

# How I Learned to Stop Worrying and Love the R Console

Irfan E Kanat

Department of Information Systems  
W.P. Carey School of Business  
Arizona State University

November 18, 2015



# Outline

- 1 Introduction
  - Familiar Examples
- 2 R Console
- 3 Importing Data
- 4 Packages
- 5 Sample Analysis and Visualizations
  - Data Manipulation
  - Descriptive Visualizations
  - Modeling
- 6 Reporting
- 7 Where to Go Next?

# Who am I?

Irfan Kanat, PhD Candidate

R user since 2006

Open Source Evangelist

# Before We Begin

Got R & R Studio Installed?

Get your workshop documents:

<https://github.com/iekanat/rworkshop>

Install caret package:

```
install.packages("caret", dependencies = c("Depends", "Suggests"))
```

# What is this about?

A brief introduction to R.

- R Console
- Importing Data
- Packages
- Sample analyses
- Basic visualization
- Where to get help?



# What is R?

From R project web site:

R is a language *and* an environment for statistical computing and graphics.

- Language
- Environment
- Statistics and Visualization



# What is R?

All this means R is very flexible, which played a huge role in its success.

My take: Low cost, high quality, open source solution for your analysis needs.

# When to Use R?

R is very strong for your classical machine learning and statistical analysis. Thousands of packages address almost all analysis needs. It is a logical first step to start analysis.



# When to Use R?

R is very strong for your classical machine learning and statistical analysis. Thousands of packages address almost all analysis needs. It is a logical first step to start analysis.

Yet it's core design is starting to show its age. There are certain down sides to traditional R:

- Everything is stored in memory<sup>1</sup>
- R is single core<sup>1</sup>

---

<sup>1</sup>Except when it is not. There are packages to overcome these issues.

# Best Part of R

**Packages** CRAN houses over 7K packages. Providing functionality way beyond what is available in commercial packages.

**Community** Millions of users mean, all your questions are either already answered or will be in hours.

**Performance** While memory and core restrictions are real, for the cost of a single user license of a commercial package, you can buy better hardware to run R. Furthermore, with the packages providing multicore and flatfile functionality, R performance is on par or better than commercial packages

## Subsection 1

### Familiar Examples

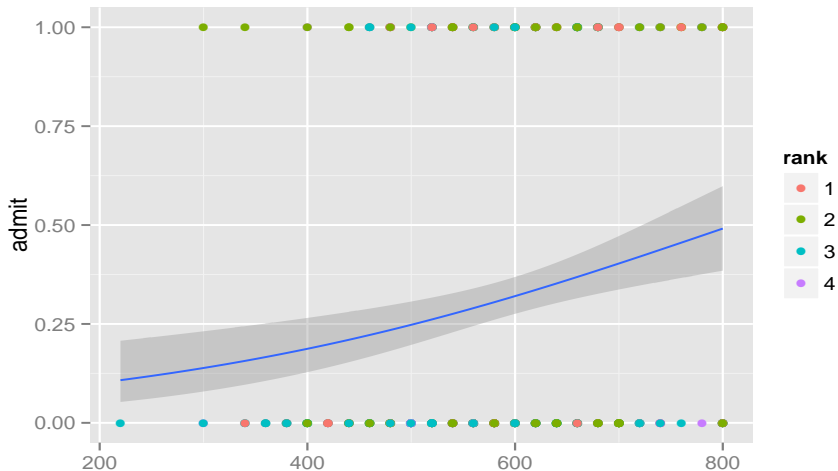
# Logistic Regression

```
# Fit the model
logit_0 <- glm(admit ~ ., admitData, family = "binomial")
# Display fitted model
summary(logit_0)
```

```
##
## Call:
## glm(formula = admit ~ ., family = "binomial", data = admitData)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.6268  -0.8662  -0.6388   1.1490   2.0790
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.989979   1.139951  -3.500  0.000465 ***
## gre          0.002264   0.001094   2.070  0.038465 *
## gpa          0.804038   0.331819   2.423  0.015388 *
## rank2       -0.675443   0.316490  -2.134  0.032829 *
## rank3       -1.340204   0.345306  -3.881  0.000104 ***
## rank4       -1.551464   0.417832  -3.713  0.000205 ***
## ---
```

# Logistic Regression

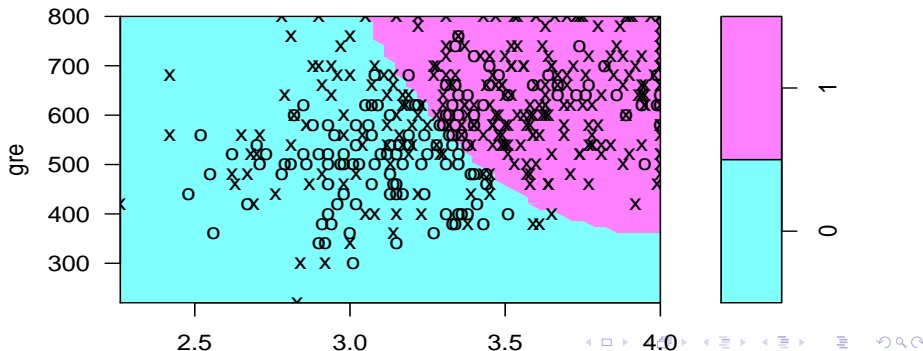
```
ggplot(admitData, aes(x = gre, y = admit)) + geom_point(aes(colour = rank)) +  
  stat_smooth(method = "glm", family = "binomial", se = T)
```



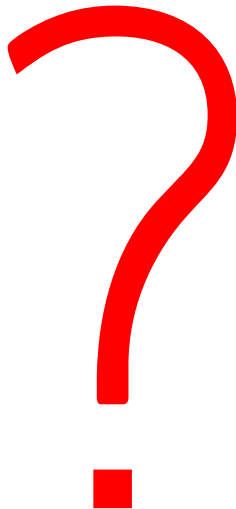
# Support Vector Machine

```
# Fit the model  
svm_0 <- svm(admit ~ ., data = admitData, type = "C-classification")  
# Plot the results  
plot(svm_0, admitData, gre ~ gpa) # Let us plot the results
```

**SVM classification plot**



# Questions



# Outline

- 1 Introduction
  - Familiar Examples
- 2 R Console
- 3 Importing Data
- 4 Packages
- 5 Sample Analysis and Visualizations
  - Data Manipulation
  - Descriptive Visualizations
  - Modeling
- 6 Reporting
- 7 Where to Go Next?



# Command Driven Interface

Command line may be intimidating

Power over Convenience

Consider the number of

- Functions
- Parameters
- Data sources
- Variables
- Replications

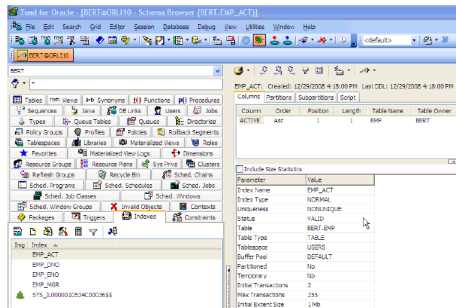
# Command Driven Interface

Command line may be intimidating

Power over Convenience

Consider the number of

- Functions
- Parameters
- Data sources
- Variables
- Replications



# R Studio

The screenshot displays the R Studio interface with four main panels:

- Source Editor (Top Left):** Contains an R script. A red box labeled 'A' highlights the first 44 lines, which include comments about project structure and basic R console examples. The script ends with a chunk of code for SVM classification.
- Environment (Top Right):** Shows the current workspace. A blue box labeled 'C' highlights the 'Data' section, listing objects: `admtData` (400 obs. of 4 variables), `csv` (24 obs. of 3 variables), `d` (10 obs. of 3 variables), `xls` (29 obs. of 3 variables), and `Values` (a list of 5 elements).
- Console (Bottom Left):** Shows the output of the script. A green box labeled 'B' highlights the console output, which includes the results of the SVM classification, such as the null deviance, residual deviance, AIC, and the number of Fisher Scoring iterations.
- Plots (Bottom Right):** Displays an SVM classification plot titled 'SVM classification plot'. The x-axis is 'gpa' (ranging from 2.5 to 4.0) and the y-axis is 'pre' (ranging from 300 to 800). The plot shows two classes of data points (circles and crosses) separated by a decision boundary. A yellow box labeled 'D' highlights a specific region of the plot.

# New Project

File > New Project

Empty Directory > Empty Project > Directory Name: Workshop

# R as a Calculator I

```
# Arithmetics
```

```
2 + 2
```

```
## [1] 4
```

```
2 * 3
```

```
## [1] 6
```

```
2^3
```

```
## [1] 8
```

```
log(100, 10)
```

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

# R as a Calculator II

```
# Logic
1 == 2

## [1] FALSE

1 != 2

## [1] TRUE

2 < 3

## [1] TRUE
```

# Variables I

```
A <- 2
```

```
A
```

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

```
a # Case sensitive
```

```
## Error in eval(expr, envir, enclos): object 'a' not found
```

```
"A" != "a" # Explanation
```

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

```
B <- 7
```

```
A + B
```

```
## [1] 9
```

# Variables II

```
C <- c(1, 3, 7, 9) # A list can be in a variable
```

```
C
```

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

```
C + A
```

```
## [1] 3 5 9 11
```

```
C * A
```

```
## [1] 2 6 14 18
```

```
C < 5
```

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



# Indexes and Data Frames I

```
1:30
```

```
## [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
```

```
C[3]
```

```
## [1] 7
```

```
C[c(2, 3)]
```

```
## [1] 3 7
```

```
C[1:3]
```

```
## [1] 1 3 7
```

# Indexes and Data Frames II

```
Countries <- data.frame(names=c("US", "TR", "DE"), supply=c(10, 8, 7),  
                        those=c(TRUE, FALSE, FALSE))
```

Countries

```
##   names supply those  
## 1    US     10  TRUE  
## 2    TR      8 FALSE  
## 3    DE      7 FALSE
```

# Indexes and Data Frames III

```
Countries[2, ]
```

```
##      names supply those  
## 2      TR      8 FALSE
```

```
Countries[, 3]
```

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

```
Countries[2, 3]
```

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

```
Countries[1:2, ]
```

```
##      names supply those  
## 1      US     10  TRUE  
## 2      TR      8 FALSE
```

# Indexes and Data Frames IV

```
Countries[, "names"]
```

```
## [1] US TR DE  
## Levels: DE TR US
```

```
Countries$names
```

```
## [1] US TR DE  
## Levels: DE TR US
```

```
Countries$those
```

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

# Loops in R

## CAUTION!

R is notoriously inefficient with your classic loops

- Structure of the Data Frame
- Memory Management

Try to use an apply function instead.

Vectorize your operations.

# For Loop in R

```
for (i in 1:3) print(i)

## [1] 1
## [1] 2
## [1] 3

# Iterating through a data frame
for (i in 1:nrow(Countries)) {
  print(Countries[i, ])
}

##      names supply those
## 1      US      10  TRUE
##      names supply those
## 2      TR       8 FALSE
##      names supply those
## 3      DE       7 FALSE
```

# Functions I

```
mean(C)  # Takes parameters
```

```
## [1] 5
```

```
mean(C, trim = 0.1, na.rm = T)  # Takes multiple parameters
```

```
## [1] 5
```

```
log(sum(C)/length(C))  # Can be combined
```

```
## [1] 1.609438
```

# Functions II

```
HelloWorld <- function(x, y = 1) {  
  for (i in 1:y) {  
    print(paste("Hello", x))  
  }  
}
```

```
HelloWorld("MSBA")
```

```
## [1] "Hello MSBA"
```

```
HelloWorld("MSBA", 2)
```

```
## [1] "Hello MSBA"
```

```
## [1] "Hello MSBA"
```



# Functions III

```
HelloWorld  # Review the source code
```

```
## function(x, y = 1) {  
##   for (i in 1:y) {  
##     print(paste("Hello", x))  
##   }  
## }
```

```
ls
```

```
## function (name, pos = -1L, envir = as.environment(pos), all.names = FALSE,  
##   pattern, sorted = TRUE)  
## {  
##   if (!missing(name)) {  
##     pos <- tryCatch(name, error = function(e) e)  
##     if (inherits(pos, "error")) {  
##       name <- substitute(name)  
##       if (!is.character(name))  
##         name <- deparse(name)  
##       warning(gettextf("%s converted to character string",  
##         sQuote(name))), domain = NA)  
##       pos <- name
```

# Commonly Used Functions I

```
ls() # Get a list of objects in the workspace

## [1] "A"          "admitData"  "B"          "C"          "Countries"
## [6] "HelloWorld" "i"          "logit_0"    "svm_0"
```

```
ls(pattern = "*_0") # partial match on object search

## [1] "logit_0" "svm_0"
```

```
rm("svm_0") # Remove an object from the workspace

# rm(list=ls(pattern=ls())) # This would remove everything if ran
```

# Commonly Used Functions II

```
mean(A)  # Mean  
  
## [1] 2  
  
sd(admitData[, "gre"])  # Standard Deviation  
  
## [1] 115.5165  
  
AIC(logit_0)  
  
## [1] 470.5175
```

# Commonly Used Functions III

```
str(Countries) # Look at the structure of objects
```

```
## 'data.frame': 3 obs. of 3 variables:
## $ names : Factor w/ 3 levels "DE","TR","US": 3 2 1
## $ supply: num 10 8 7
## $ those : logi TRUE FALSE FALSE
```

```
summary(Countries) # Get summary of data
```

```
## names      supply      those
## DE:1   Min.    : 7.000   Mode :logical
## TR:1   1st Qu.: 7.500   FALSE:2
## US:1   Median : 8.000   TRUE :1
##        Mean    : 8.333   NA's :0
##        3rd Qu.: 9.000
##        Max.    :10.000
```

# Commonly Used Functions IV

```
summary(logit_1) # Get summary of model

##
## Call:
## glm(formula = admit ~ gre, family = "binomial", data = admitData)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.1623  -0.9052  -0.7547   1.3486   1.9879
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.901344   0.606038  -4.787 1.69e-06 ***
## gre          0.003582   0.000986   3.633 0.00028 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 499.98  on 399  degrees of freedom
## Residual deviance: 486.06  on 398  degrees of freedom
## AIC: 490.06
##
## Number of Fisher Scoring iterations: 4
```

# Commonly Used Functions V

```
cor(admitData[, 1:3]) # Get correlation matrix
```

```
##           admit           gre           gpa
## admit 1.0000000 0.1844343 0.1782123
## gre   0.1844343 1.0000000 0.3842659
## gpa   0.1782123 0.3842659 1.0000000
```

# Questions



# Outline

- 1 Introduction
  - Familiar Examples
- 2 R Console
- 3 Importing Data**
- 4 Packages
- 5 Sample Analysis and Visualizations
  - Data Manipulation
  - Descriptive Visualizations
  - Modeling
- 6 Reporting
- 7 Where to Go Next?



# Importing Data

R allows importing data from a wide variety of sources.

- Comma Separated Values (CSV)
- Databases
- Flat files
- Lesser statistical packages
- and more

# Importing CSV Files

CSV has certain advantages that make it popular.

- Compatibility
- Flexibility
- Simplicity

## Sample

```
"iso2", "Supply", "Those"  
"AU", 20, 0  
"TR", 80, 1  
"US", 100, 0  
"GB", 50, 0  
"DE", 70, 0
```

We use `read.csv()` or `read.csv2()` commands to import the csv files.

```
saveData <- read.csv("PathToCSV", header = TRUE, sep = ",", quote = "\"")
```

# Working with Excel Files

Much like CSV, except it lacks the simplicity, flexibility, and compatibility of CSV.

```
# Load the necessary library  
library(xlsx)  
# Read in the data from excel file  
xlsx <- read.xlsx("country.xlsx", sheetIndex = 1)
```

# Working with Databases

No speed advantage.

Data larger than memory.

Working with databases:

- Work in the database.
- Import data from database.



# Working with Databases

```
# Load the necessary library
library(RMySQL)
# Establish connection to the database.
channel <- dbConnect(MySQL(), user = "uname", password = "pwd", host = "127.0.0.1",
  dbname = "exampledata")
# Send query and save results in R workspace
sql <- dbGetQuery(channel, "SELECT * FROM table;")
```

# Lesser Statistical Packages :P

Foreign Package

Newer file formats

- sas7bdat
- readstata13



# Questions



# Outline

- 1 Introduction
  - Familiar Examples
- 2 R Console
- 3 Importing Data
- 4 Packages**
- 5 Sample Analysis and Visualizations
  - Data Manipulation
  - Descriptive Visualizations
  - Modeling
- 6 Reporting
- 7 Where to Go Next?



# Packages: Source of R's Power

Encountered already

Make R extendible

Like libraries

Collection of:

- functions
- documentation
- data files



# Gifts from the Community

Currently over 7000 packages

for

- Statistical Modeling
- Machine Learning
- Data Manipulation
- Visualization
- ...

from

- Economics
- Computer Science
- Statistics
- Medicine
- ...



## Great but Where are My Gifts?

# Comprehensive R Archive Network (CRAN)

A Group of FTP and HTML servers hosting R packages.

R has built in package management facilities.

Most of these can be achieved through the R Studio GUI. (Area D, packages pane)

# Package Management

```
# Installing a package
install.packages("e1071") # Notice the quotes around package name
# Loading package into memory
library(e1071) # Notice the lack of quotes
# Unload package
detach("package:e1071", unload = TRUE) # Notice the package: prepended

# Get the list of packages loaded
(.packages())
# Get list of all installed packages (output omitted)
.packages(all.available = T)
```

# How to Find Packages

If you want to search a certain word in installed packages' documentation, you can always use `??` or `help.search()`

```
??mixed  
help.search("mixed model")
```

Internet searches are a bit problematic as R can be a bit ambiguous until Google learns you are interested in the statistical computing environment.

Comprehensive R Archive Network (CRAN)

R Forge

R site search also available with command `RSiteSearch()`

R seek

# Commonly Used Packages: Data Manipulation

**data.tables** Replaces traditional `data.frame`.

- Faster access/write
- Improved selection
- Improved subsetting
- Improved aggregation

Not a drop-in replacement as it breaks compatibility in some cases.

## **ddplyr**

Additional functionality for:

- selection
- filtering
- aggregation

Provides efficient back-end data structures to speed things up.

Works with databases as well.

# Commonly Used Packages: Statistics

Multiple Regression: Stats package, `lm()` (loaded by default)

Generalized Linear Models: Stats package, `glm()`

Traditional Econometric Models: `plm` package

Mixed Modeling: `nlme` and `lme4` packages

## Commonly Used Packages: Machine Learning

Most probably all you need is caret package.

Caret package is a wrapper for a host of classification and regression model training functions. It eases visualizations, data manipulation, and analytics among others. **It currently supports over 150 types of models.**

If you insist on using individual packages:

Classifiers: class package

Support Vector Machines: kernlab, e1071 packages

Clustering: Base package (kmeans(), hclust()), mclust package

Neural Networks: neuralnet package.



# Questions



# Outline

- 1 Introduction
  - Familiar Examples
- 2 R Console
- 3 Importing Data
- 4 Packages
- 5 Sample Analysis and Visualizations**
  - Data Manipulation
  - Descriptive Visualizations
  - Modeling
- 6 Reporting
- 7 Where to Go Next?

## Subsection 1

# Data Manipulation

# If Wishes Were Data...

We may wish for an ideal dataset, what we will invariably get is a mess.

We often need to:

- Combine datasets
- Transform variables
- Aggregate data

I am a `data.tables` person, but I will teach a tiny bit of `dplyr` for compatibility reasons.

# Data I

We will investigate **GDP figures from world bank** and a separate dataset for continents.

```
GDP <- read.csv("GDP.csv")
head(GDP, 3)
```

```
##      Country ISO2   X2003   X2004   X2005   X2006   X2007   X2008
## 1      Aruba  AW      NA      NA      NA      NA      NA      NA
## 2    Andorra  AD      NA      NA      NA      NA      NA      NA
## 3 Afghanistan AF 1096.756 1066.685 1145.717 1173.001 1297.821 1310.717
##      X2009   X2010   X2011   X2012
## 1      NA      NA 36016.484      NA
## 2      NA      NA      NA      NA
## 3 1547.539 1637.297 1695.153 1893.076
```

# Data II

```
Continents <- read.csv("continent.csv")  
head(Continents)
```

```
##      ISO2 Continent  
## 1     AD          EU  
## 2     AE          AS  
## 3     AF          AS  
## 4     AG          AN  
## 5     AI          AN  
## 6     AL          EU
```

# Merging Dataframes I

## Appending data.

```
rbind(Continents[1:3, ], Continents[231:233, ])
```

```
##      ISO2 Continent
## 1      AD          EU
## 2      AE          AS
## 3      AF          AS
## 231    UA          EU
## 232    UG          AF
## 233    UM          OC
```

```
cbind(Continents[1:3, ], Continents[231:233, ])
```

```
##      ISO2 Continent ISO2 Continent
## 1      AD          EU    UA          EU
## 2      AE          AS    UG          AF
## 3      AF          AS    UM          OC
```

# Merging Dataframes II

When you want to combine two data frames by a common column

```
cData <- merge(Continents, GDP, by = "ISO2")
head(cData[, 1:6])
```

##	ISO2	Continent	Country	X2003	X2004	X2005
## 1	AD	EU	Andorra	NA	NA	NA
## 2	AE	AS	United Arab Emirates	110549.415	111543.925	103139.799
## 3	AF	AS	Afghanistan	1096.756	1066.685	1145.717
## 4	AG	AN	Antigua and Barbuda	19566.329	20395.483	21414.230
## 5	AL	EU	Albania	6286.205	6699.225	7119.290
## 6	AM	AS	Armenia	4181.720	4635.303	5296.814



# Subsetting I

```
library(dplyr)
# Subset of rows
filter(cData, Continent == "OC" & X2011 > 23000)
```

##	ISO2	Continent	Country	X2003	X2004	X2005	X2006	X2007
## 1	AU	OC	Australia	37035.05	38129.81	38840.24	39416.04	40643.45
## 2	NZ	OC	New Zealand	29754.46	30355.86	30984.47	31182.26	31953.38
##	X2008	X2009	X2010	X2011	X2012			
## 1	41311.94	41170.05	41329.95	41706.00	42529.87			
## 2	31058.21	31398.28	31227.55	31683.45	32281.25			

# Subsetting II

```
# Subset of columns
```

```
select(cData[1:3, ], X2003:X2009) # All columns between X2003 and X2012
```

	X2003	X2004	X2005	X2006	X2007	X2008	X2009
## 1	NA	NA	NA	NA	NA	NA	NA
## 2	110549.415	111543.925	103139.799	96399.737	83655.038	73611.390	61725.280
## 3	1096.756	1066.685	1145.717	1173.001	1297.821	1310.717	1547.539

```
select(cData[1:3, ], -(IS02:Country)) # All except these
```

	X2003	X2004	X2005	X2006	X2007	X2008	X2009
## 1	NA	NA	NA	NA	NA	NA	NA
## 2	110549.415	111543.925	103139.799	96399.737	83655.038	73611.390	61725.280
## 3	1096.756	1066.685	1145.717	1173.001	1297.821	1310.717	1547.539

	X2010	X2011	X2012
## 1	NA	NA	NA
## 2	57379.972	56376.770	57044.578
## 3	1637.297	1695.153	1893.076

# Aggregating by Groups I

```
# Create grouped data
contiData <- group_by(cData, Continent)
contiData

## Source: local data frame [208 x 13]
## Groups: Continent
##
##   ISO2 Continent      Country      X2003      X2004      X2005
## 1   AD          EU      Andorra        NA        NA        NA
## 2   AE          AS United Arab Emirates 110549.415 111543.925 103139.799
## 3   AF          AS      Afghanistan    1096.756    1066.685    1145.717
## 4   AG          AN  Antigua and Barbuda 19566.329 20395.483 21414.230
## 5   AL          EU      Albania        6286.205    6699.225    7119.290
## 6   AM          AS      Armenia        4181.720    4635.303    5296.814
## 7   AO          AF      Angola         3818.663    4086.858    4667.346
## 8   AR          SA      Argentina        NA        NA        NA
## 9   AS          OC      American Samoa    NA        NA        NA
## 10  AT          EU      Austria        39732.713 40555.386 41142.303
## .. ...
## Variables not shown: X2006 (dbl), X2007 (dbl), X2008 (dbl), X2009 (dbl),
##   X2010 (dbl), X2011 (dbl), X2012 (dbl)
```

# Aggregating by Groups II

```
# Create variables on the fly
summarise(contiData, count = n(), GDP2012 = mean(X2012, na.rm = T))

## Source: local data frame [6 x 3]
##
##   Continent count  GDP2012
## 1         AF     52  5424.917
## 2         AN     31 17795.502
## 3         AS     49 24344.030
## 4         EU     46 29655.267
## 5         OC     18  9795.367
## 6         SA     12 12541.870
```

# Reshape in Your Own Image

```
cData_Long <- reshape(cData, varying = 4:13, direction = "long", sep = "")  
# Drop the unnecessary id column  
cData_Long <- subset(cData_Long, select = -c(id, Country))  
# Drop the missing observations  
cData_Long <- na.exclude(cData_Long)  
# Get rid of empty levels  
cData_Long$IS02 <- droplevels(cData_Long$IS02)  
# Sort based on IS02 and Year  
cData_Long <- cData_Long[order(cData_Long$IS02, cData_Long$IS02), ]  
head(cData_Long, 3)
```

```
##          IS02 Continent time          X  
## 2.2003      AE          AS 2003 110549.4  
## 2.2004      AE          AS 2004 111543.9  
## 2.2005      AE          AS 2005 103139.8
```

# Transforming Variables I

```
# Basic Log Transform
```

```
cData_Long$logGDP <- log(cData_Long$X)
```

```
head(cData_Long)
```

```
##      ISO2 Continent time      X    logGDP
## 2.2003    AE        AS  2003 110549.41 11.61322
## 2.2004    AE        AS  2004 111543.92 11.62217
## 2.2005    AE        AS  2005 103139.80 11.54384
## 2.2006    AE        AS  2006  96399.74 11.47626
## 2.2007    AE        AS  2007  83655.04 11.33446
## 2.2008    AE        AS  2008  73611.39 11.20656
```

```
# ASSUMING YOUR DATA IS SORTED
```

```
# More Interesting: Cumulative Sum By Groups
```

```
cData_Long$cumGDP <- ave(cData_Long$X, cData_Long$ISO2, FUN = cumsum)
```

# Transforming Variables II

```
# A function, given a vector, shifts every observation by 1 and drops the
# last one.
lg <- function(x) c(NA, x[1:(length(x) - 1)])
lg(1:10) # See how it works

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

# Run the function with group averages function
cData_Long$lagGDP <- ave(cData_Long$X, cData_Long$IS02, FUN = lg)

## Warning in 'split<-.default'('*tmp*', g, value = lapply(split(x, g), FUN)):
number of items to replace is not a multiple of replacement length

head(cData_Long, 3)

##          IS02 Continent time          X  logGDP  cumGDP  lagGDP
## 2.2003      AE          AS 2003 110549.4 11.61322 110549.4      NA
## 2.2004      AE          AS 2004 111543.9 11.62217 222093.3 110549.4
## 2.2005      AE          AS 2005 103139.8 11.54384 325233.1 111543.9
```

## Subsection 2

### Descriptive Visualizations



# Motor Trends Dataset

We will use 1974 Motor Trend dataset. It has 32 observations and 11 variables.

- mpg: Miles per gallon
- cyl: Number of cylinders
- disp: Displacement
- hp: Horse Power
- drat: Rear axle ratio
- wt: Weight
- qsec: quarter mile time
- vs: V - S
- am: 0 automatic, 1 manual
- gear: Gears
- carb: Number of carburetors

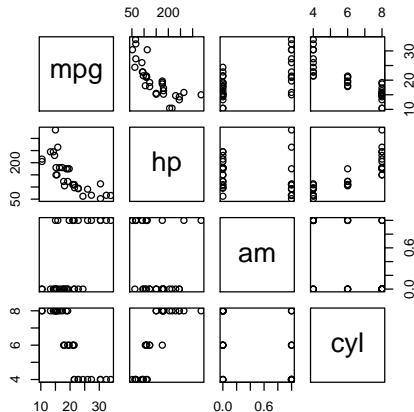
# Motor Trends Dataset I

```
summary(mtcars)
```

```
##           mpg           cyl           disp           hp
##  Min.    :10.40   Min.    :4.000   Min.    : 71.1   Min.    : 52.0
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
##
##      drat           wt           qsec           vs
##  Min.    :2.760   Min.    :1.513   Min.    :14.50   Min.    :0.0000
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000
##
##      am           gear           carb
##  Min.    :0.0000   Min.    :3.000   Min.    :1.000
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
##  Median :0.0000   Median :4.000   Median :2.000
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

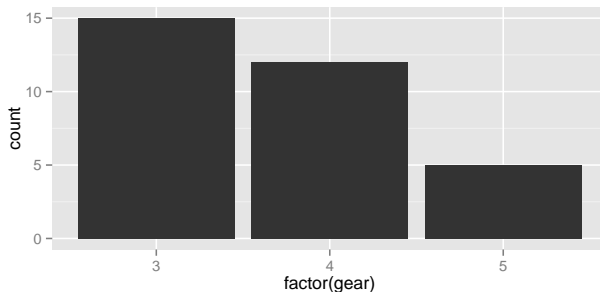
# Motor Trends Dataset II

```
pairs(mtcars[, c("mpg", "hp", "am", "cyl")]) # Visualize Correlations
```



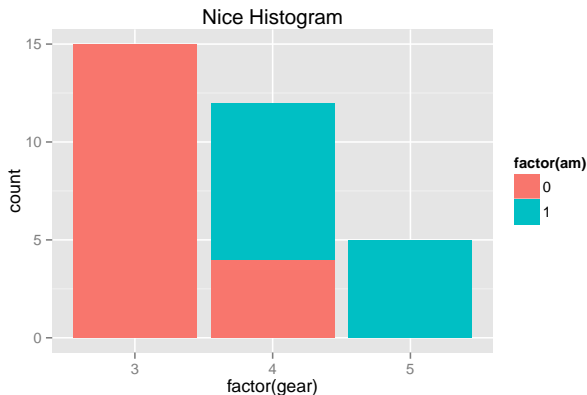
# Histograms I

```
data(mtcars) # Load the Dataset
library(ggplot2) # Load the ggplot package
# Nr of cars by number of gears
qplot(factor(gear), data = mtcars, geom = "bar")
```



# Histograms II

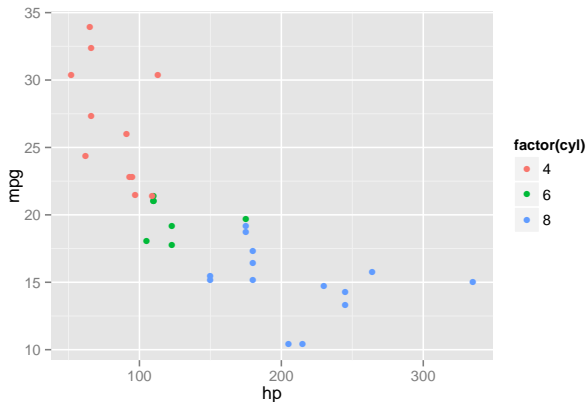
```
# If we are interested in a third categorical variable vs:  
qplot(factor(gear), data=mtcars, geom="bar", fill=factor(am)) +  
ggtitle('Nice Histogram') # This is how you add a title
```



# Scatter Plots I

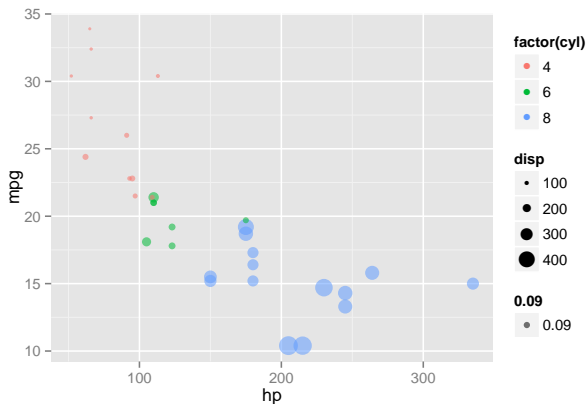
```
# Two continuous variables
```

```
qplot(hp, mpg, data = mtcars, color = factor(cyl))
```



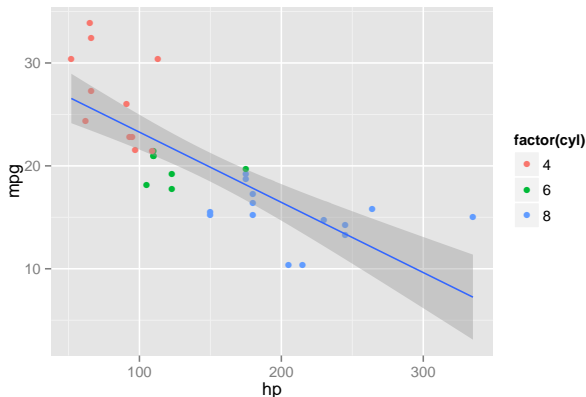
# Scatter Plots II

```
# Add two more variables represented by color and size of points  
qplot(hp, mpg, data = mtcars, color = factor(cyl), size = disp, alpha = 0.09)
```



# Scatter Plots III

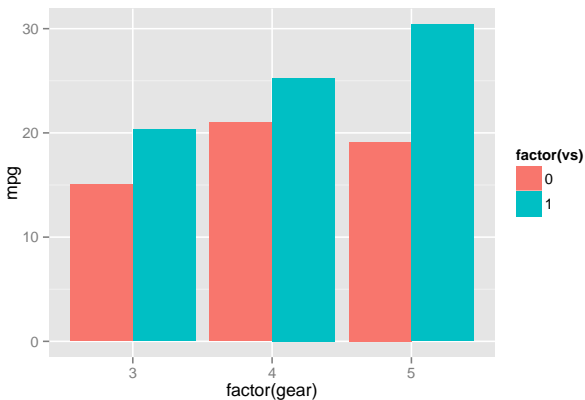
```
ggplot(mtcars, aes(x = hp, y = mpg)) + geom_point(aes(color = factor(cyl))) +  
  # Add a regression line  
  geom_smooth(method = lm)
```





# Bar Charts I

```
ggplot(mtcars, aes(x = factor(gear), y = mpg, fill = factor(vs)), color = factor(vs))  
  stat_summary(fun.y = mean, position = position_dodge(), geom = "bar")
```



# Bonus! Map Visualizations I

```
## Display GDP Data on Map

library(rworldmap) # Install if necessary

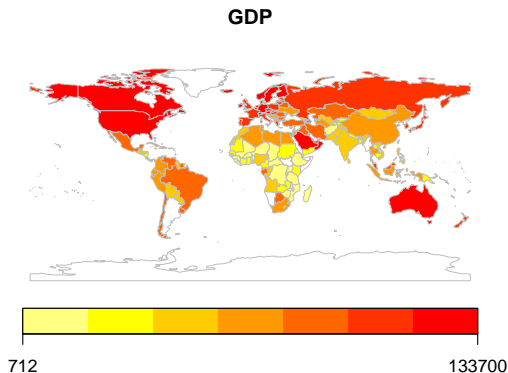
cDataL2011 <- filter(cData_Long, time==2011)
# Rename the columns
colnames(cDataL2011)[4] <- "GDP"

# Turn into map
cDataL2011<- joinCountryData2Map(cDataL2011, joinCode = "ISO2",
                                nameJoinColumn = 'ISO2')

## 187 codes from your data successfully matched countries in the map
## 0 codes from your data failed to match with a country code in the map
## 55 codes from the map weren't represented in your data
```

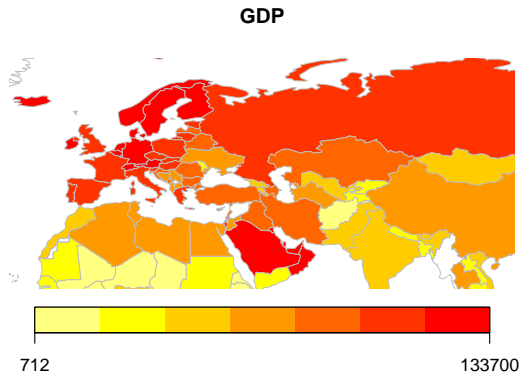
```
mapCountryData(cDataL2011, nameColumnToPlot = "GDP")
```

# Bonus! Map Visualizations II



```
mapCountryData(cDataL2011, nameColumnToPlot = "GDP", mapRegion = "eurasia")
```

# Bonus! Map Visualizations III



## Subsection 3

### Modeling

# Formula Interface

Pay close attention to how we specify the model.

## R Model

$$Y \sim x_1 + x_2$$

This basic structure will remain constant across many R packages.

# Nifty Tricks with Formula Interface

If you have lots of variables use the shortcut `'.'`.

$$\text{mpg} \sim .$$

You can do transformations on the fly, no need to create variables.

$$\log(\text{mpg} + 1) \sim .$$

Dummy variables.

$$\log(\text{mpg} + 1) \sim \text{factor}(\text{gears})$$

# Multiple Regression

We will keep using motor trends data set.

## Formula

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$$

## R Model

$$Y \sim x_1 + x_2$$



# Regression I

```
# Let us estimate gas milage
reg_0 <- lm(mpg ~ hp + cyl + am, mtcars)
summary(reg_0)

##
## Call:
## lm(formula = mpg ~ hp + cyl + am, data = mtcars)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -4.864 -1.811 -0.158  1.492  6.013
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  30.88834    2.78422   11.094 9.27e-12 ***
## hp          -0.03688    0.01452   -2.540  0.01693 *
## cyl         -1.12721    0.63417   -1.777  0.08636 .
## am           3.90428    1.29659    3.011  0.00546 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.807 on 28 degrees of freedom
## Multiple R-squared:  0.8041, Adjusted R-squared:  0.7831
## F-statistic: 38.32 on 3 and 28 DF,  p-value: 4.791e-10
```

# Regression II

```
# Access Fitted Values View first 3 predictions
```

```
reg_0$fitted.values[1:3]
```

```
##      Mazda RX4 Mazda RX4 Wag      Datsun 710  
##      23.97302      23.97302      26.85433
```

```
# Bonus: Are the residuals normally distributed
```

```
shapiro.test(reg_0$residuals)
```

```
##
```

```
##  Shapiro-Wilk normality test
```

```
##
```

```
## data:  reg_0$residuals
```

```
## W = 0.98366, p-value = 0.8961
```

# Regression III

```
# PREDICTING NEW DATA BASED ON MODEL
newCar <- mtcars[3, ] # 3rd observation is Datsun 710
newCar$am <- 0 # What if it was automatic?
reg_0$fitted.values[3] # Previous estimate

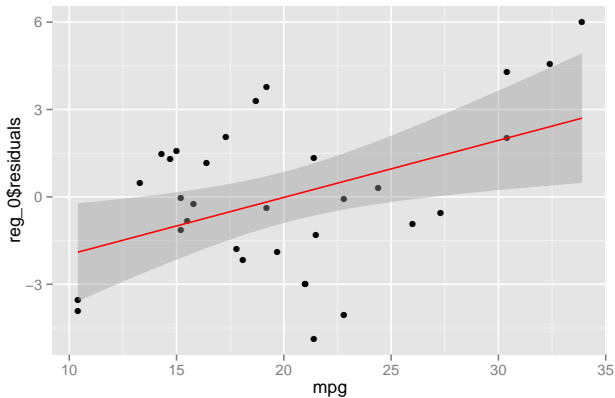
## Datsun 710
## 26.85433

predict(reg_0, newdata = newCar) # Datsun with automatic transmission

## Datsun 710
## 22.95005

## Plot the residuals against observation
qplot(data=mtcars, x = mpg, y = reg_0$residuals) + # $
  stat_smooth(method = "lm", col = "red")
```

# Regression IV



# Regression V

```
## COMPARE MODELS
reg_1 <- lm(mpg ~ hp + cyl + am + wt, mtcars) # add weight
anova(reg_0, reg_1)

## Analysis of Variance Table
##
## Model 1: mpg ~ hp + cyl + am
## Model 2: mpg ~ hp + cyl + am + wt
##   Res.Df    RSS Df Sum of Sq    F    Pr(>F)
## 1       28 220.55
## 2       27 170.00   1    50.555 8.0295 0.008603 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# AIC of the model
AIC(reg_0)

## [1] 162.5849

AIC(reg_1)

## [1] 156.2536
```

# Logistic Regression

Dependent variable will be type (binary).

It is basically a regression with a binomial link function.

## Formula

$$\log \left( \frac{p}{1-p} \right) = \beta_0 + \beta_1 x_1 + \epsilon$$

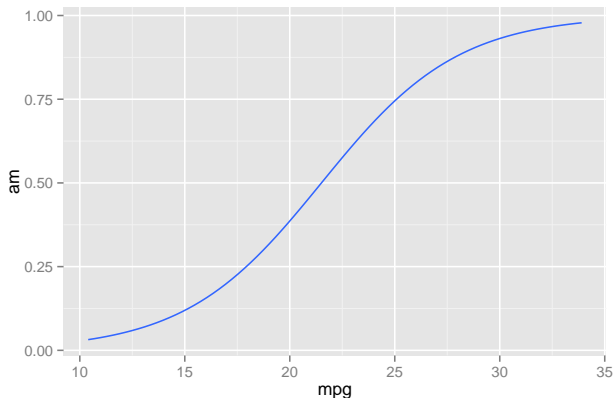
# Logit I

```
logit_2 <- glm(am ~ mpg + drat + cyl, data = mtcars, family = "binomial")
summary(logit_2)
```

```
##
## Call:
## glm(formula = am ~ mpg + drat + cyl, family = "binomial", data = mtcars)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.58367  -0.31020  -0.03757   0.17972   1.75395
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -49.4548    24.1280  -2.050   0.0404 *
## mpg           0.6378     0.4266   1.495   0.1349
## drat          7.2595     3.2702   2.220   0.0264 *
## cyl           1.6115     1.0801   1.492   0.1357
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 43.23  on 31  degrees of freedom
## Residual deviance: 17.03  on 28  degrees of freedom
## AIC: 25.03
##
## Number of Fisher Scoring iterations: 7
```

# Logit II

```
# VISUALIZE mpg - Transmission RELATION  
ggplot(mtcars, aes(x = mpg, y = am)) +  
  stat_smooth(method="glm", family="binomial", se=FALSE) #
```





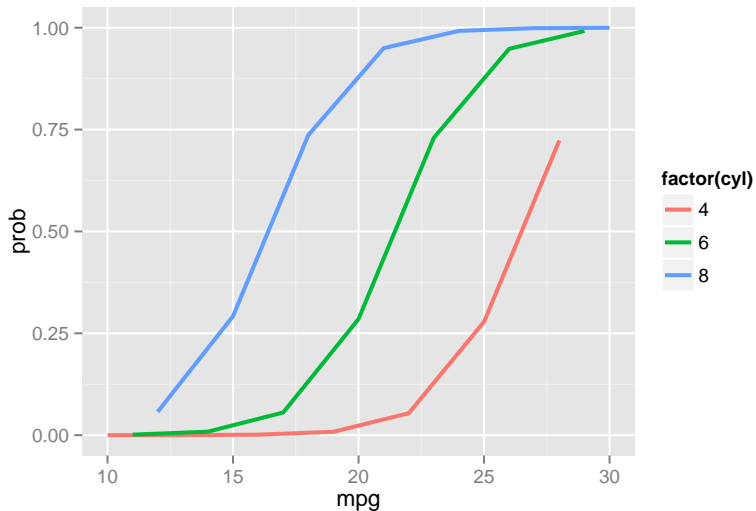
# Logit III

```
## VISUALIZE Mpg - Transmission FOR DIFFERENT NUMBERS OF CYLINDERS

# Create a new dataset with varying number of cylinders and other variables
# fixed at mean levels.
mtcars2 <- data.frame(mpg = rep(10:30, 3), drat = mean(mtcars$drat), disp = mean(mtcars$disp),
  cyl = rep(c(4, 6, 8), 21))
# Predict probability of new data
mtcars2$prob <- predict(logit_2, newdata = mtcars2, type = "response")

# Plot the results
ggplot(mtcars2, aes(x = mpg, y = prob)) + geom_line(aes(colour = factor(cyl)),
  size = 1) # a different color for each category$
```

# Logit IV



# Logit V

```
## DIAGNOSTICS

# Let us compare predicted values to real values
mtcars$prob <- predict(logit_2, type = "response")
# Prevalence of Manual Transmission
mean(mtcars$am)

## [1] 0.40625

# Create predict variable
mtcars$pred <- 0
# If probability is greater than .6 (1-prevalence), set prediction to 1
mtcars[mtcars$prob > 0.6, "pred"] <- 1
```

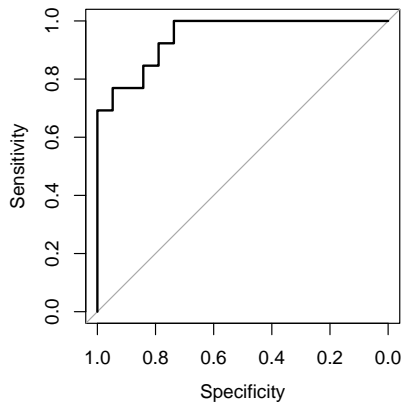
# Logit VI

```
## ROC CURVE

# Load the necessary library
library(pROC)
# Calculate the ROC curve using the predicted probability vs actual values
logit_2_roc <- roc(am ~ prob, mtcars)
# Plot ROC curve
plot(logit_2_roc)

##
## Call:
## roc.formula(formula = am ~ prob, data = mtcars)
##
## Data: prob in 19 controls (am 0) < 13 cases (am 1).
## Area under the curve: 0.9474
```

# Logit VII



# Logit VIII

```
library(caret) # Needed for Confusion Matrix
confusionMatrix(table(mtcars[, c("am", "pred"))))
```

```
## Confusion Matrix and Statistics
##
##      pred
## am  0  1
##    0 18  1
##    1  3 10
##
##               Accuracy : 0.875
##               95% CI : (0.7101, 0.9649)
##      No Information Rate : 0.6562
##      P-Value [Acc > NIR] : 0.005004
##
##               Kappa : 0.7344
##  Mcnemar's Test P-Value : 0.617075
##
##      Sensitivity : 0.8571
##      Specificity : 0.9091
##      Pos Pred Value : 0.9474
##      Neg Pred Value : 0.7692
##      Prevalence : 0.6562
##      Detection Rate : 0.5625
##      Detection Prevalence : 0.5938
##      Balanced Accuracy : 0.8831
##
##      'Positive' Class : 0
##
```

# caret Package

The Caret package is a wrapper that combines functionality from 27 R packages.

Functions Provided:

- Visualization
- Data Manipulation
- Model Training & Selection
- *Parallel Processing*

Since so many packages involved, the installation takes a while.

```
install.packages("caret", dependencies = c("Depends", "Suggests"))
```

For this part of the exercise I will focus on Caret Package, following its [vignette](#).

# What does caret Do?

Estimate model parameters

Tune variables through re-sampling strategies

Calculate performance



# Classification Trees

For this exercise I will be using Kuhn's 'Predictive Modeling with R and caret Package' examples.

We will fit a classification tree on segmentationData dataset.

Dataset is of segmentation of some cell images.

The data is 2019 observations of 61 variables.

# Classification Trees in R: Data I

```
# Obtain Dataset  
data(segmentationData)  
segmentationData <- segmentationData[, !(colnames(segmentationData) == "Cell")] #
```

```
# Data set has a variable that separates training vs testing  
Training <- segmentationData[segmentationData$Case == "Train", ] ## One Way  
Testing <- subset(segmentationData, Case == "Test") # Another way  
Training <- Training[, !(colnames(Training) == "Case")] # Drop Case column  
Testing <- subset(Testing, select = -c(Case)) # Another way
```

# Classification Trees in R I

```
library(rpart) # Load necessary library
rpart_1 <- rpart(Class ~ ., data = Training, control = rpart.control(maxdepth = 2))

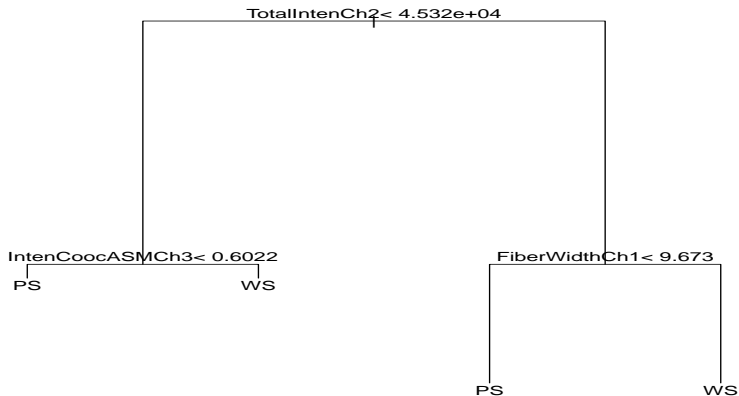
rpart_1 # View the results

## n= 1009
##
## node), split, n, loss, yval, (yprob)
##      * denotes terminal node
##
## 1) root 1009 373 PS (0.63032706 0.36967294)
##    2) TotalIntenCh2< 45324.5 454 34 PS (0.92511013 0.07488987)
##      4) IntenCoocASMCh3< 0.6021832 447 27 PS (0.93959732 0.06040268) *
##      5) IntenCoocASMCh3>=0.6021832 7 0 WS (0.00000000 1.00000000) *
##    3) TotalIntenCh2>=45324.5 555 216 WS (0.38918919 0.61081081)
##      6) FiberWidthCh1< 9.673245 154 47 PS (0.69480519 0.30519481) *
##      7) FiberWidthCh1>=9.673245 401 109 WS (0.27182045 0.72817955) *
```

# Classification Trees in R II

```
# Visualize the Tree  
plot(rpart_1)  
text(rpart_1)
```

# Classification Trees in R III



# Classification Trees in R IV

```
# Fit a larger tree and prune it rpart does 10 fold cross validation  
rpart_2 <- rpart(Class ~ ., data = Training)  
plot(rpart_2)  
text(rpart_2)
```

# Classification Trees in R V



# Pruning Trees with caret I

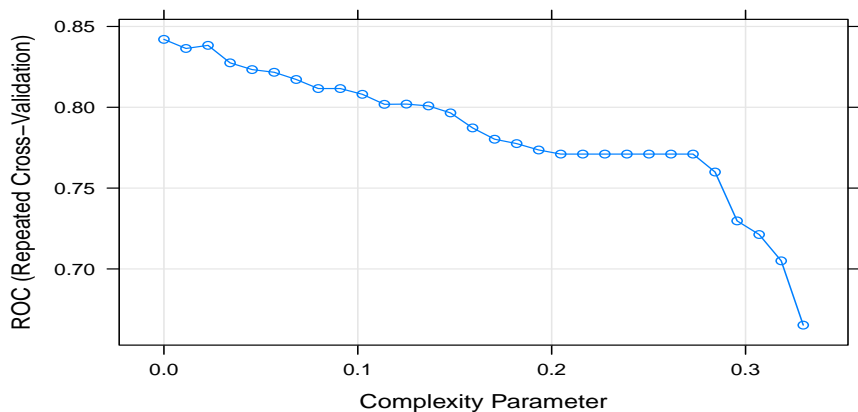
```
library(caret)
## Set Training Parameters Triple Cross Validation
cvCtrl <- trainControl(method = "repeatedcv", repeats = 3, summaryFunction = twoClassSummary,
  classProbs = TRUE)

# Use caret to fine tune the fit
CarrotTree <- train(Class ~ ., data = Training, method = "rpart", trControl = cvCtrl,
  metric = "ROC", tuneLength = 30)

plot(CarrotTree)
```



# Pruning Trees with caret II



# Pruning Trees with caret III

```
# Testing
CarrotTree_Test <- predict(CarrotTree, Testing)

confusionMatrix(CarrotTree_Test, Testing$class)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  PS   WS
##           PS 554 104
##           WS 110 242
##
##           Accuracy : 0.7881
##           95% CI : (0.7616, 0.8129)
##           No Information Rate : 0.6574
##           P-Value [Acc > NIR] : <2e-16
##
##           Kappa : 0.5316
##           Mcnemar's Test P-Value : 0.7325
##
##           Sensitivity : 0.8343
##           Specificity : 0.6994
##           Pos Pred Value : 0.8419
##           Neg Pred Value : 0.6875
##           Prevalence : 0.6574
##           Detection Rate : 0.5485
##           Detection Prevalence : 0.6515
##           Balanced Accuracy : 0.7669
##
##           'Positive' Class : PS
```

# Discriminant Analysis with Caret

Here we will conduct a PLS DA with caret

The example here closely follows the Caret Vignette example.

# Data Splitting I

```
## OBTAIN DATASET Dataset comes with mlbench package
library(mlbench)
# Load dataset into the current workspace
data(Sonar)
# 208 observations and 61 variables
```

```
## SPLIT THE DATA

# caret provides functionality
library(caret)
# Set random number seed for reproducibility
set.seed(107)
# Create an index of observations to be included in Training
indexTrain <- createDataPartition(y = Sonar$Class, p = 0.75, list = FALSE)

# $plit the data using the index
Train <- Sonar[indexTrain, ]
Test <- Sonar[-indexTrain, ]
```

# Train a PLS Discriminant Model I

```
## Declare Tuning Control Parameters
ctrl <- trainControl(method = "repeatedcv", # K-fold cross validation
                     repeats = 3, # Repeat resampling 3 times
                     classProbs = TRUE, # Calculate predicted prob for ROC
                     summaryFunction = twoClassSummary) # Set performance metrics for

## Train the Classifier
plsFit <- train(Class ~ ., data = Train, method = "pls",
               tuneLength = 10, # Number of component sets to be evaluated (more
               trControl = ctrl, # Use control parameters from above
               metric = "ROC", # Criteria ROC
               preProc = c("center", "scale")) # Center and scale the predictors
```

# Train a PLS Discriminant Model II

```

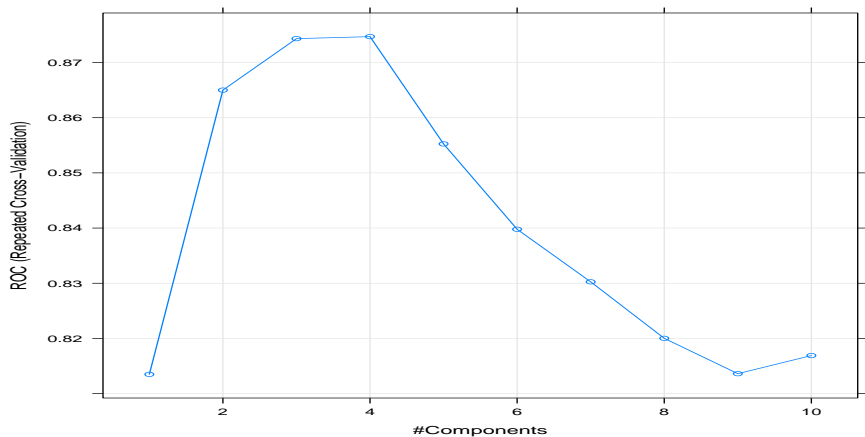
plsFit

## Partial Least Squares
##
## 157 samples
## 60 predictors
## 2 classes: 'M', 'R'
##
## Pre-processing: centered (60), scaled (60)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 142, 141, 142, 142, 142, ...
## Resampling results across tuning parameters:
##
##      ncomp  ROC      Sens      Spec      ROC SD      Sens SD      Spec SD
##      1      0.8134921  0.7291667  0.7291667  0.11844879  0.1387811  0.1935289
##      2      0.8649967  0.7694444  0.8041667  0.08381907  0.1416676  0.1521373
##      3      0.8743221  0.7476852  0.8363095  0.08548836  0.1726683  0.1375303
##      4      0.8746858  0.7578704  0.7642857  0.08443793  0.1512983  0.1539276
##      5      0.8552497  0.7152778  0.7767857  0.09587112  0.1771056  0.1584577
##      6      0.8397817  0.7337963  0.7732143  0.09814150  0.1726297  0.1749730
##      7      0.8302579  0.7101852  0.7916667  0.10762747  0.1992674  0.1655561
##      8      0.8200231  0.7157407  0.7607143  0.12724476  0.1805642  0.1621515
##      9      0.8136161  0.7245370  0.7696429  0.12961432  0.1847286  0.1593987
##     10      0.8168981  0.7203704  0.7559524  0.11678705  0.1912620  0.1436503
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was ncomp = 4.

```

# Train a PLS Discriminant Model III

```
# evaluate the performance of different number of components extracted  
plot(plsFit)
```



# Validate with Test Data I

```
plsPredict <- predict(plsFit, newdata = Test)  # Predict results  
head(plsPredict)  # View predictions
```

```
## [1] R M M R M R  
## Levels: M R
```

```
head(Test$Class)  # View actual$
```

```
## [1] R R R R R R  
## Levels: M R
```



# Validate with Test Data II

```
# Get confusion matrix (predicted vs actual)
confusionMatrix(data = plsPredict, Test$Class) ##

## Confusion Matrix and Statistics
##
##           Reference
## Prediction  M   R
##           M 17   6
##           R 10  18
##
##              Accuracy : 0.6863
##              95% CI : (0.5411, 0.8089)
##      No Information Rate : 0.5294
##      P-Value [Acc > NIR] : 0.01674
##
##              Kappa : 0.3761
##  Mcnemar's Test P-Value : 0.45325
##
##              Sensitivity : 0.6296
##              Specificity : 0.7500
##      Pos Pred Value : 0.7391
##      Neg Pred Value : 0.6429
##              Prevalence : 0.5294
##      Detection Rate : 0.3333
##      Detection Prevalence : 0.4510
##      Balanced Accuracy : 0.6898
##
##      'Positive' Class : M
##
```

# Read caret Documentations

I provide here a brief glimpse into what caret is capable of.

Do read for yourself.

# Questions



# Outline

- 1 Introduction
  - Familiar Examples
- 2 R Console
- 3 Importing Data
- 4 Packages
- 5 Sample Analysis and Visualizations
  - Data Manipulation
  - Descriptive Visualizations
  - Modeling
- 6 Reporting**
- 7 Where to Go Next?

# Markdown and R Markdown

Think HTML simplified.

Add (R) code into it.

Stir well with knitr

# R Markdown Cheat Sheet

# H1

## H2

### H3

*\*Italic\**

**\*\*Bold\*\***

***\*\*\* Bold Italic\*\*\****

```
``` { r, options }
```

```
R CODE HERE
```

```
```
```

# Professional Looking Documents

All the documentation of this workshop has been prepared in R Studio.

Look into the Rmd documents to see how this was done.

`summary()` function does not provide the best looking model summaries.  
Try the `texreg` package (do as I say, not as I do).

Use themes for your plots.

# Other Reporting Alternatives

Sweave:  $\text{\LaTeX}$

Beamer:  $\text{\LaTeX}$  presentations

Shiny: Web Applications



# Questions



# Outline

- 1 Introduction
  - Familiar Examples
- 2 R Console
- 3 Importing Data
- 4 Packages
- 5 Sample Analysis and Visualizations
  - Data Manipulation
  - Descriptive Visualizations
  - Modeling
- 6 Reporting
- 7 Where to Go Next?

# Quo Vadis?

8 hours is not enough!

RTFM<sup>2</sup>: Documentation is your friend, read the vignettes and manuals.

For tutorials: [Institute For Digital Research and Education, UCLA](#)

For questions: StackExchange<sup>3</sup>

- [Cross Validated](#) specializes on Statistics
- [Stack Overflow](#) for R programming

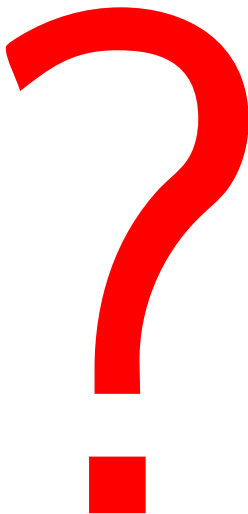
R Programming is quirky, learn about efficient R programming.

---

<sup>2</sup>Read the Friendly Manual

<sup>3</sup>Word of caution, [pay attention to how you ask your questions](#). It is more important than you realize!

# Questions



# License



How I Learned to Stop Worrying and Love the R Console by **Irfan E Kanat** is licensed under a **Creative Commons Attribution 4.0 International License**. Based on a work at <http://github.com/iekanat/rworkshop>.