Name-Fenil Tailor NId-N18730085 Net Id-fst216

Assignment-2

Q-1A)

First,JoJo data is not Stationary as we can see an upward trend in it.To make it stationary I have first use diff function and then remove seasonality by removing the seasonal component which I got by getting seasonal component from decomposing time series.

Code:-

```
dataset<-read.table(file.choose(),header=FALSE,sep=",")

JoJo_value<-diff(dataset$V1)   #to remove trend

tsData2 <- ts(JoJo_value, frequency = 4); # quarterly data

x1=decompose(tsData2)

x1remainder<-tsData2-x1$seasonal   #to remove seasonality

adf.test(x1remainder,alternative = "stationary")
```

```
        Augmented Dickey-Fuller Test

data:  x1remainder
Dickey-Fuller = -4.1014, Lag order = 4, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(x1remainder, alternative = "stationary") :
 p-value smaller than printed p-value
```

```
kpss.test(x1remainder)
```

```
        KPSS Test for Level Stationarity

data:  x1remainder
KPSS Level = 0.1153, Truncation lag parameter = 2, p-value = 0.1

Warning message:
In kpss.test(x1remainder) : p-value greater than printed p-value
```

Conclusion

As we can see from the Augmented Dickey-Fuller Test that p-value is nearly around Zero(too small,not significant from zero).Same way by looking At kpss test we can say that p-value is significant from Zero. So we can now say that time series is stationary

Name-Fenil Tailor NId-N18730085 Net Id-fst216

Q-1B)

Autoregressive (AR) processes have theoretical autocorrelation functions (ACFs) that decay toward zero, instead of cutting off to zero. The autocorrelation coefficients might alternate in sign frequently, or show a wave-like pattern, but in all cases, they tail off toward zero. By contrast, AR processes with order p have theoretical partial autocorrelation functions (PACF) that cut off to zero after lag p. (**The lag length of the final PACF spike equals the AR order of the process, p.**)

The theoretical ACFs of MA (moving average) processes with order q cut off to zero after lag q, the MA order of the process. However, their theoretical PACFs decay toward zero. (**The lag length of the final ACF spike equals the MA order of the process, q.**)

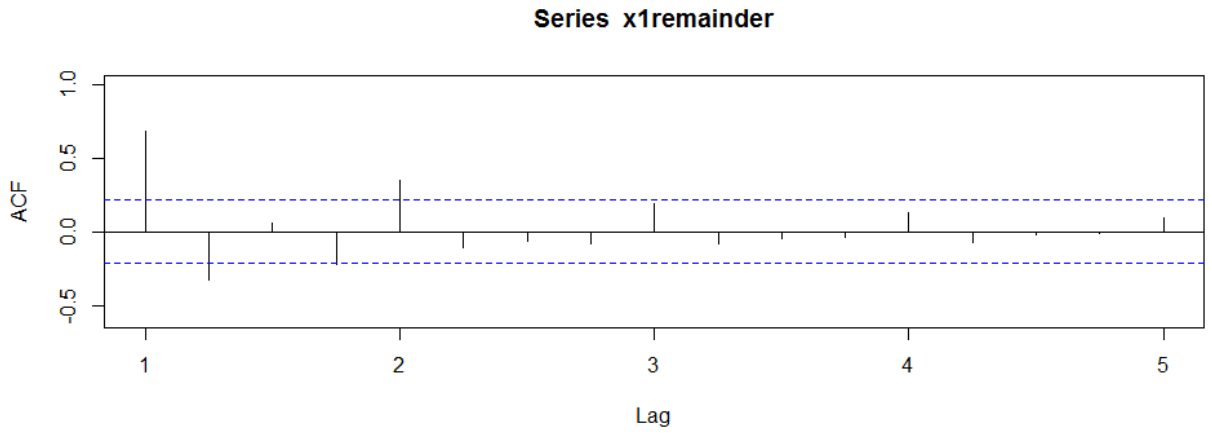
Code:-

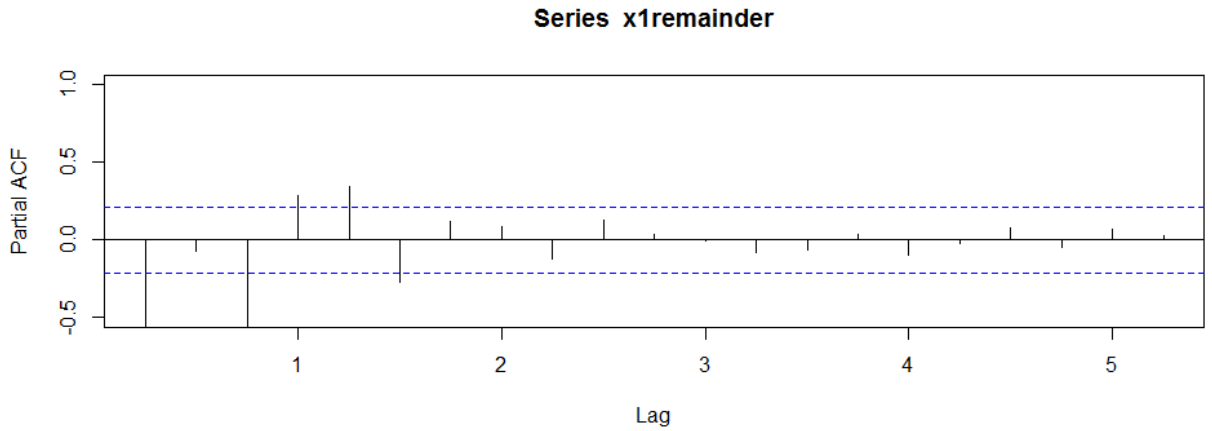
```
acf(x1remainder)

par(mfrow=c(2,1))

acf(x1remainder,21,xlim=c(1,5))   # set the x-axis limits to start at 1 then

pacf(x1remainder,21,ylim=c(-.5,1))
```

Conclusion:-

As the lag length of the final ACF spike equals the MA order of the process we can say that MA order for x1remainder (Stationary Time series data) is 2.

As ,the lag length of the final PACF spike equals the AR order of the process **we can say that MA order for x1remainder (Stationary Time series data) is 2(taken ceil of 1.5) .**





Bases on the ACF and PACF plots, we can conclude that model ARMA(p,q) where p=2(AR=2) and q=2(MA=2) are the appropriate

Q1-C)

Code:-

```
arima(x1remainder,order=c(1,0,1))    (p=1,q=1)
      arima(x = x1remainder, order = c(1, 0, 1))

      Coefficients:
                ar1      ma1  intercept
            -0.3829  -0.5088     0.1681
      s.e.   0.1283   0.0913     0.0344

      sigma^2 estimated as 0.7529:  log likelihood = -106.4,  aic = 220.8


arima(x = x1remainder, order = c(1, 0, 2)) (p=1,q=2)
```

```
Coefficients:
         ar1      ma1     ma2  intercept
      0.4621  -1.7060  1.0000     0.1735
s.e.  0.1040   0.1358  0.1573     0.0423

sigma^2 estimated as 0.5143:  log likelihood = -93.85,  aic = 197.7
```

arima(x = x1remainder, order = c(1, 0, 3)) (p=1,q=3)

```
Coefficients:
          ar1     ma1      ma2      ma3  intercept
      -0.8512  0.7513  -0.0790  -0.7289     0.1666
s.e.   0.0582  0.0835   0.0892   0.0770     0.0380

sigma^2 estimated as 0.4381:  log likelihood = -87.46,  aic = 186.91
```

arima(x = x1remainder, order = c(2, 0, 1)) (p=2,q=1)

```
Coefficients:
          ar1      ar2      ma1  intercept
      -0.3965  -0.0188  -0.4996     0.1682
s.e.   0.1651   0.1451   0.1157     0.0342

sigma^2 estimated as 0.7527:  log likelihood = -106.39,  aic = 222.79
```

arima(x = x1remainder, order = c(2, 0, 2)) (p=2,q=2)

```
Coefficients:
         ar1      ar2      ma1     ma2  intercept
      0.0719  -0.2904  -1.2276  0.8228     0.1687
s.e.  0.1390   0.1306   0.0918  0.0976     0.0403

sigma^2 estimated as 0.5674:  log likelihood = -95.68,  aic = 203.37
```

arima(x = x1remainder, order = c(2, 0, 3)) (p=2,q=3)

```
Coefficients:
          ar1      ar2     ma1      ma2      ma3  intercept
      -1.0366  -0.2000  0.8106  -0.0091  -0.6779     0.1664
s.e.   0.1286   0.1243  0.0863   0.1021   0.0759     0.0369

sigma^2 estimated as 0.4278:  log likelihood = -86.23,  aic = 186.45
```

arima(x = x1remainder, order = c(2, 0, 4)) (p=2,q=4)

```
Coefficients:
          ar1      ar2      ma1     ma2      ma3     ma4  intercept
      -0.2715  -0.2548  -0.6297  0.1901  -0.6296  1.0000     0.1722
s.e.   0.1263   0.1228   0.1187  0.1085   0.1380  0.1434     0.0371

sigma^2 estimated as 0.3159:  log likelihood = -76.43,  aic = 168.86
```

```
arima(x = x1remainder, order = c(3, 0, 5))  (p=3,q=5)

      Coefficients:
               ar1      ar2      ar3     ma1     ma2     ma3     ma4      ma5  inte
      rcept
            -0.9481  -0.9493  -0.9101  0.3749  0.6513  0.2206  0.4013  -0.0806
      0.1731
      s.e.   0.0800   0.0679   0.0533  0.1231  0.1275  0.1282  0.1001   0.1473
      0.0337

      sigma^2 estimated as 0.2063:  log likelihood = -55.72,  aic = 131.44

arima(x = x1remainder, order = c(4, 0, 7))  (p=4,q=7)

      Coefficients:
               ar1      ar2     ar3     ar4      ma1     ma2      ma3     ma4
      ma5     ma6      ma7
            0.0813  -0.1186  0.0003  0.8238  -0.8706  0.3776  -0.4117  0.5846  -0.7
      861  0.7662  -0.1648
      s.e.  0.1074   0.0707  0.0941  0.0742   0.2218  0.2790   0.1222  0.4227   0.6
      678  0.5812   0.2770
            intercept
               0.1726
      s.e.     0.0835

      sigma^2 estimated as 0.1458:  log likelihood = -44.31,  aic = 114.61
```

Conclusion

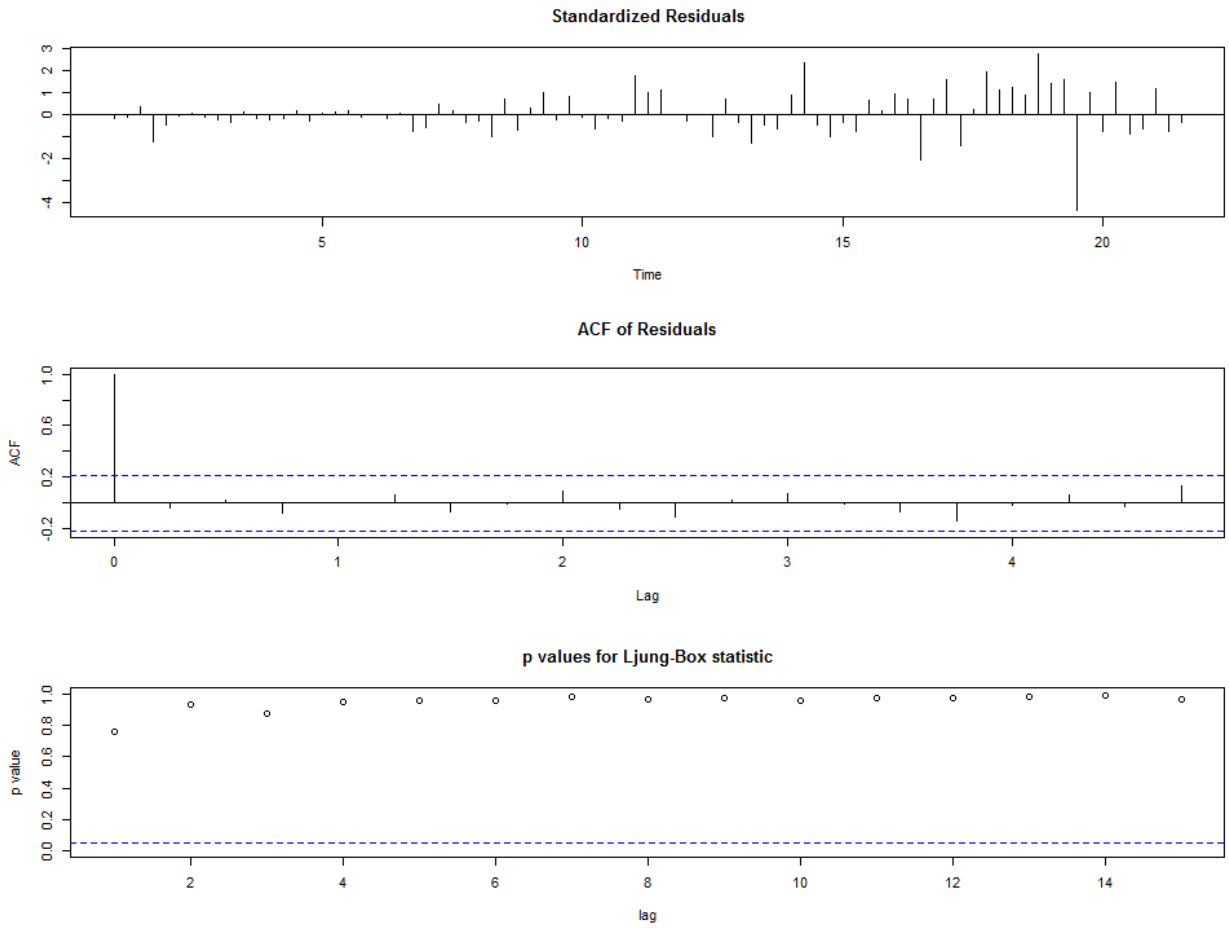
As we can see that aic value of the arima model of the order (p,d,q)=(4,0,7) is the minimum(114.61) among all the models analyzed . So we can take this model as appropriate to make further analysis.

Q-1D)

Code:-

```
m1.jojo=arima(x1remainder,order=c(4,0,7))

tsdiag1<-tsdiag(m1.jojo,gof=15,omit.initial=FALSE)

Box.test(m1.jojo$residuals,lag=1)
```

```
                    Box-Pierce test

      data:  m1.jojo$residuals
      X-squared = 0.088802, df = 1, p-value = 0.7657
```



Conclusion

We can see from Box-pierce test that p-value is significant (>0.5) .So the residuals are not stationary.

Q-1E)

Code:-

```
arima101<-arima(x1remainder,order=c(1,0,1))
mydatapred1<-predict(arima101,n.ahead=4)
mydatapred1
```

Name-Fenil Tailor NId-N18730085 Net Id-fst216

```
$pred
          Qtr1        Qtr2        Qtr3        Qtr4
21                                       2.76482973
22 -0.82601865  0.54877076  0.02241996

$se
        Qtr1      Qtr2      Qtr3      Qtr4
21                                0.8677111
22 1.1625420 1.1996853 1.2050336
```

Conclusion

we are satisfied with the fit of an ARIMA(1,0,1)–model

By using this model we can predict the quarterly earnings per share values of next quarters of the year(2030 which is 22 nd year from 2009) which are mentioned above.

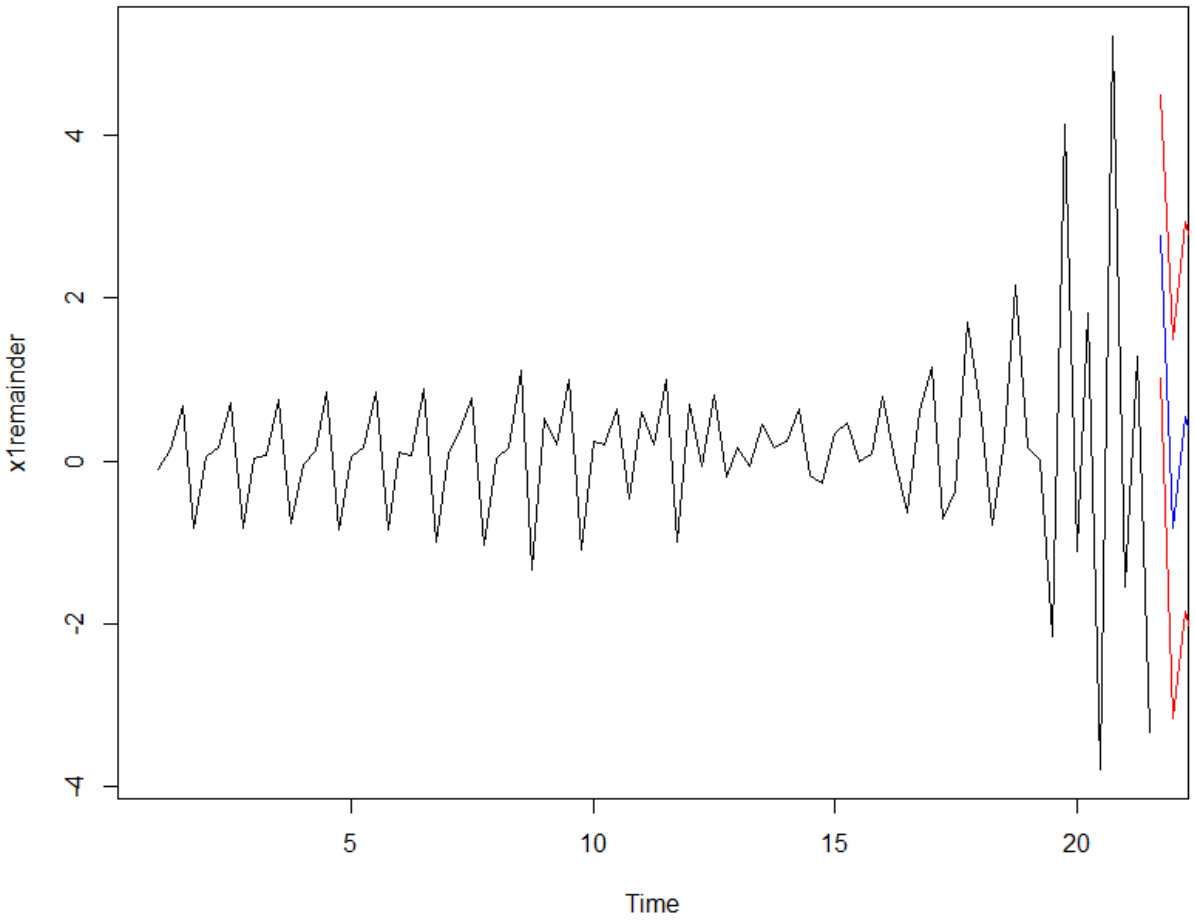
I have taken difference first so got only 83 values.It's predicting last quarter value of 2029 and for all remaining quarter values for 2030

Q-1F)

Code:-

```
arima101<-arima(x1remainder,order=c(1,0,1))

mydatapred1<-predict(arima101,n.ahead=4)

plot(x1remainder)

lines(mydatapred1$pred,col="blue")

lines(mydatapred1$pred+2*mydatapred1$se, col="red")

lines(mydatapred1$pred-2*mydatapred1$se, col="red")
```

Name-Fenil Tailor NId-N18730085 Net Id-fst216



Conclusion:-

Confidence interval lies between the two red lines. Took (2*(+,-) standard deviation) from predicted value to define and plot confidence interval

Code:-

```
> mydatapred1$pred
          Qtr1        Qtr2        Qtr3        Qtr4
21                                       2.76482973
22 -0.82601865  0.54877076  0.02241996
> mydatapred1$pred+2*mydatapred1$se
       Qtr1     Qtr2     Qtr3     Qtr4
21                             4.500252
22 1.499065 2.948141 2.432487
> mydatapred1$pred-2*mydatapred1$se
        Qtr1      Qtr2      Qtr3      Qtr4
21                                1.029407
22 -3.151103 -1.850600 -2.387647
```

Q-2A)

Code:-

```
#----Read Glassdata file
glassdata<-read.table(file.choose(),header=F,sep=",")
glassdataframe<-data.frame(glassdata)
collection<-c(1,2,3,4)
#----V11 column is glass_type column
glassdataframe["bi"]<-ifelse(glassdataframe$V11 %in% collection,0,1)
header1<-c("id","RI","Na","Mg","Al","Si","K","Ca","Ba","Fe","Type","bi")
#----unlist the header
colnames(glassdataframe)<-unlist(header1)
View(glassdataframe)
```

Conclusion:-

Created new DataFrame Column “bi”

1)set the value of column “bi” to 0 if the glass type is 1 through 4

2) set the value of column “bi” to 1 if the glass type is 5 through 7

Q-2B)

```
#--feature matrix (fea)
fea=matrix(c(RI,Na,Mg,Al,Si,K,Ca,Ba,Fe),nrow=214,ncol =9 ,byrow = F)
m2=lm(glassdataframe$bi~RI+Na+Mg+Al+Si+K+Ca+Ba+Fe)
#--Response vector (y)
y=m2$model$`glassdataframe$bi`
```

Conclusion:-

Created "fea" matrix by using features (RI,Na,Mg,Al,Si,K,Ca,Ba,Fe)

Created response vector "y" from the "bi" column

Q-2C)

Code:-

```
#Normalize the data so that higher magnitude values don't influence while predicting response values for testing data

normalize<-function(x)

{

return ((x-min(x))/(max(x)-min(x)))

}


glassdata_normalize<-as.data.frame(lapply(glassdataframe[,c(2,3,4,5,6,7,8,9,10)],normalize))

dataframecoll=data.frame(y)

smp<-floor(0.60*nrow(glassdata_normalize))

set.seed(123)

train<-sample(seq_len(nrow(glassdata_normalize)),size=smp)

traindata<-glassdata_normalize[train,]

trainres<-(dataframecoll[train,])

testdata<-glassdata_normalize[-train,]

testres<-(dataframecoll[-train,])
```

Conclusion:-

#Randomly selected 60% rows from the dataframe and assign them as training data. Selected training data randomly so that we can have higher probability of choosing data rows for each of the glass type presented in original data file. By choosing training data that way we can predict values of the responses more efficiently for the testing data set.

Selected 60% values (more than 50% value) for training data. Reason for doing this is that the more data we have to train the model,the more accuracy we can get for predicting the response values for testing data(40% values).

Q-2D)

Code:-

```
#Fit  knn model by using
#train parameter as traindata(training data)
#test  parameter as testdata(testing data)
#cl  parameter as trainres (training result)
glassdata_normalize<-as.data.frame(lapply(glassdataframe[,c(2,3,4,5,6,7,8,9,10)],normalize))
dataframecoll=data.frame(y)
smp<-floor(0.60*nrow(glassdata_normalize))
set.seed(123)
train<-sample(seq_len(nrow(glassdata_normalize)),size=smp)
traindata<-glassdata_normalize[train,]
trainres<-(dataframecoll[train,])
testdata<-glassdata_normalize[-train,]
testres<-(dataframecoll[-train,])
library(class)
knn_req<-knn(train=traindata,test=testdata,cl=trainres,k=5)
```

Conclusion:-

Fit knn model by using train parameter as traindata(training data), test parameter as testdata(testing data) and cl parameter as trainres (training result) and K=5;

Q-2E)

Code:-

```
#Fit  knn model by using
#train parameter as traindata(training data)
#test  parameter as testdata(testing data)
#cl  parameter as trainres (training result)
glassdata_normalize<-as.data.frame(lapply(glassdataframe[,c(2,3,4,5,6,7,8,9,10)],normalize))
```

```
dataframecoll=data.frame(y)

smp<-floor(0.60*nrow(glassdata_normalize))

set.seed(123)

train<-sample(seq_len(nrow(glassdata_normalize)),size=smp)

traindata<-glassdata_normalize[train,]

trainres<-(dataframecoll[train,])

testdata<-glassdata_normalize[-train,]

testres<-(dataframecoll[-train,])

knn_req<-knn(train=traindata,test=testdata,cl=trainres,k=5)

install.packages('e1071', repo='http://nbcgib.uesc.br/mirrors/cran/')

library(e1071)

library(class)

library(caret)

confusionMatrix(testres,knn_req)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 58  4
         1  3 21

               Accuracy : 0.9186
                 95% CI : (0.8395, 0.9666)
    No Information Rate : 0.7093
    P-Value [Acc > NIR] : 1.949e-06

                  Kappa : 0.8003
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9508
            Specificity : 0.8400
         Pos Pred Value : 0.9355
         Neg Pred Value : 0.8750
             Prevalence : 0.7093
         Detection Rate : 0.6744
   Detection Prevalence : 0.7209
      Balanced Accuracy : 0.8954

       'Positive' Class : 0
```

Conclusion

As we can see by the Confusion matrix statistics that we have accuracy (58+21)/(58+4+3+21) =(79)/(86)=91.86%

Q-2F)

Code:-

```
library(class)
seq<-c(3,5,7,9,11,13)
for(val in seq)
{
knn_req<-knn(train=traindata,test=testdata,cl=trainres,k=val)
print("confusion matrix for k=")
print(val)
print(confusionMatrix(testres,knn_req))
}
```

```
[1] "confusion matrix for k="
[1] 3
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 59  3
         1  2 22

               Accuracy : 0.9419
                 95% CI : (0.8695, 0.9809)
    No Information Rate : 0.7093
    P-Value [Acc > NIR] : 6.98e-08

                  Kappa : 0.8573
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9672
            Specificity : 0.8800
         Pos Pred Value : 0.9516
         Neg Pred Value : 0.9167
             Prevalence : 0.7093
         Detection Rate : 0.6860
   Detection Prevalence : 0.7209
      Balanced Accuracy : 0.9236

       'Positive' Class : 0
```

```
[1] "confusion matrix for k="
[1] 5
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 58  4
         1  3 21

               Accuracy : 0.9186
                 95% CI : (0.8395, 0.9666)
    No Information Rate : 0.7093
    P-Value [Acc > NIR] : 1.949e-06

                  Kappa : 0.8003
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9508
            Specificity : 0.8400
         Pos Pred Value : 0.9355
         Neg Pred Value : 0.8750
             Prevalence : 0.7093
         Detection Rate : 0.6744
   Detection Prevalence : 0.7209
      Balanced Accuracy : 0.8954

       'Positive' Class : 0

[1] "confusion matrix for k="
[1] 7
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 59  3
         1  4 20

               Accuracy : 0.9186
                 95% CI : (0.8395, 0.9666)
    No Information Rate : 0.7326
    P-Value [Acc > NIR] : 1.434e-05

                  Kappa : 0.7951
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9365
            Specificity : 0.8696
         Pos Pred Value : 0.9516
         Neg Pred Value : 0.8333
             Prevalence : 0.7326
         Detection Rate : 0.6860
   Detection Prevalence : 0.7209
      Balanced Accuracy : 0.9030

       'Positive' Class : 0

[1] "confusion matrix for k="
[1] 9
Confusion Matrix and Statistics
```

```
          Reference
Prediction  0  1
         0 60  2
         1  5 19

               Accuracy : 0.9186
                 95% CI : (0.8395, 0.9666)
    No Information Rate : 0.7558
    P-Value [Acc > NIR] : 9.296e-05

                  Kappa : 0.7897
 Mcnemar's Test P-Value : 0.4497

            Sensitivity : 0.9231
            Specificity : 0.9048
         Pos Pred Value : 0.9677
         Neg Pred Value : 0.7917
             Prevalence : 0.7558
         Detection Rate : 0.6977
   Detection Prevalence : 0.7209
      Balanced Accuracy : 0.9139

       'Positive' Class : 0

[1] "confusion matrix for k="
[1] 11
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 60  2
         1  5 19

               Accuracy : 0.9186
                 95% CI : (0.8395, 0.9666)
    No Information Rate : 0.7558
    P-Value [Acc > NIR] : 9.296e-05

                  Kappa : 0.7897
 Mcnemar's Test P-Value : 0.4497

            Sensitivity : 0.9231
            Specificity : 0.9048
         Pos Pred Value : 0.9677
         Neg Pred Value : 0.7917
             Prevalence : 0.7558
         Detection Rate : 0.6977
   Detection Prevalence : 0.7209
      Balanced Accuracy : 0.9139

       'Positive' Class : 0

[1] "confusion matrix for k="
[1] 13
Confusion Matrix and Statistics

          Reference
Prediction  0  1
```

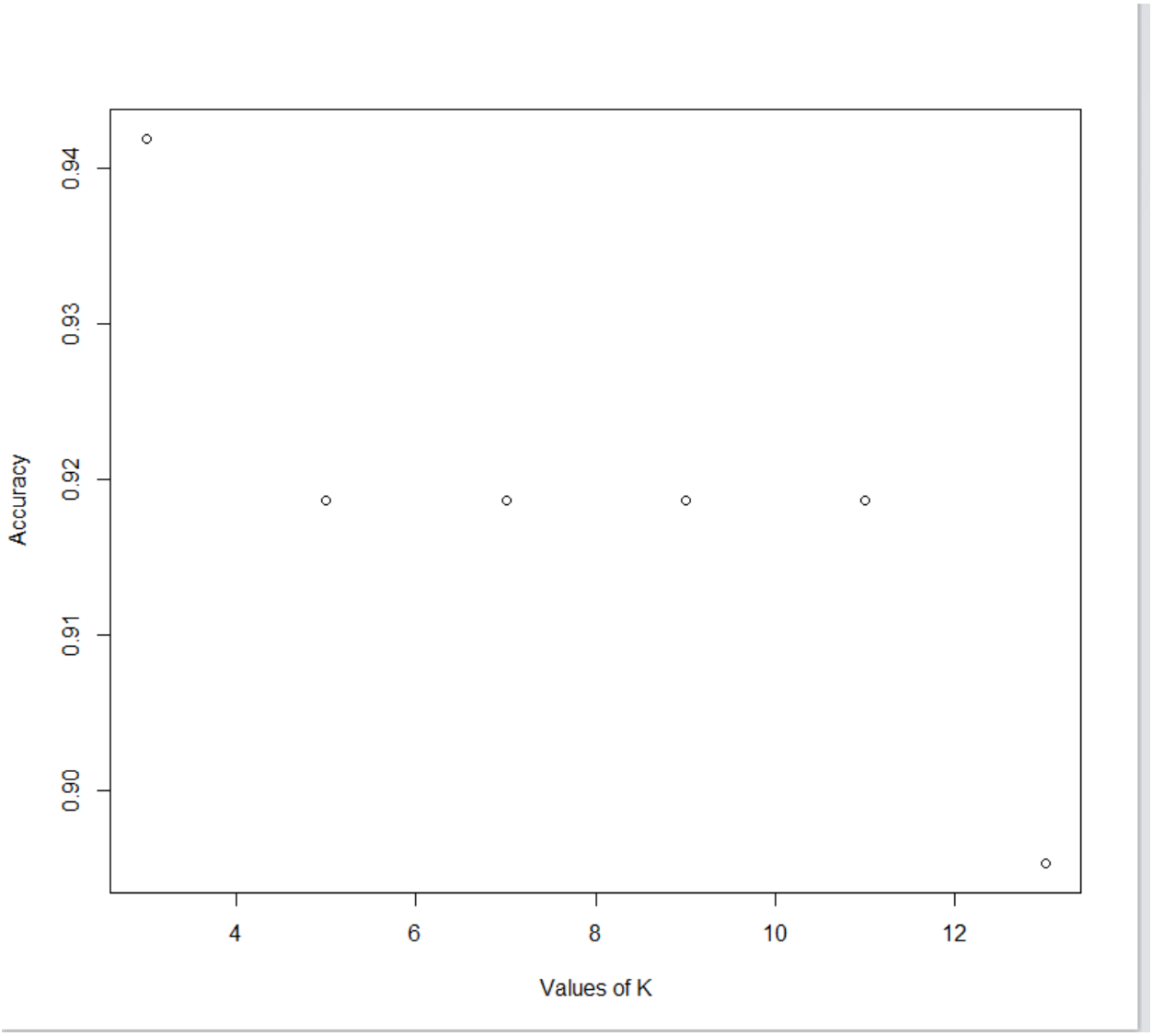
```
         0 59  3
         1  6 18

               Accuracy : 0.8953
                 95% CI : (0.8106, 0.951)
    No Information Rate : 0.7558
    P-Value [Acc > NIR] : 0.0009299

                  Kappa : 0.7296
 Mcnemar's Test P-Value : 0.5049851

            Sensitivity : 0.9077
            Specificity : 0.8571
         Pos Pred Value : 0.9516
         Neg Pred Value : 0.7500
             Prevalence : 0.7558
         Detection Rate : 0.6860
   Detection Prevalence : 0.7209
      Balanced Accuracy : 0.8824

       'Positive' Class : 0
```

Conclusion:-
Wrote the loop that computes the testing accuracy for odd k values ranging 3 to floor(sqrt(no of observations)) I have taken odd numbers ranging from 3 to floor(sqrt(no of observations) as reasonable value of k.Reason of taking only odd number is that there are no tie situations in predicting response vector values.So there are no ambiguity while choosing value for prediction.

Q2-G)

Code:-

```
seq<-c(3,5,7,9,11,13)

Accuracyvector<-integer()


for(val in seq){

knn_req<-knn(train=traindata,test=testdata,cl=trainres,k=val)

 GetAccuracy<-confusionMatrix(testres,knn_req)

 Accuracyvector<-c(Accuracyvector,GetAccuracy$overall[1])

 }

plot(seq,Accuracyvector,xlab="Values of K",ylab="Accuracy")
```



Conclusion:-

From the graph we can see that we can get maximum accuracy of prediction for k value=3.

Optimal value of k=3.

Q2-H)

Code:-

```
glassdata_normalize<-as.data.frame(lapply(glassdataframe[,c(2,3,4,5,6,7,8,9,10)],normalize))
dataframecoll=data.frame(glassdataframe$Type)
smp<-floor(0.60*nrow(glassdata_normalize))
set.seed(123)
```

```
train<-sample(seq_len(nrow(glassdata_normalize)),size=smp)

traindata<-glassdata_normalize[train,]

trainres<-(dataframecoll[train,])

testdata<-glassdata_normalize[-train,]

testres<-(dataframecoll[-train,])

library(class)

knn_req<-knn(train=traindata,test=testdata,cl=trainres,k=5)

confusionMatrix(testres,knn_req)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3  5  6  7
         1 15  8  0  0  0  0
         2  7 23  0  2  1  1
         3  3  2  0  0  0  0
         5  0  2  0  2  1  2
         6  0  1  0  0  1  2
         7  0  0  0  1  2 10

Overall Statistics

               Accuracy : 0.593
                 95% CI : (0.4817, 0.6978)
    No Information Rate : 0.4186
    P-Value [Acc > NIR] : 0.0008332

                  Kappa : 0.4371
 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 5 Class: 6 Class: 7
Sensitivity            0.6000   0.6389       NA  0.40000  0.20000   0.6667
Specificity            0.8689   0.7800  0.94186  0.93827  0.96296   0.9577
Pos Pred Value         0.6522   0.6765       NA  0.28571  0.25000   0.7692
Neg Pred Value         0.8413   0.7500       NA  0.96203  0.95122   0.9315
Prevalence             0.2907   0.4186  0.00000  0.05814  0.05814   0.1744
Detection Rate         0.1744   0.2674  0.00000  0.02326  0.01163   0.1163
Detection Prevalence   0.2674   0.3953  0.05814  0.08140  0.04651   0.1512
Balanced Accuracy      0.7344   0.7094       NA  0.66914  0.58148   0.8122
```

Conclusion:- Most frequent class is Glass Type(ranging from 1,2,3,4,5,6,7) in glassdataframe.

Predicting the value of the most frequent class(null accuracy) by choosing predictors from RI to Fe.

Accuracy for prediction we got is 59.3%

Q-Bonus)

Code:-

```
normalize<-function(x) {
 return ((x-min(x))/(max(x)-min(x)))
 }
glassdata_normalize<-as.data.frame(lapply(glassdataframe[,c(4,5,8)],normalize))
dataframecoll=data.frame(y)
smp<-floor(0.60*nrow(glassdata_normalize))
set.seed(123)
train<-sample(seq_len(nrow(glassdata_normalize)),size=smp)
traindata<-glassdata_normalize[train,]
trainres<-(dataframecoll[train,])
testdata<-glassdata_normalize[-train,]
testres<-(dataframecoll[-train,])
knn_req<-knn(train=traindata,test=testdata,cl=trainres,k=5)
table(testres,knn_req)
confusionMatrix(testres,knn_req)  #Accuracy 95.35%
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 58  4
         1  0 24

               Accuracy : 0.9535
                 95% CI : (0.8852, 0.9872)
    No Information Rate : 0.6744
    P-Value [Acc > NIR] : 2.479e-10

                  Kappa : 0.89
 Mcnemar's Test P-Value : 0.1336

            Sensitivity : 1.0000
            Specificity : 0.8571
         Pos Pred Value : 0.9355
         Neg Pred Value : 1.0000
```

```
             Prevalence : 0.6744
         Detection Rate : 0.6744
   Detection Prevalence : 0.7209
      Balanced Accuracy : 0.9286

       'Positive' Class : 0
```

Conclusion:-

I took Mg,Al and Ca as a good predictor and by redoing the exercise , I got higher accuracy

95.35% than if I would choose vector collection (RI,Na,Mg,Al,Si,K,Ca,Ba,Fe,Type) as a predictor.

References for understanding

http://www.r-bloggers.com/time-series-analysis-building-a-model-on-non-stationary-time-series/

https://www.youtube.com/watch?v=DkLNb0CXw84&ebc=ANyPxKrZ4XfRJK7TKuwf8JeLi5uoC2obJpTLkltZOD4-1g1PDNGrIILuON5QAqd1dQ9Bq9I39ybaDyUiicmDrA4qIh8Wg027_w

https://www.youtube.com/watch?v=GtgJEVxl7DY

statosphere.com.au/check-time-series-stationary-r

http://support.minitab.com/en-us/minitab/17/topic-library/modeling-statistics/time-series/time-series-models/what-are-ar-ma-and-arma/

http://www.r-bloggers.com/arma-models-for-trading/

https://drive.google.com/file/d/0BwogTI8d6EEiVjNUcFVSNjhlemc/edit?pref=2&pli=1