VARDHAMAN COLLEGE OF ENGINEERING

(Autonomous)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Shamshabad-501218, Hyderabad

VCE-R19

DATA ANALYTICS USING R

LAB MANUAL

VII Semester

A5519

Prepared by

Mr C Satya Kumar

Mr A Ramesh

Mr GS Prasada Reddy

(Assistant Professor,CSE Dept)

Week-1: Introduction to R Lab, Basic Programming.

a).Write a R program to create a sequence of numbers from 20 to 50 and find the mean of numbers from 20 to 60 and sum of numbers from 51 to 91

R Programming Code :

```
print("Sequence of numbers from 20 to 50:")
print(seq(20,50))
print("Mean of numbers from 20 to 60:")
print(mean(20:60))
print("Sum of numbers from 51 to 91:")
print(sum(51:91))
```

Sample Output:

```
[1] "Sequence of numbers from 20 to 50:"
 [1] 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
[26] 45 46 47 48 49 50
[1] "Mean of numbers from 20 to 60:"
[1] 40
[1] "Sum of numbers from 51 to 91:"
[1] 2911
```

b). Write a R program to get the first 10 Fibonacci numbers.

R Programming Code :

```
Fibonacci <- numeric(10)
Fibonacci[1] <- Fibonacci[2] <- 1
for (i in 3:10) Fibonacci[i] <- Fibonacci[i - 2] + Fibonacci[i - 1]
print("First 10 Fibonacci numbers:")
print(Fibonacci)
```

Sample Output:

```
[1] "First 10 Fibonacci numbers:"
 [1]  1  1  2  3  5  8 13 21 34 55
```

Week-1: Introduction to Loops

a). Write a R program to get all prime numbers up to a given number.

R Programming Code :

```
prime_numbers <- function(n) {
if (n >= 2) {
 x = seq(2, n)
 prime_nums = c()
 for (i in seq(2, n)) {
 if (any(x == i)) {
 prime_nums = c(prime_nums, i)
 x = c(x[(x %% i) != 0], i)
 }
 }
 return(prime_nums)
 }
 else
 {
 stop("Input number should be at least 2.")
 }
 }
prime_numbers(12)
```

Sample Output:

```
[1]  2  3  5  7 11
```

Week-2: Programs using Vectors.

a)Write a R program to create a vector of a specified type and length. Create vector of numeric, complex, logical and character types of length 6.

R Programming Code :

```
x = vector("numeric", 5)
print("Numeric Type:")
print(x)
c = vector("complex", 5)
print("Complex Type:")
print(c)
l = vector("logical", 5)
print("Logical Type:")
print(l)
chr = vector("character", 5)
print("Character Type:")
print(chr)
```

Sample Output:

```
[1] "Numeric Type:"
[1] 0 0 0 0 0
[1] "Complex Type:"
[1] 0+0i 0+0i 0+0i 0+0i 0+0i
[1] "Logical Type:"
[1] FALSE FALSE FALSE FALSE FALSE
[1] "Character Type:"
[1] "" "" "" "" ""
```

Week-2: Programs using Matrices.

a)Write a R program to create a matrix taking a given vector of numbers as input and define the column and row names. Display the matrix.

R Programming Code:

```
row_names = c("row1", "row2", "row3", "row4")
col_names = c("col1", "col2", "col3", "col4")
M = matrix(c(1:16), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))
print("Original Matrix:")
print(M)
```

Sample Output:

```
[1] "Original Matrix:"
     col1 col2 col3 col4
row1    1    2    3    4
row2    5    6    7    8
row3    9   10   11   12
row4   13   14   15   16
```

Week-2: Programs using Factors.

a)Write a R program to find the levels of factor of a given vector

R Programming Code :

```
v = c(1, 2, 3, 3, 4, NA, 3, 2, 4, 5, NA, 5)
print("Original vector:")
print(v)
print("Levels of factor of the said vector:")
print(levels(factor(v)))
```

Sample Output:

```
[1] "Original vector:"
 [1]  1  2  3  3  4 NA  3  2  4  5 NA  5
[1] "Levels of factor of the said vector:"
[1] "1" "2" "3" "4" "5"
```

Week-2: Programs using List

c).Write a R program to create a list containing strings, numbers, vectors and a logical values.

R Programming Code :

```
list_data = list("Python", "PHP", c(5, 7, 9, 11), TRUE, 125.17, 75.83)
print("Data of the list:")
print(list_data)
```

Sample Output:

```
[1] "Data of the list:"
[[1]]
[1] "Python"

[[2]]
[1] "PHP"

[[3]]
[1]  5  7  9 11

[[4]]
[1] TRUE

[[5]]
[1] 125.17

[[6]]
[1] 75.83
```

Week-3: Programs using Statics.

a) Apply all statistical concepts using R.

R is known to be one of the most popular statistical programming languages. It is vital to understand the in-built statistical functions in R. This section will outline the most common statistical calculations that are performed by data scientists.

a.1 Filling Missing Values:

One of the most common tasks in a data science project is to fill the missing values. We can use the is.na() to find the elements that are empty (NA or NAN):

```
vec <- c("test", NA, "another test")
is.na(vec)
```

This will print FALSE TRUE FALSE, indicating that the second element in NA.
To understand them better, is.na() will return all of those elements/objects that are NA. is.nan() will return all of the NaN objects. It's important to note that the NaN is an NA but an NA is not a NaN.
Note: Many statistical functions such as mean, median, etc, take in an argument: na.rm which indicates whether we want to remove the na (missing values).

The next few calculations will be based on the following two vectors:

```
A <- c(1,2,5,6.4,6.7,7,7,7,8,9,3,4,1.5,0,10,5.1,2.4,3.4, 4.5, 6.7)
B <- c(4,4.1,0,1.4,2,1,6.7,7,5,5,8,9,3,2,2.5,0,10,5.1,4.3,5.7)print(length(A)) #20
print(length(B)) #20
```

Both of the vectors, A and B, contain numerical values of 20 elements.

a.2 Mean

Mean is computed by summing the values in a collection and then dividing by the total number of values:

```
my_mean <- mean(A)
print(my_mean)
```

a.3 Median

Median is the middle value in a sorted collection. If there are an even number of values then it's the average of the two middle values:

```
my_median <- median(A)
print(my_median)
```

a.4 Mode

A mode is the most frequent value. R is missing a standard built-in function to calculate the mode. However, we can create a function to calculate it as shown below.

```
distinct_A <- unique(A)
matches <- match(A, distinct_A)
table_A <- tabulate(matches)
max_A <- which.max(table_A)
```

```
mode<-distinct_A[max_A]
print(mode)
```

The function performs the following steps:

1. Computes the distinct values of the collection
2. Then it finds the frequency of each of the item and creates a table out of it
3. Lastly, it finds the index of the term that has the highest occurrence and returns it as the mode.

a.5 Standard deviation

Standard deviation is the deviation of the values from the mean.

```
sd <- sd(A)
print(sd)
```

a.6 Variance

Variance is the square of the standard deviation:

```
var <- var(A)
print(var)
```

a.7 Correlation

Correlation helps us understand whether the collections have a relationship with each other and whether they co-move with each other along with the strength of the relationship:

```
cor <- cor(A, B)
print(cor)
```

We can pass in a specific correlation method such as Kendall or Spearman. Pearson is the default correlation method. Pass in the correlation method in the method argument.

a.8 Covariance

Covariance is created to inform us about the relationship between variables.

```
covariance <- cov(A, B)
print(covariance)
```

a.9 Standardise and normalise the data set

We are often required to normalise data such as by using min-max normalisation or calculate the z-score using the standardisation mechanism. Standardising data means having a data set with mean = 0 and standard deviation = 1. It requires subtracting the mean from each of the observation and then dividing it by the standard deviation. We can use the scale function. If we want to subtract the mean from each of the observation then set its center parameter to True. If we want to standardise data then we need to set its scale parameter to True.

```
normal_A <- scale(A, center=TRUE, scale = FALSE)
print(normal_A)standard_A <- scale(A, center=TRUE, scale = TRUE)
print(standard_A)
```

Week-4: Programs using Linear Regression

Analysis: Dataset cars includes 50 observations with 2 variables - dist containing stoppping distance in feet and speed containing speed of a car before applying the brakes in miles per hour. Data were recorded in the 1920s.

```
summary(cars)
##     speed         dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
plot(cars$speed, cars$dist, xlab='Speed (mph)', ylab='Stopping Distance (ft)',
    main='Stopping Distance vs. Speed')
```

Let us build a linear model and find the best fitting line.

```
cars_lm <- lm(cars$dist ~ cars$speed)
cars_lm
## Call:
## lm(formula = cars$dist ~ cars$speed)
## Coefficients:
## (Intercept)   cars$speed
##     -17.579       3.932
plot(cars$speed, cars$dist, xlab='Speed (mph)', ylab='Stopping Distance (ft)',
    main='Stopping Distance vs. Speed')
abline(cars_lm)
summary(cars_lm)
## Call:
## lm(formula = cars$dist ~ cars$speed)
## Residuals:
##     Min      1Q  Median      3Q     Max
## -29.069  -9.525  -2.272   9.215  43.201
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.5791     6.7584  -2.601   0.0123 *
## cars$speed    3.9324     0.4155   9.464 1.49e-12 ***
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## Residual standard error: 15.38 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
plot(cars_lm$fitted.values, cars_lm$residuals, xlab='Fitted Values', ylab='Residuals')
abline(0,0)
qqnorm(cars_lm$residuals)
qqline(cars_lm$residuals)
speed <- cars$speed
speed2 <- speed^2
dist <- cars$dist
cars_qm <- lm(dist ~ speed + speed2)
summary(cars_qm)
## Call:
## lm(formula = dist ~ speed + speed2)
## Residuals:
```

```
##    Min     1Q  Median     3Q    Max
## -28.720  -9.184  -3.188   4.628  45.152
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.47014   14.81716   0.167    0.868
## speed        0.91329    2.03422   0.449    0.656
## speed2       0.09996    0.06597   1.515    0.136
## Residual standard error: 15.18 on 47 degrees of freedom
## Multiple R-squared:  0.6673, Adjusted R-squared:  0.6532
## F-statistic: 47.14 on 2 and 47 DF,  p-value: 5.852e-12
speedvalues <- seq(0, 25, 0.1)
predictedcounts <- predict(cars_qm,list(speed=speedvalues, speed2=speedvalues^2))
plot(speed, dist, pch=16, xlab='Speed (mph)', ylab='Stopping Distance (ft)')
lines(speedvalues, predictedcounts)
plot(cars_qm$fitted.values, cars_qm$residuals, xlab='Fitted Values', ylab='Residuals')
abline(0,0)
qqnorm(cars_qm$residuals)
qqline(cars_qm$residuals)
```

Week-5: Programs using Logistic Regression

The Dataset: *mtcars(motor trend car road test) comprises fuel consumption, performance and 10 aspects of automobile design for 32 automobiles. It comes pre installed with dplyr package in R.*

```
# Installing the package
```

```
install.packages("dplyr")

# Loading package
library(dplyr)

# Summary of dataset in package
summary(mtcars)
```

Performing Logistic regression on dataset
Logistic regression is implemented in R using glm() by training the model using features or variables in the dataset.

```
# Installing the package
install.packages("caTools")    # For Logistic regression
install.packages("ROCR")       # For ROC curve to evaluate model

# Loading package
library(caTools)
library(ROCR)

# Splitting dataset
split <- sample.split(mtcars, SplitRatio = 0.8)
split

train_reg <- subset(mtcars, split == "TRUE")
test_reg <- subset(mtcars, split == "FALSE")

# Training model
logistic_model <- glm(vs ~ wt + disp,
                      data = train_reg,
                      family = "binomial")
logistic_model

# Summary
summary(logistic_model)

# Predict test data based on model
predict_reg <- predict(logistic_model,
                       test_reg, type = "response")
predict_reg

# Changing probabilities
predict_reg <- ifelse(predict_reg >0.5, 1, 0)

# Evaluating model accuracy
# using confusion matrix
table(test_reg$vs, predict_reg)

missing_classerr <- mean(predict_reg != test_reg$vs)
print(paste('Accuracy =', 1 - missing_classerr))

# ROC-AUC Curve
ROCPred <- prediction(predict_reg, test_reg$vs)
```

```
ROCPer <- performance(ROCPred, measure = "tpr",
                      x.measure = "fpr")

auc <- performance(ROCPred, measure = "auc")
auc <- auc@y.values[[1]]
auc

# Plotting curve
plot(ROCPer)
plot(ROCPer, colorize = TRUE,
    print.cutoffs.at = seq(0.1, by = 0.1),
    main = "ROC CURVE")
abline(a = 0, b = 1)

auc <- round(auc, 4)
legend(.6, .4, auc, title = "AUC", cex = 1)
```

wt influences dependent variables positively and one unit increase in wt increases the log of odds for vs =1 by 1.44. disp influences dependent variables negatively and one unit increase in disp decreases the log of odds for vs =1 by 0.0344. Null deviance is 31.755(fit dependent variable with intercept) and Residual deviance is 14.457(fit dependent variable with all independent variable). AIC(Alkaline Information criteria) value is 20.457 i.e the lesser the better for the model. Accuracy comes out to be 0.75 i.e 75%.

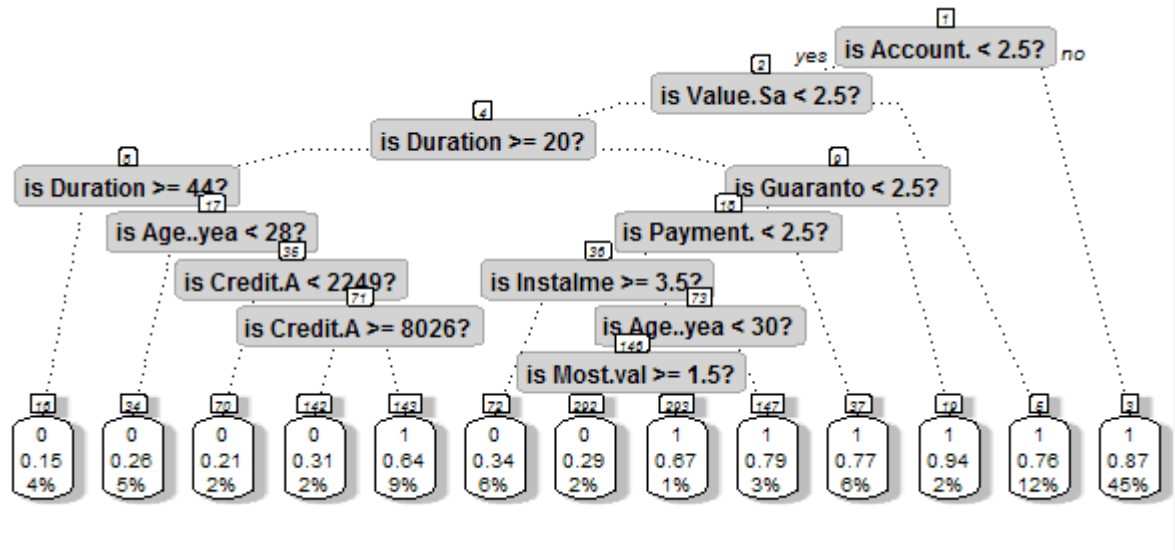
Week 6: Implement Decision trees

```
#read data file
mydata= read.csv("C:\\Users\\german_credit.csv")
# Check attributes of data
str(mydata)
```

```
'data.frame': 1000 obs. of  21 variables:
 $ Creditability                 : Factor w/ 2 levels "0","1": 2 2 2 2
 $ Account.Balance               : int  1 1 2 1 1 1 1 1 4 2 ...
 $ Duration.of.Credit..month.    : int  18 9 12 12 12 10 8 6 18 24 ...
 $ Payment.Status.of.Previous.Credit: int  4 4 2 4 4 4 4 4 4 2 ...
 $ Purpose                       : int  2 0 9 0 0 0 0 0 3 3 ...
 $ Credit.Amount                 : int  1049 2799 841 2122 2171 2241
 $ Value.Savings.Stocks          : int  1 1 2 1 1 1 1 1 1 3 ...
 $ Length.of.current.employment  : int  2 3 4 3 3 2 4 2 1 1 ...
 $ Instalment.per.cent           : int  4 2 2 3 4 1 1 2 4 1 ...
 $ Sex...Marital.Status          : int  2 3 2 3 3 3 3 3 2 2 ...
 $ Guarantors                    : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Duration.in.Current.address   : int  4 2 4 2 4 3 4 4 4 4 ...
 $ Most.valuable.available.asset : int  2 1 1 1 2 1 1 1 3 4 ...
 $ Age..years.                   : int  21 36 23 39 38 48 39 40 65 23 ...
 $ Concurrent.Credits            : int  3 3 3 3 1 3 3 3 3 3 ...
 $ Type.of.apartment             : int  1 1 1 1 2 1 2 2 2 1 ...
 $ No.of.Credits.at.this.Bank    : int  1 2 1 2 2 2 2 1 2 1 ...
 $ Occupation                    : int  3 3 2 2 2 2 2 2 1 1 ...
 $ No.of.dependents              : int  1 2 1 2 1 2 1 2 1 1 ...
 $ Telephone                     : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Foreign.Worker                : int  1 1 1 2 2 2 2 2 1 1 ...
# Check number of rows and columns
dim(mydata)
# Make dependent variable as a factor (categorical)
mydata$Creditability = as.factor(mydata$Creditability)
# Split data into training (70%) and validation (30%)
dt = sort(sample(nrow(mydata), nrow(mydata)*.7))
train<-mydata[dt,]
val<-mydata[-dt,] # Check number of rows in training data set
nrow(train)
# To view dataset
edit(train)
# Decision Tree Model
library(rpart)
mtree <- rpart(Creditability~., data = train, method="class", control = rpart.control(minsplit =
20, minbucket = 7, maxdepth = 10, usesurrogate = 2, xval =10 ))

mtree

#Plot tree
plot(mtree)
text(mtree)

#Beautify tree
library(rattle)
library(rpart.plot)
library(RColorBrewer)

#view1
prp(mtree, faclen = 0, cex = 0.8, extra = 1)

#view2 - total count at each node
tot_count <- function(x, labs, digits, varlen)
```

```
{paste(labs, "\n\nn =", x$frame$n)}

prp(mtree, faclen = 0, cex = 0.8, node.fun=tot_count)

#view3- fancy Plot
rattle()
fancyRpartPlot(mtree)

############################
########Pruning#############
############################

printcp(mtree)
bestcp <- mtree$cptable[which.min(mtree$cptable[,"xerror"]),"CP"]

# Prune the tree using the best cp.
pruned <- prune(mtree, cp = bestcp)

# Plot pruned tree
prp(pruned, faclen = 0, cex = 0.8, extra = 1)

# confusion matrix (training data)
conf.matrix <- table(train$Creditability, predict(pruned,type="class"))
rownames(conf.matrix) <- paste("Actual", rownames(conf.matrix), sep = ":")
colnames(conf.matrix) <- paste("Pred", colnames(conf.matrix), sep = ":")
print(conf.matrix)

#Scoring
library(ROCR)
val1 = predict(pruned, val, type = "prob")
#Storing Model Performance Scores
pred_val <-prediction(val1[,2],val$Creditability)

# Calculating Area under Curve
perf_val <- performance(pred_val,"auc")
perf_val

# Plotting Lift curve
plot(performance(pred_val, measure="lift", x.measure="rpp"), colorize=TRUE)

# Calculating True Positive and False Positive Rate
perf_val <- performance(pred_val, "tpr", "fpr")

# Plot the ROC curve
plot(perf_val, col = "green", lwd = 1.5)

#Calculating KS statistics
ks1.tree <- max(attr(perf_val, "y.values")[[1]] - (attr(perf_val, "x.values")[[1]]))
ks1.tree
# Advanced Plot
prp(pruned, main="Decision Tree",
   extra=106,
   nn=TRUE,
   fallen.leaves=TRUE,
```

```
branch=.5,
faclen=0,
trace=1,
shadow.col="gray",
branch.lty=3,
split.cex=1.2,
split.prefix="is ",
split.suffix="?",
split.box.col="lightgray",
split.border.col="darkgray",
split.round=.5)
```



Decision Tree