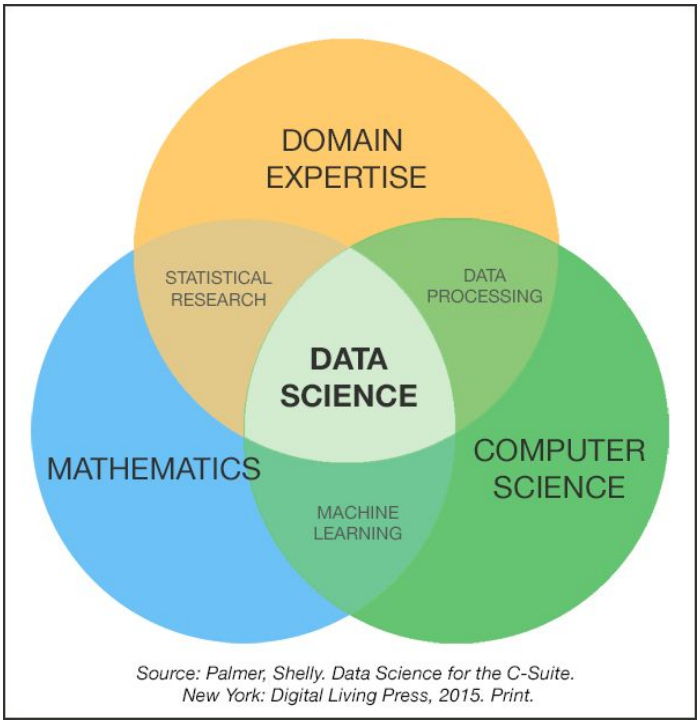
Computing for Data Science

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

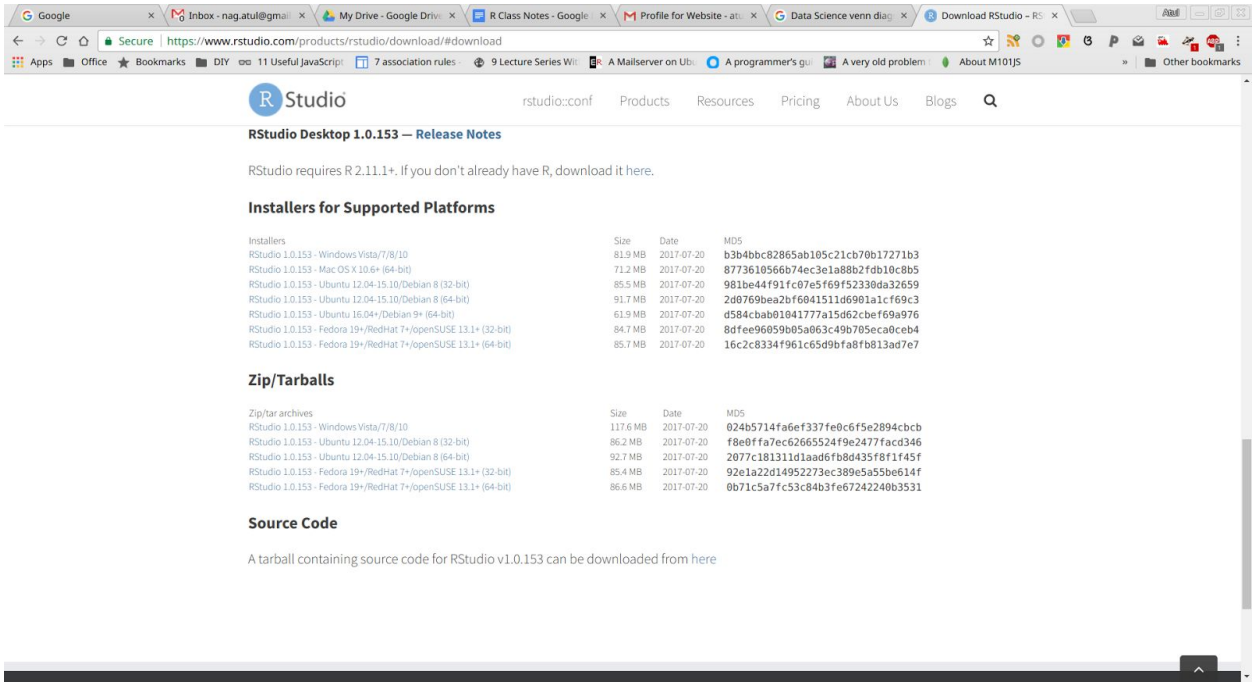


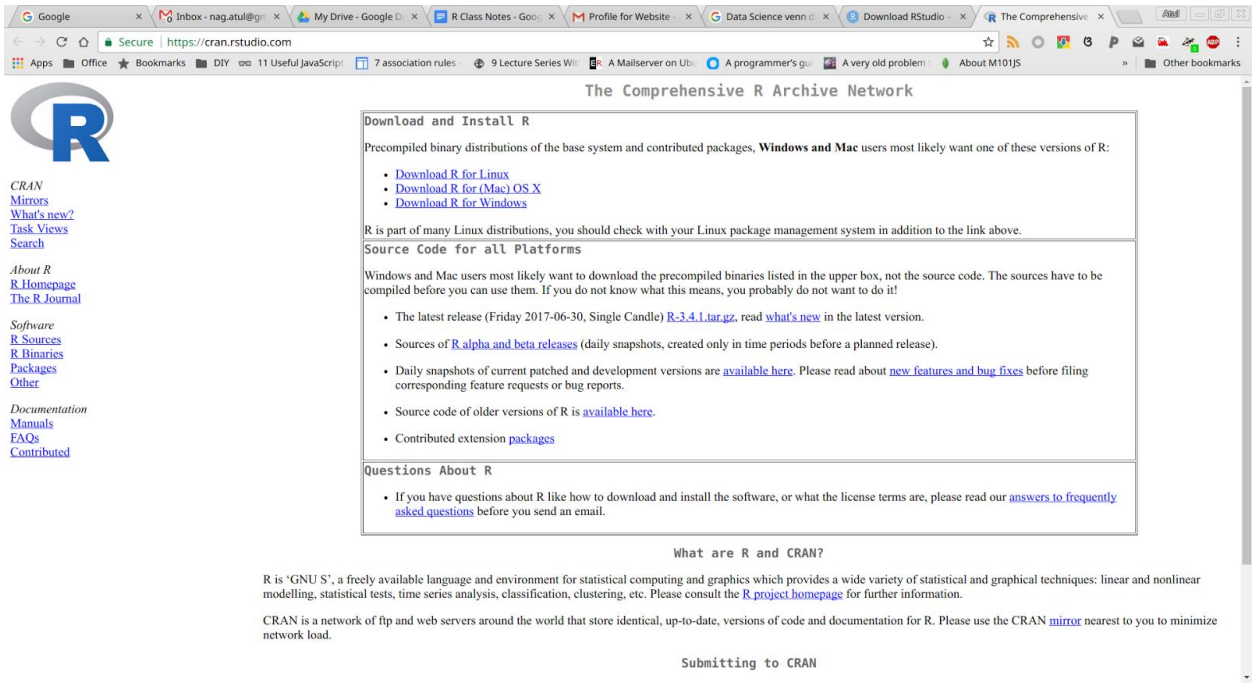
Source: Palmer, Shelly. Data Science for the C-Suite. New York: Digital Living Press, 2015. Print.

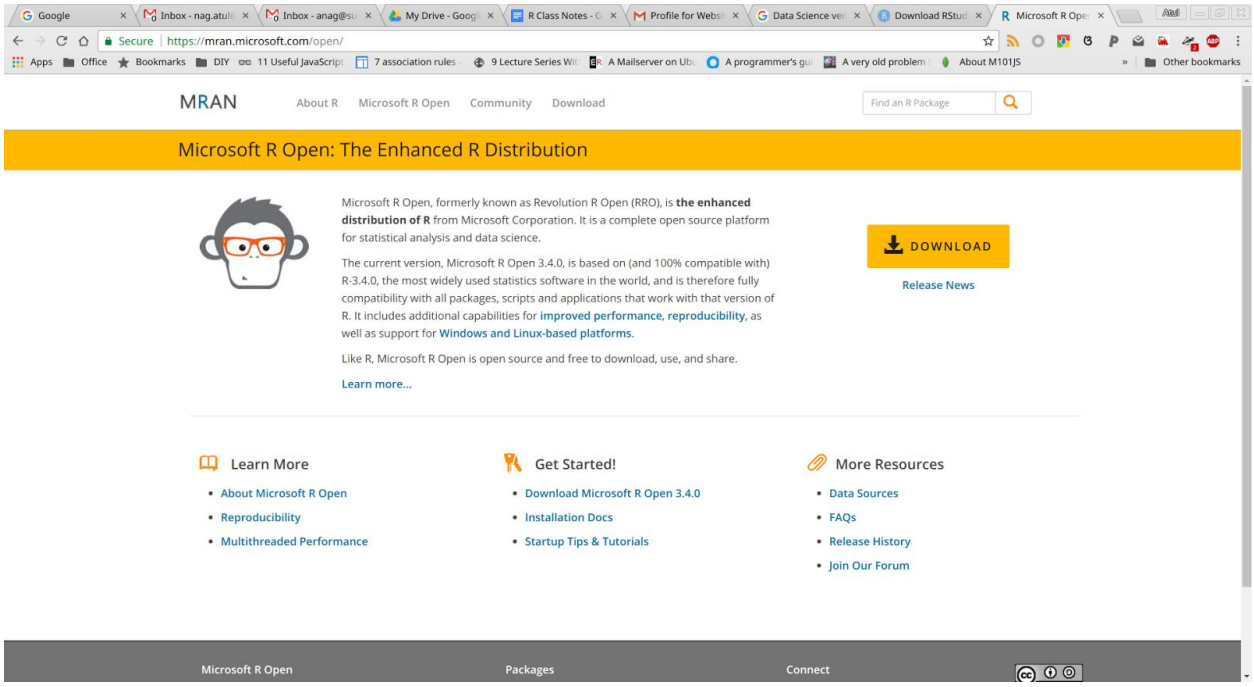
Developmental Setup

Windows

R Studio/R Download -



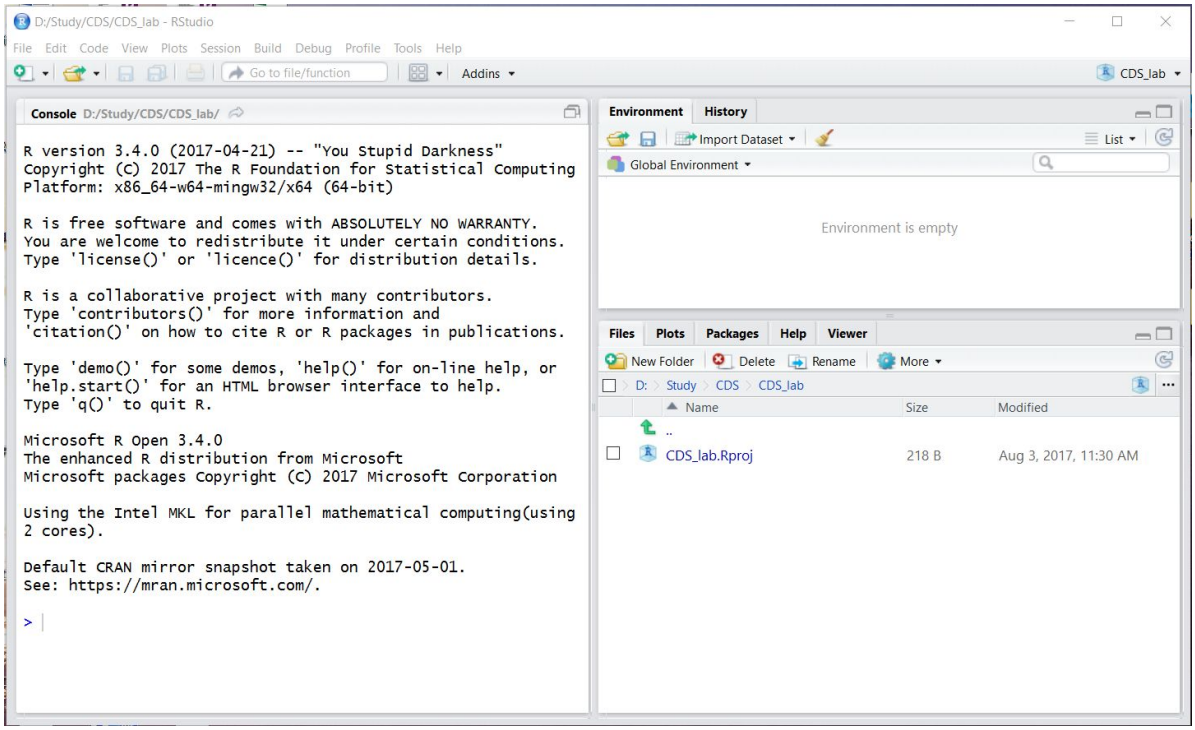




Linux (Ubuntu/Debian)

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

For R Studio download the .deb package from site.



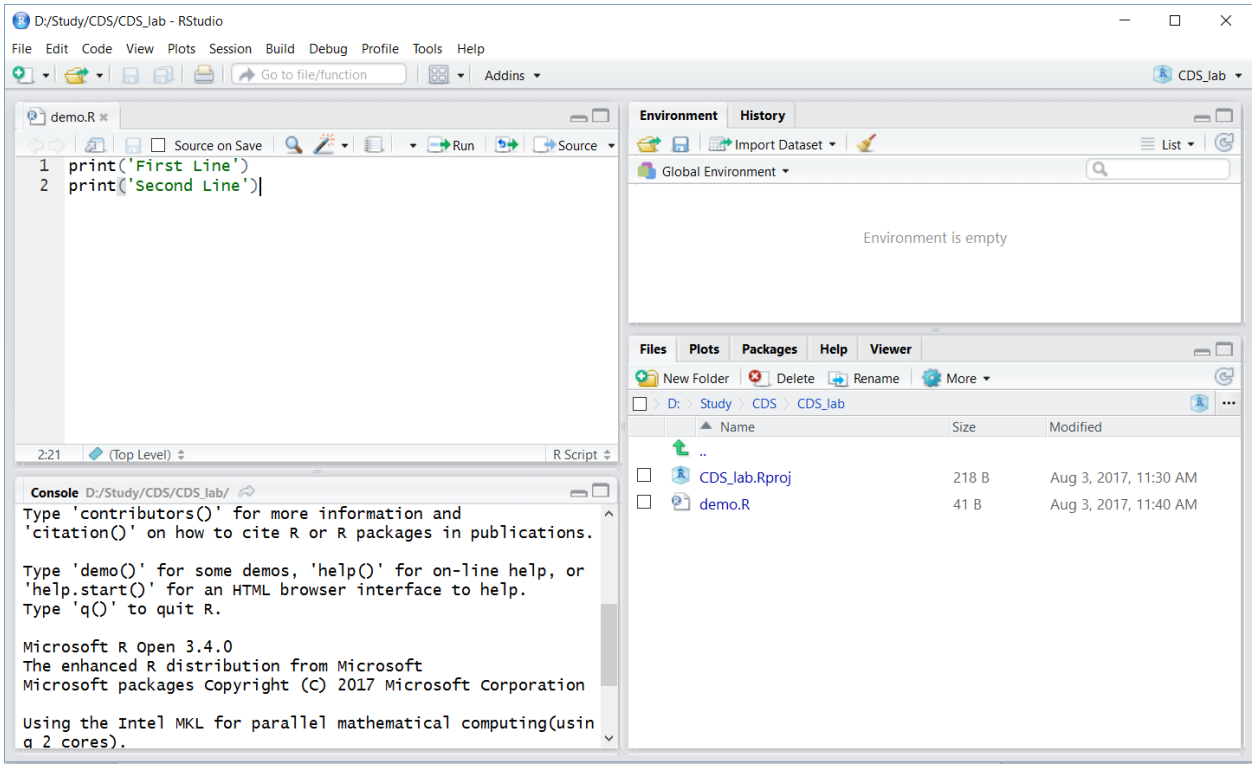
Print in the R console

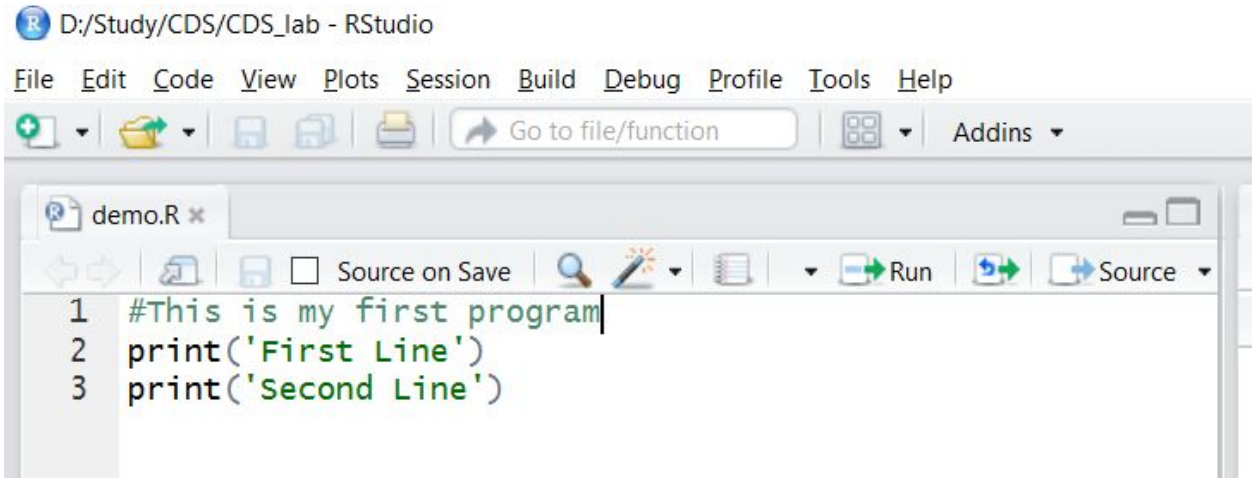
```
print('Hello World')
O/P:[1] "Hello World"
```

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
```