Question 1

a)

Read the data from .dat file into R, make it stationary and verify the same.

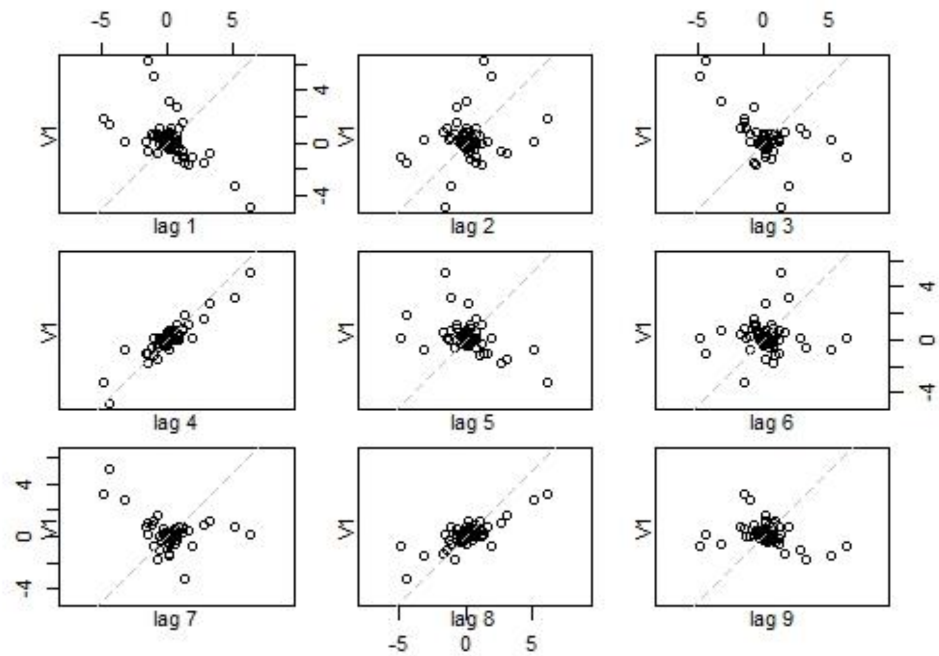
```
> install.packages("timeSeries")
> require(timeSeries)
> install.packages("tseries")
> require(tseries)
> jnjData<-read.table(file.choose(),header = FALSE)
> adf.test(diff(ts(data=jnjData,deltat = 1/4)))

Augmented Dickey-Fuller Test

data:  diff(ts(data = jnjData, deltat = 1/4))
Dickey-Fuller = -3.9886, Lag order = 4, p-value = 0.01421
alternative hypothesis: stationary

> kpss.test(diff(ts(data = jnjData,deltat = 1/4)))

        KPSS Test for Level Stationarity

data:  diff(ts(data = jnjData, deltat = 1/4))
KPSS Level = 0.075166, Truncation lag parameter = 2, p-value = 0.1

Warning message:
In kpss.test(diff(ts(data = jnjData, deltat = 1/4))) :
  p-value greater than printed p-value

> jnjDataDiff<-diff(ts(data = jnjData,deltat = 1/4))
```

Conclusion:

Based on the low p-value got in Dickey-Fuller Test and the higher p-value got in the KPSS Test, we can conclude that after doing ts(to remove trend) and diff(to remove seasonality), our time series is now stationary.

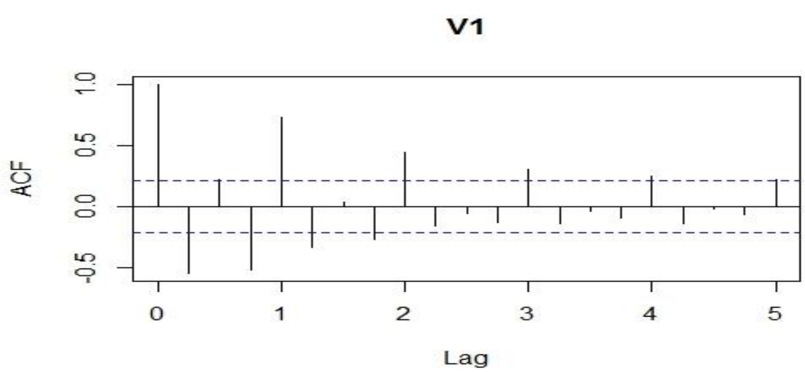
b)

```
> lag.plot(jnjDataDiff,9,do.lines=F)
```

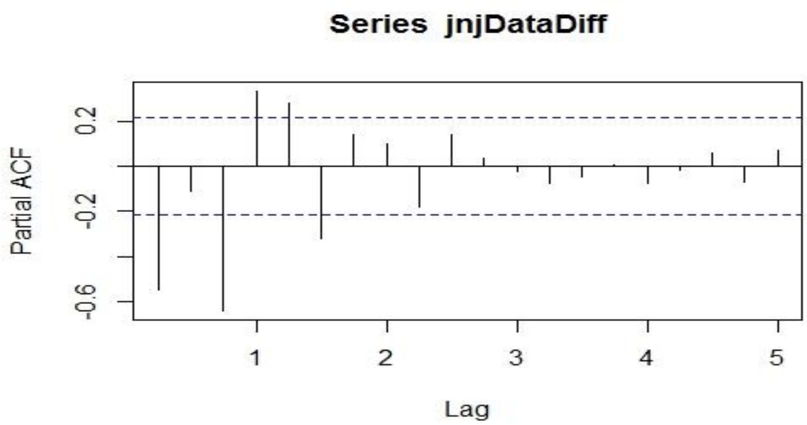


From the figure, we can see lag 4 & 8 to be more even as compared to other lags.

```
> acf(jnjDataDiff,lag.max = 20)
```



```
> pacf(jnjDataDiff,lag.max = 20)
```



Conclusion:

If the PACF of the differenced series displays a sharp cutoff and/or the lag-1 autocorrelation is positive--i.e., if the series appears slightly "underdifferenced"--then consider adding an AR term to the model. The lag at which the PACF cuts off is the indicated number of AR terms.

If the ACF of the differenced series displays a sharp cutoff and/or the lag-1 autocorrelation is negative--i.e., if the series appears slightly "overdifferenced"--then consider adding an MA term to the model. The lag at which the ACF cuts off is the indicated number of MA terms.

Hence, based on the above statement, AR terms are 2 & MA terms are 5.

N14724679
Kunal B. Relia (kbr263)

c)

```
> install.packages("forecast")
> require("forecast")

> arima(jnjDataDiff,order = c(2,0,5))

Call:
arima(x = jnjDataDiff, order = c(2, 0, 5))

Coefficients:
          ar1      ar2     ma1      ma2      ma3     ma4     ma5  intercept
      -1.2016  -0.2163  0.5671  -0.7052  -0.6466  0.8158  0.7980      0.171
s.e.   0.1663   0.1556  0.1430   0.1032   0.1101  0.1297  0.1375      0.046

sigma^2 estimated as 0.3159:  log likelihood = -77.26,  aic = 172.51

> arima(jnjDataDiff,order = c(1,0,4))

Call:
arima(x = jnjDataDiff, order = c(1, 0, 4))

Coefficients:
          ar1      ma1      ma2      ma3    ma4  intercept
      -0.6659  -0.2092  -0.5395  -0.2092  1.000     0.1697
s.e.   0.0974   0.1157   0.0806   0.0986  0.127     0.0407

sigma^2 estimated as 0.3614:  log likelihood = -82.24,  aic = 178.48

> arima(jnjDataDiff,order = c(0,0,3))

Call:
arima(x = jnjDataDiff, order = c(0, 0, 3))

Coefficients:
          ma1     ma2     ma3  intercept
      -1.4141  0.5212  0.2822     0.1722
s.e.   0.1821  0.2053  0.1008     0.0357

sigma^2 estimated as 0.7141:  log likelihood = -107.78,  aic = 225.57

> arima(jnjDataDiff,order = c(0,0,2))

Call:
arima(x = jnjDataDiff, order = c(0, 0, 2))

Coefficients:
          ma1     ma2  intercept
      -1.3973  0.7595     0.1722
s.e.   0.1125  0.0689     0.0359

sigma^2 estimated as 0.8172:  log likelihood = -110.75,  aic = 229.5

> arima(jnjDataDiff,order = c(0,0,1))

Call:
arima(x = jnjDataDiff, order = c(0, 0, 1))
```

```
Coefficients:
          ma1  intercept
      -0.6973      0.164
s.e.   0.0561      0.037

sigma^2 estimated as 1.171:  log likelihood = -124.67,  aic = 255.33

> arima(jnjDataDiff,order = c(0,0,0))

Call:
arima(x = jnjDataDiff, order = c(0, 0, 0))

Coefficients:
      intercept
         0.1313
s.e.     0.1554

sigma^2 estimated as 2.005:  log likelihood = -146.64,  aic = 297.28

> arima(jnjDataDiff,order = c(2,0,5))

Call:
arima(x = jnjDataDiff, order = c(2, 0, 5))

Coefficients:
          ar1      ar2     ma1      ma2      ma3     ma4     ma5  intercept
      -1.2016  -0.2163  0.5671  -0.7052  -0.6466  0.8158  0.7980      0.171
s.e.   0.1663   0.1556  0.1430   0.1032   0.1101  0.1297  0.1375      0.046

sigma^2 estimated as 0.3159:  log likelihood = -77.26,  aic = 172.51

> arima(jnjDataDiff,order = c(3,0,6))

Call:
arima(x = jnjDataDiff, order = c(3, 0, 6))

Coefficients:
          ar1      ar2      ar3     ma1     ma2     ma3     ma4      ma5
ma6  intercept
      -0.9463  -1.0097  -0.9103  0.3591  0.6388  0.0586  0.6628  -0.0598  0.4
533     0.1706
s.e.   0.0562   0.0340   0.0444  0.2820  0.1280  0.1258  0.2816   0.2082  0.1
958     0.0373

sigma^2 estimated as 0.1807:  log likelihood = -53.77,  aic = 129.54

> arima(jnjDataDiff,order = c(4,0,7))

Call:
arima(x = jnjDataDiff, order = c(4, 0, 7))

Coefficients:
          ar1      ar2      ar3     ar4      ma1     ma2      ma3     ma4
ma5     ma6
      -0.0338  -0.0732  -0.0108  0.8992  -0.8092  0.3323  -0.4105  0.4092  -0
.3818  0.3922
```

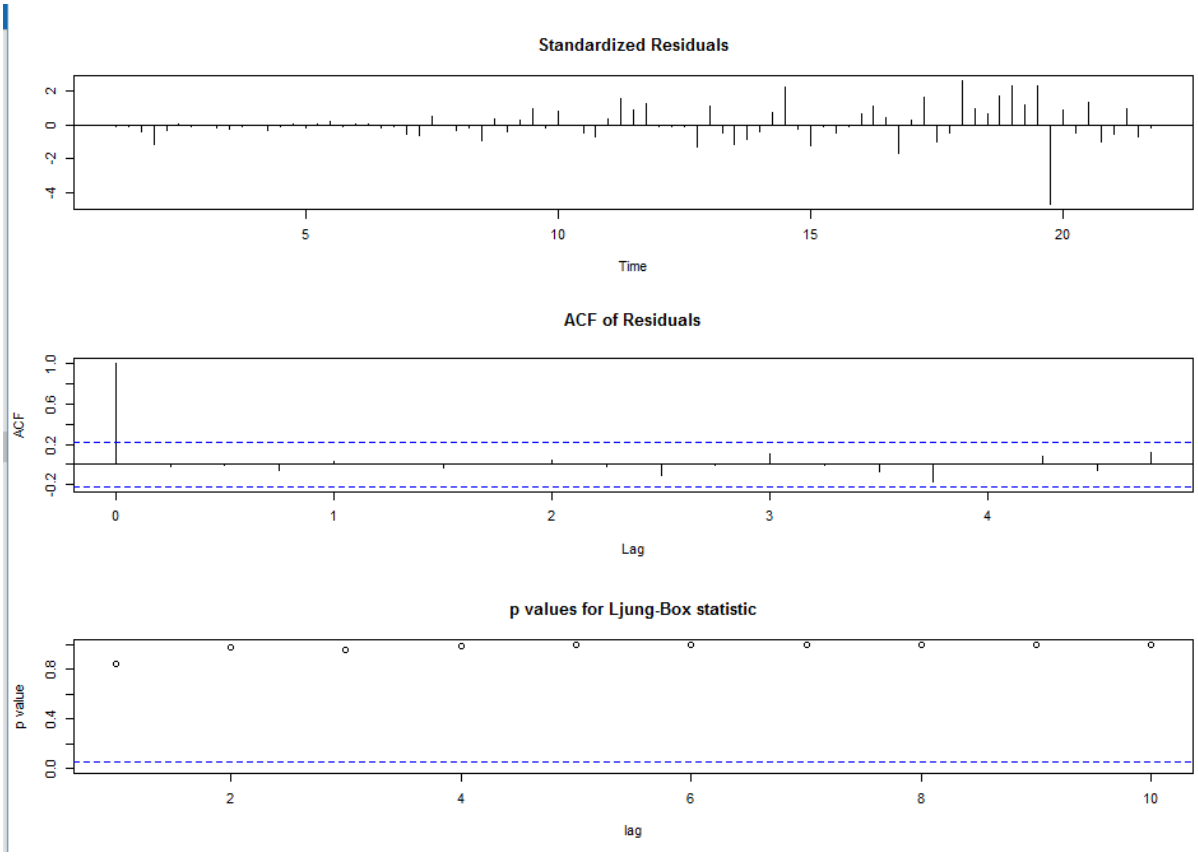
```
s.e.    0.0740   0.0588   0.0707  0.0591   0.1326  0.1509   0.1370  0.1726   0
.2481  0.2465
          ma7  intercept
      -0.0185      0.173
s.e.   0.1717      0.086

sigma^2 estimated as 0.1547:  log likelihood = -45.95,  aic = 117.9

> arima(jnjDataDiff,order = c(5,0,8))

Call:
arima(x = jnjDataDiff, order = c(5, 0, 8))

Coefficients:
          ar1      ar2      ar3     ar4      ar5      ma1     ma2      ma3
ma4      ma5     ma6
      -0.2815  -0.1750  -0.0169  0.8058  0.2362  -0.4903  0.2319  -0.5515  0.
6210  -0.6065  0.6150
s.e.   0.3242   0.0998   0.1027  0.1014  0.2894   0.3341  0.2623   0.1810  0.
2153   0.2375  0.2429
          ma7     ma8  intercept
      -0.1766  0.2034     0.1666
s.e.   0.2840  0.1721     0.0730

sigma^2 estimated as 0.1438:  log likelihood = -45.33,  aic = 120.67
```

Conclusion:

As shown above, we started with AR=2 & MA=5 for which we got aic = 172.51. Then we tried various values of AR & MA. After seeing the parabolic trend in aic values, we concluded that for AR=4 & MA=7, we get an aic value = 117.9 which is the lowest. Hence, ARIMA(4,0,7) is the best model for our data.

d)

```
> tsdiag(arima(jnjDataDiff,order = c(4,0,7)))
```




```
> Box.test(arima(jnjDataDiff,order = c(4,0,7))$residuals,lag=1)

        Box-Pierce test

data:  arima(jnjDataDiff, order = c(4, 0, 7))$residuals
X-squared = 0.035342, df = 1, p-value = 0.8509
```

Conclusion:

Based on the plots received in tsdiag() especially where lag in ACF of residual is 0 and the high p-value received in Box.test, we can conclude that the residuals are NOT stationary and this means relationship exists.

e)

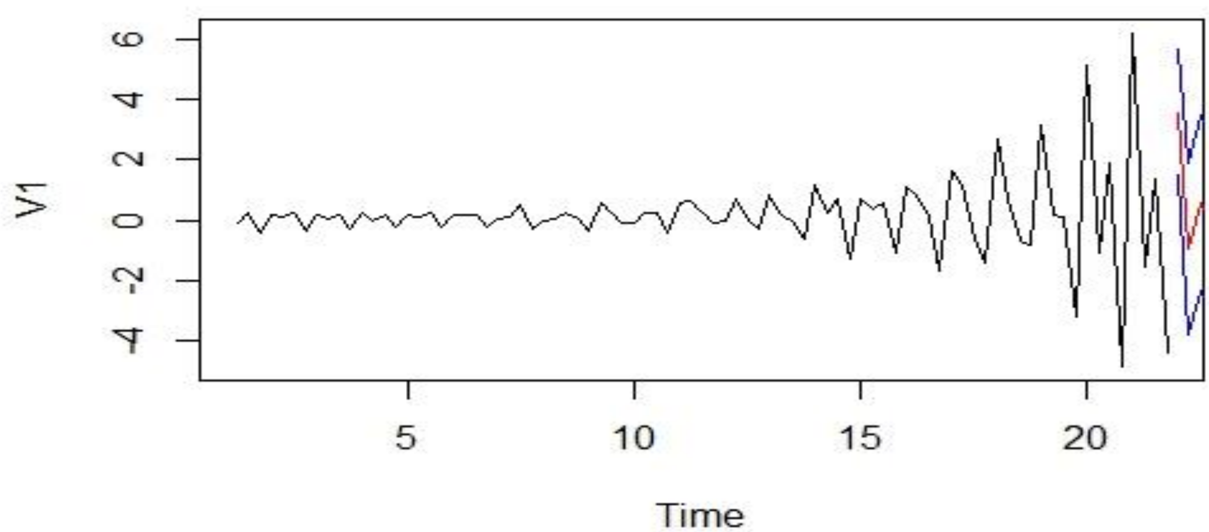
```
> predict(arima(jnjDataDiff, order = c(1,0,1)),n.ahead=4)
$pred
          Qtr1        Qtr2        Qtr3        Qtr4
22  3.57354343 -0.97381367  0.54697927  0.03837369

$se
       Qtr1     Qtr2     Qtr3     Qtr4
22 1.042805 1.421325 1.457562 1.461559
```

Conclusion:

For Arima(1,0,1), the prediction values are as shown above.

f)

```
> plot(jnjDataDiff)
> lines(predict(arima(jnjDataDiff, order = c(1,0,1)),n.ahead=4)$pred,col="red
")
> lines(predict(arima(jnjDataDiff, order = c(1,0,1)),n.ahead=4)$pred+2*predic
t(arima(jnjDataDiff, order = c(1,0,1)),n.ahead=4)$se,col="blue")
> lines(predict(arima(jnjDataDiff, order = c(1,0,1)),n.ahead=4)$pred-2*predic
t(arima(jnjDataDiff, order = c(1,0,1)),n.ahead=4)$se,col="blue")
```



Question 2

a)

```
> glassData<-read.table(file.choose(),header=F,sep=",")
> glassDataFrame<-data.frame(glassData)
> collectionnos<-c(1,2,3,4)
> glassDataFrame["bi"]<-ifelse(glassDataFrame$V11 %in% collectionnos,0,1)
> header<-c("id","RI","Na","Mg","Al","Si","K","Ca","Ba","Fe","Type","bi")
> colnames(glassDataFrame)<-unlist(header)
> View(glassDataFrame)
```

glassDataFrame

Filter

	id	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type	bi
149	149	1.51670	13.24	3.57	1.38	72.70	0.56	8.44	0.00	0.10	3	0
150	150	1.51643	12.16	3.52	1.35	72.89	0.57	8.53	0.00	0.00	3	0
151	151	1.51665	13.14	3.45	1.76	72.48	0.60	8.38	0.00	0.17	3	0
152	152	1.52127	14.32	3.90	0.83	71.50	0.00	9.49	0.00	0.00	3	0
153	153	1.51779	13.64	3.65	0.65	73.00	0.06	8.93	0.00	0.00	3	0
154	154	1.51610	13.42	3.40	1.22	72.69	0.59	8.32	0.00	0.00	3	0
155	155	1.51694	12.86	3.58	1.31	72.61	0.61	8.79	0.00	0.00	3	0
156	156	1.51646	13.04	3.40	1.26	73.01	0.52	8.58	0.00	0.00	3	0
157	157	1.51655	13.41	3.39	1.28	72.64	0.52	8.65	0.00	0.00	3	0
158	158	1.52121	14.03	3.76	0.58	71.79	0.11	9.65	0.00	0.00	3	0
159	159	1.51776	13.53	3.41	1.52	72.04	0.58	8.79	0.00	0.00	3	0
160	160	1.51796	13.50	3.36	1.63	71.94	0.57	8.81	0.00	0.09	3	0
161	161	1.51832	13.33	3.34	1.54	72.14	0.56	8.99	0.00	0.00	3	0
162	162	1.51934	13.64	3.54	0.75	72.65	0.16	8.89	0.15	0.24	3	0
163	163	1.52211	14.19	3.78	0.91	71.36	0.23	9.14	0.00	0.37	3	0
164	164	1.51514	14.01	2.68	3.50	69.89	1.68	5.87	2.20	0.00	5	1
165	165	1.51915	12.73	1.85	1.86	72.69	0.60	10.09	0.00	0.00	5	1
166	166	1.52171	11.56	1.88	1.56	72.86	0.47	11.41	0.00	0.00	5	1
167	167	1.52151	11.03	1.71	1.56	73.44	0.58	11.62	0.00	0.00	5	1
168	168	1.51969	12.64	0.00	1.65	73.75	0.38	11.53	0.00	0.00	5	1
169	169	1.51666	12.86	0.00	1.83	73.88	0.97	10.17	0.00	0.00	5	1
170	170	1.51994	13.27	0.00	1.76	73.03	0.47	11.32	0.00	0.00	5	1
171	171	1.52369	13.44	0.00	1.58	72.22	0.32	12.24	0.00	0.00	5	1
172	172	1.51316	13.02	0.00	3.04	70.48	6.21	6.96	0.00	0.00	5	1
173	173	1.51321	13.00	0.00	3.02	70.70	6.21	6.93	0.00	0.00	5	1
174	174	1.52043	13.38	0.00	1.40	72.25	0.33	12.50	0.00	0.00	5	1

Showing 148 to 175 of 214 entries

b)

Create a feature Matrix "fea" using all the features

```
> attach(glassDataFrame)
> fea=matrix(c(RI,Na,Mg,Al,Si,K,Ca,Ba,Fe),nrow = 214, ncol=9,byrow = F)
```

Create a response vector "y" using "bi"

```
> model=lm(glassDataFrame$bi~RI+Na+Mg+Al+Si+K+Ca+Ba+Fe)
> y=model$model$'glassDataFrame$bi'
```

c)

Based on the "Pareto Principle" of 80:20, we can divide the divide the data in to 80:20 ratio. But to be on a safer margin, the data here is divided into a ratio of 75:25 :: training data : testing data.

```
> predColDF<-data.frame(y)
> normalize<-function(x)
 {
     return ((x-min(x))/(max(x)-min(x)))
 }
> normalizeGlassData<-as.data.frame(lapply(glassDataFrame[,c(2,3,4,5,6,7,8,9,
10)],normalize))
> sampleTrainSet<-floor(0.75*nrow(normalizeGlassData))
> set.seed(123)
> train<-sample(seq_len(nrow(normalizeGlassData)),size=sampleTrainSet)
> traindata<-normalizeGlassData[train,]
> trainres<-(predColDF[train,])
> testdata<-normalizeGlassData[-train,]
> testres<-(predColDF[-train,])
```

Conclusion:

The training set:testing set ratio is taken as 75:25. Mostly 80:20 or 70:30 test ratio is taken, so as a safe margin 75:25 ratio is chosen.

This analysis is based on experimental performance of the model trained on the training partition when applied to the test partition. The bands show the maximum and minimum performance recorded for a particular training set size. When the training set is below 10%, the accuracy and precision are both poor. This shows the importance of having sufficient data for training. As the training set increases, the accuracy of the model increases very gradually but as it approaches 100%, the precision decreases. This shows the importance of having sufficient test data.

By inspection, a training partition size between 40% and 80% looks like it would give a good combination of accuracy and precision. These results were cited from *information-gain.blogspot.com*.

d)

```
> install.packages("class")
> require(class)
> knn(train=traindata, test=testdata,cl=trainres,k=5)
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0 0
0 0 0 0 1 0 1 1 0 1 1
[48] 1 1 1 1 1 1 1
Levels: 0 1
```

e)

```
> install.packages("e1071")
> require(e1071)
> install.packages("caret")
> require(caret)
> confusionMatrix(testres,knn(train=traindata, test=testdata,cl=trainres,k=5)
)
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 37  3
         1  2 12

               Accuracy : 0.9074
                 95% CI : (0.797, 0.9692)
    No Information Rate : 0.7222
    P-Value [Acc > NIR] : 0.0008214

                  Kappa : 0.7644
 Mcnemar's Test P-Value : 1.0000000

            Sensitivity : 0.9487
            Specificity : 0.8000
         Pos Pred Value : 0.9250
         Neg Pred Value : 0.8571
             Prevalence : 0.7222
         Detection Rate : 0.6852
   Detection Prevalence : 0.7407
      Balanced Accuracy : 0.8744

       'Positive' Class : 0
```

Conclusion:

Various inferences can be made from the above got confusionMatrix.

```
          Reference
Prediction  0  1
         0  A  B
         1  C  D
```

Sensitivity = A/(A+C)

Specificity = D/(B+D)

Prevalence = (A+C)/(A+B+C+D)

PPV = (sensitivity * Prevalence)/((sensitivity*Prevalence) + ((1-specificity)*(1-Prevalence)))

NPV = (specificity * (1-Prevalence))/(((1-sensitivity)*Prevalence) + ((specificity)*(1-Prevalence)))

Detection Rate = A/(A+B+C+D)

Detection Prevalence = (A+B)/(A+B+C+D)

Balanced Accuracy = (Sensitivity+Specificity)/2

Hence, as per our output accuracy = (37+12)/(37+12+3+2) = 49/54 = 90.74%

f)

```
> kValues<-c(1,2,3,4,6,7,9,12,15,18,21)
> for(val in kValues)
 {
 print("k=")
 print(val)
 print(confusionMatrix(testres,knn(train=traindata, test=testdata,cl=trainres
,k=val)))
 }
[1] "k="
[1] 1
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 39  1
         1  1 13

               Accuracy : 0.963
                 95% CI : (0.8725, 0.9955)
    No Information Rate : 0.7407
    P-Value [Acc > NIR] : 1.788e-05

                  Kappa : 0.9036
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9750
            Specificity : 0.9286
         Pos Pred Value : 0.9750
         Neg Pred Value : 0.9286
             Prevalence : 0.7407
         Detection Rate : 0.7222
   Detection Prevalence : 0.7407
      Balanced Accuracy : 0.9518

       'Positive' Class : 0

[1] "k="
[1] 2
Confusion Matrix and Statistics
```

```
          Reference
Prediction  0  1
         0 39  1
         1  3 11

               Accuracy : 0.9259
                 95% CI : (0.8211, 0.9794)
    No Information Rate : 0.7778
    P-Value [Acc > NIR] : 0.0036

                  Kappa : 0.7978
 Mcnemar's Test P-Value : 0.6171

            Sensitivity : 0.9286
            Specificity : 0.9167
         Pos Pred Value : 0.9750
         Neg Pred Value : 0.7857
             Prevalence : 0.7778
         Detection Rate : 0.7222
   Detection Prevalence : 0.7407
      Balanced Accuracy : 0.9226

       'Positive' Class : 0

[1] "k="
[1] 3
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 38  2
         1  1 13

               Accuracy : 0.9444
                 95% CI : (0.8461, 0.9884)
    No Information Rate : 0.7222
    P-Value [Acc > NIR] : 3.84e-05

                  Kappa : 0.8586
 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9744
            Specificity : 0.8667
         Pos Pred Value : 0.9500
         Neg Pred Value : 0.9286
             Prevalence : 0.7222
         Detection Rate : 0.7037
   Detection Prevalence : 0.7407
      Balanced Accuracy : 0.9205

       'Positive' Class : 0

[1] "k="
[1] 4
Confusion Matrix and Statistics

          Reference
```

```
Prediction  0  1
         0 36  4
         1  2 12

               Accuracy : 0.8889
                 95% CI : (0.7737, 0.9581)
    No Information Rate : 0.7037
    P-Value [Acc > NIR] : 0.001136

                  Kappa : 0.7235
 Mcnemar's Test P-Value : 0.683091

            Sensitivity : 0.9474
            Specificity : 0.7500
         Pos Pred Value : 0.9000
         Neg Pred Value : 0.8571
             Prevalence : 0.7037
         Detection Rate : 0.6667
   Detection Prevalence : 0.7407
      Balanced Accuracy : 0.8487

       'Positive' Class : 0

[1] "k="
[1] 6
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 37  3
         1  1 13

               Accuracy : 0.9259
                 95% CI : (0.8211, 0.9794)
    No Information Rate : 0.7037
    P-Value [Acc > NIR] : 6.93e-05

                  Kappa : 0.8157
 Mcnemar's Test P-Value : 0.6171

            Sensitivity : 0.9737
            Specificity : 0.8125
         Pos Pred Value : 0.9250
         Neg Pred Value : 0.9286
             Prevalence : 0.7037
         Detection Rate : 0.6852
   Detection Prevalence : 0.7407
      Balanced Accuracy : 0.8931

       'Positive' Class : 0

[1] "k="
[1] 7
Confusion Matrix and Statistics

          Reference
Prediction  0  1
```

```
         0 36  4
         1  2 12

               Accuracy : 0.8889
                 95% CI : (0.7737, 0.9581)
    No Information Rate : 0.7037
    P-Value [Acc > NIR] : 0.001136

                  Kappa : 0.7235
 Mcnemar's Test P-Value : 0.683091

            Sensitivity : 0.9474
            Specificity : 0.7500
         Pos Pred Value : 0.9000
         Neg Pred Value : 0.8571
             Prevalence : 0.7037
         Detection Rate : 0.6667
   Detection Prevalence : 0.7407
      Balanced Accuracy : 0.8487

       'Positive' Class : 0

[1] "k="
[1] 9
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 36  4
         1  2 12

               Accuracy : 0.8889
                 95% CI : (0.7737, 0.9581)
    No Information Rate : 0.7037
    P-Value [Acc > NIR] : 0.001136

                  Kappa : 0.7235
 Mcnemar's Test P-Value : 0.683091

            Sensitivity : 0.9474
            Specificity : 0.7500
         Pos Pred Value : 0.9000
         Neg Pred Value : 0.8571
             Prevalence : 0.7037
         Detection Rate : 0.6667
   Detection Prevalence : 0.7407
      Balanced Accuracy : 0.8487

       'Positive' Class : 0

[1] "k="
[1] 12
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 36  4
```

```
         1  3 11

               Accuracy : 0.8704
                 95% CI : (0.751, 0.9463)
    No Information Rate : 0.7222
    P-Value [Acc > NIR] : 0.007921

                  Kappa : 0.6702
 Mcnemar's Test P-Value : 1.000000

            Sensitivity : 0.9231
            Specificity : 0.7333
         Pos Pred Value : 0.9000
         Neg Pred Value : 0.7857
             Prevalence : 0.7222
         Detection Rate : 0.6667
   Detection Prevalence : 0.7407
      Balanced Accuracy : 0.8282

       'Positive' Class : 0

[1] "k="
[1] 15
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 36  4
         1  3 11

               Accuracy : 0.8704
                 95% CI : (0.751, 0.9463)
    No Information Rate : 0.7222
    P-Value [Acc > NIR] : 0.007921

                  Kappa : 0.6702
 Mcnemar's Test P-Value : 1.000000

            Sensitivity : 0.9231
            Specificity : 0.7333
         Pos Pred Value : 0.9000
         Neg Pred Value : 0.7857
             Prevalence : 0.7222
         Detection Rate : 0.6667
   Detection Prevalence : 0.7407
      Balanced Accuracy : 0.8282

       'Positive' Class : 0

[1] "k="
[1] 18
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 36  4
         1  3 11
```

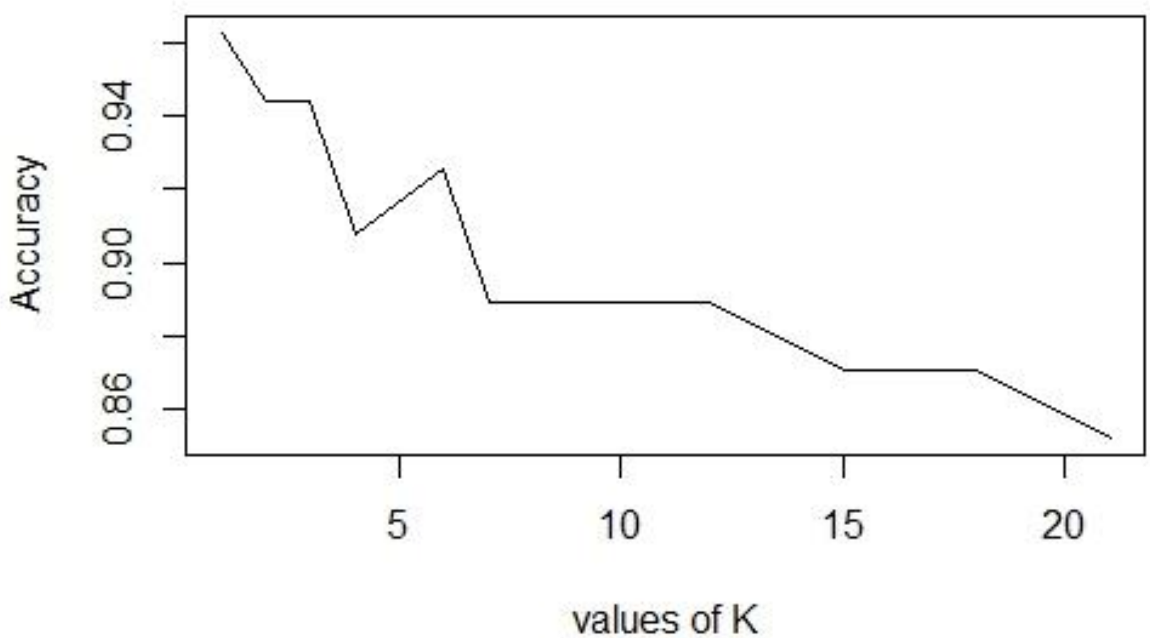
```
               Accuracy : 0.8704
                 95% CI : (0.751, 0.9463)
    No Information Rate : 0.7222
    P-Value [Acc > NIR] : 0.007921

                  Kappa : 0.6702
 Mcnemar's Test P-Value : 1.000000

            Sensitivity : 0.9231
            Specificity : 0.7333
         Pos Pred Value : 0.9000
         Neg Pred Value : 0.7857
             Prevalence : 0.7222
         Detection Rate : 0.6667
   Detection Prevalence : 0.7407
      Balanced Accuracy : 0.8282

       'Positive' Class : 0

[1] "k="
[1] 21
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 36  4
         1  4 10

               Accuracy : 0.8519
                 95% CI : (0.7288, 0.9338)
    No Information Rate : 0.7407
    P-Value [Acc > NIR] : 0.03833

                  Kappa : 0.6143
 Mcnemar's Test P-Value : 1.00000

            Sensitivity : 0.9000
            Specificity : 0.7143
         Pos Pred Value : 0.9000
         Neg Pred Value : 0.7143
             Prevalence : 0.7407
         Detection Rate : 0.6667
   Detection Prevalence : 0.7407
      Balanced Accuracy : 0.8071

       'Positive' Class : 0
```

Conclusion:

From the above observations, we can conclude that we get the maximum accuracy for k=1. Then all the corresponding values go on decreasing, not strictly decreasing, but definitely lesser than accuracy for k=1. Ideally, the likeliness that accuracy is high for lower k values is more hence the values have been chosen accordingly.

g)

```
> acc<-integer()
> for(val in kValues)
 {
 ans<-confusionMatrix(testres,knn(train=traindata, test=testdata,cl=trainres,
k=val))
 acc<-c(acc,ans$overall[1])
 }
> plot(kValues,acc,xlab="values of K",ylab="Accuracy",type="l")
```



Conclusion:

From the above observations of the graph, we can conclude that we get the maximum accuracy for k=1. The most optimal value of k is 1.

h)

```
> mean(testres)
[1] 0.2592593
```

Conclusion:

The testing accuracy that could be achieved is 74.08% [(1-0.2592)*100]. This can be done by always predicting the most frequent class in the testing set. The "bi" column that was created by us used in this. We find its mean to check for the null accuracy.

BONUS

```
> attach(glassDataFrame)
> fea=matrix(c(RI,Na,Mg,Al,Si,K,Ca,Ba,Fe),nrow = 214, ncol=9,byrow = F)
> model=lm(glassDataFrame$bi~Na+Si+K+Ca)
> summary(model)

Call:
lm(formula = glassDataFrame$bi ~ Na + Si + K + Ca)

Residuals:
     Min       1Q   Median       3Q      Max 
-0.88329 -0.16059 -0.08745  0.01973  1.19781 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) -22.06865    2.54430  -8.674 1.18e-15 ***
Na            0.38615    0.03187  12.117  < 2e-16 ***
Si            0.21597    0.03171   6.811 1.01e-10 ***
K             0.31286    0.04175   7.493 1.87e-12 ***
Ca            0.14334    0.01927   7.439 2.58e-12 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3234 on 209 degrees of freedom
Multiple R-squared:  0.4371,	Adjusted R-squared:  0.4263 
F-statistic: 40.58 on 4 and 209 DF,  p-value: < 2.2e-16

> y=model$model$'glassDataFrame$bi'
> predColDF<-data.frame(y)
> normalize<-function(x)
+ {
+ return ((x-min(x))/(max(x)-min(x)))
+ }
> normalizeGlassData<-as.data.frame(lapply(glassDataFrame[,c(3,6,7,8)],normal
ize))
> sampleTrainSet<-floor(0.75*nrow(normalizeGlassData))
> set.seed(123)
> train<-sample(seq_len(nrow(normalizeGlassData)),size=sampleTrainSet)
> traindata<-normalizeGlassData[train,]
> trainres<-(predColDF[train,])
> testdata<-normalizeGlassData[-train,]
> testres<-(predColDF[-train,])
> confusionMatrix(testres,knn(train=traindata, test=testdata,cl=trainres,k=5)
)
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 39  1
         1  2 12
                                          
               Accuracy : 0.9444          
                 95% CI : (0.8461, 0.9884)
    No Information Rate : 0.7593          
    P-Value [Acc > NIR] : 0.0003311       
```

```
                  Kappa : 0.8519
 Mcnemar's Test P-Value : 1.0000000

            Sensitivity : 0.9512
            Specificity : 0.9231
         Pos Pred Value : 0.9750
         Neg Pred Value : 0.8571
             Prevalence : 0.7593
         Detection Rate : 0.7222
   Detection Prevalence : 0.7407
      Balanced Accuracy : 0.9371

       'Positive' Class : 0
```

Conclusion:

Different predictors lead to different accuracy levels. Here, in this case where I use the core components that make up glass (Si,K,Na,Ca) led us to a higher level of accuracy for k=5 (94.44%). This can lead us to a conclusion that it is indeed these 4 components that contribute to determine the type of glass it is and not all the elements. These 4 elements are the main predictors as the glass is mainly composed of these elements as its core. This idea is further validated by the linear regression done by me before proceeding to check the actual accuracy. In short, this choice of predictors works out well.

Assignment 2 N14724679
Foundations of Data Science Kunal B. Relia (kbr263)

Group Work:

Some of the questions of this assignment were discussed with Fenil Tailor. Only the idea of solving a few questions were discussed by us followed by individual application of the ideas.