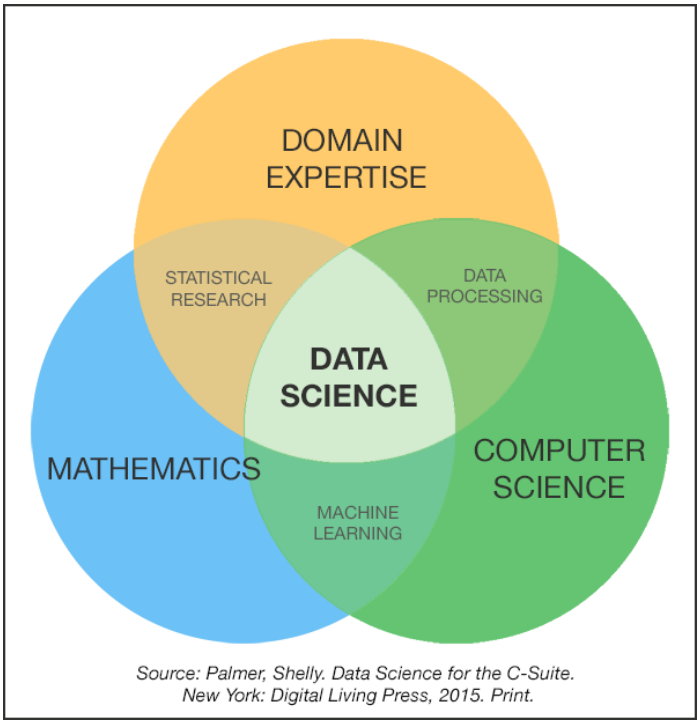
Computing for Data Science

Introduction to Data Science

- Data Scientist is the No. 1 job at Glassdoor
- Highly searched item - Google Trends
- Data Science can be applied to various fields - Image Processing, Speech Recognition, Medical Informatics, Business Processes
- Indeed.com - Steady rise in Data Scientist positions
- Harvard Business Review - sexist job of 21st Century

Why is this Explosion in Opportunities ?

1. More data than ever before
2. Large computing facility - Amazon EC2, Google Compute Engine
3. Programming tools - R, Python
4. Demand of skills is high

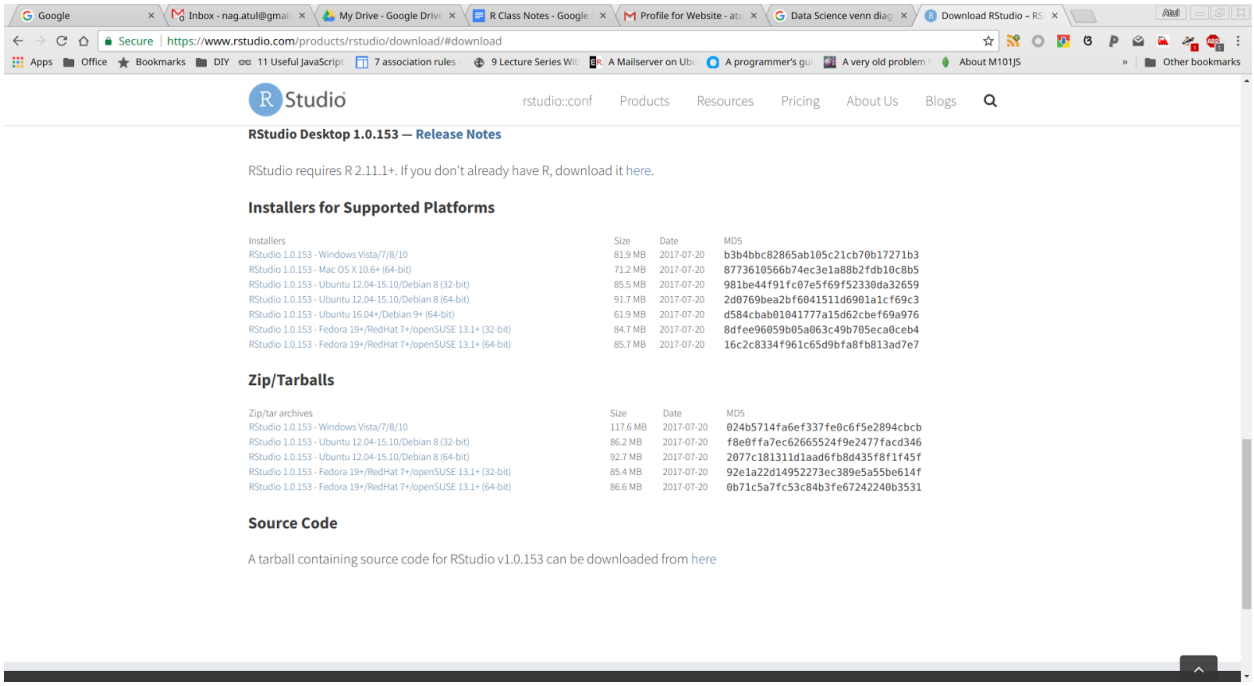


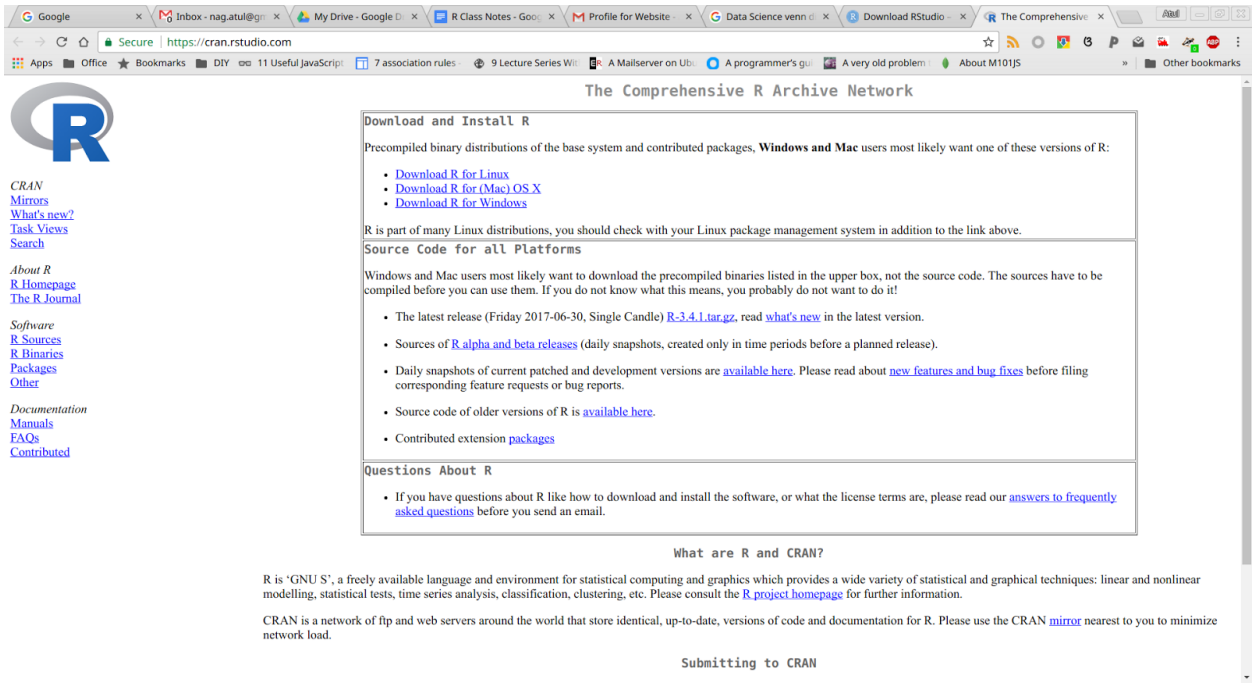
Source: Palmer, Shelly. Data Science for the C-Suite. New York: Digital Living Press, 2015. Print.

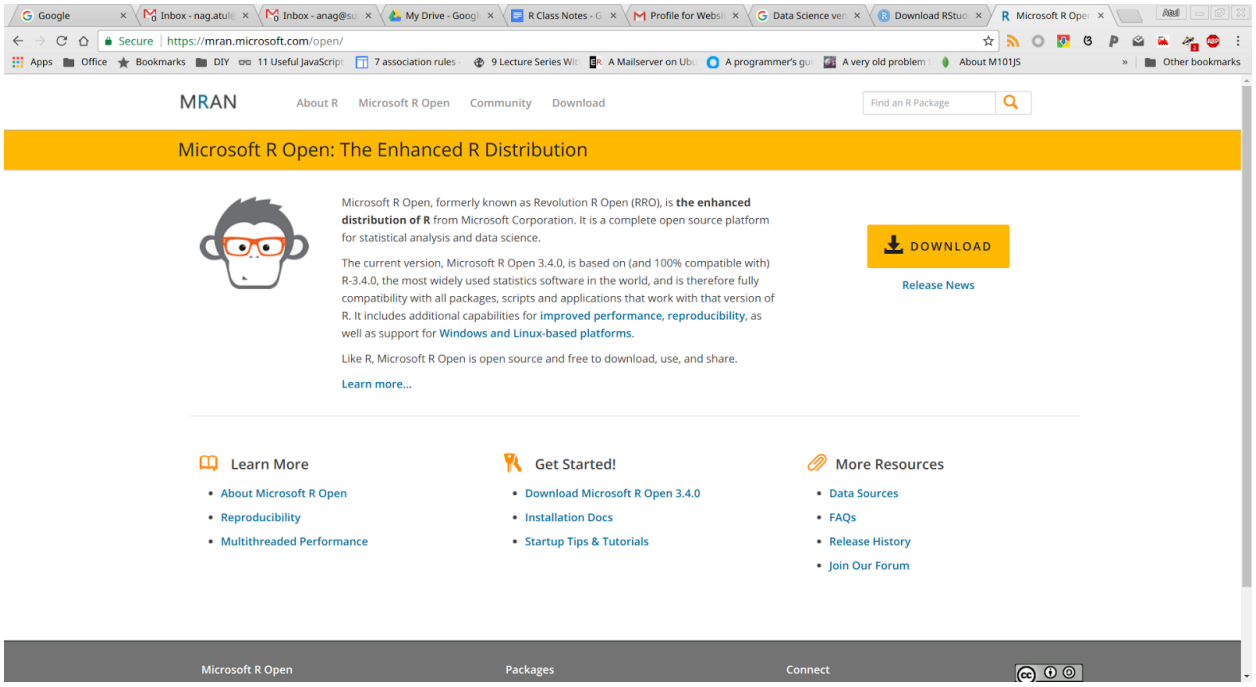
Developmental Setup

Windows

R Studio/R Download -



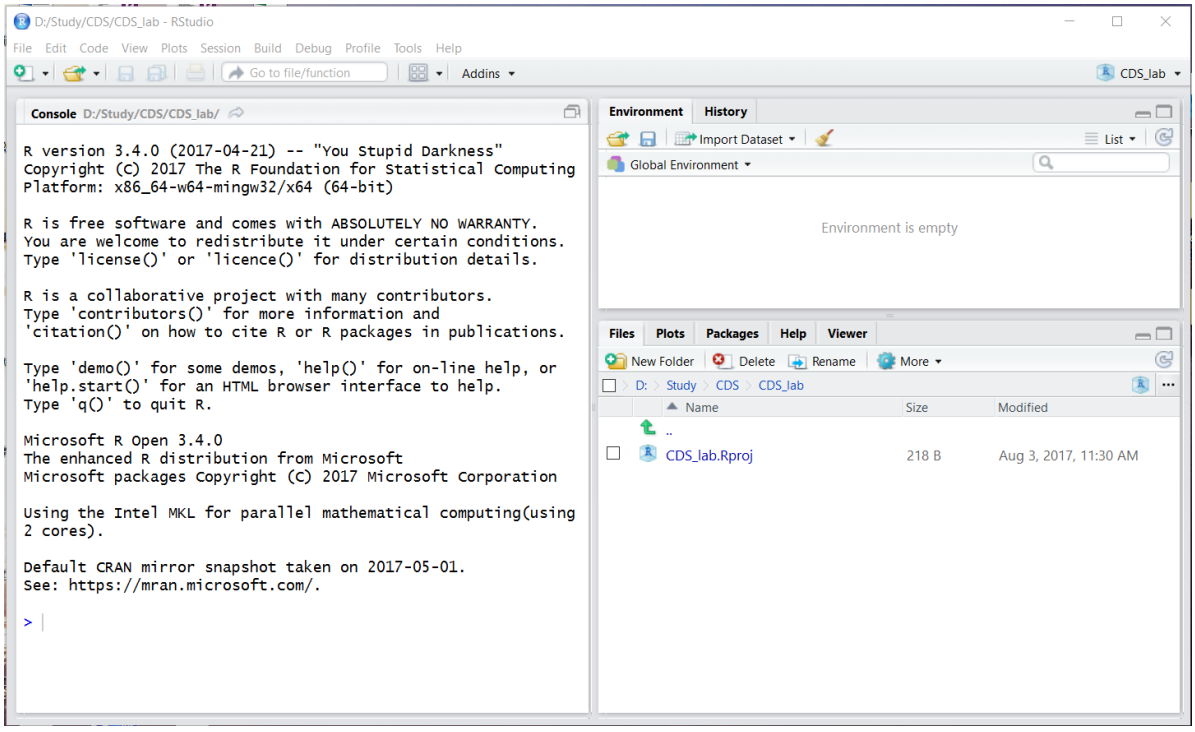




Linux (Ubuntu/Debian)

For R - `sudo apt install r-base r-base-dev`

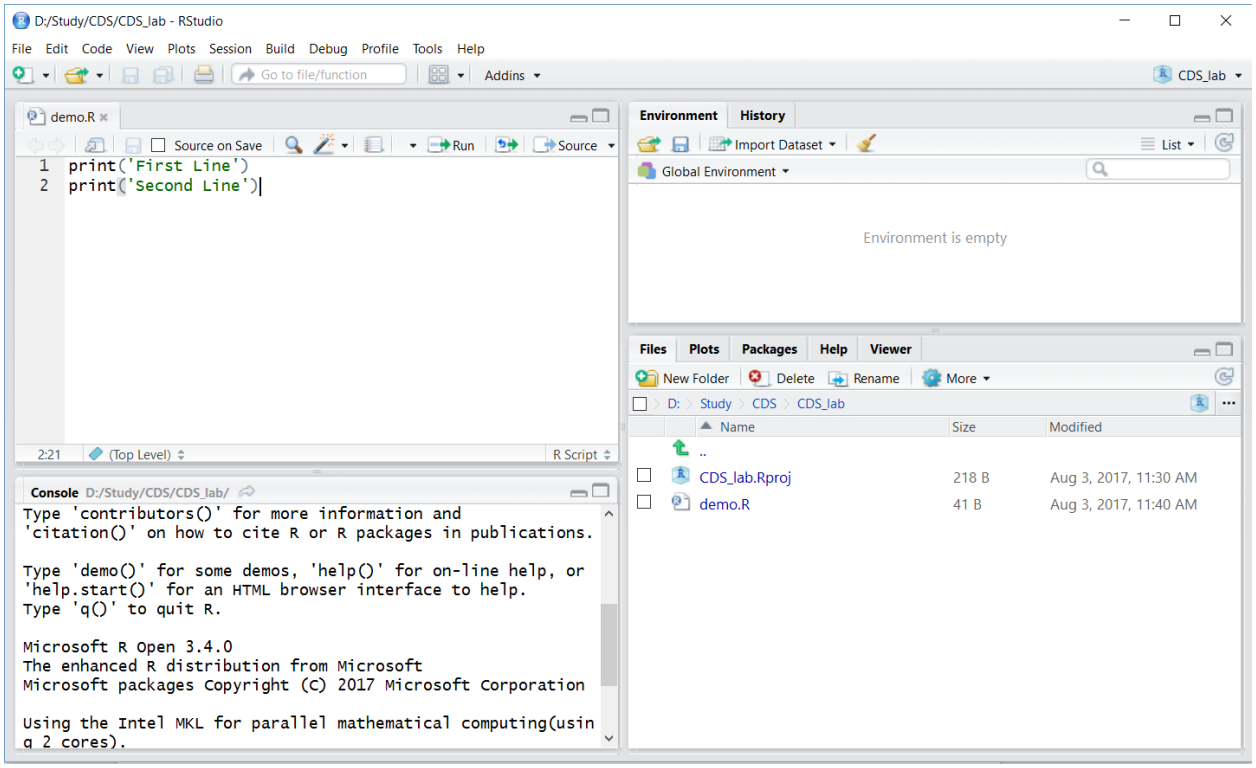
For R Studio download the .deb package from site.

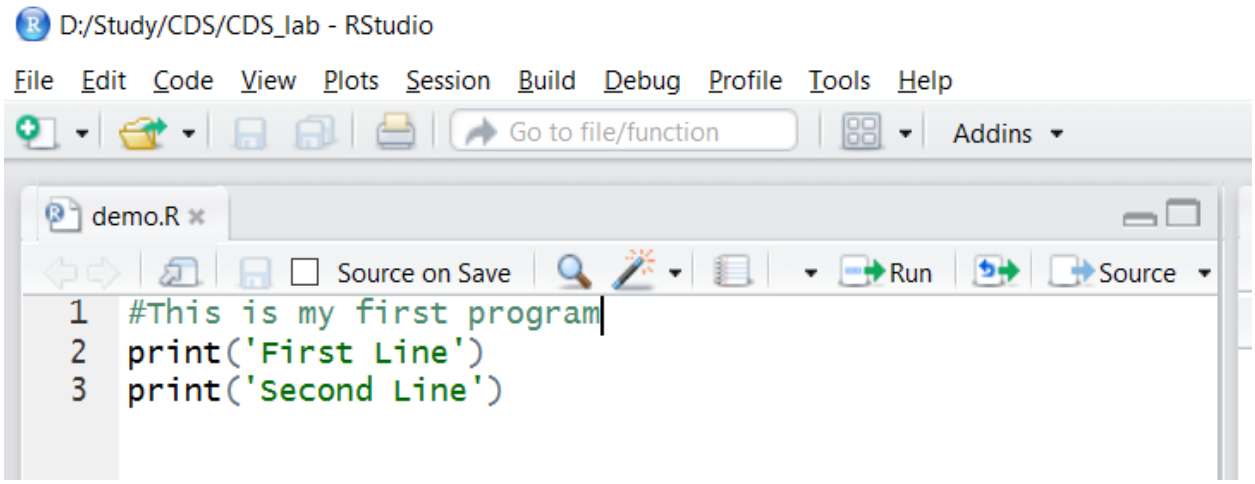


Print in the R console

```
print('Hello World')
O/P:[1] "Hello World"
Variable
a <- 2
a
getwd()
setwd("C:\\Users\\ASUS\\Desktop")
```

```
> getwd()
[1] "D:/Study/CDS/CDS_lab"
> setwd('C:\Users\ASUS\Desktop')
Error: '\U' used without hex digits in character string star
ting "'C:\U"
> setwd('C:\\Users\\ASUS\\Desktop')
> getwd()
[1] "C:/Users/ASUS/Desktop"
>
```




D:/Study/CDS/CDS_lab - RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
demo.R
Source on Save
Run
Source
1 #This is my first program
2 print('First Line')
3 print('Second Line')

R Basics

Arithmetic with R

Basic Math - Calculator
Addition, `+, 4 + 2`
Subtraction, `-, 4 - 2`
Division, `/, 4 / 2` (true division)
Exponent, `^, 2^3 - 8`
Reminder (modulus) , `%%, 5 %% 2` Output - `1`

Order of Operations (parentheses can be used to change the order) -
`100 * 2 + 50 / 2` Output - `225`

Comments - using #
```
# this is a comment
```

Variables

Convention is to use lower case letters
```
bank <- 1000
bank
```
For multiple worded variables
`bank.account <- 1000` (most preferable)
`bankAccount <- 1000` (camelcase - preferable)
`bank_account <- 1000` (Not preferable)

R Data Types

R has 3 generic data types

1) Numeric - decimal/floating point values/Integers
   ```
   a <- 2.2
   b <- 2
   ```
2) Logical - All Caps `TRUE (T)` or `FALSE (F)`
   ```
   a <- TRUE
   b <- FALSE
   ```
3) Character - enclosed by ‘ ’ or “ ”
   ```
   a <- “hello”
   ```

Function to detect type of variable - `class`
```
class(a)
```

Vectors Basics

- One dimensional arrays
- Can be created using combine function - `c`

```
nvec <- c(1, 2, 3, 4)
nvec
class(nvec)
cvec <- c(“I”, “N”, “D”, “I”, “A”)
cvec
lvec <- c(T, T, F, F)
lvec
```

Vectors cannot mix data types. In case they are mixed the get converted to a single type

```
v <- c(T, 10, 20) # Mix of logical with numeric
V
```

T gets converted to 1

```
x <- c(“India”, 10, 20) # Mix of character with numeric
x
```

`10` and `20` get converted to character type

Vector Operations

```
v1 <-  c(1,2,3)
v2 <- c(5,6,7)
```

Element by element operation

```
v1 + v2
v1 - v2
v1 * v2
v1 / v2
```

Built-in functions to work on vectors

```
sum(v1)
mean(v1)
sd(v1)
max(v1)
min(v1)
prod(v1)  # Product of elements
```

Vector indexing/slicing

```
v1 <- c(100, 200,300)
```

```
v2 <- c(‘a’, ‘b’, ‘c’)
```

Indexing starts at 1

```
v1[1]
v1[2]

v2[2]

v2[c(1,2)]

v3 <- c(1,2,3,4,5,6,7,8,9,10)
v3[2:4]
v3[7:10]

v <- c(1, 2, 3, 4)
names(v) <- c(‘a’, ‘b’, ‘c’, ‘d’)

v[2] is same as v[‘b’]
v[c(‘c’, ‘d’, ‘a’)]

days <- c(“Mon”, “Tue”, “Wed”, “Thu”, “Fri”, “Sat”)
temp <- c(76, 77, 78, 79, 80, 81)

names(temp) <- days
temp
temp[‘Mon’]
```

Help with R

```
help('vector')
??vector
help.search(“vectors”)
```

Comparison operators

>, <, >=, <=, ==, !=

```
v <- c(1, 2, 3, 4, 5)
v < 2
```
This gives a boolean vector containing TRUE/FALSE for each element of the vector

```
v == 3
```

Comparison using boolean or logical operators

```
#All values greater than 2
v[v > 2]
```

Compare two vectors

```
v2 <- c( 2, 2, 3, 5, 4)
```

R Matrices

- 2D objects
- Stepping stone for data frames

Creating Matrices

```
v <- 1:10          #sequential vector contains integers 1 to 10
matrix(v, nrow = 2)          #nrow - number of rows
matrix(1:12, byrow = F, nrow = 4)
matrix(1:12, byrow = T, nrow = 4)

goog <- c(450, 451, 452, 445, 468)
msft <- c(230, 231, 232, 233, 220)

stocks <- c(goog, msft)
stocks.matrix <- matrix(stocks, byrow = T, nrow = 2)

days <- c("Mon", "Tue", "Wed", "Thu", "Fri")
stock.names <- c('GOOG', 'MSFT')

colnames(stocks.matrix) <- days
rownames(stocks.matrix) <- stock.names

print(stocks.matrix)
```

Matrix Arithmetic

```
mat <- matrix(1:25, byrow = T, nrow = 5)
```

Scalar with matrix

```
mat * 2
mat / 2
mat ^ 2
1 / mat
mat > 15          #boolean matrix
mat[mat > 15]     #Vector of values satisfying the condition
```

Matrix with Matrix

```
mat + mat
mat / mat
```

```
mat * mat          #Not true matrix multiplication but each element against element
```

For true matrix multiplication
```
mat %*% mat
```

Matrix Operations

```
stocks.matrix

colSums(stocks.matrix)
rowSums(stocks.matrix)
rowMeans(stocks.matrix)
colMeans(stocks.matrix)
```

Adding rows & columns to Matrices

```
cbind - bind new columns
rbind - bind new rows

FB <- c(111, 112, 113, 120, 145)
tech.stocks <- rbind(stocks.matrix,FB)
avg <- rowMeans(tech.stocks)
tech.stocks <- cbind(tech.stocks, avg)
print(tech.stocks)
```

Matrix selection & indexing

```
mat <- matrix(1:50, byrow = T, nrow = 5)
```

Format: `mat[`*`row`*`, `*`column`*`]`

First Row `- mat[1,]`
First Column `- mat[,1]`
3 rows `- mat[1:3,]`
```
mat[1:2, 1:3]
```

Factor & Categorical Matrices

```
factor()

animals <- c('d','c','d','c','c')
factor(animals)     #Displays Levels: c d
```

Categorical Variables are of two types -

1. Ordinal (having order)

2. Nominal(No order)

Temperature has order, hence ordinal variable

```
ord.cat <- c('cold', 'med', 'hot')
temps <- c('cold', 'med', 'hot', 'hot', 'hot', 'cold', 'med')
summary(temps)
fact.temps <- factor(temps, ordered = T, levels = ord.cat)
print(fact.temps)
summary(fact.temps)
```

R data frames

Matrix , Vectors contain same data types
Data Frames can mix data types
Data frames provided labelled rows & columns (Like excel sheet)

R has various inbuilt datasets (via datasets package). Can be viewed by - `data()`

```
mtcars    # View mtcars dataset
head(mtcars)   # View first 6 rows
head(mtcars, 7) # View first 7 rows
tail(mtcars)   # View last 6 rows
str(mtcars)    # View structure of the data frame
summary(mtcars)   # Statistical Summary of columns
```

Creating data frames

```
days <- c('Mon', 'Tue', 'Wed', 'Thu', 'Fri')
temps <- c(22.2, 21, 23, 24.3, 25)
rain <- c(T, T, F, F, T)

df <- data.frame(days, temps, rain)
str(df)
summary(df)
```

Data frame selection & Indexing

```
df[1, ]      #retrieves 1st row as data frame
df[, 1]      #retrieves 1st column as data frame

df[-2, ]   # -ve sign to select everything but a particular row

df[,'rain']  #Retrieve using column name
df[1:3, c('days', 'temps')]  #days and temps values for First 3 rows
as data frame

df$days    #All days as vectors
df$temps   #All temps as vectors
```

Subset function returns data frames based on condition

```
subset(df, subset = rain == TRUE)  #All rows containing rain as TRUE
subset(df, subset = temps > 23)   #All rows where temps > 23
```

Sorting data frames

```
sorted.temps <- order(df['temps'])     # sorts temps column
sorted.temps
df[sorted.temps, ]  #returns dataframe sorted by temps

desc.temps <- order(-df['temps'])   # sort in descending order
df[desc.temps, ]
```

Data Frame Operations

```
empty <- data.frame() # Creates empty data frame

c1 <- 1:10
letters    #built -in vector a - z

c2 <- letters[1:10]
df <- data.frame(col.name.1 = c1, col.name.2 = c2)
df
```

Import/Export CSV

```
d2 <- read.csv(file = 'test.csv', header = T)
write.csv(d2, file = 'saved_test.csv')
df2 <- read.csv('saved_test.csv')
df2
```

Data frame Information

```
nrow(df)
ncol(df)
colnames(df)  #Vector of column names
rownames(df)  #Vector of row names
str(df)   #Number of observations and type of variables
summary(df)  #Statistical summary
```

Referencing Cells

```
df[[row, column]]
df[[5, 2]]   #retrieve value at 5th row 2nd column
df[[5, 'col.name.2']]  #Does the same thing
```

```
df[[2, 'col.name.1']] <- 99  #Change the value
df

mtcars
head(mtcars)
mtcars$mpg      # retrieve mpg column
mtcars[,'mpg']  #Same thing
mtcars[,1]      #Same thing
mtcars[['mpg']] #Same thing

mtcars['mpg']  #Returns data frame
mtcars[1]      #same thing

head(mtcars[ c('mpg', 'cyl')])  #Multiple columns as data frame
```

Adding rows/columns to dataframe

```
df2 <- data.frame(col.name.1 = 2000, col.name.2 = 'new')
df2
df.new <- rbind(df,df2)
df.new
```

Add columns

```
df$newcol <- 2 * df$col.name.1

df$newcol.copy <- df$newcol  #1st way to add column
df[, 'newcol.copy2'] <- df$newcol #2nd way to add column
```

Setting column names

```
colnames(df) <- c(1:5)  #Column names are 1 through 5
colnames(df)[1] <- 'New col name'  #Individual column named
```

Conditional Selection

```
mtcars[mtcars$mpg > 20,7 ]
mtcars[mtcars$mpg > 20 & mtcars$cyl == 6 ]
mtcars[mtcars$mpg > 20 & mtcars$cyl == 6, c('mpg', 'cyl', 'hp')]
subset(mtcars, mpg > 20 & cyl == 6)
```

Dealing with Missing Data

NA indicates Null or missing data

```
is.na(mtcars)   #returns matrix of boolean values indicating missing
values
                # returns TRUE for missing values else FALSE
any(is.na(mtcars))  #TRUE if any missing values present else FALSE
any(is.na(mtcars$mpg))  #Look for NA in a particular column
```

Replace Missing values

```
df[is.na(df)] <- 0    #Replace all NA with 0

#Replace missing values with the mean of the column
mtcars$mpg[is.na(mtcars$mpg)] <- mean(mtcars$mpg)
```

R Lists

- Variety of data structures in a single variable
- Mainly used for organizing variables/data frames

```
v <- c(1,2,3)
m <- matrix(1:10, nrow = 2)
df <- mtcars

my.list <- list(v, m, df)
my.name.list <- list(sample.vec = v, my.matrix = m, sample.df = df)


#Pull out values
my.name.list$sample.df  #Returns variable in this case a data frame
my.list[3]      #Returns list
my.name.list['sample.vec']  #Returns List
my.name.list[['sample.vec']] #Returns variable - vector

#Combine lists
double.list <- c(my.name.list, my.name.list)
str(double.list)
```

Data Input/Output with R

CSV files with R

Read/Write CSV files in R

```
write.csv(mtcars, file = "sample.csv")

#Reading a CSV file
my.df <- read.csv('sample.csv')
# Checking import
head(my.df)
tail(my.df)
str(my.df)
```

The read.table function is the general form of read.csv, in fact read.csv is actually just a thin wrapper around read.table which just makes it easier to use sometimes.

```
read.table('example.csv')

read.table(file = 'example.csv', sep = ',')
```

fread() is similar to read.table but faster and more convenient:

```
fread('example.csv')
```

Output to CSV

```
write.csv(df, file = "foo.csv")
fread('foo.csv')

write.csv(df, file = "foo.csv",row.names = FALSE)
fread('foo.csv')
```

Excel files in R

R has the ability to read and write to excel.

```
install.packages("readxl")  #Install package readxl to read Excel
files

library(readxl)  #Load readxl library

excel_sheets('Sample-Sales-Data.xlsx')  #View all the sheets in the
excel workbook
```

```
df <- read_excel('Sample-Sales-Data.xlsx', sheet = 'Sheet1') #Read
sheet into data frame

#usual operations on data frame
head(df)
str(df)
summary(df)

#To import multiple sheets
entire_workbook <- lapply(excel_sheets("Sample-Sales-Data.xlsx"),
read_excel,
                          path = 'Sample-Sales-Data.xlsx')
entire_workbook  #Display entire sheets


install.packages('xlsx')   #Install module to write to excel
library(xlsx)  #Load xlsx library

df <- mtcars
write.xlsx(df, "output.xlsx")

read_excel('output.xlsx')
```

SQL with R

connecting R to a SQL database is completely dependent on the type of database you are using (MYSQL, Oracle, etc...).

The RODBC library is one way of connecting to databases.
Recommended method is to google search consisting of your database of choice + R.
Here's an example use of RODBC

```
install.packages("RODBC")
# RODBC Example of syntax
library(RODBC)

myconn <-odbcConnect("Database_Name", uid="User_ID", pwd="password")
dat <- sqlFetch(myconn, "Table_Name")
querydat <- sqlQuery(myconn, "SELECT * FROM table")
close(myconn)
```

For MySQL - RMySQL, Oracle - ROracle, JDBC - RJDBC

Web Scraping using R

Using R to extract web pages using URLs.(Should understand HTML & CSS).
If you don't know HTML or CSS, you may be able to use an auto-web-scrape tool, like import.io.
Check it out, it will auto scrape and create a csv file for you.
10 Web Scraping tools

```
install.packages('rvest')   #Web scraping package
demo(package = 'rvest')  #Demo packages in rvest
demo(package = 'rvest',topic = 'tripadvisor') #Load tripadvisor demo
in rvest
```

Programming with R

Logical Operators

Logical operators allows for combining multiple comparison operators.

- AND (&)
- OR(|)
- NOT(!)

```
x <- 10

x < 20 #TRUE
x > 5  #TRUE
x < 20 & x > 5 #TRUE
x < 20 & x > 5 & x == 10  #TRUE

(x < 20) & (x > 5) & (x == 10)  #TRUE Brackets to enhance readability

x == 2 & x > 1  #FALSE
x == 2 | x > 1  #TRUE
x == 1 | x == 12 #FALSE


df <- mtcars   #Dataframe of mtcars dataset
df
df[df['mpg'] >= 20, ]  #Models with greater than or equal to 20 mpg
subset(df, mpg >= 20)  #using subset function
df[(df['mpg'] >= 20) & (df['hp'] > 100),]  #rows with cars of at least
20mpg and over 100 hp
```

Logical Operators with vectors

Two options when use logical operators, a comparison of the entire vectors element by element, or just a comparison of the first elements in the vectors, to make sure the output is a single

```
# Boolean vectors
tf <- c(T,F)
tt <- c(T,T)
ft <- c(F,T)

tt & tf  # Element by Element comparison
tt | tf  # Element by Element comparison

#Compare only first elements
```

```
ft && tt  #FALSE
tt && tf  #TRUE
tt || tf  #TRUE
tt || ft  #TRUE
```

If, else & else if statements

Adding logic to code.
Here is the syntax for an **if** statement in R:

```
if (condition){
    # Execute some code
}
```

```
hot <- F
temp <- 60

if (temp > 80){
  hot <- T
}
hot  #FALSE

temp <- 100
if (temp > 80){
  hot <- T
}
hot  #TRUE
```

If we want to execute another block that occurs if the **if** statement is false, we can use an **else** statement to do this! It has the syntax:

```
if (condition) {
  # Code to execute if true
} else {
  # Code to execute if above was not true
}
```

```
temp <- 30

if (temp > 90){
  print("Hot outside!!")
} else {
  print("It's not hot today")
}
```

we can use the **else if** statement to add multiple condition checks, using **else** at the end to execute code if none of our conditions match up with and if or else if.

```
temp <- 30

if (temp > 80){
  print("Hot outside!")
} else if(temp< 80 & temp > 50){
  print('Nice outside!')
} else if(temp < 50 & temp > 32){
  print("It’s cooler outside!")
} else{
  print("It’s really cold outside!")
}
```

While loops

while loops are a while to have your program continuously run some block of code until a condition is met (made TRUE). The syntax is:

```
while (condition){
    # Code executed here
    # while condition is true
}

x <- 0

while(x < 10){

  cat('x is currently: ',x)  #To concatenate & print
  print(' x is still less than 10, adding 1 to x')

  # add one to x
  x <- x+1
}
```

```
x <- 0

while(x < 10){

  cat('x is currently: ',x)
  print(' x is still less than 10, adding 1 to x')

  # add one to x
  x <- x+1
  if(x==10){
    print("x is equal to 10! Terminating loop")
  }
}
```

break statement

You can use **break** to break out of a loop.

```
x <- 0

while(x < 10){

  cat('x is currently: ',x)
  print(' x is still less than 10, adding 1 to x')

  # add one to x
  x <- x+1
  if(x==10){
    print("x is equal to 10!")
    break
    print("I will also print, woohoo!")
  }
}
```

For loops

A **for loop** allows us to iterate over an object (such as a vector) and we can then perform and execute blocks of codes *for* every loop we go through. The syntax for a for loop is:

```
for (temporary_variable in object){
    # Execute some code at every loop
}
```

```
vec <- c(1, 2, 3, 4, 5)

#Looping over vector
for(temp_var in vec){
  print(temp_var)
}

for(i in 1:length(vec)){
  print(vec[i])
}


li <- list(1, 2, 3, 4, 5)

#Looping over list
for(temp_var in li){
  print(temp_var)
}

for (i in 1:length(li)){
  print(li[[i]])
}

mat <- matrix(1:25, nrow = 5)
mat

#looping over matrix
for (num in mat){
  print(num)
}


#Nested for
for (row in 1:nrow(mat)){
  for (col in 1:ncol(mat)){
    print(paste('row:',row,' col:',col,': ',mat[row,col]))
  }
}
```

Functions in R

A function is a useful device that groups together a set of statements so they can be run more than once. They can also let us specify parameters that can serve as inputs to the functions. functions allow us to not have to repeatedly write the same code again and again.

We already have seen built-in functions
Here is the syntax for writing your own function:

```
name_of_function <- function(arg1,arg2,...){
               # Code that gets executed when function is called
}

name_of_function(input1,input2,....)

# Simple function, no inputs!
hello <- function(){
  print('hello!')
}
hello()

helloyou <- function(name){
  print(paste('hello ',name))
}
helloyou('Sammy')

add_num <- function(num1,num2){
  print(num1+num2)
}
add_num(5,10)

hello_someone <- function(name='Frankie'){
  print(paste('Hello ',name))
}
hello_someone()  #Uses default values
hello_someone('Sammy')  #Overwrite default values

formal <- function(name='Sam',title='Sir'){
  return(paste(title,' ',name))
}
formal()
formal('Isaac Newton')
```

```
var <- formal('Marie Curie','Ms.')
var
```

Scope

Scope is the term we use to describe how objects and variable get defined within R.

```
# Power to 2
pow_two <- function(input) {
  result <- input ^ 2
  return(result)
}

pow_two(4)
result    #error
input     #error

v <- "I'm global v"
stuff <- "I'm global stuff"

fun <- function(stuff){
  print(v)
  stuff <- 'Reassign stuff inside func'
  print(stuff)
}

print(v) #print global v
print(stuff) #print global stuff
fun(stuff) # pass stuff to function
# reassignment only happens in scope of function
print(stuff)

double <- function(a) {
  a <- 2*a
  a
}
var <- 5
double(var)
var
```

Advanced Programming in R

Built-in R Features

R contains quite a few useful built-in functions to work with data structures. Here are some of the key functions to know:

`seq()`: Create sequences
`sort()`: Sort a vector
`rev()`: Reverse elements in object
`str()`: Show the structure of an object
`append()`: Merge objects together (works on vectors and lists)

```
#Sequence of numbers
#seq(start, end, step size)
seq(0, 100, by = 3)

v <- c(1, 4, 6, 7, 2, 13, 2)
sort(v)  #Ascending order sort
sort(v, decreasing = T) # Descending order sort

rev(v) #Reverse vector

v2 <- c(1, 2, 3, 4, 5)

str(v2)  #Structure

append(v, v2) #Add elements of v2 of to v

sort(append(v, v2))
```

Data type Check & Convert

- `is.*()`: Check the class of an R object
- `as.*()`: Convert R objects

```
v <- c(1,2,3)
is.vector(v) #Check if it is a vector
is.list(v) #Check if it is a list

as.list(v) #Convert to list
as.matrix(v) #Convert to matrix
```

Apply Functions

learn about 3 different apply() functions. The basic idea of an apply() is to apply a function over some iterable object.

lapply()

lapply() will apply a function over a list or vector:
`lapply(X, FUN, ...)`
where X is your list/vector and FUN is your function
lapply returns a list of the same length as X, each element of which is the result of applying FUN to the corresponding element of X.

```
#Sample 1 random number between 1 to 10
sample(x = 1:10, 1)

v <- 1:5

addrand <- function(x){
  ran <- sample(x = 1:10, 1) #Random number between 1 to 10

  return(ran + x) #Return x + random number
}

lapply(v,addrand)  #returns list
sapply(v,addrand)  #returns vector
```

Anonymous functions

create an anonymous function (called so because it is never named). Generally used if you want to use a function just once, hence need not define it.

syntax for an anonymous function in R:
`function(a){code here}`

```
#Anon function with lapply
lapply(v, function(a){a + sample(x = 1:10, 1)})

#Anon function to add 2 to every element
sapply(v, function(x){x + 2})
```

Handling multiple arguments

```
add_choice <- function(num, choice){
  return(num + choice)
}

add_choice(2,3)

sapply(v, add_choice)  #Error as function has multiple arguments

sapply(v, add_choice, choice = 10) #Pass other parameters as arguments
```

Limitations of sapply()

sapply() won't be able to automatically return a vector if applied function doesn't return something for all elements in that vector.

```
#Checks for even numbers
even <- function(x){
  return(x[x %% 2 == 0])
}

nums <- 1:5

sapply(nums,even) #Does not return values for all elements
```

Other apply functions

Check out StackOverflow

Math Functions in R

```
v <- -1:5
abs(-2)  #Absolute value
abs(v)   #Applying over function

sum(v) #Sum of all elements

mean(v) #Arithmetic mean of all elements

round(23.1231)
round(23.123134, 2)
```

For more functions use R Reference Card

Pattern Matching in R (Regular Expressions)

Regular expressions is a general term which covers the idea of pattern searching, typically in a string (or a vector of strings).
For now learn about two useful functions for regular expressions and pattern searching:

- grepl(), which returns a logical indicating if the pattern was found
- grep(), which returns a vector of index locations of matching pattern instances

For both of these functions pass in a pattern and then the object to search:

```
text <- "Hi there, do you know who I am "

grepl('there', text) #Returns logical

grepl('Atul', text)

v <- letters[1:5]

grep('a', v) #returns location of a
grep('c', v)
```

Date & Timestamp

Dates

use the as.Date() function to convert a character string to a Date object, which will allow it to contain more time information. The string will need to be in a standard time format.

Code	Value
%d	Day of the month (decimal number)
%m	Month (decimal number)
%b	Month (abbreviated)
%B	Month (full name)
%y	Year (2 digit)
%Y	Year (4 digit)

```
Sys.Date()  #Current Date

today <- Sys.Date()  #Store as variable
today

as.Date('2010-08-01')  #convert string to Date Object

as.Date("Aug-01-10", format = "%b-%d-%y")  #Convert from String to
Date using format

as.Date("August-01-2010", format = "%B-%d-%Y")
```

Time

We can also convert strings and work with them for time information. R uses a POSIXct object type to store time information. POSIX represents a portable operating system interface, primarily for UNIX systems, but available on other operating systems as well. We can use as.POSIXct() for converting string to a POSIXct object type for time series analysis

```
as.POSIXct("11:02:03", format="%H:%M:%S") #converts string to time
with warning

as.POSIXct("August-01-2010 11:02:03", format="%B-%d-%Y %H:%M:%S")
```

actually be using the **strptime()** function, instead of POSIXct

```
strptime("11:02:03", format = "%H:%M:%S")
```

Data Manipulation using dplyr & tidyr

Using the dplyr package

dplyr package by Hadley Wickham for data manipulation. Functions of dplyr covered -

- filter() (and slice())
- arrange()
- select() (and rename())
- distinct()
- mutate() (and transmute())
- summarise()
- sample_n() and sample_frac()

Installation of package & example data

```
install.packages("dplyr")    #Install dplyr
install.packages("nycflights13")  #Example Dataset

#Use the installed libraries
library(dplyr)
library(nycflights13)

#Explore nycflights
str(flights)
summary(flights)

#Dimensions of the dataset
dim(flights)
```

filter()

filter() allows you to select a subset of rows in a data frame. The first argument is the name of the data frame. The second and subsequent arguments are the expressions that filter the data frame:

select all flights on November 3rd that were from American Airlines (AA)

```
#Using dataframe
head(flights[flights$month == 11 & flights$day == 3 &
               flights$carrier == 'AA', ])
```

```
#Select subset of rows on condition
head(filter(flights,month == 11, day ==3, carrier=='AA'))
```

slice()

select rows by position using **slice()**

```
#Select rows by position
slice(flights, 10:20)
```

arrange()

arrange() works similarly to filter() except that instead of filtering or selecting rows, it reorders them. It takes a data frame, and a set of column names (or more complicated expressions) to order by. If you provide more than one column name, each additional column will be used to break ties in the values of preceding columns:

```
#reorder rows & columns
head(arrange(flights,year,month,day,arr_time)) #Default ascending

head(arrange(flights,desc(dep_delay))) #In descending order
```

select()

select() allows you to rapidly zoom in on a useful subset using operations that usually only work on numeric variable positions:

```
#select particular subset
head(select(flights, carrier))
```

rename()

rename() to rename columns, (not "in-place") need to reassign the renamed data structures.

```
#Rename columns
head(rename(flights, airline_car = carrier))
```

distinct()

A common use of select() is to find the values of a set of variables. This is particularly useful in conjunction with the distinct() verb which only returns the unique values in a table.

```
#To return unique values in dataset
distinct(select(flights, carrier))
```

mutate()

Besides selecting sets of existing columns, it's often useful to add new columns that are functions of existing columns. This is the job of mutate():

```
#To add new columns that are functions of existing columns
head(mutate(flights, new_col = arr_delay - dep_delay))
```

transmute()

Use transmute if you only want the new columns:

```
#For new columns only
head(transmute(flights, new_col = arr_delay - dep_delay))
```

summerise()

You can use summarise() to quickly collapse data frames into single rows using functions that aggregate results. Remember to use na.rm=TRUE to remove NA values.

```
#Aggregate or summarize results
summarise(flights, avg_air_time=mean(air_time,na.rm = T))
```

sample_n() and sample_frac()

You can use sample_n() and sample_frac() to take a random sample of rows: use sample_n() for a fixed number and sample_frac() for a fixed fraction.

```
#Pull out random sample of rows
sample_n(flights, 10)  #For fixed number

sample_frac(flights, 0.00005) #For fixed fraction
# replace=TRUE for bootstrap sampling
```

Using the Pipe operator %>%

it can be very useful when trying to perform multiple operations/functions on a data set. The pipe operator will allow you to avoid either a long nested operation or doing a bunch of assignments.

```
library(dplyr)
df <- mtcars #mtcars dataset
```

```
#Using nesting
arrange(sample_n(filter(df,mpg>20),size = 5),desc(mpg))

#Using multiple assignment
a <- filter(df, mpg > 20)
b <- sample_n(a, size = 5)
c <- arrange(b,desc(mpg))
c

#Using pipe
df %>% filter(mpg > 20) %>% sample_n(size = 5) %>% arrange(desc(mpg))
```

Using the *airqulaity* dataset

```
head(airquality)
str(airquality)
library(dplyr)


filter(airquality, Temp > 70)
filter(airquality, Temp > 80, Month > 5)

mutate(airquality, TempInC = (Temp - 32) * 5/9)

summarise(airquality, mean(Temp, na.rm = T))

summarise(group_by(airquality, Month), mean(Temp, na.rm = T))

sample_n(airquality, size = 10)
sample_frac(airquality, size = 0.1)

count(airquality, Month)

arrange(airquality, desc(Month), Day)

filteredData <- filter(airquality, Month != 5)
groupedData <- group_by(filteredData, Month)
summarise(groupedData, mean(Temp, na.rm = TRUE))

airquality %>%
  filter(Month != 5) %>%
  group_by(Month) %>%
```

```
  summarise(mean(Temp, na.rm = TRUE))
```

Using the tidyr package

tidyr which is a complementary package that will help us create tidy data sets.
Tidy data is when we have a data set where every row is an observation and every column is a variable, this way the data is organized in such a way where every cell is a value for a specific variable of a specific observation.

Data tables

data.table is a package that extends data.frames. Two of its most notable features are speed and cleaner syntax.

Differences with data frames

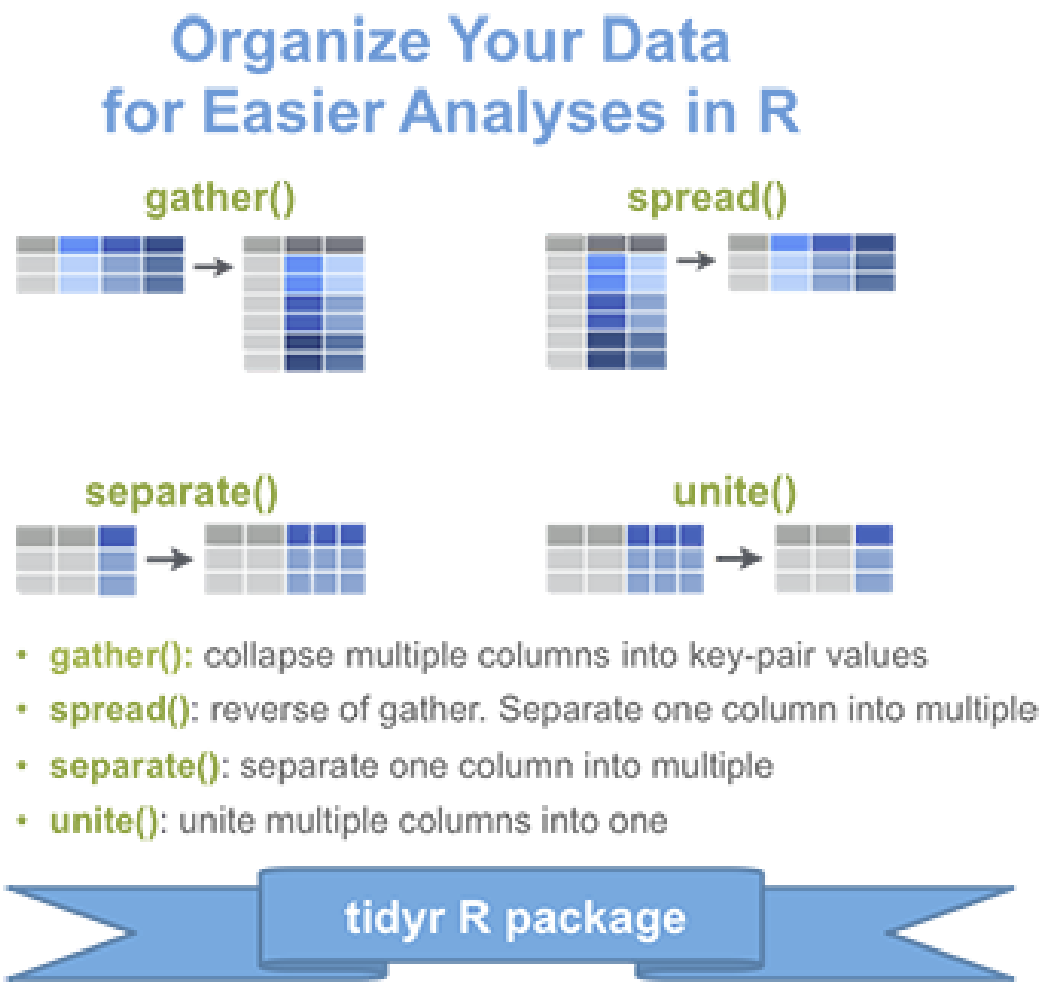
- much faster and very intuitive by operations
- You won't accidentally print out a huge data.frame with the need to press Ctrl-C, data.table prevents this sort of accident
- faster and better file reading with fread
- the package also provides a number of other utility functions, like %between% or rbindlist
- pretty much faster for a lot of basic operations, since a lot of data.frame operations copy the entire thing needlessly

tidyr functions

most useful functions in tidyr -

- gather()
- spread()
- separate()
- unite()

```
install.packages('tidyr')
install.packages('data.table') #extends data frame (speed)

library(tidyr)
library(data.table)
```




```
#Fake example dataset
comp <- c(1,1,1,2,2,2,3,3,3)
yr <- c(1998,1999,2000,1998,1999,2000,1998,1999,2000)
q1 <- runif(9, min=0, max=100)
q2 <- runif(9, min=0, max=100)
q3 <- runif(9, min=0, max=100)
q4 <- runif(9, min=0, max=100)

df <- data.frame(comp=comp,year=yr,Qtr1 = q1,Qtr2 = q2,Qtr3 = q3,
                 Qtr4 = q4)
```

gather() & spread()

Analogous to pivot tables in excel

The gather() function will collapse multiple columns into key-pair values.

The data frame above is considered wide since the time variable (represented as quarters) is structured such that each quarter represents a variable. To restructure the time component as

an individual variable, we can gather each quarter within one column variable and also gather the values associated with each quarter in a second column variable.

```
#Gather
gather(df, Quater, Revenue,Qtr1:Qtr4) #using function
help(gather)
df %>% gather(Quater, Revenue, Qtr1:Qtr4) #using pipe #Using pipe
operator
```

spread() is a complement of gather()

```
#Spread
stocks <- data.frame(
  time = as.Date('2009-01-01') + 0:9,
  X = rnorm(10, 0, 1),
  Y = rnorm(10, 0, 2),
  Z = rnorm(10, 0, 4)
)
stocks

#gather data
stocks.gather <- stocks %>% gather(stock, price,-time)
stocks.gather

stocks.gather %>% spread(stock, price)
stocks.gather %>% spread(time, price)
```

separate() & unite()

separate() - Given either regular expression or a vector of character positions, separate() turns a single character column into multiple columns.

```
#Separate
df <- data.frame(x = c(NA, "a.x", "b.y", "c.z"))
df

df %>% separate(x, c("ABC", "XYZ"))

df <- data.frame(x = c(NA, "a_x", "b_y", "c_z"))
df

df.sep <- df %>% separate(x, c("ABC","XYZ"), sep = "_")
```

unite() - convenience function to paste together multiple columns into one.

```
#Unite
unite(df.sep, new.join.col,ABC,XYZ)
unite(df.sep, new.join.col,ABC,XYZ, sep = "---")

head(mtcars)
unite_(mtcars, "vs.am", c("vs", "am"), sep= ".")
```

Visualization using ggplot2

Introduction

- ggplot2 is the most popular data visualization package for R
- Created by Hadley Wickham
- Follows "*Grammar of Graphics*"
- Each component being a layer

ggplot2 has several advantages:

- Plot specification at a high level of abstraction
- Very flexible
- Theme system for polishing plot appearance
- Mature and complete graphics system
- Many users, active mailing list
- Lot's of online help available (StackOverflow, etc...)

What ggplot2 not ideal for:

- Interactive graphics
- Graph Theory Plots (Graph Nodes)
- 3-D Graphics

Histograms are a great way of quickly exploring your data! We have a couple of options for quickly producing histograms off the columns of a data frame. We have:

- hist()
- qplot()
- ggplot()

```
library(ggplot2)
library(data.table)

#Load dataset
df <- fread('ggplot/state_real_estate_data.csv')

#Explore dataset
head(df)
tail(df)
str(df)
summary(df)
```

```
#Histograms
hist(df[['Home.Value']])
qplot(df[['Home.Value']])

# Using ggplot, lots of ability to customize, but bit more
complicated!
ggplot(data = df,aes(df$Home.Value))+geom_histogram()
```

qplot

The qplot() function can be used to create the most common graph types. While it does not expose ggplot's full power, it can create a very wide range of useful plots. The format is:

```
qplot(x, y, data=, color=, shape=, size=, alpha=, geom=, method=,
formula=, facets=, xlim=, ylim= xlab=, ylab=, main=, sub=)
```

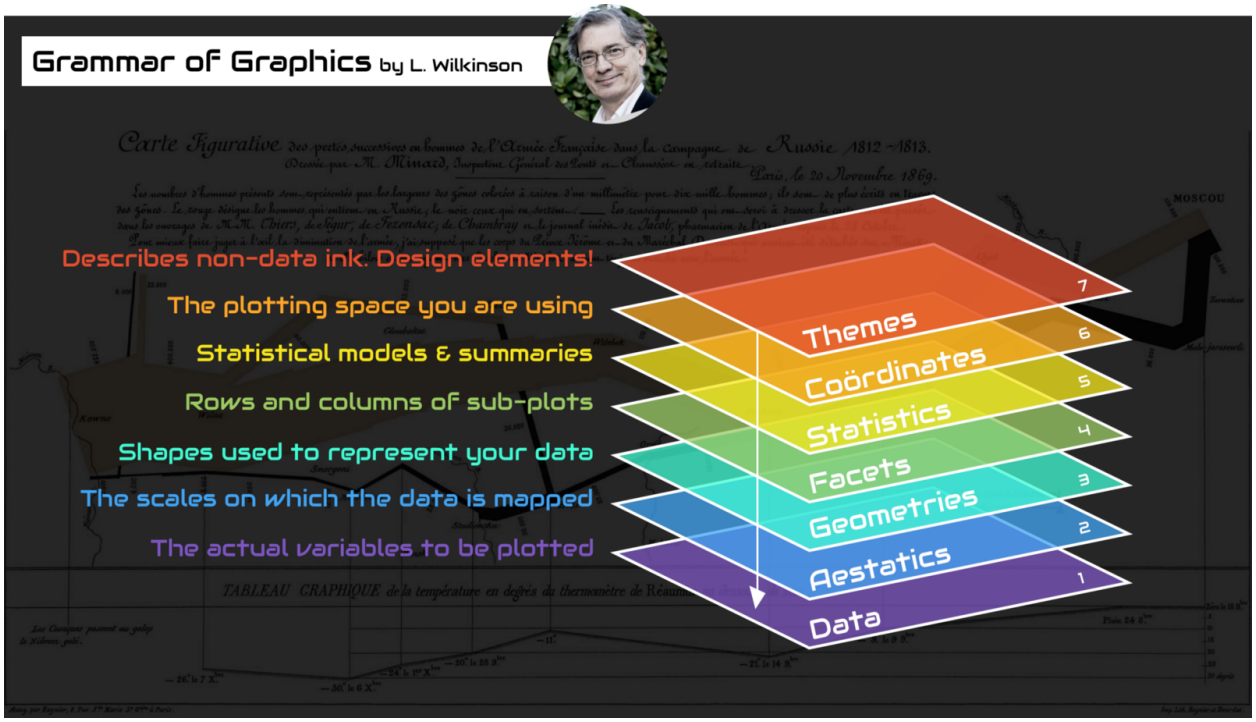
Grammar of graphics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same few components:

- a data set,
- a set of geoms—visual marks that represent data points
- a coordinate system.

To display data values, map variables in the data set to aesthetic properties of the geom like size, color, and x and y locations.

```
install.packages("ggplot2")
library(ggplot2) #Import library

#Grammar of Graphics
ggplot(data = mtcars) #Data
ggplot(data = mtcars, aes(x=mpg, y=hp)) # Data & Aesthetics

# Data, Aesthetics & Geometries
pl <- ggplot(data = mtcars, aes(x=mpg, y=hp))
pl + geom_point()  #point for basic scatter plot

# Data, Aesthetics, Geometries & Facets
pl <- ggplot(data = mtcars, aes(x = mpg,y = hp)) + geom_point()
pl + facet_grid(cyl ~ .)



# Data, Aesthetics, Geometries, Facets & Statistics
pl <- ggplot(data = mtcars, aes(x = mpg,y = hp)) + geom_point()
pl + facet_grid(cyl ~ .) + stat_smooth()

# Data, Aesthetics, Geometries, Facets, Statistics & Coordinates
pl <- ggplot(data = mtcars, aes(x = mpg,y = hp)) + geom_point()
pl2 <- pl + facet_grid(cyl ~ .) + stat_smooth()
pl2 + coord_cartesian(xlim = c(15, 25))

# Data, Aesthetics, Geometries, Facets, Statistics, Coordinates &
Theme
pl <- ggplot(data = mtcars, aes(x = mpg,y = hp)) + geom_point()
pl2 <- pl + facet_grid(cyl ~ .) + stat_smooth()
pl2 + coord_cartesian(xlim = c(15, 25)) + theme_bw()
```



Histograms with ggplot2

```
install.packages("ggplot2movies")
library(ggplot2)
library(ggplot2movies)

#use movie dataset
df <- movies[sample(nrow(movies), 1000), ]
head(df)


#qplot
qplot(rating, data = df, geom = 'histogram', binwidth=0.1, alpha =
0.8)


#ggplot (data, aesthetics)
pl <- ggplot(df, aes(x=rating))

#Adding histogram geometry
pl + geom_histogram()
```

```
#Adding color
pl <- ggplot(df, aes(x=rating))
pl + geom_histogram(binwidth = 0.1, color = 'red', fill = 'pink')

#Alpha
#Adding color
pl <- ggplot(df, aes(x=rating))
pl + geom_histogram(binwidth = 0.1, color = 'red', fill = 'pink',
alpha = 0.4)

#Adding labels
pl <- ggplot(df, aes(x=rating))
pl + geom_histogram(binwidth = 0.1, color='red', fill='pink') +
xlab('Movie Ratings') + ylab('Occurrences') + ggtitle('Movie Ratings')

#Change transperancy (alpha)
pl <- ggplot(df, aes(x=rating))
pl + geom_histogram(binwidth = 0.1, fill = 'blue', alpha = 0.4) +
xlab('Movie Ratings') + ylab('Occurrences')
```

We have the options: "blank", "solid", "dashed", "dotted", "dotdash", "longdash", and "twodash".

```
#Line types
pl <- ggplot(df,aes(x=rating))
pl +
geom_histogram(binwidth=0.1,color='blue',fill='pink',linetype='dotted')
+ xlab('Movie Ratings')+ ylab('Occurrences')+ggtitle("Movir Ratings")

#Advanced aesthetics

#Adding labels
pl <- ggplot(df,aes(x=rating))
pl + geom_histogram(binwidth=0.1,aes(fill=..count..)) + xlab('Movie
Ratings')+ ylab('Occurences')

# Adding Labels
pl <- ggplot(df,aes(x=rating))
pl2 <- pl + geom_histogram(binwidth=0.1,aes(fill=..count..)) +
xlab('Movie Ratings')+ ylab('Occurences')

# scale_fill_gradient('Label',low=color1,high=color2)
pl2 + scale_fill_gradient('Count',low='blue',high='red')

# scale_fill_gradient('Label',low=color1,high=color2)
```

```
pl2 + scale_fill_gradient('Count',low='darkgreen',high='lightblue')

#Adding density plot
# Adding Labels
pl <- ggplot(df,aes(x=rating))
pl + geom_histogram(aes(y=..density..)) + geom_density(color='red')
```

Scatter plots with R

```
library(ggplot2)
library(ggplot2movies)

df <- mtcars
head(df)

#quick plot
qplot(wt,mpg,data = df)

#Adding color gradient
qplot(wt,mpg, data = df, color = cyl)

#Adding size
qplot(wt,mpg, data = df, size = cyl)

#both
qplot(wt,mpg,data=df,size=cyl,color=cyl)

# Show 4 features
qplot(wt,mpg,data=df,size=cyl,color=hp,alpha=0.6)


#Using ggplot

#data & aesthetics & geometry
pl <- ggplot(data=df,aes(x = wt,y=mpg))
pl + geom_point()

#Addng 3rd features

#using color
pl <- ggplot(data=df,aes(x = wt,y=mpg))
pl + geom_point(aes(color=cyl))
```

```
pl <- ggplot(data=df,aes(x = wt,y=mpg))
pl + geom_point(aes(color=factor(cyl)))

pl <- ggplot(data=df,aes(x = wt,y=mpg))
pl + geom_point(aes(size=hp))

pl <- ggplot(data=df,aes(x = wt,y=mpg))
pl + geom_point(aes(size=cyl))

pl <- ggplot(data=df,aes(x = wt,y=mpg))
pl + geom_point(aes(size=factor(cyl)))


# With Shapes
pl <- ggplot(data=df,aes(x = wt,y=mpg))
pl + geom_point(aes(shape=factor(cyl)))


# Better version
# With Shapes
pl <- ggplot(data=df,aes(x = wt,y=mpg))
pl +
geom_point(aes(shape=factor(cyl),color=factor(cyl)),size=4,alpha=0.6)

#Colors
pl <- ggplot(df, aes(x=wt, y=mpg))
pl2 <- pl + geom_point(aes(color=factor(cyl)), size = 3,color='blue')
print(pl2)

#Gradient Scale
pl <- ggplot(data=df,aes(x = wt,y=mpg))
pl + geom_point(aes(colour = hp),size=4)

pl <- ggplot(data=df,aes(x = wt,y=mpg))
pl + geom_point(aes(colour = hp),size=4) +
scale_colour_gradient(high='red',low = "blue")
```

Bar Plots with ggplot2

Barplots are a useful way of displaying occurrence counts (for categorical data).There are two types of bar charts, determined by what is mapped to bar height.

1) By default, geom_bar uses stat="count" which makes the height of the bar proportion to the number of cases in each group (or if the weight aesthetic is supplied, the sum of the weights)
2) If you want the heights of the bars to represent values in the data, use stat="identity" and map a variable to the y aesthetic.

```
library(ggplot2)

# counts (or sums of weights)
g <- ggplot(mpg, aes(class))
# Number of cars in each class:
g + geom_bar()


# Bar charts are automatically stacked when multiple bars are placed
# at the same location
g <- ggplot(mpg, aes(class))
g + geom_bar(aes(fill = drv))

g <- ggplot(mpg, aes(class))
g + geom_bar(aes(fill = drv), position = "fill")


# You can instead dodge, or fill them
g <- ggplot(mpg, aes(class))
g + geom_bar(aes(fill = drv), position = "dodge")
```

Boxplots with ggplot2

Boxplots are convenient way of graphically depicting groups of numerical data through their quartiles. Box plots may also have lines extending vertically from the boxes (whiskers) indicating variability outside the upper and lower quartiles, hence the terms box-and-whisker plot and box-and-whisker diagram. Outliers may be plotted as individual points.

```
library(ggplot2)

df <- mtcars
```

```
head(df)

qplot(factor(cyl), mpg, data = mtcars, geom = "boxplot")

#Using ggplot
pl <- ggplot(mtcars, aes(factor(cyl), mpg))
pl + geom_boxplot()

pl <- ggplot(mtcars, aes(factor(cyl), mpg))
pl + geom_boxplot() + coord_flip()


pl <- ggplot(mtcars, aes(factor(cyl), mpg))
pl + geom_boxplot(aes(fill = factor(cyl)))

pl <- ggplot(mtcars, aes(factor(cyl), mpg))
pl + geom_boxplot(fill = "grey", color = "blue")
```

2 variable plotting

compare two variables from a dataset.

```
library(ggplot2)
library(ggplot2movies)

df <- movies

qplot(x=year, y=rating, data = df, geom = "density2d")

#2d binchart
pl <- ggplot(movies,aes(x = year,y=rating))
pl + geom_bin2d()

pl <- ggplot(movies,aes(x = year,y=rating))
# Control bin sizes
pl + geom_bin2d(binwidth=c(2,1))

#Density plots
pl <- ggplot(movies,aes(x = year,y=rating))
pl + geom_density2d()
```

Coordinates & Faceting

Learning how to deal with coordinates will allow us to size our plots correctly. Faceting will allow us to place several plots next to each other, these plots are usually related by the same dataset.

```
library(ggplot2)

pl <- ggplot(mpg,aes(x=displ,y=hwy)) + geom_point()
pl

#Setting x & y limits
pl + coord_cartesian(xlim=c(1,4),ylim=c(15,30))

#Change aspect ratio
# aspect ratio, expressed as y / x
pl + coord_fixed(ratio = 1/3)


p <- ggplot(mpg, aes(displ, cty)) + geom_point()

p + facet_grid(. ~ cyl)

p + facet_grid(drv ~ .)

p + facet_grid(drv ~ cyl)
```

Themes

There are a lot of built-in themes in ggplot and you can use them in two ways, by stating before your plot to set the theme or by adding them to your plot directly. There is also a great library called ggthemes which adds even more built-in themes for ggplot.

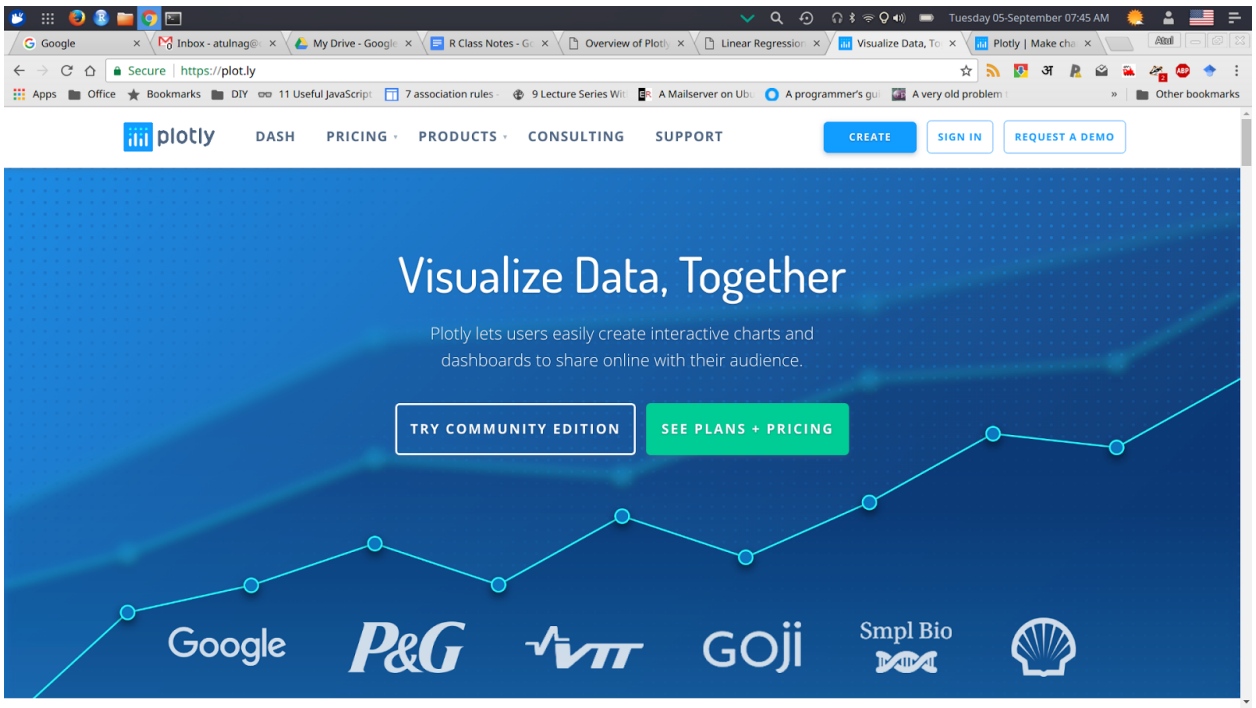
```
library(ggplot2)

#set theme
theme_set(theme_bw())

#Add theme to plot
my_plot + theme_bw()

df <- mtcars
```

```
head(df)
pl <- ggplot(df,aes(x=mpg,y=hp)) + geom_point()
print(pl)

#Themes
pl + theme_bw()

pl + theme_classic()

pl + theme_dark()

pl + theme_get()

pl + theme_light()

pl + theme_linedraw()

pl + theme_minimal()

pl + theme_void()

install.packages("ggthemes")
library(ggthemes)

pl + theme_excel()

pl + theme_economist()

pl + theme_economist_white()
```

Interactive Visualization

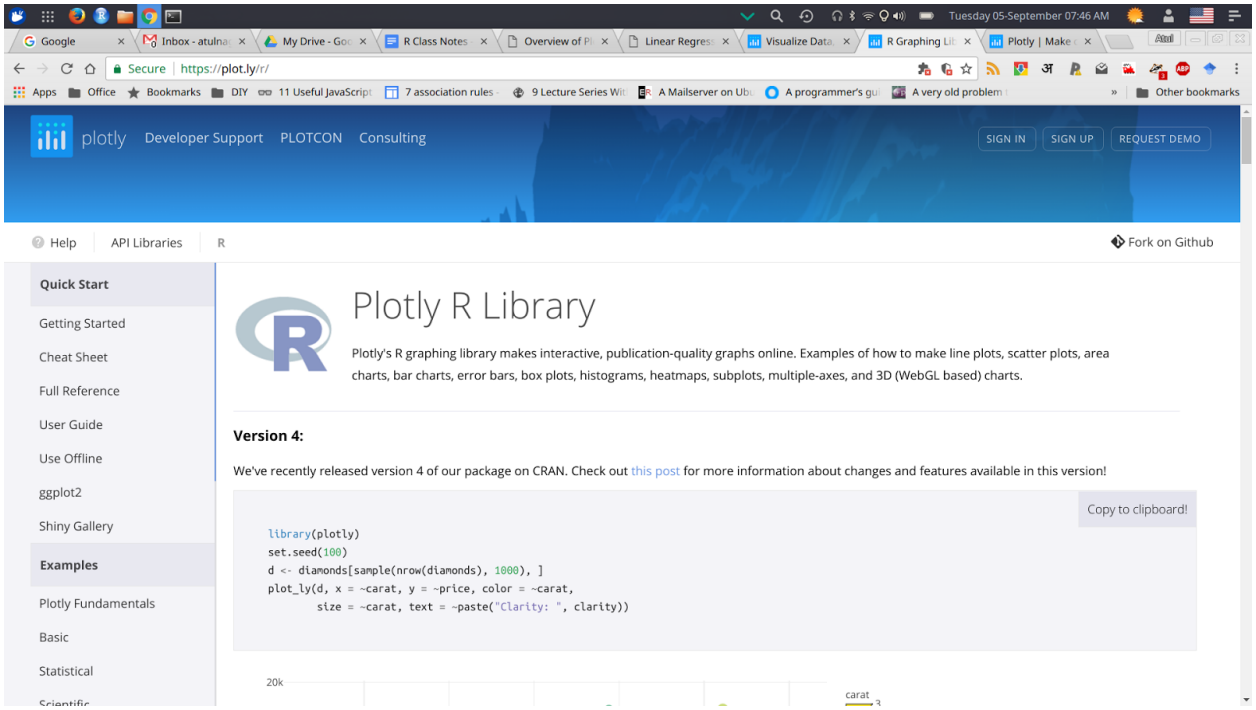
Overview of Plotly and R

Plotly is an R package for creating interactive web-based graphs via plotly's JavaScript graphing library, plotly.js.



The plotly R libary contains a function ggplotly which will convert ggplot2 figures into graphs drawn with plotly.js which can be saved to an online plotly account or rendered locally.

Plotly has its own graphing call using plot_ly() but it also has ggplotly which allows you directly call ggplot() code! Sometimes this code needs some slight format change.

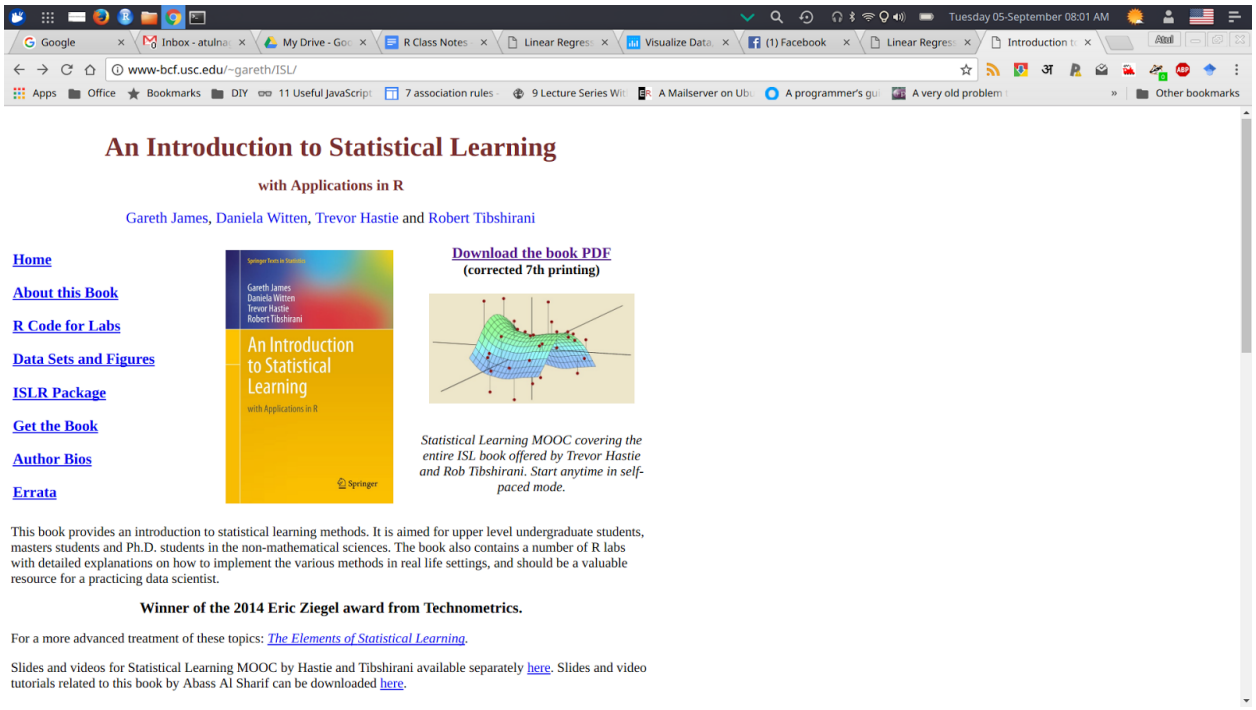



```
#install plotly
install.packages("plotly")

#Load plotly
library(plotly)
library(ggplot2)

#ggplo2
pl <- ggplot(mtcars, aes(mpg, wt)) + geom_point()
print(pl)

#pass ggplot to ggplotly
gpl <- ggplotly(pl)

print(gpl)
```


plotly
Developer Support
PLOTCON
Consulting
SIGN IN
SIGN UP
REQUEST DEMO
Shiny Gallery
Examples
Plotly Fundamentals
Basic
Statistical
Scientific
Financial
Maps
3D
Subplots
library(plotly)
set.seed(100)
d <- diamonds[sample(nrow(diamonds), 1000),]
plot_ly(d, x = ~carat, y = ~price, color = ~carat,
size = ~carat, text = ~paste("Clarity: ", clarity))
price
20k
15k
10k
5k
0
0.5
1
1.5
2
2.5
3
carat
EDIT CHART

Introduction to Machine Learning

using Introduction to Statistical Learning by Gareth James as a companion book.



What is Machine Learning ?

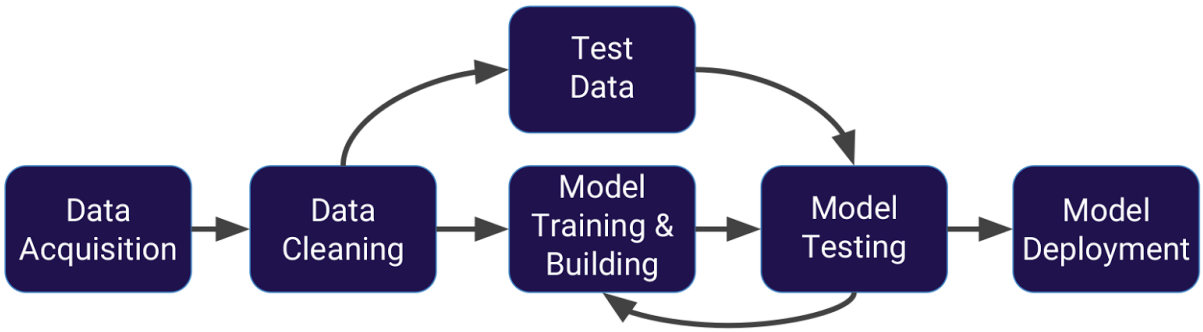
- Machine learning is a method of data analysis that automates analytical model building.
- Using algorithms that iteratively learn from data, machine learning allows computers to find hidden insights without being explicitly programmed where to look.

Uses of Machine Learning

- Fraud detection.
- Recommendation Engines
- Web search results.
- Customer Segmentation
- Real-time ads on web pages
- Text Sentiment Analysis
- Credit scoring and next-best offers.
- Predicting Customer Churn
- Prediction of equipment failures.
- Pattern and image recognition.
- New pricing models.

- Email spam filtering.
- Network intrusion detection.
- Financial Modeling

Machine Learning Process



Supervised Learning

- Supervised learning algorithms are trained using labeled examples, such as an input where the desired output is known.
- For example, a piece of equipment could have data points labeled either “F” (failed) or “R” (runs).
- The learning algorithm receives a set of inputs along with the corresponding correct outputs, and the algorithm learns by comparing its actual output with correct outputs to find errors.
- It then modifies the model accordingly.
- Through methods like classification, regression, prediction and gradient boosting, supervised learning uses patterns to predict the values of the label on additional unlabeled data.
- Supervised learning is commonly used in applications where historical data predicts likely future events.
- For example, it can anticipate when credit card transactions are likely to be fraudulent or which insurance customer is likely to file a claim.
- Or it can attempt to predict the price of a house based on different features for houses for which we have historical price data.

Unsupervised Learning

- Unsupervised learning is used against data that has no historical labels.
- The system is not told the "right answer." The algorithm must figure out what is being shown.
- The goal is to explore the data and find some structure within.
- Or it can find the main attributes that separate customer segments from each other.
- Popular techniques include self-organizing maps, nearest-neighbor mapping, k-means clustering and singular value decomposition.
- These algorithms are also used to segment text topics,recommend items and identify data outliers.

Reinforcement Learning

- Reinforcement learning is often used for robotics,gaming and navigation.
- With reinforcement learning, the algorithm discovers through trial and error which actions yield the greatest rewards.
- This type of learning has three primary components: the agent (the learner or decision maker), the environment (everything the agent interacts with) and actions (what the agent can do).
- The objective is for the agent to choose actions that maximize the expected reward over a given amount of time.
- The agent will reach the goal much faster by following a good policy.
- So the goal in reinforcement learning is to learn the best policy.

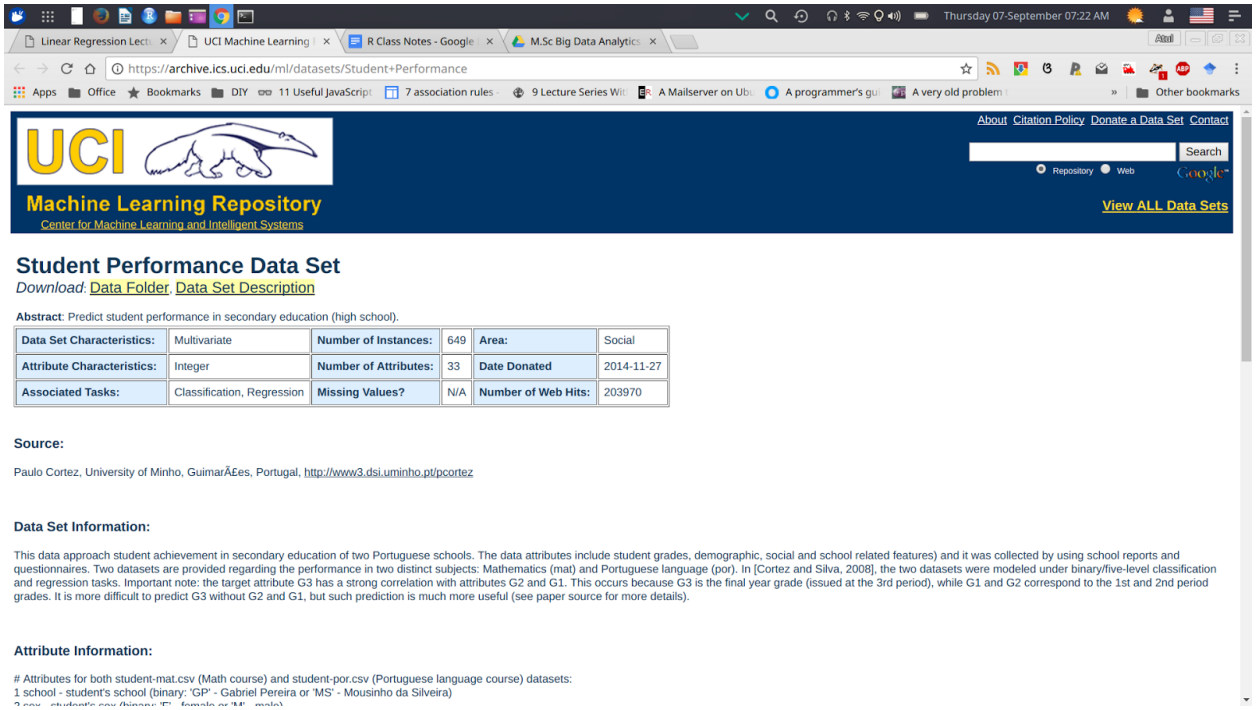
Linear Regression

Linear Regression is a supervised learning algorithm, meaning we'll have labeled data and try to predict new labels on unlabeled data. We'll explore some of the following concepts:

- Data Acquisition
- Exploratory Data Analysis (EDA)
- Clean our Data
- Review of Model Form
- Train and Test Groups
- Linear Regression Model

Data Acquisition

We will use the Student Performance Data Set from UC Irvine's Machine Learning Repository!



We'll specifically look at the math class (student-mat.csv) (the delimiter is a semi-colon.)

```
# Read CSV, note the delimiter (sep)
df <- read.csv('student-mat.csv',sep=';')

head(df)

summary(df)
```

Attribute Information

Attributes for both student-mat.csv (Math course) and student-por.csv (Portuguese language course) datasets:

1. school - student's school (binary: "GP" - Gabriel Pereira or "MS" - Mousinho da Silveira)
2. sex - student's sex (binary: "F" - female or "M" - male)
3. age - student's age (numeric: from 15 to 22)
4. address - student's home address type (binary: "U" - urban or "R" - rural)
5. famsize - family size (binary: "LE3" - less or equal to 3 or "GT3" - greater than 3)
6. Pstatus - parent's cohabitation status (binary: "T" - living together or "A" - apart)
7. Medu - mother's education (numeric: 0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education)
8. Fedu - father's education (numeric: 0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education)
9. Mjob - mother's job (nominal: "teacher", "health" care related, civil "services" (e.g. administrative or police), "at_home" or "other")
10. Fjob - father's job (nominal: "teacher", "health" care related, civil "services" (e.g. administrative or police), "at_home" or "other")
11. reason - reason to choose this school (nominal: close to "home", school "reputation", "course" preference or "other")
12. guardian - student's guardian (nominal: "mother", "father" or "other")
13. traveltime - home to school travel time (numeric: 1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour)
14. studytime - weekly study time (numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours)
15. failures - number of past class failures (numeric: n if 1<=n<3, else 4)
16. schoolsup - extra educational support (binary: yes or no)
17. famsup - family educational support (binary: yes or no)
18. paid - extra paid classes within the course subject (Math or Portuguese) (binary: yes or no)
19. activities - extra-curricular activities (binary: yes or no)
20. nursery - attended nursery school (binary: yes or no)
21. higher - wants to take higher education (binary: yes or no)
22. internet - Internet access at home (binary: yes or no)
23. romantic - with a romantic relationship (binary: yes or no)
24. famrel - quality of family relationships (numeric: from 1 - very bad to 5 - excellent)
25. freetime - free time after school (numeric: from 1 - very low to 5 - very high)
26. goout - going out with friends (numeric: from 1 - very low to 5 - very high)
27. Dalc - workday alcohol consumption (numeric: from 1 - very low to 5 - very high)
28. Walc - weekend alcohol consumption (numeric: from 1 - very low to 5 - very high)
29. health - current health status (numeric: from 1 - very bad to 5 - very good)
30. absences - number of school absences (numeric: from 0 to 93)

these grades are related with the course subject, Math or Portuguese:
31 G1 - first period grade (numeric: from 0 to 20)
31 G2 - second period grade (numeric: from 0 to 20)
32 G3 - final grade (numeric: from 0 to 20, output target)

Data Cleaning

Check for NA values

```
#Check for NA values
any(is.na(df))
```

- Most real data sets will probably have NA or Null values,
- to deal with them, either dropping them if they aren't too many, or imputing other values, like the mean value.

Categorical Features

Check categorical variables have a factor set to them. For example, the MJob column refers to categories of Job Types, not some numeric value from 1 to 5.

```
#Check variables for categories factor
str(df)
```

Exploratory Data Analysis

```
library(ggplot2)
library(ggthemes)
library(dplyr)
```

Correlation & CorrPlots

Correlation defined as - In statistics, dependence or association is any statistical relationship, whether causal or not, between two random variables or two sets of data.
Correlation is any of a broad class of statistical relationships involving dependence, though in common usage it most often refers to the extent to which two variables have a linear relationship with each other. Familiar examples of dependent phenomena include the correlation between the physical statures of parents and their offspring, and the correlation between the demand for a product and its price.
Correlation plots are a great way of exploring data and seeing if there are any interaction terms.
Beginning with numeric data (no correlation for categorical data)

For Visualizing correlation

```
#Correlation
# Grab only numeric columns
num.cols <- sapply(df, is.numeric)

# Filter to numeric columns for correlation
cor.data <- cor(df[,num.cols])

install.packages('corrgram')
install.packages('corrplot')

library(corrgram)
library(corrplot)

help("corrplot")

corrplot(cor.data,method='color')
```

- Clearly we have very high correlation between G1, G2, and G3 which makes sense since those are grades.
- Meaning good students do well each period, and poor students do poorly each period, etc.
- Also a high G1,G2, or G3 value has a negative correlation with failure (number of past class failures).
- Also Mother and Father education levels are correlated, which also makes sense.

corrgram which allows to just automatically do these type of figures by just passing in the dataframe directly.

```
corrgram(df,order=TRUE, lower.panel=panel.shade,
         upper.panel=panel.pie, text.panel=panel.txt)
```

Since we're going to eventually try to predict the G3 score let's see a histogram of these scores:

```
#histogram
ggplot(df,aes(x=G3)) + geom_histogram(bins=20,alpha=0.5,fill='blue') +
theme_minimal()
```

Looks like quite a few students get a zero. This is a good place to ask questions, like are students missing the test? Also why is the mean occurrence so high? Is this test curved?

Building a Model

General form

The general model of building a linear regression model in R looks like this:

```
model <- lm(y ~ x1 + x2,data)
```

or to use all the features in your data

```
model <- lm(y ~. , data) # Uses all features
```

Training & Test Data

We'll need to split our data into a training set and a testing set in order to test our accuracy. We can do this easily using the caTools library:

```
install.packages("caTools")
```

```
# Import Library
library(caTools)
```

```
# Set a random see so your "random" results are the same as this
notebook
set.seed(101)
```

```
# Split up the sample, basically randomly assigns a booleans to a new
column "sample"
sample <- sample.split(df$age, SplitRatio = 0.70) # SplitRatio =
percent of sample==TRUE
```

```
# Training Data
train = subset(df, sample == TRUE)
```

```
# Testing Data
test = subset(df, sample == FALSE)
```

Training

Let's train out model on our training data, then ask for a summary of that model:

```
#Training
```

```
model <- lm(G3 ~ .,train)
summary(model)
```

Model Interpretation

#	Name	Description
1	Residuals	The residuals are the difference between the actual values of the variable you're predicting and predicted values from your regression--y - ŷ. For most regressions you want your residuals to look like a normal distribution when plotted. If our residuals are normally distributed, this indicates the mean of the difference between our predictions and the actual values is close to 0 (good) and that when we miss, we're missing both short and long of the actual value, and the likelihood of a miss being far from the actual value gets smaller as the distance from the actual value gets larger. Think of it like a dartboard. A good model is going to hit the bullseye some of the time (but not everytime). When it doesn't hit the bullseye, it's missing in all of the other buckets evenly (i.e. not just missing in the 16 bin) and it also misses closer to the bullseye as opposed to on the outer edges of the dartboard.
2	Significance Stars	The stars are shorthand for significance levels, with the number of asterisks displayed according to the p-value computed. *** for high significance and * for low significance. In this case, *** indicates that it's unlikely that no relationship exists b/w absences and G3 scores.
3	Estimated Coeffecient	The estimated coefficient is the value of slope calculated by the regression. It might seem a little confusing that the Intercept also has a value, but just think of it as a slope that is always multiplied by 1. This number will obviously vary based on the magnitude of the variable you're inputting into the regression, but it's always good to spot check this number to make sure it seems reasonable.
4	Standard Error of the Coefficient Estimate	Measure of the variability in the estimate for the coefficient. Lower means better but this number is relative to the value of the coefficient. As a rule of thumb, you'd like this value to be at least an order of magnitude less than the coefficient estimate.
5	t-value of the Coefficient Estimate	Score that measures whether or not the coefficient for this variable is meaningful for the model. You probably won't use this value itself, but know that it is used to calculate the p-value and the significance levels.
6	Variable p-value	Probability the variable is *NOT* relevant. You want this number to be as small as possible. If the number is *really* small, R will display it in scientific notation.

7	Significance Legend	The more punctuation there is next to your variables, the better. Blank=bad, Dots=pretty good, Stars=good, More Stars=very good
8	Residual Std Error / Degrees of Freedom	The Residual Std Error is just the standard deviation of your residuals. You'd like this number to be proportional to the quantiles of the residuals in #1. For a normal distribution, the 1st and 3rd quantiles should be 1.5 +/- the std error. The Degrees of Freedom is the difference between the number of observations included in your training sample and the number of variables used in your model (intercept counts as a variable).
9	R-squared	Metric for evaluating the goodness of fit of your model. Higher is better with 1 being the best. Corresponds with the amount of variability in what you're predicting that is explained by the model. WARNING: While a high R-squared indicates good correlation, correlation does *not* always imply causation.
10	F-statistic & resulting p-value	Performs an F-test on the model. This takes the parameters of our model (in our case we only have 1) and compares it to a model that has fewer parameters. In theory the model with more parameters should fit better. If the model with more parameters (your model) doesn't perform better than the model with fewer parameters, the F-test will have a high p-value (probability *NOT* significant boost). If the model with more parameters is better than the model with fewer parameters, you will have a lower p-value. The DF, or degrees of freedom, pertains to how many variables are in the model. In our case there is one variable so there is one degree of freedom.

Looks like Absences, G1, and G2 scores are good predictors. With age and activities also possibly contributing to a good model.

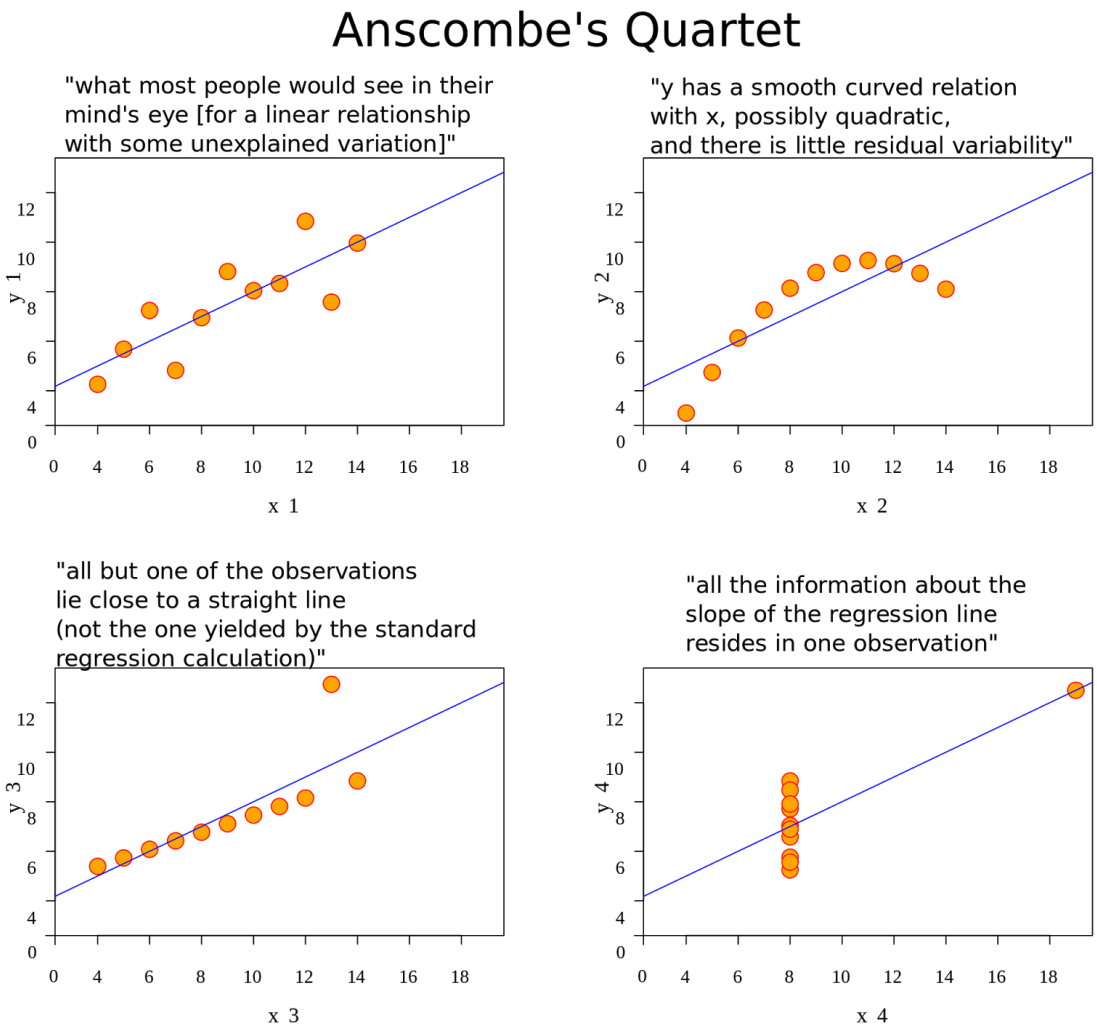
Visualize Model

We can visualize our linear regression model by plotting out the residuals, the residuals are basically a measure of how off we are for each point in the plot versus our model (the error).

```
# Grab residuals

res <- residuals(model)

# Convert to DataFrame for gglpot

res <- as.data.frame(res)

head(res)
```

Residuals Plot

We want a histogram of our residuals to be normally distributed, something with a strong bimodal distribution may be a warning that our data was not a good fit for lienar regression. However, this can also be hidden from out model. A famous example is Anscombe's Quartet




```
# Histogram of residuals

ggplot(res,aes(res)) +  geom_histogram(fill='blue',alpha=0.5)
```

Looks like there are some suspicious residual values that have a value less than -5. We can further explore this by just calling plot on our model.

```
plot(model)
```

Basically after looking at these plots what you will realize is that our model (behaving as a continuous line, predicted students would get negative scores on their test! Let's make these all zeros when running our results against our predictions.

Predictions

Let's test our model by predicting on our testing set:

```
#Predictions
G3.predictions <- predict(model,test)
```

Now we can get the root mean squared error, a standardized measure of how off we were with our predicted values:

```
results <- cbind(G3.predictions,test$G3)
colnames(results) <- c('pred','real')
results <- as.data.frame(results)
```

Now let's take care of negative predictions! Lot's of ways to this, here's a more complicated way, but its a good example of creating a custom function for a custom problem:

```
to_zero <- function(x){
  if  (x < 0){
    return(0)
  }else{
    return(x)
  }
}

results$pred <- sapply(results$pred,to_zero)
```

There's lots of ways to evaluate the prediction values, for example the MSE (mean squared error):

```
mse <- mean((results$real-results$pred)^2)
print(mse)
```

Or the root mean squared error: `mse^0.5`

Or just the R-Squared Value for our model (just for the predictions)

```
SSE = sum((results$pred - results$real)^2)
SST = sum( (mean(df$G3) - results$real)^2)

R2 = 1 - SSE/SST
R2
```

Pros & Cons of Linear Regression

quick overview of pros and cons right now of Linear Regression:
Pros:

- Simple to explain
- Highly interpretable
- Model training and prediction are fast
- No tuning is required (excluding regularization)
- Features don't need scaling
- Can perform well with a small number of observations
- Well-understood

Cons:

- Assumes a linear relationship between the features and the response
- Performance is (generally) not competitive with the best supervised learning methods due to high bias
- Can't automatically learn feature interactions