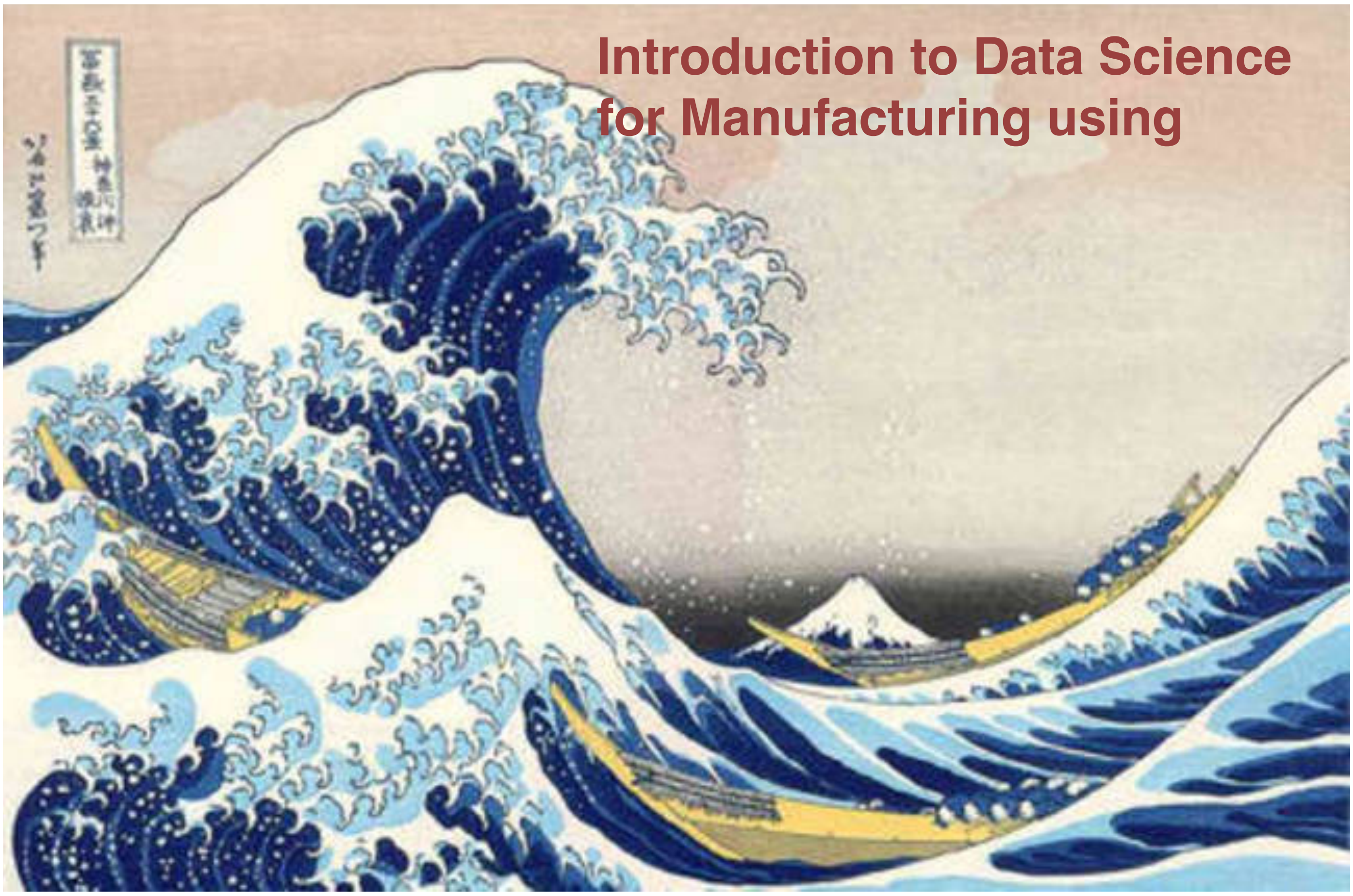
Introduction to Data Science for Manufacturing using



Veerasak Kritsanapraphan - Software Park Thailand

Agenda

Day 1

1. Introduction to Data Science
    - Definition
    - Sample from other industry
    - Data Science Process
2. Basic Statistics using R
    - Hypothesis Setting
    - T-Test
    - ANOVA
    - Regression
    - Design of Experiment
3. Exploratory Data Analysis
    - Exploring Times Series Data
    - Basic Statistic Chart
    - Advance Charting

Day 2

4. Statical Quality Control using R
    - Control Chart
    - Finding outlier
5. Multivariate Data Analysis
    - Multiple Regression
    - Logistic Regression
6. Machine Learning Technique in R
    - Basic Idea
    - Training and Testing

Day 3

7. Predictive Analytics
    - Decision Tree
    - Naive Bayes
    - Neural Nets
8. Clustering Analytics
    - K-Means
    - Hierarchical Clustering
9. Other Machine Learning
    - Text-Mining
10. Using Multi-Model
11. Conclusion
    - Wrap-up

About myself?

- Graduated from San Francisco State University in Master of Science in Computer Information System, 1997
- PhD. Candidate at Chulalongkorn University, research focus on Data Science, Big Data, Mobile Computing and Internet of Thing (IOT)
- Chief Technology Innovation Officer at Greenline Synergy (Subsidiary of BDMS - Bangkok Hospital Group)
- Instructor for Software Park in Data Science, Requirement Discovery and Practical Enterprise Integration
- Instructor for Chulalongkorn University in Data Mining

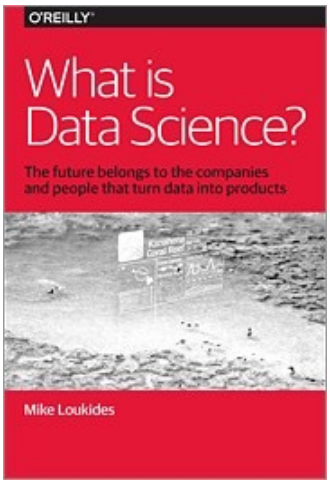
Slide and Sample Data

https://github.com/vkrit/r-programming

github
SOCIAL CODING

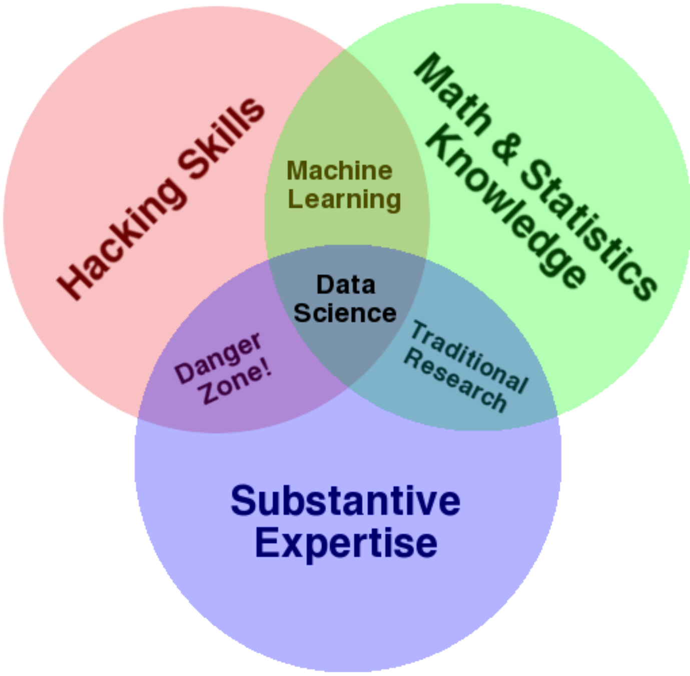
What is Data Science?

- Data Science aims to derive **knowledge** from **big data**, **efficiently** and **intelligently**
- Data Science encompasses the **set of activities**, **tools**, and **methods** that enable **data-driven activities** in science, business, medicine, and government



http://www.oreilly.com/data/free/what-is-data-science.csp

Data Science - Drew Convey's



http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram

Data Science vs. Databases

Element	Databases	Data Science
Data Value	Precious	Cheap
Data Volume	Modest	Massive
Examples	Bank records, Personnel records, Census, Medical records	Online clicks, GPS logs, Tweets, tree sensor readings
Priorities	Consistency, Error recovery, Auditability	Speed, Availability, Query richness
Structured	Strongly (Schema)	Weakly or none (Text)
Properties	Transactions, ACID+	CAP* theorem (2/3), eventual consistency
Realizations	Structured Query Language (SQL)	NoSQL: Riak, Memcached, Apache Hbase, Apache River, MongoDB, Apache Cassandra, Apache CouchDB, …

*CAP = Consistency, Availability, Partition Tolerance"
+ACID = Atomicity, Consistency, Isolation and Durability"

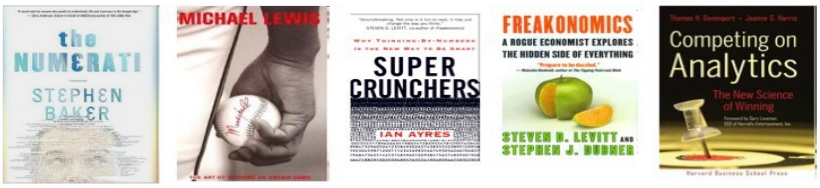
Data Science vs. Databases

Databases	Data Science
Querying the past	Querying the future

Related – Business Analytics

» Goal: obtain “actionable insight” in complex environments"

» Challenge: vast amounts of disparate, unstructured data and limited time"



Data Science vs. Traditional Machine Learning

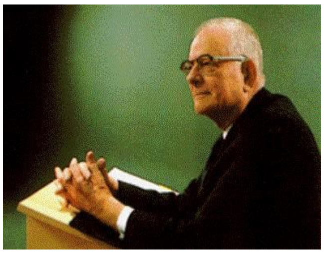
Traditional Machine Learning	Data Science
Develop new (individual) models	Explore many models, build and tune hybrids
Prove mathematical properties of models	Understand empirical properties of models
Improve/validate on a few, relatively clean, small datasets	Develop/use tools that can handle massive datasets
Publish a paper	Take action!

History of Data Science

- R. A. Fisher
  » 1935: "The Design of Experiments"

  "correlation does not imply causation"

- W. E. Demming
  » 1939: "Quality Control"

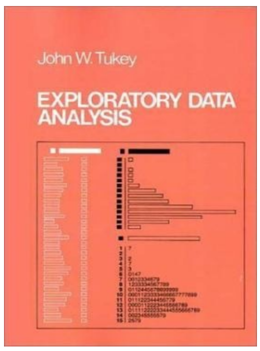


Images: http://culturacientifica.wikispaces.com/CONTRIBUCIONES+DE+SIR+RONALD+FISHER+A+LA+ESTADISTICA+GENETICA
http://es.wikipedia.org/wiki/William_Edwards_Deming

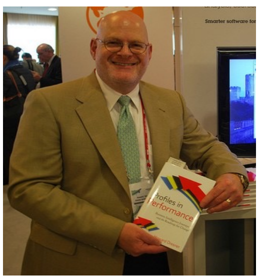
Brief Data Analysis History

- Peter Luhn
  » 1958: "A Business Intelligence System"

- John W. Tukey
  » 1977: "Exploratory Data Analysis

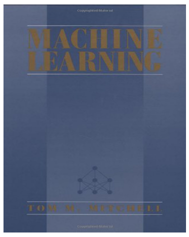


- Howard Dresner
  » 1989: "Business Intelligence"



Images: http://www.businessintelligence.info/definiciones/business-intelligence-system-1958.html http://www.betterworldbooks.com/exploratory-data-analysis-id-0201076160.aspx
https://www.flickr.com/photos/42266634@N02/4621418442

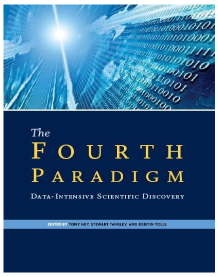
Brief Data Analysis History



- Tom Mitchell
  » 1997: "Machine Learning book"

- Google
  » 1996: "Prototype Search Engine"

- Data-Driven Science eBook
  » 2007: "[The Fourth Paradigm](http://research.microsoft.com/en-us/collaboration/fourthparadigm/)"



Images: http://www.amazon.com/Machine-Learning-Tom-M-Mitchell/dp/0070428077 http://www.google.com/about/company/history/
http://research.microsoft.com/en-us/collaboration/fourthparadigm/

Brief Data Analysis History

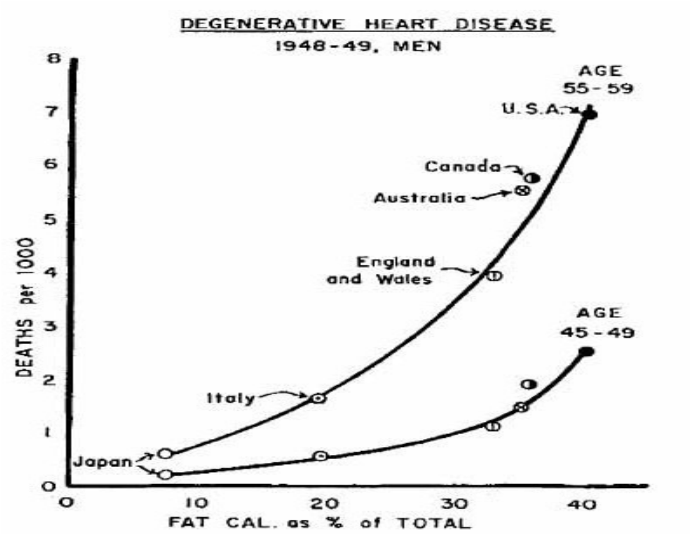
- Peter Norvig
  » 2009: "The Unreasonable Effectiveness of Data"

- Exponential growth in data volume
  » 2010: "The Data Deluge"

Images: http://en.wikipedia.org/wiki/Peter_Norvig
http://www.economist.com/node/15579717

Data Science?

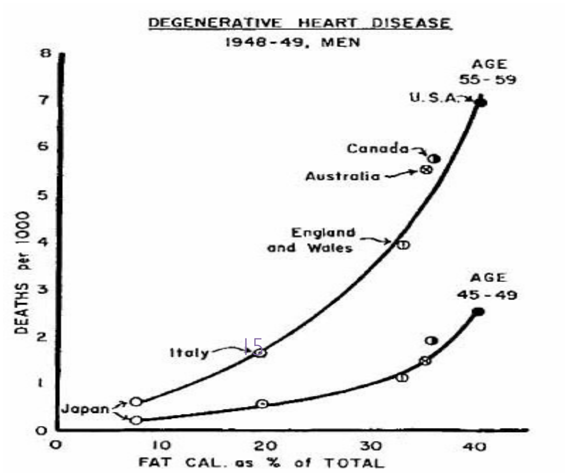
- Seven Countries Study (Ancel Keys)
  - » Started in 1958, followed13,000 subjects total for 5-40 years



http://en.wikipedia.org/wiki/Seven_Countries_Study

Data Science?

- Seven Countries Study (Ancel Keys)
  - » Started in 1958, followed13,000 subjects total for 5-40 years



Significant controversy
- Only studied subset of 21 countries with data
- Failed to consider other factors (e.g., per capita annual sugar consumption in pounds)

http://en.wikipedia.org/wiki/Seven_Countries_Study

Data Science?

- Nowcasting vs Forecasting

- Example – Google Flu Trends:
  - » February 2010 detected outbreak two weeks ahead of CDC data
  - » Initially 97% accurate but overestimated during 2011-13 including one interval in 2012-13 period where GFT was off by 2x
  - » New models are estimating which cities are most at risk for spread of the Ebola virus

https://www.google.org/flutrends/

Data Science?

elections2012

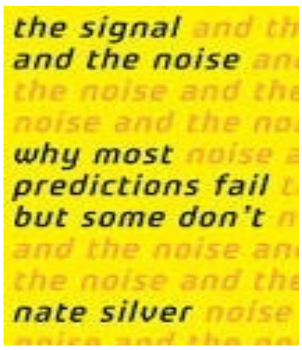
Live results | President | Senate | House | Governor | Choose your

Numbers nerd Nate Silver's forecasts prove all right on election night

FiveThirtyEight blogger predicted the outcome in all 50 states, assuming Barack Obama's Florida victory is confirmed

Luke Harding
guardian.co.uk, Wednesday 7 November 2012 10.45 EST

USA 2012 Presidential Election



http://www.theguardian.com/world/2012/nov/07/nate-silver-election-forecasts-right

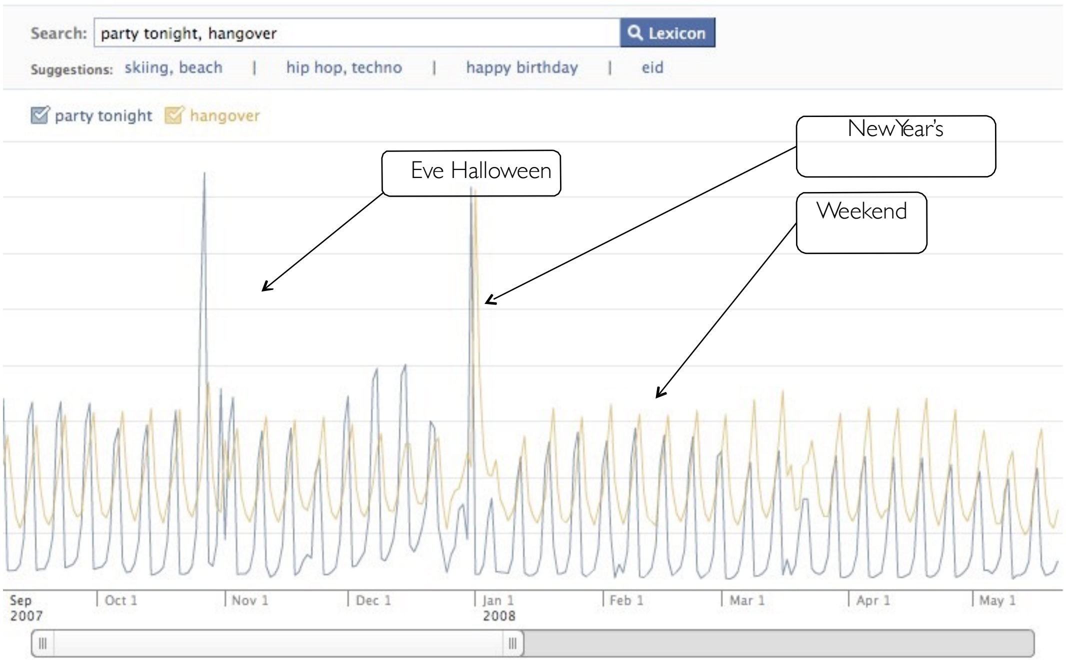
Data Science?

…that was just one of several ways that Mr. Obama's campaign operations, some unnoticed by Mr. Romney's aides in Boston, helped save the president's candidacy. In Chicago, the campaign recruited a team of behavioral scientists to build an extraordinarily sophisticated database

…that allowed the Obama campaign not only to alter the very nature of the electorate, making it younger and less white, but also to create a portrait of shifting voter allegiances. The power of this operation stunned Mr. Romney's aides on election night, as they saw voters they never even knew existed turn out in places like Osceola County, Fla.

New York Times, Wed Nov 7, 2012

Data Science?



Data Science?

Data Science?

Epidemiological modeling of online social network dynamics

John Cannarella[1], Joshua A. Spechler[1,*]
1 Department of Mechanical and Aerospace Engineering, Princeton University, Princeton, NJ, USA
* E-mail: Corresponding spechler@princeton.edu

Abstract

The last decade has seen the rise of immense online social networks (OSNs) such as MySpace and Facebook. In this paper we use epidemiological models to explain user adoption and abandonment of OSNs, where adoption is analogous to infection and abandonment is analogous to recovery. We modify the traditional SIR model of disease spread by incorporating infectious recovery dynamics such that contact between a recovered and infected member of the population is required for recovery. The proposed infectious recovery SIR model (irSIR model) is validated using publicly available Google search query data for "MySpace" as a case study of an OSN that has exhibited both adoption and abandonment phases. The irSIR model is then applied to search query data for "Facebook," which is just beginning to show the onset of an abandonment phase. Extrapolating the best fit model into the future predicts a rapid decline in Facebook activity in the next few years.

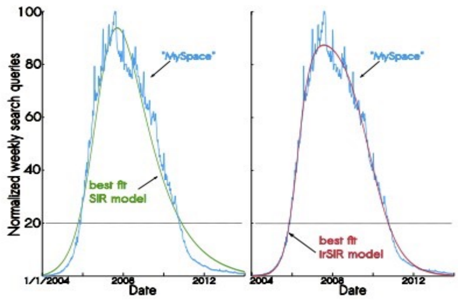
"Extrapolating the best fit model into the future predicts a rapid decline in Facebook activity in the next few years."

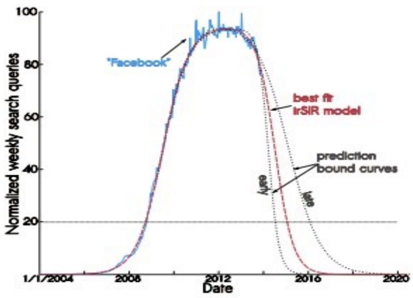
http://arxiv.org/abs/1401.4208

Data Science?

Google Trends searches for "MySpace"

Two Figures from the paper

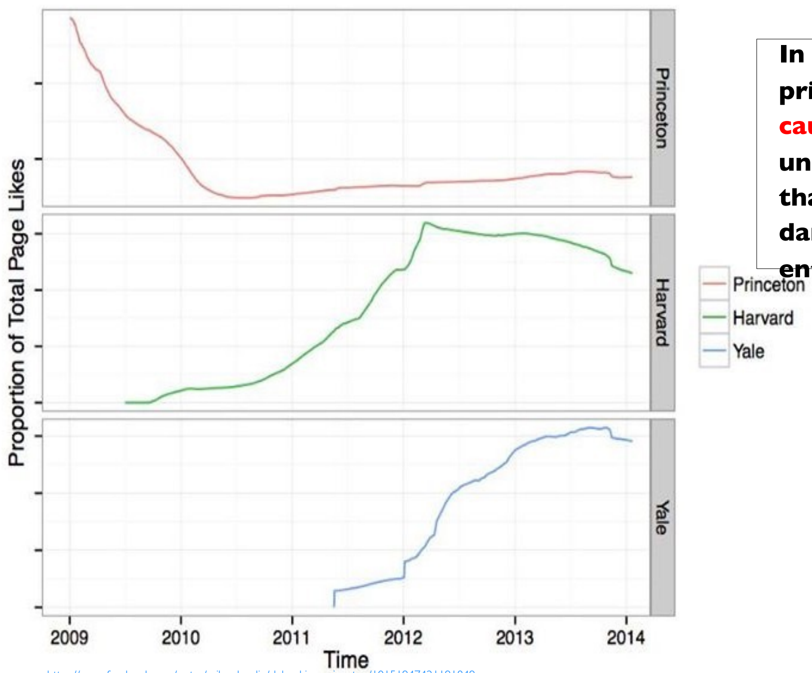




Searches for "Facebook"

http://arxiv.org/abs/1401.4208

Data Science?



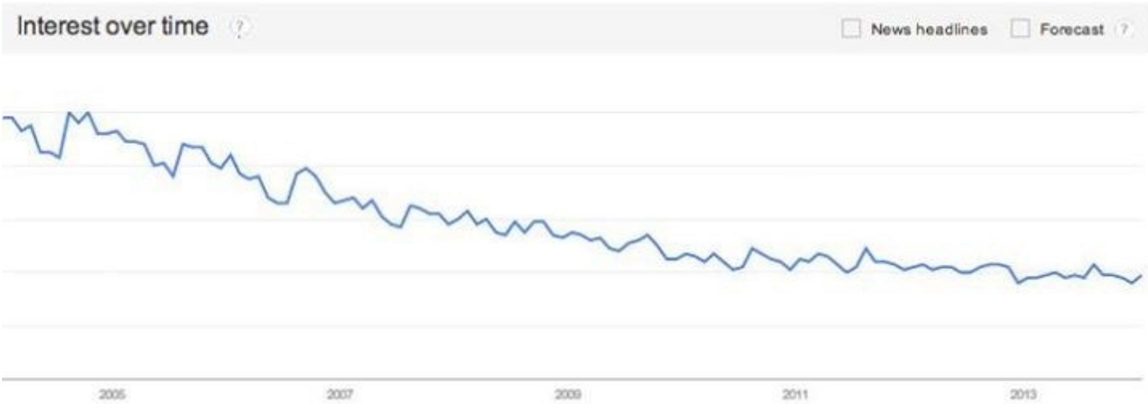
In keeping with the scientific principle "correlation equals causation," our research unequivocally demonstrated that Princeton may be in danger of disappearing entirely.

https://www.facebook.com/notes/mike-develin/debunking-princeton/10151947421191849

Data Science?

… and based on Princeton search trends:

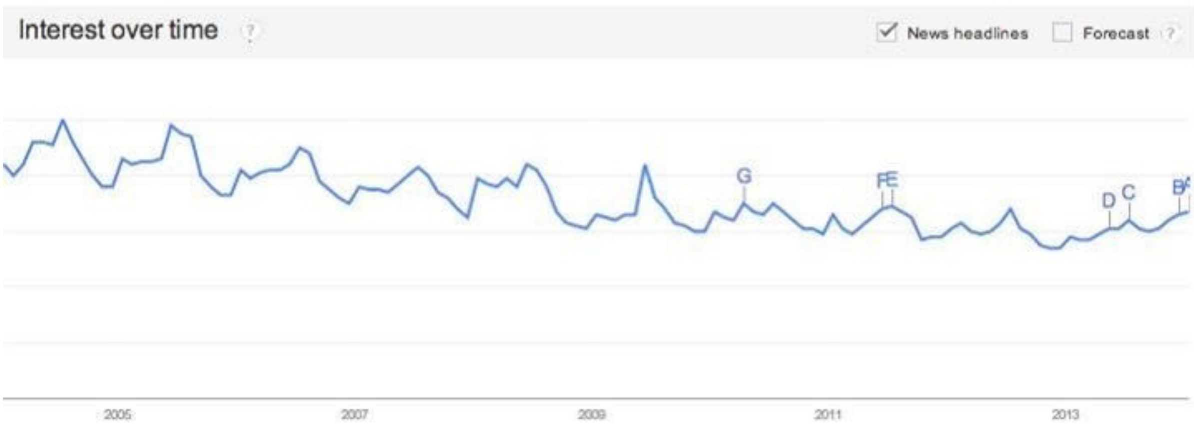
“This trend suggests that Princeton will have only half its current enrollment by 2018, and by 2021 it will have no students at all,…”



https://www.facebook.com/notes/mike-develin/debunking-princeton/10151947421191849

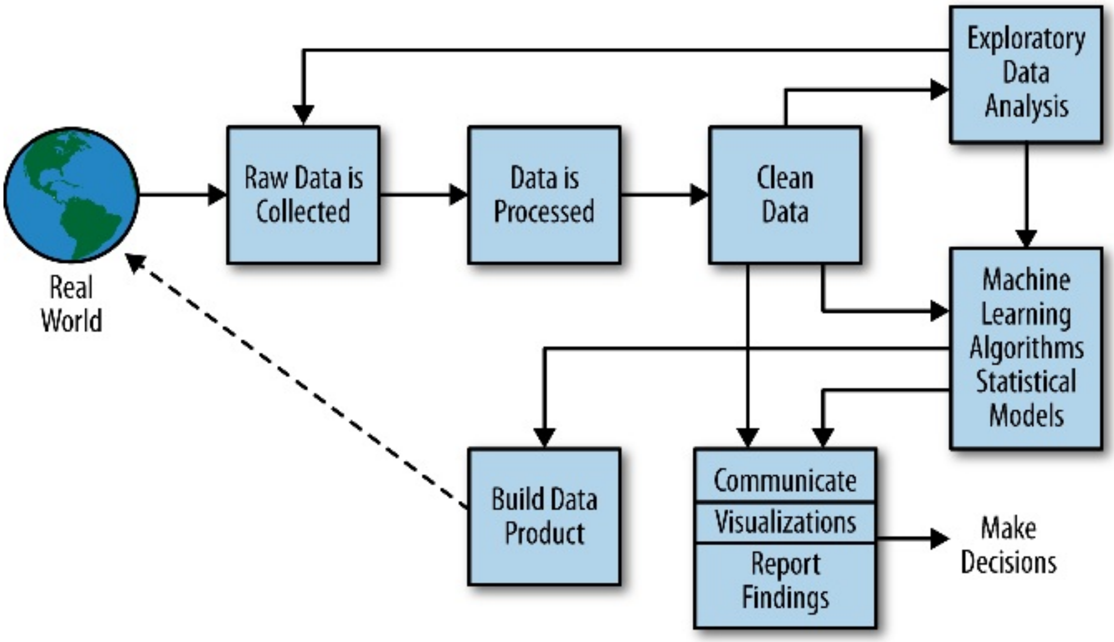
Data Science?

While we are concerned for Princeton University, we are even more concerned about the fate of the planet — Google Trends for “air” have also been declining steadily, and our projections show that by the year 2060 there will be no air left:



https://www.facebook.com/notes/mike-develin/debunking-princeton/10151947421191849

Data Science Process



R Programming

What is R?

- R is a system for statistical computation and graphics.
- It is heavily influenced by the **S** language
- **R** was initially written by **R**oss Ihaka and Robert Gentleman at the Department of Statistics of the University of Auckland in Auckland, New Zealand.
- The “**R** Core Team” maintain the source code for the software and release regular updates

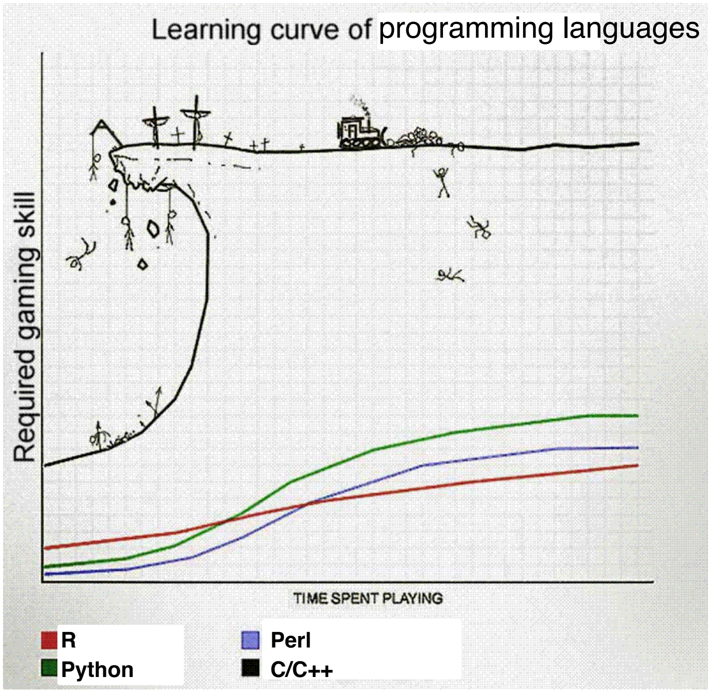
What is R?

- In addition, the R project is added to by many of its users, who write source code for many different types of analytical procedures
- Everything from analytical chemistry to epidemiology to linguistics
- Currently 6,000+ different user--written libraries available (http://cran.r-project.org)

Why use R?

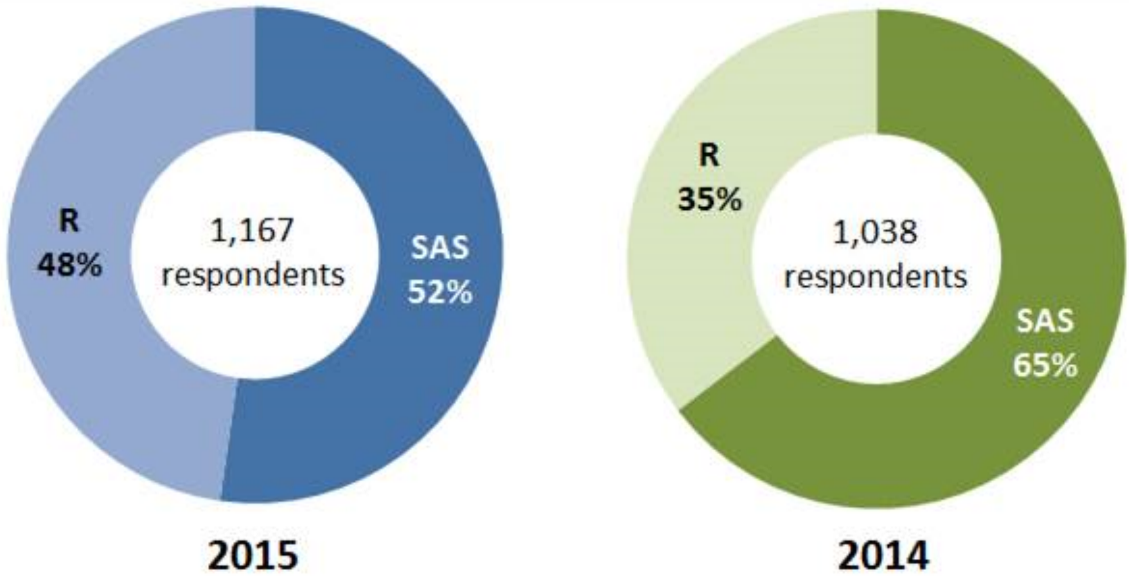
- R is Open-Source Software
- Many built-in functions and installable packages that will cover nearly every possible need
- R is an interpreted language
  - Code doesn't have to be compiled
- Interactive console makes testing and debugging easy
- Cons to using R
  - Slower than compiled languages
  - Can have runtime errors

Why use R?



http://www.vayapotra.es/wordpress/wp-content/uploads/2009/02/2rmqi6o.gif

R vs SAS



Tools

R
www.r-project.org

The engine*

RStudio
www.rstudio.org

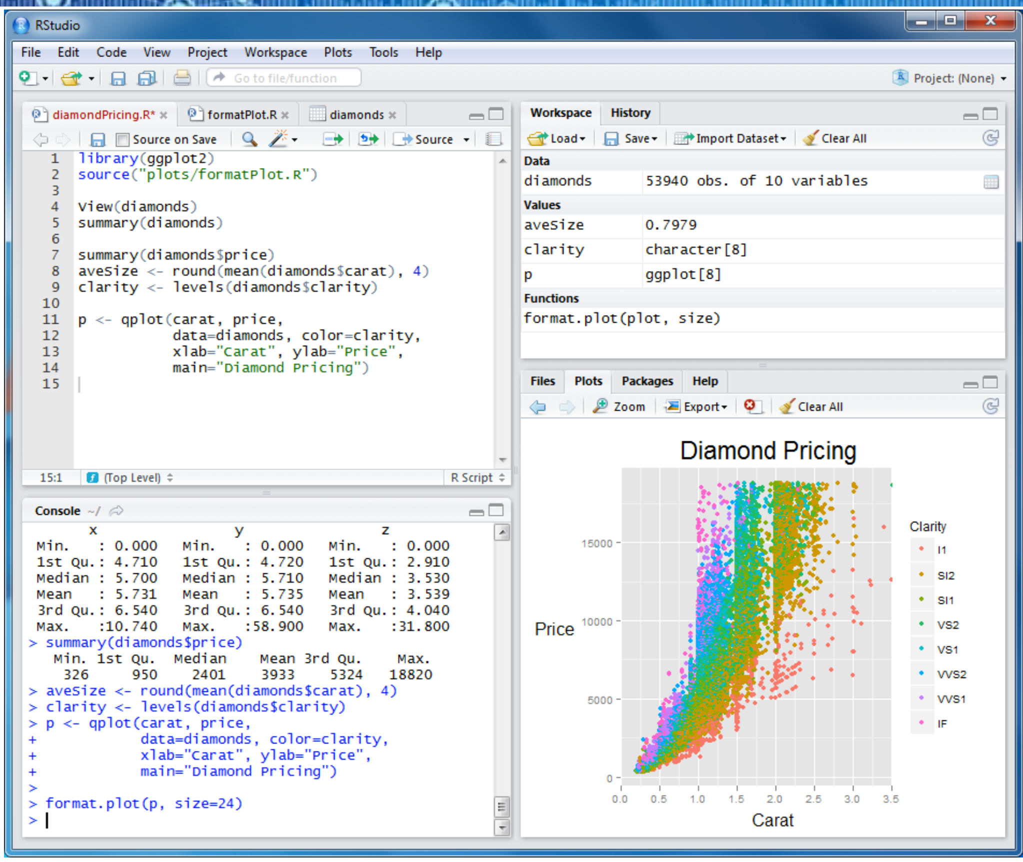
The pretty face**

** Many alternatives exist. Smallest learning curve.*
*** A few alternatives exist. This happens to be the easiest at the moment.*

RStudio Features

- Code completion
- Command history search
- Command history to R script / file
- Function extraction from Rscript
- Sweave and Knitr support

Introducing R-Studio



RStudio Panes

Rstudio has 4 main windows ('panes'):

- **The Source pane**: create a file that you can save and run later
- **The Console pane**: type or paste in commands to get output from R
- **The Workspace/History pane**: see a list of variables or previous commands
- **The Files/Plots/Packages/Help pane**: see plots, help pages, and other items in this window.

Source and Console Panes

Source window: Create a file here, so that you can save and run it later (or turn in as homework)

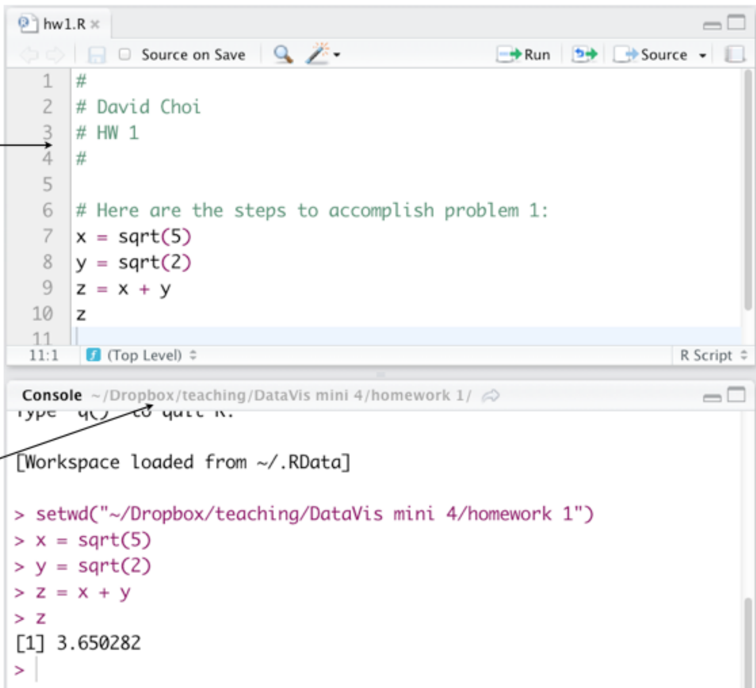
Any non-command line should start with a #

Console: Type or paste commands in here to get results from R

If you are loading a data file, you will need to be in the correct directory

> denotes that R is waiting for a command

+ denotes that R is waiting for you to finish the previous command (not shown here)



Console Panes

Variables

- Save the results of a command in memory by **giving it a name**:

```
> x = 3
> y = 2
> x
[1] 3
> y
[1] 2
> z = sqrt(x^2+y^2)
> z
[1] 3.605551
> x = 5
> z
[1] 3.605551
> z = sqrt(x^2+y^2)
> z
[1] 5.385165
> str = 'Hi there'
> str
[1] "Hi there"
> str2 = "It's cold outside"
> str2
[1] "It's cold outside"
```

Create `x` and `y`

`z` uses `x` and `y`:

Values are not linked; updating `x` doesn't change `z`

Redefine `z` using the new value for `x`

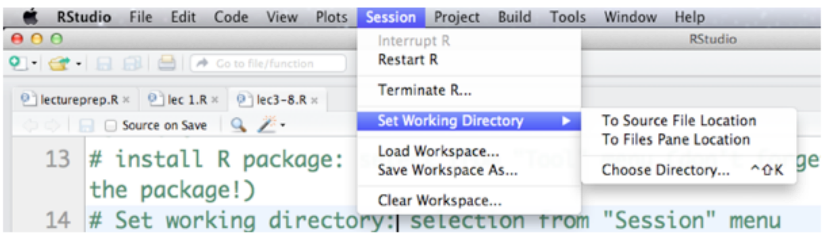
Use " " or ' ' to distinguish between variable names and regular text

Note:
x <- 3: same as x=3

RStudio Menu

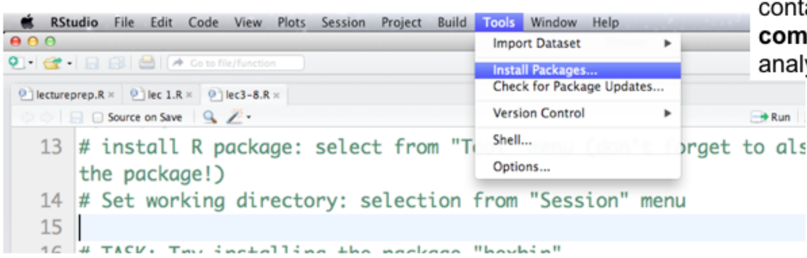
Two helpful menu items in Rstudio

- Set the current directory:

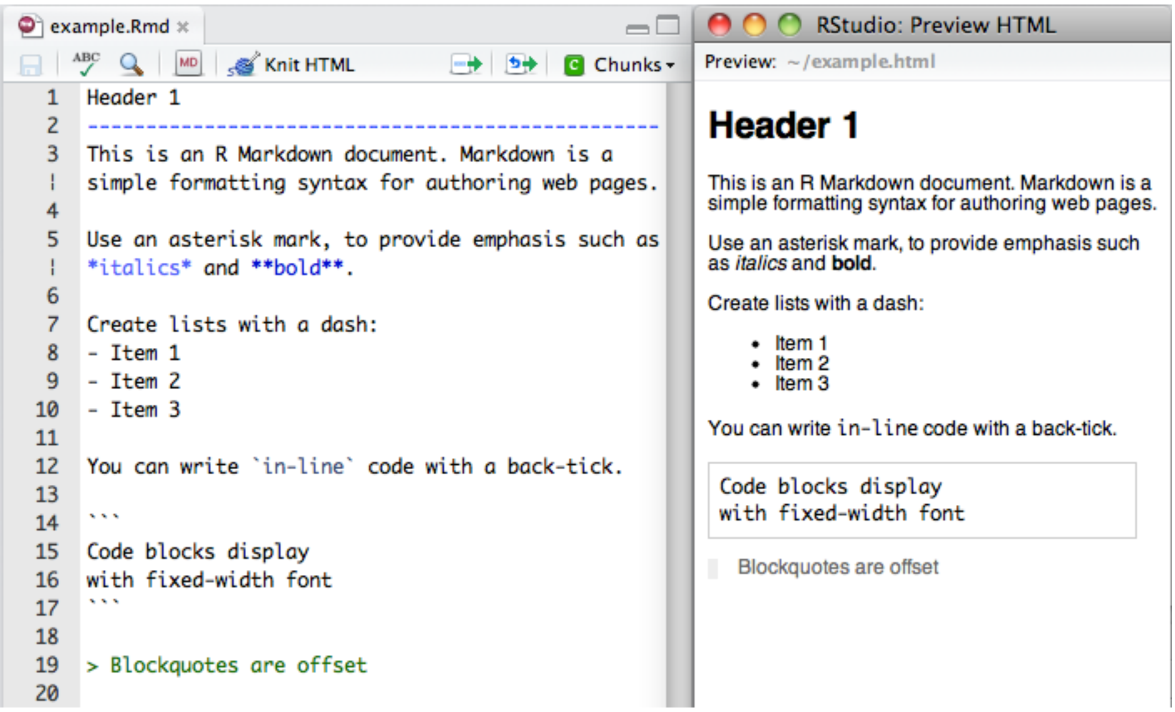


- Install a package

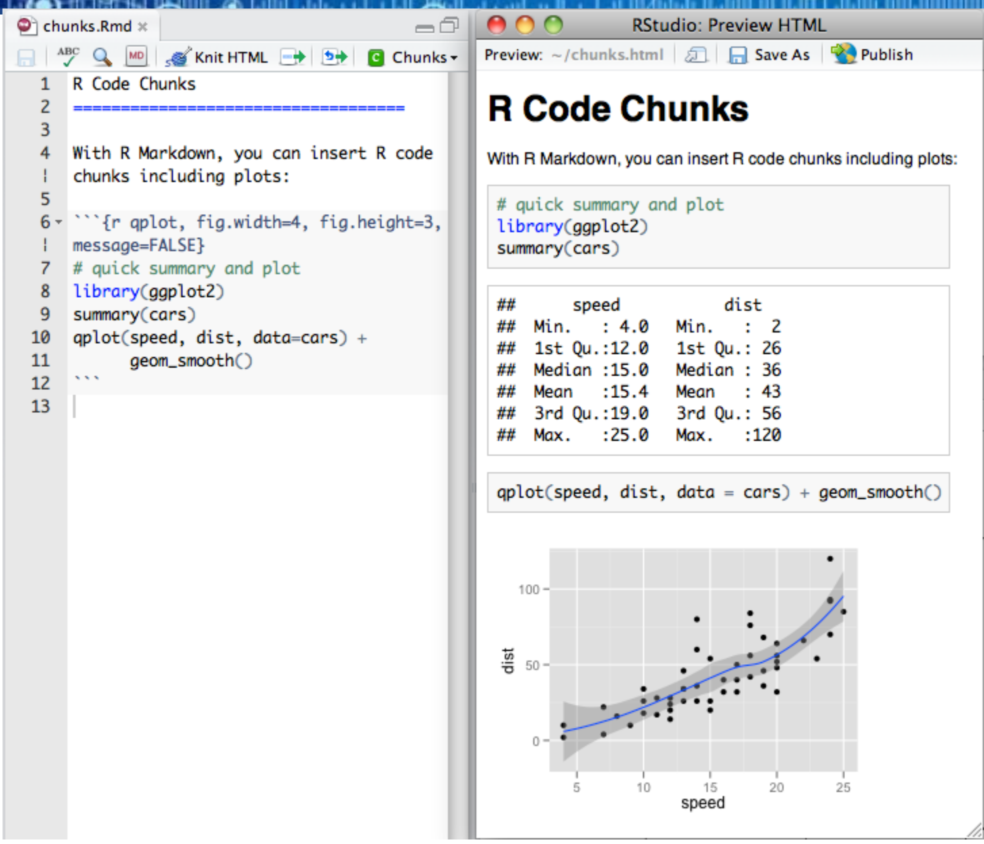
Packages are extensions to R, containing **new commands** for analysis or graphics



R Markdown



R Markdown



Workshop 1: Hello World

Open RStudio on your machine

- File > New File > R Markdown …

- Change summary(cars) in the first code block to print("Hello world!")

- Click Knit HTML to produce an HTML file.

- Save your Rmd file as helloworld.Rmd

Data Types

You'll encounter different kinds of data types

- **Booleans** Direct binary values: TRUE or FALSE in R
- **Integers**: whole numbers (positive, negative or zero)
- **Characters** fixed-length blocks of bits, with special coding; strings = sequences of characters
- **Floating point numbers**: a fraction (with a finite number of bits) times an exponent, like 1.87 * 10^6
- **Missing** or ill-defined values: NA, NaN, etc.

Operators (Functions)

Command	Description
`+,-,*,\`	add, subtract, multiply, divide
`^`	raise to the power of
`%%`	remainder after division (ex: `8 %% 3 = 2`)
`( )`	change the order of operations
`log(), exp()`	logarithms and exponents (ex: `log(10) = 2.302`)
`sqrt()`	square root
`round()`	round to the nearest whole number (ex: `round(2.3) = 2`)
`floor(), ceiling()`	round down or round up
`abs()`	absolute value

Operators

```
7 + 5 # Addition
```

```
[1] 12
```

```
7 - 5 # Subtraction
```

```
[1] 2
```

```
7 * 5 # Multiplication
```

```
[1] 35
```

```
7 ^ 5 # Exponentiation
```

```
[1] 16807
```

```r
7 / 5 # Division
```

```
[1] 1.4
```

```r
7 %% 5 # Modulus
```

```
[1] 2
```

```r
7 %/% 5 # Integer division
```

```
[1] 1
```


Operators

Comparisons are also **binary operators**; they take two objects, like numbers, and give a Boolean

```r
7 > 5
```

```
[1] TRUE
```

```r
7 < 5
```

```
[1] FALSE
```

```r
7 >= 7
```

```
[1] TRUE
```

```r
7 <= 5
```

```
[1] FALSE
```

```r
7 == 5
```

```
[1] FALSE
```

```r
7 != 5
```

```
[1] TRUE
```


Boolean Operators

Basically “and” and “or”:

```r
(5 > 7) & (6*7 == 42)
```

```
[1] FALSE
```

```r
(5 > 7) | (6*7 == 42)
```

```
[1] TRUE
```

Basic Math / Basic Logic

+ addition
- subtraction
* multiplication
/ division
% modulus (remainder)
^ to the power

`?Arithmetic`

- ! NOT
- & bitwise AND
- | bitwise OR
- && short circuit AND
- || short circuit OR
- == equality
- != NOT equality

`?Logic`

Also try `?Syntax`, `?Comparison`

More types

typeof() function returns the type

is.foo() functions return Booleans for whether the argument is of type foo

as.foo() (tries to) “cast” its argument to type foo — to translate it sensibly into a foo-type value

```r
typeof(7)
```

```
[1] "double"
```

```r
is.numeric(7)
```

```
[1] TRUE
```

```r
is.na(7)
```

```
[1] FALSE
```



```r
is.character(7)
```

```
[1] FALSE
```

```r
is.character("7")
```

```
[1] TRUE
```

```r
is.character("seven")
```

```
[1] TRUE
```

```r
is.na("seven")
```

```
[1] FALSE
```

Variables

We can give names to data objects; these give us **variables**

A few variables are built in:

```
pi
```

```
[1] 3.141593
```

Variables can be arguments to functions or operators, just like constants:

```
pi*10
```

```
[1] 31.41593
```

```
cos(pi)
```

```
[1] -1
```


Assignment Operator

Most variables are created with the **assignment operator**, <– or =

```
approx.pi <- 22 / 7
approx.pi
```

```
[1] 3.142857
```

```
diameter.in.cubits = 10
approx.pi*diameter.in.cubits
```

```
[1] 31.42857
```

Variables

- Using names and variables makes code: easier to design, easier to debug, less prone to bugs, easier to improve, and easier for others to read
- Avoid "magic constants"; use named variables
- Use descriptive variable names
  - Good: `num.students <- 35`
  - Bad: `ns <- 35`

Workspace

What names have you defined values for?

```
ls()
```

```
[1] "approx.pi"              "circumference.in.cubits"
[3] "diameter.in.cubits"
```

Getting rid of variables:

```
rm("circumference.in.cubits")
ls()
```

```
[1] "approx.pi"          "diameter.in.cubits"
```

Data Structure : Vector

- Group related data values into one object, a **data structure**
- A **vector** is a sequence of values, all of the same type
- `c()` function returns a vector containing all its arguments in order

```
students <- c("Sean", "Louisa", "Frank", "Farhad", "Li")
midterm <- c(80, 90, 93, 82, 95)
```

- Typing the variable name at the prompt causes it to display

```
students
```

```
[1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
```


Indexing Vector

- `vec[1]` is the first element, `vec[4]` is the 4th element of `vec`

```
students
```

```
[1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
```

```
students[4]
```

```
[1] "Farhad"
```

- `vec[-4]` is a vector containing all but the fourth element

```
students[-4]
```

```
[1] "Sean"   "Louisa" "Frank"  "Li"
```

Vector Arithmetic

Operators apply to vectors “pairwise” or “elementwise”:

```
final <- c(78, 84, 95, 82, 91) # Final exam scores
midterm # Midterm exam scores
```

```
[1] 80 90 93 82 95
```

```
midterm + final # Sum of midterm and final scores
```

```
[1] 158 174 188 164 186
```

```
(midterm + final)/2 # Average exam score
```

```
[1] 79 87 94 82 93
```

```
course.grades <- 0.4*midterm + 0.6*final # Final course grade
course.grades
```

```
[1] 78.8 86.4 94.2 82.0 92.6
```


Pairwise Comparison

Is the final score higher than the midterm score?

```
midterm
```

```
[1] 80 90 93 82 95
```

```
final
```

```
[1] 78 84 95 82 91
```

```
final > midterm
```

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

Boolean operators can be applied elementwise:

```
(final < midterm) & (midterm > 80)
```

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

Functions on Vectors

Command	Description
`sum(vec)`	sums up all the elements of `vec`
`mean(vec)`	mean of `vec`
`median(vec)`	median of `vec`
`min(vec), max(vec)`	the largest or smallest element of `vec`
`sd(vec), var(vec)`	the standard deviation and variance of `vec`
`length(vec)`	the number of elements in `vec`
`pmax(vec1, vec2), pmin(vec1, vec2)`	example: `pmax(quiz1, quiz2)` returns the higher of quiz 1 and quiz 2 for each student
`sort(vec)`	returns the `vec` in sorted order
`order(vec)`	returns the index that sorts the vector `vec`
`unique(vec)`	lists the unique elements of `vec`
`summary(vec)`	gives a five-number summary
`any(vec), all(vec)`	useful on Boolean vectors

Function on Vectors

```
course.grades
```

```
[1] 78.8 86.4 94.2 82.0 92.6
```

```
mean(course.grades) # mean grade
```

```
[1] 86.8
```

```
median(course.grades)
```

```
[1] 86.4
```

```
sd(course.grades) # grade standard deviation
```

```
[1] 6.625708
```

More on Functions

```
sort(course.grades)
```

```
[1] 78.8 82.0 86.4 92.6 94.2
```

```
max(course.grades) # highest course grade
```

```
[1] 94.2
```

```
min(course.grades) # lowest course grade
```

```
[1] 78.8
```


Referencing elements of Vectors

```
students
```

```
[1] "Sean"   "Louisa" "Frank"  "Farhad" "Li"
```

Vector of indices:

```
students[c(2,4)]
```

```
[1] "Louisa" "Farhad"
```

Vector of negative indices

```
students[c(-1,-3)]
```

```
[1] "Louisa" "Farhad" "Li"
```

More on Referencing

`which()` returns the `TRUE` indexes of a Boolean vector:

```
course.grades
```

```
[1] 78.8 86.4 94.2 82.0 92.6
```

```
a.threshold <- 90 # A grade = 90% or higher
course.grades >= a.threshold # vector of booleans
```

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

```
a.students <- which(course.grades >= a.threshold) # Applying which()
a.students
```

```
[1] 3 5
```

```
students[a.students] # Names of A students
```

```
[1] "Frank" "Li"
```


Workshop 2

Using the same Rmarkdown file from Workshop 1, do the following:-

1) Creating sequences

`:` **Colon operator:**

```
1:10 # Numbers 1 to 10
```

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

```
127:132 # Numbers 127 to 132
```

```
## [1] 127 128 129 130 131 132
```

Workshop 2

`seq` **function:** `seq(from, to, by)`

```
seq(1,10,1) # Numbers 1 to 10
```

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

```
seq(1,10,2) # Odd numbers from 1 to 10
```

```
## [1] 1 3 5 7 9
```

```
seq(2,10,2) # Even numbers from 2 to 10
```

```
## [1]  2  4  6  8 10
```


Workshop 2 : Use RStudio to answer questions below

(a) Use : to output the sequence of numbers from 3 to 12

(b) Use seq() to output the sequence of numbers from 3 to 30 in increments of 3

(c) Save the sequence from (a) as a variable x, and the sequence from (b) as a variable y. Output their product x*y

Difference between = and <-

- The operators <- and = assign into the environment in which they are evaluated.
- The operator <- can be used anywhere, whereas the operator = is only allowed at the top level (e.g., in the complete expression typed at the command prompt) or as one of the subexpressions in a braced list of expressions.

```
matrix(1,nrow=2)

matrix(1,nrow<-2)
```


Reading and Getting Data into R

Combine Command

```
c(1, 2, 3, 4)
c(item1, item2, item3, item4)
c("item1", "item2", "item3")
```

Scan Command

```
our.data = scan()

scan(what = 'character')
data5 = scan(sep = ',', what = 'char')
data6 = scan(file = 'data.txt')
```

Working Directory

```
getwd()
```

Reading Bigger Data Files

```
read.csv()
read.csv(file, sep = ',', header = TRUE, row.names)
fw = read.csv(file.choose())

my.ssv = read.table(file.choose(), header = TRUE)
my.tsv = read.delim(file.choose())
my.tsv = read.csv(file.choose(), sep = '\t')
my.tsv = read.table(file.choose(), header = TRUE,
sep = '\t')
```


Importing Data

- To import tabular data into R, we use the **`read.table()`** command

```
survey <- read.table("survey_data.csv", header=TRUE, sep=",")
```

- Let's parse this command one component at a time
  - The data is in a file called `survey_data.csv`, which is an online file
  - The file contains a `header` as its first row
  - The csv format means that the data is comma-separated, so `sep=","`
- Could've also used `read.csv()`, which is just `read.table()` with the preset `sep=","`

Exploring the Data

- R imports data into a `data.frame` object

```
class(survey)
```

```
[1] "data.frame"
```

- To view the first few rows of the data, use `head()`

```
head(survey, 3)
```

```
  Program              PriorExp      Rexperience OperatingSystem TVhours
1    MISM       Some experience       Never used         Windows       1
2   Other       Some experience Basic competence         Windows       8
3    MISM Extensive experience Basic competence         Windows       4
          Editor
1 Microsoft Word
2 Microsoft Word
3 Microsoft Word
```

- `head(data.frame, n)` returns the first `n` rows of the data frame
- In the Console, you can also use `View(survey)` to get a spreadsheet view

Simple Summary

- Use the `str()` function to get a simple summary of your data set

```
str(survey)
```

```
'data.frame':   31 obs. of  6 variables:
 $ Program        : Factor w/ 3 levels "MISM","Other",..: 1 2 1 3 2 3 2 1 3 3 ...
 $ PriorExp       : Factor w/ 3 levels "Extensive experience",..: 3 3 1 2 2 3 3 3
2 1 ...
 $ Rexperience    : Factor w/ 3 levels "Basic competence",..: 3 1 1 2 3 3 3 3 3 3
...
 $ OperatingSystem: Factor w/ 2 levels "Mac OS X","Windows": 2 2 2 2 2 1 1 2 1 2
...
 $ TVhours        : int  1 8 4 5 3 10 0 10 15 4 ...
 $ Editor         : Factor w/ 1 level "Microsoft Word": 1 1 1 1 1 1 1 1 1 1 ...
```

- This says that TVhours is a numeric variable, while all the rest are factors (categorical)

Another Summary

```r
summary(survey)
```

```
  Program                          PriorExp                     Rexperience
 MISM :10   Extensive experience    : 5   Basic competence     : 2
 Other: 6   Never programmed before:12   Installed on machine: 7
 PPM  :15   Some experience         :14   Never used           :22



 OperatingSystem     TVhours                     Editor
 Mac OS X:14      Min.   : 0.000   Microsoft Word:31
 Windows :17      1st Qu.: 1.000
                  Median : 4.000
                  Mean   : 5.742
                  3rd Qu.: 9.000
                  Max.   :40.000
```


Data Frames

- 2 Dimensional Objects, it has rows and columns. R treats the columns as separate samples or variables, rows represent the replicates or observations.
- To see what an R object is made up of, you can use `attributes()`

```r
attributes(survey)
```

```
$names
[1] "Program"          "PriorExp"          "Rexperience"       "OperatingSystem"
[5] "TVhours"          "Editor"

$class
[1] "data.frame"

$row.names
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
[24] 24 25 26 27 28 29 30 31
```

An R **data frame** is a *list* whose columns you can refer to by *name* or *index*

Data Frame dimensions

- We can use `nrow()` and `ncol` to determine the number of survey responses and the number of survey questions

```
nrow(survey) # Number of rows (responses)
```

```
[1] 31
```

```
ncol(survey) # Number of columns (questions)
```

```
[1] 6
```

- When writing reports, you will often want to say how large your sample size was
- To do this *inline*, use the syntax:

```
`r nrow(survey)`
```

- This allows us to write "31 students responded to the survey", and have the number displayed automatically change when `nrow(survey)` changes.

Indexing Data Frame

- There are many different ways of indexing the same piece of a data frame

```
survey[["Program"]]  # "Program" element
```

```
 [1] MISM  Other MISM  PPM   Other PPM   Other MISM  PPM   PPM   MISM 
[12] PPM   PPM   PPM   PPM   PPM   PPM   MISM  MISM  MISM  PPM   PPM  
[23] MISM  Other PPM   PPM   MISM  PPM   Other Other MISM 
Levels: MISM Other PPM
```

```
survey$Program # "Program" element
```

```
 [1] MISM  Other MISM  PPM   Other PPM   Other MISM  PPM   PPM   MISM 
[12] PPM   PPM   PPM   PPM   PPM   PPM   MISM  MISM  MISM  PPM   PPM  
[23] MISM  Other PPM   PPM   MISM  PPM   Other Other MISM 
Levels: MISM Other PPM
```

```
survey[,1] # Data from 1st column
```

```
 [1] MISM  Other MISM  PPM   Other PPM   Other MISM  PPM   PPM   MISM 
[12] PPM   PPM   PPM   PPM   PPM   PPM   MISM  MISM  MISM  PPM   PPM  
[23] MISM  Other PPM   PPM   MISM  PPM   Other Other MISM 
Levels: MISM Other PPM
```

More Indexing

- Note that single brackets and double brackets have different effects

```
survey[["Program"]]
```

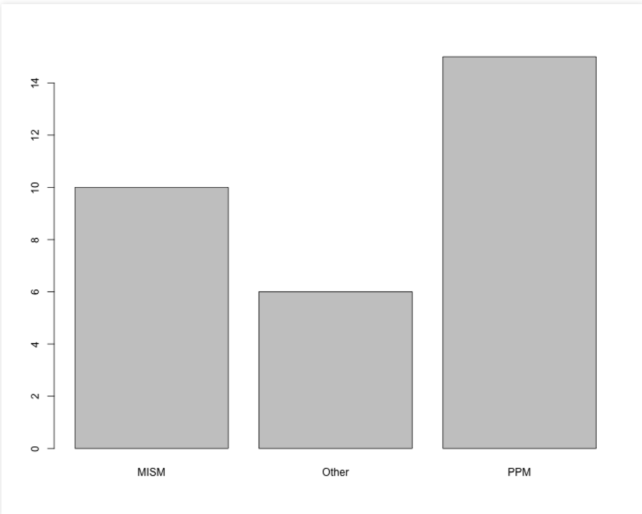
```
 [1] MISM  Other MISM  PPM   Other PPM   Other MISM  PPM   PPM   MISM 
[12] PPM   PPM   PPM   PPM   PPM   PPM   MISM  MISM  MISM  PPM   PPM  
[23] MISM  Other PPM   PPM   MISM  PPM   Other Other MISM 
Levels: MISM Other PPM
```

```
survey["Program"] # sub-data frame containing only "Program"
```

```
   Program
1     MISM
2    Other
3     MISM
4      PPM
5    Other
6      PPM
7    Other
8     MISM
9      PPM
10     PPM
11    MISM
12     PPM
13     PPM
14     PPM
```

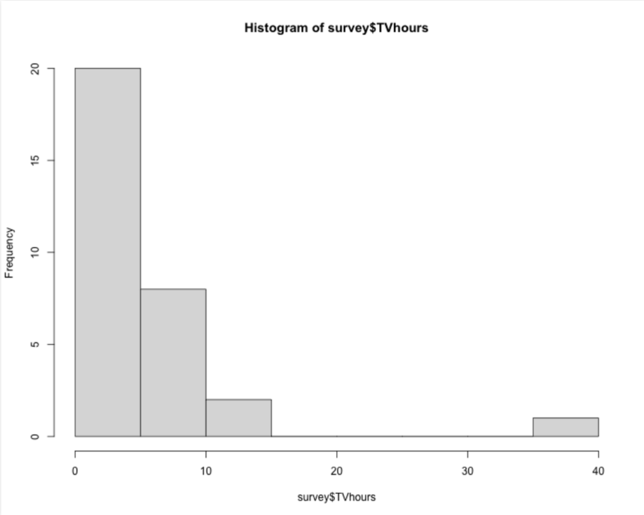

Bar Plot (Categorical Data)

```
plot(survey[["Program"]])
```



Histogram (Continuous Data)

```
hist(survey$TVhours, col="lightgray")
```



Indexing Multiple Columns

```
head(survey[,c(1,5)]) # Data from 1st and 5th columns
```

```
  Program TVhours
1    MISM       1
2   Other       8
3    MISM       4
4     PPM       5
5   Other       3
6     PPM      10
```

```
head(survey[c("Program", "Editor")]) # Data from "Program" and "Editor"
```

```
  Program         Editor
1    MISM Microsoft Word
2   Other Microsoft Word
3    MISM Microsoft Word
4     PPM Microsoft Word
5   Other Microsoft Word
6     PPM Microsoft Word
```

Indexing Row and Column

- Data frames have two dimensions to index across

```
survey[6,] # 6th row
```

```
  Program        PriorExp Rexperience OperatingSystem TVhours
6     PPM Some experience  Never used        Mac OS X      10
          Editor
6 Microsoft Word
```

```
survey[6,5] # row 6, column 5
```

```
[1] 10
```

```
survey[6, "Program"] # Program of 6th survey respondent
```

```
[1] PPM
Levels: MISM Other PPM
```

```
survey[["Program"]][6]  # Program of 6th survey respondent
```

```
[1] PPM
Levels: MISM Other PPM
```


More Indexing

- We can use this operator for indexing

```
survey[1:3,]  # equivalent to head(survey, 3)
```

```
  Program             PriorExp      Rexperience OperatingSystem TVhours
1    MISM      Some experience       Never used         Windows       1
2   Other      Some experience Basic competence         Windows       8
3    MISM Extensive experience Basic competence         Windows       4
          Editor
1 Microsoft Word
2 Microsoft Word
3 Microsoft Word
```

```
survey[3:5, c(1,5)]
```

```
  Program TVhours
3    MISM       4
4     PPM       5
5   Other       3
```

Subsets of Data

- We are often interested in learning something a specific subset of the data

```
survey[survey$Program=="MISM",] # Data from the MISM students
```

```
   Program              PriorExp          Rexperience OperatingSystem
1     MISM       Some experience           Never used         Windows
3     MISM Extensive experience     Basic competence         Windows
8     MISM       Some experience           Never used         Windows
11    MISM Extensive experience Installed on machine        Mac OS X
18    MISM Extensive experience           Never used        Mac OS X
19    MISM Extensive experience           Never used         Windows
20    MISM       Some experience           Never used         Windows
23    MISM       Some experience           Never used         Windows
27    MISM       Some experience           Never used        Mac OS X
31    MISM       Some experience Installed on machine         Windows
   TVhours          Editor
1        1 Microsoft Word
3        4 Microsoft Word
8       10 Microsoft Word
11       0 Microsoft Word
18       3 Microsoft Word
19       0 Microsoft Word
20       1 Microsoft Word
23       3 Microsoft Word
27       0 Microsoft Word
31       0 Microsoft Word
```


More subsets example

- Let's pull all of the PPM students who have never used R before

```
survey[survey$Program=="PPM" & survey$Rexperience=="Never used",]
```

```
   Program               PriorExp Rexperience OperatingSystem TVhours
6      PPM        Some experience  Never used        Mac OS X      10
9      PPM Never programmed before  Never used        Mac OS X      15
10     PPM   Extensive experience  Never used         Windows       4
12     PPM Never programmed before  Never used         Windows       0
13     PPM        Some experience  Never used        Mac OS X      10
14     PPM Never programmed before  Never used        Mac OS X       4
15     PPM        Some experience  Never used         Windows      10
16     PPM        Some experience  Never used        Mac OS X       2
22     PPM Never programmed before  Never used         Windows       7
25     PPM Never programmed before  Never used        Mac OS X       6
           Editor
6  Microsoft Word
9  Microsoft Word
10 Microsoft Word
12 Microsoft Word
13 Microsoft Word
14 Microsoft Word
15 Microsoft Word
16 Microsoft Word
22 Microsoft Word
25 Microsoft Word
```

Subset

- When the subset conditions get long or messy, it is preferable to use the `subset()` function
- Here's an example of selecting the OperatingSystem and TVhours responses from all of the students who are either in PPM or Other and who listed their R experience as "Basic competence".

```
subset(survey, select=c("OperatingSystem", "TVhours"),  subset=(Program == "PPM"
| Program == "Other") & Rexperience == "Basic competence")
```

```
  OperatingSystem TVhours
2         Windows       8
```


Calculation from subset

```
mean(survey$TVhours[survey$Program == "PPM"]) # Average time PPM's spent watching
TV
```

```
[1] 8.8
```

```
mean(survey$TVhours[survey$Program == "MISM"]) # Average time MISM's spent
watching TV
```

```
[1] 2.2
```

```
mean(survey$TVhours[survey$Program == "Other"]) # Average time "Others" spent
watching TV
```

```
[1] 4
```

Coding Style

- Coding style (and code commenting) will become increasingly more important as we get into more advanced and involved programming tasks
- A few R "style guides" exist:
  - http://r-pkgs.had.co.nz/style.html
  - Google's R Style Guide (https://google.github.io/styleguide/Rguide.xml)

Example Style Guide

Assignment operator. USE **<-**

```
student.names <- c("Eric", "Hao", "Jennifer")  # Good
student.names = c("Eric", "Hao", "Jennifer") # Bad
```

- Note: When specifying function arguments, only = is valid

```
sort(tv.hours, decreasing=TRUE) # Good
sort(tv.hours, decreasing<-TRUE) # Bad!!
```

Style Guide : Variable Name

- To make code easy to read, debug, and maintain, you should use **concise** but **descriptive** variable names
- Terms in variable names should be separated by _ or .

```
# Accepted
day_one    day.one    day_1    day.1    day1

# Bad
d1    DayOne    dayone

# Can be made more concise:
first.day.of.the.month
```

- Avoid using variable names that are already pre-defined in R

```
# EXTREMELY bad:
c    T    pi    sum    mean
```


Workshop 3: Importing and Indexing data

Data practice

survey <- read.table("survey.csv");

Use command to answer these questions?

(a) How many survey respondents are from MISM or Other?

(b) What % of survey respondents are from PPM?

Index practice

(a) Use $ notation to pull the OperatingSystem column from the survey data

(b) Do the same thing with [,] notation, referring to OperatingSystem by name

(c) Repeat part (b), this time referring to OperatingSystem by column number

More on Data Frame

```
library(MASS)
head(Cars93, 3)
```

```
  Manufacturer   Model    Type Min.Price Price Max.Price MPG.city
1        Acura Integra   Small      12.9  15.9      18.8       25
2        Acura  Legend Midsize      29.2  33.9      38.7       18
3         Audi      90 Compact      25.9  29.1      32.3       20
  MPG.highway            AirBags DriveTrain Cylinders EngineSize
1          31               None      Front         4        1.8
2          25 Driver & Passenger      Front         6        3.2
3          26        Driver only      Front         6        2.8
  Horsepower  RPM Rev.per.mile Man.trans.avail Fuel.tank.capacity
1        140 6300         2890             Yes               13.2
2        200 5500         2335             Yes               18.0
3        172 5500         2280             Yes               16.9
  Passengers Length Wheelbase Width Turn.circle Rear.seat.room
1          5    177       102    68          37           26.5
2          5    195       115    71          38           30.0
3          5    180       102    67          37           28.0
  Luggage.room Weight  Origin          Make
1           11   2705 non-USA Acura Integra
2           15   3560 non-USA  Acura Legend
3           14   3375 non-USA       Audi 90
```


Add column using “Transform”

- `transform()` returns a new data frame with columns modified or added as specified by the function call

```
Cars93.metric <- transform(Cars93,
                           KMPL.city = 0.425 * MPG.city,
                           KMPL.highway = 0.425 * MPG.highway)
tail(names(Cars93.metric))
```

```
[1] "Luggage.room" "Weight"       "Origin"       "Make"
[5] "KMPL.city"    "KMPL.highway"
```

- Our data frame has two new columns, giving the fuel consumption in km/l

Another Approach

```
KMPL.city.2 <- 0.425 * Cars93$MPG.city
# Add a new column called KMPL.city.2
Cars93.metric$KMPL.city.2 <- KMPL.city.2
tail(names(Cars93.metric))
```

```
[1] "Weight"       "Origin"       "Make"         "KMPL.city"
[5] "KMPL.highway" "KMPL.city.2"
```

- Let's check that both approaches did the same thing

```
identical(Cars93.metric$KMPL.city, Cars93.metric$KMPL.city.2)
```

```
[1] TRUE
```


Changing level of a factor

```
manufacturer <- Cars93$Manufacturer
head(manufacturer, 10)
```

```
 [1] Acura    Acura    Audi     Audi     BMW      Buick    Buick
 [8] Buick    Buick    Cadillac
32 Levels: Acura Audi BMW Buick Cadillac Chevrolet Chrylser ... Volvo
```

We'll use the `mapvalues(x, from, to)` function from the `plyr` library.

```
library(plyr)

# Map Chevrolet, Pontiac and Buick to GM
manufacturer.combined <- mapvalues(manufacturer,
                                   from = c("Chevrolet", "Pontiac", "Buick"),
                                   to = rep("GM", 3))

head(manufacturer.combined, 10)
```

```
 [1] Acura    Acura    Audi     Audi     BMW      GM       GM
 [8] GM       GM       Cadillac
30 Levels: Acura Audi BMW GM Cadillac Chrylser Chrysler Dodge ... Volvo
```

Another example

- A lot of data comes with integer encodings of levels
- You may want to convert the integers to more meaningful values for the purpose of your analysis
- Let's pretend that in the class survey 'Program' was coded as an integer with 1 = MISM, 2 = Other, 3 = PPM
- Here's how we would get back the program codings using the transform(), as.factor() and mapvalues() functions


```r
survey <- transform(survey, Program = as.factor(mapvalues(Program, c(1, 2, 3),
c("MISM", "Other", "PPM"))))
head(survey)
```

```
  Program                PriorExp           Rexperience OperatingSystem
1    MISM         Some experience            Never used         Windows
2   Other         Some experience      Basic competence         Windows
3    MISM    Extensive experience      Basic competence         Windows
4     PPM Never programmed before Installed on machine         Windows
5   Other Never programmed before            Never used         Windows
6     PPM         Some experience            Never used        Mac OS X
  TVhours         Editor
1       1 Microsoft Word
2       8 Microsoft Word
3       4 Microsoft Word
4       5 Microsoft Word
5       3 Microsoft Word
6      10 Microsoft Word
```

table() function

- The `table()` function builds **contingency tables** showing counts at each combination of factor levels

```
table(Cars93$AirBags)
```

```
Driver & Passenger          Driver only                 None 
                16                   43                   34 
```

```
table(Cars93$Origin)
```

```
    USA non-USA 
     48      45 
```

```
table(Cars93$AirBags, Cars93$Origin)
```

```
                     USA non-USA
  Driver & Passenger   9       7
  Driver only         23      20
  None                16      18
```

- Looks like US and non-US cars had about the same distribution of AirBag types

more table()

- When `table()` is supplied a data frame, it produces contingency tables for all combinations of factors

```
head(Cars93[c("AirBags", "Origin")], 3)
```

```
             AirBags  Origin
1               None non-USA
2 Driver & Passenger non-USA
3        Driver only non-USA
```

```
table(Cars93[c("AirBags", "Origin")])
```

```
                    Origin
AirBags              USA non-USA
  Driver & Passenger   9       7
  Driver only         23      20
  None                16      18
```

Basic of Lists

A list is a **data structure** that can be used to store **different kinds** of data

- Recall: a vector is a data structure for storing *similar kinds of data*
- To better understand the difference, consider the following example.

```
my.vector.1 <- c("Michael", 165, TRUE) # (name, weight, is.male)
my.vector.1
```

```
[1] "Michael" "165"     "TRUE"
```

```
typeof(my.vector.1)  # All the elements are now character strings!
```

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


List vs Vector

```
my.vector.2 <- c(FALSE, TRUE, 27) # (is.male, is.citizen, age)
typeof(my.vector.2)
```

```
[1] "double"
```

- Vectors expect elements to be all of the same type (e.g., `Boolean`, `numeric`, `character`)
- When data of different types are put into a vector, the R converts everything to a common type

Lists

- To store data of different types in the same object, we use lists
- Simple way to build lists: use `list()` function

```
my.list <- list("Michael", 165, TRUE)
my.list
```

```
[[1]]
[1] "Michael"

[[2]]
[1] 165

[[3]]
[1] TRUE
```

```
sapply(my.list, typeof)
```

```
[1] "character" "double"    "logical"
```


Named Elements

```
patient.1 <- list(name="Michael", weight=165, is.male=TRUE)
patient.1
```

```
$name
[1] "Michael"

$weight
[1] 165

$is.male
[1] TRUE
```

Referencing element inside list

```
patient.1$name # Get "name" element (returns a string)
```

```
[1] "Michael"
```

```
patient.1[["name"]] # Get "name" element (returns a string)
```

```
[1] "Michael"
```

```
patient.1["name"] # Get "name" slice (returns a sub-list)
```

```
$name
[1] "Michael"
```

```
c(typeof(patient.1$name), typeof(patient.1["name"]))
```

```
[1] "character" "list"
```


Function

- We have used a lot of built-in functions: `mean()`, `subset()`, `plot()`, `read.table()`...
- An important part of programming and data analysis is to write custom functions
- Functions help make code **modular**
- Functions make debugging easier
- Remember: this entire class is about applying *functions* to *data*

what is a function?

A function is a machine that turns **input objects** (arguments) into an **output object** (return value) according to a definite rule.

- Let's look at a really simple function

```
addOne <- function(x) {
  x + 1
}
```

- `x` is the **argument** or **input**

- The function **output** is the input `x` incremented by 1

```
addOne(12)
```

```
[1] 13
```


Another example

- Here's a function that returns a % given a numerator, denominator, and desired number of decimal values

```
calculatePercentage <- function(x, y, d) {
  decimal <- x / y  # Calculate decimal value
  round(100 * decimal, d)  # Convert to % and round to d digits
}

calculatePercentage(27, 80, 1)
```

```
[1] 33.8
```

- If you're calculating several %'s for your report, you should use this kind of function instead of repeatedly copying and pasting code

Function return a list

- Here's a function that takes a person's full name (FirstName LastName), weight in lb and height in inches and converts it into a list with the person's first name, person's last name, weight in kg, height in m, and BMI.

```
createPatientRecord <- function(full.name, weight, height) {
  name.list <- strsplit(full.name, split=" ")[[1]]
  first.name <- name.list[1]
  last.name <- name.list[2]
  weight.in.kg <- weight / 2.2
  height.in.m <- height * 0.0254
  bmi <- weight.in.kg / (height.in.m ^ 2)
  list(first.name=first.name, last.name=last.name, weight=weight.in.kg, 
height=height.in.m,
       bmi=bmi)
}
```


Try out function

```
createPatientRecord("Michael Smith", 185, 12 * 6 + 1)
```

```
$first.name
[1] "Michael"

$last.name
[1] "Smith"

$weight
[1] 84.09091

$height
[1] 1.8542

$bmi
[1] 24.45884
```

Another example

- Calculate mean, median and standard deviation

```
threeNumberSummary <- function(x) {
  c(mean=mean(x), median=median(x), sd=sd(x))
}
x <- rnorm(100, mean=5, sd=2) # Vector of 100 normals with mean 5 and sd 2
threeNumberSummary(x)
```

```
    mean   median       sd
4.926875 5.050984 1.953703
```


If-else statement

- Oftentimes we want our code to have different effects depending on the features of the input
- Example: Calculating a student's letter grade
  - If grade >= 90, assign A
  - Otherwise, if grade >= 80, assign B
  - Otherwise, if grade >= 70, assign C
  - In all other cases, assign F
- To code this up, we use if-else statements

If-else example

```
calculateLetterGrade <- function(x) {
  if(x >= 90) {
    grade <- "A"
  } else if(x >= 80) {
    grade <- "B"
  } else if(x >= 70) {
    grade <- "C"
  } else {
    grade <- "F"
  }
  grade
}

course.grades <- c(92, 78, 87, 91, 62)
sapply(course.grades, FUN=calculateLetterGrade)
```

```
[1] "A" "C" "B" "A" "F"
```


return()

- In the previous examples we specified the output simply by writing the output variable as the last line of the function
- More explicitly, we can use the `return()` function

```
addOne <- function(x) {
  return(x + 1)
}

addOne(12)
```

```
[1] 13
```

- We will generally avoid the `return()` function, but you can use it if necessary or if it makes writing a particular function easier.

Workshop 4 : Tranform

For the first two problems we'll use the Cars93 data set from the MASS library.

```r
library(MASS)
```

1. **Manipulating data frames**

Use the **transform() and log()** functions to create a new data frame called Cars93.log that has MPG.highway and MPG.city replaced with log(MPG.highway) and log(MPG.city).

2. **Functions, lists, and if-else**

(a) Write a function called **isPassingGrade** whose input x is a number, and which returns FALSE if x is lower than 50 and TRUE otherwise.

(b) Write a function called **sendMessage** whose input x is a number, and which prints **Congratulations** if **isPassingGrade(x)** is TRUE and prints **Oh no!** if **isPassingGrade(x)** is FALSE.

Data Wrangling and Visualization

Common Problem

- One of the most common problems you'll encounter when importing manually-entered data is inconsistent data types within columns
- For a simple example, let's look at TVhours column in a messy version of the survey data

```
survey.messy <- read.csv("survey_messy.csv", header=TRUE)
survey.messy$TVhours
```

```
 [1] 1                 8h                4                 5
 [5] 3                 ~10               0                 10
 [9] 15 (incl movies) 4                 0                 0
[13] 10                4                 10                2hours
[17] 5                 3                 0                 1
[21] 2                 7                 3                 10
[25] 6.5               40                0                 12
[29] adfjalkj          3                 0
16 Levels: ~10 0 1 10 12 15 (incl movies) 2 2hours 3 4 40 5 6.5 7 ... adfjalkj
```


What are the problems?

```
str(survey.messy)
```

```
'data.frame':	31 obs. of  6 variables:
 $ Program        : Factor w/ 3 levels "MISM","Other",..: 1 2 1 3 2 3 2 1 3 3 ...
 $ PriorExp       : Factor w/ 3 levels "Extensive experience",..: 3 3 1 2 2 3 3 3 
2 1 ...
 $ Rexperience    : Factor w/ 3 levels "Basic competence",..: 3 1 1 2 3 3 3 3 3 3 
...
 $ OperatingSystem: Factor w/ 2 levels "Mac OS X","Windows": 2 2 2 2 2 1 1 2 1 2 
...
 $ TVhours        : Factor w/ 16 levels "~10","0","1",..: 3 15 10 12 9 1 2 4 6 10 
...
 $ Editor         : Factor w/ 1 level "Microsoft Word": 1 1 1 1 1 1 1 1 1 1 ...
```

- Several of the entries have non-numeric values in them (they contain strings)
- As a result, `TVhours` is being imported as `factor`

Fix the type

```
as.character(tv.hours.messy)[1:30]
```

```
 [1] "1"              "8h"             "4"
 [4] "5"              "3"              "~10"
 [7] "0"              "10"             "15 (incl movies)"
[10] "4"              "0"              "0"
[13] "10"             "4"              "10"
[16] "2hours"         "5"              "3"
[19] "0"              "1"              "2"
[22] "7"              "3"              "10"
[25] "6.5"            "40"             "0"
[28] "12"             "adfjalkj"       "3"
```

```
as.numeric(as.character(tv.hours.messy))[1:30]
```

```
 [1]  1.0   NA  4.0  5.0  3.0   NA  0.0 10.0   NA  4.0  0.0  0.0 10.0  4.0
[15] 10.0   NA  5.0  3.0  0.0  1.0  2.0  7.0  3.0 10.0  6.5 40.0  0.0 12.0
[29]   NA  3.0
```

```
typeof(as.numeric(as.character(tv.hours.messy)))  # Success!! (Almost...)
```

```
[1] "double"
```


A small improvement

- All the corrupted cells now appear as `NA`, which is R's missing indicator
- We can do a little better by cleaning up the vector once we get it to `character` form

```
tv.hours.strings <- as.character(tv.hours.messy)
tv.hours.strings
```

```
 [1] "1"              "8h"             "4"
 [4] "5"              "3"              "~10"
 [7] "0"              "10"             "15 (incl movies)"
[10] "4"              "0"              "0"
[13] "10"             "4"              "10"
[16] "2hours"         "5"              "3"
[19] "0"              "1"              "2"
[22] "7"              "3"              "10"
[25] "6.5"            "40"             "0"
[28] "12"             "adfjalkj"       "3"
[31] "0"
```

Deleting non-numeric (or .) characters

```
tv.hours.strings
```

```
 [1] "1"                 "8h"                "4"
 [4] "5"                 "3"                 "~10"
 [7] "0"                 "10"                "15 (incl movies)"
[10] "4"                 "0"                 "0"
[13] "10"                "4"                 "10"
[16] "2hours"            "5"                 "3"
[19] "0"                 "1"                 "2"
[22] "7"                 "3"                 "10"
[25] "6.5"               "40"                "0"
[28] "12"                "adfjalkj"          "3"
[31] "0"
```

```
# Use gsub() to replace everything except digits and '.' with a blank ""
gsub("[^0-9.]", "", tv.hours.strings)
```

```
 [1] "1"   "8"   "4"   "5"   "3"   "10"  "0"   "10"  "15"  "4"   "0"
[12] "0"   "10"  "4"   "10"  "2"   "5"   "3"   "0"   "1"   "2"   "7"
[23] "3"   "10"  "6.5" "40"  "0"   "12"  ""    "3"   "0"
```


Redo

```
tv.hours.messy
```

```
 [1] 1                8h               4                5
 [5] 3                ~10              0                10
 [9] 15 (incl movies) 4                0                0
[13] 10               4                10               2hours
[17] 5                3                0                1
[21] 2                7                3                10
[25] 6.5              40               0                12
[29] adfjalkj         3                0
16 Levels: ~10 0 1 10 12 15 (incl movies) 2 2hours 3 4 40 5 6.5 7 ... adfjalkj
```

```
tv.hours.clean <- as.numeric(gsub("[^0-9.]", "", tv.hours.strings))
tv.hours.clean
```

```
 [1]  1.0  8.0  4.0  5.0  3.0 10.0  0.0 10.0 15.0  4.0  0.0  0.0 10.0  4.0
[15] 10.0  2.0  5.0  3.0  0.0  1.0  2.0  7.0  3.0 10.0  6.5 40.0  0.0 12.0
[29]   NA  3.0  0.0
```

Another approach

- We can also handle this problem by setting `stringsAsFactors = FALSE` when importing our data.

```
survey.messy <- read.csv("survey_messy.csv", header=TRUE, stringsAsFactors=FALSE)
str(survey.messy)
```

```
'data.frame':	31 obs. of  6 variables:
 $ Program        : chr  "MISM" "Other" "MISM" "PPM" ...
 $ PriorExp       : chr  "Some experience" "Some experience" "Extensive
experience" "Never programmed before" ...
 $ Rexperience    : chr  "Never used" "Basic competence" "Basic competence"
"Installed on machine" ...
 $ OperatingSystem: chr  "Windows" "Windows" "Windows" "Windows" ...
 $ TVhours        : chr  "1" "8h" "4" "5" ...
 $ Editor         : chr  "Microsoft Word" "Microsoft Word" "Microsoft Word"
"Microsoft Word" ...
```

- Now everything is a `character` instead of a `factor`

One-line Cleanup

- Let's clean up the `TVhours` column and cast it to numeric all in one command

```
survey <- transform(survey.messy, TVhours = as.numeric(gsub("[^0-9.]", "",
TVhours)))
str(survey)
```

```
'data.frame':	31 obs. of  6 variables:
 $ Program        : chr  "MISM" "Other" "MISM" "PPM" ...
 $ PriorExp       : chr  "Some experience" "Some experience" "Extensive
experience" "Never programmed before" ...
 $ Rexperience    : chr  "Never used" "Basic competence" "Basic competence"
"Installed on machine" ...
 $ OperatingSystem: chr  "Windows" "Windows" "Windows" "Windows" ...
 $ TVhours        : num  1 8 4 5 3 10 0 10 15 4 ...
 $ Editor         : chr  "Microsoft Word" "Microsoft Word" "Microsoft Word"
"Microsoft Word" ...
```

What about other character column?

```r
table(survey[["Program"]])
```

```

 MISM Other   PPM 
   10     6    15 
```

```r
table(as.factor(survey[["Program"]]))
```

```

 MISM Other   PPM 
   10     6    15 
```

- Having factors coded as characters may be OK for many parts of our analysis

Let’s fix it

```r
# Figure out which columns are coded as characters
chr.indexes <- sapply(survey, FUN = is.character)
chr.indexes
```

```
        Program        PriorExp     Rexperience OperatingSystem 
           TRUE            TRUE            TRUE            TRUE 
        TVhours          Editor 
          FALSE            TRUE 
```

```r
# Re-code all of the character columns to factors
survey[chr.indexes] <- lapply(survey[chr.indexes], FUN = as.factor)
```

Here is the outcome

```
str(survey)
```

```
'data.frame':	31 obs. of  6 variables:
 $ Program        : Factor w/ 3 levels "MISM","Other",..: 1 2 1 3 2 3 2 1 3 3 ...
 $ PriorExp       : Factor w/ 3 levels "Extensive experience",..: 3 3 1 2 2 3 3 3
2 1 ...
 $ Rexperience    : Factor w/ 3 levels "Basic competence",..: 3 1 1 2 3 3 3 3 3 3
...
 $ OperatingSystem: Factor w/ 2 levels "Mac OS X","Windows": 2 2 2 2 2 1 1 2 1 2
...
 $ TVhours        : num  1 8 4 5 3 10 0 10 15 4 ...
 $ Editor         : Factor w/ 1 level "Microsoft Word": 1 1 1 1 1 1 1 1 1 1 ...
```

- **Success!**

Another example

- When data is entered manually, misspellings and case changes are very common
- E.g., a column showing life support mechanism may look like,

```
life.support <- as.factor(c("dialysis", "Ventilation", "Dialysis", "dialysis",
"none", "None", "nnone", "dyalysis", "dialysis", "ventilation", "none"))
summary(life.support)
```

```
   dialysis    Dialysis    dyalysis       nnone        none        None
          3           1           1           1           2           1
ventilation Ventilation
          1           1
```

```
summary(life.support)
```

```
   dialysis    Dialysis    dyalysis       nnone        none        None 
          3           1           1           1           2           1 
ventilation Ventilation 
          1           1 
```

- This factor has 8 levels even though it should have 3 (dialysis, ventilation, none)
- We can fix many of the typos by running spellcheck in Excel before importing data, or by changing the values on a case-by-case basis later

[l/s/t/]apply() functions?

- These are all efficient ways of applying a function to margins of an array or elements of a list
- Before we talk about the details of `apply()` and its relatives, we should first understand loops
- **loops** are ways of iterating over data
- The `apply()` functions can be thought of as good *alternatives* to loops

Loop

```
for(i in 1:4) {
  print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
```

```
phrase <- "Good Night, "
for(word in c("and", "Good", "Luck")) {
  phrase <- paste(phrase, word)
  print(phrase)
}
```

```
[1] "Good Night,  and"
[1] "Good Night,  and Good"
[1] "Good Night,  and Good Luck"
```


Loop Syntax

A **for loop** executes a chunk of code for every value of an **index variable** in an **index set**

- The basic syntax takes the form

```
for(index.variable in index.set) {
  code to be repeated at every value of index.variable
}
```

- The index set is often a vector of integers, but can be more general

Example

```r
index.set <- rnorm(4)  # 4 random standard normal variables
index.set
```

```
[1] -1.9912818  0.7119512  0.3999575  0.5872188
```

```r
for(i in index.set) {
  print(abs(i))
}
```

```
[1] 1.991282
[1] 0.7119512
[1] 0.3999575
[1] 0.5872188
```


Example

```r
index.set <- list(name="Michael", weight=185, is.male=TRUE) # a list
for(i in index.set) {
  print(c(i, typeof(i)))
}
```

```
[1] "Michael"   "character"
[1] "185"     "double"
[1] "TRUE"     "logical"
```

Example

```
fake.data <- matrix(rnorm(500), ncol=5) # create fake 100 x 5 data set
head(fake.data,2) # print first two rows
```

```
          [,1]       [,2]       [,3]      [,4]      [,5]
[1,] 0.9551196 -0.3985364 -0.2338078 -1.018674 -0.684313
[2,] 0.6483192 -0.5487889  0.2091660  2.016935 -1.030668
```

```
col.sums <- numeric(ncol(fake.data)) # variable to store running column sums
for(i in 1:nrow(fake.data)) {
  col.sums <- col.sums + fake.data[i,] # add ith observation to the sum
}
print(col.sums)
```

```
[1] -12.808804 -11.552563   3.211727   4.049877  -9.901475
```

```
colSums(fake.data) # A better approach (see also colMeans())
```

```
[1] -12.808804 -11.552563   3.211727   4.049877  -9.901475
```


While loop

- **while loops** repeat a chunk of code while the specified condition remains true

```
day <- 1
num.days <- 365
while(day <= num.days) {
  day <- day + 1
}
```

- We won't really be using while loops in this class

Various app() function

Command	Description
`apply(X, MARGIN, FUN)`	Obtain a vector/array/list by applying `FUN` along the specified `MARGIN` of an array or matrix `X`
`lapply(X, FUN)`	Obtain a list by applying `FUN` to the elements of a list `X`
`sapply(X, FUN)`	Simplified version of `lapply`. Returns a vector/array instead of list.
`tapply(X, INDEX, FUN)`	Obtain a table by applying `FUN` to each combination of the factors given in `INDEX`

- These functions are (good!) alternatives to loops
- They are typically *more efficient* than loops (often run considerably faster on large data sets)
- Take practice to get used to, but make analysis easier to debug and less prone to error when used effectively

Example: apply()

```
colMeans(fake.data)
```

```
[1] -0.12808804 -0.11552563  0.03211727  0.04049877 -0.09901475
```

```
apply(fake.data, MARGIN=2, FUN=mean) # MARGIN = 1 for rows, 2 for columns
```

```
[1] -0.12808804 -0.11552563  0.03211727  0.04049877 -0.09901475
```

```
# Function that calculates proportion of vector indexes that are > 0
propPositive <- function(x) mean(x > 0)
apply(fake.data, MARGIN=2, FUN=propPositive)
```

```
[1] 0.51 0.47 0.51 0.51 0.46
```

Example: lapply(), sapply()

```
lapply(survey, is.factor) # Returns a list
```

```
$Program
[1] TRUE

$PriorExp
[1] TRUE

$Rexperience
[1] TRUE

$OperatingSystem
[1] TRUE

$TVhours
[1] FALSE

$Editor
[1] TRUE
```

```
sapply(survey, FUN = is.factor) # Returns a vector with named elements
```

```
        Program        PriorExp     Rexperience OperatingSystem 
           TRUE            TRUE            TRUE            TRUE 
        TVhours          Editor 
          FALSE            TRUE 
```


Example: apply, lapply, sapply

```
apply(cars, 2, FUN=mean) # Data frames are arrays
```

```
speed  dist 
15.40 42.98 
```

```
lapply(cars, FUN=mean) # Data frames are also lists
```

```
$speed
[1] 15.4

$dist
[1] 42.98
```

```
sapply(cars, FUN=mean) # sapply() is just simplified lapply()
```

```
speed  dist 
15.40 42.98 
```

tapply()

- Think of `tapply()` as a generalized form of the `table()` function

```
library(MASS)
# Get a count table, data broken down by Origin and DriveTrain
table(Cars93$Origin, Cars93$DriveTrain)
```

```
         4WD Front Rear
USA        5    34    9
non-USA    5    33    7
```

```
# Calculate average MPG.City, broken down by Origin and Drivetrain
tapply(Cars93$MPG.city, INDEX = Cars93[c("Origin", "DriveTrain")], FUN=mean)
```

```
         DriveTrain
Origin     4WD    Front     Rear
  USA     17.6 22.14706 18.33333
  non-USA 23.4 24.93939 19.14286
```


Example: tapply()

- Let's get the average horsepower by car `Origin` and `Type`

```
tapply(Cars93[["Horsepower"]], INDEX = Cars93[c("Origin", "Type")], FUN=mean)
```

```
         Type
Origin     Compact    Large  Midsize    Small   Sporty    Van
  USA     117.4286 179.4545 153.5000 89.42857 166.5000 158.40
  non-USA 141.5556       NA 189.4167 91.78571 151.6667 138.25
```

- What's that `NA` doing there?

```
any(Cars93$Origin == "non-USA" & Cars93$Type == "Large")
```

```
[1] FALSE
```

- None of the non-USA manufacturers produced Large cars!

with()

- Thus far we've repeatedly typed out the data frame name when referencing its columns
- This is because the data variables don't exist in our working environment
- Using **with**`(data, expr)` lets us specify that the code in `expr` should be evaluated in an environment that contains the elements of `data` as variables

```
with(Cars93, table(Origin, Type))
```

```
         Type
Origin    Compact Large Midsize Small Sporty Van
  USA           7    11      10     7      8   5
  non-USA       9     0      12    14      6   4
```


Example: with()

```
any(Cars93$Origin == "non-USA" & Cars93$Type == "Large")
```

```
[1] FALSE
```

```
with(Cars93, any(Origin == "non-USA" & Type == "Large")) # Same effect!
```

```
[1] FALSE
```

```
with(Cars93, tapply(Horsepower, INDEX = list(Origin, Type), FUN=mean))
```

```
         Compact    Large  Midsize    Small   Sporty    Van
USA     117.4286 179.4545 153.5000 89.42857 166.5000 158.40
non-USA 141.5556       NA 189.4167 91.78571 151.6667 138.25
```

- Using with() makes code simpler, easier to read, and easier to debug

Workshop 6: Loop

Loop practice

(a) Write a function called calculateRowMeans that uses a for loop to calculate the row means of a matrix x.

(b) Try out your function on the random matrix fake.data defined below.

(b) Use the apply() function to calculate the row means of the matrix fake.data

(c) Compare this to the output of the rowMeans() function to check that your calculation is correct.

Data Summary and Standard Graphic

Let's get started

- We're going to start by operating on the `birthwt` dataset from the MASS library
- Let's get it loaded and see what we're working with

```
library(MASS)
str(birthwt)
```

```
'data.frame':   189 obs. of  10 variables:
 $ low  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ age  : int  19 33 20 21 18 21 22 17 29 26 ...
 $ lwt  : int  182 155 105 108 107 124 118 103 123 113 ...
 $ race : int  2 3 1 1 1 3 1 3 1 1 ...
 $ smoke: int  0 0 1 1 1 0 0 0 1 1 ...
 $ ptl  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ ht   : int  0 0 0 0 0 0 0 0 0 0 ...
 $ ui   : int  1 0 0 1 1 0 0 0 0 0 ...
 $ ftv  : int  0 3 1 2 0 0 1 1 1 0 ...
 $ bwt  : int  2523 2551 2557 2594 2600 2622 2637 2637 2663 2665 ...
```


Renaming columns

- The dataset doesn't come with very descriptive variable names
- Let's get better column names (use `help(birthwt)` to understand the variables and come up with better names)

```
colnames(birthwt)
```

```
 [1] "low"    "age"    "lwt"    "race"   "smoke"  "ptl"    "ht"     "ui"    
 [9] "ftv"    "bwt"   
```

```
# The default names are not very descriptive

colnames(birthwt) <- c("birthwt.below.2500", "mother.age", "mother.weight",
    "race", "mother.smokes", "previous.prem.labor", "hypertension",
"uterine.irr",
    "physician.visits", "birthwt.grams")

# Better names!
```

Renaming the variables

- The dataset doesn't come with very descriptive variable names
- Let's get better column names (use `help(birthwt)` to understand the variables and come up with better names)

```
colnames(birthwt)
```

```
 [1] "low"    "age"    "lwt"    "race"   "smoke"  "ptl"    "ht"     "ui"
 [9] "ftv"    "bwt"
```

```
# The default names are not very descriptive

colnames(birthwt) <- c("birthwt.below.2500", "mother.age", "mother.weight",
    "race", "mother.smokes", "previous.prem.labor", "hypertension",
"uterine.irr",
    "physician.visits", "birthwt.grams")

# Better names!
```


Renaming the factors

- All the factors are currently represented as integers
- Let's use the `transform()` and `mapvalues()` functions to convert variables to factors and give the factors more meaningful levels

```
library(plyr)
birthwt <- transform(birthwt,
            race = as.factor(mapvalues(race, c(1, 2, 3),
                              c("white","black", "other"))),
            mother.smokes = as.factor(mapvalues(mother.smokes,
                              c(0,1), c("no", "yes"))),
            hypertension = as.factor(mapvalues(hypertension,
                              c(0,1), c("no", "yes"))),
            uterine.irr = as.factor(mapvalues(uterine.irr,
                              c(0,1), c("no", "yes"))),
            birthwt.below.2500 = as.factor(mapvalues(birthwt.below.2500,
                              c(0,1), c("no", "yes")))
            )
```

Summary of the data

- Now that things are coded correctly, we can look at an overall summary

```
summary(birthwt)
```

```
birthwt.below.2500   mother.age    mother.weight      race
no :130            Min.   :14.00   Min.   : 80.0   black:26
yes: 59            1st Qu.:19.00   1st Qu.:110.0   other:67
                   Median :23.00   Median :121.0   white:96
                   Mean   :23.24   Mean   :129.8
                   3rd Qu.:26.00   3rd Qu.:140.0
                   Max.   :45.00   Max.   :250.0
mother.smokes previous.prem.labor hypertension uterine.irr
no :115       Min.   :0.0000      no :177      no :161
yes: 74       1st Qu.:0.0000      yes: 12      yes: 28
              Median :0.0000
              Mean   :0.1958
              3rd Qu.:0.0000
              Max.   :3.0000
physician.visits birthwt.grams
Min.   :0.0000   Min.   : 709
1st Qu.:0.0000   1st Qu.:2414
Median :0.0000   Median :2977
Mean   :0.7937   Mean   :2945
3rd Qu.:1.0000   3rd Qu.:3487
Max.   :6.0000   Max.   :4990
```


A simple table

- Let's use the `tapply()` function to see what the average birthweight looks like when broken down by race and smoking status

```
with(birthwt, tapply(birthwt.grams, INDEX = list(race, mother.smokes), FUN =
mean))
```

```
             no      yes
black 2854.500 2504.000
other 2815.782 2757.167
white 3428.750 2826.846
```

Output table

- Let's use the header `{r, results='asis'}`, along with the `kable()` function from the `knitr` library

```
library(knitr)
kable(with(birthwt, tapply(birthwt.grams, INDEX = list(race, mother.smokes), FUN
= mean)), format = "markdown")
```

	no	yes
black	2854.500	2504.000
other	2815.782	2757.167
white	3428.750	2826.846

- `kable()` outputs the table in a way that Markdown can read and nicely display
- Note: changing the CSS changes the table appearance

aggregate() function

- Let's first recall what `tapply()` does
- Command: `tapply(X, INDEX, FUN)`
  - Applies `FUN` to `X` grouped by factors in `INDEX`
- **`aggregate()`** performs a similar operation, but presents the results in a form that is at times more convenient
- There are many ways to call the `aggregate()` function
- Analog of `tapply` call: `aggregate(X, by, FUN)`
  - Here, `by` is exactly like `INDEX`

Example: tapply vs aggregate

```
library(MASS)
with(birthwt, tapply(birthwt.grams, INDEX = list(race, mother.smokes), FUN =
mean)) # tapply
```

```
            no      yes
black 2854.500 2504.000
other 2815.782 2757.167
white 3428.750 2826.846
```

```
with(birthwt, aggregate(birthwt.grams, by = list(race, mother.smokes), FUN =
mean)) # aggregate
```

```
  Group.1 Group.2        x
1   black      no 2854.500
2   other      no 2815.782
3   white      no 3428.750
4   black     yes 2504.000
5   other     yes 2757.167
6   white     yes 2826.846
```


Different syntax

- Here's a convenient alternative way to call `aggregate`
- It uses the R `formula` syntax, which we'll learn more about when we discuss regression

```
aggregate(birthwt.grams ~ race + mother.smokes, FUN=mean, data=birthwt)
```

```
   race mother.smokes birthwt.grams
1 black            no      2854.500
2 other            no      2815.782
3 white            no      3428.750
4 black           yes      2504.000
5 other           yes      2757.167
6 white           yes      2826.846
```

- We'll see later that `aggregate` output can be more convenient for plotting

```
weight.smoke.tbl <- with(birthwt, table(birthwt.below.2500, mother.smokes))
weight.smoke.tbl
```

```
                   mother.smokes
birthwt.below.2500 no yes
               no  86  44
               yes 29  30
```

- The odds of low bwt among non-smoking mothers is

```
or.smoke.bwt <- (weight.smoke.tbl[2,2] / weight.smoke.tbl[1,2]) /
(weight.smoke.tbl[2,1] / weight.smoke.tbl[1,1])
or.smoke.bwt
```

```
[1] 2.021944
```

- So the odds of low birth weight are 2 times higher when the mother smokes

- Is the mother's age correlated with birth weight?

```
with(birthwt, cor(birthwt.grams, mother.age))  # Calculate correlation
```

```
[1] 0.09031781
```

- Does this change when we account for smoking status?

```
with(birthwt, cor(birthwt.grams[mother.smokes == "yes"],
mother.age[mother.smokes == "yes"]))
```

```
[1] -0.1441649
```

```
with(birthwt, cor(birthwt.grams[mother.smokes == "no"], mother.age[mother.smokes
== "no"]))
```

```
[1] 0.2014558
```

by() function

- Think of the `by(data, INDICES, FUN)` function as a `tapply()` function that operates on data frames instead of just vectors
- When using `tapply(X, INDEX, FUN)`, `X` is generally a numeric vector
- To calculate correlations, we need to allow `X` to be a data frame or matrix

```
by(data = birthwt[c("birthwt.grams", "mother.age")],
   INDICES = birthwt["mother.smokes"],
   FUN = function(x) {cor(x[,1], x[,2])})
```

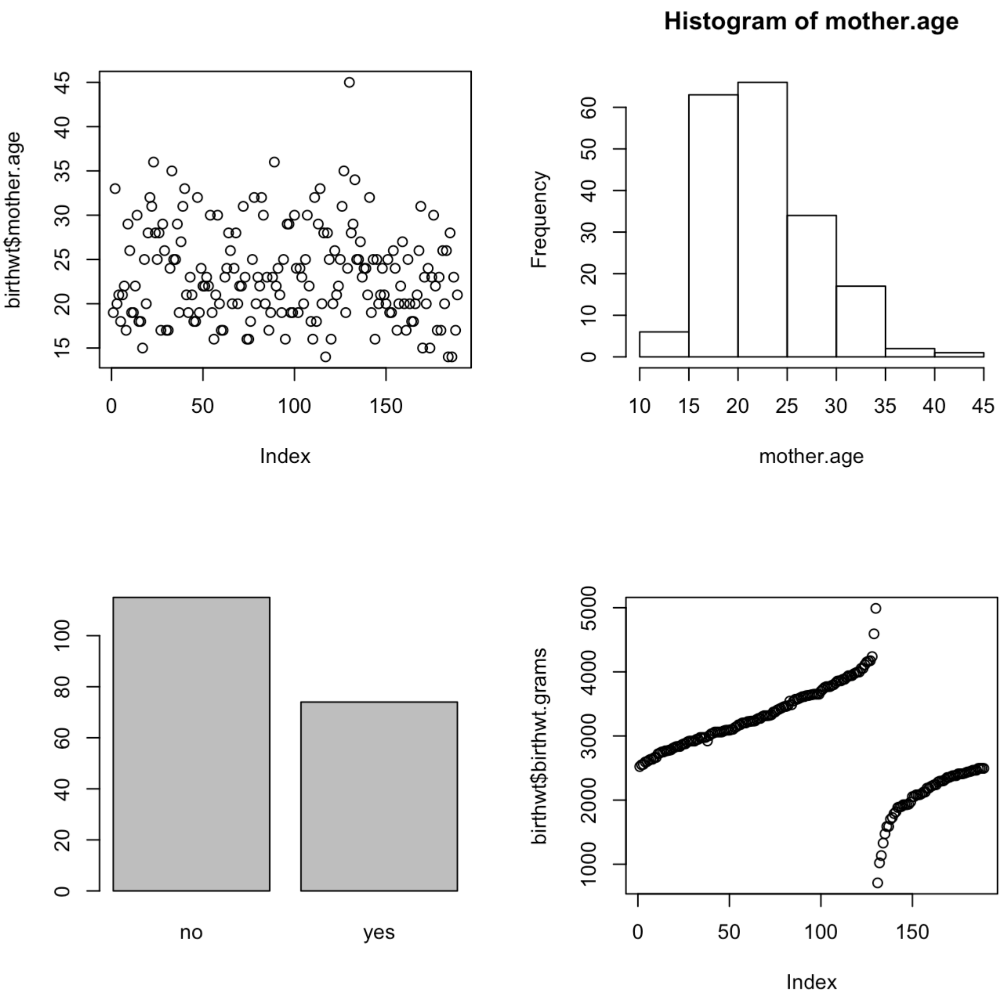
```
mother.smokes: no
[1] 0.2014558
--------------------------------------------------------
mother.smokes: yes
[1] -0.1441649
```


Basic single plot

Let's continue with the `birthwt` data from the `MASS` library.

Here are some basic single-variable plots.

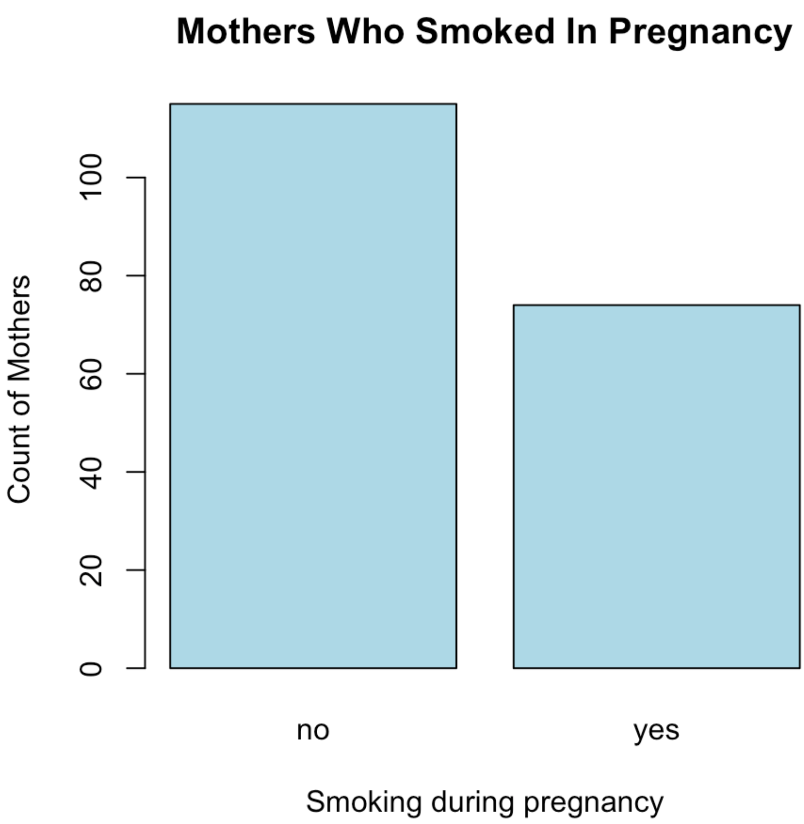
```
par(mfrow = c(2,2)) # Display plots in a single 2 x 2 figure
plot(birthwt$mother.age)
with(birthwt, hist(mother.age))
plot(birthwt$mother.smokes)
plot(birthwt$birthwt.grams)
```



Another example

Let's add more information to the smoking bar plot, and also change the color by setting the `col` option.

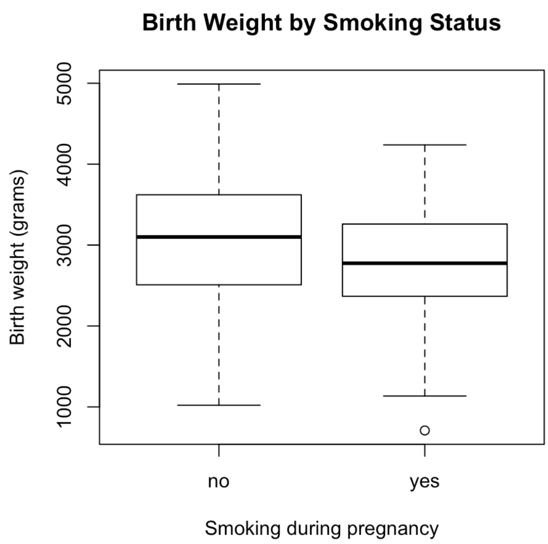
```
plot(birthwt$mother.smokes,
     main = "Mothers Who Smoked In Pregnancy",
     xlab = "Smoking during pregnancy",
     ylab = "Count of Mothers",
     col = 'lightblue')
```



Plots with several variables

If we call `plot(x,y,...)` with `x` a factor and `y` numeric, R will produce boxplots of `y` at every level of `x`.

```
with(birthwt, plot(mother.smokes, birthwt.grams,
                   main = "Birth Weight by Smoking Status",
                   xlab = "Smoking during pregnancy",
                   ylab = "Birth weight (grams)"))
```



More on ggplot2 demo

Workshop 7: Plot data

Plotting the Cars93 data

(a) Use qplot to create a scatterplot with Price on the y-axis and EngineSize on the x-axis.

(b) Repeat part (a) using the ggplot function and geom_point() layer.

(c) Repeat part (c), but this time specifying that the color mapping should depend on Type and the shape mapping should depend on DriveTrain.

Other Important Packages

To load data

RODBC, RMySQL, RPostgresSQL, RSQLite - If you'd like to read in data from a database, these packages are a good place to start. Choose the package that fits your type of database.

XLConnect, xlsx - These packages help you read and write Micorsoft Excel files from R. You can also just export your spreadsheets from Excel as .csv's.

foreign - Want to read a SAS data set into R? Or an SPSS data set? Foreign provides functions that help you load data files from other programs into R.

R can handle plain text files – no package required. Just use the functions read.csv, read.table, and read.fwf..

xlsx

```
df <- read.xlsx("<name and extension of your file>",
                sheetIndex = 1)
```

Note that it is necessary to add a sheet name or a sheet index to this function. In the example above, the first sheet of the Excel file was assigned. If you have a bigger data set, you might get better performance when using the `read.xlsx2()` function:

```
df <- read.xlsx2("<name and extension of your file>",
                 sheetIndex = 1,
                 startRow=2,
                 colIndex = 2)
```



```
write.xlsx(df,
           "df.xlsx",
           sheetName="Data Frame")
```

The function requires you first to specify what data frame you want to export. In the second argument, you specify the name of the file that you are outputting. If, however, you want to write the data frame to a file that already exists, you can execute the following command:

```
write.xlsx(df,
           "<name and extension of your existing file>",
           sheetName="Data Frame"
           append=TRUE)
```

To manipulate data

dplyr - Essential shortcuts for subsetting, summarizing, rearranging, and joining together data sets. dplyr is our go to package for fast data manipulation.

tidyr - Tools for changing the layout of your data sets. Use the gather and spread functions to convert your data into the tidy format, the layout R likes best.

stringr - Easy to learn tools for regular expressions and character strings.

lubridate - Tools that make working with dates and times easier.

dplyr

install.packages("dplyr")

library(dplyr)

dplyr verbs	Description
`select()`	select columns
`filter()`	filter rows
`arrange()`	re-order or arrange rows
`mutate()`	create new columns
`summarise()`	summarise values
`group_by()`	allows for group operations in the "split-apply-combine" concept

select

Select a set of columns: the name and the sleep_total columns.

```
sleepData <- select(msleep, name, sleep_total)
head(sleepData)
```

```
##                          name sleep_total
## 1                     Cheetah        12.1
## 2                  Owl monkey        17.0
## 3             Mountain beaver        14.4
## 4  Greater short-tailed shrew        14.9
## 5                         Cow         4.0
## 6            Three-toed sloth        14.4
```


Subtraction (-)

To select all the columns *except* a specific column, use the "-" (subtraction) operator (also known as negative indexing)

```
head(select(msleep, -name))
```

```
##        genus  vore        order conservation sleep_total sleep_rem
## 1   Acinonyx carni    Carnivora           lc        12.1        NA
## 2      Aotus  omni     Primates         <NA>        17.0       1.8
## 3 Aplodontia herbi     Rodentia           nt        14.4       2.4
## 4    Blarina  omni Soricomorpha           lc        14.9       2.3
## 5        Bos herbi Artiodactyla domesticated         4.0       0.7
## 6   Bradypus herbi       Pilosa         <NA>        14.4       2.2
##   sleep_cycle awake brainwt  bodywt
## 1          NA  11.9      NA  50.000
## 2          NA   7.0 0.01550   0.480
## 3          NA   9.6      NA   1.350
## 4   0.1333333   9.1 0.00029   0.019
## 5   0.6666667  20.0 0.42300 600.000
## 6   0.7666667   9.6      NA   3.850
```

colon (:)

To select a range of columns by name, use the ":" (colon) operator

```
head(select(msleep, name:order))
```

```
##                          name      genus  vore        order
## 1                     Cheetah   Acinonyx carni    Carnivora
## 2                  Owl monkey      Aotus  omni     Primates
## 3             Mountain beaver Aplodontia herbi     Rodentia
## 4  Greater short-tailed shrew    Blarina  omni Soricomorpha
## 5                         Cow        Bos herbi Artiodactyla
## 6            Three-toed sloth   Bradypus herbi       Pilosa
```


To select all columns that start with the character string "sl", use the function `starts_with()`

```
head(select(msleep, starts_with("sl")))
```

```
##   sleep_total sleep_rem sleep_cycle
## 1        12.1        NA          NA
## 2        17.0       1.8          NA
## 3        14.4       2.4          NA
## 4        14.9       2.3   0.1333333
## 5         4.0       0.7   0.6666667
## 6        14.4       2.2   0.7666667
```

Some additional options to select columns based on a specific criteria include

1. `ends_with()` = Select columns that end with a character string
2. `contains()` = Select columns that contain a character string
3. `matches()` = Select columns that match a regular expression
4. `one_of()` = Select columns names that are from a group of names

filter

Filter the rows for mammals that sleep a total of more than 16 hours.

```
filter(msleep, sleep_total >= 16)
```

```
##                         name        genus    vore           order conservation
## 1                  Owl monkey        Aotus    omni        Primates         <NA>
## 2    Long-nosed armadillo      Dasypus   carni       Cingulata           lc
## 3 North American Opossum    Didelphis    omni Didelphimorphia           lc
## 4             Big brown bat    Eptesicus insecti      Chiroptera           lc
## 5    Thick-tailed opposum   Lutreolina   carni Didelphimorphia           lc
## 6          Little brown bat       Myotis insecti      Chiroptera         <NA>
## 7         Giant armadillo   Priodontes insecti       Cingulata           en
## 8 Arctic ground squirrel Spermophilus   herbi        Rodentia           lc
##   sleep_total sleep_rem sleep_cycle awake brainwt bodywt
## 1        17.0       1.8          NA   7.0 0.01550  0.480
## 2        17.4       3.1   0.3833333   6.6 0.01080  3.500
## 3        18.0       4.9   0.3333333   6.0 0.00630  1.700
## 4        19.7       3.9   0.1166667   4.3 0.00030  0.023
## 5        19.4       6.6          NA   4.6      NA  0.370
## 6        19.9       2.0   0.2000000   4.1 0.00025  0.010
## 7        18.1       6.1          NA   5.9 0.08100 60.000
## 8        16.6        NA          NA   7.4 0.00570  0.920
```

filter with and

Filter the rows for mammals that sleep a total of more than 16 hours *and* have a body weight of greater than 1 kilogram.

```
filter(msleep, sleep_total >= 16, bodywt >= 1)
```

```
##                      name      genus    vore           order conservation
## 1   Long-nosed armadillo    Dasypus   carni       Cingulata           lc
## 2 North American Opossum  Didelphis    omni Didelphimorphia           lc
## 3        Giant armadillo Priodontes insecti       Cingulata           en
##   sleep_total sleep_rem sleep_cycle awake brainwt bodywt
## 1        17.4       3.1   0.3833333   6.6  0.0108    3.5
## 2        18.0       4.9   0.3333333   6.0  0.0063    1.7
## 3        18.1       6.1          NA   5.9  0.0810   60.0
```

Pipe (%>%)

```
head(select(msleep, name, sleep_total))
```

```
##                          name sleep_total
## 1                     Cheetah        12.1
## 2                  Owl monkey        17.0
## 3             Mountain beaver        14.4
## 4  Greater short-tailed shrew        14.9
## 5                         Cow         4.0
## 6            Three-toed sloth        14.4
```



```
msleep %>%
    select(name, sleep_total) %>%
    head
```

```
##                          name sleep_total
## 1                     Cheetah        12.1
## 2                  Owl monkey        17.0
## 3             Mountain beaver        14.4
## 4  Greater short-tailed shrew        14.9
## 5                         Cow         4.0
## 6            Three-toed sloth        14.4
```

sqldf

library(sqldf)

```
setwd()
crashes <- read.csv("crashes.csv")
roads <- read.csv("roads.csv")
head(crashes)
```

```
##   Year           Road N_Crashes Volume
## 1 1991 Interstate 65        25  40000
## 2 1992 Interstate 65        37  41000
## 3 1993 Interstate 65        45  45000
## 4 1994 Interstate 65        46  45600
## 5 1995 Interstate 65        46  49000
## 6 1996 Interstate 65        59  51000
```

```
tail(crashes)
```

```
##     Year            Road N_Crashes Volume
## 105 2007 Interstate 275        32  21900
## 106 2008 Interstate 275        21  21850
## 107 2009 Interstate 275        25  22100
## 108 2010 Interstate 275        24  21500
## 109 2011 Interstate 275        23  20300
## 110 2012 Interstate 275        22  21200
```



```
print(roads)
```

```
##             Road       District Length
## 1 Interstate 65      Greenfield    262
## 2 Interstate 70       Vincennes    156
## 3         US-36 Crawfordsville    139
## 4         US-40      Greenfield    150
## 5         US-52 Crawfordsville    172
```

Outer join

```
join_string <- "select
                crashes.*
              , roads.District
              , roads.Length
              from crashes
                left join roads
                on crashes.Road = roads.Road"
```

```
crashes_join_roads <- sqldf(join_string,stringsAsFactors = FALSE)
```

```
## Loading required package: tcltk
```

```
head(crashes_join_roads)
```

```
##   Year          Road N_Crashes Volume   District Length
## 1 1991 Interstate 65        25  40000 Greenfield    262
## 2 1992 Interstate 65        37  41000 Greenfield    262
## 3 1993 Interstate 65        45  45000 Greenfield    262
## 4 1994 Interstate 65        46  45600 Greenfield    262
## 5 1995 Interstate 65        46  49000 Greenfield    262
## 6 1996 Interstate 65        59  51000 Greenfield    262
```


Inner join

By using an inner join, only matching rows will be kept.

```
join_string2 <- "select
                crashes.*
              , roads.District
              , roads.Length
              from crashes
                inner join roads
                on crashes.Road = roads.Road"
```

```
crashes_join_roads2 <- sqldf(join_string2, stringsAsFactors = FALSE)
head(crashes_join_roads2)
```

```
##   Year          Road N_Crashes Volume   District Length
## 1 1991 Interstate 65        25  40000 Greenfield    262
## 2 1992 Interstate 65        37  41000 Greenfield    262
## 3 1993 Interstate 65        45  45000 Greenfield    262
## 4 1994 Interstate 65        46  45600 Greenfield    262
## 5 1995 Interstate 65        46  49000 Greenfield    262
## 6 1996 Interstate 65        59  51000 Greenfield    262
```

```r
tail(crashes_join_roads2)
```

```
##     Year   Road N_Crashes Volume        District Length
## 83 2007 US-36         49  24000 Crawfordsville    139
## 84 2008 US-36         52  24500 Crawfordsville    139
## 85 2009 US-36         55  24700 Crawfordsville    139
## 86 2010 US-36         35  23000 Crawfordsville    139
## 87 2011 US-36         33  21000 Crawfordsville    139
## 88 2012 US-36         31  20500 Crawfordsville    139
```


Where

```r
join_string2 <- "select
                crashes.*
              , roads.District
              , roads.Length
                from crashes
                    inner join roads
                    on crashes.Road = roads.Road
                where crashes.Road = 'US-40'"
crashes_join_roads4 <- sqldf(join_string2,stringsAsFactors = FALSE)
head(crashes_join_roads4)
```

```
##   Year   Road N_Crashes Volume   District Length
## 1 1991 US-40        46  21000 Greenfield    150
## 2 1992 US-40       101  21500 Greenfield    150
## 3 1993 US-40        76  23000 Greenfield    150
## 4 1994 US-40        72  21000 Greenfield    150
## 5 1995 US-40        75  24000 Greenfield    150
## 6 1996 US-40       136  23500 Greenfield    150
```

```r
tail(crashes_join_roads4)
```

```
##    Year   Road N_Crashes Volume   District Length
## 17 2007 US-40        45  59500 Greenfield    150
## 18 2008 US-40        23  61000 Greenfield    150
## 19 2009 US-40        67  65000 Greenfield    150
## 20 2010 US-40       102  67000 Greenfield    150
## 21 2011 US-40        87  67500 Greenfield    150
## 22 2012 US-40        32  67500 Greenfield    150
```

To visualize data

ggplot2 - R's famous package for making beautiful graphics. ggplot2 lets you use the grammar of graphics to build layered, customizable plots.

ggvis - Interactive, web based graphics built with the grammar of graphics.

rgl - Interactive 3D visualizations with R

htmlwidgets - A fast way to build interactive (javascript based) visualizations with R. Packages that implement htmlwidgets include:

- **leaflet** (maps)
- **dygraphs** (time series)
- **DT** (tables)
- **diagrammeR** (diagrams)
- **network3D** (network graphs)
- **threeJS** (3D scatterplots and globes).

googleVis - Let's you use Google Chart tools to visualize data in R. Google Chart tools used to be called Gapminder, the graphing software Hans Rosling made famous in hie TED talk.

To model data

car - car's Anova function is popular for making type II and type III Anova tables.

mgcv - Generalized Additive Models

lme4/nlme - Linear and Non-linear mixed effects models

randomForest - Random forest methods from machine learning

multcomp - Tools for multiple comparison testing

vcd - Visualization tools and tests for categorical data

glmnet - Lasso and elastic-net regression methods with cross validation

survival - Tools for survival analysis

To report results

shiny - Easily make interactive, web apps with R. A perfect way to explore data and share findings with non-programmers.

R Markdown - The perfect workflow for reproducible reporting. Write R code in your markdown reports. When you run render, R Markdown will replace the code with its results and then export your report as an HTML, pdf, or MS Word document, or a HTML or pdf slideshow. The result? Automated reporting. R Markdown is integrated straight into RStudio.

xtable - The xtable function takes an R object (like a data frame) and returns the latex or HTML code you need to paste a pretty version of the

Introduction to Shiny

- Open Sourced by RStudio November 2012
- Default widgets and settings make it easy to generate apps
- Don't need to know HTML, CSS and javascript to get started
- Twitter Bootstrap for default UI - looks good
- Web sockets for communication between client and server
- Reactive Programming model
- Works on Windows, Mac, Linux

Introduction to Shiny

Ready to use shiny

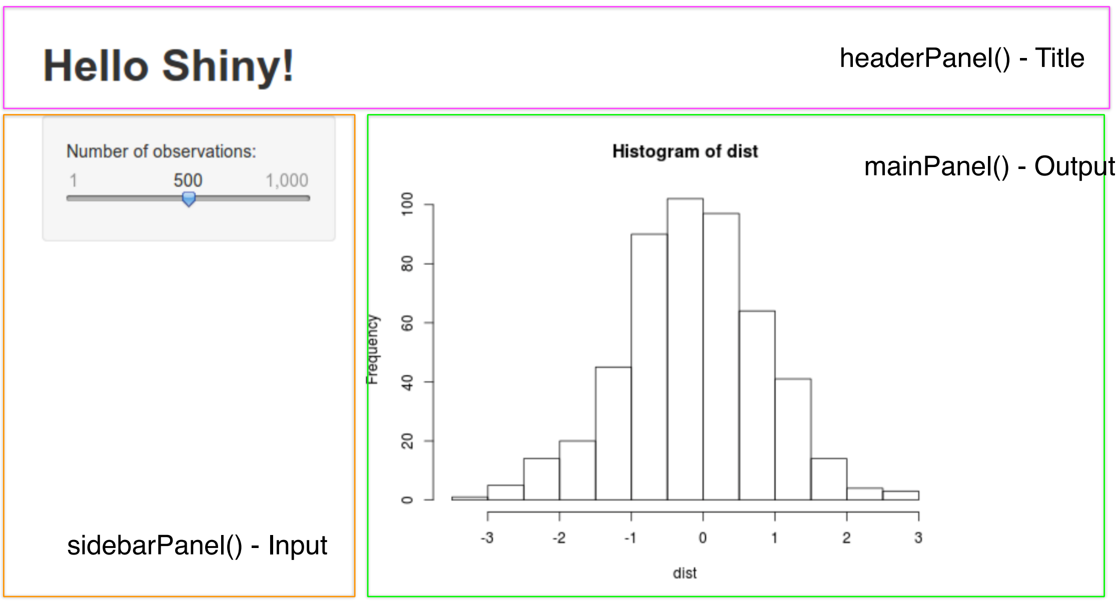
- Install R from CRAN

- Useful to have Chrome, Firefox, Safari...

- Install Shiny using R command: `install.packages("shiny")`

Simple Example

- `library(shiny)`
- `runExample("01_hello")`



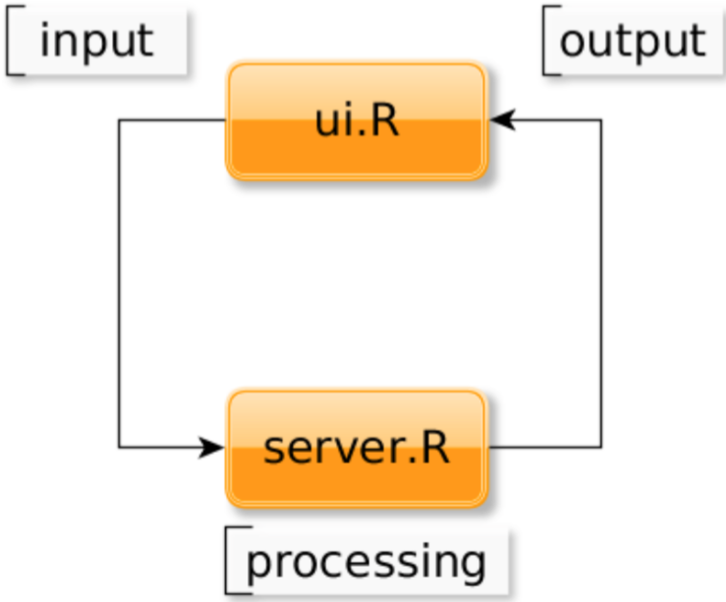
ui.R : : Controls the look of the App

```r
library(shiny)

# Define UI for application that draws a histogram
shinyUI(fluidPage(

  # Application title
  titlePanel("Hello Shiny!"),

  # Sidebar with a slider input for the number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),

    # Show a plot of the generated distribution
    mainPanel(
      plotOutput("distPlot")
    )
  )
))
```

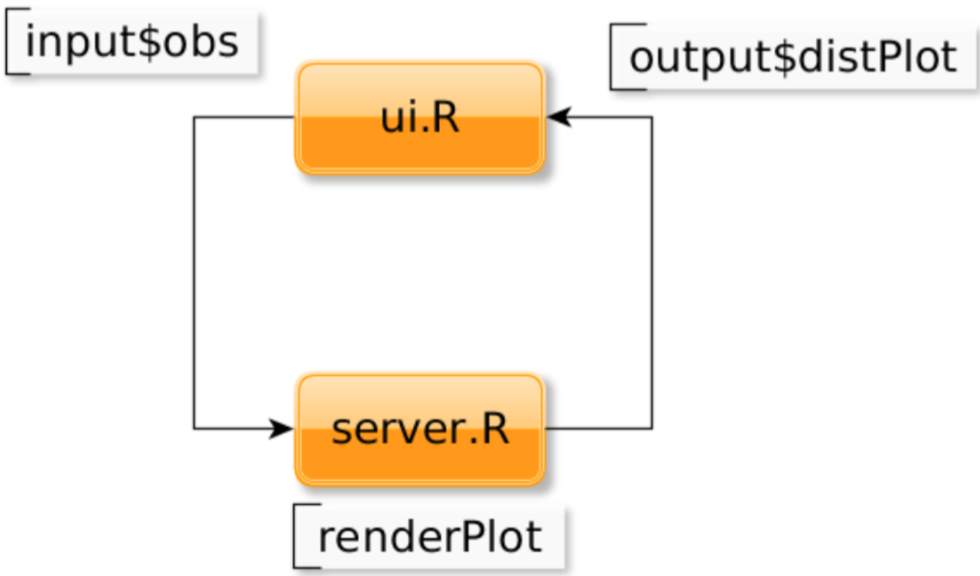

server.R : Specifies what R is doing

```r
library(shiny)

# Define server logic required to draw a histogram
shinyServer(function(input, output) {

  # Expression that generates a histogram. The expression is
  # wrapped in a call to renderPlot to indicate that:
  #
  #  1) It is "reactive" and therefore should be automatically
  #     re-executed when inputs change
  #  2) Its output type is a plot

  output$distPlot <- renderPlot({
    x    <- faithful[, 2]  # Old Faithful Geyser data
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })

})
```

Relationship of ui.R and server.R



Relationship of ui.R and server.R



For Spatial data

sp, maptools - Tools for loading and using spatial data including shapefiles.

maps - Easy to use map polygons for plots.

ggmap - Download street maps straight from Google maps and use them as a background in your ggplots.

To write your own R packages

devtools - An essential suite of tools for turning your code into an R package.

testthat - testthat provides an easy way to write unit tests for your code projects.

roxygen2 - A quick way to document your R packages. roxygen2 turns inline code comments into documentation pages and builds a package namespace.

Hypothesis testing in R

Fundamental of Hypothesis Testing

There two types of **statistical inferences**, **Estimation** and **Hypothesis Testing**

Hypothesis Testing: A hypothesis is a **claim** (assumption) about one or more **population** parameters.

- Average price of a six-pack in the U.S. is $\mu = \$4.90$
- The population mean monthly cell phone bill of this city is: $\mu = \$42$
- The average number of TV sets in U.S. Homes is equal to three; $\mu = 3$

It Is always about a population parameter, not about a sample statistic

Sample evidence is used to assess the probability that the claim about the population parameter is true

A. It starts with Null Hypothesis, H_0

$$H_0: \quad =$$

1. We begin with the assumption that H_0 is true and any difference between the sample statistic and true population parameter is due to chance and not a real (systematic) difference.
2. Similar to the notion of "innocent until proven guilty"
3. That is, "innocence" is a null hypothesis.

Null Hypo, Continued

4. Refers to the status quo
5. Always contains "=" , "≤" or "≥" sign
6. May or may not be rejected

B. Next we state the Alternative Hypothesis, H_1

1. Is the opposite of the null hypothesis
    1. e.g., The average number of TV sets in U.S. homes is not equal to 3 ($H_1: \mu \neq 3$)
2. Challenges the status quo
3. Never contains the "=" , "≤" or "≥" sign
4. May or may not be proven
5. Is generally the hypothesis that the researcher is trying to prove. Evidence is always examined with respect to **H_1**, never with respect to **H_0**.
6. We never "accept" **H_0**, we either "reject" or "not reject" it

Summary:

- In the process of hypothesis testing, the null hypothesis initially is assumed to be true
- Data are gathered and examined to determine whether the evidence is strong enough with respect to the alternative hypothesis to reject the assumption.
- In another words, the burden is placed on the researcher to show, using sample information, that the null hypothesis is false.
- If the sample information is sufficient enough in favor of the alternative hypothesis, then the null hypothesis is rejected. This is the same as saying if the persecutor has enough evidence of guilt, the "innocence is rejected.
- Of course, erroneous conclusions are possible, type I and type II errors.

Reason for Rejecting H_0

Illustration: Let say, we **assume** that average age in the US is 50 years (H0=50). If in fact this is the true (unknown) population mean, it is unlikely that we get a sample mean of 20. So, if we have a sample that produces an average of 20, them we **reject** that the null hypothesis that average age is 50. (note that we are rejecting our assumption or claim). (would we get 20 if the true population mean was 50? NO. That is why we reject 50)

How Is the Test done?

We use the distribution of a Test Statistic, such as Z or t as the criteria.

A. Rejection Region Method:

Divide the distribution into rejection and non-rejection regions

Defines the unlikely values of the sample statistic if the null hypothesis is true, the critical value(s)

Defines rejection region of the sampling distribution

Rejection region(s) is designated by α , (level of significance)

Typical values are .01, .05, or .10

α is selected by the researcher at the beginning

α provides the critical value(s) of the test

Rejection Region or Critical Value Approach:

Level of significance = α

H_0: $\mu = 12$
H_1: $\mu \neq 12$

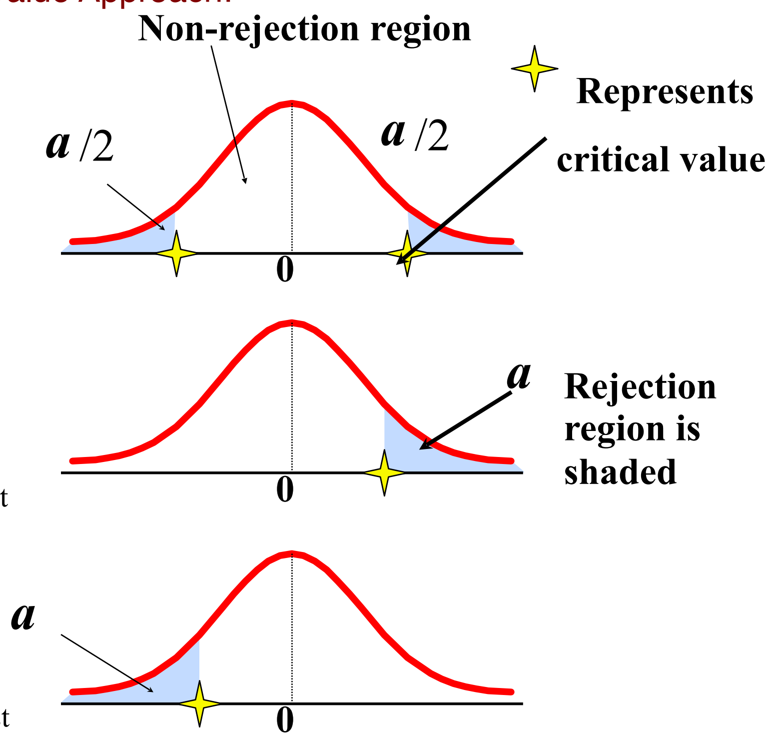
Two-tail test

H_0: $\mu \leq 12$ H_1: $\mu > 12$

Upper-tail test

H_0: $\mu \geq 12$
H_1: $\mu < 12$

Lower-tail test



P-Value Approach –

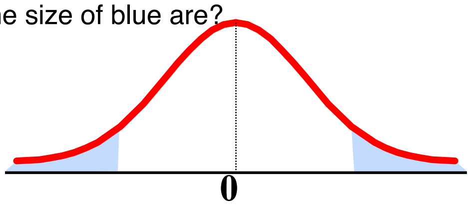
P-value=Max. Probability of (Type I Error), calculated from the sample.

Given the sample information what is the size of blue are?

H_0: $\mu = 12$

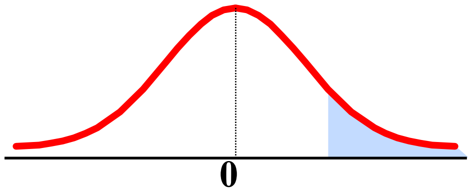
H_1: $\mu \neq 12$

Two-tail test



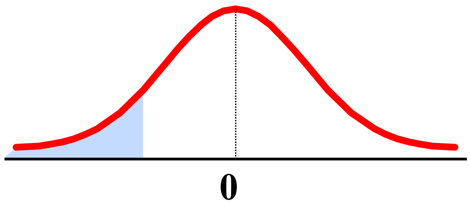
H_0: $\mu \leq 12$ H_1: $\mu > 12$

Upper-tail test



H_0: $\mu \geq 12$

H_1: $\mu < 12$



Type I and II Errors:

The size of α , the rejection region, affects the risk of making different types of incorrect decisions.

Type I Error

Rejecting a true null hypothesis when it should NOT be rejected

Considered a serious type of error

The probability of Type I Error is α

It is also called level of significance of the test

Type II Error

Fail to reject a false null hypothesis that should have been rejected

The probability of Type II Error is β

Decision	Actual Situation			
	Hypothesis Testing		Legal System	
	H0 True	H0 False	Innocence	Not innocence
Do Not Reject H_0	**No Error** $(1-\alpha)$	**Type II Error** (β)	No Error (not guilty, found not guilty) $(1-\alpha)$	Type II Error (guilty, found not guilty) (β)
Reject H_0	**Type I Error** (α)	**No Error** $(1-\beta)$	Type I Error (Not guilty, found guilty) (α)	No Error (guilty, found guilty) $(1-\beta)$

Type I and Type II errors cannot happen at the same time

1. Type I error can only occur if H_0 is true
2. Type II error can only occur if H_0 is false
3. There is a tradeoff between type I and II errors. If the probability of type I error (α) increased, then the probability of type II error (β) declines.
4. When the difference between the hypothesized parameter and the actual true value is small, the probability of type two error (the non-rejection region) is larger.
5. Increasing the sample size, n, for a given level of α**,** reduces β

B. P-Value approach to Hypothesis Testing:

1. The rejection region approach allows you to examine evidence but restrict you to not more than a certain probability (say α = 5%) of rejecting a true H_0 by mistake.

2. The P-value approach allows you to use the information from the sample and then calculate the maximum probability of rejecting a true H_0 by mistake.

3. Another way of looking at P-value is the probability of observing a sample information of “A=11.5” when the true population parameter is “12=B”. The P-value is the maximum probability of such mistake taking place.

4. That is to say that P-value is the smallest value of α for which H_0 can be rejected based on the sample information

5. Convert Sample Statistic (e.g., sample mean) to Test Statistic (e.g., Z statistic)

6. Obtain the p-value from a table or computer

7. Compare the p-value with α

If p-value < α , reject H_0

If p-value ≥ α , do not reject H_0

Test of Hypothesis for the Mean

σ known

The test statistic is:

$$Z = \frac{\bar{X} - \mu}{\frac{\sigma}{\sqrt{n}}}$$

σ Unknown

The test statistic is:

$$t_{n-1} = \frac{\bar{X} - \mu}{\frac{S}{\sqrt{n}}}$$

Steps to Hypothesis Testing

1. State the H_0 and H_1 clearly
2. Identify the test statistic (two-tail, one-tail, and Z or t distribution
3. Depending on the type of risk you are willing to take, specify the level of significance,
4. Find the decision rule, critical values, and rejection regions. If –CV<actual value (sample statistic) <+CV, then do not reject the H_0
5. Collect the data and do the calculation for the actual values of the test statistic from the sample

Make statistical decision

Do not Reject H_0 → Conclude H_0 may be true

Reject H_0 → Conclude H_1 is "true"

(There is sufficient evidence of H1)

Make management/business/
administrative decision

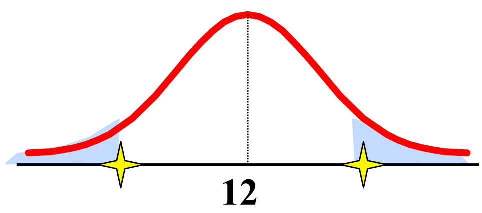
When do we use a two-tail test?

The answer depends on the question you are trying to answer.

A two-tail is used when the researcher has no idea which direction the study will go, interested in both direction. (example: testing a new technique, a new product, a new theory and we don't know the direction)

A new machine is producing 12 fluid once can of soft drink. The quality control manager is concern with cans containing too much or too little. Then, the test is a two-tailed test. That is the two rejection regions in tails is most likely (higher probability) to provide evidence of H_1.

$$H_0 : \mu = 12 \text{ oz}$$
$$H_1 : \mu \neq 12 \text{ oz}$$



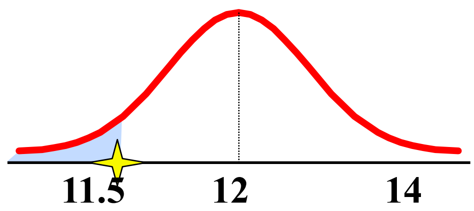
One-tail test is used when the researcher is interested in the direction.

Example: The soft-drink company puts a label on cans claiming they contain 12 oz. A consumer advocate desires to test this statement. She would assume that each can contains at least 12 oz and tries to find evidence to the contrary. That is, she examines the evidence for less than 12 0z.

What tail of the distribution is the most logical (higher probability) to find that evidence? The only way to reject the claim is to get evidence of less than 12 oz, left tail.

$H_0 : \mu \geq 12\ oz$

$H_1 : \mu < 12\ oz$



Review of Hypo. Testing

What is HT?

Probability of making erroneous conclusions

- Type I – only when Null Hypo is true
- Type II – only when Null Hypo is false

Two Approaches

- The Rejection or Critical Value Approach
- The P-value Approach (we calculate the observed level of significance)

Test Statistics

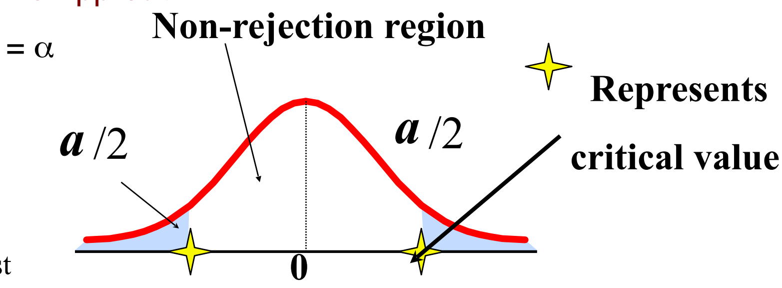
- Z- distribution if Population Std. Dev. is Know
- t-distribution if the Population Std. Dev. is unknown

Rejection Region or Critical Value Approach:

The given level of significance = α

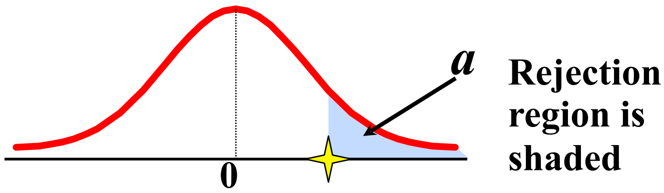
H_0: $\mu = 12$

H_1: $\mu \neq 12$



Two-tail test

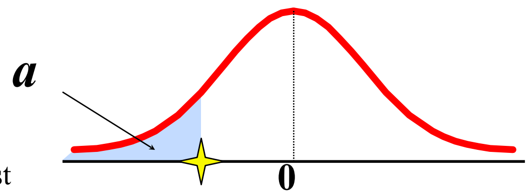
H_0: $\mu \leq 12$ H_1: $\mu > 12$



Upper-tail test

H_0: $\mu \geq 12$

H_1: $\mu < 12$



Lower-tail test

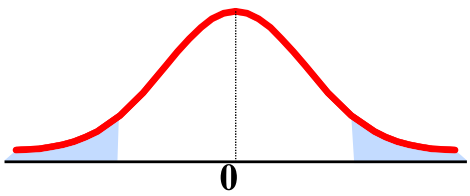
P-Value Approach –

P-value=Max. Probability of (Type I Error), calculated from the sample.

Given the sample information what is the size of the blue areas? (The observed level of significance)

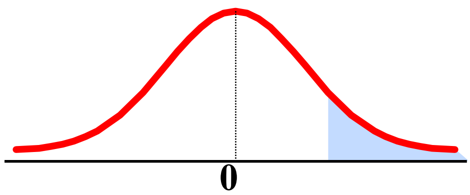
H_0: $\mu = 12$

H_1: $\mu \neq 12$



Two-tail test

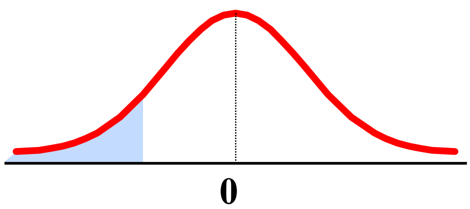
H_0: $\mu \leq 12$ H_1: $\mu > 12$



Upper-tail test

H_0: $\mu \geq 12$

H_1: $\mu < 12$



Example 1:

Let's assume a sample of 25 cans produced a sample mean of 11.5 0z and the population std dev=1 0z.

Question 1:

At a 5% level of significance (that is allowing for a maximum of 5% prob. of rejecting a true null hypo), is there evidence that the population mean is different from 12 oz?

Null Hypo is:?
Alternative Hypo is?

Can both approaches be used to answer this question?

A: Rejection region approach: calculate the actual test statistics and compare it with the critical values

B: P-value approach: calculate the actual probability of type I error given the sample information. Then compare it with 1%, 5%, or 10% level of significance.

Interpretation of Critical Value/Rejection Region Approach:

Interpretation of P-value Approach:

Question 2:

At a 5% level of significance (that is allowing for a maximum of 5% prob. of rejecting a true null hypo), is the evidence that the population mean is less than 12 oz?

Null Hypo is:?

Alternative Hypo is?

Can both approaches be used to answer this question?

Interpretation of Critical Value Approach:

Interpretation of P-value Approach:

Question 3:

If in fact the pop. mean is 12 oz, <u>what is the probability</u> of obtaining a sample mean of 11.5 or less oz (sample size 25)? Null

Null Hypo is:?

Alternative Hypo is?

Question 4:

If in fact the pop. mean is 12 oz, and the sample mean is 11.5 (or less), <u>what is the probability</u> of erroneously rejecting the null hypo that the pop. mean is 12 oz?

Null Hypo is:?

Alternative Hypo is?

Can both approaches be used to answer these question?

Connection to Confidence Intervals

While the confidence interval estimation and hypothesis testing serve different purposes, they are based on same concept and conclusions reached by two methods are consistent for a two-tail test.

In CI method we estimate an interval for the population mean with a degree of confidence. If the estimated interval contains the hypothesized value under the hypothesis testing, then this is equivalent of not rejecting the null hypothesis. For example: for the beer sample with mean 5.20, the confidence interval is:

$P(4.61 \le \mu \le 5.78)=95\%$

Since this interval contains the Hypothesized mean ($4.90), we do not (did not) reject the null hypothesis at $\alpha = .05$

Did not reject and within the interval, thus consistent results.

Introduction to Regression Analysis

Regression analysis is used to:

Predict values of a dependent variable, Y, based on its relationship with values of at least one independent variable, X.

Explain the impact of changes in an independent variable on the dependent variable by estimating the **numerical value** of the relationship

Dependent variable: the variable we wish to explain

Independent variable: the variable used to explain the dependent variable

Simple Linear Regression Model

Only **one** independent variable (thus, simple), X

Relationship between X and Y is described by a linear function

Changes in Y are assumed to be caused by changes in X, that is,

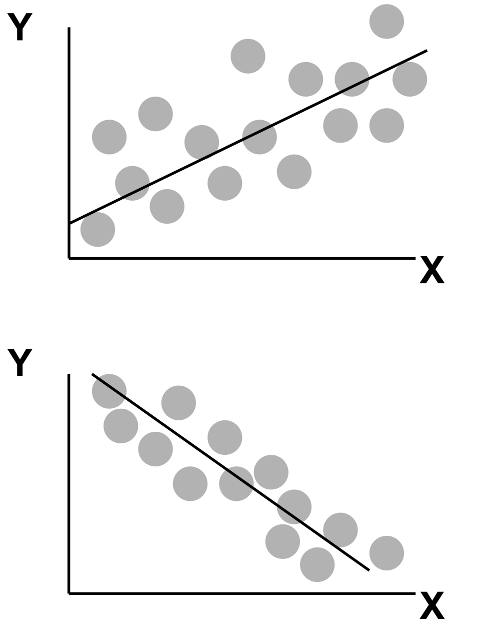
Change In **X** $\xrightarrow{\quad}$ **Causes** Change in Y

Important points before we start a regression analysis:

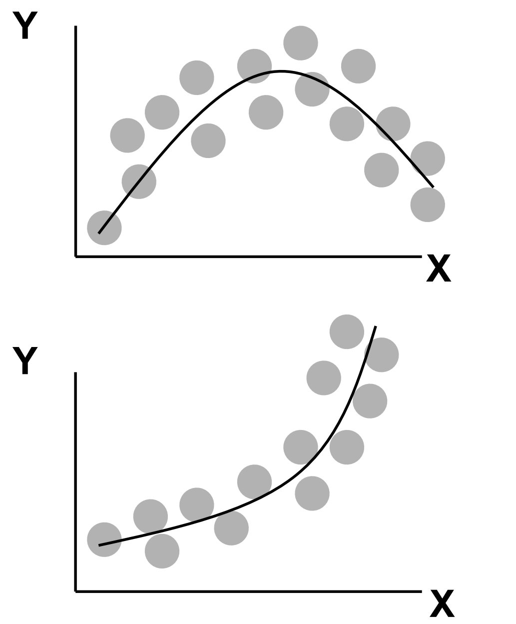
- The most important thing in deciding whether or not there is a relationship between X and Y is to have a systematic model that is based on logical reasons.
- Investigate the nature of the relationship between X and Y (use scatter diagram, covariance, correlation of coefficient)
- Remember that regression is not an exact or deterministic mathematical equation. It is a **behavioral relationship** that is subject to randomness.
- Remember that X is not the only thing that explains the behavior of Y. There are other factor that you may not have information about.
- All you are trying to do is to have an estimate of the relationship using the **best linear fit** possible

Types of Relationships

Linear relationships



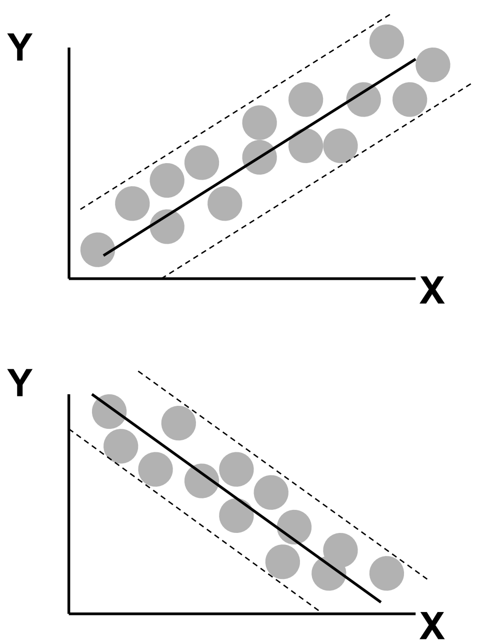
Curvilinear relationships



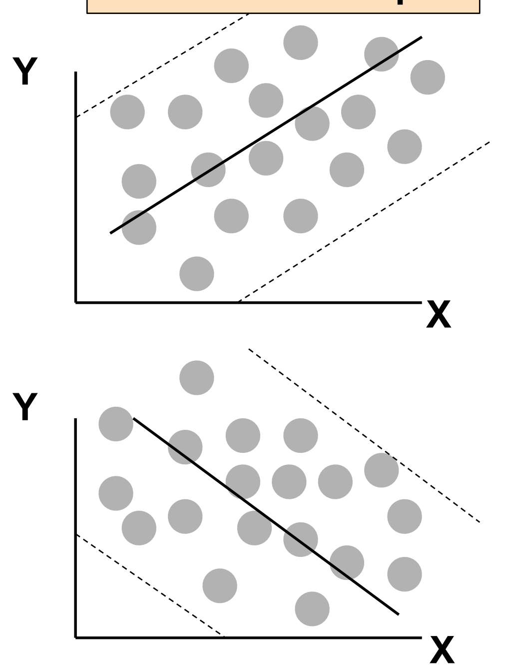
Types of Relationships

(continued)

Strong relationships



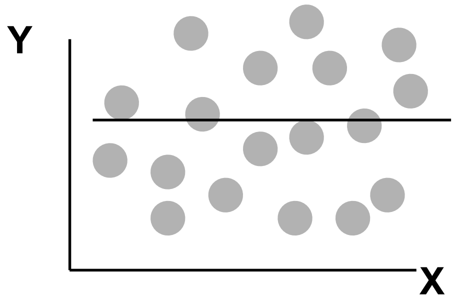
Weak relationships

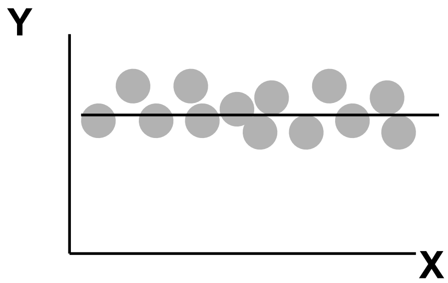


Types of Relationships

(continued)

No relationship





Simple Linear Regression Conceptual Model

The population regression model: This is a conceptual model, a hypothesis, or a postulation

Dependent Variable

Population Y intercept

Population Slope Coefficient

Independent Variable

Random Error term

$$Y_i = \underbrace{\beta_0 + \beta_1 X_i}_{\text{Linear component}} + \underbrace{\varepsilon_i}_{\text{Random Error component}}$$

- The model to be estimated from sample data is:

$$Y_i = b_0 + b_1 X_i + e_i$$

Residual (random error from the sample)

- The actual estimated from the sample

Estimated (or predicted) Y value for observation i

Estimate of the regression intercept

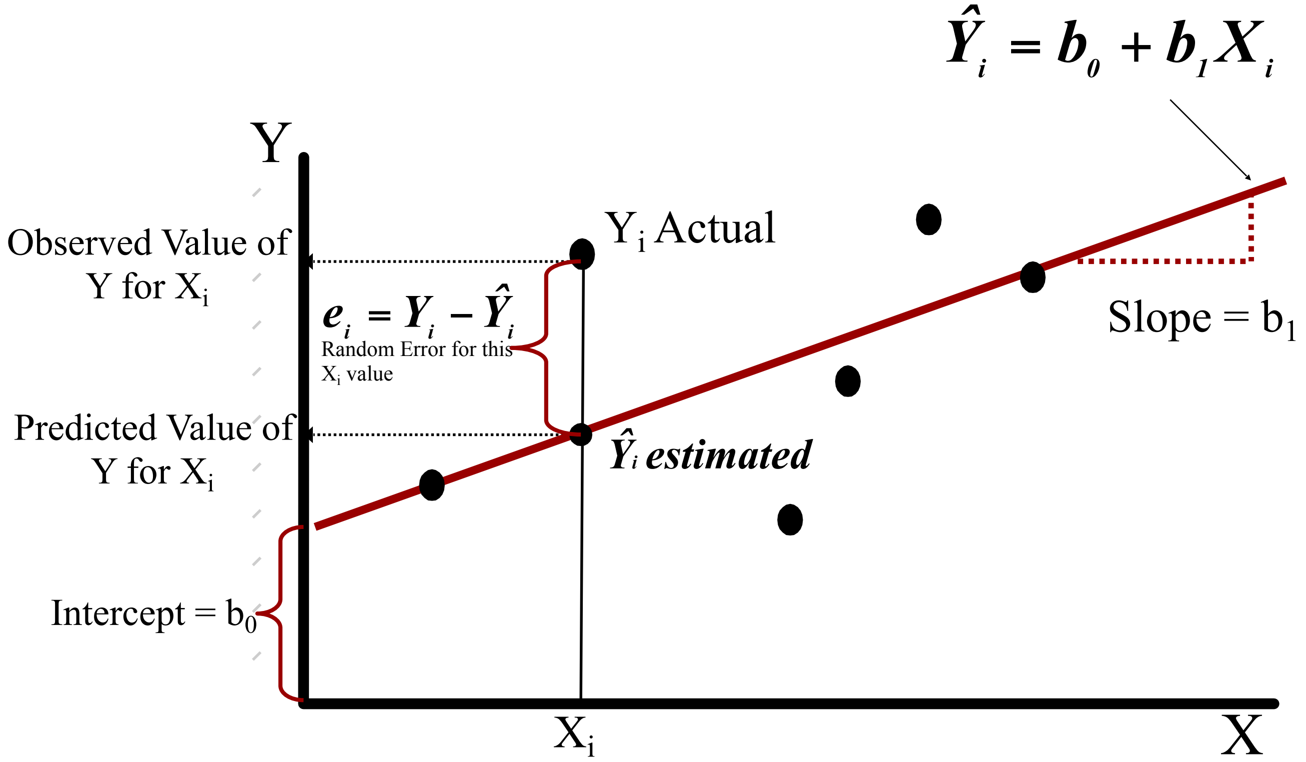
Estimate of the regression slope

$$\hat{Y}_i = b_0 + b_1 X_i$$

Value of X for observation i

 – Where $e_i = Y_i \quad \hat{Y}_i$

Simple Linear Regression Model



Interpretation of the slope and the intercept

$$\beta_0 = E(Y \mid X = 0)\ ; \quad \beta_1 = \Delta\, E(Y|X)/\Delta\,(X);$$

b_0 is the estimated average value of Y when the value of X is b_0 zero

b_1 is the estimated change in the average value of Y as a result of a one-unit change in X

Units of measurement of X and Y are very important for the correct interpretation of the slop and the intercept

Example: $\widehat{App\ Val} = 165.03 + 6.93\,(Lot\ size)$

Predict the app. Value of a house with 10,000 s.f. lot size

$$\widehat{App\ Val} = 165.03 + 6.93\,(10) = \$234{,}330$$

How Good is this prediction?

How Good is the Model's prediction Power?

Total variation is made up of two parts:

$$SST \quad = \quad SSR \quad + \quad SSE$$

Total Sum of Squares	Regression Sum of Squares	Error Sum of Squares

$$SST = \sum (Y_i - \bar{Y})^2 \quad SSR = \sum (\hat{Y}_i - \bar{Y})^2 \quad SSE = \sum (Y_i - \hat{Y}_i)^2$$

where:

$\bar{Y}$ = Average value of the dependent variable

Y_i = Observed values of the dependent variable

$\hat{Y}_i$ = Predicted value of Y for the given X_i value

SST = total sum of squares

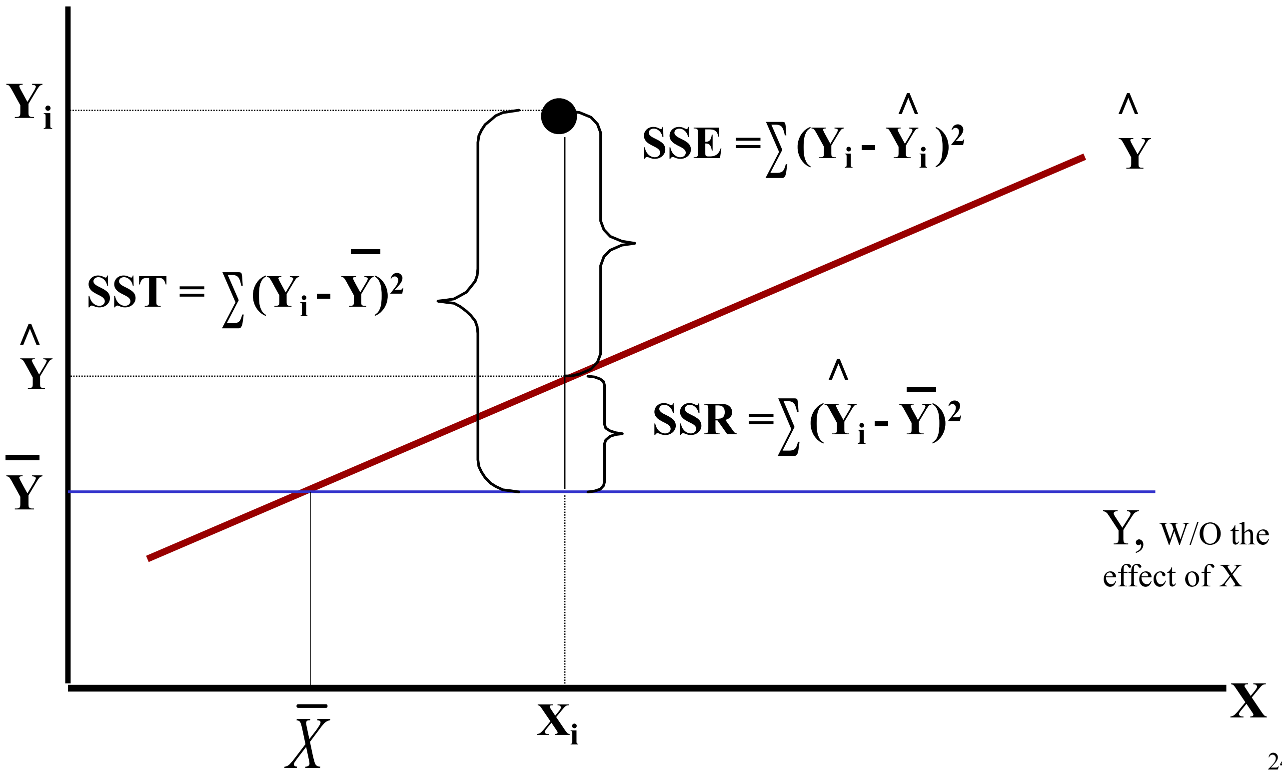
Measures total variation of the Y_i values around their mean

SSR = regression sum of squares (Explained)

Explained portion of total variation attributed to Y's relationship with X

SSE = error sum of squares (Unexplained)

Variation of Y values attributable to other factors than its relationship with X



How Good is the Model's prediction Power?

The coefficient of determination is the portion of the total variation in the dependent variable,Y, that is explained by variation in the independent variable, X

The coefficient of determination is also called r-squared and is denoted as r^2

$$r^2 = \frac{SSR}{SST} = \frac{\text{regression sum of squares}}{\text{total sum of squares}}$$

$$0 \le r^2 \le 1$$

Standard Error of Estimate

The standard deviation of the variation of observations around the regression line is estimated by

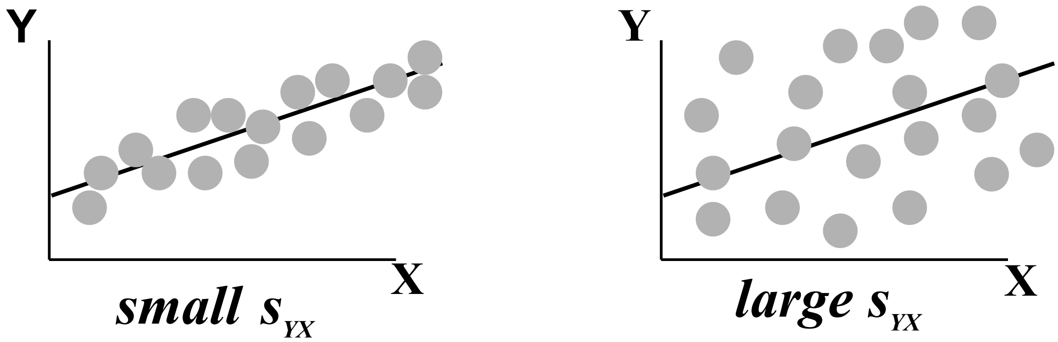
Where SSE = error sum of squares; n = sample size

$$S_{YX} = \sqrt{\frac{SSE}{n-2}} = \sqrt{\frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{n-2}} = \sqrt{MSE}$$

The concept is the same as the standard deviation (average difference) around the mean of a univariate

Comparing Standard Errors

S_{YX} is a measure of the variation of observed Y values from the regression line



The magnitude of S_{YX} should always be judged relative to the size of the Y values in the sample data

i.e., S_{YX} = \$36.34K is moderately small relative to house prices in the \$200 - \$300K range (average 215K)

Assumptions of Regression

Normality of Error

Error values (ε) are normally distributed for any given value of X

Homoscedasticity

The probability distribution of the errors has constant variance

Independence of Errors

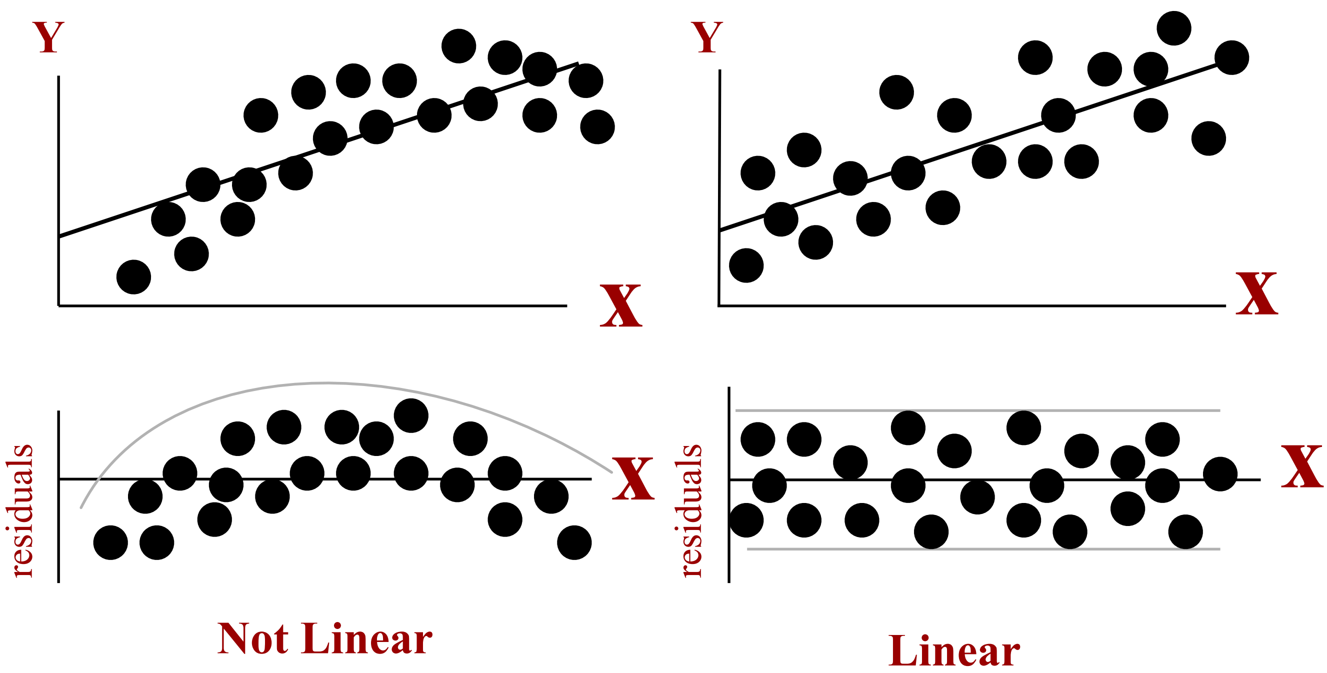
Error values are statistically independent

How to investigate the appropriateness of the fitted model

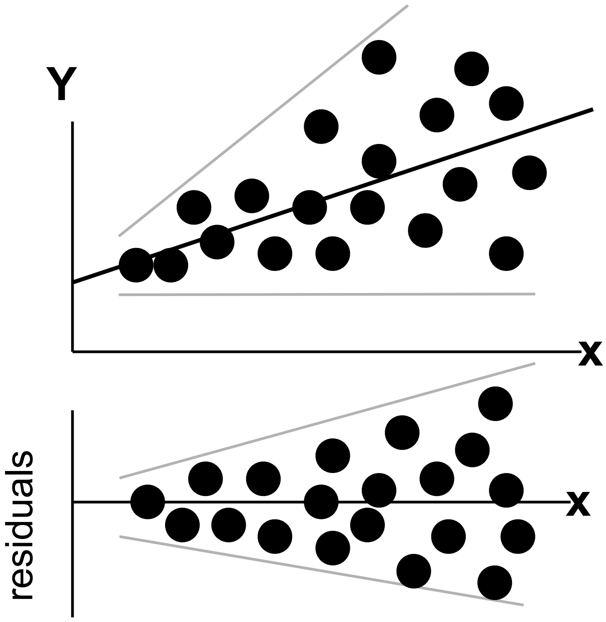
- The residual for observation i, e_i, is the difference between its observed and predicted value;
- Check the assumptions of regression by examining the residuals $e_i = Y_i - \hat{Y}_i$
  - Examine for linearity assumption
  - Examine for constant variance for all levels of X (homoscedasticity)
  - Evaluate normal distribution assumption
  - Evaluate independence assumption
- Graphical Analysis of Residuals

  Can plot residuals vs. X

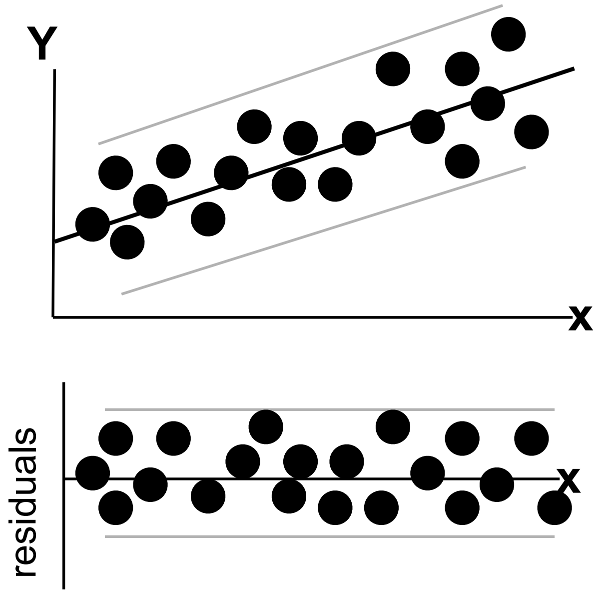
Residual Analysis for Linearity



Residual Analysis for Homoscedasticity



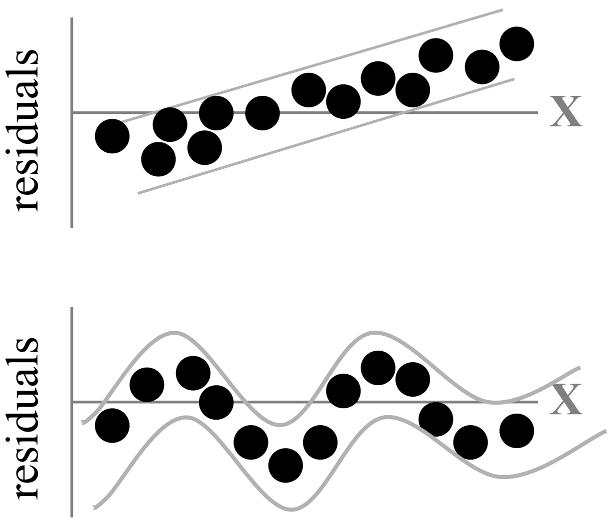
Non-constant variance



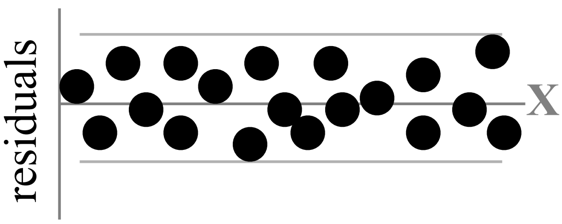
Constant variance

Residual Analysis for Independence

Not Independent



Independent



Testing differences in mean between two groups

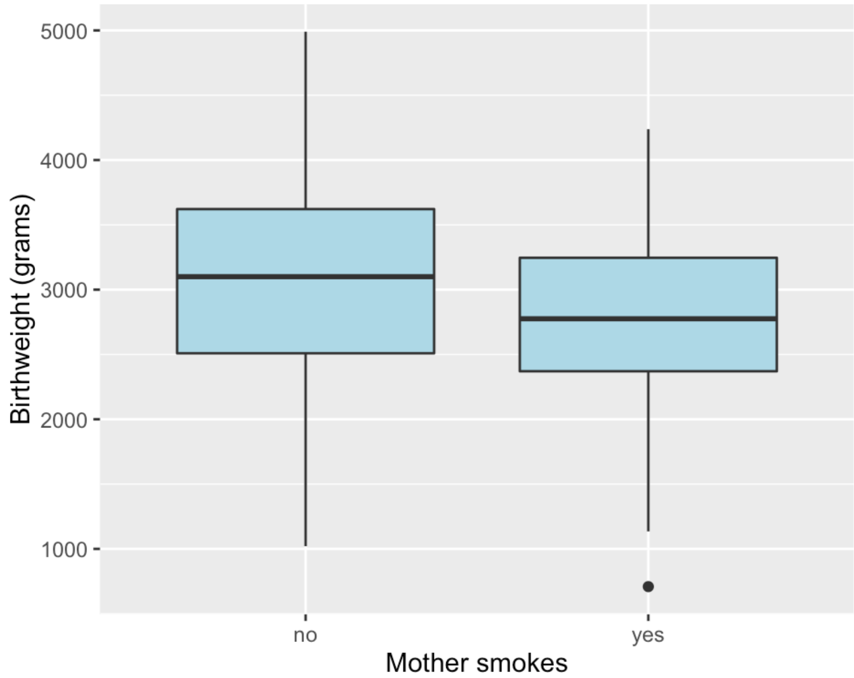
Let's begin by loading the packages we'll need to get started

```
library(MASS)
library(plyr)
library(ggplot2)
```

```
# Rename the columns to have more descriptive names
colnames(birthwt) <- c("birthwt.below.2500", "mother.age", "mother.weight",
    "race", "mother.smokes", "previous.prem.labor", "hypertension", "uterine.irr",
    "physician.visits", "birthwt.grams")

# Transform variables to factors with descriptive levels
birthwt <- transform(birthwt,
            race = as.factor(mapvalues(race, c(1, 2, 3),
                              c("white","black", "other"))),
            mother.smokes = as.factor(mapvalues(mother.smokes,
                              c(0,1), c("no", "yes"))),
            hypertension = as.factor(mapvalues(hypertension,
                              c(0,1), c("no", "yes"))),
            uterine.irr = as.factor(mapvalues(uterine.irr,
                              c(0,1), c("no", "yes")))
            )
```


To start, it always helps to plot things

```
# Create boxplot showing how birthwt.grams varies between
# smoking status
qplot(x = mother.smokes, y = birthwt.grams,
      geom = "boxplot", data = birthwt,
      xlab = "Mother smokes",
      ylab = "Birthweight (grams)",
      fill = I("lightblue"))
```



This plot suggests that smoking is associated with lower birth weight.

How can we assess whether this difference is statistically significant?

Let's compute a summary table

```
aggregate(birthwt.grams ~ mother.smokes, data = birthwt,
          FUN = function(x) {c(mean = mean(x), sd = sd(x))})
```

```
##   mother.smokes birthwt.grams.mean birthwt.grams.sd
## 1            no          3055.6957         752.6566
## 2           yes          2771.9189         659.6349
```

The standard deviation is good to have, but to assess statistical significance we really want to have the standard error (which the standard deviation adjusted by the group size).

```
aggregate(birthwt.grams ~ mother.smokes, data = birthwt,
          FUN = function(x) {c(mean = mean(x),
                                          se = sd(x) / sqrt(length(x)))})
```

```
##   mother.smokes birthwt.grams.mean birthwt.grams.se
## 1            no         3055.69565         70.18559
## 2           yes         2771.91892         76.68100
```


t-test via t.test()

This difference is looking quite significant. To run a two-sample t-test, we can simple use the `t.test()` function.

```
birthwt.t.test <- t.test(birthwt.grams ~ mother.smokes, data = birthwt)
birthwt.t.test
```

```
## 
##  Welch Two Sample t-test
## 
## data:  birthwt.grams by mother.smokes
## t = 2.7299, df = 170.1, p-value = 0.007003
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   78.57486 488.97860
## sample estimates:
##  mean in group no mean in group yes
##          3055.696          2771.919
```

p-value

```r
birthwt.t.test$p.value   # p-value
```

```
## [1] 0.007002548
```

```r
birthwt.t.test$estimate  # group means
```

```
##  mean in group no mean in group yes
##          3055.696          2771.919
```

```r
birthwt.t.test$conf.int  # confidence interval for difference
```

```
## [1]  78.57486 488.97860
## attr(,"conf.level")
## [1] 0.95
```


Workshop 8: t-test

Testing means between two groups

(a) Using the Cars93 data and the t.test() function, run a t-test to see if average MPG.highway is different between US and non-US vehicles. Try doing this both using the formula style input and the x, y style input.

(b) What is the confidence interval for the difference?

(c) Repeat part (a) using the wilcox.test() function.

(d) Are your results for (a) and (c) very different?

ANOVA and Regression

Prepare data

```r
library(MASS)
library(plyr)
library(ggplot2)
library(knitr)
```

```r
# Rename the columns to have more descriptive names
colnames(birthwt) <- c("birthwt.below.2500", "mother.age", "mother.weight",
    "race", "mother.smokes", "previous.prem.labor", "hypertension", "uterine.irr",
    "physician.visits", "birthwt.grams")

# Transform variables to factors with descriptive levels
birthwt <- transform(birthwt,
            race = as.factor(mapvalues(race, c(1, 2, 3),
                              c("white","black", "other"))),
            mother.smokes = as.factor(mapvalues(mother.smokes,
                              c(0,1), c("no", "yes"))),
            hypertension = as.factor(mapvalues(hypertension,
                              c(0,1), c("no", "yes"))),
            uterine.irr = as.factor(mapvalues(uterine.irr,
                              c(0,1), c("no", "yes")))
            )
```

One-way ANOVA example

Question: Is there a significant association between race and birthweight?

Here's a table showing the mean and standard error of birthweight by race.

```
aggregate(birthwt.grams ~ race, data = birthwt, FUN = mean)
```

```
##    race birthwt.grams
## 1 black      2719.692
## 2 other      2805.284
## 3 white      3102.719
```


Terminology: a k-way ANOVA is used to assess whether the mean of an outcome variable is constant across all combinations of k factors. The most common examples are 1-way ANOVA (looking at a single factor), and 2-way ANOVA (looking at two factors).

We'll use the `aov()` function. For convenience, `aov()` allows you to specify a formula.

```
summary(aov(birthwt.grams ~ race, data = birthwt))
```

```
##              Df   Sum Sq Mean Sq F value  Pr(>F)
## race          2  5015725 2507863   4.913 0.00834 **
## Residuals   186 94953931  510505
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Workshop 9: ANOVA

Let's form our favourite birthwt data set.

```
# Rename the columns to have more descriptive names
colnames(birthwt) <- c("birthwt.below.2500", "mother.age", "mother.weight",
    "race", "mother.smokes", "previous.prem.labor", "hypertension", "uterine.irr",
    "physician.visits", "birthwt.grams")

# Transform variables to factors with descriptive levels
birthwt <- transform(birthwt,
            race = as.factor(mapvalues(race, c(1, 2, 3),
                              c("white","black", "other"))),
            mother.smokes = as.factor(mapvalues(mother.smokes,
                              c(0,1), c("no", "yes"))),
            hypertension = as.factor(mapvalues(hypertension,
                              c(0,1), c("no", "yes"))),
            uterine.irr = as.factor(mapvalues(uterine.irr,
                              c(0,1), c("no", "yes")))
            )
```


(a) Create a new factor that categorizes the number of physician visits into three levels: 0, 1, 2, 3 or more.

Hint: One way of doing this is with mapvalues, by mapping all instances of 3, 4,… etc, to "3 or more".

(b) Run an ANOVA to determine whether the average birth weight varies across number of physician visits.

Quality Control using R

Use cases - Manufacturing

- Predictively model equipment failure rates
- Streamline inventory management
- Target energy-inefficient components
- Optimize factory floor space

- Sensor data to look at failures
- Quality management
- Identifying out-of-bounds manufacturing
- Visual inspection/computer vision
- Optimal run speeds
- Demand forecasting/inventory management

Manufacturing sample

Pellets density

- Company manufacture ceramic component which in pellets size (g/cm3)

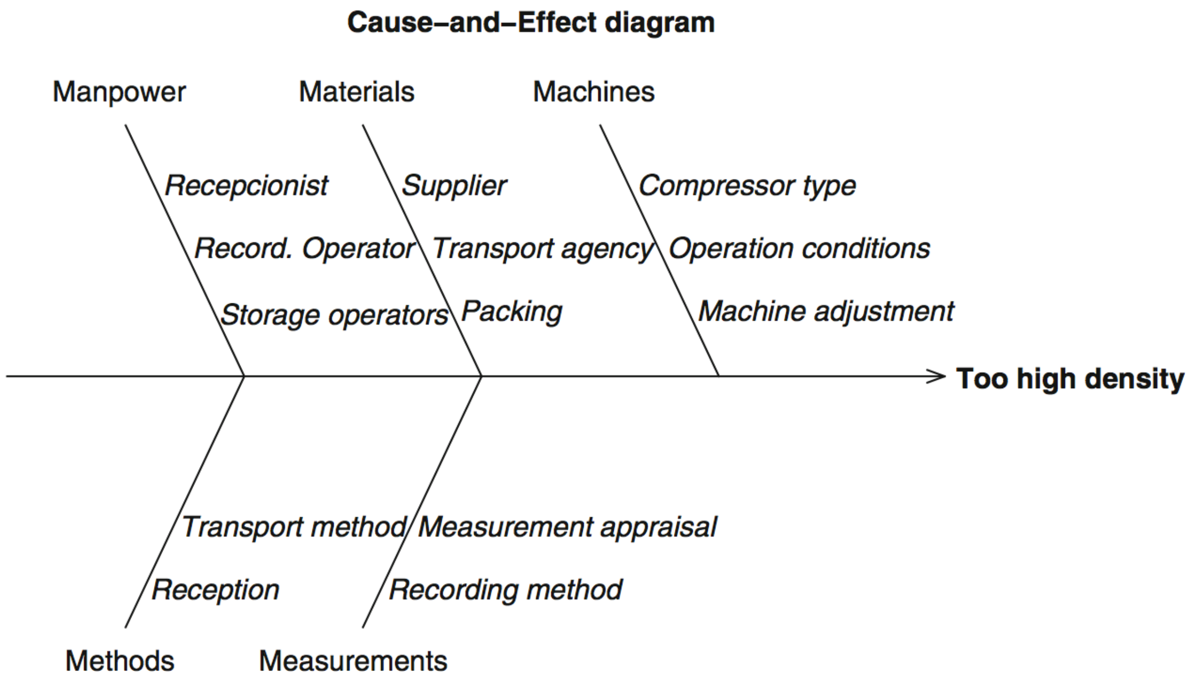
- Manpower
  - Receptionist
  - Recording Operator
  - Storage operators
- Materials
  - Supplier
  - Transport agency
  - Packing
- Machines
  - Compressor
  - Operation conditions
  - Machine adjustment
- Methods
  - Raw materials reception
  - Transport method
- Measurements
  - Recording method
  - Measurement apparatus

```r
cManpower <- c("Recepcionist", "Record. Operator",
               "Storage operators")
cMaterials <- c("Supplier", "Transport agency",
                "Packing")
cMachines <- c("Compressor type",
               "Operation conditions",
               "Machine adjustment")
cMethods <- c("Reception", "Transport method")
cMeasurements <- c("Recording method",
                   "Measurement appraisal")
cGroups <- c("Manpower", "Materials", "Machines",
             "Methods", "Measurements")
cEffect <- "Too high density"
```


Ishikawa diagram, or “fishbone”

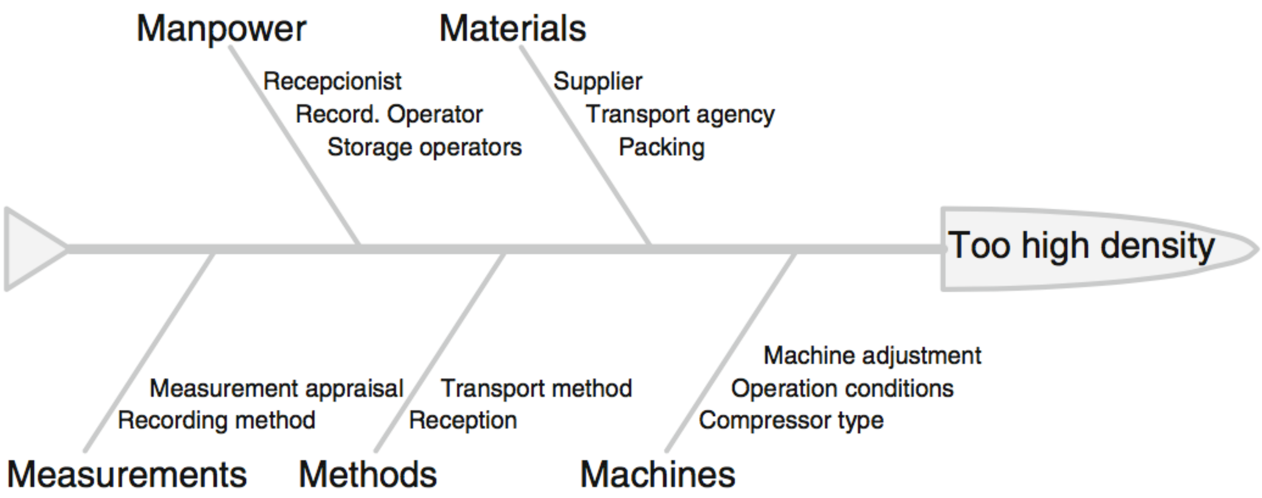
```r
library(qcc)
cause.and.effect(
  cause = list(Manpower = cManpower,
               Materials = cMaterials,
               Machines = cMachines,
               Methods = cMethods,
               Measurements = cMeasurements),
  effect = cEffect)
```

Ishikawa diagram, or “fishbone”




```r
library(SixSigma)
ss.ceDiag(
  effect = cEffect,
  causes.gr <- cGroups,
  causes = list(cManpower, cMaterials, cMachines,
                cMethods, cMeasurements),
  main = "Cause-and-effect diagram",
  sub = "Pellets Density")
```

Cause–and–effect diagram



Pellets Density

Out of control pellets density check sheet

Quality Control Department

31/01/2015

Instructions: Mark ticks for the more likely cause of the out-of-control point. Cross every four ticks to make five.

	Group	Cause	A_supplier	B_supplier	C_supplier
1	Manpower	Recepcionist	///		
2	Manpower	Record. Operator			/
3	Manpower	Storage operators		/	
4	Machines	Compressor type	//	/	卌 /
5	Machines	Operation conditions	/	//	
6	Machines	Machine adjustment	卌 //	/	//
7	Materials	Supplier	/	卌 卌 //	//
8	Materials	Transport agency	///	/	////
9	Materials	Packing	卌 /	//	///
10	Methods	Reception		/	
11	Methods	Transport method	/		/
12	Measurements	Recording method	//		
13	Measurements	Measurement appraisal		/	//

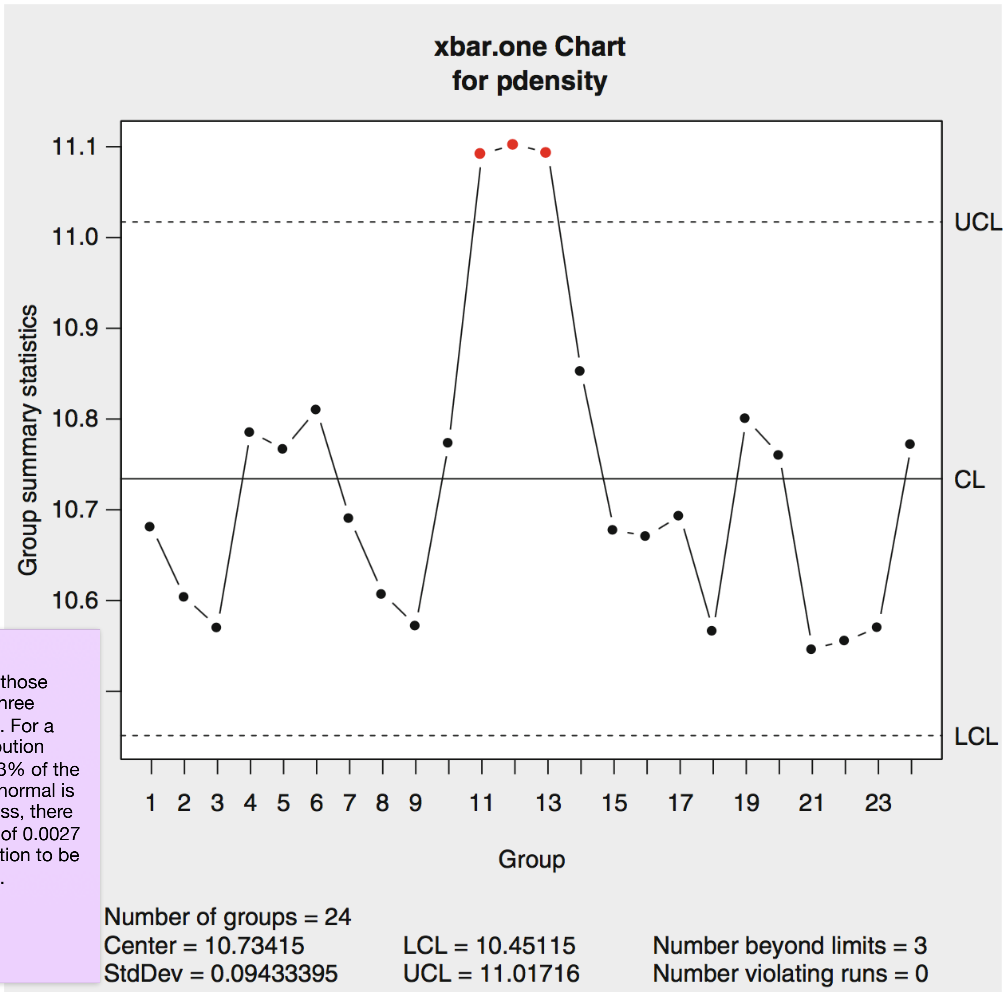
Week	Operator	Signature
2015-03	Emilio	

Control Chart

A control chart is a two-dimensional chart whose y-axis represents the variable we are monitoring. The x-axis of the chart is an identification of such j sample.

- Sample: We have a machine that generate a pallet with size (g/cm3) below:-

```
pdensity <- c(10.6817, 10.6040, 10.5709, 10.7858,
              10.7668, 10.8101, 10.6905, 10.6079,
              10.5724, 10.7736, 11.0921, 11.1023,
              11.0934, 10.8530, 10.6774, 10.6712,
              10.6935, 10.5669, 10.8002, 10.7607,
              10.5470, 10.5555, 10.5705, 10.7723)
```

Typical control limits are those between the mean and three standard deviations (´ 3). For a normal probability distribution these limits include 99.73% of the data. Thus, if nothing abnormal is taking place in the process, there will only be a probability of 0.0027 for an individual observation to be outside the control limits.

1. **Center line (CL)**: This is the central value the statistic should vary around. For example, the mean of the process;

2. **Lower control limit (LCL)**. This is the value below which it is very unlikely for the statistic to occur when the process is in control;

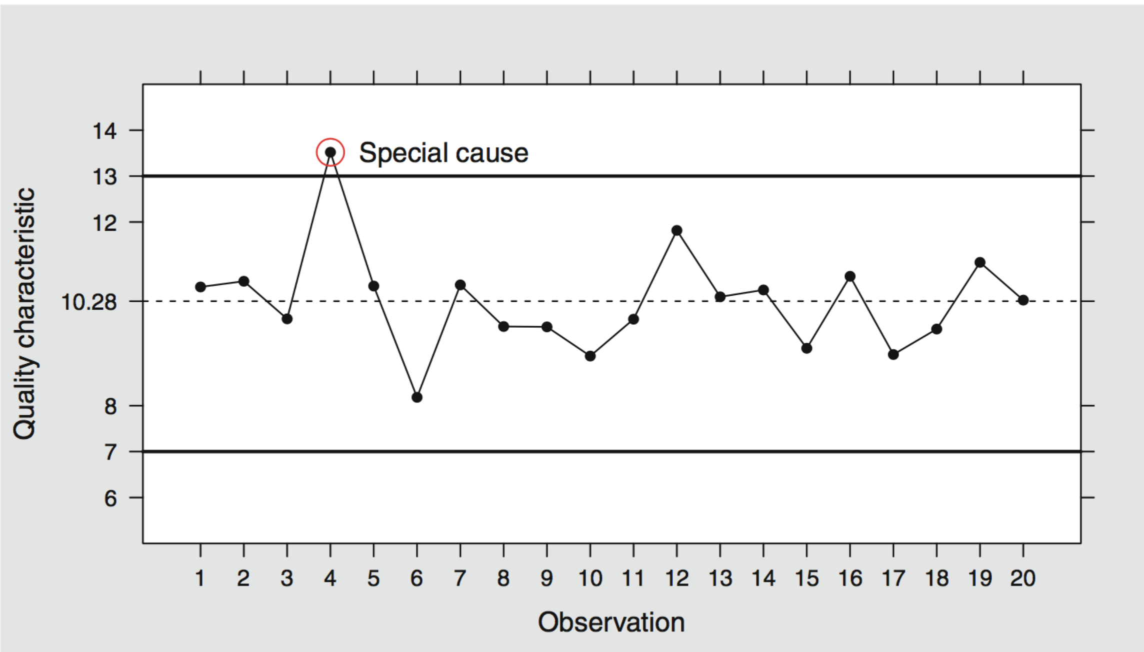
3. **Upper control limit (UCL)**. This is the counterpart of the LCL on the upper side of the CL. The LCL and UCL are symmetric if the probability distribution of the statistic to be monitored is symmetric (e.g., normal).


```
myControlChart <- qcc(data = pdensity,
                      type = "xbar.one")
summary(myControlChart)
```

```
##
## Call:
## qcc(data = pdensity, type = "xbar.one")
##
## xbar.one chart for pdensity
##
## Summary of group statistics:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   10.55   10.60   10.69   10.73   10.79   11.10
##
## Group sample size:  1
## Number of groups:  24
## Center of group statistics:  10.73415
## Standard deviation:  0.09433395
##
## Control limits:
##       LCL      UCL
##  10.45115 11.01716
```

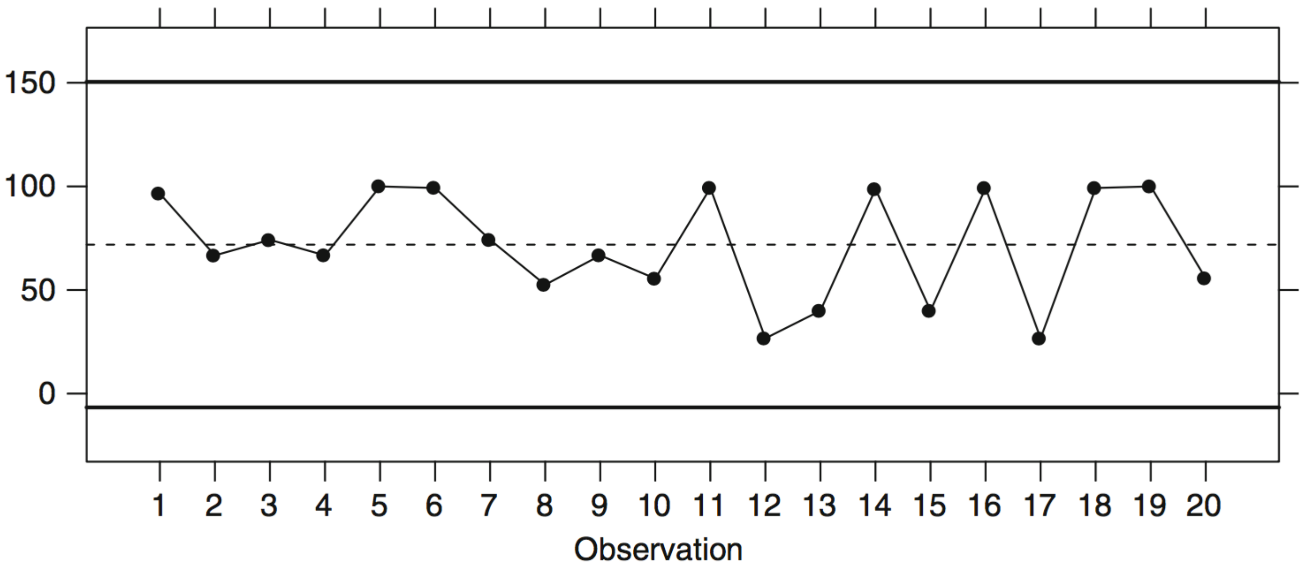
```
myControlChart$violations

  ## $beyond.limits
  ## [1] 11 12 13
  ##
  ## $violating.runs
  ## numeric(0)
```


Check Special Cause



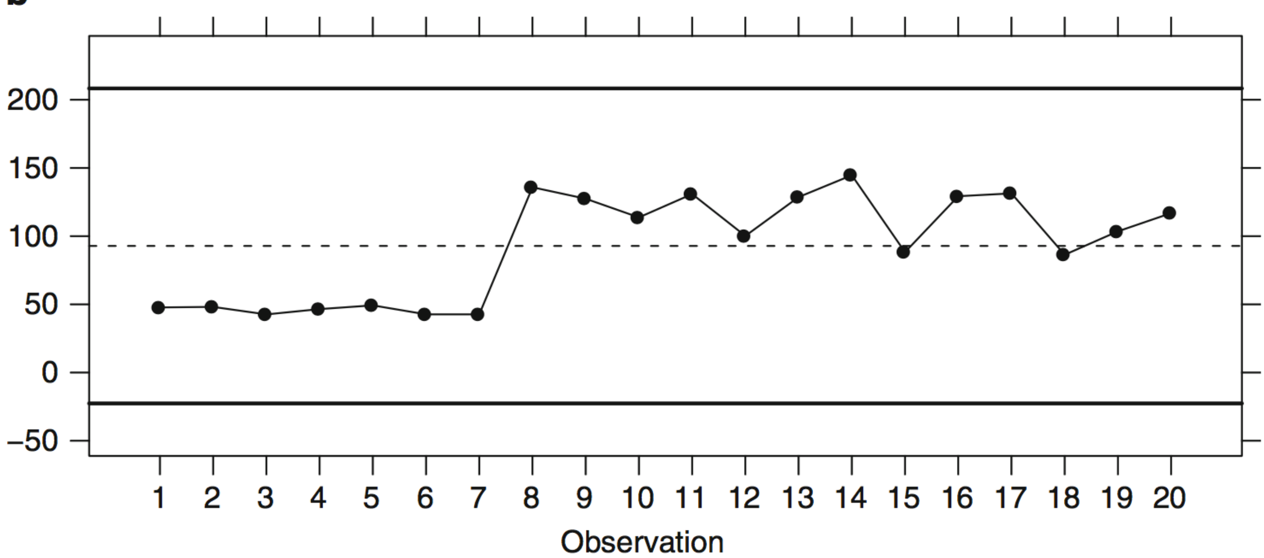
Seasonality

a

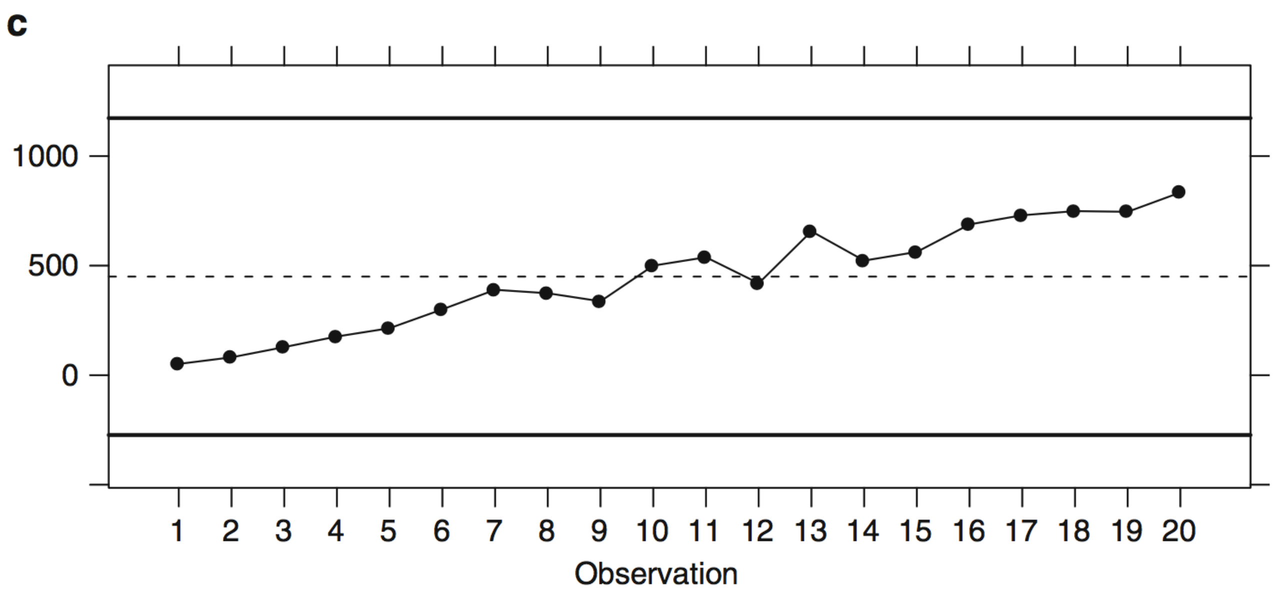


Shift

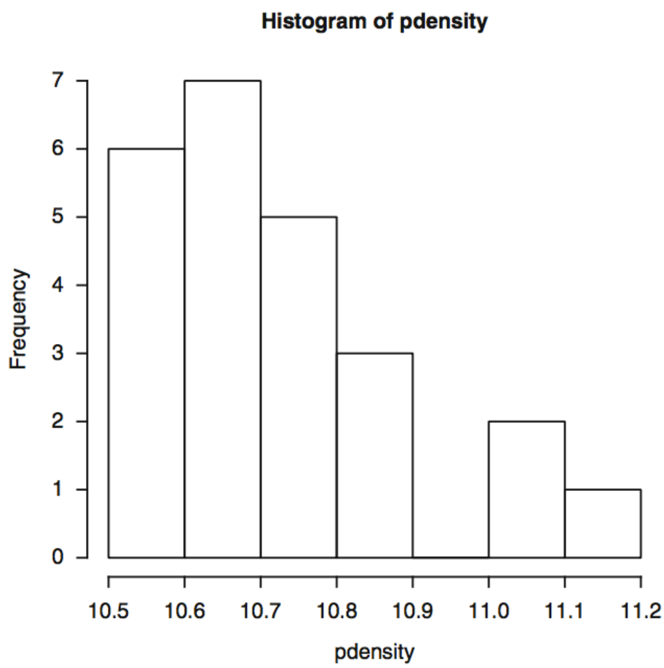
b



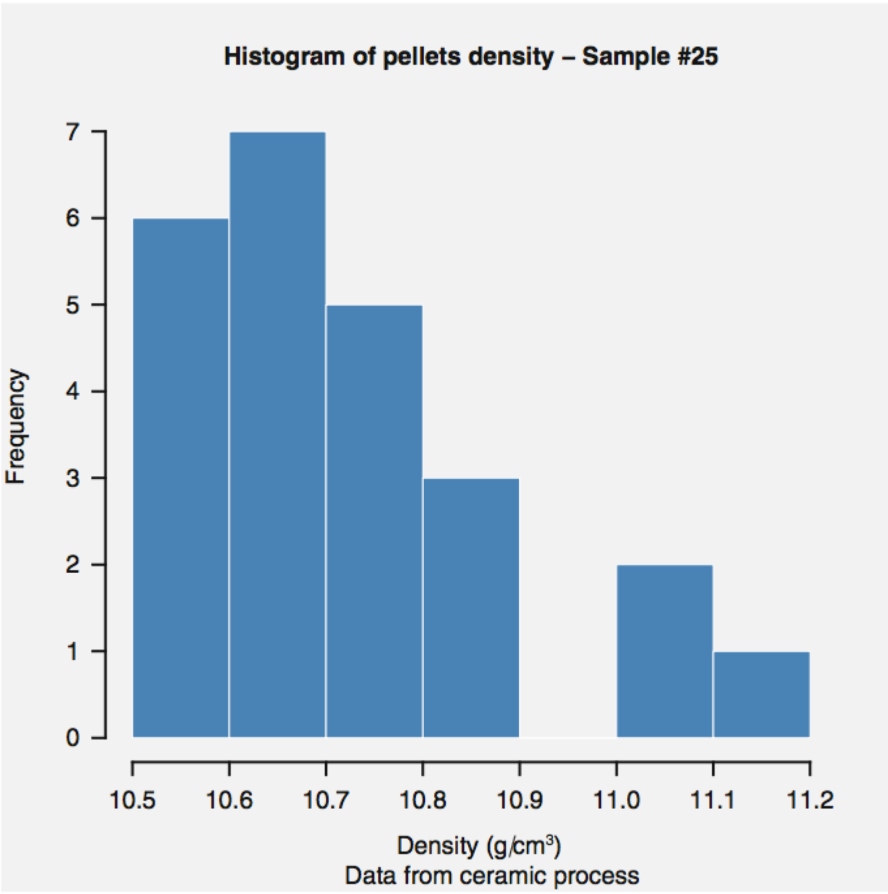
Trend



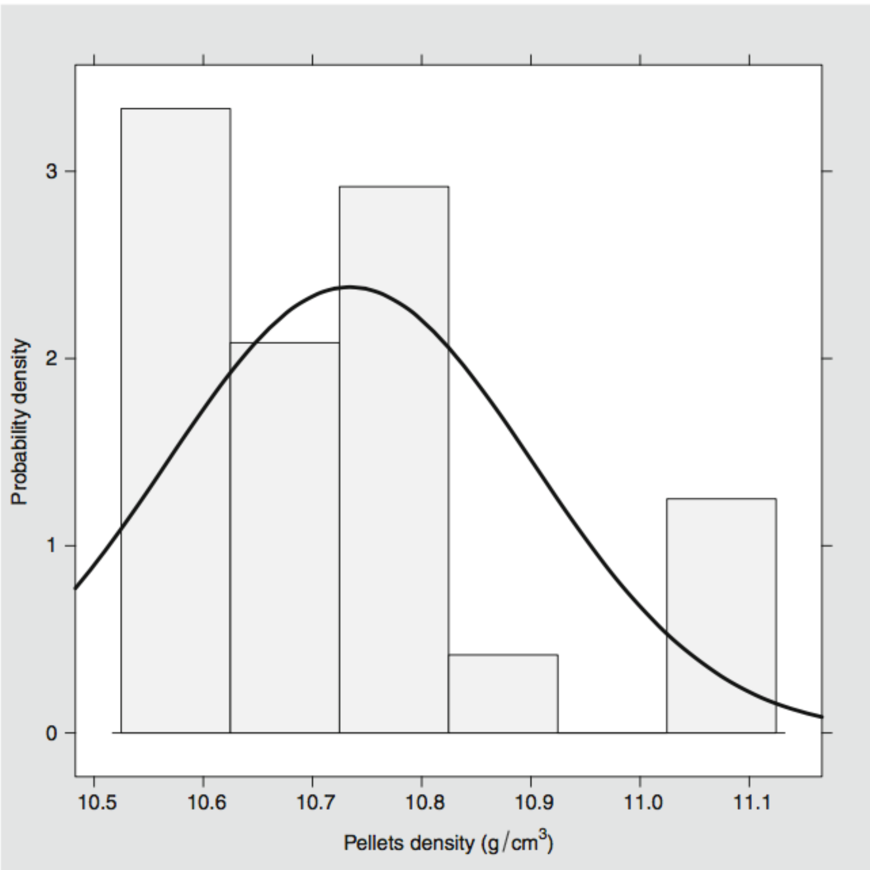

```
hist(pdensity)
```



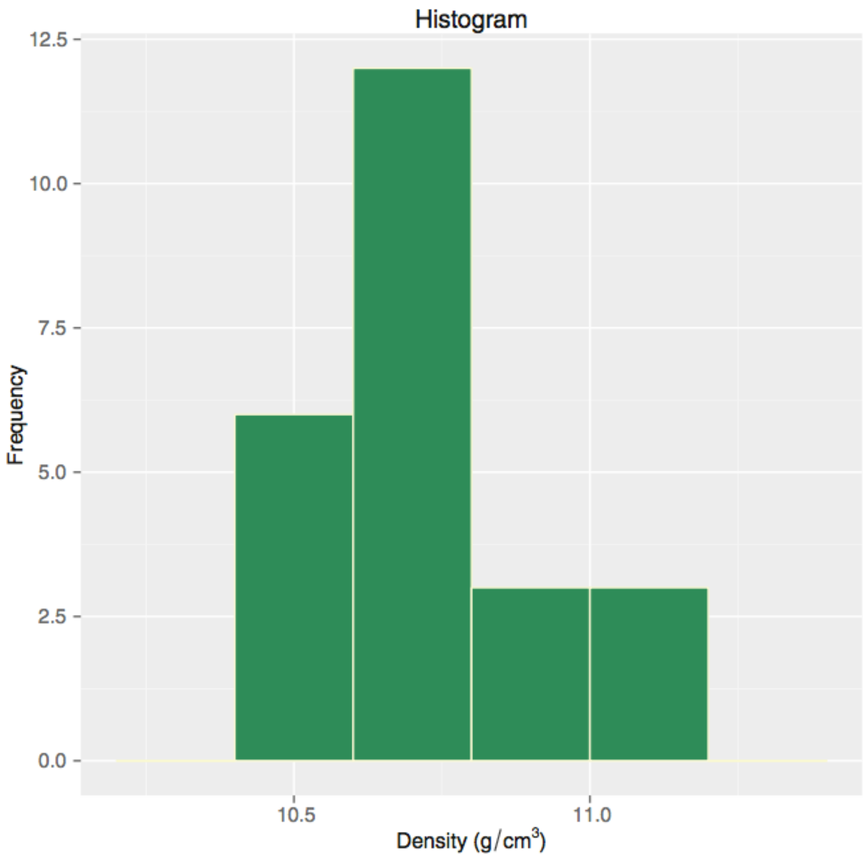
```
par(bg = "gray95")
hist(pdensity,
     main = "Histogram of pellets density - Sample #25",
     sub = "Data from ceramic process",
     xlab = expression("Density (g"/"cm"^3*")"),
     col = "steelblue",
     border = "white",
     lwd = 2,
     las = 1,
     bg = "gray")
```

```r
library(lattice)
histogram(pdensity,
    xlab = expression("Pellets density (g"/"cm"^3*")"),
    ylab = "Probability density",
    type = "density",
    panel = function(x, ...) {
        panel.histogram(x, ...)
        panel.mathdensity(dmath = dnorm,
                          col = "black",
                          lwd = 3,
                          args = list(mean = mean(x),
                                      sd = sd(x)))
    } )
```

```r
library(ggplot2)
ggplot(data = data.frame(pdensity),
       aes(x = pdensity)) +
  geom_histogram(fill = "seagreen",
                 colour = "lightgoldenrodyellow",
                 binwidth = 0.2) +
  labs(title = "Histogram",
       x = expression("Density ("*g/cm^3*")"),
       y = "Frequency")
```

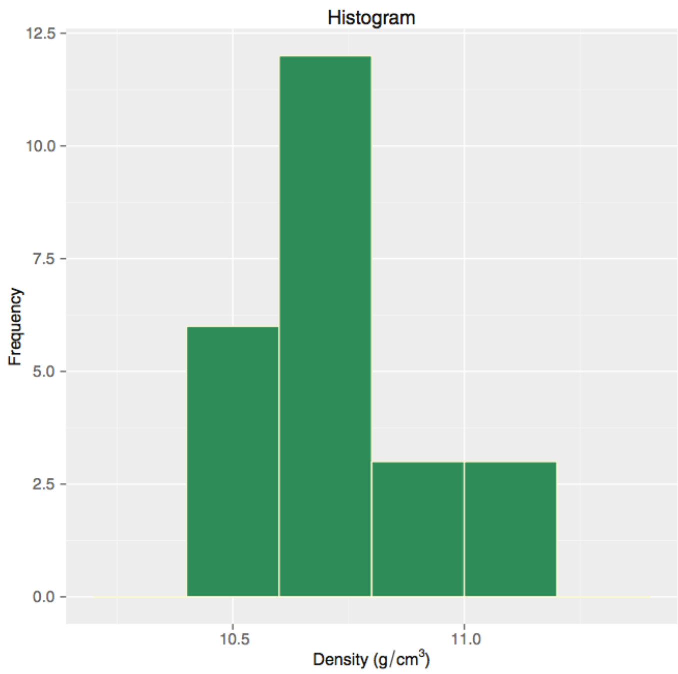
```r
data_checkSheet$A_supplier <- c(2, 0, 0, 2, 1, 7, 1,
                                3, 6, 0, 1, 2, 0)
data_checkSheet$B_supplier <- c(0, 0, 1, 1, 2, 1, 12,
                                1, 2, 1, 0, 0, 1)
data_checkSheet$C_supplier <- c(0, 1, 0, 6, 0, 2, 2,
                                4, 3, 0, 1, 0, 2)
data_checkSheet$Total <- data_checkSheet$A_supplier +
  data_checkSheet$B_supplier +
  data_checkSheet$C_supplier
```



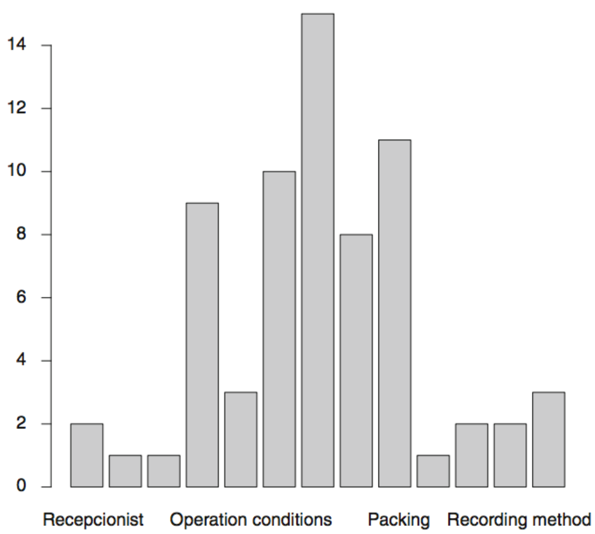
```r
data_checkSheet
```

```
##           Group                   Cause A_supplier
## 1      Manpower            Recepcionist          2
## 2      Manpower        Record. Operator          0
## 3      Manpower       Storage operators          0
## 4      Machines         Compressor type          2
## 5      Machines   Operation conditions          1
## 6      Machines      Machine adjustment          7
## 7     Materials                Supplier          1
## 8     Materials        Transport agency          3
## 9     Materials                 Packing          6
## 10      Methods               Reception          0
## 11      Methods        Transport method          1
## 12 Measurements        Recording method          2
## 13 Measurements Measurement appraisal          0
##    B_supplier C_supplier Total
## 1           0          0     2
## 2           0          1     1
## 3           1          0     1
## 4           1          6     9
## 5           2          0     3
## 6           1          2    10
## 7          12          2    15
```

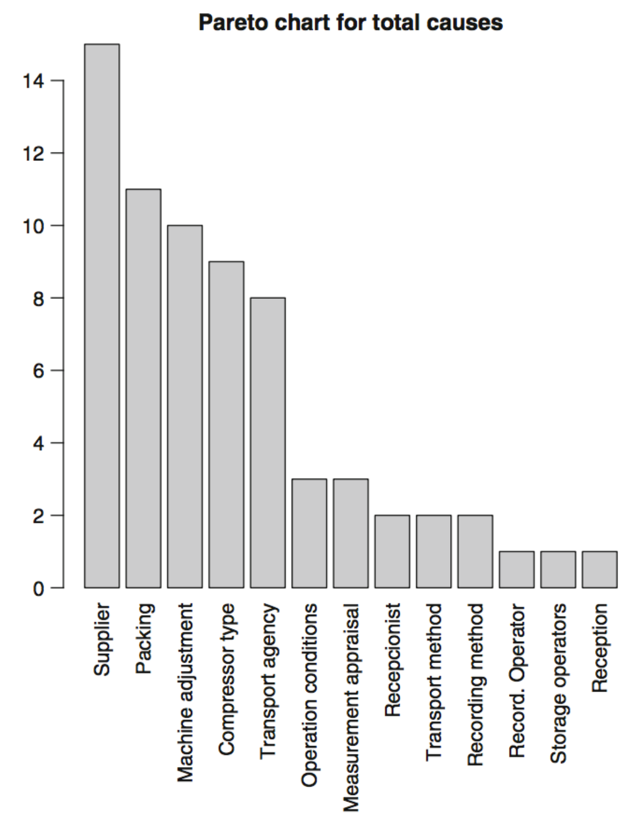
```r
barplot(height = data_checkSheet$Total,
        names.arg = data_checkSheet$Cause)
```



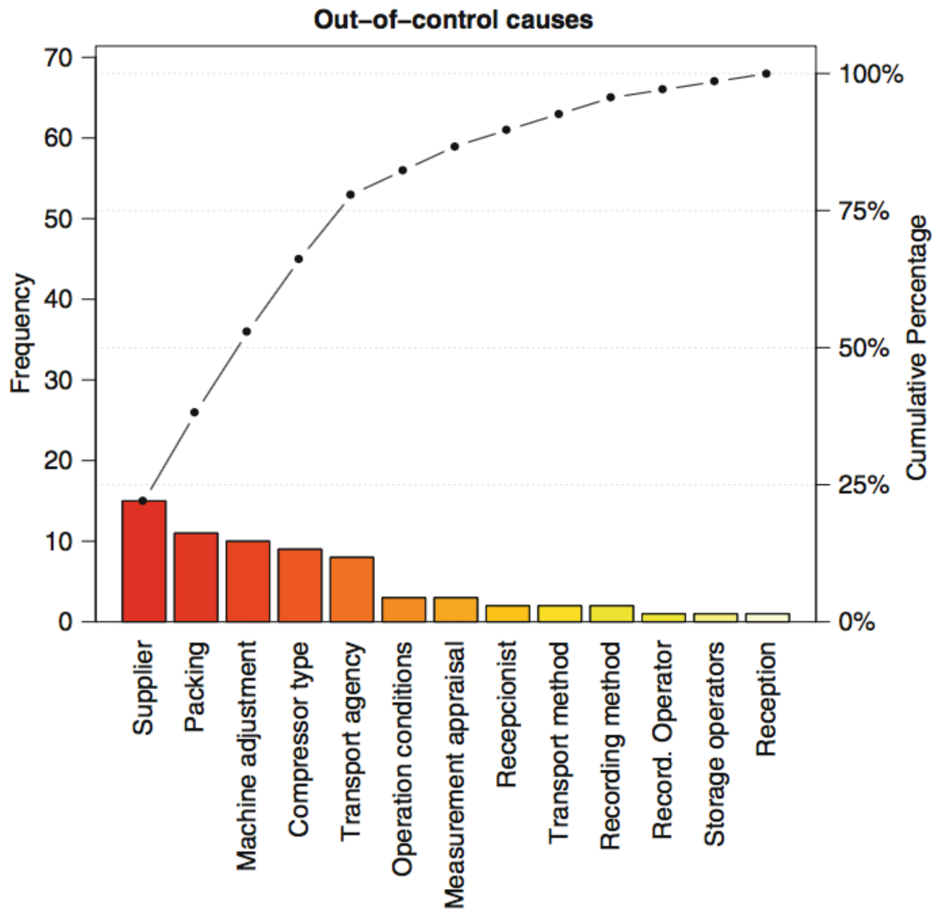

```r
barplot(height = data_checkSheet$Total,
        names.arg = data_checkSheet$Cause)
```



```
data_pareto <- data_checkSheet[order(
  data_checkSheet$Total,
  decreasing = TRUE), ]
par(mar = c(8, 4, 4, 2) + 0.1)
barplot(height = data_pareto$Total,
        names.arg = data_pareto$Cause,
        las = 2,
        main = "Pareto chart for total causes")
```

```r
library(qcc)
data_pareto2 <- data_pareto$Total
names(data_pareto2) <- data_pareto$Cause
pareto.chart(x = data_pareto2,
             main = "Out-of-control causes")
```

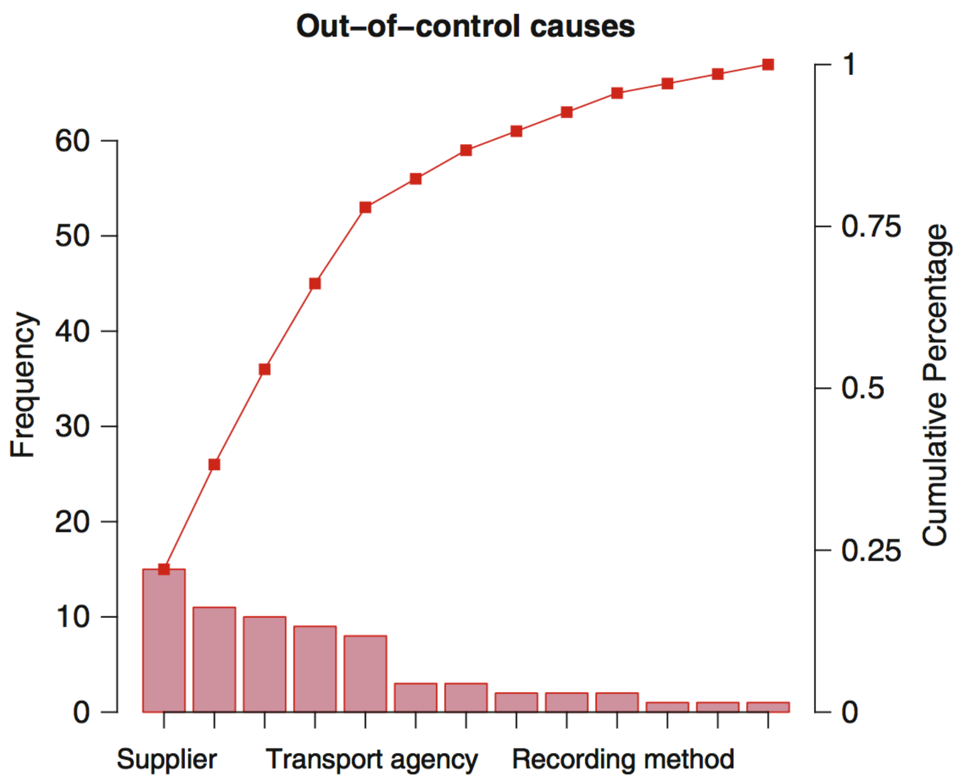
```
##
## Pareto chart analysis for data_pareto2
##                          Frequency Cum.Freq.
##   Supplier                      15        15
##   Packing                       11        26
##   Machine adjustment            10        36
##   Compressor type                9        45
##   Transport agency               8        53
##   Operation conditions           3        56
##   Measurement appraisal          3        59
##   Recepcionist                   2        61
##   Transport method               2        63
##   Recording method               2        65
##   Record. Operator               1        66
##   Storage operators              1        67
##   Reception                      1        68
##
## Pareto chart analysis for data_pareto2
##                          Percentage Cum.Percent.
##   Supplier                22.058824     22.05882
##   Packing                 16.176471     38.23529
##   Machine adjustment      14.705882     52.94118
##   Compressor type         13.235294     66.17647
##   Transport agency        11.764706     77.94118
##   Operation conditions     4.411765     82.35294
##   Measurement appraisal    4.411765     86.76471
##   Recepcionist             2.941176     89.70588
##   Transport method         2.941176     92.64706
##   Recording method         2.941176     95.58824
##   Record. Operator         1.470588     97.05882
##   Storage operators        1.470588     98.52941
##   Reception                1.470588    100.00000
```



```
library(qualityTools)
paretoChart(x = data_pareto2,
            main = "Out-of-control causes")
```

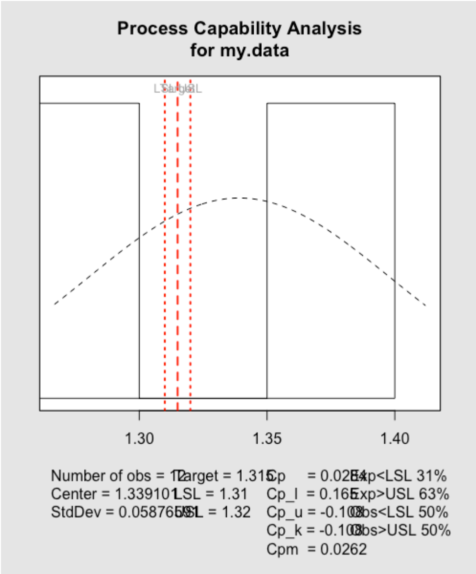
```
##
## Frequency           15    11    10     9     8     3
## Cum. Frequency      15    26    36    45    53    56
## Percentage       22.1% 16.2% 14.7% 13.2% 11.8%  4.4%
## Cum. Percentage  22.1% 38.2% 52.9% 66.2% 77.9% 82.4%
```

```
##
## Frequency          3    2    2    2    1    1
## Cum. Frequency    59   61   63   65   66   67
## Percentage       4.4% 2.9% 2.9% 2.9% 1.5% 1.5%
## Cum. Percentage 86.8% 89.7% 92.6% 95.6% 97.1% 98.5%
##
## Frequency          1
## Cum. Frequency    68
## Percentage       1.5%
## Cum. Percentage 100.0%
##
## Frequency       15.00000 11.00000 10.00000  9.00000
## Cum. Frequency  15.00000 26.00000 36.00000 45.00000
## Percentage      22.05882 16.17647 14.70588 13.23529
## Cum. Percentage 22.05882 38.23529 52.94118 66.17647
##
## Frequency        8.00000  3.000000  3.000000
## Cum. Frequency  53.00000 56.000000 59.000000
## Percentage      11.76471  4.411765  4.411765
## Cum. Percentage 77.94118 82.352941 86.764706
##
## Frequency        2.000000  2.000000  2.000000
## Cum. Frequency  61.000000 63.000000 65.000000
## Percentage       2.941176  2.941176  2.941176
## Cum. Percentage 89.705882 92.647059 95.588235
##
## Frequency        1.000000  1.000000   1.000000
## Cum. Frequency  66.000000 67.000000  68.000000
## Percentage       1.470588  1.470588   1.470588
## Cum. Percentage 97.058824 98.529412 100.000000
```

Frequency	15	11	10	9	8	3	3	2	2	2	1	1	1
Cum. Frequency	15	26	36	45	53	56	59	61	63	65	66	67	68
Percentage	22	16	15	13	12	4	4	3	3	3	1	1	1
Cum. Percentage	22	38	53	66	78	82	87	90	93	96	97	99	100

```r
lsl <- 10.85 # Fill in YOUR LSL here!
usl <- 10.95 # Fill in YOUR USL here!
process.capability(myControlChart, spec.limits=c(lsl,usl))
```




```r
plot(pdensity ~ ptemp,
     col = "gray40",
     pch = 20,
     main = "Pellets density vs. temperature",
     xlab = "Temperature (Celsius)",
     ylab = expression("Density ("*g/cm^3*")"))
```

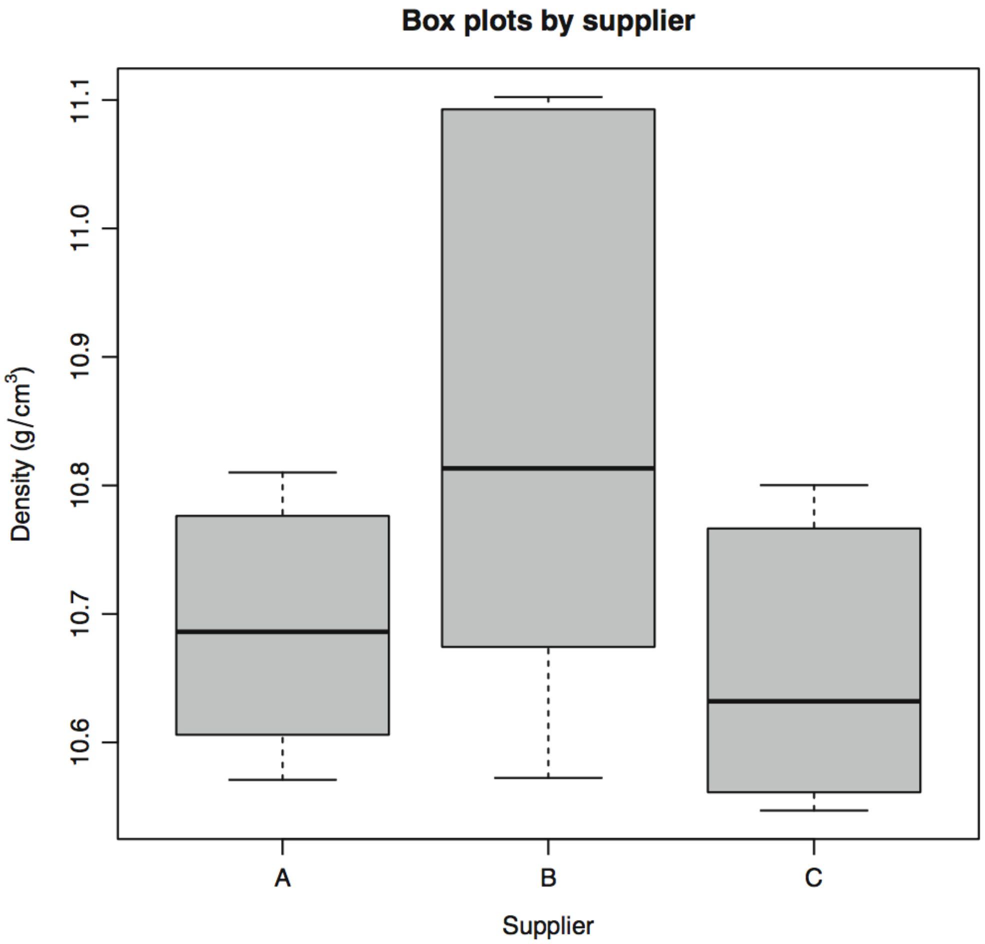
```r
library(qicharts)
spreadvector <- rep(names(data_pareto2),
    times = data_pareto2)
paretochart(spreadvector)
```

```
##                         Frequency
## Supplier                       15
## Packing                        11
## Machine adjustment             10
## Compressor type                 9
## Transport agency                8
## Measurement appraisal           3
## Operation conditions            3
## Recepcionist                    2
## Recording method                2
## Transport method                2
## Reception                       1
## Record. Operator                1
## Storage operators               1
##                         Cumulative Frequency
## Supplier                                  15
## Packing                                   26
## Machine adjustment                        36
## Compressor type                           45
```



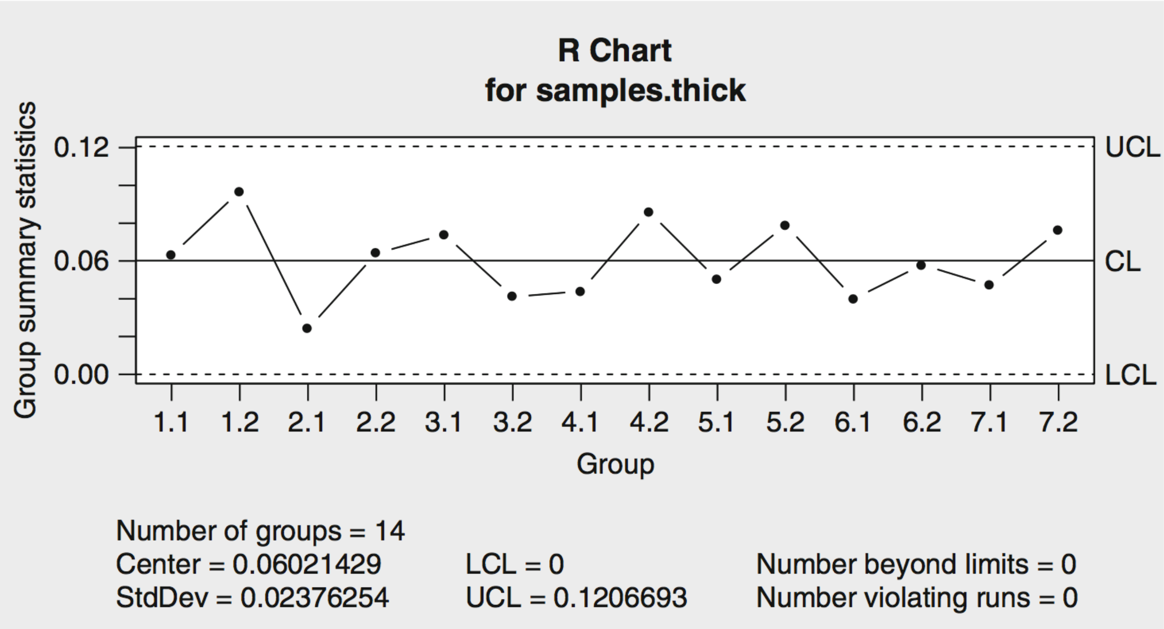
```r
psupplier <- rep(c("A", "B", "C"), each = 8)
```

```r
boxplot(pdensity ~ psupplier,
        col = "gray70",
        xlab = "Supplier",
        ylab = expression("Density ("*g/cm^3*")"),
        main = "Box plots by supplier")
```



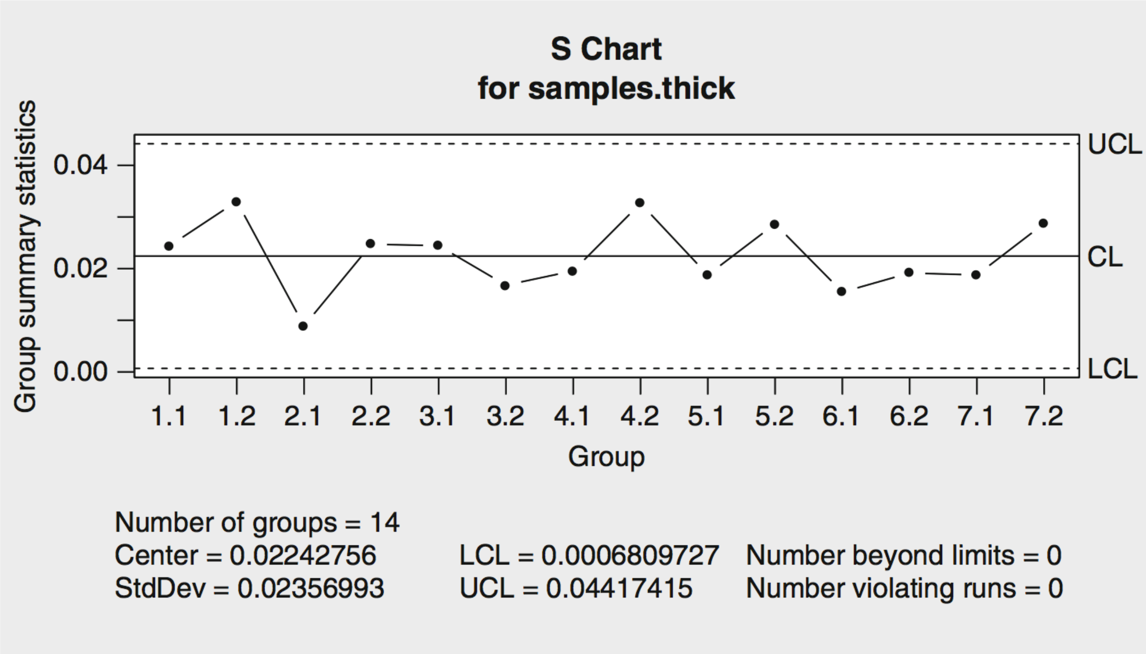
Range Chart

```r
r.thick <- qcc(data = samples.thick, type = "R")
```



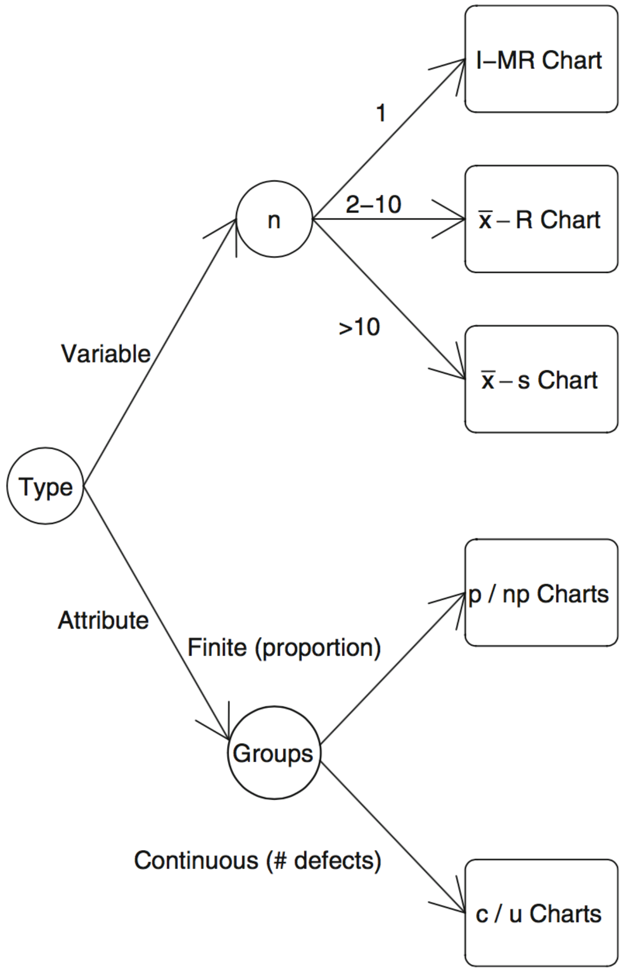
Standard Deviation Chart

```
r.thick <- qcc(data = samples.thick, type = "S")
```



Other QCC Chart

- Attribute Data (Discrete Data)
    - n / np QCC Chart
- Error per Interval basis
    - c Chart
- Defect per items
    - u Chart



Workshop 11

Use the data frame ss.data.thickness2 is in the SixSigma package to create control chart for x-bar

Machine Learning for Data Science

What is Machine Learning?

- **Fundamental Question of Computer Science:**

How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?

Why machine learning?

- Learn relationships from large sets of complex data: Data mining
  - Predict clinical outcome from tests
  - Decide whether someone is a good credit risk
- Do tasks too complex to program by hand
  - Autonomous driving
- Customize programs to users/customers need
  - Recommend book/movie based on previous likes

Why Machine Learning?

- Economically efficient
- Can consider larger data spaces and hypothesis spaces than people can
- Can formalize learning problem to explicitly identify/ describe goals and criteria

Machine Learning Paradigms

- **Supervised Learning**
  - Classification
  - Regression

- **Unsupervised Learning**
  - Clustering

- **Re-enforcement Learning**

Supervised Learning

Approaches
- Classification (discrete predictions)
- Regression (continuous predictions)

- Common considerations
  - Representation (Features)
  - Feature Selection
  - Functional form
  - Evaluation of predictive power

Classification vs Regression

- If I want to predict whether a patient will die from a disease within six months, that is **classification**
- If I want to predict how long the patient will live, that is **regression**

Representation

Definition of thing or things to be predicted

– Classification: classes

– Regression: regression variable

• Definition of things (instances) to make predictions for

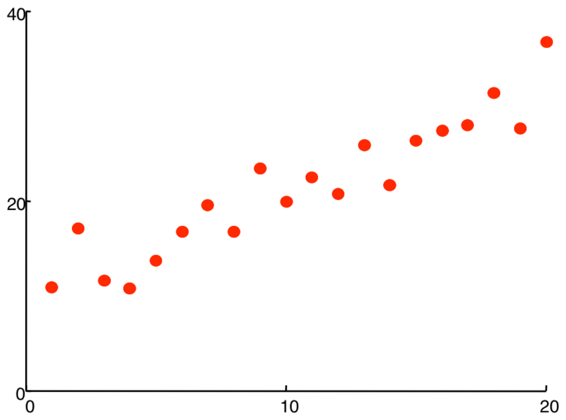
– Individuals

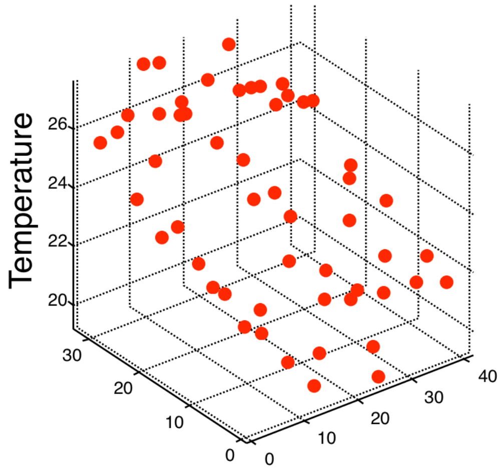
– Families

– Neighborhoods, etc.

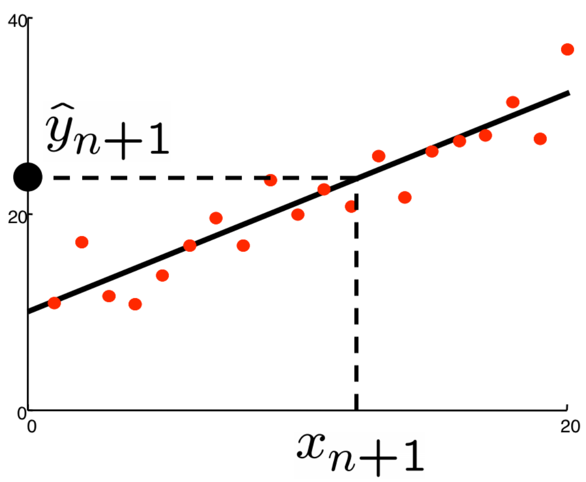
• Choice of descriptors (features) to describe different aspects of instances

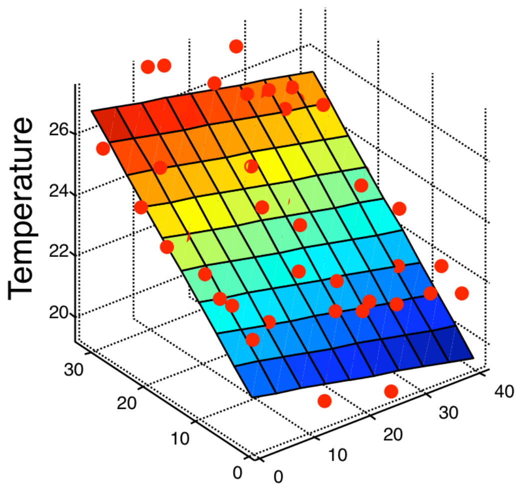
Linear Regression



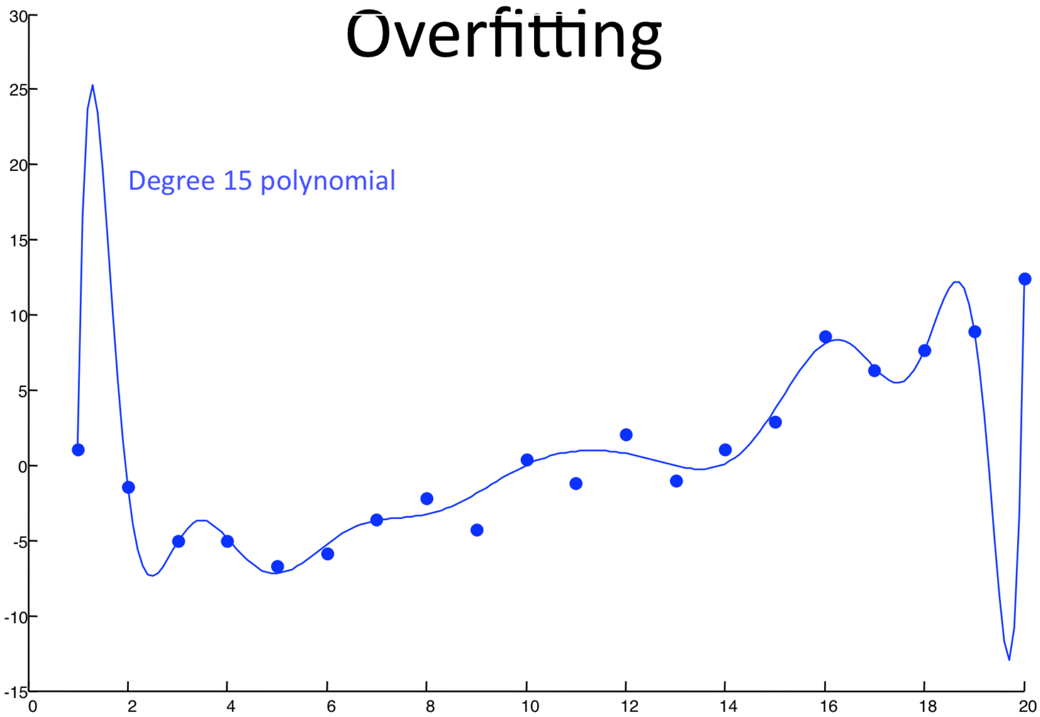


Linear Regression





Overfitting



Linear Regression

Let’s first import the data set and see what we’re working with.

```
# Import data set
crime <- read.table("crime_simple.txt", sep = "\t", header = TRUE)
```

The variable names that this data set comes with are very confusing, and even misleading.

R: Crime rate: # of offenses reported to police per million population

Age: The number of males of age 14-24 per 1000 population

S: Indicator variable for Southern states (0 = No, 1 = Yes)

Ed: Mean # of years of schooling x 10 for persons of age 25 or older

Ex0: 1960 per capita expenditure on police by state and local government

Ex1: 1959 per capita expenditure on police by state and local government

LF: Labor force participation rate per 1000 civilian urban males age 14-24

M: The number of males per 1000 females

N: State population size in hundred thousands

NW: The number of non-whites per 1000 population

U1: Unemployment rate of urban males per 1000 of age 14-24

U2: Unemployment rate of urban males per 1000 of age 35-39

W: Median value of transferable goods and assets or family income in tens of $

X: The number of families per 1000 earning below 1/2 the median income

We really need to give these variables better names

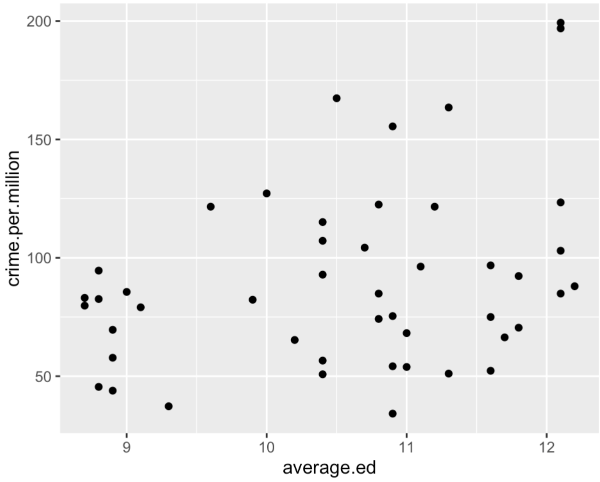
```
# Assign more meaningful variable names
colnames(crime) <- c("crime.per.million", "young.males", "is.south", "average.ed",
                     "exp.per.cap.1960", "exp.per.cap.1959", "labour.part",
                     "male.per.fem", "population", "nonwhite",
                     "unemp.youth", "unemp.adult", "median.assets", "num.low.salary")

# Convert is.south to a factor
# Divide average.ed by 10 so that the variable is actually average education
# Convert median assets to 1000's of dollars instead of 10's
crime <- transform(crime, is.south = as.factor(is.south),
                          average.ed = average.ed / 10,
                          median.assets = median.assets / 100)

# print summary of the data
summary(crime)
```



```
##  crime.per.million  young.males    is.south   average.ed
##  Min.   : 34.20    Min.   :119.0   0:31      Min.   : 8.70
##  1st Qu.: 65.85    1st Qu.:130.0   1:16      1st Qu.: 9.75
##  Median : 83.10    Median :136.0             Median :10.80
##  Mean   : 90.51    Mean   :138.6             Mean   :10.56
##  3rd Qu.:105.75    3rd Qu.:146.0             3rd Qu.:11.45
##  Max.   :199.30    Max.   :177.0             Max.   :12.20
##  exp.per.cap.1960 exp.per.cap.1959  labour.part      male.per.fem
##  Min.   : 45.0    Min.   : 41.00   Min.   :480.0   Min.   : 934.0
##  1st Qu.: 62.5    1st Qu.: 58.50   1st Qu.:530.5   1st Qu.: 964.5
##  Median : 78.0    Median : 73.00   Median :560.0   Median : 977.0
##  Mean   : 85.0    Mean   : 80.23   Mean   :561.2   Mean   : 983.0
##  3rd Qu.:104.5    3rd Qu.: 97.00   3rd Qu.:593.0   3rd Qu.: 992.0
##  Max.   :166.0    Max.   :157.00   Max.   :641.0   Max.   :1071.0
##    population        nonwhite      unemp.youth       unemp.adult
##  Min.   :  3.00   Min.   :  2.0   Min.   : 70.00   Min.   :20.00
##  1st Qu.: 10.00   1st Qu.: 24.0   1st Qu.: 80.50   1st Qu.:27.50
##  Median : 25.00   Median : 76.0   Median : 92.00   Median :34.00
##  Mean   : 36.62   Mean   :101.1   Mean   : 95.47   Mean   :33.98
##  3rd Qu.: 41.50   3rd Qu.:132.5   3rd Qu.:104.00   3rd Qu.:38.50
##  Max.   :168.00   Max.   :423.0   Max.   :142.00   Max.   :58.00
##  median.assets    num.low.salary
##  Min.   :2.880   Min.   :126.0
##  1st Qu.:4.595   1st Qu.:165.5
##  Median :5.370   Median :176.0
##  Mean   :5.254   Mean   :194.0
##  3rd Qu.:5.915   3rd Qu.:227.5
##  Max.   :6.890   Max.   :276.0
```

First step: some plotting and summary statistics

```r
# Scatter plot of outcome (crime.per.million) against average.ed
qplot(average.ed, crime.per.million, data = crime)
```

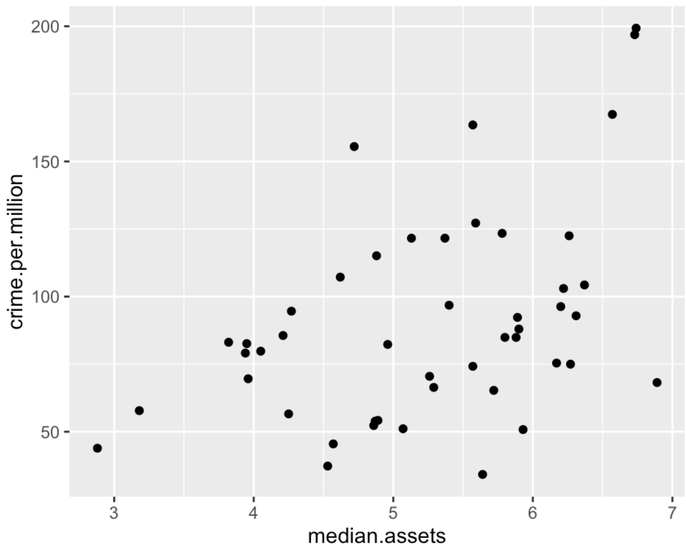



```r
# correlation between education and crime
with(crime, cor(average.ed, crime.per.million))
```

```
## [1] 0.3228349
```



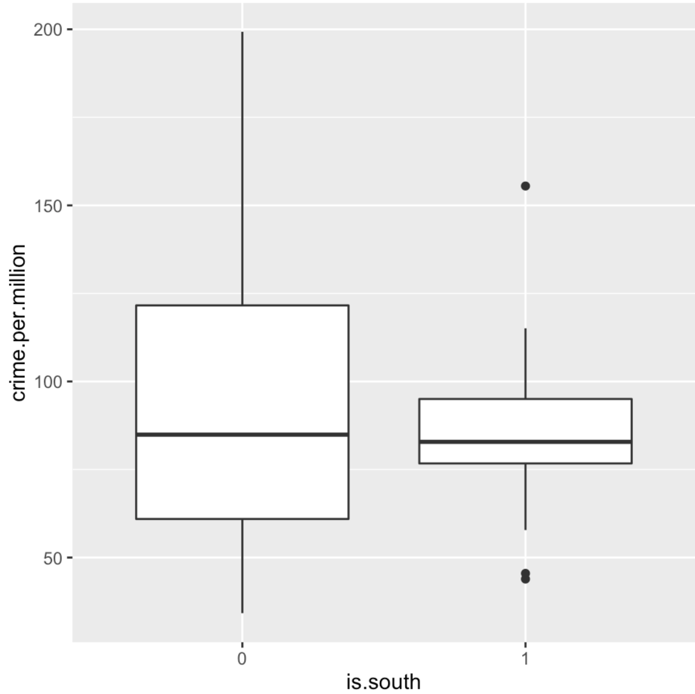
```r
# Scatter plot of outcome (crime.per.million) against median.assets
qplot(median.assets, crime.per.million, data = crime)
```




```r
# correlation between education and crime
with(crime, cor(median.assets, crime.per.million))
```

```
## [1] 0.4413199
```

```r
# Boxplots showing crime rate broken down by southern vs non-southern state
qplot(is.south, crime.per.million, geom = "boxplot", data = crime)
```



Constructing a regression model

```r
crime.lm <- lm(crime.per.million ~ ., data = crime)
# Summary of the linear regression model
crime.lm
```

```
##
## Call:
## lm(formula = crime.per.million ~ ., data = crime)
##
## Coefficients:
##      (Intercept)       young.males         is.south1        average.ed
##       -6.918e+02         1.040e+00        -8.308e+00         1.802e+01
## exp.per.cap.1960  exp.per.cap.1959       labour.part       male.per.fem
##        1.608e+00        -6.673e-01        -4.103e-02         1.648e-01
##       population          nonwhite       unemp.youth       unemp.adult
##       -4.128e-02         7.175e-03        -6.017e-01         1.792e+00
##    median.assets    num.low.salary
##        1.374e+01         7.929e-01
```

```r
summary(crime.lm)
```

```
##
## Call:
## lm(formula = crime.per.million ~ ., data = crime)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -34.884 -11.923  -1.135  13.495  50.560
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)      -6.918e+02  1.559e+02  -4.438 9.56e-05 ***
## young.males       1.040e+00  4.227e-01   2.460  0.01931 *
## is.south1        -8.308e+00  1.491e+01  -0.557  0.58117
## average.ed        1.802e+01  6.497e+00   2.773  0.00906 **
## exp.per.cap.1960  1.608e+00  1.059e+00   1.519  0.13836
## exp.per.cap.1959 -6.673e-01  1.149e+00  -0.581  0.56529
## labour.part      -4.103e-02  1.535e-01  -0.267  0.79087
## male.per.fem      1.648e-01  2.099e-01   0.785  0.43806
## population       -4.128e-02  1.295e-01  -0.319  0.75196
## nonwhite          7.175e-03  6.387e-02   0.112  0.91124
## unemp.youth      -6.017e-01  4.372e-01  -1.376  0.17798
## unemp.adult       1.792e+00  8.561e-01   2.093  0.04407 *
## median.assets     1.374e+01  1.058e+01   1.298  0.20332
## num.low.salary    7.929e-01  2.351e-01   3.373  0.00191 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 21.94 on 33 degrees of freedom
## Multiple R-squared:  0.7692, Adjusted R-squared:  0.6783
## F-statistic: 8.462 on 13 and 33 DF,  p-value: 3.686e-07
```



```r
options(scipen=4)  # Set scipen = 0 to get back to default
```

```r
summary(crime.lm)
```

```
##
## Call:
## lm(formula = crime.per.million ~ ., data = crime)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -34.884 -11.923  -1.135  13.495  50.560
##
## Coefficients:
##                    Estimate  Std. Error t value  Pr(>|t|)
## (Intercept)      -691.837588  155.887918  -4.438 0.0000956 ***
## young.males         1.039810    0.422708   2.460   0.01931 *
## is.south1          -8.308313   14.911588  -0.557   0.58117
## average.ed         18.016011    6.496504   2.773   0.00906 **
## exp.per.cap.1960    1.607818    1.058667   1.519   0.13836
## exp.per.cap.1959   -0.667258    1.148773  -0.581   0.56529
## labour.part        -0.041031    0.153477  -0.267   0.79087
## male.per.fem        0.164795    0.209932   0.785   0.43806
## population         -0.041277    0.129516  -0.319   0.75196
## nonwhite            0.007175    0.063867   0.112   0.91124
## unemp.youth        -0.601675    0.437154  -1.376   0.17798
## unemp.adult         1.792263    0.856111   2.093   0.04407 *
## median.assets      13.735847   10.583028   1.298   0.20332
## num.low.salary      0.792933    0.235085   3.373   0.00191 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 21.94 on 33 degrees of freedom
## Multiple R-squared:  0.7692, Adjusted R-squared:  0.6783
## F-statistic: 8.462 on 13 and 33 DF,  p-value: 0.0000003686
```

Exploring the lm object

What kind of output do we get when we run a linear model (`lm`) in R?

```
# List all attributes of the linear model
attributes(crime.lm)
```

```
## $names
##  [1] "coefficients"  "residuals"     "effects"       "rank"
##  [5] "fitted.values" "assign"        "qr"            "df.residual"
##  [9] "contrasts"     "xlevels"       "call"          "terms"
## [13] "model"
##
## $class
## [1] "lm"
```



```
# coefficients
crime.lm$coef
```

```
##      (Intercept)      young.males        is.south1      average.ed
##    -691.837587905      1.039809653     -8.308312889    18.016010601
## exp.per.cap.1960 exp.per.cap.1959      labour.part    male.per.fem
##       1.607818377     -0.667258285     -0.041031047     0.164794968
##        population         nonwhite     unemp.youth     unemp.adult
##      -0.041276887      0.007174688     -0.601675298     1.792262901
##     median.assets   num.low.salary
##      13.735847285      0.792932786
```

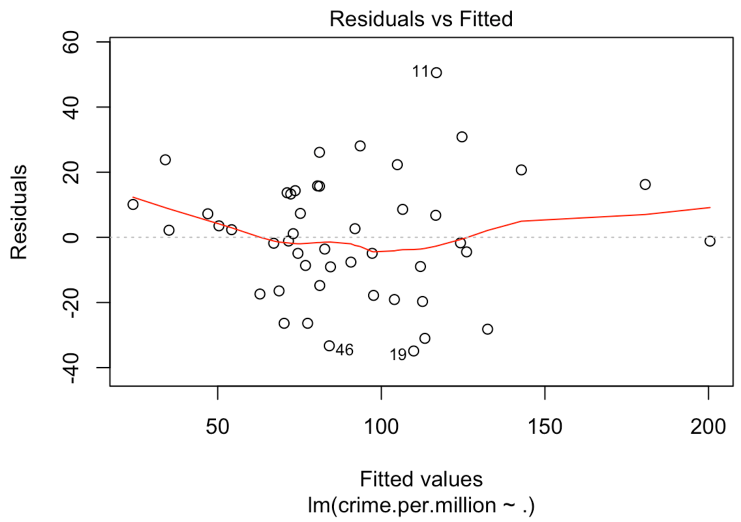
None of the attributes seem to give you p-values. Here's what you can do to get a table that allows you to extract p-values.

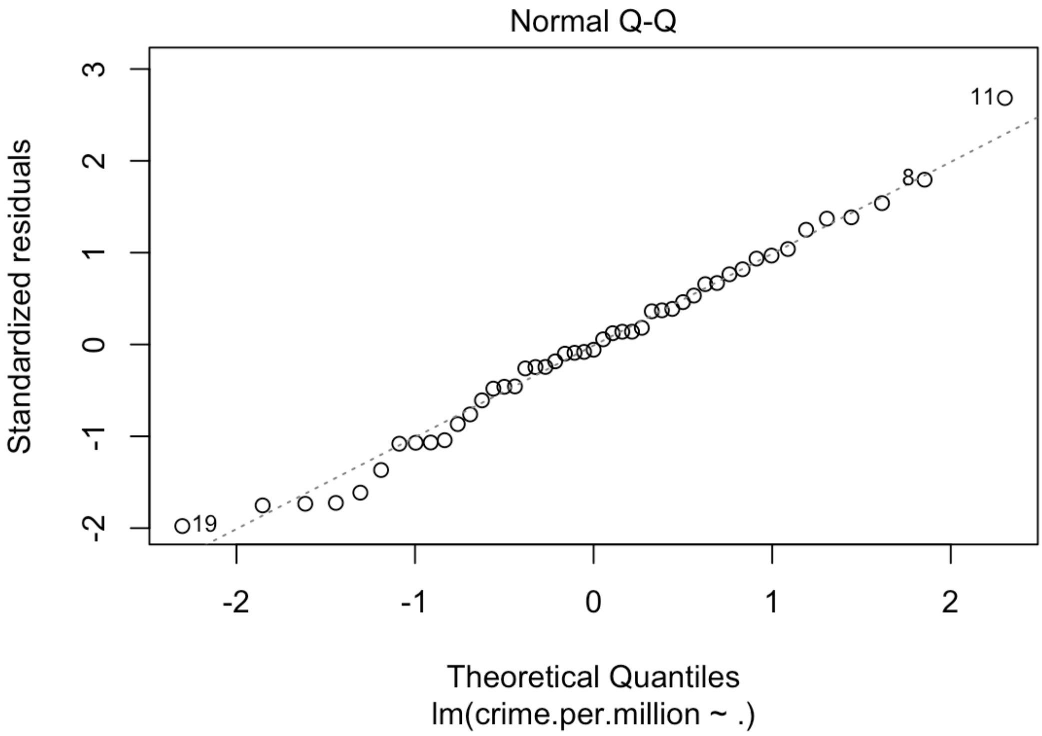
```
# Pull coefficients element from summary(lm) object
round(summary(crime.lm)$coef, 3)
```

```
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -691.838    155.888  -4.438    0.000
## young.males          1.040      0.423   2.460    0.019
## is.south1           -8.308     14.912  -0.557    0.581
## average.ed          18.016      6.497   2.773    0.009
## exp.per.cap.1960     1.608      1.059   1.519    0.138
## exp.per.cap.1959    -0.667      1.149  -0.581    0.565
## labour.part         -0.041      0.153  -0.267    0.791
## male.per.fem         0.165      0.210   0.785    0.438
## population          -0.041      0.130  -0.319    0.752
## nonwhite             0.007      0.064   0.112    0.911
## unemp.youth         -0.602      0.437  -1.376    0.178
## unemp.adult          1.792      0.856   2.093    0.044
## median.assets       13.736     10.583   1.298    0.203
## num.low.salary       0.793      0.235   3.373    0.002
```

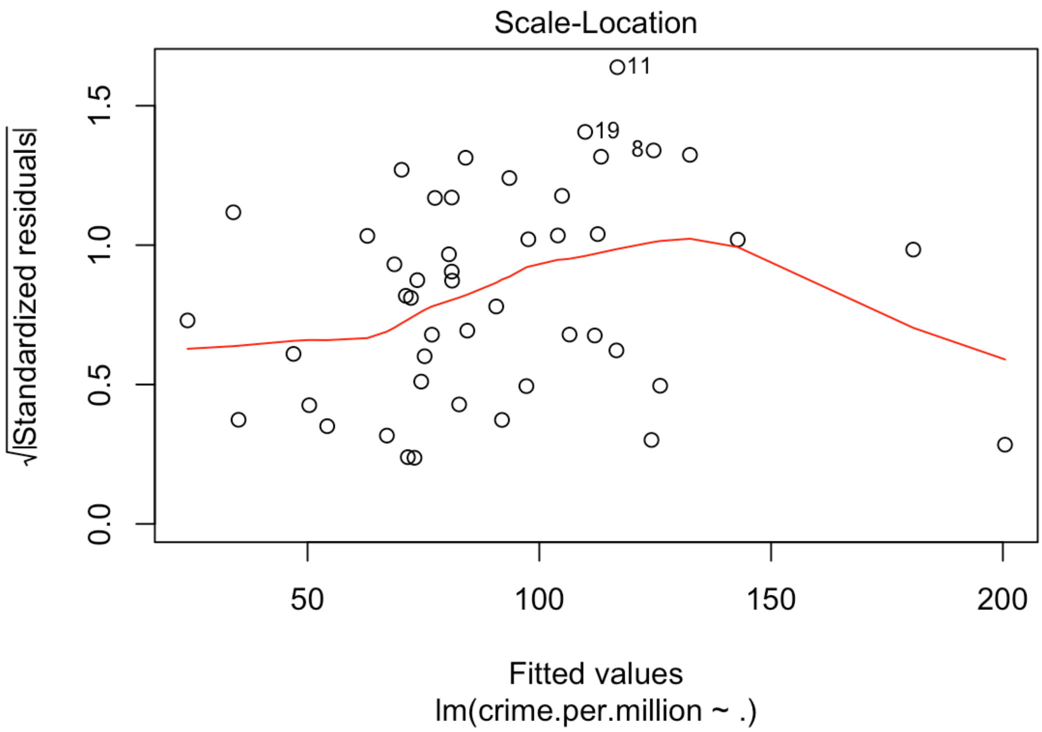

Plotting the lm object

```
plot(crime.lm)
```

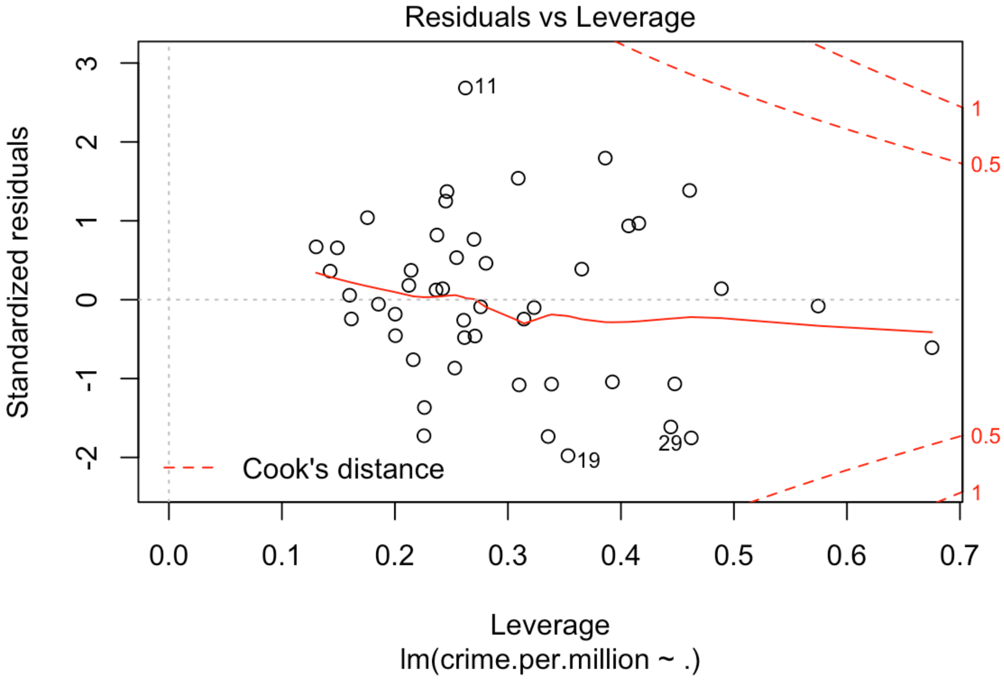




Normal Q-Q
Standardized residuals
Theoretical Quantiles
lm(crime.per.million ~ .)
-2
-1
0
1
2
3
19
8
11



Scale-Location
√|Standardized residuals|
Fitted values
lm(crime.per.million ~ .)
0.0
0.5
1.0
1.5
50
100
150
200
11
19
8



Residuals vs Fitted Plot

Residuals vs. Fitted When a linear model is appropriate, we expect

1 the residuals will have constant variance when plotted against fitted values; and

2 the residuals and fitted values will be uncorrelated.

If there are clear trends in the residual plot, or the plot looks like a funnel, these are clear indicators that the given linear model is inappropriate.

Normal QQ Plot

Normal QQ plot You can use a linear model for prediction even if the underlying normality assumptions don’t hold. However, in order for the p-values to be believable, the residuals from the regression must look approximately normally distributed.

Scale-Location Plot

Scale-location plot This is another version of the residuals vs fitted plot. There should be no discernible trends in this plot.

Residuals vs Leverage

Residuals vs Leverage. Leverage is a measure of how much an observation influenced the model fit. It's a one-number summary of how different the model fit would be if the given observation was excluded, compared to the model fit where the observation is included. Points with high residual (poorly described by the model) and high leverage (high influence on model fit) are outliers. They're skewing the model fit away from the rest of the data, and don't really seem to fit with the rest of the data.

Workshop 10: Regression

We'll begin by loading some packages.

```
library(MASS)
library(plyr)
```

Interaction terms in regression

```
# Building up the familiar birthwt data...

# Rename the columns to have more descriptive names
colnames(birthwt) <- c("birthwt.below.2500", "mother.age", "mother.weight",
    "race", "mother.smokes", "previous.prem.labor", "hypertension", "uterine.irr",
    "physician.visits", "birthwt.grams")

# Transform variables to factors with descriptive levels
birthwt <- transform(birthwt,
            race = as.factor(mapvalues(race, c(1, 2, 3),
                              c("white","black", "other"))),
            mother.smokes = as.factor(mapvalues(mother.smokes,
                              c(0,1), c("no", "yes"))),
            hypertension = as.factor(mapvalues(hypertension,
                              c(0,1), c("no", "yes"))),
            uterine.irr = as.factor(mapvalues(uterine.irr,
                              c(0,1), c("no", "yes")))
            )
```

Workshop 10 : Regression

(a) Run a linear regression to better understand how birthweight varies with the mother's age and smoking status (do not include interaction terms).

(b) What is the coefficient of mother.age in your regression? How do you interpret this coefficient?

(c) How many coefficients are estimated for the mother's smoking status variable? How do you interpret these coefficients?

Predictive ML

- Predictive Data Mining
  - Two Phases of Processing
    - Training Phase : Learn a model from training data
    - Predicting Phase : Deploy the model to production and use that to predict the future outcome

Data

Iris Data Set from UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Iris)

Attribute Information:

1. Sepal Length in cm
2. Sepal width in cm
3. Petal length in cm
4. Petal length in cm
5. Classes:
- Iris Setosa
- Iris Versicolour
- Iris Virginica

UCI Machine Learning Repository
Center for Machine Learning and Intelligent Systems

Getting Data

> iris <- read.csv(“iris.data.csv”, header=TRUE)

> library(datasets)

> iris

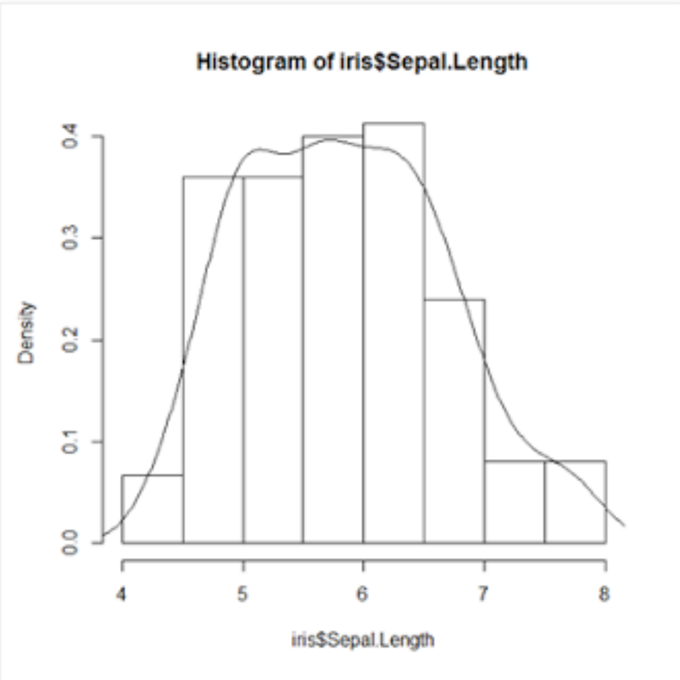
> colnames(iris) <- c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width", "Species")

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> nrow(iris)
[1] 150
> table(iris$Species)

    setosa versicolor  virginica
        50         50         50
>
```

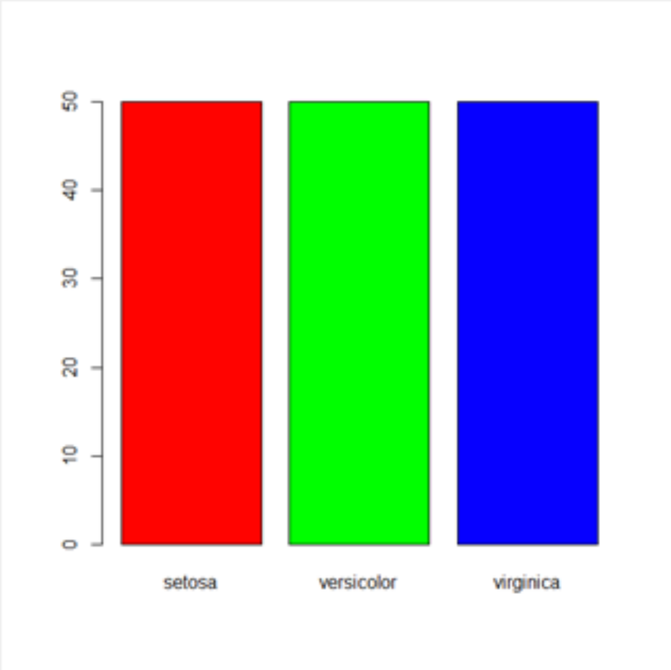
Data Visualization

- Visualizing existing data is a very useful way to come up with ideas about what features should be included.
- "Dataframe" in R is a common way where data samples are organized in a tabular structure.

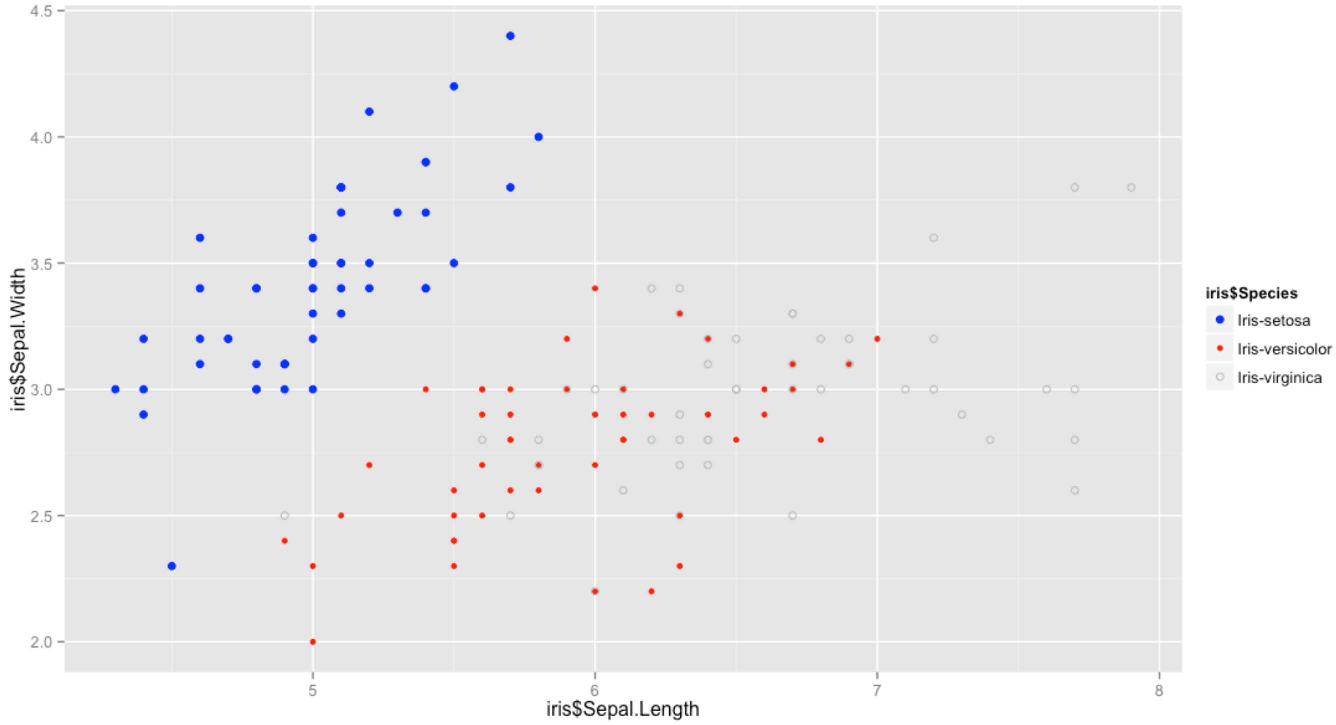

```
> # Plot the histogram
> hist(iris$Sepal.Length, breaks=10, prob=T)
> # Plot the density curve
> lines(density(iris$Sepal.Length))
>
```



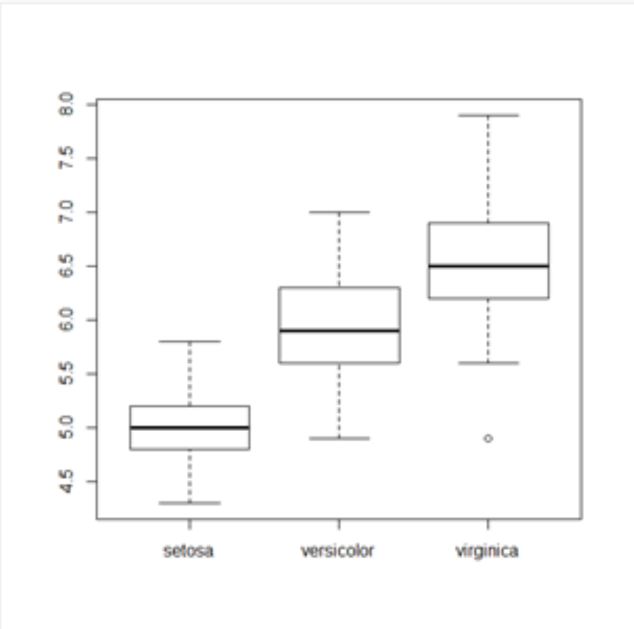
```
> categories <- table(iris$Species)
> barplot(categories, col=c('red', 'green', 'blue'))
>
```



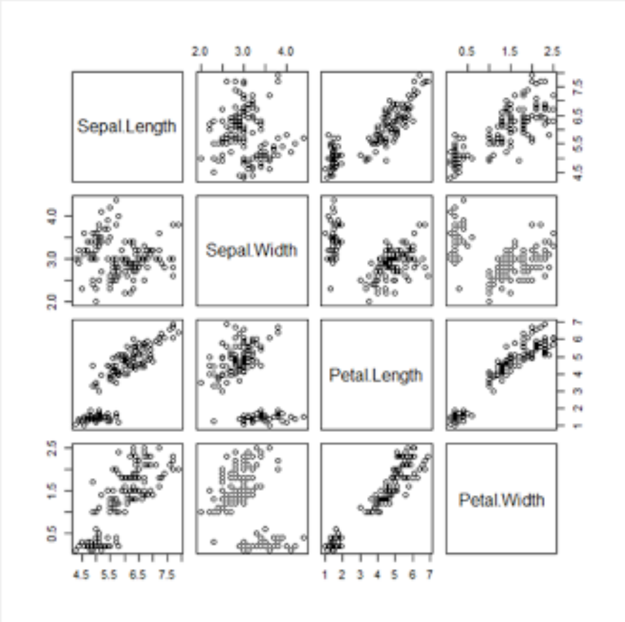

```
ggplot(iris, aes(x=iris$SepalLength, y=iris$SepalWidth,
group=iris$Species,shape=iris$Species)) +
geom_point(aes(colour=iris$Species)) +
  scale_shape_manual(values=c(19,20,21))+
  scale_colour_manual(values=c("blue", "red","gray"))
```



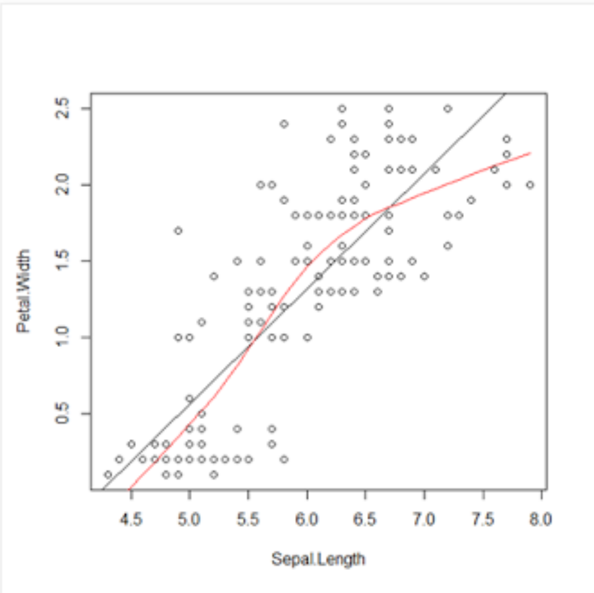

```
> boxplot(Sepal.Length~Species, data=iris)
>
```



```
> # Scatter plot for all pairs
> pairs(iris[,c(1,2,3,4)])
> # Compute the correlation matrix
> cor(iris[,c(1,2,3,4)])
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000  -0.1170695    0.8716902   0.8179410
Sepal.Width    -0.1170695   1.0000000   -0.4284401  -0.3661259
Petal.Length    0.8716902  -0.4284401    1.0000000   0.9628654
Petal.Width     0.8179410  -0.3661259    0.9628654   1.0000000
>
```




```
> # First plot the 2 variables
> plot(Petal.Width~Sepal.Length, data=iris)
> # Learn the regression model
> model <- lm(Petal.Width~Sepal.Length, data=iris)
> # Plot the regression line
> abline(model)
> # Now learn the local linear model
> model2 <- lowess(iris$Petal.Width~iris$Sepal.Length)
> lines(model2, col="red")
>
```



Preparing Training Data

At this step, the purpose is to transform the raw data in a form that can fit into the data mining model.

- Data sampling
- Data validation and handle missing data
- Normalize numeric value into a uniform range
- Compute aggregated value (a special case is to compute frequency counts)
- Expand categorical field to binary fields
- Discretize numeric value into categories
- Create derived fields from existing fields
- Reduce dimensionality
- Power and Log transformation

Data Sampling

```
> # select 10 records out from iris with replacement
> index <- sample(1:nrow(iris), 10, replace=T)
> index
 [1] 133  36 107 140  66  67  36   3  97  37
> irissample <- iris[index,]
> irissample
     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
133           6.4         2.8          5.6         2.2  virginica
36            5.0         3.2          1.2         0.2     setosa
107           4.9         2.5          4.5         1.7  virginica
140           6.9         3.1          5.4         2.1  virginica
66            6.7         3.1          4.4         1.4 versicolor
67            5.6         3.0          4.5         1.5 versicolor
36.1          5.0         3.2          1.2         0.2     setosa
3             4.7         3.2          1.3         0.2     setosa
97            5.7         2.9          4.2         1.3 versicolor
37            5.5         3.5          1.3         0.2     setosa
>
```

Impute missing data

- Discard the whole record
- Infer missing value based on the data of other record. Approach is to fill the missing data with the average or the median.

```
> # Create some missing data
> irissample[10, 1] <- NA
> irissample
      Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
133            6.4         2.8          5.6         2.2  virginica
36             5.0         3.2          1.2         0.2     setosa
107            4.9         2.5          4.5         1.7  virginica
140            6.9         3.1          5.4         2.1  virginica
66             6.7         3.1          4.4         1.4 versicolor
67             5.6         3.0          4.5         1.5 versicolor
36.1           5.0         3.2          1.2         0.2     setosa
3              4.7         3.2          1.3         0.2     setosa
97             5.7         2.9          4.2         1.3 versicolor
37              NA         3.5          1.3         0.2     setosa
```



```
> library(e1071)
Loading required package: class
Warning message:
package 'e1071' was built under R version 2.14.2
> fixIris1 <- impute(irissample[,1:4], what='mean')
> fixIris1
     Sepal.Length Sepal.Width Petal.Length Petal.Width
133      6.400000         2.8          5.6         2.2
36       5.000000         3.2          1.2         0.2
107      4.900000         2.5          4.5         1.7
140      6.900000         3.1          5.4         2.1
66       6.700000         3.1          4.4         1.4
67       5.600000         3.0          4.5         1.5
36.1     5.000000         3.2          1.2         0.2
3        4.700000         3.2          1.3         0.2
97       5.700000         2.9          4.2         1.3
37       5.655556         3.5          1.3         0.2
> fixIris2 <- impute(irissample[,1:4], what='median')
> fixIris2
     Sepal.Length Sepal.Width Petal.Length Petal.Width
133           6.4         2.8          5.6         2.2
36            5.0         3.2          1.2         0.2
107           4.9         2.5          4.5         1.7
140           6.9         3.1          5.4         2.1
66            6.7         3.1          4.4         1.4
67            5.6         3.0          4.5         1.5
36.1          5.0         3.2          1.2         0.2
3             4.7         3.2          1.3         0.2
97            5.7         2.9          4.2         1.3
37            5.6         3.5          1.3         0.2
>
```

Normalize numeric value

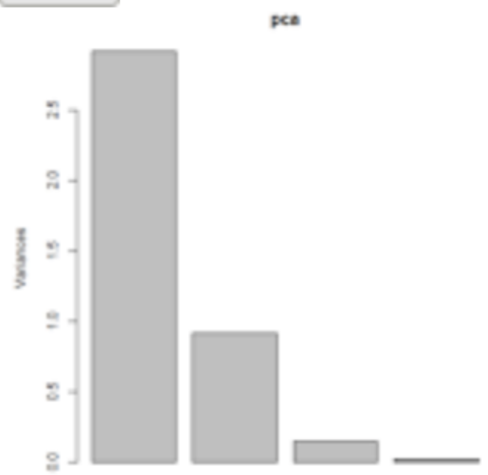
```
> # scale the columns
> # x-mean(x)/standard deviation
> scaleiris <- scale(iris[, 1:4])
> head(scaleiris)
     Sepal.Length Sepal.Width Petal.Length Petal.Width
[1,]   -0.8976739  1.01560199    -1.335752   -1.311052
[2,]   -1.1392005 -0.13153881    -1.335752   -1.311052
[3,]   -1.3807271  0.32731751    -1.392399   -1.311052
[4,]   -1.5014904  0.09788935    -1.279104   -1.311052
[5,]   -1.0184372  1.24503015    -1.335752   -1.311052
[6,]   -0.5353840  1.93331463    -1.165809   -1.048667
>
```

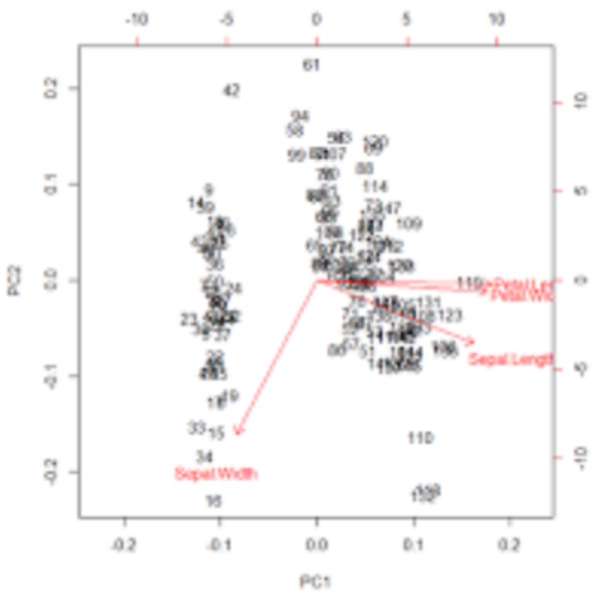

Reduce dimensionality

There are two ways to reduce the number of input attributes.

1. Removing irrelevant input variables.
2. Removing redundant input variables.

```
> # Use iris data set
> cor(iris[, -5])
             Sepal.Length   Sepal.Width  Petal.Length   Petal.Width
Sepal.Length  1.0000000000 -0.1175697841  0.8717537759  0.8179411263
Sepal.Width  -0.1175697841  1.0000000000 -0.4284401043 -0.3661259325
Petal.Length  0.8717537759 -0.4284401043  1.0000000000  0.9628654314
Petal.Width   0.8179411263 -0.3661259325  0.9628654314  1.0000000000
> # Some attributes shows high correlation, compute PCA
> pca <- prcomp(iris[,-5], scale=T)
> summary(pca)
Importance of components:
                            PC1       PC2       PC3       PC4
Standard deviation     1.708361 0.9560494 0.3830886 0.1439265
Proportion of Variance 0.729620 0.2285100 0.0366900 0.0051800
Cumulative Proportion  0.729620 0.9581300 0.9948200 1.0000000
> # Notice PC1 and PC2 covers most variation
> plot(pca)
> pca$rotation
                       PC1            PC2            PC3            PC4
Sepal.Length  0.5210659147 -0.37741761556  0.7195663527  0.2612862800
Sepal.Width  -0.2693474425 -0.92329565954 -0.2443817795 -0.1235096196
Petal.Length  0.5804130958 -0.02449160909 -0.1421263693 -0.8014492463
Petal.Width   0.5648565358 -0.06694198697 -0.6342727371  0.5235971346
> # Project first 2 records in PCA direction
> predict(pca)[1:2,]
              PC1           PC2          PC3           PC4
[1,] -2.257141176 -0.4784238321 0.1272796237 0.02408750846
[2,] -2.074013015  0.6718826870 0.2338255167 0.10266284468
> # plot all points in top 2 PCA direction
> biplot(pca)
```



Add derived attributes

```
> iris2 <- transform(iris, ratio=round(Sepal.Length/Sepal.Width, 2))
> head(iris2)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species ratio
1          5.1         3.5          1.4         0.2  setosa  1.46
2          4.9         3.0          1.4         0.2  setosa  1.63
3          4.7         3.2          1.3         0.2  setosa  1.47
4          4.6         3.1          1.5         0.2  setosa  1.48
5          5.0         3.6          1.4         0.2  setosa  1.39
6          5.4         3.9          1.7         0.4  setosa  1.38
```


Discretize numeric value into categories

```
> # Equal width cuts
> segments <- 10
> maxL <- max(iris$Petal.Length)
> minL <- min(iris$Petal.Length)
> theBreaks <- seq(minL, maxL,
                   by=(maxL-minL)/segments)
> cutPetalLength <- cut(iris$Petal.Length,
                        breaks=theBreaks,
                        include.lowest=T)
> newdata <- data.frame(orig.Petal.Len=iris$Petal.Length,
                        cut.Petal.Len=cutPetalLength)
> head(newdata)
  orig.Petal.Len cut.Petal.Len
1            1.4      [1,1.59]
2            1.4      [1,1.59]
3            1.3      [1,1.59]
4            1.5      [1,1.59]
5            1.4      [1,1.59]
6            1.7   (1.59,2.18]
>
> # Constant frequency / height
> myBreaks <- quantile(iris$Petal.Length,
                       probs=seq(0,1,1/segments))
> cutPetalLength2 <- cut(iris$Petal.Length,
                         breaks=myBreaks,
                         include.lowest=T)
> newdata2 <- data.frame(orig.Petal.Len=iris$Petal.Length,
                         cut.Petal.Len=cutPetalLength2)
> head(newdata2)
  orig.Petal.Len cut.Petal.Len
1            1.4       [1,1.4]
2            1.4       [1,1.4]
3            1.3       [1,1.4]
4            1.5     (1.4,1.5]
5            1.4       [1,1.4]
6            1.7     (1.7,3.9]
>
```

Binarize categorical attributes

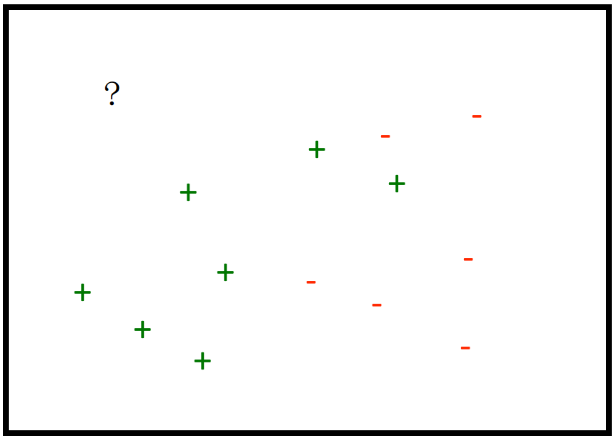
```
> cat <- levels(iris$Species)
> cat
[1] "setosa"     "versicolor" "virginica"
> binarize <- function(x) {return(iris$Species == x)}
> newcols <- sapply(cat, binarize)
> colnames(newcols) <- cat
> data <- cbind(iris[,c('Species')], newcols)
> data[45:55,]
        setosa versicolor virginica
 [1,] 1      1          0         0
 [2,] 1      1          0         0
 [3,] 1      1          0         0
 [4,] 1      1          0         0
 [5,] 1      1          0         0
 [6,] 1      1          0         0
 [7,] 2      0          1         0
 [8,] 2      0          1         0
 [9,] 2      0          1         0
[10,] 2      0          1         0
[11,] 2      0          1         0
```


Iris Data Preparation

```
> set.seed(1234)
> ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))
> trainData <- iris[ind==1,]
> testData <- iris[ind==2,]
```

Decision Tree

Feature #2
(e.g.., roundness)



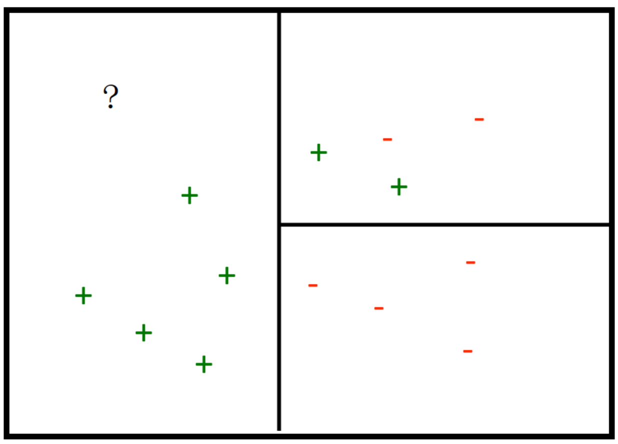
Feature #1 (e.g.., 'area')

Decision Tree

We build tree to divide them up.

Feature #2
(e.g.., roundness)

40

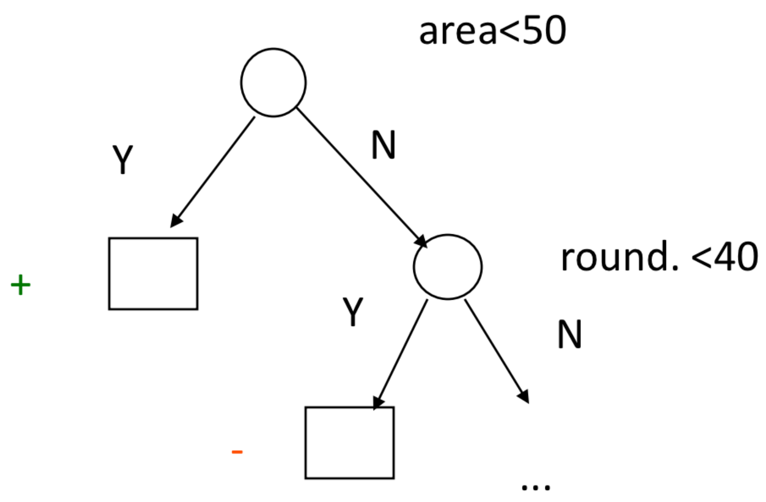


50

Feature #1 (e.g.., 'area')

Decision Tree

Goal: split address space in (almost) homogeneous regions



Decision Tree

Conditional Inference Trees

Description

Recursive partitioning for continuous, censored, ordered, nominal and multivariate response variables in a conditional inference framework.

Usage

```
ctree(formula, data, subset = NULL, weights = NULL,
      controls = ctree_control(), xtrafo = ptrafo, ytrafo = ptrafo,
      scores = NULL)
```

Arguments

`formula`	a symbolic description of the model to be fit. Note that symbols like `:` and `-` will not work and the tree will make use of all variables listed on the rhs of `formula`.
`data`	a data frame containing the variables in the model.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`weights`	an optional vector of weights to be used in the fitting process. Only non-negative integer valued weights are allowed.
`controls`	an object of class `TreeControl`, which can be obtained using `ctree_control`.
`xtrafo`	a function to be applied to all input variables. By default, the `ptrafo` function is applied.
`ytrafo`	a function to be applied to all response variables. By default, the `ptrafo` function is applied.
`scores`	an optional named list of scores to be attached to ordered factors.

Decision Tree - Create Model

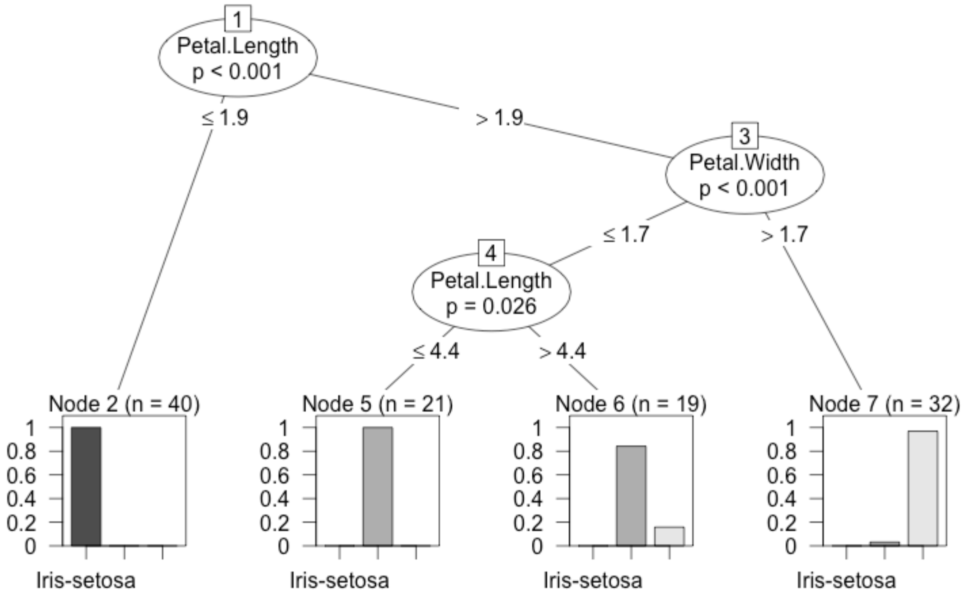
```
> library(party)
> myFormula <- Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
> iris_ctree <- ctree(myFormula, data=trainData)
> # check the prediction
> table(predict(iris_ctree), trainData$Species)
```

```
           setosa versicolor virginica
setosa         40          0         0
versicolor      0         37         3
virginica       0          1        31
```

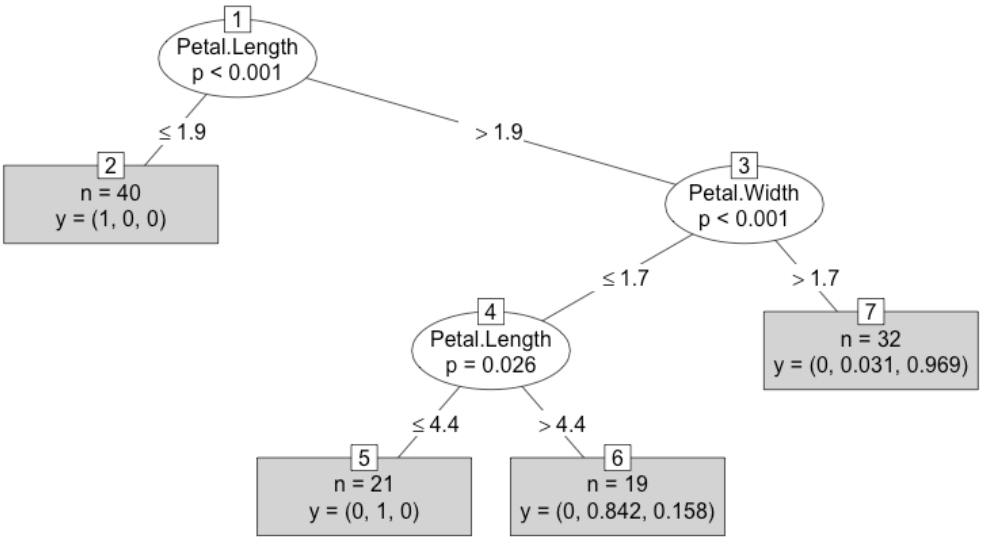


```
> print(iris_ctree)

         Conditional inference tree with 4 terminal nodes

Response:  Species
Inputs:  Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
Number of observations:  112

1) Petal.Length <= 1.9; criterion = 1, statistic = 104.643
  2)*  weights = 40
1) Petal.Length > 1.9
  3) Petal.Width <= 1.7; criterion = 1, statistic = 48.939
    4) Petal.Length <= 4.4; criterion = 0.974, statistic = 7.397
      5)*  weights = 21
    4) Petal.Length > 4.4
      6)*  weights = 19
  3) Petal.Width > 1.7
    7)*  weights = 32
```

```
> plot(iris_ctree)
```




```
> plot(iris_ctree, type="simple")
```

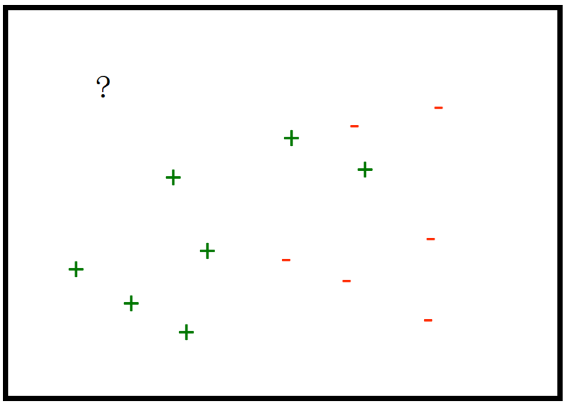


Decision Tree - Prediction

```
> # predict on test data
> testPred <- predict(iris_ctree, newdata = testData)
> table(testPred, testData$Species)

testPred     setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         12         2
  virginica       0          0        14
```


K-Nearest Neighbor

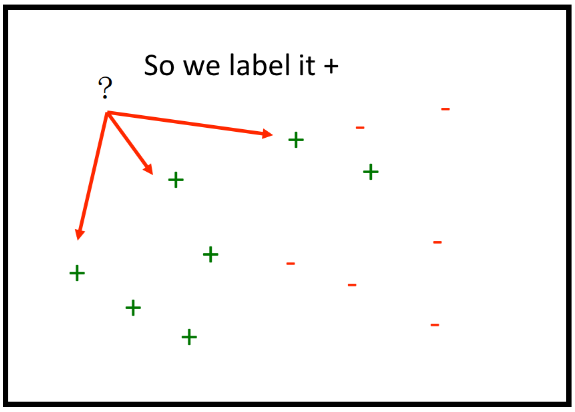
Feature #2
(e.g.., roundness)



Feature #1 (e.g.., 'area')

K-NN

Feature #2
(e.g.., roundness)



for k=3,

nearest
neighbors
are

Feature #1 (e.g.., 'area')

K-NN

k-Nearest Neighbour Classification

Description

k-nearest neighbour classification for test set from training set. For each row of the test set, the `k` nearest (in Euclidean distance) training set vectors are found, and the classification is decided by majority vote, with ties broken at random. If there are ties for the `k`th nearest vector, all candidates are included in the vote.

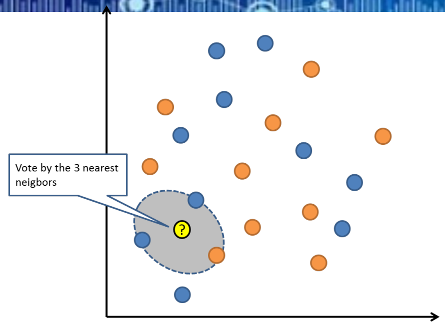
Usage

```
knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
```

Arguments

`train`	matrix or data frame of training set cases.
`test`	matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.
`cl`	factor of true classifications of training set
`k`	number of neighbours considered.
`l`	minimum vote for definite decision, otherwise `doubt`. (More precisely, less than `k-l` dissenting votes are allowed, even if `k` is increased by ties.)
`prob`	If this is true, the proportion of the votes for the winning class are returned as attribute `prob`.
`use.all`	controls handling of ties. If true, all distances equal to the `k`th largest are included. If false, a random selection of distances equal to the `k`th is chosen to use exactly `k` neighbours.

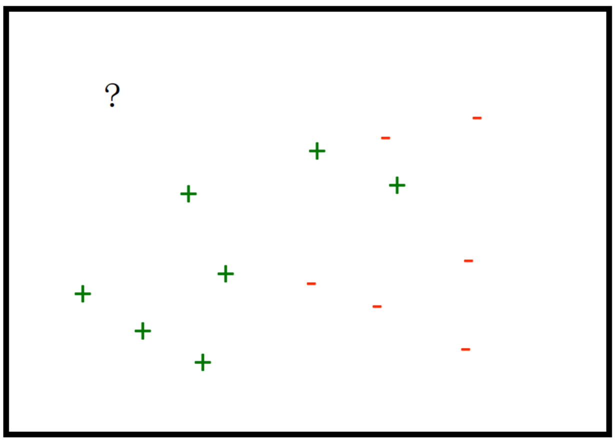
K-NN




```
> library(class)
> train_input <- as.matrix(iristrain[,-5])
> train_output <- as.vector(iristrain[,5])
> test_input <- as.matrix(iristest[,-5])
> prediction <- knn(train_input, test_input,
                    train_output, k=5)
> table(prediction, iristest$Species)

prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         1
  virginica       0          0         9
>
```


Support Vector Machines (SVMs)

Feature #2
(e.g.., roundness)

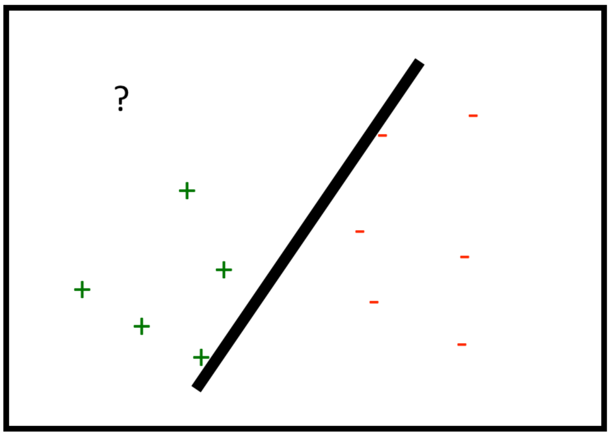


Feature #1 (e.g.., ‘area’)

Support Vector Machines (SVMs)

So, we use linear line to divide them.

round.



area

Support Vector Machine

svm {e1071} R Documentation

Support Vector Machines

Description

`svm` is used to train a support vector machine. It can be used to carry out general regression and classification (of nu and epsilon-type), as well as density-estimation. A formula interface is provided.

Usage

```
## S3 method for class 'formula'
svm(formula, data = NULL, ..., subset, na.action =
na.omit, scale = TRUE)
## Default S3 method:
svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
"radial", degree = 3, gamma = if (is.vector(x)) 1 else 1 / ncol(x),
coef0 = 0, cost = 1, nu = 0.5,
class.weights = NULL, cachesize = 40, tolerance = 0.001, epsilon = 0.1,
shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRUE,
..., subset, na.action = na.omit)
```

Arguments

`formula`
a symbolic description of the model to be fit.

`data`
an optional data frame containing the variables in the model. By default the variables are taken from the environment which 'svm' is called from.

`x`
a data matrix, a vector, or a sparse matrix (object of class `Matrix` provided by the **Matrix** package, or of class `matrix.csr` provided by the **SparseM** package, or of class `simple_triplet_matrix` provided by the **slam** package).

`y`
a response vector with one label for each row/component of `x`. Can be either a factor (for classification tasks) or a numeric vector (for regression).

Support Vector Machine

```
> library(e1071)
> tune <- tune.svm(Species~., data=iristrain, gamma=10^(-6:-1), cost=10^(1:4))
> summary(tune)
Parameter tuning of ‘svm’:
- sampling method: 10-fold cross validation
- best parameters:
 gamma  cost
 0.001 10000
- best performance: 0.03333333
> model <- svm(Species~., data=iristrain, method=”C-classification”,
kernel=”radial”, probability=T, gamma=0.001, cost=10000)
> prediction <- predict(model, iristest, probability=T)
> table(iristest$Species, prediction)
            prediction
             setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         0
  virginica       0          3         7
>
```


Naive Bayes

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Naive Bayes

Naive Bayes Classifier

Description

Computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule.

Usage

```
## S3 method for class 'formula'
naiveBayes(formula, data, laplace = 0, ..., subset, na.action = na.pass)
## Default S3 method:
naiveBayes(x, y, laplace = 0, ...)


## S3 method for class 'naiveBayes'
predict(object, newdata,
  type = c("class", "raw"), threshold = 0.001, eps = 0, ...)
```


Naive Bayes

Arguments

`x`	A numeric matrix, or a data frame of categorical and/or numeric variables.
`y`	Class vector.
`formula`	A formula of the form `class ~ x1 + x2 + ...`. Interactions are not allowed.
`data`	Either a data frame of predictors (categorical and/or numeric) or a contingency table.
`laplace`	positive double controlling Laplace smoothing. The default (0) disables Laplace smoothing.
`...`	Currently not used.
`subset`	For data given in a data frame, an index vector specifying the cases to be used in the training sample. (NOTE: If given, this argument must be named.)
`na.action`	A function to specify the action to be taken if `NA`s are found. The default action is not to count them for the computation of the probability factors. An alternative is na.omit, which leads to rejection of cases with missing values on any required variable. (NOTE: If given, this argument must be named.)
`object`	An object of class `"naiveBayes"`.
`newdata`	A dataframe with new predictors (with possibly fewer columns than the training data). Note that the column names of `newdata` are matched against the training data ones.
`type`	If `"raw"`, the conditional a-posterior probabilities for each class are returned, and the class with maximal probability else.
`threshold`	Value replacing cells with probabilities within `eps` range.
`eps`	double for specifying an epsilon-range to apply laplace smoothing (to replace zero or close-zero probabilities by `theshold`.)

Naive Bayes

```
> library(e1071)
> # Can handle both categorical and numeric input,
> # but output must be categorical
> model <- naiveBayes(Species~., data=iristrain)
> prediction <- predict(model, iristest[,-5])
> table(prediction, iristest[,5])

prediction   setosa versicolor virginica
  setosa         10          0         0
  versicolor      0         10         2
  virginica       0          0         8
```


Neural Network

Training of neural networks

Description

`neuralnet` is used to train neural networks using backpropagation, resilient backpropagation (RPROP) with (Riedmiller, 1994) or without weight backtracking (Riedmiller and Braun, 1993) or the modified globally convergent version (GRPROP) by Anastasiadis et al. (2005). The function allows flexible settings through custom-choice of error and activation function. Furthermore the calculation of generalized weights (Intrator O. and Intrator N., 1993) is implemented.

Usage

```
neuralnet(formula, data, hidden = 1, threshold = 0.01,
          stepmax = 1e+05, rep = 1, startweights = NULL,
          learningrate.limit = NULL,
          learningrate.factor = list(minus = 0.5, plus = 1.2),
          learningrate=NULL, lifesign = "none",
          lifesign.step = 1000, algorithm = "rprop+",
          err.fct = "sse", act.fct = "logistic",
          linear.output = TRUE, exclude = NULL,
          constant.weights = NULL, likelihood = FALSE)
```

Neural Network

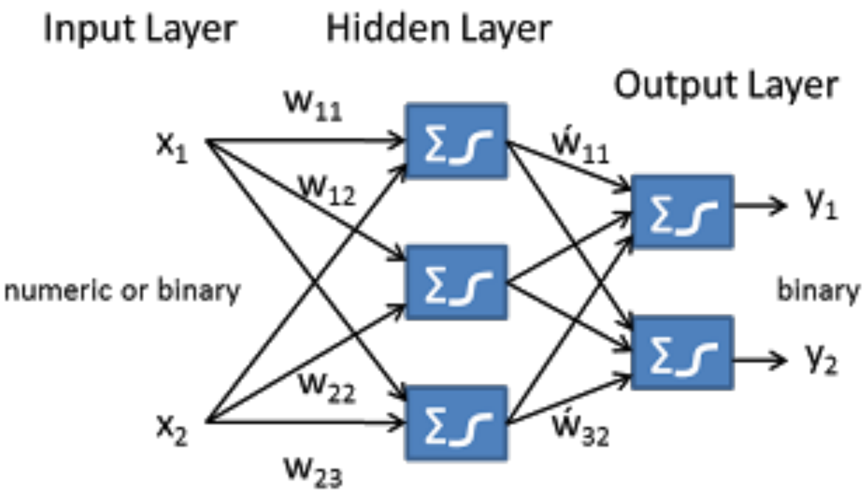
Arguments

`formula`	a symbolic description of the model to be fitted.
`data`	a data frame containing the variables specified in `formula`.
`hidden`	a vector of integers specifying the number of hidden neurons (vertices) in each layer.
`threshold`	a numeric value specifying the threshold for the partial derivatives of the error function as stopping criteria.
`stepmax`	the maximum steps for the training of the neural network. Reaching this maximum leads to a stop of the neural network's training process.
`rep`	the number of repetitions for the neural network's training.
`startweights`	a vector containing starting values for the weights. The weights will not be randomly initialized.
`learningrate.limit`	a vector or a list containing the lowest and highest limit for the learning rate. Used only for RPROP and GRPROP.
`learningrate.factor`	a vector or a list containing the multiplication factors for the upper and lower learning rate. Used only for RPROP and GRPROP.
`learningrate`	a numeric value specifying the learning rate used by traditional backpropagation. Used only for traditional backpropagation.
`lifesign`	a string specifying how much the function will print during the calculation of the neural network. 'none', 'minimal' or 'full'.

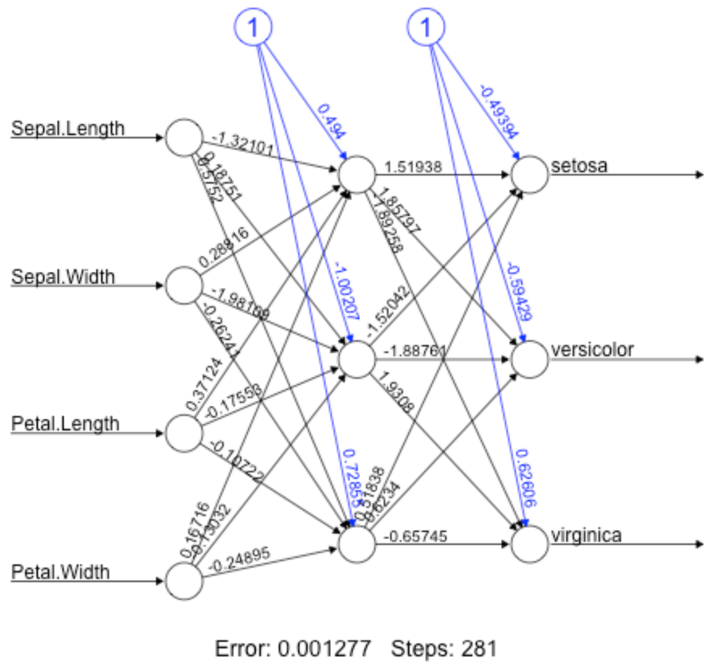
Neural Network

`lifesign.step`	an integer specifying the stepsize to print the minimal threshold in full lifesign mode.
`algorithm`	a string containing the algorithm type to calculate the neural network. The following types are possible: 'backprop', 'rprop+', 'rprop-', 'sag', or 'slr'. 'backprop' refers to backpropagation, 'rprop+' and 'rprop-' refer to the resilient backpropagation with and without weight backtracking, while 'sag' and 'slr' induce the usage of the modified globally convergent algorithm (grprop). See Details for more information.
`err.fct`	a differentiable function that is used for the calculation of the error. Alternatively, the strings 'sse' and 'ce' which stand for the sum of squared errors and the cross-entropy can be used.
`act.fct`	a differentiable function that is used for smoothing the result of the cross product of the covariate or neurons and the weights. Additionally the strings, 'logistic' and 'tanh' are possible for the logistic function and tangent hyperbolicus.
`linear.output`	logical. If act.fct should not be applied to the output neurons set linear output to TRUE, otherwise to FALSE.
`exclude`	a vector or a matrix specifying the weights, that are excluded from the calculation. If given as a vector, the exact positions of the weights must be known. A matrix with n-rows and 3 columns will exclude n weights, where the first column stands for the layer, the second column for the input neuron and the third column for the output neuron of the weight.
`constant.weights`	a vector specifying the values of the weights that are excluded from the training process and treated as fix.
`likelihood`	logical. If the error function is equal to the negative log-likelihood function, the information criteria AIC and BIC will be calculated. Furthermore the usage of confidence.interval is meaningfull.

Neural Network




```r
library(neuralnet)
iris <- read.csv("iris.data.csv", header=TRUE)
# Prepare iris
set.seed(567)
ind <- sample(2, nrow(iris), replace=TRUE, prob=c(0.7, 0.3))
trainData <- iris[ind==1,]
testData <- iris[ind==2,]
nnet_iristrain <- trainData

#Binarize the categorical output
nnet_iristrain <- cbind(nnet_iristrain, trainData$Species == 'Iris-setosa')
nnet_iristrain <- cbind(nnet_iristrain, trainData$Species == 'Iris-versicolor')
nnet_iristrain <- cbind(nnet_iristrain, trainData$Species == 'Iris-virginica')
names(nnet_iristrain)[6] <- 'Setosa'
names(nnet_iristrain)[7] <- 'Versicolor'
names(nnet_iristrain)[8] <- 'Virginica'

nn <- neuralnet(Setosa+Versicolor+Virginica ~ Sepal.Length+Sepal.Width+Petal.Length+Petal.Width,
                data=nnet_iristrain, hidden=c(3))

plot(nn)
mypredict <- compute(nn, testData[-5])$net.result
# Put multiple binary output to categorical output
maxidx <- function(arr) {
  return(which(arr == max(arr)))
}
idx <- apply(mypredict, c(1), maxidx)
prediction <- c('Iris-setosa', 'Iris-versicolor', 'Iris-virginica')[idx]
table(prediction, testData$Species)
```




```
> table(prediction, testData$Species)

prediction        Iris-setosa Iris-versicolor Iris-virginica
  Iris-setosa              15               0              0
  Iris-versicolor           0              13              0
  Iris-virginica            0               0             15
>
```


Performance and visualizing classifier performance

Classification

Binary classification

(Instances, Class labels): $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$

y_i {1,-1} - valued

Classifier: provides class prediction $\hat{Y}$ for an instance

Outcomes for a prediction:

True class

Predicted class

	1	-1
1	**True positive (TP)**	**False positive (FP)**
-1	**False negative (FP)**	**True negative (TN)**

Some basic performance measures

$P(\hat{Y} = Y)$: accuracy

$P(\hat{Y} = 1 \mid Y = 1)$: true positive rate

$P(\hat{Y} = 1 \mid Y = -1)$: false positive rate

$P(Y = 1 \mid \hat{Y} = 1)$: precision

True class

Predicted class

	1	-1
1	**True positive (TP)**	**False positive (FP)**
-1	**False negative (FP)**	**True negative (TN)**

Performance

Precision - Percentage of positive labels that are correct

Precision = (# true positives) / (# true positives + # false positives)

Recall - Percentage of positive examples that are correctly labeled

Recall = (# true positives) / (# true positives + # false negatives)

Accuracy - Percentage of correct labels

Accuracy = (# true positives + # true negatives) / (# of samples)

Performance trade-offs

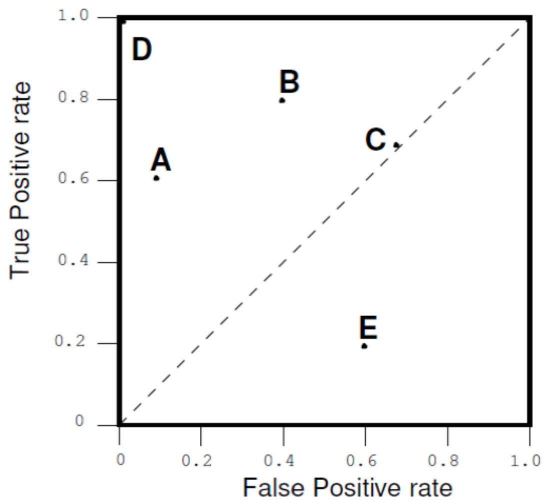
Often: Improvement in measure X →

measure Y becomes worse

Idea: Visualize trade-off in a two-dimensional plot

Examples:

- True pos. rate vs.
  false pos. rate
- Precision vs. recall
- Lift charts
- …



Scoring classifiers

Output: continuous

(instead of actual

class prediction)

Discretized by choosing

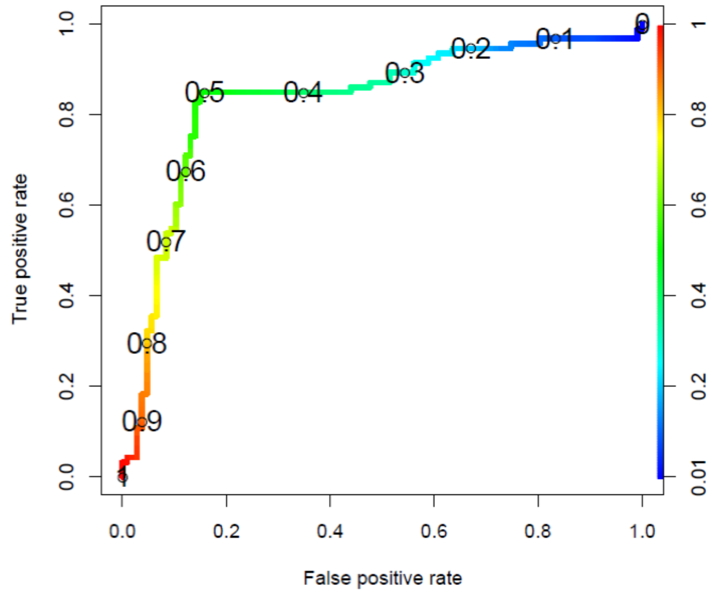
a cut-off

f(x) ≥ c ➔ class „1"

f(x) < c ➔ class „-1"

Trade-off visualizations:

cutoff-parameterized curves



ROCR

Only three commands

```
pred <- prediction( scores, labels )
```

(pred: S4 object of class prediction)

```
perf <- performance( pred, measure.Y, measure.X)
```

(pred: S4 object of class performance)

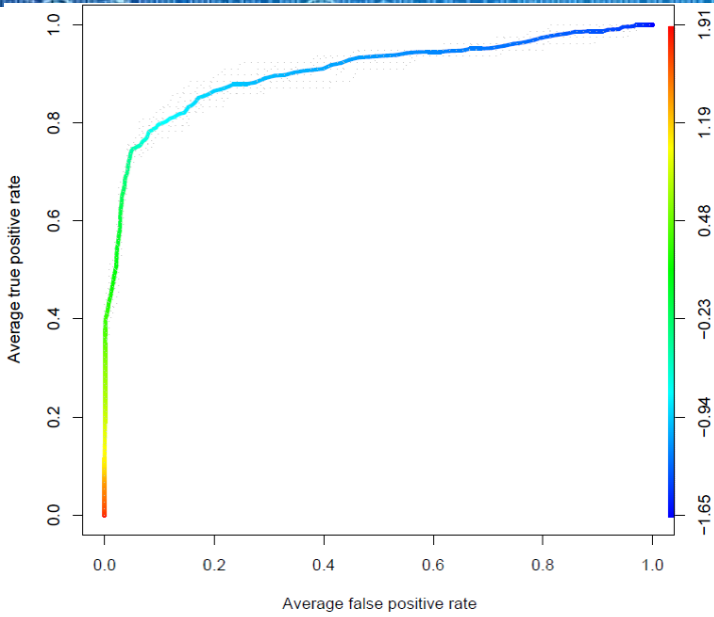
```
plot( perf )
```

Input format

Single run:
vectors (`scores`: numeric; `labels`: anything)

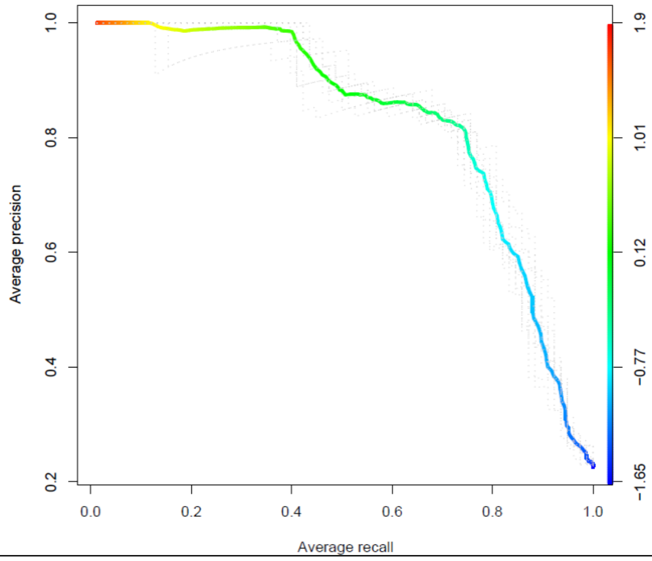
Multiple runs (cross-validation, bootstrapping, …):

Examples (1/8): ROC curves




```
pred <- prediction(scores, labels)
perf <- performance(pred, "tpr", "fpr")
plot(perf, colorize=T)
```

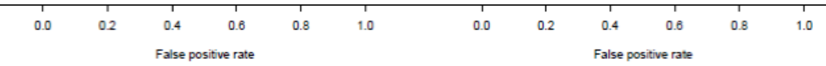

Examples (2/8): Precision/recall curves

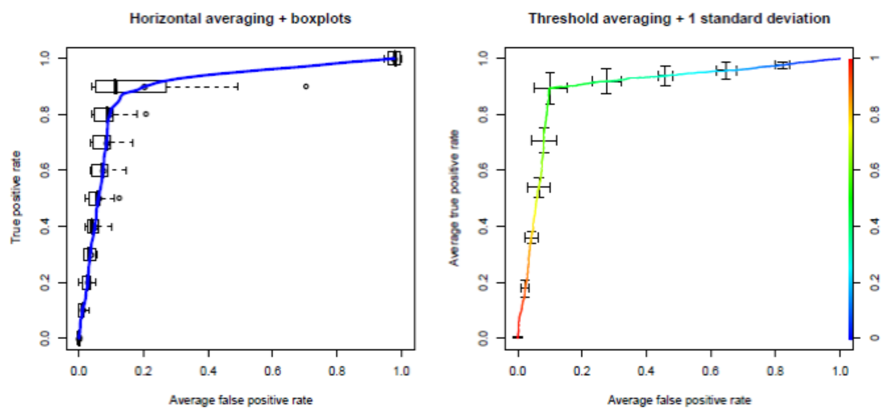



```
pred <- prediction(scores, labels)
perf <- performance(pred, "prec", "rec")
plot(perf, colorize=T)
```

Examples (3/8): Averaging across


```
pred <- prediction(scores, labels)

perf <- performance(pred, "tpr", "fpr")

plot(perf, avg='threshold',
spread.estimate='stddev', colorize=T)
```





Clustering Analysis

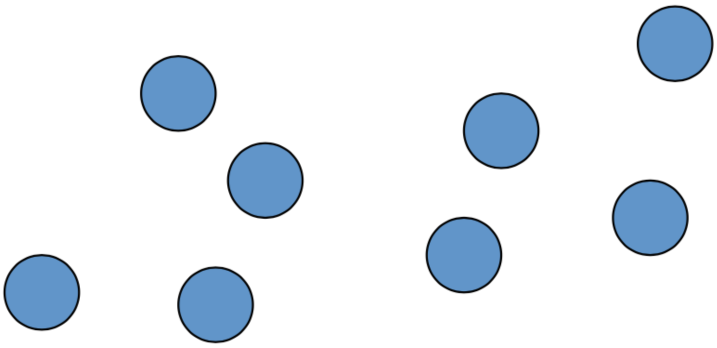
- Supervised learning (Classification) assumes classes are known
- Unsupervised learning (Cluster analysis) seeks to discover the classes

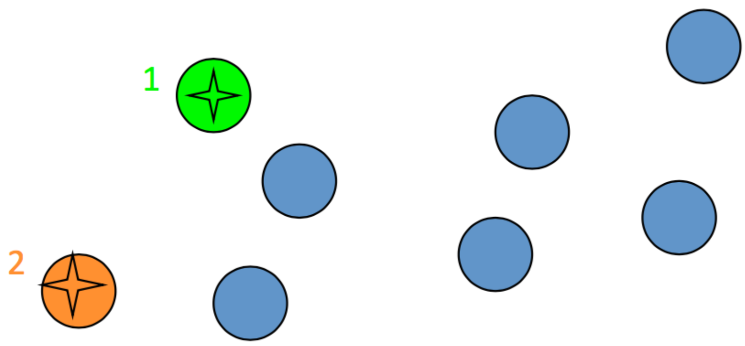
Clustering

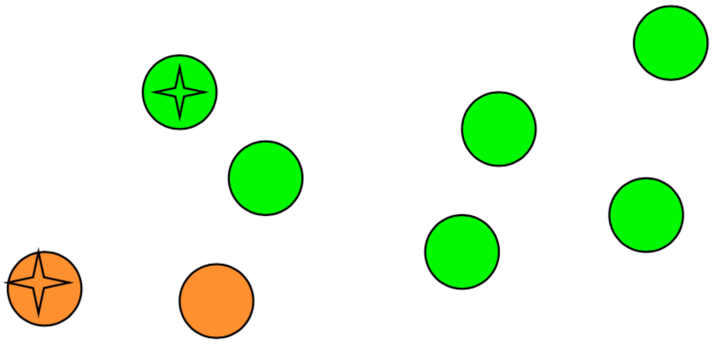
- Two most popular clustering algorithms
  1. K-Means Clustering
  2. Hierarchical Clustering
- k-means starts with k randomly chosen seed points, assigns each remaining point to the nearest seed, and repeats this until no point moves
- Hierarchical builds tree sequentially from the closest pair of points (wells/cells/probes/ conditions)

K-Means Clustering

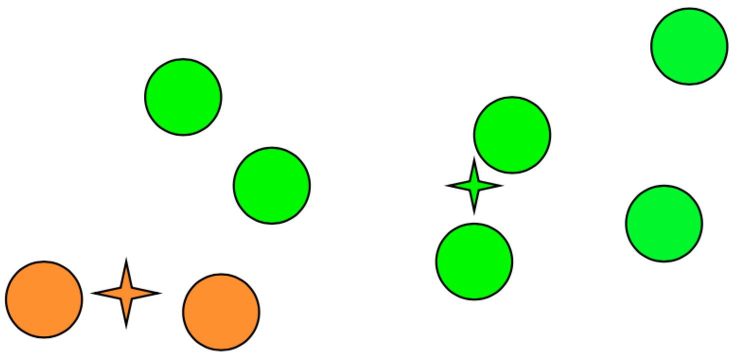
1. Pick an initial set of K centroids (this can be random or any other means)
2. For each data point, assign it to the member of the closest centroid according to the given distance function
3. Adjust the centroid position as the mean of all its assigned member data points. Go back to (2) until the membership isn't change and centroid position is stable.
4. Output the centroids.

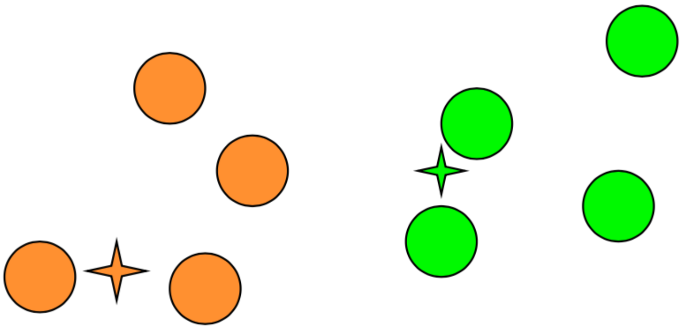


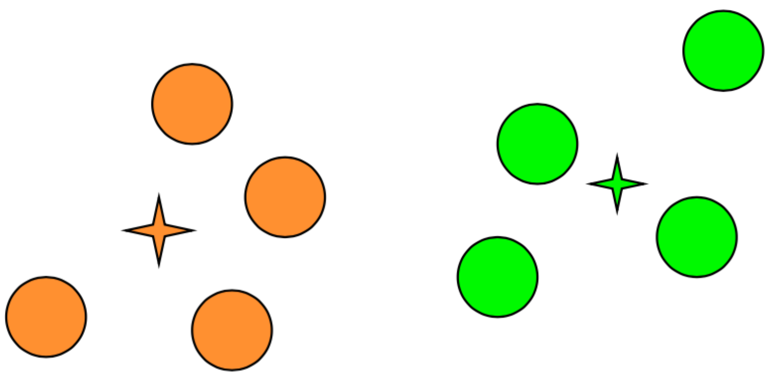

1
2

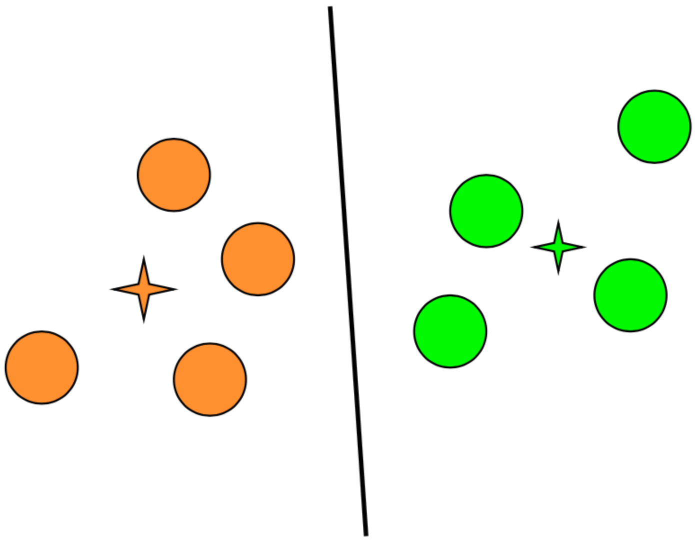


413









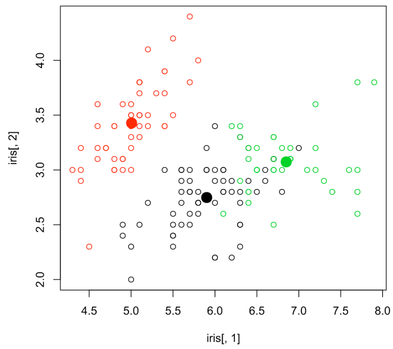
library(stats)

set.seed(101)

km <- kmeans(iris[,1:4], 3)

plot(iris[,1], iris[,2], col=km$cluster)

points(km$centers[,c(1,2)], col=1:3, pch=19, cex=2)



table(km$cluster, iris$Species)

```
   setosa versicolor virginica
1       0         48        14
2      50          0         0
3       0          2        36
>
```

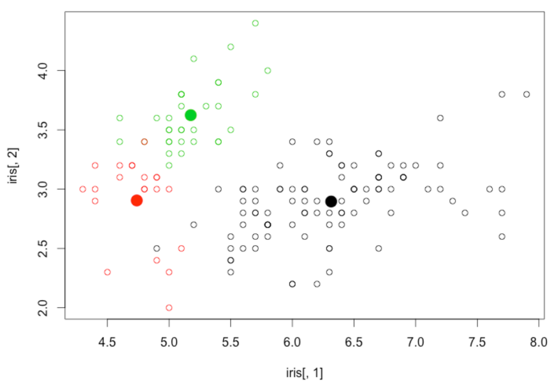

Another round

set.seed(900)

km <- kmeans(iris[,1:4], 3)

plot(iris[,1], iris[,2], col=km$cluster)

points(km$centers[,c(1,2)], col=1:3, pch=19, cex=2)



table(km$cluster, iris$Species)

```
   setosa versicolor virginica
  1      0         46        50
  2     17          4         0
  3     33          0         0
>
```


Hierarchical Clustering

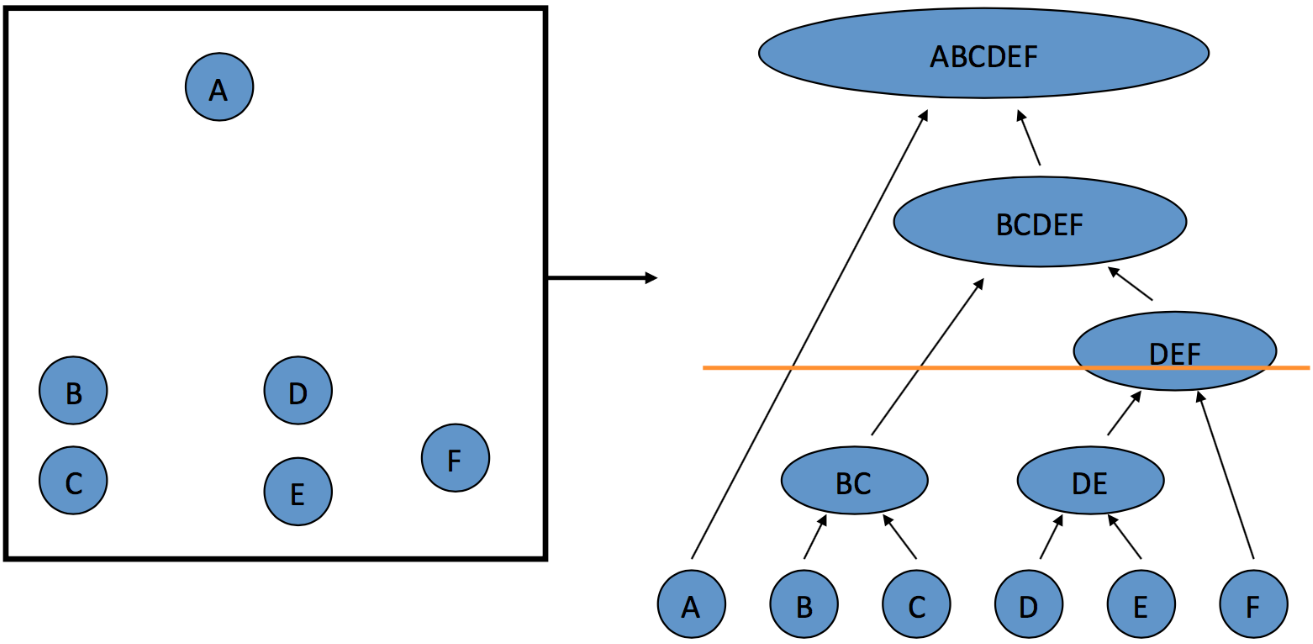
Compute distance between every pairs of point/cluster.

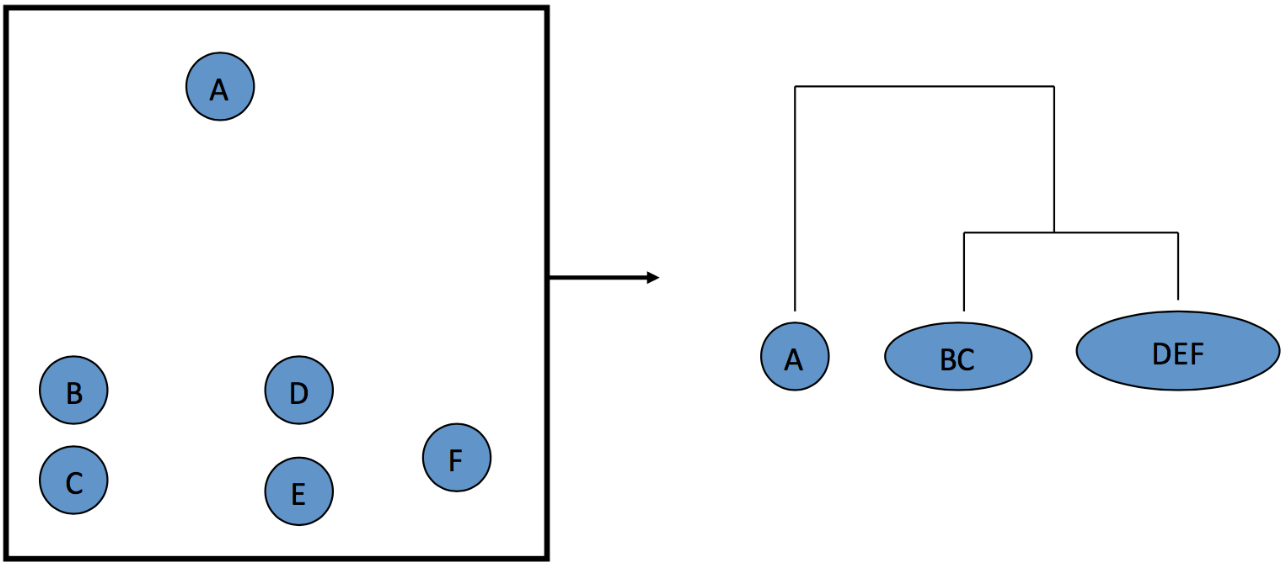
(a) Distance between point is just using the distance function.

(b) Compute distance between pointA to clusterB may involve many choices (such as the min/max/avg distance between the pointA and points in the clusterB).

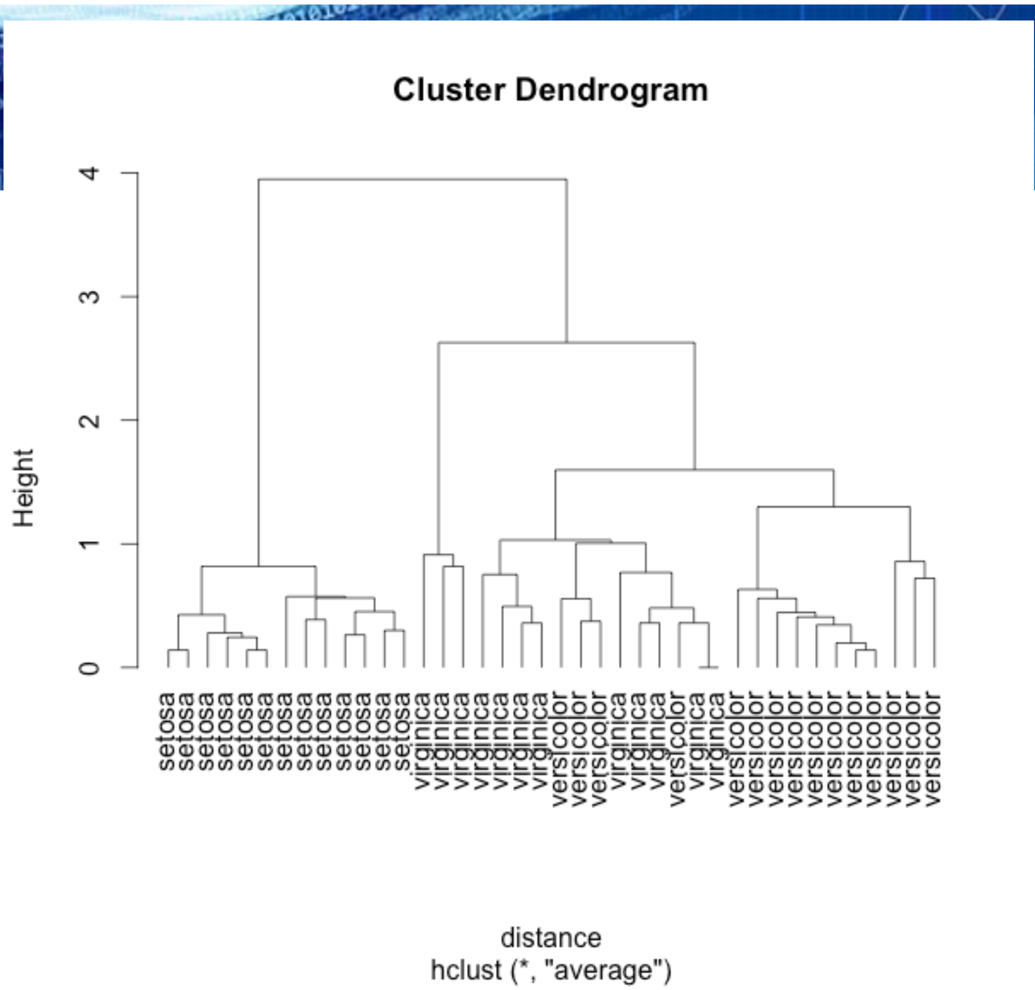
(c) Compute distance between clusterA to clusterB may first compute distance of all points pairs (one from clusterA and the other from clusterB) and then pick either min/max/avg of these pairs.

Combine the two closest point/cluster into a cluster. Go back to (1) until only one big cluster remains


A
B
D
C
E
F
ABCDEF
BCDEF
DEF
BC
DE
A
B
C
D
E
F


A
B
D
C
E
F
A
BC
DEF

```
set.seed(101)
sampleiris <- iris[sample(1:150, 40),] # get samples from iris dataset
# each observation has 4 variables, ie, they are interpreted as 4-D points
distance   <- dist(sampleiris[,-5], method="euclidean")
cluster    <- hclust(distance, method="average")
plot(cluster, hang=-1, label=sampleiris$Species)
```

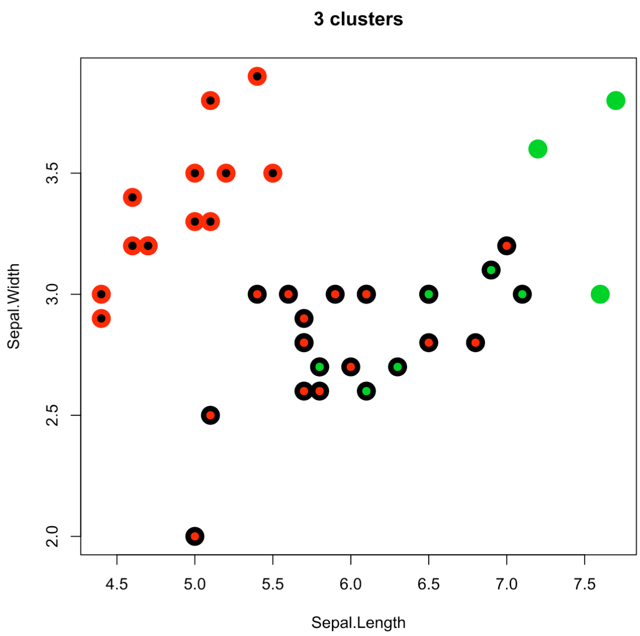
It's possible to prune the result tree.

```
par(mfrow=c(1,2))
group.3 <- cutree(cluster, k = 3)  # prune the tree by 3 clusters
table(group.3, sampleiris$Species) # compare with known classes
```

```
group.3 setosa versicolor virginica
      1      0         15         9
      2     13          0         0
      3      0          0         3
>
```

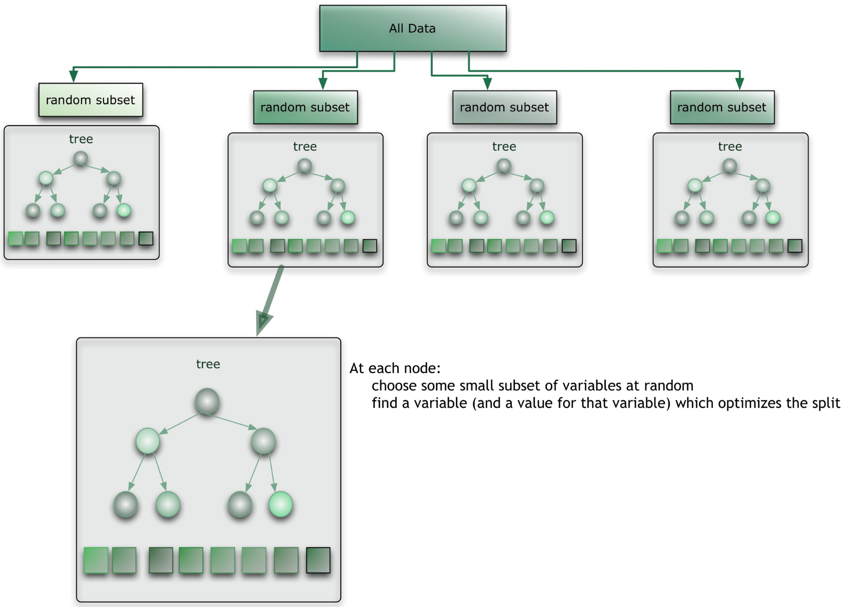


```
plot(sampleiris[,c(1,2)], col=group.3, pch=19, cex=2.5, main="3 clusters")
points(sampleiris[,c(1,2)], col=sampleiris$Species, pch=19, cex=1)
```



Ensemble : Bagging

Random Forest



Random Forest

Here is how such a system is trained; for some number of trees T:

1) Sample N cases at random with replacement to create a subset of the data. The subset should be about 66% of the total set.

2) At each node:

   a) For some number m (see below), m predictor variables are selected at random from all the predictor variables.

   b) The predictor variable that provides the best split, according to some objective function, is used to do a binary split on that node.

   c) At the next node, choose another m variables at random from all predictor variables and do the same.

Bagging

```
> library(randomForest)
#Train 100 trees, random selected attributes
> model <- randomForest(Species~., data=iristrain, nTree=500)
#Predict using the forest
> prediction <- predict(model, newdata=iristest, type='class')
> table(prediction, iristest$Species)
> importance(model)
             MeanDecreaseGini
Sepal.Length         7.807602
Sepal.Width          1.677239
Petal.Length        31.145822
Petal.Width         38.617223
```


Boosting

```
> library(adabag)
> iris.adaboost <- boosting(Species~., data=iristrain, boost=TRUE,
mfinal=5)
> iris.adaboost$class
> table(iris.adaboost$class, iristrain$Species)
> prediction <- predict(iris.adaboost, newdata=iristest)
> table(prediction$class, iristest$Species)
```

Association Rules (Market Basket Analysis)

Support: The fraction of which our item set occurs in our dataset.

Confidence: probability that a rule is correct for a new transaction with items on the left.

Lift: The ratio by which by the confidence of a rule exceeds the expected confidence.

Note: if the lift is 1 it indicates that the items on the left and right are independent

Support, Confidence and Lift

There are several measures used to understand various aspects of associated products.

Let's understand the measures with the help of an example.

- In a store, there are 1000 transactions overall.
- Item A appears in 80 transactions and
- Item B occurs in 100 transactions.
- Items A and B appear in 20 transactions together.

Support is the ratio of number of times two or more items occur together to the total number of transactions.

- **Support of A** = Pr(A) = 80/1000 = 8% and
- **Support of B** = Pr(B) = 100/1000 = 10%.

Confidence is a conditional probability that a randomly selected transaction will include Item A given Item B.

- **Confidence of A** = Pr(A/B) = 20/100 = 20%.

Lift can be expressed as the ratio of the probability of Items A and B occurring together to the multiple of the two individual probabilities for Item A and Item B.

- **Lift** = Pr(A,B) / Pr(A).Pr(B) = (20/1000)/((80/1000)x(100/1000)) = 2.5.

How would you use Support, Confidence and Lift?

Support of a product or product bundle indicates the popularity of the product or product bundle in the transaction set. Higher the support, more popular is the product or product bundle. This measure can help in identifying driver of traffic to the store. Hence, if Barbie dolls have a higher support then they can be attractively priced to attract traffic to a store.
Confidence can be used for product placement strategy and increasing profitability. Place high-margin items with associated high selling (driver) items. If Market Basket Analysis indicates that customers who bought high selling Barbie dolls also bought high-margin candies, then candies should be placed near Barbie dolls.
Lift indicates the strength of an association rule over the random co-occurrence of Item A and Item B, given their individual support. Lift provides information about the change in probability of Item A in presence of Item B. Lift values greater than 1.0 indicate that transactions containing Item B tend to contain Item A more often than transactions that do not contain Item B.

Apriori Algorithm

?apriori

Usage

```
apriori(data, parameter = NULL, appearance = NULL, control = NULL)
```

Arguments

`data`
: object of class `transactions` or any data structure which can be coerced into `transactions` (e.g., a binary matrix or data.frame).

`parameter`
: object of class `APparameter` or named list. The default behavior is to mine rules with support 0.1, confidence 0.8, and maxlen 10.

`appearance`
: object of class `APappearance` or named list. With this argument item appearance can be restricted. By default all items can appear unrestricted.

`control`
: object of class `APcontrol` or named list. Controls the performance of the mining algorithm (item sorting, etc.)

Apriori Algorithm

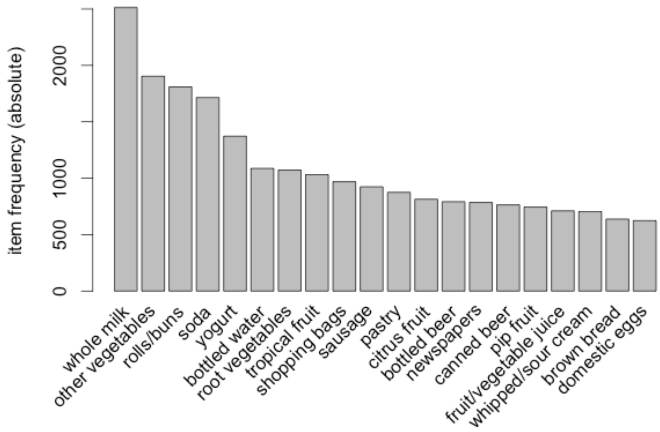
So lets get started by loading up our libraries and data set.

```
# Load the libraries
library(arules)
library(arulesViz)
library(datasets)

# Load the data set
data(Groceries)
```

Explore Data

Create an item frequency plot for the top 20 items

itemFrequencyPlot(Groceries, topN=20,type="absolute")



rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8))

Show the top 5 rules, but only 2 digits

options(digits=2)

inspect(rules[1:5])

```
  lhs                    rhs              support confidence lift
1 {liquor,
   red/blush wine} => {bottled beer}  0.0019       0.90 11.2
2 {curd,
   cereals}        => {whole milk}    0.0010       0.91  3.6
3 {yogurt,
   cereals}        => {whole milk}    0.0017       0.81  3.2
4 {butter,
   jam}            => {whole milk}    0.0010       0.83  3.3
5 {soups,
   bottled beer}   => {whole milk}    0.0011       0.92  3.6
>
```

summary(rules)

```
set of 410 rules

rule length distribution (lhs + rhs):sizes
  3   4   5   6
 29 229 140  12

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    3.0     4.0     4.0     4.3     5.0     6.0

summary of quality measures:
    support           confidence        lift
 Min.    :0.00102   Min.   :0.80   Min.   : 3.1
 1st Qu.:0.00102   1st Qu.:0.83   1st Qu.: 3.3
 Median :0.00122   Median :0.85   Median : 3.6
 Mean   :0.00125   Mean   :0.87   Mean   : 4.0
 3rd Qu.:0.00132   3rd Qu.:0.91   3rd Qu.: 4.3
 Max.   :0.00315   Max.   :1.00   Max.   :11.2

mining info:
      data ntransactions support confidence
 Groceries          9835   0.001        0.8
```


Sort Rules

rules<-sort(rules, by="confidence", decreasing=TRUE)

```
  lhs                        rhs           support confidence lift
1 {rice,
   sugar}                 => {whole milk}  0.0012          1  3.9
2 {canned fish,
   hygiene articles}      => {whole milk}  0.0011          1  3.9
3 {root vegetables,
   butter,
   rice}                  => {whole milk}  0.0010          1  3.9
4 {root vegetables,
   whipped/sour cream,
   flour}                 => {whole milk}  0.0017          1  3.9
5 {butter,
   soft cheese,
   domestic eggs}         => {whole milk}  0.0010          1  3.9
>
```

Change to have limit association in

change to have maximum of 3

rules <- apriori(Groceries, parameter = list(supp = 0.001, conf = 0.8,maxlen=3))

inspect(rules[1:5])

```
  lhs                   rhs            support confidence lift
1 {liquor,
   red/blush wine}  => {bottled beer}  0.0019       0.90 11.2
2 {curd,
   cereals}         => {whole milk}    0.0010       0.91  3.6
3 {yogurt,
   cereals}         => {whole milk}    0.0017       0.81  3.2
4 {butter,
   jam}             => {whole milk}    0.0010       0.83  3.3
5 {soups,
   bottled beer}    => {whole milk}    0.0011       0.92  3.6
```


 # Rules pruned

subset.matrix <- is.subset(rules, rules)

subset.matrix[lower.tri(subset.matrix, diag=T)] <- NA

redundant <- colSums(subset.matrix, na.rm=T) >= 1

rules.pruned <- rules[!redundant]

rules<-rules.pruned

summary(rules)

```
set of 330 rules

rule length distribution (lhs + rhs):sizes
  3   4   5   6
 29 216  84   1

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    3.0     4.0     4.0     4.2     5.0     6.0

summary of quality measures:
    support           confidence         lift
 Min.   :0.00102   Min.   :0.80   Min.   : 3.1
 1st Qu.:0.00102   1st Qu.:0.82   1st Qu.: 3.3
 Median :0.00122   Median :0.85   Median : 3.6
 Mean   :0.00127   Mean   :0.86   Mean   : 3.8
 3rd Qu.:0.00132   3rd Qu.:0.91   3rd Qu.: 4.3
 Max.   :0.00315   Max.   :1.00   Max.   :11.2

mining info:
      data ntransactions support confidence
 Groceries          9835   0.001        0.8
```

Targeting Items

What are customers likely to buy before buying whole milk?

What are customers likely to buy if they purchase whole milk?

This essentially means we want to set either the Left Hand Side and Right Hand Side. This is not difficult to do with R!

Find whole milk’s antecedents

rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.08), appearance = list(default="lhs",rhs="whole milk”), control = list(verbose=F))

rules<-sort(rules, decreasing=TRUE,by="confidence")

inspect(rules[1:5])

```
  lhs                        rhs          support confidence lift
1 {rice,
   sugar}                 => {whole milk}  0.0012          1  3.9
2 {canned fish,
   hygiene articles}      => {whole milk}  0.0011          1  3.9
3 {root vegetables,
   butter,
   rice}                  => {whole milk}  0.0010          1  3.9
4 {root vegetables,
   whipped/sour cream,
   flour}                 => {whole milk}  0.0017          1  3.9
5 {butter,
   soft cheese,
   domestic eggs}         => {whole milk}  0.0010          1  3.9
>
```

Likely to buy after buy whole milk

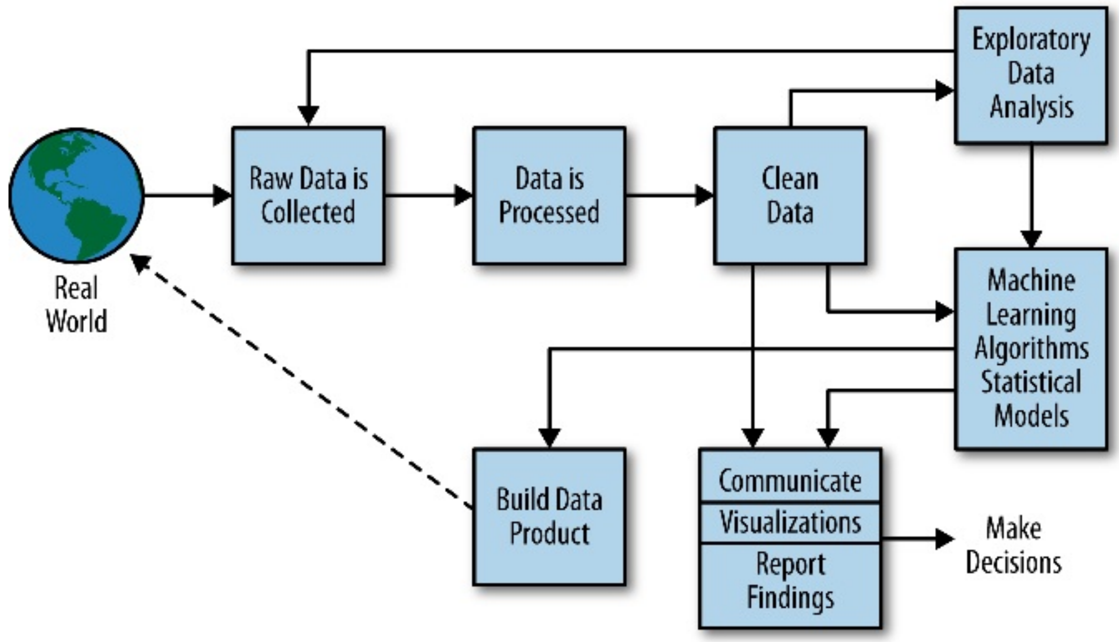
rules<-apriori(data=Groceries, parameter=list(supp=0.001,conf = 0.15,minlen=2), appearance = list(default="rhs",lhs="whole milk”), control = list(verbose=F))

rules<-sort(rules, decreasing=TRUE,by="confidence")

inspect(rules[1:5])

```
  lhs             rhs                support confidence lift
1 {whole milk} => {other vegetables}   0.075       0.29  1.5
2 {whole milk} => {rolls/buns}         0.057       0.22  1.2
3 {whole milk} => {yogurt}             0.056       0.22  1.6
4 {whole milk} => {root vegetables}    0.049       0.19  1.8
5 {whole milk} => {tropical fruit}     0.042       0.17  1.6
>
```


Data Science Process



Thank you

Email : veerasak.kritsanapraphan@gmail.com