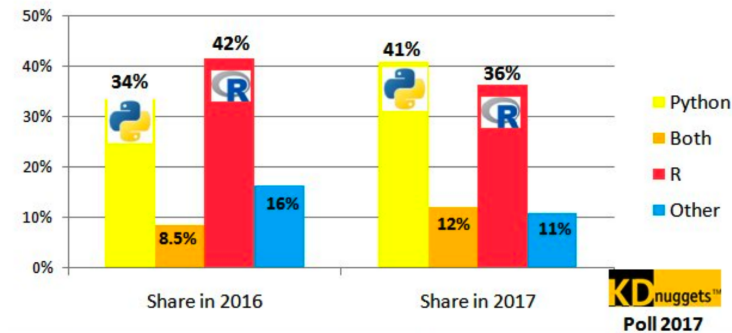
Top Analytics Tools and Trends

Here are the top 10 tools by share of usage:



R vs Other Languages - kdnuggets.com

Python, R, Both, or Other platforms for Analytics, Data Science, Machine Learning



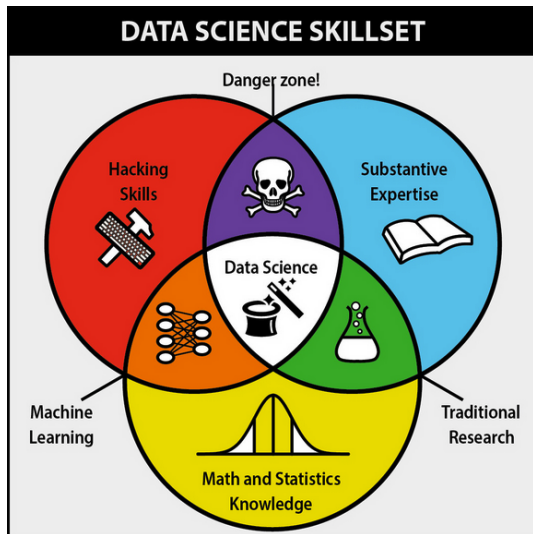
DATA SCIENTIST

Math
Statistics
Programming
Database
Domain Knowledge
Soft Skills
Communication
Visualization










(c) MarketingDistillery.com

Data Science

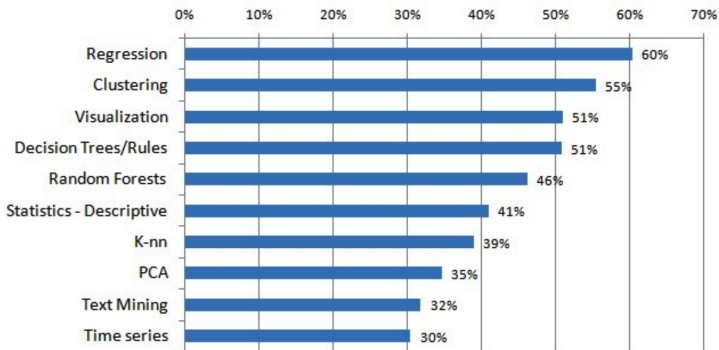


<http://i.imgur.com/aoz1BJy.jpg>

	Data science, due to its interdisciplinary nature, requires an intersection of abilities: hacking skills , math and statistics knowledge , and substantive expertise in a field of science.
	Hacking skills are necessary for working with massive amounts of electronic data that must be acquired, cleaned, and manipulated.
	Math and statistics knowledge allows a data scientist to choose appropriate methods and tools in order to extract insight from data.
	
	
	Machine learning stems from combining hacking skills with math and statistics knowledge, but does not require scientific motivation.
	Danger zone! Hacking skills combined with substantive scientific expertise without rigorous methods can beget incorrect analyses.

<http://i.imgur.com/aoz1BJy.jpg>

Top 10 Data Science, Machine Learning Methods Used, 2017



[https:](https://www.kdnuggets.com/2017/12/top-data-science-machine-learning-methods.html)

[//www.kdnuggets.com/2017/12/top-data-science-machine-learning-methods.html](https://www.kdnuggets.com/2017/12/top-data-science-machine-learning-methods.html)

Obtaining R

- Go to <https://cran.r-project.org/mirrors.html>
- Scroll down to the US Mirror sites
- Pick one
- This is in Today's "Mini Lab"

Base Packages

It is important to note that R comes with a base set of packages as part of every installation.

```
> getOption("defaultPackages")
[1] "datasets" "utils"      "grDevices" "graphics" "stats"      "methods"

> search()
[1] ".GlobalEnv"      "package:stats"    "package:graphics" "package:grDevices"
[5] "package:utils"    "package:datasets" "package:methods"   "Autoloads"
[9] "package:base"

> library(help="stats")
```


Base Packages

```
> library(help="stats")
```

Description:

```
Package:      stats
Version:      3.1.2
Priority:      base
Title:        The R Stats Package
Author:        R Core Team and contributors worldwide
Maintainer:    R Core Team <R-core@r-project.org>
Description:   R statistical functions
License:       Part of R 3.1.2
Built:         R 3.1.2; x86_64-apple-darwin13.4.0; 2014-10-31 20:19:14 UTC; unix
```

Index:

<code>.checkMFClasses</code>	Functions to Check the Type of Variables passed to Model Frames
<code>AIC</code>	Akaike's An Information Criterion
<code>ARMAacf</code>	Compute Theoretical ACF for an ARMA Process
<code>ARMAtoMA</code>	Convert ARMA Process to Infinite MA Process
<code>Beta</code>	The Beta Distribution
<code>Binomial</code>	The Binomial Distribution
<code>Box.test</code>	Box-Pierce and Ljung-Box Tests
<code>C</code>	Sets Contrasts for a Factor

Base Packages

Many packages come with example data that is helpful when attempting to understand how various functions work. To see what data sets are available in a given package, do something like:

```
> search()
[1] ".GlobalEnv" "package:lattice" "package:stats" "package:graphics"
[5] "package:grDevices" "package:utils" "package:datasets" "package:methods"
[9] "Autoloads" "package:base"

> data(package="stats") # Find data included in package "stats"
```

Data sets in package "datasets":

AirPassengers	Monthly Airline Passenger Numbers 1949-1960
BJsales	Sales Data with Leading Indicator
BJsales.lead (BJsales)	Sales Data with Leading Indicator
BOD	Biochemical Oxygen Demand
CO2	Carbon Dioxide Uptake in Grass Plants
DNase	Elisa assay of DNase
EuStockMarkets	Daily Closing Prices of Major European
..	

CRAN Packages

One of the most powerful aspects of R is the ability to install user-contributed add-on packages available in CRAN, (Comprehensive R Archive Network). As of December 2014 there are over 6,000 packages available.

To obtain information on the wide variety of packages then visit the following URL to see some of the areas covered. `cran.cnr.berkeley.edu` Also go to the “Task Views” You can also see packages grouped by domain at <http://cran.r-project.org/web/views/>

CRAN Packages

Here are some of the areas covered. There are many more of course

CRAN Task Views

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

CRAN Packages

A search engine is available:

Search all 14,246 CRAN, BioConductor and Github packages.

Search

Or explore packages in one of the [Task Views](#).

<u>Top 5 packages</u>	<u>Top 5 authors</u>	<u>Newest packages</u>
1. viridisLite	1. Hadley Wickham	1. CytobankBridgeR
2. R6	2. Kirill Müller	2. hydroscoper
3. reshape2	3. Dirk Eddelbuettel	3. pKSEA
4. yaml	4. Yihui Xie	4. spectacles
5. readxl	5. Lionel Henry	5. conformalClassification

<https://www.rdocumentation.org/>

CRAN Packages

If you are using RStudio there are menu items that can simplify the process of identifying and installing packages. However, you can also do this from the command prompt. Let's say you want to install the “actuar” package from CRAN.

```
> install.packages("actuar",dependencies=TRUE)
```

```
trying URL 'http://mirrors.nics.utk.edu/cran/bin/macosx/contrib/2.15/
actuar_1.1-5.tgz'
Content type 'application/x-gzip' length 1837121 bytes (1.8 Mb)
opened URL
```

```
=====
```

```
downloaded 1.8 Mb
```

```
> library(actuar) # Brings the package into the workspace
```

CRAN Packages

When we use the **library** command to load the contents of the **actuar** package it will show up when we execute the **search()** function. Check it out.

```
> library(actuar) # Brings the package into the workspace
```

```
> search()
```

```
[1] ".GlobalEnv" "package:actuar" "package:lattice" "package:stats"  
[5] "package:graphics" "package:grDevices" "package:utils"  
[8] "package:datasets" "package:methods" "Autoloads" "package:base"
```

CRAN Packages

On occasion you will need to install a package from a specific repository such as `omegahat.org` or `R-forge`. RStudio has menu items that can help with this but you can also do it from the command line.

```
> install.packages("GeoIP", repos = "http://www.omegahat.org/R")
```

Sometimes you download packages written by colleagues and you have to install them from your local hard drive. Again, RStudio can help but you could also do something like:

```
$ R CMD INSTALL GeoIP.tar.gz
```


CRAN Packages

There are lots of free books and tutorials on the web.

```
> install.packages("GeoIP", repos = "http://www.omegahat.org/R")
```

Sometimes you download packages written by colleagues and you have to install them from your local hard drive. Again, RStudio can help but you could also do something like:

```
$ R CMD INSTALL GeoIP.tar.gz
```

Finding Documentation

There are lots of free books on the web

Resource	URL
The R Inferno	http://www.burns-stat.com/documents/books/the-r-inferno/
R Programming Wiki	http://en.wikibooks.org/wiki/R_Programming
Intro to Stats Using R	http://ipsur.org
Stats with R	http://zoonek2.free.fr/UNIX/48_R/all.html
Lattice Graphics	http://lmdvr.r-forge.r-project.org
Contributed R Info	http://cran.r-project.org/other-docs.html
simpleR Intro Stats	http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf
DIY Intro to R	http://www.unt.edu/rss/class/Jon/R_SC/
R Bloggers	http://www.r-bloggers.com/
R Journal	http://journal.r-project.org/
R Tutorial	http://www.r-tutor.com/r-introduction
Google Style Guide	https://github.com/hadley/devtools/wiki/Style
Applied Epi Using R	http://www.medepi.net/docs/EpidemiologyUsingR.pdf

Finding Documentation

There are some good books you can buy although for this class they aren't required.

Book	Author
R Cookbook	Paul Teetor
R in a Nutshell	Joseph Adler
The Art of Programming	Norman Matloff
Data Manipulation with R	Phil Spector
ggplot2: Elegant Graphics for Data Analyses	Hadley Wickham
Intro to Scientific Programming and Simulation Using R	Jones, Maillardet, Robinson
Introductory Statistics with R	Peter Dalgaard
The R Book	Michael J. Crawley
Discovering Statistics Using R	Andy Field

Mailing Lists

- Here are some mailing lists that accept questions relative to R and BioConductor.
- Moderators and participants in these lists take questions seriously, sometimes too seriously,
- Please don't ask a question without first searching through the archives to see if your question has already been answered. Chances are it has.

Mailing Lists	URL
R-Help	http://stat.ethz.ch/mailman/listinfo/r-help
Cross Validated	http://stats.stackexchange.com
Stack Overflow	http://stackoverflow.com/questions/tagged

Getting Help

R has a number of ways to get help. Rstudio has a Help menu item. Other ways include the following:

```
> help.start()           # Launches a web browser with search capability

> help(function_name)    # Get help on "function_name"

> ?function_name         # Equivalent to the above

> args(function_name)    # See what arguments the function accepts

> example(function_name) # See an example of the function

> example(mean)
```

```
mean> x <- c(0:10, 50)
mean> xm <- mean(x)
mean> c(xm, mean(x, trim = 0.10))
[1] 8.75 5.50
```

Getting Help

R has a number of ways to get help. Rstudio has a Help menu item. Other ways include the following:

```
# Find all functions and data having to do with time series
```

```
> help.search("time series")
```

```
> ?? "time series"      # Equivalent to the above
```

Help files with alias or concept or title matching "time series" using fuzzy matching:

<code>boot::tsboot</code>	Bootstrapping of Time Series
<code>datasets::austres</code>	Quarterly Time Series of the Number of Australian Residents
<code>datasets::beavers</code>	Body Temperature Series of Two Beavers
<code>ggplot2::economics</code>	US economic time series.
<code>lattice::xyplot.ts</code>	Time series plotting methods
<code>MASS::beav1</code>	Body Temperature Series of Beaver 1
<code>MASS::beav2</code>	Body Temperature Series of Beaver 2
<code>stats::StructTS</code>	Fit Structural Time Series
<code>stats::ar</code>	Fit Autoregressive Models to Time Series
<code>stats::ar.ols</code>	Fit Autoregressive Models to Time Series by OLS
<code>..</code>	

Things to Know !

- Everything in R is an object
- The great thing about R is that there are many different ways to do something
- The bad thing about R is that there are many different ways to do something
- Everything that happens in R is a function call
- Supports procedural programming with functions and object oriented programming
- R is based on a “read-eval-print” loop
- Interpreted language

Walkthrough

```
url <- "http://pittardsp.github.io/YOUTUBE.DIR/table_7_3.csv"
```

```
engine <- read.table(url, sep = ",", header=TRUE)
engine <- engine[,-1]
```

```
head(engine) # 3 engine pollutants
```

```
      hc co nox
1 0.50 5.01 1.28
2 0.65 14.67 0.72
3 0.46 8.60 1.17
4 0.41 4.42 1.31
5 0.41 4.95 1.16
```

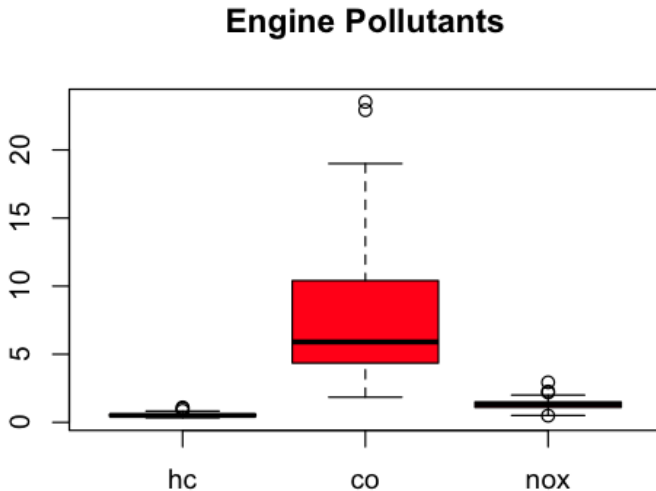
```
summary(engine)
```

```
      en hc co nox
Min.   : 1.00   Min.   :0.3400   Min.   : 1.850   Min.   :0.490
1st Qu.:12.75   1st Qu.:0.4375   1st Qu.: 4.388   1st Qu.:1.110
Median :24.50   Median :0.5100   Median : 5.905   Median :1.315
Mean   :24.00   Mean   :0.5502   Mean   : 7.879   Mean   :1.340
3rd Qu.:35.25   3rd Qu.:0.6025   3rd Qu.:10.015   3rd Qu.:1.495
Max.   :46.00   Max.   :1.1000   Max.   :23.530   Max.   :2.940
```

```
http://www.cyclismo.org/tutorial/R/hwI.html
```


Walkthrough

```
boxplot(engine,col="red",main="Engine Pollutants")
```



Walkthrough

```
par(mfrow=c(1,3))
```

```
boxplot(engine$co,main="Carbon Monoxide")
```

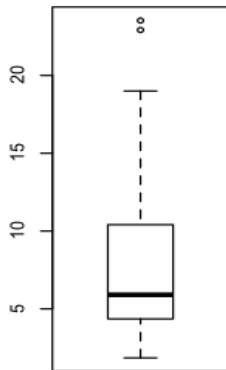
```
hist(engine$co)
```

```
qqnorm(engine$co,main="Carbon Monoxide")
```

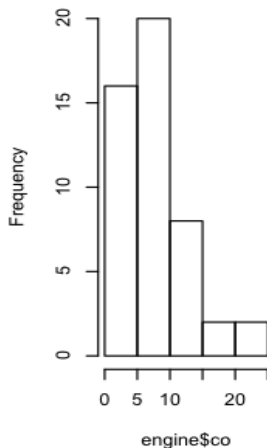
```
qqline(engine$co)
```

Walkthrough

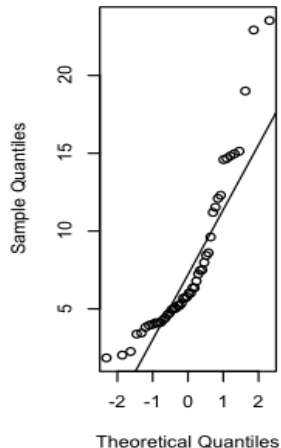
Carbon Monoxide



Histogram of engine\$co



Carbon Monoxide



Walkthrough

The null hypothesis is that the data is normal

```
shapiro.test(engine$co)
```

Shapiro-Wilk normality test

```
data: engine$co
```

```
W = 0.8357, p-value = 9.289e-06
```

Take the log of the CO

```
log.engine <- log(engine$co)
```

```
shapiro.test(log.engine)
```

Shapiro-Wilk normality test

```
data: log.engine
```

```
W = 0.9693, p-value = 0.2379
```

Walkthrough

```
par(mfrow=c(2,2))

log.engine <- log(engine$co)

boxplot(log.engine,main="Carbon Monoxide")

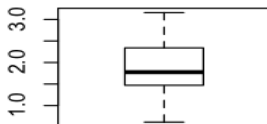
hist(log.engine,main="Carbon Monoxide")

qqnorm(log.engine,main="QQ Plot for the Log of the
        Carbon Monoxide")

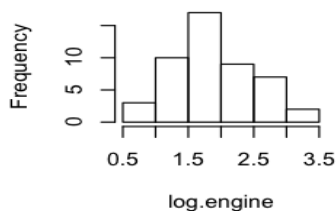
qqline(log.engine)
```

Walkthrough

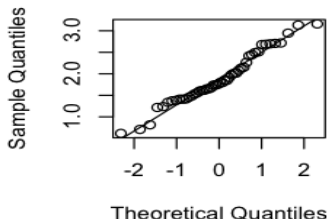
Carbon Monoxide



Carbon Monoxide



QQ Plot for the Log of the Carbon Monox



Walkthrough

```
# Let's build a confidence interval
```

```
my.mean <- mean(log.engine)
my.sd <- sd(log.engine)
n <- length(log.engine)
```

```
# Get standard error
se <- my.sd/sqrt(n)
```

```
error <- se*qt(0.975,df=n-1)
```

```
left <- my.mean - error
```

```
right <- my.mean + error
```

```
c(left,right)
[1] 1.709925 2.057431
```

```
c(exp(left),exp(right))
[1] 5.528548 7.825840
```

Walkthrough

```
# Test H0: mu = 5.4
# HA:mu != 5.4

lNull <- log(5.4) - error

rNull <- log(5.4) + error

c(lNull,rNull)
[1] 1.512646 1.860152

my.mean
[1] 1.883678
```

So the mean is outside the range thus we reject the null. There is a low probability that we would have obtained our sample mean if the true mean really was 5.4

Walkthrough

We could have calculated a p-value by hand

```
p.val <- 2*(1-pt((my.mean-log(5.4))/se,df=n-1))
```

```
p.val  
[1] 0.02692539
```

But its easier to call a procedure to do it all !!!!

```
t.test(log.engine,mu = log(5.4),alternative = "two.sided")
```

One Sample t-test

data: log.engine

t = 2.2841, df = 47, p-value = 0.02693

alternative hypothesis: true mean is not equal to 1.686399

95 percent confidence interval:

1.709925 2.057431

sample estimates:

mean of x

1.883678

First R Session

```
?mean                # Get help on the mean function

example(kmeans)      # Run an example of kmeans (if it exists)

pi                    # Some popular quantities are built-in to R
[1] 3.141593

sqrt(2) # Basic arithmetic
[1] 1.414214

print(pi) # Print the comments of the pi variable
[1] 3.141593

X <- 3; Y <- 4 # Semicolon lets you enter 2 commands on the same line

Z <- sqrt(X^2 + Y^2) # Variables contain information

# List all variables in the "environment"

ls()
[1] "X" "Y" "Z"
```

<code>log(10)</code> [1] 2.302585	<code>ceiling(6.8)</code> [1] 7	<code>2+3</code> [1] 5
<code>log10(100)</code> [1] 2	<code>round(6.889,2)</code> [1] 6.89	<code>3/2</code> [1] 1.5
<code>sin(pi/2)</code> [1] 1	<code>3/0</code> [1] Inf	<code>2^3</code> [1] 8
<code>cos(pi/2)</code> [1] 6.123234e-17	<code>0/0</code> [1] NaN	<code>(56-14)/6 - 4*7*10/(5^2-5)</code> [1] -7
<code>1.3e6</code> [1] 1300000	<code>is.finite(3)</code> [1] TRUE	<code>abs(2-4)</code> [1] 2
<code>9 %% 2</code> [1] 1	<code>x <- c(1:8,NA)</code> [1] 1 2 3 4 5 6 7 8 NA	
<code>floor(5.7)</code> [1] 5	<code>mean(x)</code> NA	

Common Operators

RELATIONAL OPERATORS

Equal to	==	if (myvar == "test") {print("EQ")}
	==	if (mynum == 3) {print("EQ")}
Not equal to	!=	if (myvar != "test") {print("NE")}
Less than or equal to	<=	if (number <= 5) {print("LTE")}
Less than	<	if (number < 10) {print("LT")}
Greater than or equal to	>=	if (number >= 10) {print("GTE")}
Greater than	>	if (number > 12) {print("GT")}

BOOLEAN OPERATORS

And	&	if ((myvar == "test") & (num <= 10)) { print("Equal and less than") }
Not	!	if (!complete.cases(myvec)) { print("Non complete cases") }
Or		if ((num > 3) (num < -3)) { print("Only one of these has to be true") }

More Examples

Here are some popular math formulas rewritten in R. Note that the variables must first exist in order for the formula to do an actual computation.

```
#  $a^2 + b^2 = c^2$                                 # Pythagorean Theorem

a <- 2; b <- 4

c <- sqrt(a^2 + b^2)                            # To solve the PT for c

a <- 2; b <- 4; c <- 1

(-b + sqrt(b^2-4*a*c)) / (2*a)                 # First case quadratic formula solution

(-b - sqrt(b^2 - 4*a*c)) / (2*a)              # Second case quadratic formula solution
r <- 4; h <- 6; b <- 3

circumference <- 2*pi*r                        # circumference of a circle

area <- (b*h)/2 # Area of a triangle
```

More Examples

You can create expressions that can be evaluated later. The variables they reference don't have to exist. They are placeholders.

```
area <- expression( (b*h)/2 )
```

```
# Solve where b =3 and h = 4
```

```
b <- 3
```

```
h <- 4
```

```
eval(area)
```

```
[1] 6
```

Expressions

```
r1 <- expression((-b + sqrt(b^2 - 4*a*c)) / (2*a))  
r2 <- expression((-b - sqrt(b^2 - 4*a*c)) / (2*a))  
# Solve for  $ax^2 + bx + c$  where  $a = 1$ ,  $b=6$ , and  $c=8$ 
```

```
a = 1 ; b=6 ; c=8  
eval(r1)
```

```
[1] -2
```

```
eval(r2)
```

```
[1] -4
```

```
a*eval(r1)^2 + b*eval(r1) + c
```

```
[1] 0
```

```
a*eval(r2)^2 + b*eval(r2) + c
```

```
[1] 0
```

Expressions

We can create functions which are “grown up ” expressions.

```
my.quad <- function(a,b,c) {  
  r1 <- (-b + sqrt(b^2 - 4*a*c)) / (2*a)  
  r2 <- (-b - sqrt(b^2 - 4*a*c)) / (2*a)  
  my.roots = c(r1,r2)  
  return(my.roots)  
}  
# Solve for  $ax^2 + bx + c$  where  $a = 1$ ,  $b=6$ , and  $c=8$   
  
my.quad(1,6,8)  
  
[1] -2 -4
```


Startup

- You can use the Preferences menu item in RStudio to specify your default home directory
- When R starts it looks for a file called `.Rprofile` within your home directory
- You can influence the R environment by setting a number of “startup” variables therein
- Use your favorite editor to create/edit this file in your default folder
- You can change many of these variables or options during an R session but if you want them to be permanent then you will need to edit the `.Rprofile` file

Startup .Rprofile

```
# Things you might want to change

options(editor="notepad")
cd = setwd
pwd = getwd
lss = dir

# R interactive prompt
setwd("/Users/fender/steve.test") # Set's my default directory for me.
options(prompt="> ")
options(continue="+ ")

# General options
options(digits=3)
options(width = 130)
options(graphics.record=TRUE)
.First <- function(){                # You can load functions
  library(Hmisc)
  cat("\nWelcome at", date(), "\n")
}
.Last <- function(){
  cat("\nGoodbye at ", date(), "\n")
}
```

Workspace - Being Organized

Being organized helps ! Knowing how to find stuff quickly is essential.
Create a master folder that will contain your work in this class.

You can create subfolders according to your projects. Note that some people do this on a DropBox folder to insure that all work is backed up.

```
$ ls RProjects
RProjects
  Data_Files
  Genomes
    1000_Genomes
    Centenarians
  HIV
    Replicates
  Hepatitis
    Hep_A
    Hep_B
```

Workspace - Navigating Directories

There are a number of functions that allow you to “move” around in your folder structure. These are important to know because sometimes you will need to write code that needs to refer to specific folders and files during execution.

```
getwd()  
[1] "/Users/fender/TEST.DIR"
```

```
setwd("/Users/fender")  
getwd()  
[1] "/Users/fender"
```

```
setwd("/Users/fender/TEST.DIR")  
getwd()  
[1] "/Users/fender/TEST.DIR"
```

```
dir()  
[1] "coolpkg" "coolpkg_1.0.tar.gz" "coolpkg.pdf" "coolpkg.Rcheck"  
"g.Rd" "stuff.R"
```

Workspace - Listing Files

R also has some functions that list files in a folder. You can do this visually within R Studio although sometimes you will need to use these commands to open and read in files as part of a program.

```
myfiles <- list.files()
```

```
str(myfiles)
```

```
chr [1:29] "001.csv" "002.csv" "003.csv" "004.csv" "005.csv" "006.csv" ...
```

```
myfiles[1:5]
```

```
[1] "001.csv" "002.csv" "003.csv" "004.csv" "005.csv"
```

You could write a for-loop to process each and every file

```
for (ii in 1:length(myfiles)) {
```

```
  file <- myfiles[ii]
```

```
  # Do something
```

```
}
```

Workspace - ls()

R creates an environment for each session you initiate. This is very useful because it accumulates all your variables and objects while you experiment with data.

Over time your environment will accumulate lots of variables. In general this is good because you don't lose anything. The **ls()** function can show you what objects you currently have in your environment.

```
ls()
[1] "access_log"      "cntr"
[3] "ii"              "init"
[5] "mpg"             "mtcars"
[7] "mymean"          "myrle"
[9] "mystr"           "nhanes1"
[11] "retvec"          "retvectr"
[13] "SacramentocrimeJanuary2006" "Sacramentorealestatetransactions"
[15] "SalesJan2009"
```

Workspace - rm()

You can remove one or more objects using the **rm()** function

```
ls()
[1] "access_log"          "cntr"
[3] "ii"                  "init"
[5] "mpg"                 "mtcars"
[7] "mymean"              "myrle"
[9] "mystr"               "nhanes1"
[11] "retvec"              "retvectr"
[13] "SacramentocrimeJanuary2006" "Sacramentorealestatetransactions"
[15] "SalesJan2009"
```

```
rm(access_log)    # Removes the object named "access_log"
```

```
access_log        # Now R can't find it
Error: object 'access_log' not found
```

```
rm(mystr,retvec,init)    # Remove more than one object at once
```

Workspace - .Rdata

When you quit R you will be asked if you wish to save your current environment to disk. If you type “y” then all objects, (and their values), will be written to a file called **.Rdata**

This is useful because when you restart R in the same folder it will read **.Rdata** which contains all previously saved information.

```
> q()
Save workspace image? [y/n/c]: y
Goodbye at Mon Oct 1 14:26:47 2012

fenders-macbook:TEST.DIR fender$ ls .Rdata
.Rdata
```

The **.Rdata** file is a “binary” file, (its contents are unintelligible to the eye), that contains all the R objects and values in between sessions. This file could be shared with others if you wanted.

Workspace - save()

You can also save one or more objects to a file using the **save()** function. The inverse of the **save()** function is the **load()** function.

```
my.lm <- lm(mpg ~ wt,mtcars)
```

```
ls(my.lm)
```

```
[1] "assign" "call" "coefficients" "df.residual" "effects" "fitted.values"  
[7] "model"   "qr" "rank" "residuals" "terms" "xlevels"
```

```
save(my.lm,file="/Users/myhome/mylmresults")
```

```
# You can come back later and load this file
```

```
mylmstuff <- load("/Users/myhome/mylmresults")
```

Variables

As in most programming languages, it is customary to store or hold the results of an operation in a variable name.

In R such results are assigned with the symbols "<-" or "=". Variable names are case sensitive.

```
A <- 2.5    # The "<-" is the preferred method of assignment
```

```
A = 2.5     # This is equivalent to the above although using the "=" is  
            # discouraged except in setting function arguments.
```

```
A  
[1] 2.5
```

```
mynewvar <- X + 3
```

```
MYNEWVAR <- X + 3    # Two different variables
```

Variables

- R has several one-letter reserved words: c, q, s, t, C, D, F, I, T
- Variables cannot begin with the period characters “.”
- Variable names are case sensitive, so “myvar” is different from “Myvar”
- Variable names cannot begin with numbers or symbols (%,\$,_)
- Variable names cannot contain spaces in the name (“my var”)

Variables

mean.height	.mean.height
smoker	_myvariable
non.smoker	_Mean.height
temp.var	1variable
patient_id	1_variable
Eye.Color	%some.var
State.Population	some.var
disease.state	"some var"
White_Cell_Count	\$myvar
jobTitle	

Reading and Writing Files

R has a number of builtin example data frames. One common way to import data is via “.csv” files. Before we consider reading a .csv file let's first create one.

```
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

```
write.table(mtcars,file="mtcars.csv",  
            row.names=TRUE,           # Row names get saved  
            col.names=TRUE,          # Header gets saved  
            sep=",")                 # Field separator is ,
```

```
$ head mtcars.csv
```

```
"mpg","cyl","disp","hp","drat","wt","qsec","vs","am","gear","carb"  
"Mazda RX4",21,6,160,110,3.9,2.62,16.46,0,1,4,4  
"Mazda RX4 Wag",21,6,160,110,3.9,2.875,17.02,0,1,4,4
```

Reading and Writing Files

The first line of `mtcars.csv` describes the column names. Each subsequent row represents an observation with each field being separated by a “,”.
Let's read it in:

```
mycars <- read.table("mtcars.csv",header=TRUE,sep=",")
```

```
head(mycars)
```

	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

Read from the Internet

You can read CSV files directly from the Internet as long as you have the URL.

```
url <- "http://pittardsp.github.io/YOUTUBE.DIR/hsb2.csv"
```

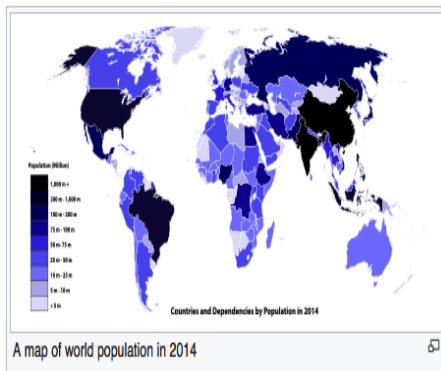
```
my.input <- read.table(url,header=T,sep=",")
```

```
head(my.input)
```

	gender	id	race	ses	scht	prgtype	read	write	math	science	socst
1	0	70	4	1	1	general	57	52	41	47	57
2	1	121	4	2	1	vocati	68	59	53	63	61
3	0	86	4	3	1	general	44	33	54	58	31
4	0	141	4	3	1	vocati	63	44	47	53	56
5	0	172	4	2	1	academic	47	52	57	53	61
6	0	113	4	2	1	academic	44	52	51	63	61

Reading Tabular Data from the Internet

You already know that you can read CSV files directly a URL. But you can also read tabular data from a Wikipedia page like the Wikipedia page for the World Population. https://en.wikipedia.org/wiki/List_of_countries_and_dependencies_by_population



Reading Tabular Data from the Internet

```
library(rvest)
url <- "https://en.wikipedia.org/wiki/List_of_countries_and_dependencies_by_population"

my_html <- read_html(url)

my_tables <- html_nodes(my_html,"table")[[1]]
populous_table <- html_table(my_tables)

populous_table <- populous_table[,-4:-6]
populous_table$Population <- as.numeric(gsub(",", "",
                                              populous_table$Population))/100000

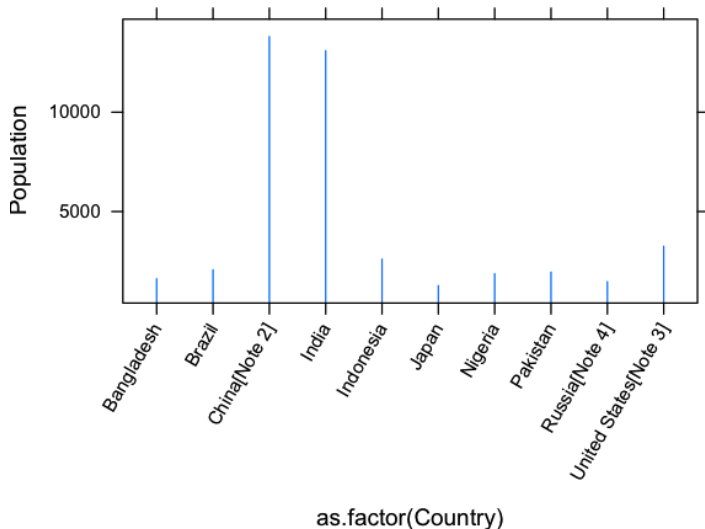
names(populous_table) = c("Rank", "Country", "Population")

# Let's plot the first 10 rows

library(lattice)
xyplot(Population ~ as.factor(Country), populous_table[1:10,],
       scales = list(x = c(rot=60)), type="h", main="Most Densely Populated Countries")
```

Reading Tabular Data from the Internet

Most Densely Populated Countries



Reading External Files

Here is a summary of tools to read in various external files, other statistical package formats, and relational databases:

Package/Function	Description
readxl	Reads Excel Worksheets and Workbooks
gdata	Reads Excel Worksheets and Workbooks
XLConnect	Reads Excel Worksheets and Workbooks
RODBC	Reads Excel Worksheets and Workbooks
reader	Read flat/tabular text files from disk
read.table	Read tabular data from disk
read.csv	Read tabular data from disk
fread	Read large data files from disk
haven	Import SAS, STATA, and SPSS files
foreign	Import SAS, STATA, SPSS, Systat, and Weka files
RMySQL	Connect to MySQL Databases
ROracle	Connect to Oracle Databases
RPostgres	Connect to Postgres Databases

Reading Excel Files

It is possible to read an Excel spreadsheet although the best thing to do is to first save the spreadsheet into a .csv file and then import it into R using **read.table()** function. However, you can read the spreadsheet directly from a file using the add on **RODBC** package.

```
library(RODBC)
```

```
channel <- odbcConnectExcel("examp.xls")
```

```
## list the spreadsheets
```

```
sqlTables(channel)
```

```
TABLE_CAT TABLE_SCHEM TABLE_NAME TABLE_TYPE REMARKS
1 C:\\bdr NA Sheet1$ SYSTEM TABLE NA
2 C:\\bdr NA Sheet2$ SYSTEM TABLE NA
3 C:\\bdr NA Sheet3$ SYSTEM TABLE NA
4 C:\\bdr NA Sheet1$Print_Area TABLE NA
```

```
## retrieve the contents of sheet 1, by either of
```

```
sh1 <- sqlFetch(channel, "Sheet1")
```

```
sh1 <- sqlQuery(channel, "select * from [Sheet1$]")
```

Reading Files from Other Packages

It is possible to read data sets from other statistical packages although I think the best way is to first export data to CSV files and then into R. But here are the available functions.

Function(s)	Purpose
<code>read.epinfo</code>	Read saved objects from EpiInfo
<code>read.xport</code>	Read saved objects in SAS export format
<code>read.spss</code>	Read saved objects from SPSS written using the save or export command
<code>read.systat</code>	Read saved objects from SYSTAT rectangular (mtype=1) data only
<code>read.dta</code>	Read saved objects from STATA (versions 5-9)
<code>read.mtp</code>	Read Minitab Portable Worksheet Files
<code>read.octave</code>	Read saved objects from GNU octave
<code>read.dbf</code>	Read or write saved objects from DBF files (FoxPro, dBase, etc)

Reading Files from Other Packages

R can process XML files which is a format that underlies many websites that distribute interesting data. As an example we can use R and XML to “geocode” cities.

Google Maps API Web Services  191

[Introduction](#)

[Directions API](#)

[Distance Matrix API](#)

[Elevation API](#)

[Geocoding API](#)

[Time Zone API](#)

[Blog](#)

[Forum](#)

[FAQ](#)

The Google Geocoding API

[What is Geocoding?](#)

[Audience](#)

[Usage Limits](#)

[Geocoding Requests](#)

[Geocoding Responses](#)

[JSON Output Formats](#)

[XML Output Formats](#)

[Status Codes](#)

[Results](#)

[Address Component Types](#)

[Reverse Geocoding](#)

[Viewport Biasing](#)

[Region Biasing](#)

[Component Filtering](#)

<https://developers.google.com/maps/documentation/geocoding/>

Reading XML

```
- <GeocodeResponse>
  <status>OK</status>
  - <result>
    <type>locality</type>
    <type>political</type>
    <formatted_address>Atlanta, GA, USA</formatted_address>
    - <address_component>
      <long_name>Atlanta</long_name>
      <short_name>Atlanta</short_name>
      <type>locality</type>
      <type>political</type>
    </address_component>
    - <address_component>
      <long_name>Fulton</long_name>
      <short_name>Fulton</short_name>
      <type>administrative_area_level_2</type>
      <type>political</type>
    </address_component>
```

Reading XML

As an example we'll get the latitude and longitude corresponding to the city of Atlanta, Georgia

```
library(RCurl)    # Install these if necessary
library(XML)

google.url <- "http://maps.googleapis.com/maps/api/geocode/xml?address="
query.url <- paste(google.url, "Atlanta,GA","&sensor=false", sep="")
txt <- getURL(query.url)
xml.report <- xmlTreeParse(txt,useInternalNodes=TRUE)

place <- getNodeSet(xml.report,
                     "//GeocodeResponse/result[1]/geometry/location[1]/*")
as.numeric(sapply(place,xmlValue))
[1] 33.74900 -84.38798
```