7BUIS008C.2 Data Mining and Machine Learning

Coursework One

Vishvaka Neomal Ranasinghe
2019677 (IIT)
w1790596 (UOW)

1st Objective (partitioning clustering)

Data pre-processing

- First, I've converted the xlxs format file to csv since it's much easier to work with in R Studio.
- Next, I imported the csv file into R Studio and check the column names to see if it matches what is given in the coursework description.

```
> library(NbClust)
> whiteWine<-read.csv("D:/Projects/WhiteWine.csv")
> names(whiteWine)
 [1] "fixed.acidity"        "volatile.acidity"     "citric.acid"          "residual.sugar"       "chlorides"
 [6] "free.sulfur.dioxide"  "total.sulfur.dioxide" "density"              "pH"                   "sulphates"
[11] "alcohol"              "quality"
```

- Then I needed to know the types of variables and if the variable values are normalized.

```
> str(whiteWine)
'data.frame':   4898 obs. of  12 variables:
 $ fixed.acidity       : num  7 6.3 8.1 7.2 7.2 8.1 6.2 7 6.3 8.1 ...
 $ volatile.acidity    : num  0.27 0.3 0.28 0.23 0.23 0.28 0.32 0.27 0.3 0.22 ...
 $ citric.acid         : num  0.36 0.34 0.4 0.32 0.32 0.4 0.16 0.36 0.34 0.43 ...
 $ residual.sugar      : num  20.7 1.6 6.9 8.5 8.5 6.9 7 20.7 1.6 1.5 ...
 $ chlorides           : num  0.045 0.049 0.05 0.058 0.058 0.05 0.045 0.045 0.049 0.044 ...
 $ free.sulfur.dioxide : num  45 14 30 47 47 30 30 45 14 28 ...
 $ total.sulfur.dioxide: num  170 132 97 186 186 97 136 170 132 129 ...
 $ density             : num  1.001 0.994 0.995 0.996 0.996 ...
 $ pH                  : num  3 3.3 3.26 3.19 3.19 3.26 3.18 3 3.3 3.22 ...
 $ sulphates           : num  0.45 0.49 0.44 0.4 0.4 0.44 0.47 0.45 0.49 0.45 ...
 $ alcohol             : num  8.8 9.5 10.1 9.9 9.9 10.1 9.6 8.8 9.5 11 ...
 $ quality             : int  6 6 6 6 6 6 6 6 6 6 ...
```

```
> summary(whiteWine)
 fixed.acidity    volatile.acidity  citric.acid     residual.sugar     chlorides         free.sulfur.dioxide total.sulfur.dioxide
 Min.   : 3.800   Min.   :0.0800   Min.   :0.0000   Min.   : 0.600   Min.   :0.00900   Min.   :  2.00      Min.   :  9.0
 1st Qu.: 6.300   1st Qu.:0.2100   1st Qu.:0.2700   1st Qu.: 1.700   1st Qu.:0.03600   1st Qu.: 23.00      1st Qu.:108.0
 Median : 6.800   Median :0.2600   Median :0.3200   Median : 5.200   Median :0.04300   Median : 34.00      Median :134.0
 Mean   : 6.855   Mean   :0.2782   Mean   :0.3342   Mean   : 6.391   Mean   :0.04577   Mean   : 35.31      Mean   :138.4
 3rd Qu.: 7.300   3rd Qu.:0.3200   3rd Qu.:0.3900   3rd Qu.: 9.900   3rd Qu.:0.05000   3rd Qu.: 46.00      3rd Qu.:167.0
 Max.   :14.200   Max.   :1.1000   Max.   :1.6600   Max.   :65.800   Max.   :0.34600   Max.   :289.00      Max.   :440.0
    density             pH          sulphates         alcohol          quality
 Min.   :0.9871   Min.   :2.720   Min.   :0.2200   Min.   : 8.00   Min.   :3.000
 1st Qu.:0.9917   1st Qu.:3.090   1st Qu.:0.4100   1st Qu.: 9.50   1st Qu.:5.000
 Median :0.9937   Median :3.180   Median :0.4700   Median :10.40   Median :6.000
 Mean   :0.9940   Mean   :3.188   Mean   :0.4898   Mean   :10.51   Mean   :5.878
 3rd Qu.:0.9961   3rd Qu.:3.280   3rd Qu.:0.5500   3rd Qu.:11.40   3rd Qu.:6.000
 Max.   :1.0390   Max.   :3.820   Max.   :1.0800   Max.   :14.20   Max.   :9.000
```

- This dataset contains 11 numerical variables and 1 categorical variable (quality). Based on the summary we can see that the variables are not normalized.

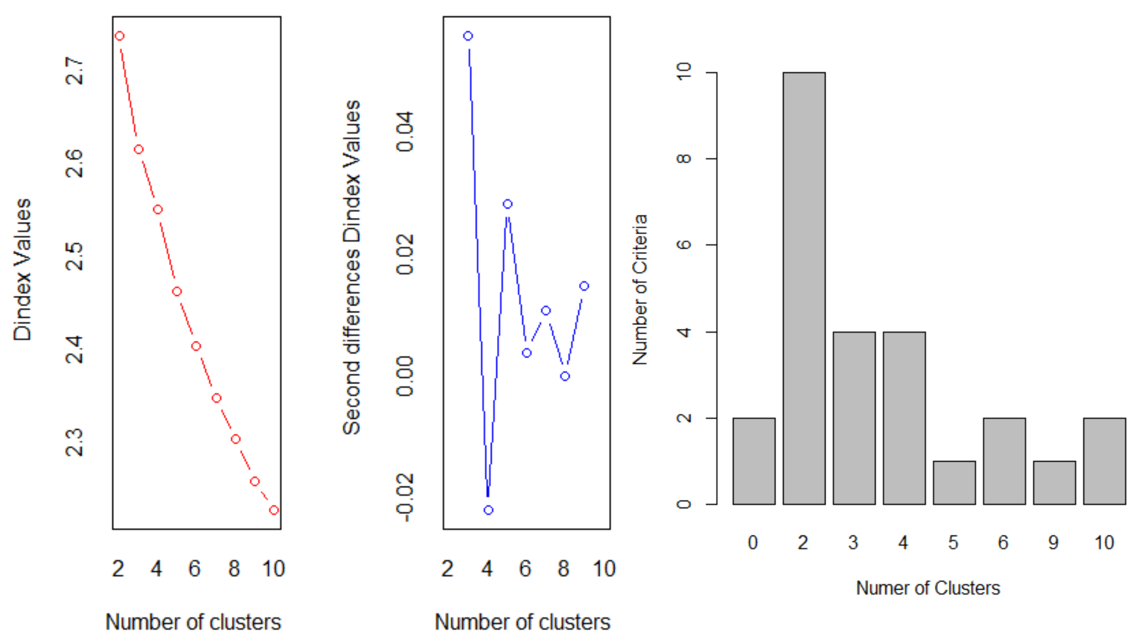
- So, I've scaled the data while removing the quality variable column.

```
> data.train <- scale(whiteWine[1:11])
> summary(data.train)
 fixed.acidity       volatile.acidity   citric.acid       residual.sugar      chlorides         free.sulfur.dioxide total.sulfur.dioxide
 Min.    :-3.61998   Min.    :-1.9668   Min.    :-2.7615   Min.    :-1.1418   Min.    :-1.6831   Min.    :-1.95848    Min.    :-3.0439
 1st Qu.:-0.65743   1st Qu.:-0.6770   1st Qu.:-0.5304   1st Qu.:-0.9250   1st Qu.:-0.4473   1st Qu.:-0.72370    1st Qu.:-0.7144
 Median :-0.06492   Median :-0.1810   Median :-0.1173   Median :-0.2349   Median :-0.1269   Median :-0.07691    Median :-0.1026
 Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000    Mean   : 0.0000
 3rd Qu.: 0.52758   3rd Qu.: 0.4143   3rd Qu.: 0.4612   3rd Qu.: 0.6917   3rd Qu.: 0.1935   3rd Qu.: 0.62867    3rd Qu.: 0.6739
 Max.   : 8.70422   Max.   : 8.1528   Max.   :10.9553   Max.   :11.7129   Max.   :13.7417   Max.   :14.91679    Max.   : 7.0977
    density               pH              sulphates          alcohol
 Min.   :-2.31280   Min.   :-3.10109   Min.   :-2.3645   Min.   :-2.04309
 1st Qu.:-0.77063   1st Qu.:-0.65077   1st Qu.:-0.6996   1st Qu.:-0.82419
 Median :-0.09608   Median :-0.05475   Median :-0.1739   Median :-0.09285
 Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.00000
 3rd Qu.: 0.69298   3rd Qu.: 0.60750   3rd Qu.: 0.5271   3rd Qu.: 0.71974
 Max.   :15.02976   Max.   : 4.18365   Max.   : 5.1711   Max.   : 2.99502
```

Finding the ideal number of clusters

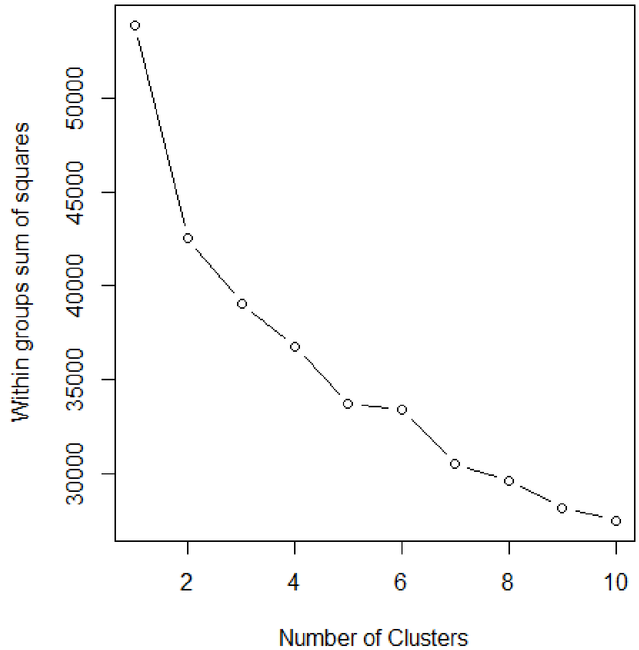
Using `NbClust` package to find the ideal number of clusters between range of 2 to 10 clusters using Euclidean distance.

```
> set.seed(1024)
> data.nc <- NbClust(data=data.train, distance="euclidean", min.nc=2, max.nc=10, method="kmeans", index="all")
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot.

*** : The D index is a graphical method of determining the number of clusters.
                In the plot of D index, we seek a significant knee (the significant peak in Dindex
                second differences plot) that corresponds to a significant increase of the value of
                the measure.

*******************************************************************
* Among all indices:
* 10 proposed 2 as the best number of clusters
* 4 proposed 3 as the best number of clusters
* 4 proposed 4 as the best number of clusters
* 1 proposed 5 as the best number of clusters
* 2 proposed 6 as the best number of clusters
* 1 proposed 9 as the best number of clusters
* 2 proposed 10 as the best number of clusters

                   ***** Conclusion *****

* According to the majority rule, the best number of clusters is  2


*******************************************************************
```



Its conclusion was to have 2 clusters. In order to verify this further I check the sum of square error plot to identify the bend.

```
> wss <- 0
> for (i in 1:10){
+   wss[i] <-
+     sum(kmeans(data.train, centers=i)$withinss)
+ }
> plot(1:10,
+      wss,
+      type="b",
+      xlab="Number of Clusters",
+      ylab="Within groups sum of squares")
> 
```



Looking at the square error plot while there is a sharp drop from 1 – 2 cluster there is also a significant drop from 6 -7 clusters.

So, the 2 best cluster counts to investigate further is 2 and 7.

Validating cluster with size of 2

```
> set.seed(2048)
> fit.km <- kmeans(data.train, 2)
> fit.km
K-means clustering with 2 clusters of sizes 1957, 2941

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar  chlorides free.sulfur.dioxide total.sulfur.dioxide    density         pH
1     0.1758942       0.04687908   0.2342012      0.8479233  0.3983139           0.5972147            0.7656087  0.9473672 -0.2015099
2    -0.1170435      -0.03119428  -0.1558422     -0.5642251 -0.2650460          -0.3973986           -0.5094513 -0.6303970  0.1340887
    sulphates    alcohol
1  0.05986124 -0.7907243
2 -0.03983286  0.5261637
```

Cluster Means when k=2

fixed.acidity	volatile.acidity	citric.acid	residual.sugar
0.1758942	0.04687908	0.2342012	0.8479233
-0.1170435	-0.03119428	-0.1558422	-0.5642251

chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density
0.3983139	0.5972147	0.7656087	0.9473672
-0.265046	-0.3973986	-0.5094513	-0.630397

pH	sulphates	alcohol
-0.2015099	0.05986124	-0.7907243
0.1340887	-0.03983286	0.5261637

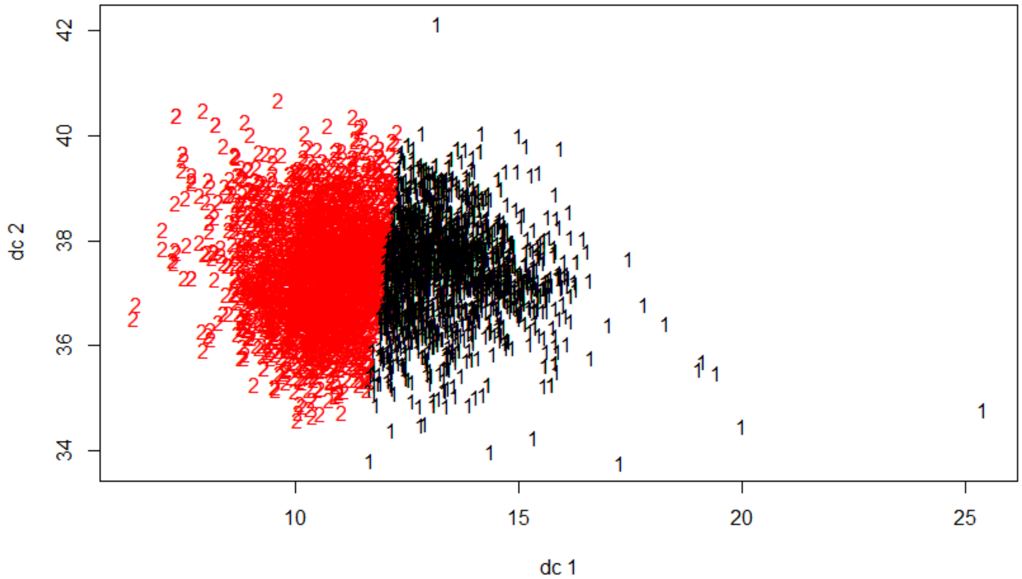
Once they are denormalized:

fixed.acidity	volatile.acidity	citric.acid	residual.sugar
5.62929968	0.127816662	0.38877399	55.88459916
2.5827476	0.048181834	-0.2586981	-36.18747652

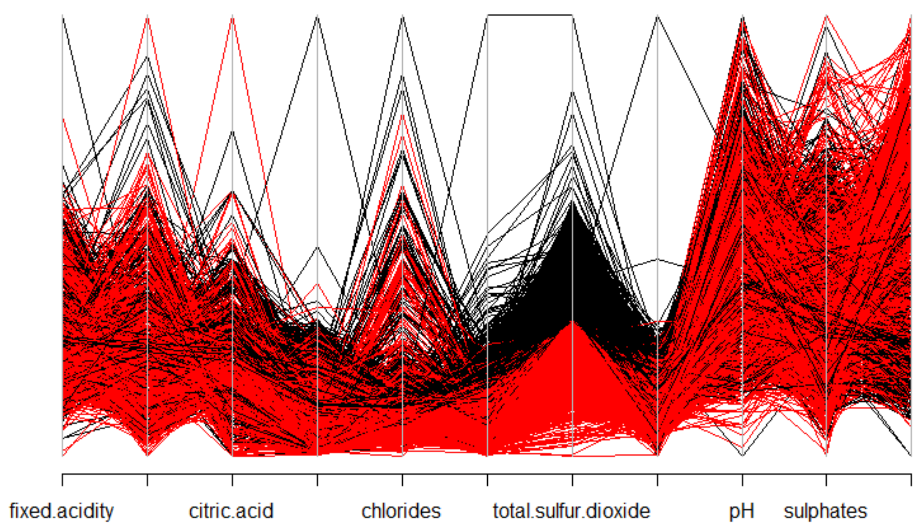
chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density
0.1432318	173.4006189	338.9773497	1.0362499
-0.080321	-112.0533982	-210.5735103	0.9544113

pH	sulphates	alcohol
2.49833911	0.271480666	3.09750934
2.86749757	0.18574374	11.2622149

Discriminant projection plot



Parallel coordinates plot



Confusion matrix

Quality	Cluster 1 Count	Cluster 2 Count
3	12	8
4	53	110
5	848	609
6	860	1338
7	155	725
8	28	147
9	1	4

Adjusted Rand index: 0.026

```
> confuseTable.km <- table(whiteWine$quality, fit.km$cluster)
> confuseTable.km

       1    2
  3   12    8
  4   53  110
  5  848  609
  6  860 1338
  7  155  725
  8   28  147
  9    1    4
> randIndex(confuseTable.km)
       ARI
0.02553198
> 
```

Validating cluster with size of 7

```
> set.seed(4096)
> fit.km <- kmeans(data.train, 7)
> fit.km
K-means clustering with 7 clusters of sizes 454, 103, 827, 527, 1377, 837, 773

Cluster means:
  fixed.acidity volatile.acidity citric.acid residual.sugar   chlorides free.sulfur.dioxide
1    -0.1400975       1.47055214 -0.95486145    -0.06824657  0.20076166         -0.48403759
2    -0.1949314       0.31941980  0.96014734    -0.35979677  5.44159702          0.32240448
3    -0.6292056      -0.55394693 -0.31100445    -0.41791912 -0.08499643         -0.05565529
4    -0.1585792      -0.26625393  0.05912868    -0.64157951 -0.30528648         -0.16221125
5     0.1699279      -0.07960008  0.30861124     1.15781255  0.13666323          0.76764162
6     1.1039547      -0.29885385  0.38217338    -0.46937989 -0.22866076         -0.52612440
7    -0.6085309       0.33331049 -0.23827298    -0.58171074 -0.53977810         -0.38631142
  total.sulfur.dioxide    density         pH   sulphates    alcohol
1          -0.07130350  0.1984290 -0.1027369 -0.33370851 -0.5675310
2           0.12972659  0.1108545 -0.6051197 -0.23090066 -0.7563435
3          -0.07850884 -0.1801148  1.0241516 -0.08988325 -0.1083125
4          -0.28332874 -0.6084174  0.1949485  1.70158555  0.5775274
5           0.90200827  1.1536013 -0.2897311  0.07751036 -0.8835185
6          -0.51482758 -0.3405600 -0.7122475 -0.43938835  0.2865696
7          -0.74761150 -1.2100571  0.1997018 -0.49945591  1.4198268
```

Cluster Means when k=7

fixed.acidity	volatile.acidity	citric.acid	residual.sugar
-0.1400975	1.47055214	-0.95486145	-0.06824657
-0.1949314	0.3194198	0.96014734	-0.35979677
-0.6292056	-0.55394693	-0.31100445	-0.41791912
-0.1585792	-0.26625393	0.05912868	-0.64157951
0.1699279	-0.07960008	0.30861124	1.15781255
1.1039547	-0.29885385	0.38217338	-0.46937989
-0.6085309	0.33331049	-0.23827298	-0.58171074

chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density
0.20076166	-0.48403759	-0.0713035	0.198429
5.44159702	0.32240448	0.12972659	0.1108545
-0.08499643	-0.05565529	-0.07850884	-0.1801148
-0.30528648	-0.16221125	-0.28332874	-0.6084174
0.13666323	0.76764162	0.90200827	1.1536013
-0.22866076	-0.5261244	-0.51482758	-0.34056
-0.5397781	-0.38631142	-0.7476115	-1.2100571

pH	sulphates	alcohol
-0.1027369	-0.33370851	-0.567531
-0.6051197	-0.23090066	-0.7563435
1.0241516	-0.08988325	-0.1083125
0.1949485	1.70158555	0.5775274
-0.2897311	0.07751036	-0.8835185
-0.7122475	-0.43938835	0.2865696
0.1997018	-0.49945591	1.4198268

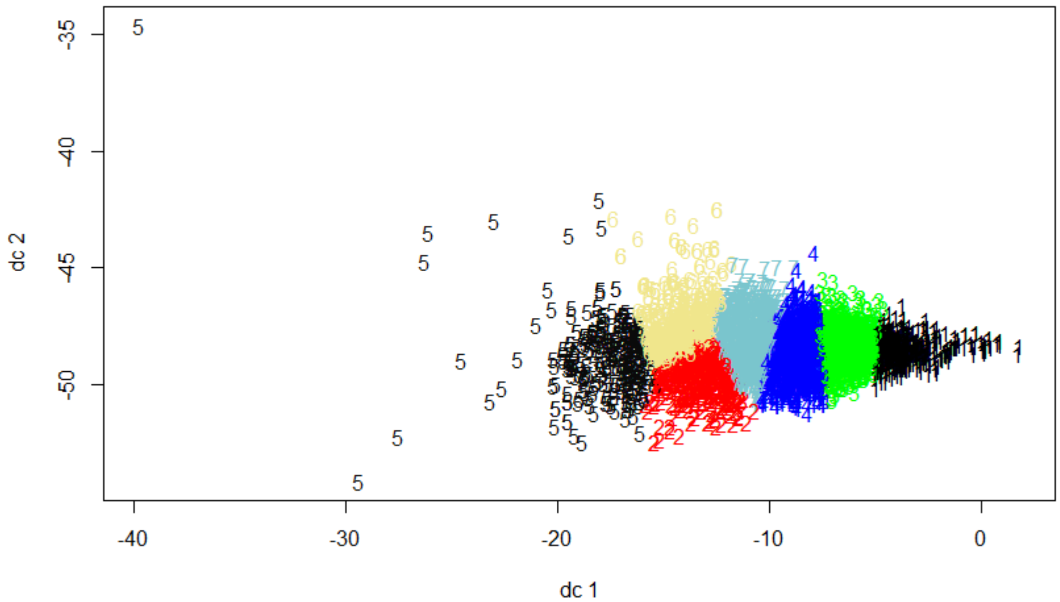
Once they are denormalized:

fixed.acidity	volatile.acidity	citric.acid	residual.sugar
2.342986	1.579963183	-1.58507001	-3.849676364
1.77271344	0.405808196	1.593844584	-22.8587494
-2.74373824	-0.485025869	-0.51626739	-26.64832662
2.15077632	-0.191579009	0.098153609	-41.23098405
5.56725016	-0.001192082	0.512294658	76.08937826
15.28112888	-0.224830927	0.634407811	-30.00356883
-2.52872136	0.4199767	-0.39553315	-37.32754025

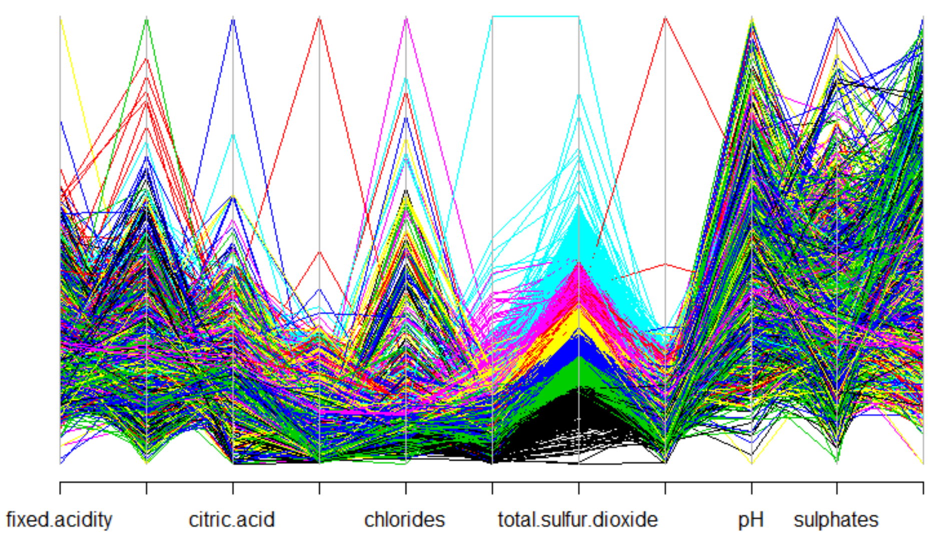
chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density
0.076656679	-136.9187883	-21.7318085	0.99740251
1.842818196	94.53008576	64.91216029	0.99286002
-0.0196438	-13.97306823	-24.83731004	0.97776745
-0.09388154	-44.55462875	-113.1146869	0.95555139
0.055055509	222.3131449	397.7655644	1.0469473
-0.06805868	-148.9977028	-212.890687	0.96944515
-0.17290522	-108.8713775	-313.2205565	0.92434434

pH	sulphates	alcohol
2.60698941	-0.066989319	4.4813078
2.05436833	0.021425432	3.3106703
3.84656676	0.142700405	7.3284625
2.93444335	1.683363573	11.5806699
2.40129579	0.28665891	2.5221853
1.93652775	-0.157873981	9.77673152
2.93967198	-0.209532083	16.8029262

Discriminant projection plot



Parallel coordinates plot



Confusion matrix

Quality	Cluster 1 Count	Cluster 2 Count	Cluster 3 Count	Cluster 4 Count	Cluster 5 Count	Cluster 6 Count	Cluster 7 Count
3	2	1	2	0	8	4	3
4	53	3	17	8	26	42	14
5	257	49	191	73	590	248	49
6	126	46	446	247	616	384	333
7	12	2	147	164	116	137	302
8	4	2	24	35	21	21	68
9	0	0	0	0	0	1	4

Adjusted Rand index: 0.043

```
> confuseTable.km <- table(whiteWine$quality, fit.km$cluster)
> confuseTable.km

      1   2   3   4   5   6   7
  3   2   1   2   0   8   4   3
  4  53   3  17   8  26  42  14
  5 257  49 191  73 590 248  49
  6 126  46 446 247 616 384 333
  7  12   2 147 164 116 137 302
  8   4   2  24  35  21  21  68
  9   0   0   0   0   0   1   4
> randIndex(confuseTable.km)
       ARI
0.04325919
>
```

Conclusion: 7 Clusters is the optimal.

While the ARI for both k=2 and k=7 is very low, its slightly better for k=7. Therefore, for the given dataset 7 is the optimal number of clusters.

Code:

```
library(NbClust)
library(fpc)
library(MASS)
library(flexclust)

whiteWine<-read.csv("D:/Projects/WhiteWine.csv")
names(whiteWine)
str(whiteWine)
summary(whiteWine)
data.train <- scale(whiteWine[1:11])
summary(data.train)

set.seed(1024)
data.nc <- NbClust(data=data.train, distance="euclidean", min.nc=2, max.nc=10,
method="kmeans", index="all")
table(data.nc$Best.n[1,])
barplot(table(data.nc$Best.n[1,]),
        xlab="Numer of Clusters",
        ylab="Number of Criteria")
wss <- 0
for (i in 1:10){
  wss[i] <-
    sum(kmeans(data.train, centers=i)$withinss)
}
plot(1:10,
     wss,
     type="b",
     xlab="Number of Clusters",
     ylab="Within groups sum of squares")

set.seed(2048)
fit.km1 <- kmeans(data.train, 2)
fit.km1$centers
plotcluster(data.train, fit.km1$cluster)
parcoord(data.train, fit.km$cluster)
confuseTable.km1 <- table(whiteWine$quality, fit.km1$cluster)
confuseTable.km1
randIndex(confuseTable.km1)

set.seed(4096)
fit.km2 <- kmeans(data.train, 7)
fit.km2
plotcluster(data.train, fit.km2$cluster)
parcoord(data.train, fit.km2$cluster)
confuseTable.km2 <- table(whiteWine$quality, fit.km2$cluster)
confuseTable.km2
randIndex(confuseTable.km2)
```

2nd Objective (MLP)

Data pre-processing

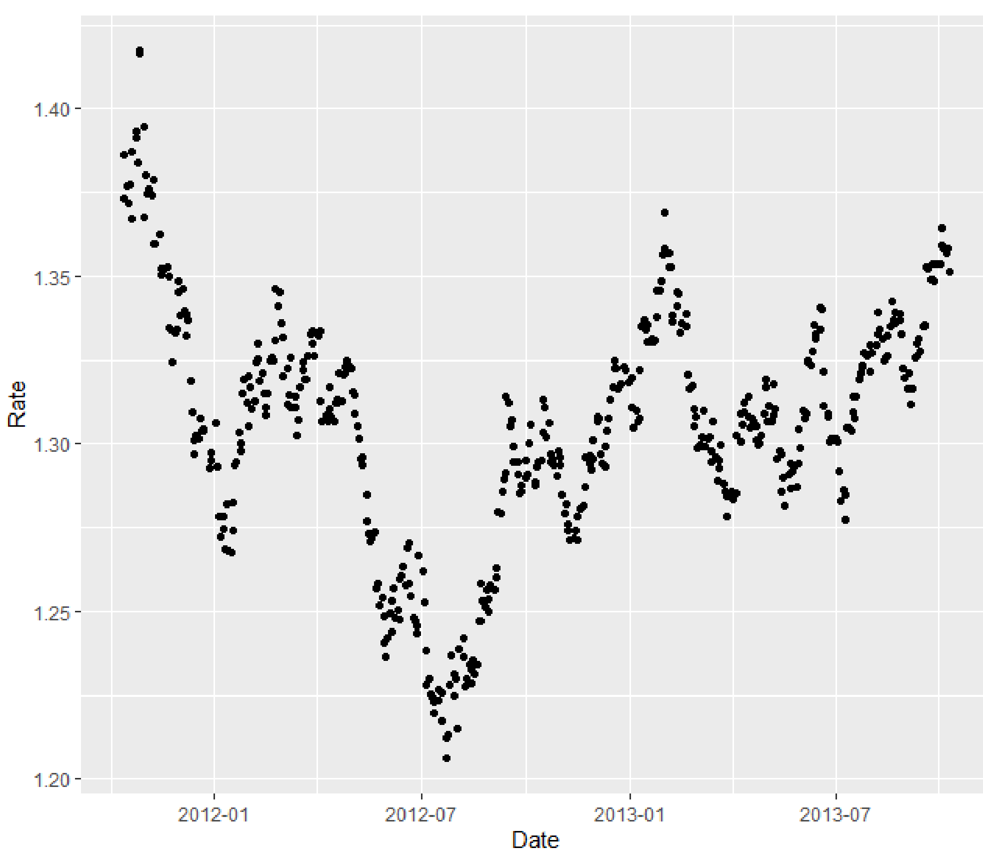
- First, I've converted the xlxs format file to csv since it's much easier to work with in R Studio.
- Then I needed to know the types of variables (I've renamed the column names to make it easier to work with)

```
> #Load CSV from Disk
> exchangeRates<-read.csv("D:/Projects/ExchangeUSD.csv")
>
> str(exchangeRates)
'data.frame':	500 obs. of  3 variables:
 $ Date: Factor w/ 500 levels "1/10/2012","1/10/2013",..: 49 50 53 55 57 61 62 65 67 69 ...
 $ Day : Factor w/ 5 levels "Fri","Mon","Thu",..: 3 1 2 4 5 3 1 2 4 5 ...
 $ Rate: num  1.37 1.39 1.38 1.37 1.38 ...
```

- Next the Date column was converted to `DateTime` data type

```
> #Update the Date column to data type date
> exchangeRates$Date <- as.Date(exchangeRates$Date, format = "%m/%d/%Y")
> str(exchangeRates)
'data.frame':	500 obs. of  3 variables:
 $ Date: Date, format: "2011-10-13" "2011-10-14" "2011-10-17" "2011-10-18" ...
 $ Day : Factor w/ 5 levels "Fri","Mon","Thu",..: 3 1 2 4 5 3 1 2 4 5 ...
 $ Rate: num  1.37 1.39 1.38 1.37 1.38 ...
```

- A plot was drawn to visualize the data as Rate vs Date



- As the variation of rate is relatively small, I decided not to normalize it. Also, I decided to not include the Day column to train the model.

```
ds ← exchangeRates$Date
y ← exchangeRates$Rate
df ← data.frame(ds, y)
qplot(ds, y, data=df)
```

Training the model

- A model was trained using Facebook's prophet package.

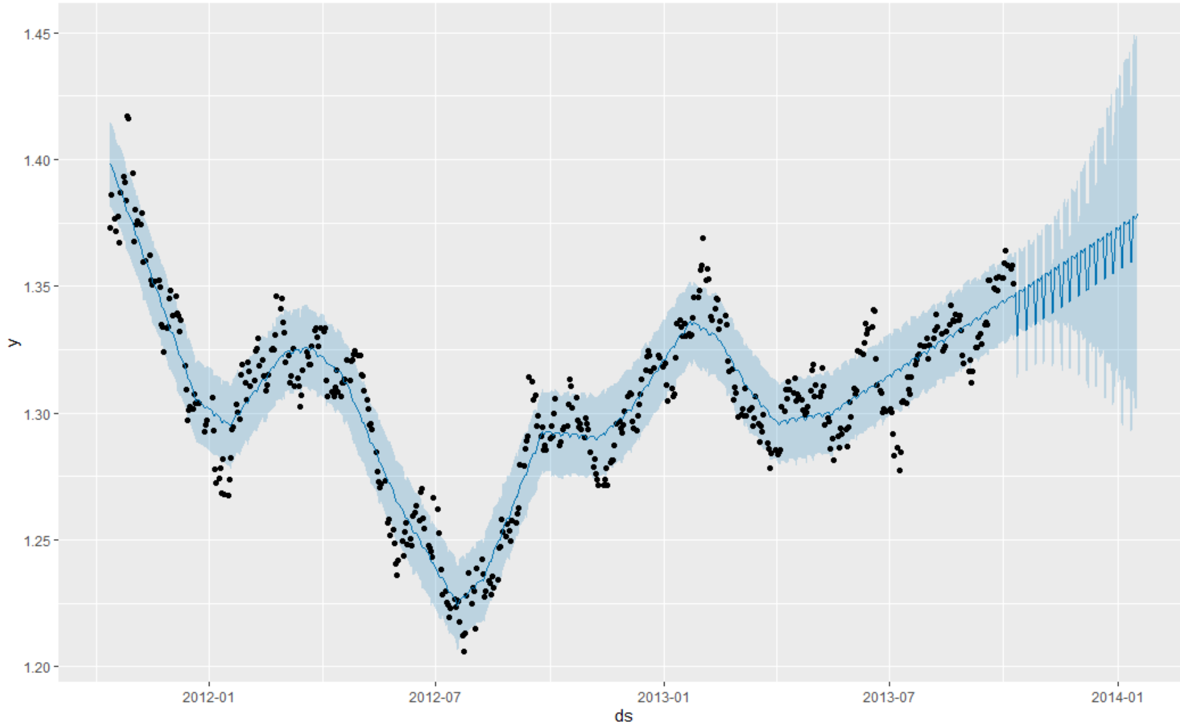
```
> # Creating a model with Facebook's prophet package
> m ← prophet(df)
Disabling yearly seasonality. Run prophet with yearly.seasonality=TRUE to override this.
Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.
```

Validating the model

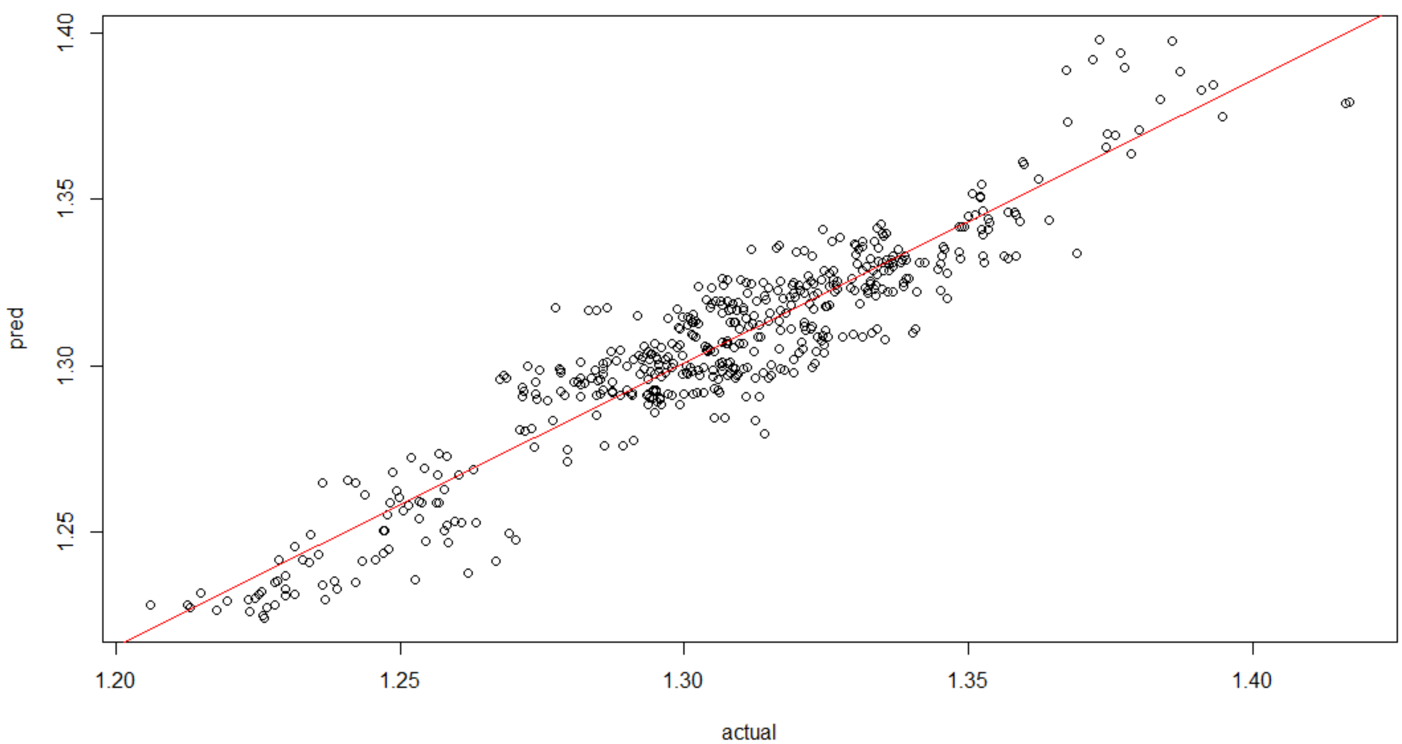
- To validate the model 100 predictions was made

```
> # Make 100 days predictions using the model
> future ← make_future_dataframe(m, periods = 100)
> forecast ← predict(m, future)
```

- Graph was plotted to show the confidence levels of the model.
  - Black dots are actual data points
  - Blue line is the predictions
  - Light blue area is the uncertainty level



- Furthermore, actual vs predicted graph was plotted, red line denotes the regressed diagonal line.



- The model R^2 was calculated as follows, ***R2 of the model is 0.8788***

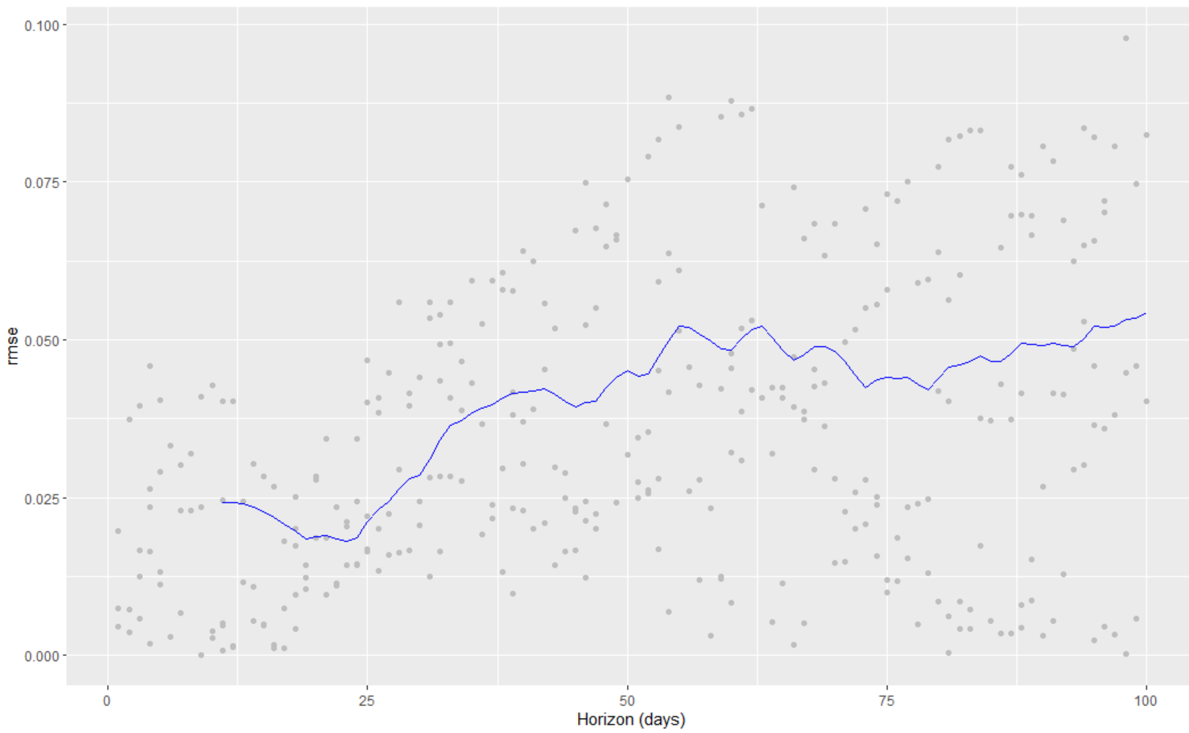
```
> summary(lm(pred~actual))

Call:
lm(formula = pred ~ actual)

Residuals:
      Min        1Q    Median        3Q       Max 
-0.033204 -0.007531  0.000081  0.007917  0.035974 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  0.19463    0.01847   10.54   <2e-16 ***
actual       0.85089    0.01414   60.16   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.01136 on 498 degrees of freedom
Multiple R-squared:  0.879,     Adjusted R-squared:  0.8788 
F-statistic:  3619 on 1 and 498 DF,  p-value: < 2.2e-16
```

- RMSE, MAE and MAPE of the model is calculated by:

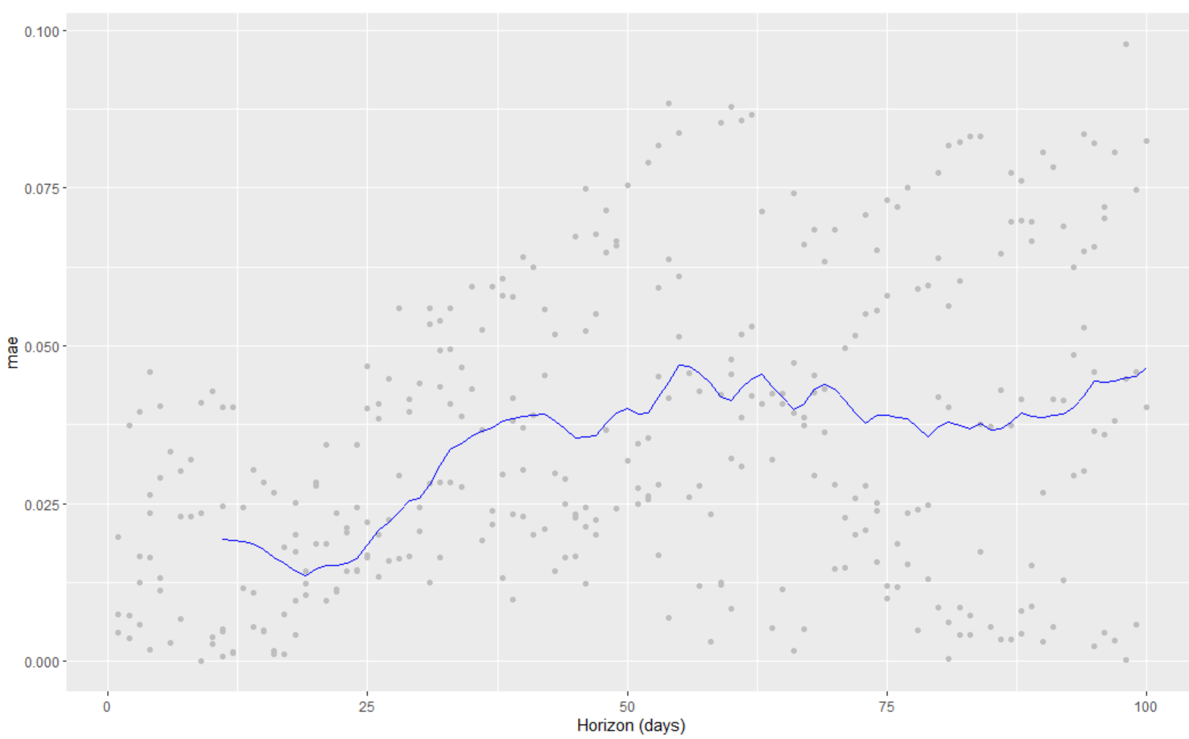
```
> x ← cross_validation(m, horizon = 100, initial = 400, units = 'days')
Making 5 forecasts with cutoffs between 2012-12-13 and 2013-07-01
> performance_metrics(x, rolling_window = 0.1)
```

```
> plot_cross_validation_metric(x,
+                              metric = 'mae',
+                              rolling_window = 0.1)
> plot_cross_validation_metric(x,
+                              metric = 'rmse',
+                              rolling_window = 0.1)
> plot_cross_validation_metric(x,
+                              metric = 'mape',
+                              rolling_window = 0.1)
>
```

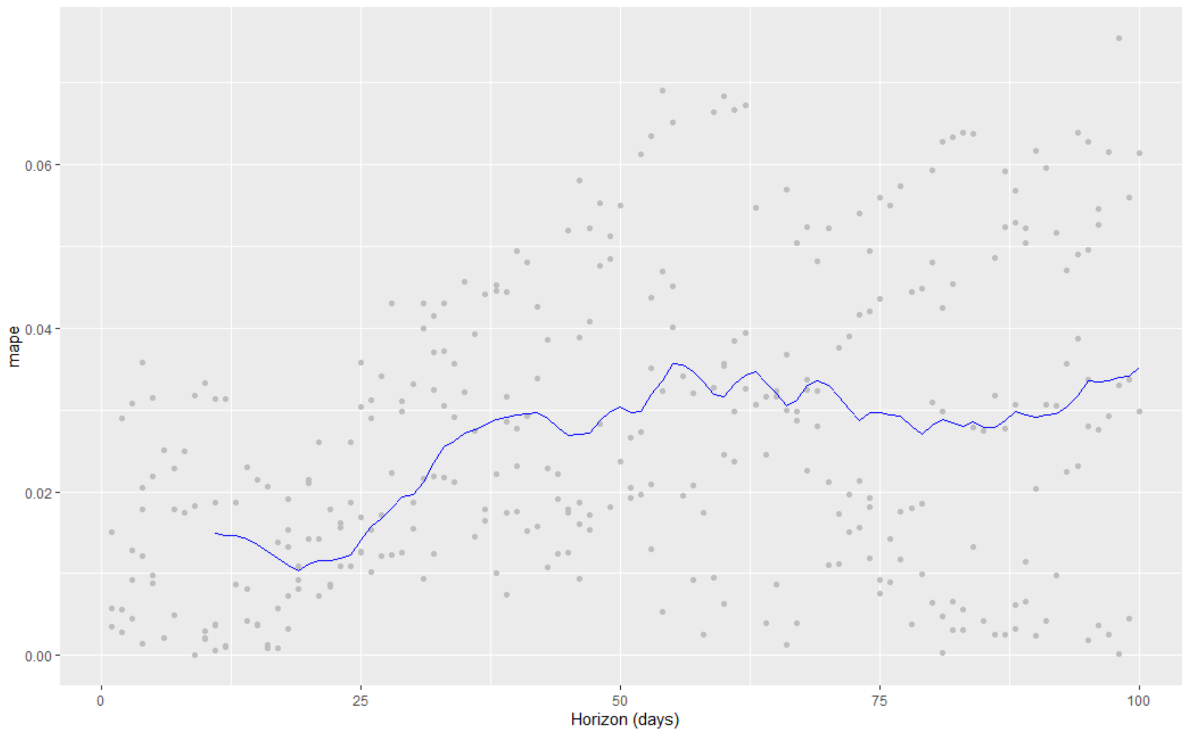
RMSE: 0.039136758



MAE: 0.034326899



MAPE: 0. 026071806



Code:

```
library(ggplot2)
library(prophet)

#Load CSV from Disk
exchangeRates<-read.csv("D:/Projects/ExchangeUSD.csv")

str(exchangeRates)

#Update the Date column to data type date
exchangeRates$Date <- as.Date(exchangeRates$Date, format = "%m/%d/%Y")
str(exchangeRates)
qplot(Date, Rate, data=exchangeRates)


ds <- exchangeRates$Date
y <- exchangeRates$Rate
df <- data.frame(ds, y)
qplot(ds, y, data=df)

# Creating a model with Facebook's prophet package
m <- prophet(df)

# Make 100 days predictions using the model
future <- make_future_dataframe(m, periods = 100)

forecast <- predict(m, future)

# Plot forecast
plot(m, forecast)
prophet_plot_components(m, forecast)


# Model performance
pred <- forecast$yhat[1:500]
actual <- m$history$y
plot(actual, pred)

#The ideal predictions lines
abline(lm(pred~actual), col = 'red')
summary(lm(pred~actual))

x <- cross_validation(m, horizon = 100, initial = 400, units = 'days')
performance_metrics(x, rolling_window = 0)

plot_cross_validation_metric(x,
                             metric = 'mae',
                             rolling_window = 0.1)

plot_cross_validation_metric(x,
                             metric = 'rmse',
                             rolling_window = 0.1)
```

```
plot_cross_validation_metric(x,
                             metric = 'mape',
                             rolling_window = 0.1)
```

3rd Objective (SVR)

Data pre-processing

Same process as for 2nd object was followed to pre-process the data.

Creating the SVM

- First a new data frame was created with only the Date and Rate columns.

```
> Dates <- exchangeRates$Date
> Rates <- exchangeRates$Rate
> df <- data.frame(Dates, Rates)
>
```

- The SVN model was created taking the Rates as a function of Dates

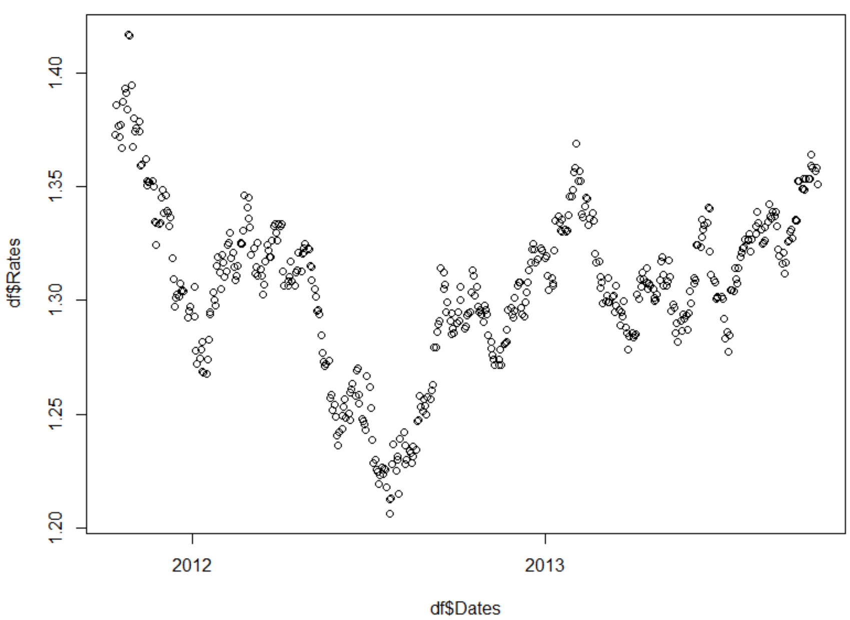
```
> model <- svm(Rates~Dates, df)
>
```

Validating the Model

- Prediction was made using the model to be used in the validation process

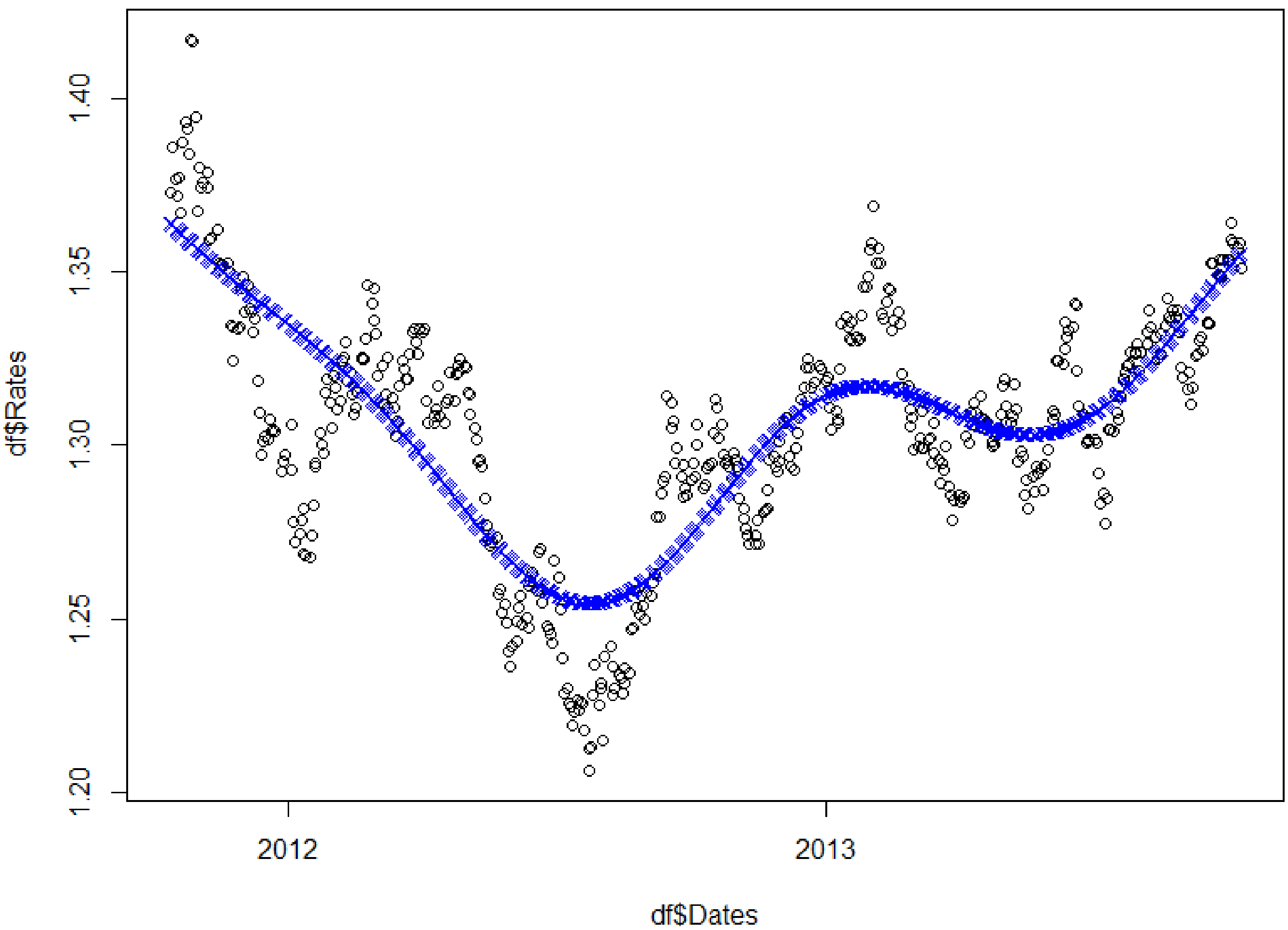
```
> prediction <- predict(model, data = df$Dates)
>
```

- A plot was drawn with the actual data Date vs Rate



- On the same plot the predictions from the model was plotted. (in blue)

```
> plot(
+   y = df$Rates,
+   x = df$Dates)
> points(
+   y = prediction,
+   x = df$Dates,
+   col = 'blue',
+   pch = 4)
```



- Root Mean Squared Error and the model summery was check as follows

```
> rmse ← function(error)
+ {
+   sqrt(mean(error^2))
+ }
>
> rmse(model$residuals)
[1] 0.02094349
> summary(model)

Call:
svm(formula = Rates ~ Dates, data = df)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  1
    epsilon:  0.1


Number of Support Vectors:  431
```

- **RMSE for this model is 0.021**

Improving the model

This looking at this I decided to the model cloud be improved by tuning the model and used following parameters to tune the model.

```
svmTune ← tune(svm, Rates~Dates, data = df, ranges = list(epsilon = seq(0,1,0.01), cost = 2^(2:9)))
```

This process completed with the following results:

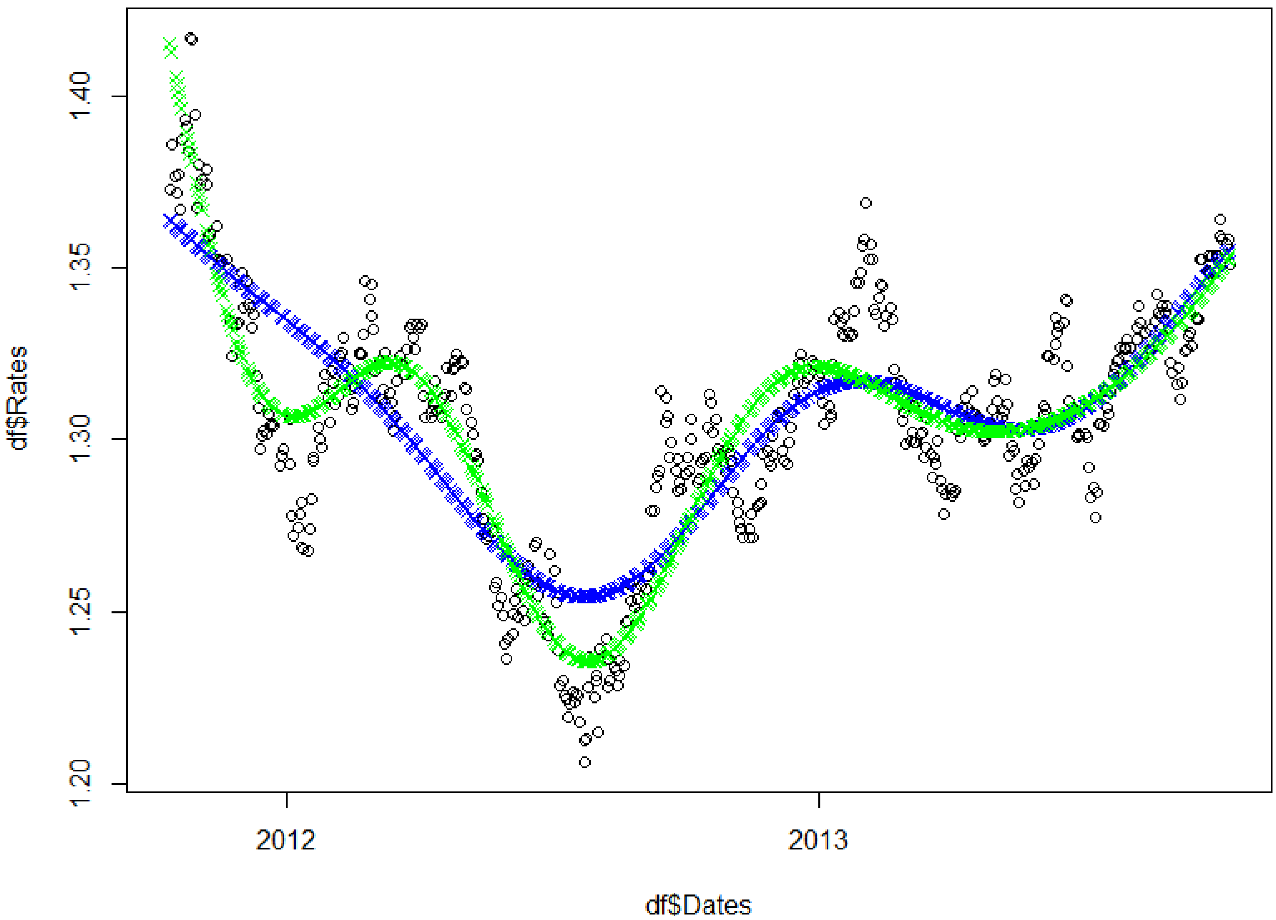
```
> print(svm_tune)

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 epsilon cost
    0.28  512

- best performance: 0.0002845068
```

From the svm_tune I got the updated model and check again for the root mean squared error.

```
> best_model <- svm_tune$best.model
> best_prediction <- predict(best_model, data = df$Dates)
> rmse(best_model$residuals)
[1] 0.0164688
```

With the tuning the RMSE is lowered to 0.016.

The tuned model was used to do anther prediction and a plot was drawn. (Green – Tuned model, Blue – initial model, Black – actual values)



Code:

```
library('e1071')

exchangeRates<-read.csv("D:/Projects/ExchangeUSD.csv")
str(exchangeRates)
exchangeRates$Date <- as.Date(exchangeRates$Date, format = "%m/%d/%Y")

Dates <- exchangeRates$Date
Rates <- exchangeRates$Rate
df <- data.frame(Dates, Rates)

model <- svm(Rates~Dates, df)

prediction <- predict(model, data = df$Dates)

plot(
  y = df$Rates,
  x = df$Dates)

points(
  y = prediction,
  x = df$Dates,
  col = 'blue',
  pch = 4)

rmse <- function(error)
{
  sqrt(mean(error^2))
}

rmse(model$residuals)
summary(model)

svm_tune <- tune(svm, Rates~Dates, data = df, ranges = list(epsilon =
seq(0,1,0.01), cost = 2^(2:9)))
print(svm_tune)


best_model <- svm_tune$best.model
best_prediction <- predict(best_model, data = df$Dates)
rmse(best_model$residuals)

points(
  y = best_prediction,
  x = df$Dates,
  col = 'green',
  pch = 4)

plot(svm_tune)
```