Hadoop Learning

TABLE OF CONTENTS

1. R LEARNING

1.1 R BASICS

https://www.datacamp.com/community/tutorials/r-tutorial-apply-family

```
#creating sample dataframe and Sample Code
m <- matrix(data=cbind(rnorm(30, 0), rnorm(30, 2), rnorm(30, 5)), nrow=30, ncol=3)

################################################################################
#R Data structure
################################################################################
#R, we have objects which are Functions and objects which are data

#Boolean,Single character, Vectors (can contain only one data type).
#special type of Vectors are factors.
#factors can be generated by "gl" (generate level) fuction,
gl(2,4,labels=c("male","female"))
as.factor(c(rep("male",10),rep("Female",10)))

#Matrix, are standard form of cross reference numbers.
matrix(c(1,2,3,4,5,6)+pi, nrow=2)
matrix(c(1,2,3,4,5,6)+pi, nrow=2)<6

#DataFrames are set of parallel vectors, where vectors can be different types.
data.frame(treatment=c("active","active","placebo"),bp=c(80,8,90))

#Compare against Matrixs -> R would convert the data into character
cbind(treatment=c("active","active","placebo"),bp=c(80,8,90))

#List, Elements can be of different type and length and can also be another list.
#When vectors are not of equal length then we can have list.
```

1.1.1 DIFFERENT PACKAGES

```
str(Cars93)
df_main <- Cars93
#subsetting
library(dplyr) # data manipulation
library(reshape2) # data manipulation
library(MASS) # get the sample data set
library(chron) # get month, day, year
library(ggplot2)
library(lattice)
```

library(astsa) # package for visualization not installed

1.1.2 MANAGE R OBJECTS

```
#getting internal help
?mean
example(mean)
# Often we don't know exactly what we are looking for
??"fitting linear model"
or
help.search("fitting linear model")

#Find the length
length(x)

# Remove the entire List from Memory
remove(list=ls())
rm(x)

#saving object .RData
save.image(file = 'var.RData')

#List all objects
ls()
#display rownames & column names
rownames(x)
colnames(x)

# Getting dimension and column info
#Summary Function
summary(airquality$Wind)
summary(airquality)
```

1.1.3 VECTORS MATRIX'S AND LIST

Vectors: All elements must be of same type.
Matrix: It's a special kind of Vector, with two additional attributes With rows and columns.
List: List can contain elements of different types.
DataFrame: It's used for storing data tables, a list of Vectors of equal length.

```
# Set number of decimals in screen
pi
options(digits=22)
pi

# Infinity as object
```

```
1/0
2*Inf
-1/0

# Not a number(NaN)
0/0

# Not available(NA)
x <- c(1,2,3,4,NA,6)
mean(x)

# Identify the type
typeof(df_main)

#Convert as dataframe
df_main <- as.data.frame(df_main)

# Subsetting vector
x <- c(10,9,8,1,2,3,4,5,6)
y <- c(6,7,8,4,5,6)
z <- c(x,y)
r <- x[x>5]
r
r <- z[z >5]
r
z
x1 <- x[c(2,5)]
x1


# 3d Array doubt?
B2 <- array(c(1:3),c(2,3))
B2

c2 <- array(seq(1,3,length=12),c(2,3,2))
c2
c2[,,1]
dim(c2)

# Some Query
a <- c(((-3+sqrt(3^2-4*1*1))/2),((-3-sqrt(3^2-4*1*1))/2))
a
c(-.4,-2.6)/a -1

df <- df_main
#display column names
names(df)
class(df$name)
nrow(df)
```

```
ncol(df)
str(df)
head(airquality,3)
tail(airquality,3)
sqrt(225)
abs(-13)
round(3.1415)
round(sales.by.month / days.per.month)
round(3.14165, 2)
round(x = 3.1415, digits = 2)
#See the Column List
names(df)
#Clear the Memory
remove(list=ls())
#check how many rows and columns
dim(sales.by.month)

#subsetting Vector
A <- matrix((-4):5,nrow=2,ncol=5)
A
A[A<0] <- 0
A
A[2,]
A[,c(2,4)]
A
class(A)

sales.by.month <- c(0, 100, 200, 50, 0, 0, 0, 0, 0, 0, 0, 0)
sales.by.month[2:4] #accessing second to fourth row
sales.by.month[c(1,3,6)] #accessing (1,3,6) row
sales.by.month[5,1] #access 5th row and 1st column
sales.by.month[5] <- 25 #assign 25 to 5th row
```

1.2 READ/ WRITE/ TO PDF

1.2.1 CONNECTING TO DATABASE

Step 1: Install and Load the Package

```
install.packages('RJDBC')
```

library(RJDBC)

Step 2: Download Oracle RJDBC Driver

Go to http://www.oracle.com/technetwork/database/enterprise-edition/jdbc-112010-090769.html.

Select the appropriate edition and download the ojdbc6.jar file. Place it in a permanent directory.

Step 3: Create a Driver Object in R

jdbcDriver =JDBC("oracle.jdbc.OracleDriver",classPath="/directory/ojdbc6.jar")

Step 4: Create a Connection to the Oracle Database

jdbcConnection =dbConnect(jdbcDriver, "jdbc:oracle:thin:@//database.hostname.com:port/service_name_or_sid", "username", "password")

Step 5: Run Oracle SQL Query

Examples:

```
# dbReadTable: read a table into a data frame
table1=dbReadTable(con,'table1')

# dbGetQuery: read the result from a SQL statement to a data frame
table2=dbGetQuery(con,'select * from table1 where name=\'string\'')

# dbWriteTable: write a data frame to the schema. It is typically very slow with large tables.

dbWriteTable(con,'Table_Name',data)

# dbSendUpdate: execute SQL command
dbSendUpdate(con,'drop table Dummy_Table')

dbSendUpdate(con,'select * from table')

#it does not return anything
```

1.2.2 READ.CSV

```
#get and set working directory
getwd()
setwd("H:/Analytics/VishnuAnalytics/Spatial_Analytics/Learning")
```

```
#Load the file
data<-read.csv("C:/Analytics/path/file1.csv") #Load the file

df <-read.table(filename,header=True,sep=" ",dec="."
        nrows=1000,na.strings=" ", skip=3,
        commecnt.char="DatabaseServerName")

#dec, decimal separator
#nrow, no. of rows to be loaded
#na.strings, what string represents a missing value.
#skip, how many lines to skip before start reading.
#comment.char, what char in the beginning of aline should indicate that the line should be skipped.

read.csv() # Comma separated, dot as decimal point
read.csv2() # semicolon separated, comma as decimal point, conventions for western europe
read.fwf() #fixed width format
```

1.2.3 SCAN

```
# Scan similar to read.table but little tricky to use but its very flexible
# you can load it into either vector or list
Scan()

# readLines(), Reads entire lines
vec <- readLines("sometextfile.txt")
vec

#split the vector
vec[2] <- strsplit(vec[2]," ")
#vec[2]<- as.numeric(vec[2]," "[[1]])
as.numeric(strsplit(vec[2]," ")[[1]])
```

1.2.4 READLINES

```
# readLines(), Reads entire lines
vec <- readLines("sometextfile.txt")
vec
vec <- strsplit(vec[2]," ")
vec <- as.numeric(vec[[1]])
vec

# readLines(), Reads entire lines
vec <- readLines("sometextfile.txt")
vec
vec <- strsplit(vec[2]," ")
```

```
vec <- as.numeric(vec[[1]])
vec
```

1.2.5 FILE

```
#File Connections can opne a file for reading different sections in different ways
# File syntax supports http, https, ftp
#Open the connection to the file
f1 <- file("sometextfile.txt", open="r")
#Scan for character, one line
df1 <- scan(f1,what="",nlines=1)
df1
#Scan for neumeric. for one line
df2 <- scan(f1,what=double(),nlines=1)
df2
#
df3 <- readLines(f1)
df3
close(f1)
```

1.2.6 DOWNLOAD AS PDF

```
#Download PDFs
pdf("myPlot.pdf")
x <- 1:50
y=log(x)
plot(x, y)
graphics.off()
```

1.2.7 WRITE

```
write.table(df,file="somefile1.txt",row.names = FALSE,col.names = TRUE,sep = ",")
write.csv()
write.csv2()
```

1.2.8 CAT,WRITELINE,SINK

```
#cat(),writeLine(),sink()

cat("Test file for cat\n",round(rnorm(5),3),"\n", file="cattest.txt")

lin <- c("Count down", paste(rev(1:10), collapse="-"),"Go")
```

```
writeLines(lin, con="writeLineTest.txt")


sink("sinktest.txt") # opne the open for the file sinktest.txt
x <- 1:5
y <- 1:3
outer(x,y) # output is sinked to the connection and not visible to the screen
sink() #Close the connection
```

1.2.9 DUMP,DPUT

```
#dump() and dput()
#if you wish to save the R objects instead of R o/p use dump fuction
x <- 1:3
y <- rpois(10,4)
dump(c("x","y"), file="dumptest.txt")
#L in the o/p file signifies number is an integer
#dump o/p will have value of x and y before the dump

#with dput fuction you don't need to write the definition but
#its merely an identifier
lis <- list(x=1:5,y=3,z=c("a","b","c"))
dput(lis, file="dputtest.txt")

#dget() inverse dput(), Note that the dget() commans below doesn't restore lis, but creates
#an object to lis, which can be assigned to other objects:
dget("dputtest.txt")

#using file Connections

f2 <- file("filetestout.txt", open="w")
cat("Header of file\n\n", file=f2)
mat <- matrix(round(rnorm(12),8),ncol=3)
write.table(mat,file=f2,row.names=FALSE,col.names=FALSE)
close(f2)

#Using append
write.table(mat,file=f2,row.names=FALSE,col.names=FALSE,append=TRUE)

#Working with binary files: Using save() and load()
#R also has its own internal binary format
#To save data and functions to it use EX;

x <- rnorm(3)
lis <- list(y=1:5,z="lalalal",fun=function()cat("ha-ha-ha-ha\n"))
save(x,lis, file="test1.RData")

#To read back into R simple use
```

1.3 CLEANING DATA

1.3.1 HANDLING NA

```
# Missing Values
# R uses the Special NA to code missing values
# Results of arithmetic involving NA's becomes NA as well
colMeans(airquality)
NA==NA

# Remove the NA
dataset2_3 <- na.omit(dataset2_3)

#Assign Zero to NA
df[is.na(df)] <- 0

#Assign Value for NA from different column
dataset1 <- dataset1 %>%
                mutate(Months.since.last.delinquent =
                ifelse(is.na(Months.since.last.delinquent),0,Months.since.last.delinquent))

#Assign Value for NA from different column
dataset5$Annual.Income <- ifelse(!is.na(dataset5$Annual.Income.x), dataset5$Annual.Income.x,
dataset5$Annual.Income.y)


# is.na is used to filter out the NA's
s <- subset(airquality, !is.na(Ozone))

# Note that the agrument na.rm=TRUE can be passed to most summary functions e.g. sum(),mean(),sd()
mean(airquality$Ozone, na.rm=TRUE)

#Assign zero to null values
df1[is.na(v_colname)] <- 0
```

1.3.2 RENAME THE COLUMNS

```
#Rename the column list in the dataframe.
v_colnames<-c("Colname1","colname2")
names(df) <- v_colnames

#Rename the few columns.
df <- rename(df,c(oldcol1="newcol1", oldcol2="newcol2"))
```

```
#Rename the variables
names(data)[names(data) == 'State'] <- 'cus_state'
names(data)[names(data) == 'State_Prov'] <- 'site_state'
```

1.3.3 ADDING OR DROPPING A COLUMN/ ROWNAME

```
#dropping columns by Column number
df1<- df1[c(-41)]

#dropping columns
dat$col2 <- NULL
dat

# Assign the rownames with some column name
rownames(Dis_tbl_overall) <- Dis_tbl_overall$State_tag
```

1.3.4 LOGICAL FILTERING

```
x <- (-5):5
x

x[4:8]
x[-c(1:3)]
x[-c(1:3,9:11)]

#logical vector can be defined by
index <- abs(x) <3
index

#use this vector to extract the unwanted data
x[index]
```

1.3.5 REMOVE THE DUPLICATES

```
# duplicate record removal
df <- arrange(non_con_tr, NASPID,Product_Name,Feature_Name,Geo_Site_ID,Last_Update)
df <- df[!duplicated(df[c("NASPID","Product_Name","Feature_Name","Geo_Site_ID")]), ]
```

1.3.6 MERGING

```
#merge the data
df1<-merge(x = df1, y = df2, by = "common_Colname_in_df1_df2", all.x = TRUE)

dataset5<-merge(x = dataset1, y = dataset4, by = c("Purpose","Term") , all.x = TRUE)
```

1.3.7 UPDATE / REPLACE

```
#replace or update Method1
x = c(3, 2, 1, 0, 4, 0)
replace(x, x==0, 1)

#replace or update Method2
df <- data.frame(Name=c('John Smith', 'John Smith', 'Jeff Smith'),
            State=c('MI','WI','WI'), stringsAsFactors=F)

df <- within(df, Name[Name == 'John Smith' & State == 'WI'] <- 'John Smith1')

#replace or update NA
DF$VAR3 <- ifelse(!is.na(DF$VAR1), DF$VAR1, DF$VAR2)

#Reassign values for categorical column
data$Product_Name_tag <- ifelse(data$Product_Name=="Private IP (PIP)","PIP",
                        ifelse(data$Product_Name=="Access","Access",
                            ifelse(data$Product_Name=="Internet Dedicated Services","IDS",
                                ifelse(data$Product_Name=="Private IP Gateways","PIG",
                                    ifelse(data$Product_Name=="Managed WAN","WAN",
                                        ifelse(data$Product_Name=="Verizon VoIP","VoIP","Others"))))))
```

1.3.8 CONTINOUS TO CATEGORICAL

```
df2_2$Annual.Income <- cut(df2_2$Annual.Income,
                    breaks=c(-Inf, 44830, 62302,86841, Inf),
                    labels=c("low","medium","high","SuperRich"))
```

1.3.9 DATE AND POSIX

```
# R has classes to handle dates, as.Date & as.POSIXct
# Date unit is days
# POSIX Unit is seconds

table(weekdays(df$date))
table(months(df$date))

# Convert dates to number
set.seed(449)
your.dates<-as.Date(sample(18000:20000,20), origin = "1960-01-01")
your.days<-c(julian(your.dates,origin=as.Date("1960-01-01")))
```

```
# get month, day, year
library("chron")
set.seed(449)
your.dates<-as.Date(sample(18000:20000,20), origin = "1960-01-01")
my.days.structure<-month.day.year(your.dates)
my.days.structure<-month.day.year(your.dates,origin=c(1,1,1960))

# get month, day, year
set.seed(119)
my.days<-sample(18000:20000,20)
my.days.structure<-month.day.year(my.days)
my.days.structure<-month.day.year(my.days,origin=c(1,1,1960))
my.days.structure
my.days

# Extract date info in DF
set.seed(119)
my.dates<-sample(18000:20000,20)
my.days.structure<-month.day.year(my.dates,origin=c(1,1,1960))
my.dates<-as.Date(my.days, origin = "1960-01-01")
my.dates
my.date.info<-data.frame(Weekday=weekdays(my.dates),my.days.structure)
my.date.info

#Convert character into date
non_con_tr$Last_Update1 <- strptime(x = as.character(non_con_tr$Last_Update),format = "%m/%d/%Y
%H:%M")
```

1.4 DATA MUNCHING

```
# subset the data Unique
df1<- unique(df1)
df <- unique(df_main$Man.trans.avail,df_main$Cylinders)
df <- as.data.frame(df)
```

1.4.1 SUBSET BY ROW AND COLUMN

```
#subset the data by column
vcolname <- c("RPM","Passengers","Weight")
df <- df_main[vcolname]
df
#subset the data by column
df <- df_main[,c("RPM","Passengers","Weight")]
df

#subset the data by rows & Columns
```

```
df <- subset(df_main,Origin=="USA", select=c(RPM,Passengers,Weight))
df
```

1.4.2 LOGICAL INDXING

```
#Logical indexing applies to DF
datA <- airquality[airquality$Temp>80,c("Ozone","Temp")]
```

1.4.3 TAPPLY & TABLE

```
dat <- data.frame(
gender=c("male","male","male","male","male","female","female","female","female","female"),height=c(10,5,
12,10,2,7,6,12,9,4))
dat
tapply(dat$height,dat$gender,mean)

# count of categorical variable
table(df_main$Passengers)

# count of 2 categorical variable Kind of truth table
table(df_main$Passengers,df_main$Cars)
```

1.4.4 SPLIT

```
#Split vectors into groups
g <- split(Cars93$MPG.city,Cars93$Origin)
g
g1 <- split(Cars93$MPG.city,Cars93$Cylinders)
g1
typeof(g)
median(g[[1]])
median(g[[2]])
```

1.4.5 SORT

```
df1<-df1[order(df1$v_colname),] # Ascending
df1<-df1[order(-df1$v_colname),] # descending
```

1.4.6 ACAST

```
# reshaping the data with acast.
```

```
attach(data_2)
table_5<-aggregate(LIBI_TotalDiscount ~ State_tag+Product_Name_tag, data = data_2, sum)
table_6<-acast(table_5, State_tag~Product_Name_tag,margins = c("State_tag", "Product_Name_tag"))
table_6<-acast(table_5, State_tag~Product_Name_tag,sum)
table_6<-as.data.frame(table_6)
```

1.4.7 PIVOTING –MELT AND DCAST

```
names(airquality) <- tolower(names(airquality))
head(airquality)

aql2 <- melt(airquality)
head(aql2)

aql1 <- melt(airquality, id.vars = c("month", "day"))
head(aql1)

aql3 <- melt(airquality, id.vars = c("month", "day"),
        variable.name = "climate_variable",
        value.name = "climate_value")
head(aql3)

aql <- melt(airquality, id.vars = c("month", "day"))
head(aql)
unique(aql$variable)

aqw <- dcast(aql, month + day ~ variable)
#df5 <- dcast(df4.1, Customer_Name+Opportunity_ID+Scenario_ID+Geo_Site_ID+Contracted+Term ~
Access_Provider)
head(aqw)

table(df$A,df$B) # A will be rows, B will be columns
```

1.4.8 DPLYR

```
library(dplyr)

#http://genomicsclass.github.io/book/pages/dplyr_tutorial.html

#dplyr_verbs        Description
#select()     select columns
#filter()      filter rows
#arrange()           re-order or arrange rows
#mutate()             create new columns
#summarise()       summarise values
#group_by()          allows for group operations in the "split-apply-combine" concept
#distinct()    find the distinct values
#sample_frac    get the sample of some percentage
```

```
#sample_n      get the sample of n rows

# Distinct
df3 <- distinct(df_main, Origin)
df3

# sample_frac
df3 <- sample_frac(df_main, 0.5)
df3

# sample_n
df3 <- sample_n(df_main, 25)
df3

#Filter Column
df2 <- filter(df_main, Origin=="USA")
df2

#Column selection
df2 <- dplyr :: select(df2, Manufacturer) # Since we are using data from MASS library to avoid the conflit we
use  dplyr ::
df2 <- dplyr :: select(df_main, Model:DriveTrain)
df2

#Creating additional column
df2 <- mutate(df_main, Total.Price=Min.Price+Max.Price)
str(df2)

#Summarise
df2 <- df_main %>%
        summarise(avg_price = mean(Price),
        min_Price = min(Price),
        max_Price = max(Price),
        total = n())
df2

#grouping and Summary
df3 <- df_main %>%
 group_by(Origin) %>%
 summarise(total_Price = sum(Price),
        avg_price = mean(Price),
        min_Price = min(Price),
        max_Price = max(Price),
        total = n())
df3
```

1.4.9 GROUPING AND SUMMARY (APPLY FAMILY)

#apply(matrix, 1/2, f): input is a matrix. output is a vector, where element i is f(row/col i of the matrix)
#lapply Loop through each item of a list or a vector and execute a function on each item. Outputs a list of the same length as the input. You can input a data frame as well; if you think of it as a list of column vectors.
#sapply(vec, f): input is a vector. output is a vector/matrix, where element i is f(vec[i]) [giving you a matrix if f has a multi-element output]
#lapply(vec, f): same as sapply, but output is a list?
#tapply(vector, grouping, f): output is a matrix/array, where an element in the matrix/array is the value of f at a grouping g of the vector, and g gets pushed to the row/col names
#mapply Same as lapply, but instead of looping through each item in a single vector/list, it loops through each item of multiple vectors/lists in tandem. Runs a command on the first item in vector1 and vector2, then second item of vector1 and vector2, etc. Therefore the two vectors or lists have to be of the same length.
#by(dataframe, grouping, f): let g be a grouping. apply f to each column of the group/dataframe. pretty print the grouping and the value of f at each column.
#aggregate(matrix, grouping, f): similar to by, but instead of pretty printing the output, aggregate sticks everything into a dataframe

```
my.data <- data.frame(data1=rnorm(10),data2=rnorm(10),data3=rnorm(10))
my.data

apply(my.data,1, sum)
lapply(my.data, sum)
sapply(my.data, sum)

#Applying a Function to Every Row
apply(df,1,mean) # at row level
apply(df,2,mean) # at column level

#calling user defined fuction,check how many records has negative value
apply(df, 1, function(x) length(x[x<0])) # at row level
apply(df, 2, function(x) length(x[x<0])) # at row level

#Applying a Function to each List element
#lapply & sapply funcations used based on return type list or Vector(if possible).
#It will call function once for every element in the list and store the result in list or vector.


lapply(df,mean)
#lapply(df,length(unique(df)))

sapply(df,length)
#sapply(1:3, function(x) x^2)
#lapply(1:3, function(x) x^2)

#if the called function returns vector, sapply will form the results to matrix.
# there are many ways to avoid the loop in R few of them are discussed below,
# try to sync up the object oriented

#lappy & sapply
#1.Functions which apply fucntions to Vectors, matrices,arrays or lists.
```

```
#2.lapply & sapply function apply functions to the input
#3. lapply o/p is list where as sapply() is vector or a matrix

# create a matrix of 10 rows x 2 columns
m <- matrix(c(1:10, 11:20), nrow = 10, ncol = 2)

# mean of the rows
apply(m, 1, mean)

# mean of the columns
apply(m, 2, mean)

# divide all values by 2
apply(m, 1:2, function(x) x/2)
apply(m, 2:1, function(x) x/2)
attach(iris)
head(iris)

# get the mean of the first 4 variables, by species
by(iris[, 1:4], Species, colMeans)

#tapply() applies a function or operation on subset of the vector broken down by a given factor variable.
mtcars
tapply(mtcars$mpg, list(mtcars$cyl), mean)
tapply(mtcars$mpg, list(mtcars$cyl, mtcars$am), mean)
tapply(mtcars$mpg, list(mtcars$cyl, mtcars$am, mtcars$vs), mean)

A <- matrix(1:9, nrow=3);
B <- matrix(1:16, nrow=4);
C <- matrix(1:8, nrow=2);
my.list <- list(A=A,B=B,C=C)
my.list

#Now suppose you to extract the second coulmn of these matrices

# We can't do it with sapply fuction.
lapply(my.list,"[",1,2) # We use the bracket function to extract the 1st row, 2nd column
lapply(my.list,"[",,2) # We use the bracket function to extract the 2nd column

# Another example
my.summary <- function(x){
  data.frame(Min=min(x,na.rm=TRUE),
             Median=median(x,na.rm=TRUE),
             Mean=mean(x,na.rm=TRUE),
             Max=max(x,na.rm=TRUE)
  )
}
sapply(airquality,my.summary)
```

```
# apply function within a group (tapply(),aggregate(),by())
dat <- data.frame(
gender=c("male","male","male","male","male","female","female","female","female","female"),height=c(10,5,
12,10,2,7,6,12,9,4))
dat

# tapply is ten time faster than aggregate function
tapply(dat$height,dat$gender,mean) # subset corresponds to the levels of the 2nd argument.
aggregate(height~gender,data=dat, mean) #aggregate~height and groupby~gender
aggregate(height~gender+col1,data=dat, mean) #aggregate~height and groupby~gender & col1
by(dat$height,dat$gender, mean)
```

1.4.10 TEXT ANALYSIS

```
#Text Analysis
# grep() & grepl() text Pattern search
# Sub() text pattern search and replace first occurrence
# gsub() text pattern search and replace all occurance.
# By default pattern is a regular expression, fixed = FALSE
# By default matching is case sensitive, ignore.case = FALSE

txt <- c("Hello my",
      "name is",
      "kumar")

grep("name",txt)
grepl("name",txt) #logical indexs

sub("kumar","KUMAR",txt)

#?Check for regular expressions
```

1.4.11 GROUPING AND SUMMARY (TABLE FAMILY)

```
# Tabulating data: table() and xtabs()
# Functions used for tabulating cross-referenced data.
# Also prop.table(), margin.table(), addmargins(),ftable()
# table function ignores missing values

# Creates an N-way contingency table of counts from categorical variables.
table(airquality$Ozone > 80, airquality$Month)
my.table <- with(airquality,table(OzHi = Ozone > 80, Month))
my.table
airquality

# table fucntion is used purely for frequency calcualtion
```

```
table(airquality$Month,airquality$Temp)
tapply(airquality$Month,airquality$Temp,length) # compare tapply
table(airquality$Month)

my.table.2 <- addmargins(my.table,1:2)
my.table.2

my.table.3 <- prop.table(my.table,1)
my.table.3
my.table.31 <- addmargins(my.table.3,1)
my.table.31

# Converting to percentages
round(my.table.31*100)

#xtabs
df <- as.data.frame(UCBAdmissions)
head(df)

mytable <- xtabs(Freq ~ Gender+Admit+Dept, data=df)
mytable
ftable(mytable) #flattens the table
#mytable <- xtabs(Freq ~ Gender+Dept+Admit, data=df)

# Data for department A can be extrated as
mytable
DepA <- mytable[,,1]
DepA
ftable(DepA)

# Gender Vs Admission
margin.table(mytable,1:3)
margin.table(mytable,1:2)
# Converting into frequencies
prop.table(margin.table(mytable,1:2),1)
prop.table(margin.table(mytable,2:3),1)
prop.table(margin.table(mytable,c(1,3)),1)

# grouping data with aggregate
attach(df1)
v_table1<-aggregate(V_cost ~ V_cutomername,V_productname, data = df1, sum)

#grouping data with table,margin.table,prop.table
attach(df1)
v_table1<-table(V_cutomername,V_productname)
v_table1<-margin.table(v_table1,1) #rowwise frequency
v_table1<-margin.table(v_table1,2)#columnwise frequency
v_table1<-data.frame(table_2)
v_table1<-prop.table(v_table1,1) #rowwise Percentage
```

```
v_table1<-prop.table(v_table1,2)#columnwise Percentage
v_table1<-data.frame(table_2)
detach(df1)
```

1.4.12 FUNCTIONS

```
#User generate fuctions

kum_sqrt <- function(x){
  return(x*x)
}

kum_sqrt(1:5)
```

1.4.13 FORLOOP

```
?cat
# it simply converts arguments to characters and concatenates so you can think of something like
as.character() %>% paste().
# cat invisibly returns NULL while print returns its argument.

#For loop, can sequence through list, matrix, df
for(i in 1:3){
  cat(i,"+",i,"=",i+i,"\n")
}

df <- data.frame(a = 1:2, b=2:3)
df
# Print the sum of all the columns
for(x in df){
  cat("columnsum:",sum(x),"\n")
}
```

1.4.14 WHILELOOP

```
#example1
t <- 0 # number of big part (>2)
y <- abs(rnorm(1000)) # simulated part size
i <- 0 #index of parts
count <- 0
#y
hist(rnorm(1000))
while(t<30 & i<1000)
{
```

```
  i <-i+1
  count <- count+1
  temp <- y[i]
  t <- t+(temp >2)
  #k <- K+count
  print(count)
  print(temp)
  print(t)

}
#

#example2
eye.colors <- c("brown","blue","greeen","yellow","grey")
eyecolor <- data.frame(personalId=1:100,color=sample(eye.colors,100,rep=T))
#eyecolor
list.of.ids <- numeric(0)  #patient ID list
#list.of.ids
repeat {
  i <- i+1
  if(eyecolor$color[i]=="yellow"|eyecolor$color[i]=="blue") next
  list.of.ids <- c(list.of.ids,eyecolor$personalId[i])
  #u <- 1+2

  print(i)
  print(list.of.ids)

  if(i==100|length(list.of.ids)==20) break
}


# Loop in R are less efficient than C++ etc, hence use the internal objects where ever possible

y <- matrix(rnorm(1000000), nrow=1000)
#y
#remove(list=ls())
z <- 0*y
#z
time1 <- as.numeric(Sys.time())
Sys.time()
for(i in 1:1000) {
  for(j in 1:1000){

    z[i,j] <- y[i,j]^2
  }
}
time2 <- as.numeric(Sys.time())
Sys.time()
```

```
Sys.time()
#time3 <- as.numeric(Sys.time())
z1 <- y^2
Sys.time()
time3 <- as.numeric(Sys.time())

(time2 - time1) /(time3- time2)
```

1.4.15 ATTACH AND DETACHING

```
# Attaching and detaching data
# You don't need to have the data frame name mentioned again and again
dat <- data.frame(
gender=c("male","male","male","male","male","female","female","female","female","female"),height=c(10,5,
12,10,2,7,6,12,9,4))
dat
tapply(dat$height,dat$gender,mean)
attach(dat)
tapply(height, gender, mean)
detach(dat)

# Similar identifiers in the R memory OVERRIDES adding an R Object to the R search path with attach()
x1 <- 1:3 # run below script without this
my.data <- data.frame(x1=4:6,x2=7:9)
attach(my.data)
# You cann't refer to x1 in my.data its maked in x1 in R memory
cbind(x1,x2)

# Adding a new object OVERRIDES the previous addition to the search path
my.data2 <- data.frame(x1=10:12,x2=13:15)
attach(my.data2)
cbind(x1,x2)

# This happen because, an attached object is placed on top of R's search path, but below the Global R Memory
#you can see the hierachy of R searches for identifiers at any time with the "searchpaths() function"


searchpaths()[1:3]
#R takes x1 from the Global Environment(the R Memory), and x2 from my.data2
detach(my.data2,my.data)

#With() function
x1 <- 1:3
my.data <- data.frame(x1=4:6,x2=7:9)
my.data2 <- data.frame(x1=10:12,x2=13:15)
attach(my.data)
attach(my.data2)
#with() temporarily puts the first argument in the top of R's search hierarchy
```

```
sum.and.diff <- with(my.data,cbind(x1+x2,x1-x2))
sum.and.diff
cbind(x1+x2,x1-x2)
```

1.5 VISUALIZATION

1.5.1 R SHINY

https://www.linkedin.com/pulse/building-r-shiny-applications-tianwei-zhang-2/?lipi=urn%3Ali%3Apage%3Ad_flagship3_profile_view_base%3ByjmFAWehSDO7fWrNLvpzpQ%3D%3D

1.5.2 BASIC CHART

```
x <- rnorm(100)
y <- rnorm(100)
plot(x)  # Default Scatter plot
plot(x,y) # Default Scatter plot
plot(x, type='l') # line plot
plot(x, type='p') # point Scatter
plot(x, type='b') # both above Scatter(line & scatter)
plot(x, type='h') # point bar
plot(x, type='s') # step type

# Boxplot of MPG by Car Cylinders
boxplot(mpg~cyl,data=mtcars, main="Car Milage Data",
      xlab="Number of Cylinders", ylab="Miles Per Gallon")
#Sort
plot(sort(mtcars$mpg),mtcars$cyl, type='l',xlab="x Axis", ylab="y Axis", main="Line Plot",
col='red',font.main=1)
```

1.5.3 XYPLOT

```
#####Conditional plotting
str(ChickWeight)

xyplot(mpg~wt+gear | factor(cyl) + factor(am,labels=c("A","M")), #
     data=mtcars, main="MPG vs Wt",
     xlab="Wt/1,000", ylab="MPG",pch=19,type=c("p","g"))
```

```
histogram(~weight|factor(Diet),data=ChickWeight,
        main="Weight by Diet Type",xlab="Weight in grams")

bwplot(~weight|factor(Time),data=ChickWeight,col="blue",
      main="Weight by Days Since Birth",xlab="Weight in grams")

coplot(mpg~wt|factor(cyl)+factor(am),data=mtcars)
```

1.5.4 MATRIX SCATTER PLOT

```
#####Matrix scatter plot
str(swiss)

pairs(~ Fertility + Education + Catholic, data = swiss,
     subset = Education < 20, main = "Swiss data, Education < 20")
pairs(~ BI.Speed + Customer.Segment+LI.BI...Total.Revenue  , data = data_1,
     subset =  Product.Name == "Access",main = "HELL")
```

1.5.5 HEAT CHART

```
ggplot(data = con_tr, aes(x = Access_Speed, y = site_state_tag)) +
geom_tile(aes(fill = Total_Revenue))
```

1.5.6 SCATTER PLOT BY FUNCTION

```
vis_simple <- function(col1, df=data_1, col2="LI.BI...Total.Revenue"){
  require(ggplot2)
  title=paste("plot of",col1,"Vs",col2)
  ggplot(df,aes_string(col1,col2))+
    geom_point()+
    ggtitle(title)
}

plot.cols <- c("BI.Technology",
             "LI.BI...Total.Discount",
             "LI.BI...Total.Cost")


lapply(plot.cols,vis_simple) #plot.cols
```

1.5.7 SCATTER PLOT FOR OUTLIER

```
##### Scatter for oulier (pending)
vis_outlier <- function(col1="LI.BI...Total.Discount"){
  require(ggplot2)
  title=paste("plot of",col1,"Vs LI.BI...Total.Discount")
  ggplot(data_1,aes_string(col1,"LI.BI...Total.Revenue"))+
    geom_point(aes(color=BI.Technology,
                   alpa=.5,size=4))+
    ggtitle(title)
}
plot.cols <- c("BI.Technology",
               "LI.BI...Total.Discount",
               "LI.BI...Total.Cost")
lapply(plot.cols,vis_outlier)
```

1.5.8 SINE WAVE

```
# plot a sine wave 1
kvec <- (0:99)
xk <- sin(2*pi*0.1*kvec)
xk
plot(kvec,xk)
lines(kvec, xk, col="red",lwd=1)

# plot a sine wave 2
kvec <- (0:99)
xk1 <- sin(2*pi*0.1*kvec)
xk2 <- .02*kvec
plot(sort(kvec),xk1,type='b',pch=19,ylim=c(-1,2.5),xlab="Time", ylab="Signal",col="blue")
lines(kvec, xk2, col="red",lty=2,lwd=2)
legstr <- c("Sine","Line")
legend(x= 0,y = 2.4, legend = c("sine","line"),col =
c("blue","red"),lty=1:2,pch=c(19,NA),lwd=1:2,text.width=1.2*max(strwidth("Line Types")),cex = 0.9, y.intersp =
1.3, title = "Line Types")
```

1.6 GGPLOT2

ggplots is based on the grammar of graphics, the idea that you can build the graph from the same few components like

1. data , data set
2. geoms , Visual marks that represent data points, to display data values, map variables in the data setto aesthetic properties of the geom like Size,color and X,Y locations.

```
library("ggplot2")
```

```
Cars93
attach(Cars93)
#detach(Cars93)

# Sample Plot
ggplot(mpg,aes(hwy,cty)) +
  geom_point(aes(colour =cyl)) +
  geom_smooth(method = 'lm') +
  coord_cartesian() +
  scale_color_gradient() +
  theme_bw()

Labels:
ggtitle("Title of  the Chart")
xlab("X axis")
ylab("Y axis")
OR
labs(title="New Title", x="X axis", y="Y axis")

#returns the last plot
last_plot()

# saves last plot by file named "plot.png"
ggsave("plot.png",width=5,height=5)
```

1.6.1 LEGENDS

```
#place legend at bottom,top,left or right
theme(legend.position="bottom")

#Set legend type for each aesthic: colorbar,legend or none (no legend)
guides(color = "none")

#Set legends title and labels with a scale function
scale_fill_discrete(name="Title", labels=c("A","B","C"))
```

1.6.2 FACETING

Facets divide a plot into subplots based on values of one or more discrete variables.

```
# Facet into columns based on fl
ggplot(mpg,aes(cty,hwy)) +
  geom_point() +
  facet_grid(. ~ fl)

# Facet into rows based on year
ggplot(mpg,aes(cty,hwy)) +
```

```
  geom_point() +
  facet_grid(year ~ .)

# Facet into both rows and column based on year
ggplot(mpg,aes(cty,hwy)) +
  geom_point() +
  facet_grid(year ~ fl)

# wraps Facet into rectangular layout
ggplot(mpg,aes(cty,hwy)) +
  geom_point() +
  facet_grid(~fl)

#Scales to letaxis limits vary across facets
ggplot(mpg,aes(cty,hwy)) +
  geom_point() +
  facet_grid(y ~ x,scales = "free")
```

1.6.3 SCALE - SIZE

```
# size of the scatter plot
ggplot(mpg,aes(cty,hwy)) +
  geom_point() +
  geom_point(aes(size = cyl))

# Size,Value mapped to area of circle(not radius)
ggplot(mpg,aes(cty,hwy)) +
  geom_point() +
  geom_point(aes(size = cyl)) +
  scale_size_area(max = 6)
```

1.6.4 SCALE - SHAPE

```
#Shape Scales 1
ggplot(mpg,aes(cty,hwy)) +
  geom_point() +
  geom_point(aes(shape = fl))

#Shape Scales 2
ggplot(mpg,aes(cty,hwy)) +
  geom_point() +
  geom_point(aes(shape = fl)) +
  scale_shape(solid=FALSE)
```

1.6.5 SCALE –COLOUR AND FILL

```
# discrete
ggplot(mpg,aes(fl)) +
 geom_bar(aes(fill=fl))

# discrete For palette choices
ggplot(mpg,aes(fl)) +
geom_bar(aes(fill=fl)) +
scale_fill_brewer(palette='Blues')

# discrete fill grey
ggplot(mpg,aes(fl)) +
geom_bar(aes(fill=fl)) +
scale_fill_grey(start=0.2,end=0.8,na.value="red")

# continous
ggplot(mpg,aes(hwy)) +
geom_dotplot(aes(fill= ..x..))

# continous
ggplot(mpg,aes(hwy)) +
geom_dotplot(aes(fill= ..x..))
scale_fill_gradient(low ="red",high="yellow")

# continous
ggplot(mpg,aes(hwy)) +
geom_dotplot(aes(fill= ..x..)) +
scale_fill_gradient2(low ="red",high="blue", mid="white",midpoint=25)
```

Coordinate Systems:

1.6.6 COORDINATE SYSTEMS

```
# xlim,ylim
ggplot(mpg,aes(fl)) +
  geom_bar() +
  coord_cartesian(xlim=c(0,5))

# Cartesian coordinates with fixed aspect ration between x and y units
ggplot(mpg,aes(fl)) +
  geom_bar() +
  coord_fixed(ratio=1/2)

# coord_flip
ggplot(mpg,aes(fl)) +
  geom_bar() +
  coord_flip()
```

```
# coord_polar
ggplot(mpg,aes(fl)) +
  geom_bar() +
  coord_polar(theta="x",direction=1)

# transformed cartesian coordinates
ggplot(mpg,aes(fl)) +
  geom_bar() +
  coord_trans(ytrans="sqrt")
```

2. MACHINE LEARNING

keydifferences between Correlation and covariance
http://keydifferences.com/difference-between-covariance-and-correlation.html

2.1 TEXT ANALYTICS

```
setwd("D:/R Analytics/IIT Hydrabed/July2016/Data sets/8-text analytics")
library(e1071)
library(textir)
library(tm)
library(VGAM)


d = read.csv("documents.csv", stringsAsFactors=FALSE)
d = d[,3]

myCorpus <- Corpus(VectorSource(d))

library(e1071)
library(textir)
library(tm)
library(VGAM)


# convert to lower case
myCorpus <- tm_map(myCorpus, tolower)

# remove punctuation
myCorpus <- tm_map(myCorpus, removePunctuation)

# remove numbers
myCorpus <- tm_map(myCorpus, removeNumbers)

#Stemming
```

```
myCorpus <- tm_map(myCorpus,stemDocument)

# remove stop words
myCorpus <- tm_map(myCorpus, removeWords, stopwords("english"))

# remove extra whitespace
myCorpus <- tm_map(myCorpus, stripWhitespace)

myCorpus<- tm_map(myCorpus, PlainTextDocument)

#In the above code, tm_map() is an interface to apply transformations (mappings) to corpora. A
#list of available transformations can be obtained with getTransformations(), and the mostly used
#ones are  PlainTextDocument(), removeNumbers(), removePunctuation(), stemDocument() and
stripWhitespace().


#Converting back to data frame
res = data.frame(text = sapply(myCorpus, as.character), stringsAsFactors = FALSE)

#Either weightTf or weightTfIdf for weighting.
#wighttf gives a DocumentTermMatrix in term frequency format

myTdm<- DocumentTermMatrix(myCorpus, control = list(weighting = weightTf, minWordLength=4))
dim(myTdm)
inspect(myTdm[1:5,1:20])

#  Start by removing sparse terms:
#Sparse  =  0.99 will remove only terms that are more sparse than 0.99.
#You will retain all terms for which dfj > N*(1-0.99), where N is the number of documents.
#What to keep in mind, however, is that unusual words may be very important in terms of what the content
means

myTdm <- removeSparseTerms(myTdm, 0.98)
dim(myTdm)
inspect(myTdm[1:5,1:20])

#Converting to matrix
temp = as.matrix(myTdm)


#rmlist=ls()
#ls()


rowcount= nrow(temp)
colcount  = ncol(temp)

for(i in 1:rowcount){
  t = sum(temp[i,])
```

```
 if(t>0){
   rowmean = mean(temp[i,])
   rowsd     = sd(temp[i,])*sqrt(colcount-1)
   temp[i,] = (temp[i,]-rowmean)/rowsd
 }
}

requiredperson =4661 #Obama
dotproducts = numeric(rowcount)

for( i in 1:rowcount){
 dotproducts[i] =  sum(temp[requiredperson,]*temp[i,])
}

ordering = order(dotproducts)
ordering

requiredperson


#clustering, use TfIdf and normalise the coloumns

myTdm<- DocumentTermMatrix(myCorpus, control = list(weighting = weightTfIdf, minWordLength=4))
dim(myTdm)
myTdm <- removeSparseTerms(myTdm, 0.90)
dim(myTdm)


temp = as.matrix(myTdm)
rowcount= nrow(temp)
colcount  = ncol(temp)

for(i in 1:colcount){
 colsum = sum(temp[,i])
 if(colsum !=0){
   colmean = mean(temp[,i])
   colsd      = sd(temp[,i])*sqrt(rowcount-1)
   temp[,i] = (temp[,i]-colmean)/colsd
 }
}

grpersons <- kmeans(temp, centers=25, nstart=3)
grpersons$cluster
grpersons


opinions = read.csv("amazon.csv",stringsAsFactors=FALSE)
nrow(opinions)
```

```
ds <- DataframeSource(as.data.frame(opinions[,2]))
myCorpus<-Corpus(ds)
inspect(myCorpus[1])

# convert to lower case
myCorpus <- tm_map(myCorpus, tolower)

# remove punctuation
myCorpus <- tm_map(myCorpus, removePunctuation)

# remove numbers
myCorpus <- tm_map(myCorpus, removeNumbers)

#Stemming
myCorpus <- tm_map(myCorpus,stemDocument)

# remove stop words
myCorpus <- tm_map(myCorpus, removeWords, stopwords("english"))

# remove extra whitespace
myCorpus <- tm_map(myCorpus, stripWhitespace)

myCorpus<- tm_map(myCorpus, PlainTextDocument)

myTdm<- DocumentTermMatrix(myCorpus, control = list(weighting = weightTf, stopwords = TRUE,
minWordLength=2))
dim(myTdm)
myTdm <- removeSparseTerms(myTdm, 0.90)
dim(myTdm)

temp = as.matrix(myTdm)

trainset = sample(1:nrow(temp), trunc(0.7*nrow(temp)))

classifier = naiveBayes(temp[trainset, ], as.factor(opinions[trainset, 3]))

trainpredicted = predict(classifier,temp[trainset, ])
table(trainpredicted,opinions[trainset, 3])

testpredicted = predict(classifier,temp[-trainset, ])
table(testpredicted,opinions[-trainset, 3])


myTdm<- DocumentTermMatrix(myCorpus, control = list(weighting = weightTf, stopwords = TRUE,
minWordLength=2))
dim(myTdm)
myTdm <- removeSparseTerms(myTdm, 0.99)
dim(myTdm)
```

```
temp = as.matrix(myTdm)

trainset = sample(1:nrow(temp), trunc(0.7*nrow(temp)))

classifier = naiveBayes(temp[trainset, ], as.factor(opinions[trainset, 3]))

trainpredicted = predict(classifier,temp[trainset, ])
table(trainpredicted,opinions[trainset, 3])

testpredicted = predict(classifier,temp[-trainset, ])
table(testpredicted,opinions[-trainset, 3])
```

2.2 LDA

```
# LDA works when the measurements made on independent variables for each observation are continuous quantities.
# LDA explicitly attempts to model the difference between the classes of data.
# PCA on the other hand does not take into account any difference in class.
# LDA is closely related to analysis of variance (ANOVA) and regression analysis.
# Which also attempt to express one dependent variable as a linear combination of other features or measurements.
# LDA makes some simplifying assumptions about your data:
  # That your data is Gaussian, that each variable is is shaped like a bell curve when plotted.
  # That each attribute has the same variance, that values of each variable vary around the mean by the same amount on average.
# With these assumptions, the LDA model estimates the mean and variance from your data for each class.
# This might go without saying, but LDA is intended for classification problems where the output variable is categorical. LDA supports both binary and multi-class classification.
#http://courses.cs.tamu.edu/rgutier/cs790_w02/l6.pdf

library(MASS)
iris
i1 = iris
str(i1)
plot(i1$Petal.Length,i1$Petal.Width,col=i1$Species)

# Load the library caTools
library(caTools)

# Randomly split the data into training and testing sets
set.seed(1000)
split = sample.split(i1$Species, SplitRatio = 0.65)

# Split up the data using subset
train = subset(i1, split==TRUE)
test = subset(i1, split==FALSE)
```

```
ldamodel = lda(Species~.,data=train)
plot(ldamodel)
summary(ldamodel)
table(train$Species,predict(ldamodel)$class)

table(test$Species, predict(ldamodel,newdata=test)$class)
```

2.3 PCA WITH DECISION TREE

```
library(rpart)
library(rpart.plot)
setwd("H:/Learning/Analytics/Karthik")
wine <- read.table("wine.data",sep=",")
str(wine)
wine$V1 = as.factor(wine$V1)
colnames(wine)[1] = 'WineType'

scaledwine <- as.data.frame(scale(wine[2:14]))
str(scaledwine)
head(scaledwine,3)
head(wine,3)
# call PCA
wine_pca <- prcomp(scaledwine)
summary(wine_pca)
# look for 80% cumulative variance
# screeplot(wine_pca,type='lines')
wine_pca$rotation # Gives all rotation - gives all PCs
wine_pca$rotation[,1] # display PC 1
wine_pca$rotation[,2] # PC2
new_df = as.data.frame( wine_pca$x [,1:4])
new_df$WineType = wine$WineType
str(new_df)
wine$V1

#Decision Tree
dtmod = rpart(WineType ~ PC1 + PC2 + PC3 +PC4,data=new_df,method = 'class')
prp(dtmod)
table(new_df$WineType , predict(dtmod,type='class'))
tapply(wine[,2],wine$WineType,mean)
```

2.4 KMEANS CLUSTERING

```
irisnew = iris
irisnew$Species=NULL
irisnew
```

```
#Scale
irisnew = scale (irisnew)
kc = kmeans(irisnew, 3)
### Alternative
kc = kmeans(iris[,1:4], 3)
kc

table(kc$cluster,iris$Species)
kc$size
kc$cluster
kc$centers

library(cluster)
clusplot(irisnew, irisnew$cluster, color=TRUE, shade=TRUE,labels=2, lines=0)

# Centroid Plot against 1st 2 discriminant functions
library(fpc)
plotcluster(irisnew, kc$cluster)

###Hierarichical ###
data(iris)
irisnew=iris
irisnew$Species = NULL
#row.names(irisnew) = iris$Species

d = dist(irisnew, method = "euclidean")
fit = hclust(d, method = "ward.D")
plot(fit)
rect.hclust(fit, k=3, border="red")

groups = cutree(fit, k=3)
groups
table(iris$Species,groups)
```

2.4.1 CLUSTERING

```
library(cluster)

food = read.csv("protein.csv")
foodagg=agnes(food[,-1],diss=FALSE,metric="euclidian", method="complete")
plot(foodagg)

cutree(foodagg,k=5)

cutree(foodagg,k=3)
```

2.5 LOGISTIC REGRESSION

Limitations of Logistic Regression
#Two-Class Problems. Logistic regression is intended for two-class or binary classification problems. It can be extended for multi-class classification, but is rarely used for this purpose.
#Unstable With Well Separated Classes. Logistic regression can become unstable when the classes are well separated.
#Unstable With Few Examples. Logistic regression can become unstable when there are few examples from which to estimate the parameters.

```
setwd("H:/Learning/Analytics/Karthik")
# Read in the dataset

bankloan <- read.csv('bankloan.csv')
head(bankloan,10)

splitvar = sample.split(bankloan$LoanApproval , SplitRatio = .7)
train = bankloan[ splitvar == TRUE , ]
test = bankloan[ splitvar == FALSE , ]
### Alternate train = subset(bankloan,splitvar==TRUE)

logmod = glm(LoanApproval ~ Gender + Income + CIBILScore,data=train,family='binomial')
summary(logmod)

# Get the predicted probabilities for each loan application
# There are methods to do it, Method 1
bankloan_prob = predict(logmod,type='response')
### Create the truth table
tt = table(bankloan$LoanApproval , bankloan_prob  > .5)
# Alternate methos
tt = table(train$LoanApproval,trainmod$fitted.values > .5)
print(tt)
paste('Accuracy of Model =',sum(diag(tt)) / sum(tt) * 100, '%')
paste('Sensitivity of Model =', round(tt[2,2] / sum(tt[2,]),4)*100,'%')
paste('Specificity of Model =', round(tt[1,1] / sum(tt[1,]),4)*100,'%')


### Lets plot proabilities and colour it with actual Approvals/Rejects
plot(logmod$fitted.values,col=bankloan$LoanApproval,pch=17)
abline(h=0.5)
#### Predict if the loan applicant is a Female with Income=5100 and Cibil = 450
# Create Prediction Data frame
bankloanpred = data.frame(Gender = 'Female',Income = 5100,CIBILScore = 450)
bankloanpred
## Predict for new data using the model created above
prednew = predict(logmod,type='response',newdata=bankloanpred)
prednew
### Check the probability and identify approve or reject
ifelse(prednew > .5,'Approved','Rejected')
```

```
# Validate with Test Data
predictTest = predict(trainmod,type='response',newdata=test)

### Compare with test Loan Approvals
tt = table(test$LoanApproval,predictTest > .5)
tt
paste('Accuracy of Test Model =',sum(diag(tt)) / sum(tt) * 100, '%')
paste('Sensitivity of Test Model =', round(tt[2,2] / sum(tt[2,]),4)*100,'%')
paste('Specificity of Test Model =', round(tt[1,1] / sum(tt[1,]),4)*100,'%')
```

2.6 SVM

```
library(e1071)
#SVMs were developed by Cortes & Vapnik (1995) for binary classification
#SVM is a supervised machine learning algorithm which can be used for classification or regression problems
#It does Linear and Non-Linear
#4 types of Kernel processes for classification – Linear, Polynomial, Radial Basis Function, Sigmoid
#The complex data transformations and resulting boundary plane are very difficult to interpret. This is why it's
often called a black box.
#Transform data to the format of an SVM package
#SVM requires that each data instance is represented as a vector of real numbers and hence convert
categorical to dummy variables
#Conduct simple scaling on the data (do similar scaling for both train and test)
#Consider the RBF kernel K(x, y) = e −γkx−yk 2 as first choice
#Use cross-validation to find the best parameter C and γ (for example, C = 2−5 , 2 −3 , . . , 215 , γ = 2−15 , 2 −13
, . . . , 2 3 ).
#Use the best parameter C and γ to train the whole training set
## S3 method for class 'formula':
svm((formula, data = NULL, ..., subset, na.action =na.omit, scale = TRUE))
## S3 method for class 'default':
svm((x, y = NULL, scale = TRUE, type = NULL, kernel = "radial", degree = 3, gamma = 0, cost = 1)
tuned <- tune.svm(V2~., data = trainset, gamma = 10^(-6:-1), cost = 10^(-1:1))
# The range to gamma parameter is between 0.000001 and 0.1.
# For cost parameter the range is from 0.1 until 10.
# RBF kernel, then you need to (jointly) optimize another parameter, namely the gamma parameter.
# If you use linear kernel, you just need to optimize the C parameter.
# Larger C values increase the penalty for misclassification and thus reduce the classification error rate on the
training data (which may lead to over-fitting). Your training time and number of support vectors will increase
as you  increase the value of C.
# Gamma parameter, refers to the variance of the corresponding Gaussian bell around support vectors.
# When you are using Gaussian RBF kernel, your separating surface will be based on a combination of bell-
shaped surfaces centered at each support vector. The width of each bell-shaped surface will be inversely
proportional to gamma. If this width is smaller than the minimum pair-wise distance for your data, you
essentially have overfitting. If this width is larger than the maximum pair-wise distance for your data, all your
points fall into one class and you don't have good performance either. So the optimal width should be
somewhere between these two extremes.
gamma = 1/(numberof featuress) # default value
```

2.7 LINEAR REGRESSION

```
library(car)

setwd("H:/Learning/Analytics/Karthik")
# Read in the dataset

wine <- read.csv('wine.csv')
head(wine,10)

model4 = lm(Price ~ AGST + HarvestRain + WinterRain + Age, data=wine)


vif(model4) #library(car)
#VIF greater than 10 needs to be corrected.

# Read in test set
winetest = read.csv("wine_test.csv")
str(winetest)

# Make test set predictions
predictTest = predict(model4, newdata=winetest,interval="confidence")
predictTest


plot(model4$residuals)
abline(h=0)

# Compute R-squared
SSE = sum((winetest$Price - predictTest)^2)
SST = sum((winetest$Price - mean(wine$Price))^2)
1 - (SSE/SST)
```

2.8 TIME SERIES

```
install.packages("forecast")
library("forecast")

remove(list=ls())
setwd("C:/Users/Public/Documents/AnalyticsFileholder/irfan")

====================Lab 1==================================
#Reading Time Series Data

disconnect <- scan("disconnect.txt")
```

```
disconnectts1 <- ts(disconnect, frequency=12, start=c(2014,1))

#Plotting Time Series
plot.ts(disconnectts1)

#Decomposing Time Series
disconnectts1components <- decompose(disconnectts1)
disconnectts1components$seasonal
disconnectts1components$trend
disconnectts1components$random
plot(disconnectts1components)

forcast
disconnectforecast1 <- HoltWinters(disconnectts1,gamma=FALSE)
disconnectforecast1
plot(disconnectforecast1)
disconnectforecast1$fitted
disconnectforecast1$SSE
disconnectforecast1.1 <- forecast.HoltWinters(disconnectforecast1, h=4)
disconnectforecast1.1
plot(disconnectforecast1.1)
disconnectforecast1.1$fitted
disconnectforecast1.1$SSE
```

2.9 MARKET BASKET ANALYSIS (ASSOCIATION RULE) – UNSUPERVISED

```
library(arules)
setwd("D:/R Analytics/IIT Hydrabed/July2016/Data sets/7-Market basket analysis")


Example 1:
 #Readind data and understanding
 txn = read.transactions(file="Grocery.csv",rm.duplicates= TRUE,format = "basket",sep=",")
inspect (txn) # view the observations
length (txn)  # get number of observations
size (txn)    # number of items in each observation

#Generating rules
rules <- apriori (txn, parameter = list(supp = 0.008, conf = 0.6,maxlen=3))
# Maxlen is the maximum number of elemnts in rules
inspect(rules)
#sorting
rules <- sort (rules, by="lift", decreasing=TRUE) # 'high-confidence' rules
inspect(rules)


#Controlling the output rules
```

rules <- apriori (data=txn, parameter=list (supp=0.001,conf = 0.08), appearance = list (default="lhs",rhs="whole milk"), control = list (verbose=F))

==========Lab 2==========

```
library(arulesSequences)
x <- read_baskets(con = "sequence.csv", sep=",", info = c("sequenceID","eventID"))
as(x, "data.frame")

s1 <- cspade(x, parameter = list(support = 0.4), control = list(verbose = TRUE))
as(s1, "data.frame")

inspect(s1)
```

3. STATISTICS

3.1.1 CONFIDENCE LEVEL/INTERVAL

Confidence Interval, is a plausible range of values for a population parameters. (or) is a range of values that is likely to contain an unknown population parameter.

Confidence Level, is the percentage of random sample which yield the confidence intervals that capture the true population parameter. (or) if you draw a random sample many times, a certain percentage of the confidence interval will contain the population mean.

Accuracy, whether are not confidence interval holds true population parameter.
Precision, Width of the confidence interval.

4. TIME SERIES EXPONENTIAL SMOOTHING

```
library("TTR")
library("forecast")
```

http://a-little-book-of-r-for-time-series.readthedocs.io/en/latest/src/timeseries.html

4.1 EXAMPLE: AGE OF DEATH OF ENGLAND KINGS

```
kings <- scan("http://robjhyndman.com/tsdldata/misc/kings.dat",skip=3)
kings
kingstimeseries <- ts(kings)
plot.ts(kingstimeseries)
kingstimeseriesSMA3 <- SMA(kingstimeseries,n=3)
```

```
plot.ts(kingstimeseriesSMA3)
kingstimeseriesSMA8 <- SMA(kingstimeseries,n=8)
plot.ts(kingstimeseriesSMA8)
```

4.2 EXAMPLE: NEWYORK BIRTH RATE

```
setwd("C:/Users/Public/Documents/AnalyticsFileholder/irfan")
births <- scan("NYbirths.txt")
birthstimeseries <- ts(births, frequency=12, start=c(1946,1))
#Plotting Time Series
plot.ts(birthstimeseries)
#Decomposing Time Series
birthstimeseriescomponents <- decompose(birthstimeseries)
birthstimeseriescomponents$seasonal
birthstimeseriescomponents$trend
birthstimeseriescomponents$random
plot(birthstimeseriescomponents)
#TimeSeries minus Seasonality
birthstimeseriesseasonallyadjusted <- birthstimeseries - birthstimeseriescomponents$seasonal
plot(birthstimeseriesseasonallyadjusted)
```

4.3 EXAMPLE: RAINFALL- FORECASTS USING EXPONENTIAL SMOOTHING

Exponential smoothing can be used to make short-term forecasts for time series data.

```
> rain <-scan("http://robjhyndman.com/tsdldata/hurst/precip1.dat",skip=1)
  Read 100 items
> rainseries <- ts(rain,start=c(1813))
> plot.ts(rainseries)
```

The simple exponential smoothing method provides a way of estimating the level at the current time point. Smoothing is controlled by the parameter alpha; for the estimate of the level at the current time point. The value of alpha; lies between 0 and 1. Values of alpha that are close to 0 mean that little weight is placed on the most recent observations when making forecasts of future values.

You can see from the plot that there is roughly constant level (the mean stays constant at about 25 inches). The random fluctuations in the time series seem to be roughly constant in size over time, so it is probably appropriate to describe the data using an additive model. Thus, we can make forecasts using simple exponential smoothing.

To make forecasts using simple exponential smoothing in R, we can fit a simple exponential smoothing predictive model using the “HoltWinters()” function in R. To use HoltWinters() for simple exponential smoothing, we need to set the parameters beta=FALSE and gamma=FALSE in the HoltWinters() function (the beta and gamma parameters are used for Holt’s exponential smoothing, or Holt-Winters exponential smoothing, as described below). The HoltWinters() function returns a list variable, that contains several named elements.

For example, to use simple exponential smoothing to make forecasts for the time series of annual rainfall in London, we type:

```
> rainseriesforecasts <- HoltWinters(rainseries, beta=FALSE, gamma=FALSE)
> rainseriesforecasts
```

The output of HoltWinters() tells us that the estimated value of the alpha parameter is about 0.024. This is very close to zero, telling us that the forecasts are based on both recent and less recent observations (although somewhat more weight is placed on recent observations).

By default, HoltWinters() just makes forecasts for the same time period covered by our original time series. In this case, our original time series included rainfall for London from 1813-1912, so the forecasts are also for 1813-1912.

In the example above, we have stored the output of the HoltWinters() function in the list variable "rainseriesforecasts". The forecasts made by HoltWinters() are stored in a named element of this list variable called "fitted", so we can get their values by typing:

> rainseriesforecasts$fitted

The plot shows the original time series in black, and the forecasts as a red line. The time series of forecasts is much smoother than the time series of the original data here.

As a measure of the accuracy of the forecasts, we can calculate the sum of squared errors for the in-sample forecast errors, that is, the forecast errors for the time period covered by our original time series. The sum-of-squared-errors is stored in a named element of the list variable "rainseriesforecasts" called "SSE", so we can get its value by typing:

> rainseriesforecasts$SSE

[1] 1828.855

That is, here the sum-of-squared-errors is 1828.855.

It is common in simple exponential smoothing to use the first value in the time series as the initial value for the level. For example, in the time series for rainfall in London, the first value is 23.56 (inches) for rainfall in 1813. You can specify the initial value for the level in the HoltWinters() function by using the "l.start" parameter. For example, to make forecasts with the initial value of the level set to 23.56, we type:

> HoltWinters(rainseries, beta=FALSE, gamma=FALSE, l.start=23.56)

As explained above, by default HoltWinters() just makes forecasts for the time period covered by the original data, which is 1813-1912 for the rainfall time series. We can make forecasts for further time points by using the "forecast.HoltWinters()" function in the R "forecast" package. To use the forecast.HoltWinters() function, we first need to install the "forecast" R package (for instructions on how to install an R package, see How to install an R package).

Once you have installed the "forecast" R package, you can load the "forecast" R package by typing:

> library("forecast")

When using the forecast.HoltWinters() function, as its first argument (input), you pass it the predictive model that you have already fitted using the HoltWinters() function. For example, in the case of the rainfall time series, we stored the predictive model made using HoltWinters() in the variable "rainseriesforecasts". You specify how many further time points you want to make forecasts for by using the "h" parameter in forecast.HoltWinters(). For example, to make a forecast of rainfall for the years 1814-1820 (8 more years) using forecast.HoltWinters(), we type:

```
> rainseriesforecasts2 <- forecast.HoltWinters(rainseriesforecasts, h=8)
> rainseriesforecasts2
 Point     Forecast    Lo 80    Hi 80    Lo 95    Hi 95
191324.6781919.1749330.1814516.2616933.09470
```

```
191424.6781919.1733330.1830516.2592433.09715
191524.6781919.1717330.1846516.2567933.09960
191624.6781919.1701330.1862516.2543433.10204
191724.6781919.1685330.1878516.2519033.10449
191824.6781919.1669430.1894516.2494533.10694
191924.6781919.1653430.1910516.2470133.10938
192024.6781919.1637430.1926516.2445633.11182
```

The forecast.HoltWinters() function gives you the forecast for a year, a 80% prediction interval for the forecast, and a 95% prediction interval for the forecast. For example, the forecasted rainfall for 1920 is about 24.68 inches, with a 95% prediction interval of (16.24, 33.11).

To plot the predictions made by forecast.HoltWinters(), we can use the "plot.forecast()" function:

```
> plot.forecast(rainseriesforecasts2)
```

Here the forecasts for 1913-1920 are plotted as a blue line, the 80% prediction interval as an orange shaded area, and the 95% prediction interval as a yellow shaded area.

The 'forecast errors' are calculated as the observed values minus predicted values, for each time point. We can only calculate the forecast errors for the time period covered by our original time series, which is 1813-1912 for the rainfall data. As mentioned above, one measure of the accuracy of the predictive model is the sum-of-squared-errors (SSE) for the in-sample forecast errors.

The in-sample forecast errors are stored in the named element "residuals" of the list variable returned by forecast.HoltWinters(). If the predictive model cannot be improved upon, there should be no correlations between forecast errors for successive predictions. In other words, if there are correlations between forecast errors for successive predictions, it is likely that the simple exponential smoothing forecasts could be improved upon by another forecasting technique.

To figure out whether this is the case, we can obtain a correlogram of the in-sample forecast errors for lags 1-20. We can calculate a correlogram of the forecast errors using the "acf()" function in R. To specify the maximum lag that we want to look at, we use the "lag.max" parameter in acf().

For example, to calculate a correlogram of the in-sample forecast errors for the London rainfall data for lags 1-20, we type:

```
> acf(rainseriesforecasts2$residuals, lag.max=20)
```

You can see from the sample correlogram that the autocorrelation at lag 3 is just touching the significance bounds. To test whether there is significant evidence for non-zero correlations at lags 1-20, we can carry out a Ljung-Box test. This can be done in R using the "Box.test()", function. The maximum lag that we want to look at is specified using the "lag" parameter in the Box.test() function. For example, to test whether there are non-zero autocorrelations at lags 1-20, for the in-sample forecast errors for London rainfall data, we type:

```
> Box.test(rainseriesforecasts2$residuals, lag=20, type="Ljung-Box")
   Box-Ljung test
 data:  rainseriesforecasts2$residuals
 X-squared = 17.4008, df = 20, p-value = 0.6268
```

Here the Ljung-Box test statistic is 17.4, and the p-value is 0.6, so there is little evidence of non-zero autocorrelations in the in-sample forecast errors at lags 1-20.

To be sure that the predictive model cannot be improved upon, it is also a good idea to check whether the forecast errors are normally distributed with mean zero and constant variance. To check whether the forecast errors have constant variance, we can make a time plot of the in-sample forecast errors:

```
> plot.ts(rainseriesforecasts2$residuals)
```

The plot shows that the in-sample forecast errors seem to have roughly constant variance over time, although the size of the fluctuations in the start of the time series (1820-1830) may be slightly less than that at later dates (eg. 1840-1850).

To check whether the forecast errors are normally distributed with mean zero, we can plot a histogram of the forecast errors, with an overlaid normal curve that has mean zero and the same standard deviation as the distribution of forecast errors. To do this, we can define an R function "plotForecastErrors()", below:

```
> plotForecastErrors <- function(forecasterrors)
 {
   # make a histogram of the forecast errors:
   mybinsize <- IQR(forecasterrors)/4
   mysd   <- sd(forecasterrors)
   mymin  <- min(forecasterrors) - mysd*5
   mymax  <- max(forecasterrors) + mysd*3
   # generate normally distributed data with mean 0 and standard deviation mysd
   mynorm <- rnorm(10000, mean=0, sd=mysd)
   mymin2 <- min(mynorm)
   mymax2 <- max(mynorm)
   if (mymin2 < mymin) { mymin <- mymin2 }
   if (mymax2 > mymax) { mymax <- mymax2 }
   # make a red histogram of the forecast errors, with the normally distributed data overlaid:
   mybins <- seq(mymin, mymax, mybinsize)
   hist(forecasterrors, col="red", freq=FALSE, breaks=mybins)
   # freq=FALSE ensures the area under the histogram = 1
   # generate normally distributed data with mean 0 and standard deviation mysd
   myhist <- hist(mynorm, plot=FALSE, breaks=mybins)
   # plot the normal curve as a blue line on top of the histogram of forecast errors:
   points(myhist$mids, myhist$density, type="l", col="blue", lwd=2)
 }
```

You will have to copy the function above into R in order to use it. You can then use plotForecastErrors() to plot a histogram (with overlaid normal curve) of the forecast errors for the rainfall predictions:

```
> plotForecastErrors(rainseriesforecasts2$residuals)
```

The plot shows that the distribution of forecast errors is roughly centred on zero, and is more or less normally distributed, although it seems to be slightly skewed to the right compared to a normal curve. However, the right skew is relatively small, and so it is plausible that the forecast errors are normally distributed with mean zero.

The Ljung-Box test showed that there is little evidence of non-zero autocorrelations in the in-sample forecast errors, and the distribution of forecast errors seems to be normally distributed with mean zero. This suggests that the simple exponential smoothing method provides an adequate predictive model for London rainfall, which probably cannot be improved upon. Furthermore, the assumptions that the 80% and 95% predictions intervals were based upon (that there are no autocorrelations in the forecast errors, and the forecast errors are normally distributed with mean zero and constant variance) are probably valid.

```
rain <- scan("http://robjhyndman.com/tsdldata/hurst/precip1.dat",skip=1)
```

```
rainseries <- ts(rain,start=c(1813))
plot.ts(rainseries)

rain <- scan("Rainfall.txt",skip=1)
rainseries <- ts(rain,start=c(1813))
plot.ts(rainseries)

rainseriesforecasts <- HoltWinters(rainseries, beta=FALSE, gamma=FALSE)
rainseriesforecasts
rainseriesforecasts$fitted
plot(rainseriesforecasts)
rainseriesforecasts$SSE

rainseriesforecasts1.1 <- HoltWinters(rainseries, beta=FALSE, gamma=FALSE, l.start=23.56)
rainseriesforecasts1.1
rainseriesforecasts1.1$fitted
plot(rainseriesforecasts1.1)
rainseriesforecasts1.1$SSE

#You can specify the initial value for the level in the HoltWinters() function
#by using the “l.start” parameter

rainseriesforecasts <- HoltWinters(rainseries, beta=FALSE, gamma=FALSE, l.start=23.56)

library("forecast")
rainseriesforecasts2 <- forecast.HoltWinters(rainseriesforecasts, h=8)
rainseriesforecasts2
plot.forecast(rainseriesforecasts2)


#Model validations
The in-sample forecast errors are stored in the named element “residuals” of the
list variable returned by forecast.HoltWinters(). If the predictive model cannot
be improved upon, there should be no correlations between forecast errors for
successive predictions.

#To calculate a correlogram of the in-sample forecast errors for the London
rainfall data for lags 1-20

acf(rainseriesforecasts2$residuals, lag.max=20)
plot.ts(rainseriesforecasts2$residuals)

#You can see from the sample correlogram that the autocorrelation at lag 3 is just
touching the significance bounds.  One value outside the limits might be
expected in a correlogram plotted out to lag 20 even if the time series is drawn from a random (not
                                                                    autocorrelated) population

#To test whether there is significant evidence for non-zero correlations at lags 1-20,
we can carry out a Ljung-Box test.

Box.test(rainseriesforecasts2$residuals, lag=20, type="Ljung-Box")

#Here the Ljung-Box test statistic is 17.4, and the p-value is 0.6, so there is little
evidence of non-zero autocorrelations in the in-sample forecast errors at lags 1-20


#To be sure that the predictive model cannot be improved upon, it is also a good idea to
```

check whether the forecast errors are normally distributed with mean zero and constant variance

```
plot.ts(rainseriesforecasts2$residuals)
hist(as.vector(rainseriesforecasts2$residuals))
```

4.4 EXAMPLE: SKIRTS(HOLT'S EXPONENTIAL SMOOTHING)

If you have a time series that can be described using an additive model with increasing or decreasing trend and no seasonality, you can use Holt's exponential smoothing to make short-term forecasts.

Holt's exponential smoothing estimates the level and slope at the current time point. Smoothing is controlled by two parameters, alpha, for the estimate of the level at the current time point, and beta for the estimate of the slope b of the trend component at the current time point. As with simple exponential smoothing, the paramters alpha and beta have values between 0 and 1, and values that are close to 0 mean that little weight is placed on the most recent observations when making forecasts of future values.

An example of a time series that can probably be described using an additive model with a trend and no seasonality is the time series of the annual diameter of women's skirts at the hem, from 1866 to 1911. The data is available in the file http://robjhyndman.com/tsdldata/roberts/skirts.dat (original data from Hipel and McLeod, 1994).

We can read in and plot the data in R by typing:

```
> skirts <-
scan("http://robjhyndman.com/tsdldata/roberts/skirts.dat",skip=5)
  Read 46 items
> skirtsseries <- ts(skirts,start=c(1866))
> plot.ts(skirtsseries)
```

We can see from the plot that there was an increase in hem diameter from about 600 in 1866 to about 1050 in 1880, and that afterwards the hem diameter decreased to about 520 in 1911.

To make forecasts, we can fit a predictive model using the HoltWinters() function in R. To use HoltWinters() for Holt's exponential smoothing, we need to set the parameter gamma=FALSE (the gamma parameter is used for Holt-Winters exponential smoothing, as described below).

For example, to use Holt's exponential smoothing to fit a predictive model for skirt hem diameter, we type:

```
> skirtsseriesforecasts <- HoltWinters(skirtsseries, gamma=FALSE)
> skirtsseriesforecasts
  Smoothing parameters:
  alpha:0.8383481
  beta :1
gamma:FALSE
  Coefficients:
[,1]
  a 529.308585
  b   5.690464
> skirtsseriesforecasts$SSE
[1]16954.18
```

The estimated value of alpha is 0.84, and of beta is 1.00. These are both high, telling us that both the estimate of the current value of the level, and of the slope b of the trend component, are based mostly upon very recent observations in the time series. This makes good intuitive sense, since the level and the slope of the time series both change quite a lot over time. The value of the sum-of-squared-errors for the in-sample forecast errors is 16954.

We can plot the original time series as a black line, with the forecasted values as a red line on top of that, by typing:

```
> plot(skirtsseriesforecasts)
```

We can see from the picture that the in-sample forecasts agree pretty well with the observed values, although they tend to lag behind the observed values a little bit.

If you wish, you can specify the initial values of the level and the slope b of the trend component by using the "l.start" and "b.start" arguments for the HoltWinters() function. It is common to set the initial value of the level to the first value in the time series (608 for the skirts data), and the initial value of the slope to the second value minus the first value (9 for the skirts data). For example, to fit a predictive model to the skirt hem data using Holt's exponential smoothing, with initial values of 608 for the level and 9 for the slope b of the trend component, we type:

```
> HoltWinters(skirtsseries, gamma=FALSE, l.start=608, b.start=9)
```

As for simple exponential smoothing, we can make forecasts for future times not covered by the original time series by using the forecast.HoltWinters() function in the "forecast" package. For example, our time series data for skirt hems was for 1866 to 1911, so we can make predictions for 1912 to 1930 (19 more data points), and plot them, by typing:

```
> skirtsseriesforecasts2 <- forecast.HoltWinters(skirtsseriesforecasts, h=19)
> plot.forecast(skirtsseriesforecasts2)
```

The forecasts are shown as a blue line, with the 80% prediction intervals as an orange shaded area, and the 95% prediction intervals as a yellow shaded area.

As for simple exponential smoothing, we can check whether the predictive model could be improved upon by checking whether the in-sample forecast errors show non-zero autocorrelations at lags 1-20. For example, for the skirt hem data, we can make a correlogram, and carry out the Ljung-Box test, by typing:

```
> acf(skirtsseriesforecasts2$residuals, lag.max=20)
> Box.test(skirtsseriesforecasts2$residuals, lag=20, type="Ljung-Box")
    Box-Ljung test
  data:  skirtsseriesforecasts2$residuals
  X-squared =19.7312, df =20, p-value =0.4749
```

Here the correlogram shows that the sample autocorrelation for the in-sample forecast errors at lag 5 exceeds the significance bounds. However, we would expect one in 20 of the autocorrelations for the first twenty lags to exceed the 95% significance bounds by chance alone. Indeed, when we carry out the Ljung-Box test, the p-value is 0.47, indicating that there is little evidence of non-zero autocorrelations in the in-sample forecast errors at lags 1-20.

As for simple exponential smoothing, we should also check that the forecast errors have constant variance over time, and are normally distributed with mean zero. We can do this by making a time plot of forecast errors, and a histogram of the distribution of forecast errors with an overlaid normal curve:

```
> plot.ts(skirtsseriesforecasts2$residuals)# make a time plot
> plotForecastErrors(skirtsseriesforecasts2$residuals)# make a histogram
```

4.5 EXAMPLE: SOUVENIR (HOLT'S EXPONENTIAL SMOOTHING)

4.5.1 HOLT-WINTERS EXPONENTIAL SMOOTHING

If you have a time series that can be described using an additive model with increasing or decreasing trend and seasonality, you can use Holt-Winters exponential smoothing to make short-term forecasts.

Holt-Winters exponential smoothing estimates the level, slope and seasonal component at the current time point. Smoothing is controlled by three parameters: alpha, beta, and gamma, for the estimates of the level, slope b of the trend component, and the seasonal component, respectively, at the current time point. The parameters alpha, beta and gamma all have values between 0 and 1, and values that are close to 0 mean that relatively little weight is placed on the most recent observations when making forecasts of future values.

An example of a time series that can probably be described using an additive model with a trend and seasonality is the time series of the log of monthly sales for the souvenir shop at a beach resort town in Queensland, Australia (discussed above):

To make forecasts, we can fit a predictive model using the HoltWinters() function. For example, to fit a predictive model for the log of the monthly sales in the souvenir shop, we type:

```
> logsouvenirtimeseries <-log(souvenirtimeseries)
> souvenirtimeseriesforecasts <- HoltWinters(logsouvenirtimeseries)
> souvenirtimeseriesforecasts
  Holt-Winters exponential smoothing with trend and additive seasonal
component.
  Smoothing parameters:
  alpha:0.413418
  beta :0
gamma:0.9561275
  Coefficients:
[,1]
   a   10.37661961
   b    0.02996319
   s1  -0.80952063
   s2  -0.60576477
   s3   0.01103238
   s4  -0.24160551
   s5  -0.35933517
   s6  -0.18076683
   s7   0.07788605
   s8   0.10147055
   s9   0.09649353
   s10  0.05197826
   s11  0.41793637
   s12  1.18088423
> souvenirtimeseriesforecasts$SSE
2.011491
```

The estimated values of alpha, beta and gamma are 0.41, 0.00, and 0.96, respectively. The value of alpha (0.41) is relatively low, indicating that the estimate of the level at the current time point is based upon both recent observations and some observations in the more distant past. The value of beta is 0.00, indicating that the estimate of the slope b of the trend component is not updated over the time series, and instead is set equal to its initial value. This makes good intuitive sense, as the level changes quite a bit over the time series, but the slope b of the trend component remains roughly the same. In

contrast, the value of gamma (0.96) is high, indicating that the estimate of the seasonal component at the current time point is just based upon very recent observations.

As for simple exponential smoothing and Holt's exponential smoothing, we can plot the original time series as a black line, with the forecasted values as a red line on top of that:

```
> plot(souvenirtimeseriesforecasts)
```

We see from the plot that the Holt-Winters exponential method is very successful in predicting the seasonal peaks, which occur roughly in November every year.

To make forecasts for future times not included in the original time series, we use the "forecast.HoltWinters()" function in the "forecast" package. For example, the original data for the souvenir sales is from January 1987 to December 1993. If we wanted to make forecasts for January 1994 to December 1998 (48 more months), and plot the forecasts, we would type:

```
> souvenirtimeseriesforecasts2 <-
forecast.HoltWinters(souvenirtimeseriesforecasts, h=48)
> plot.forecast(souvenirtimeseriesforecasts2)
```

The forecasts are shown as a blue line, and the orange and yellow shaded areas show 80% and 95% prediction intervals, respectively.

We can investigate whether the predictive model can be improved upon by checking whether the in-sample forecast errors show non-zero autocorrelations at lags 1-20, by making a correlogram and carrying out the Ljung-Box test:

```
> acf(souvenirtimeseriesforecasts2$residuals, lag.max=20)
> Box.test(souvenirtimeseriesforecasts2$residuals, lag=20, type="Ljung-Box")
  Box-Ljung test
  data:  souvenirtimeseriesforecasts2$residuals
  X-squared =17.5304, df =20, p-value =0.6183
```

The correlogram shows that the autocorrelations for the in-sample forecast errors do not exceed the significance bounds for lags 1-20. Furthermore, the p-value for Ljung-Box test is 0.6, indicating that there is little evidence of non-zero autocorrelations at lags 1-20.

We can check whether the forecast errors have constant variance over time, and are normally distributed with mean zero, by making a time plot of the forecast errors and a histogram (with overlaid normal curve):

```
> plot.ts(souvenirtimeseriesforecasts2$residuals)# make a time plot
> plotForecastErrors(souvenirtimeseriesforecasts2$residuals)# make a histogram
```

From the time plot, it appears plausible that the forecast errors have constant variance over time. From the histogram of forecast errors, it seems plausible that the forecast errors are normally distributed with mean zero.

Thus,there is little evidence of autocorrelation at lags 1-20 for the forecast errors, and the forecast errors appear to be normally distributed with mean zero and constant variance over time. This suggests that Holt-Winters exponential smoothing provides an adequate predictive model of the log of sales at the souvenir shop, which probably cannot be improved upon. Furthermore, the assumptions upon which the prediction intervals were based are probably valid.

5. CODE SNIPPET

```
#display odd and even element in sample vector
x<-1:10
x[1,3,5,7,9] # doesn't work
x[2*(1:5)-1]
x[rep(c(FALSE,TRUE),5)]

set.seed(9852)
my.data<-list()
for(i in 1:100){
  my.data[[i]]<-matrix(rnorm(16),nrow=4)
}

my.data
class(my.data)

my.index<-list()
for(i in 1:100){
  #my.index[i]<-(my.data[i]<0)
  my.index[[i]]<-(my.data[[i]]<0)
}

my.index
my.negatives<-matrix(rep(0,16),nrow=4)
#my.negatives

for(i in 1:100){
```

```
  my.negatives<-my.negatives+my.index[[i]]
  print(i)
}
my.negatives

my.negative.values<-numeric(0)
for(i in 1:100){
  my.negative.values<-c(my.negative.values,my.data[[i]][my.index[[i]]])
}
my.negative.values
class(my.negative.values)

summary(my.negative.values)
##########################################
#Code Snippet for Subsetting#Excercise#
##########################################

f1<-file("Assignment5.dat",open="r")

my.data<-read.table(f1,skip=4,comment.char="%",nrows=7)
my.data

my.data2<-read.table(f1,skip=3,sep=";",dec=",",nrows=2)
my.data2

my.data3<-read.table(f1,skip=5,na.strings="-9999",sep=",",nrows=2)
my.data3


my.all.data<-rbind(my.data,my.data2,my.data3)
my.all.data


#############################################################################################
                    #Vector Manupulation
#############################################################################################

options(prompt="Kumar>")

sales.by.month <- c(0, 100, 200, 50, 0, 0, 0, 0, 0, 0, 0, 0)
sales.by.month
sales.by.month <- as.matrix(sales.by.month)
sales.by.month

print(sales.by.month)
sales.by.month[5,1]
# Remove the second row
sales.by.month[-2]
sales.by.month[2:4]
```

```
sales.by.month[c(1,3,6)]

sales.by.month[5] <- 25


# Remove the first column in matrix
sales.by.month <- sales.by.month[,-1]
sales.by.month
str(sales.by.month)

sales.by.month <- data.frame(sales.by.month)
sales.by.month <- as.data.frame(sales.by.month)

length(sales.by.month)
b=seq(-2,1, by=0.25)
b=seq(-2,1, length=14)
b
b = rep(1:3, 10)
b

b=rep(c(1,2),5)
b

e= 1:20
b=rep(c(1,2),5)
d = c(b,e)
d

dim(sales.by.month)

# Constructing a Matrixs
A <- rbind(1:3,c(1,1,2))
A

B <- cbind(1:3,c(1,1,2,0))
B

C <- matrix(c(1,2,3,4,5,6,7,8,9),nrow=3, ncol=3,byrow="TRUE")
C


sales.by.month <- c(100, 100, 200, 50, 30, 40, 70, 90, 200, 100, 20,60)
sales.by.month * 7
days.per.month <- c(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,NA)


sales.by.month / days.per.month

#Assign names to the column and row for matrix
```

```
#unlike dataframes, rownames and columnnames is purely descriptive and cann't be used for reference
dimnames(x)[[2]] <- paste("data",1:3,sep="")
dimnames(x)[[1]] <- paste("obs",1:4,sep="")
```