Reg no :j20200776
University ID : w1831982

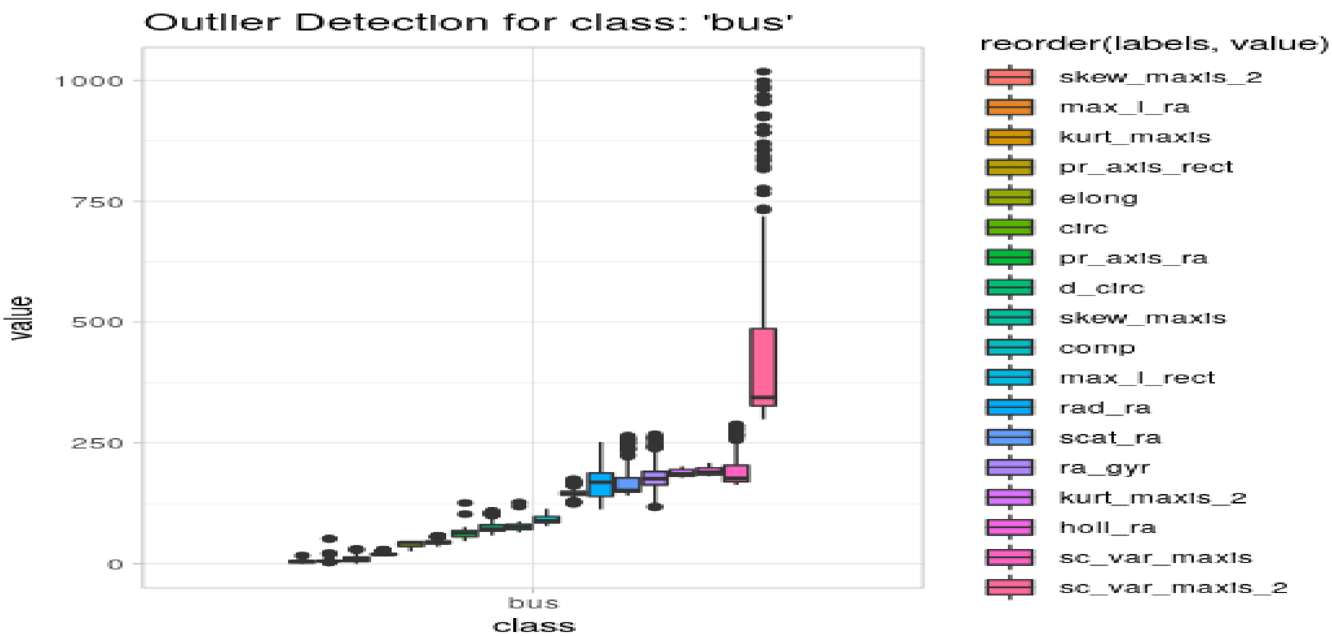
Answer for the 1st question (partitioning clustering)

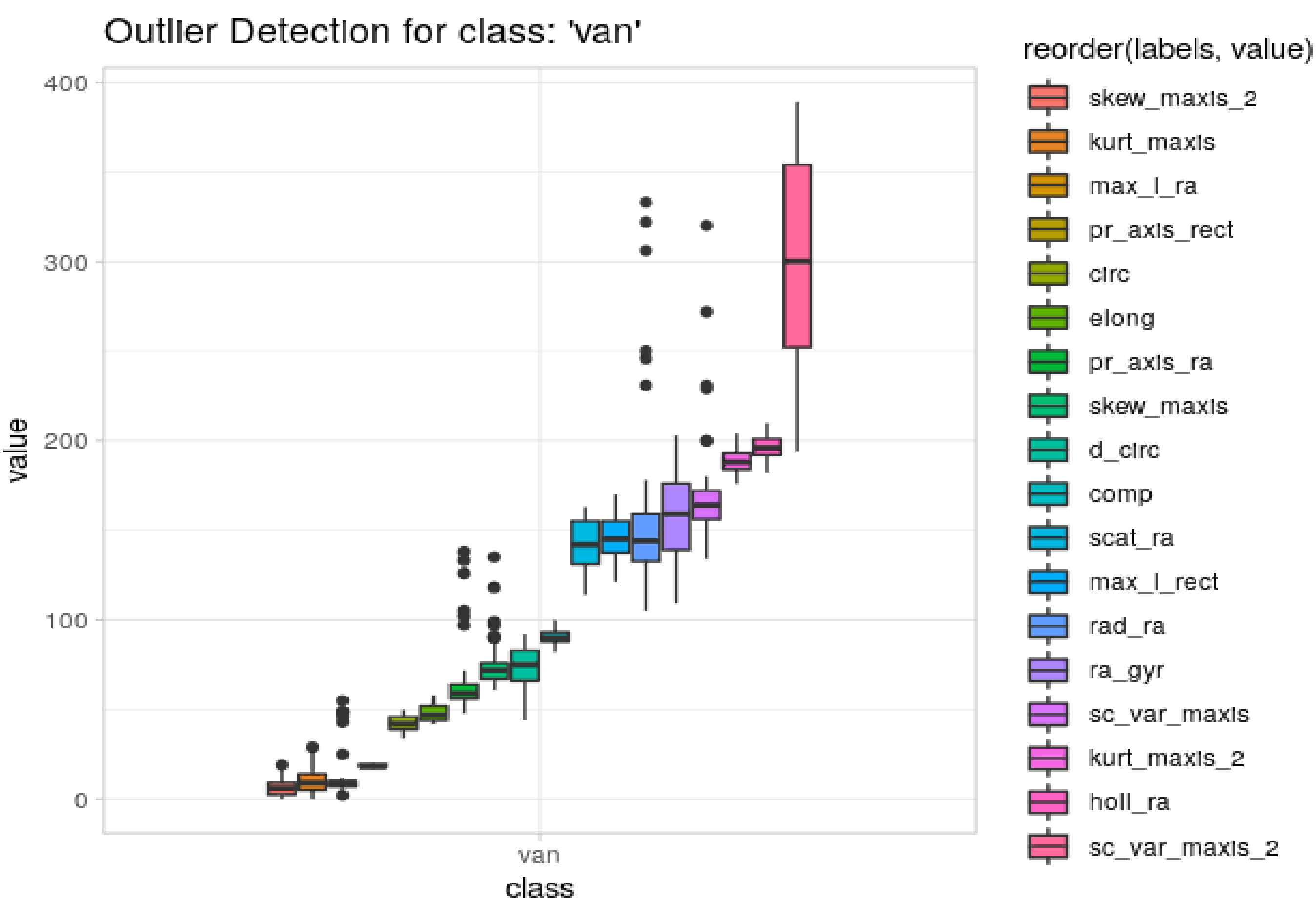
- Code is available in the folder ,"**coursework1-Q1**"

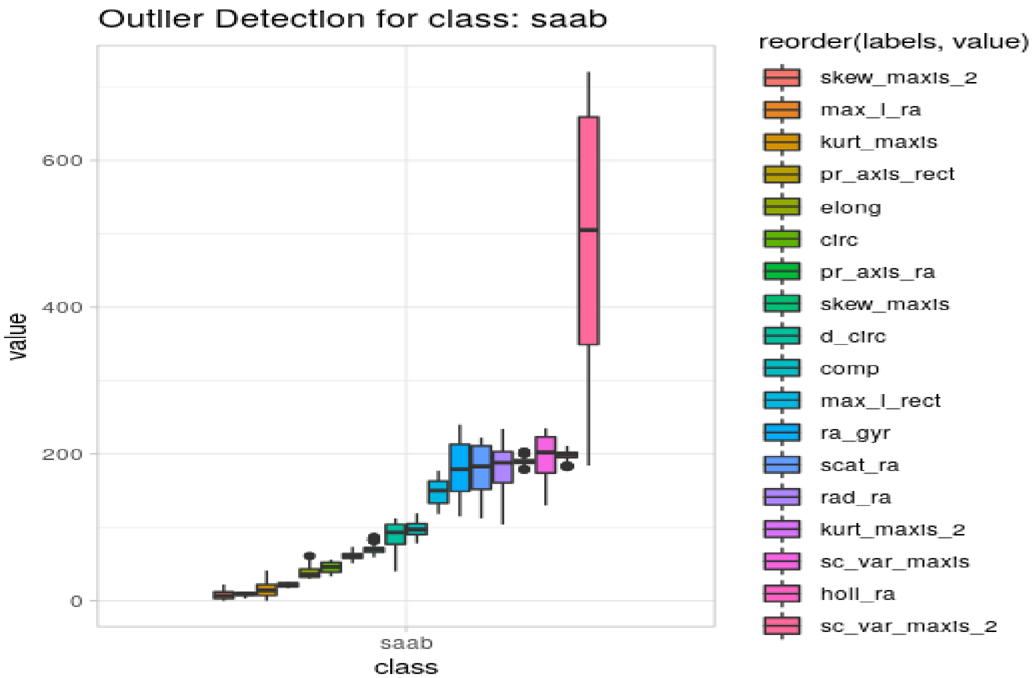
Automated and Manual process :

- **Automated process** :

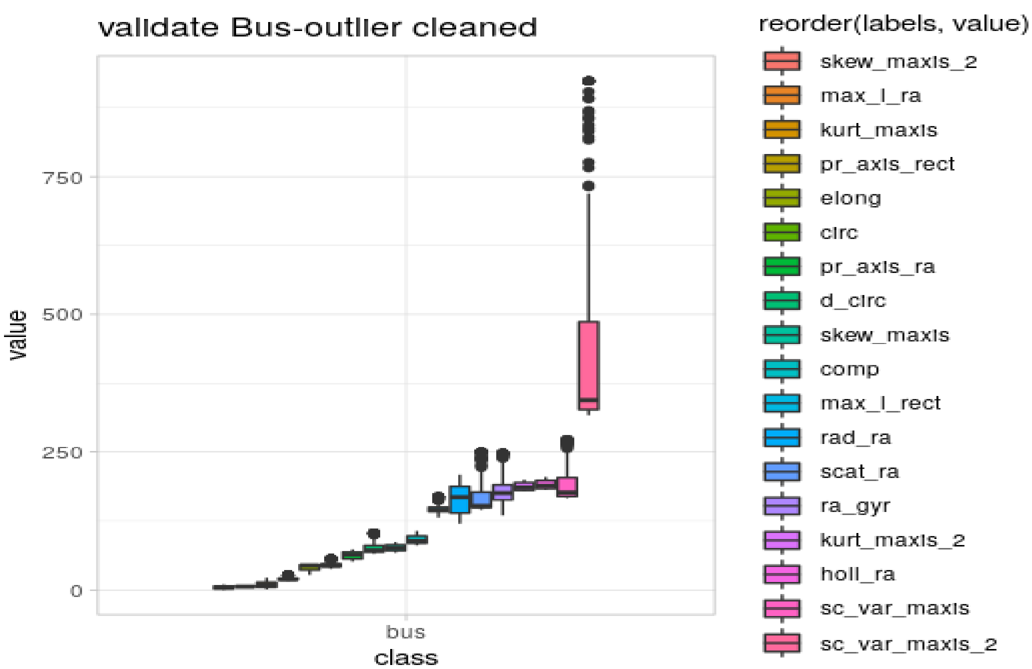
Steps:
1.Reading csv file
2.Cleaning the name of the given data fields using janitor package
Eg: Renaming the duplicates with underscore
3.Mutating
Setting the 'class' variables with factor variables and then sort the dataset by median values.
4.Checking the outlier detection (refer to the folder named "Images" for reference).


Outlier Detection for class: 'bus'
value
1000
750
500
250
0
bus
class
reorder(labels, value)
skew_maxis_2
max_l_ra
kurt_maxis
pr_axis_rect
elong
circ
pr_axis_ra
d_circ
skew_maxis
comp
max_l_rect
rad_ra
scat_ra
ra_gyr
kurt_maxis_2
holl_ra
sc_var_maxis
sc_var_maxis_2


Outlier Detection for class: 'van'
value
400
300
200
100
0
van
class
reorder(labels, value)
skew_maxis_2
kurt_maxis
max_l_ra
pr_axis_rect
circ
elong
pr_axis_ra
skew_maxis
d_circ
comp
scat_ra
max_l_rect
rad_ra
ra_gyr
sc_var_maxis
kurt_maxis_2
holl_ra
sc_var_maxis_2



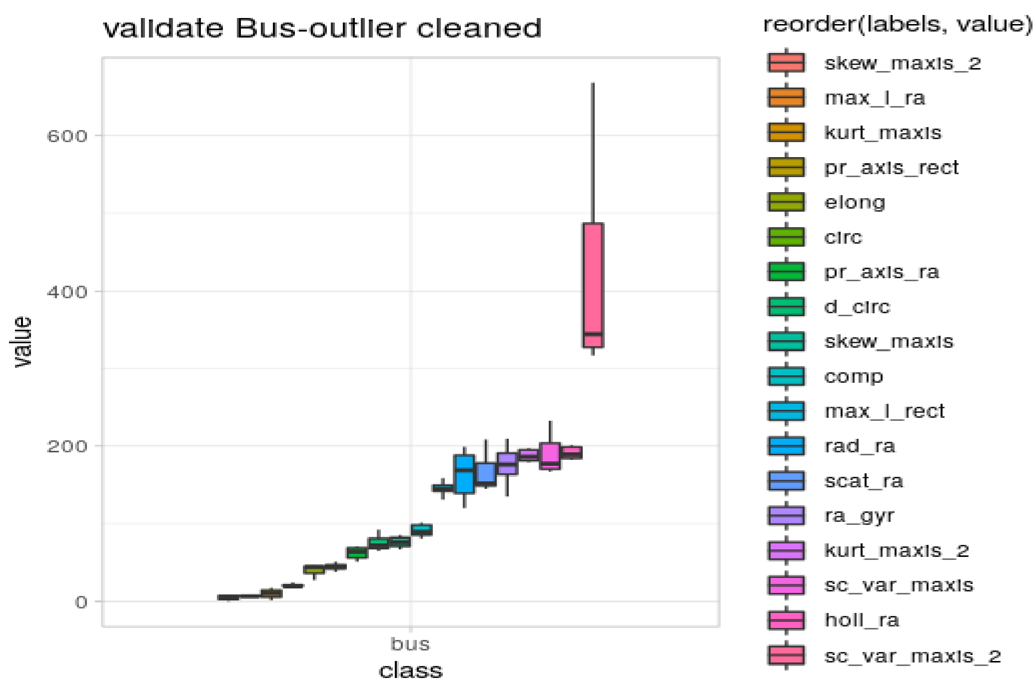
5.When we use mutate(across(2:19, ~squish(.x, quantile(.x, c(.05, .95)))))
Still the bus data has outlier ,so needs to adjust it.(refer to the folder named "0.5_0.95_transformed_images" for reference)



5.If the given code is applied to the outlier, will make the bus outlier dataset clean.

```
mutate(across(2:19, ~squish(.x, quantile(.x, c(.05, .85)))))
```

(refer to the folder “0.5_0.85_transformed_images” for reference).



6.Combine all data as below.

```
# Remove the sample name and the class name. Both of these will be remove
so that only n
#numerical data is left for the algorithm.
vehicles_data_points = combined %>%
  select(-samples, -class)
```

7.Scale the data which excluded sample and class columns.

Scope:different attributes are in different ranges, which will lead to misleading result of our prediction

```
# Now that we have the "vehicles_data_points" dataset, scaling is performed
```

```
vehicles_scaled = vehicles_data_points %>%
  mutate(across(everything(), scale))
```

8.Set seeds to reliable outputs (same outputs for others)

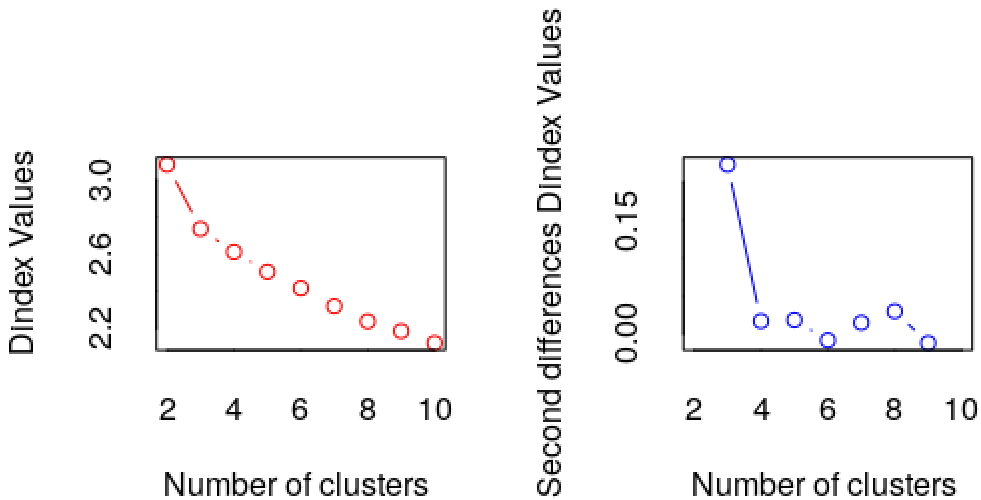
```
set.seed(123)
```

9.Perform cluster automatic packages using two different numbers as following steps.

10.Using “euclidean” algorithm

```
# Use Euclidean for distance
cluster_euclidean = NbClust(vehicles_scaled,distance="euclidean",
min.nc=2,max.nc=10,method="kmeans",index="all")
```

```
* Among all indices:
* 11 proposed 2 as the best number of clusters
* 9 proposed 3 as the best number of clusters
* 1 proposed 6 as the best number of clusters
* 1 proposed 7 as the best number of clusters
* 1 proposed 8 as the best number of clusters
* 1 proposed 10 as the best number of clusters

                   ***** Conclusion *****

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



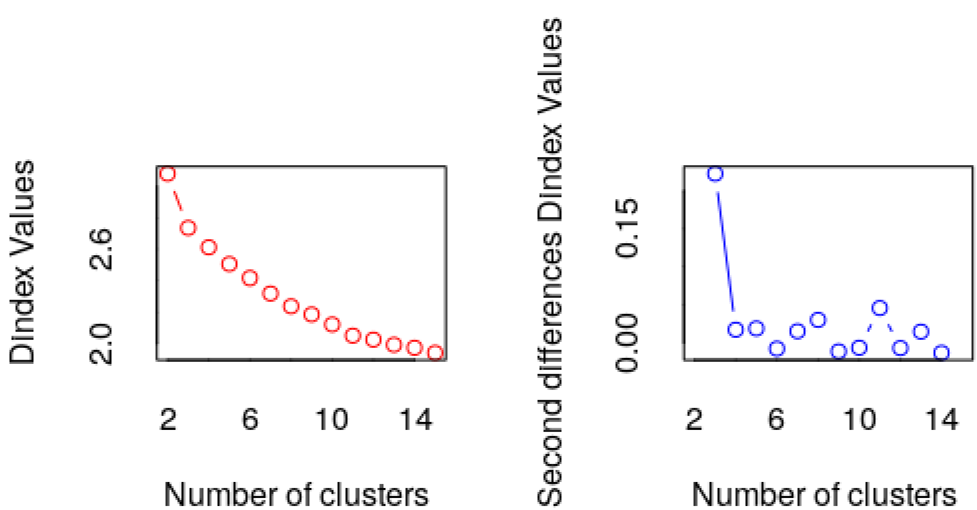

```
cluster_manhattan_2 = NbClust(vehicles_scaled,distance="manhattan",
min.nc=2,max.nc=10,method="kmeans",index="all")
```

```
* Among all indices:
* 11 proposed 2 as the best number of clusters
* 9 proposed 3 as the best number of clusters
* 1 proposed 7 as the best number of clusters
* 1 proposed 8 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
```

11.Using “manhattan” algorithm.

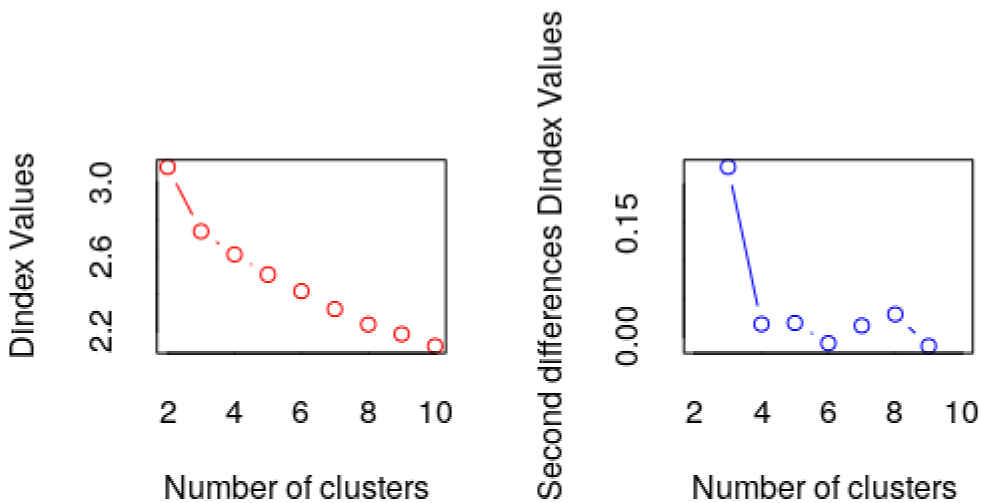
```
# Use manhattan for distance
cluster_manhattan = NbClust(vehicles_scaled,distance="manhattan",
min.nc=2,max.nc=15,method="kmeans",index="all")
```

```
* Among all indices:
* 10 proposed 2 as the best number of clusters
* 9 proposed 3 as the best number of clusters
* 1 proposed 7 as the best number of clusters
* 1 proposed 8 as the best number of clusters
* 1 proposed 11 as the best number of clusters
* 1 proposed 14 as the best number of clusters
* 1 proposed 15 as the best number of clusters

                   ***** Conclusion *****

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




```
cluster_manhattan_2 = NbClust(vehicles_scaled,distance="manhattan",
min.nc=2,max.nc=10,method="kmeans",index="all")
```

```
* Among all indices:
* 11 proposed 2 as the best number of clusters
* 9 proposed 3 as the best number of clusters
* 1 proposed 7 as the best number of clusters
* 1 proposed 8 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
```

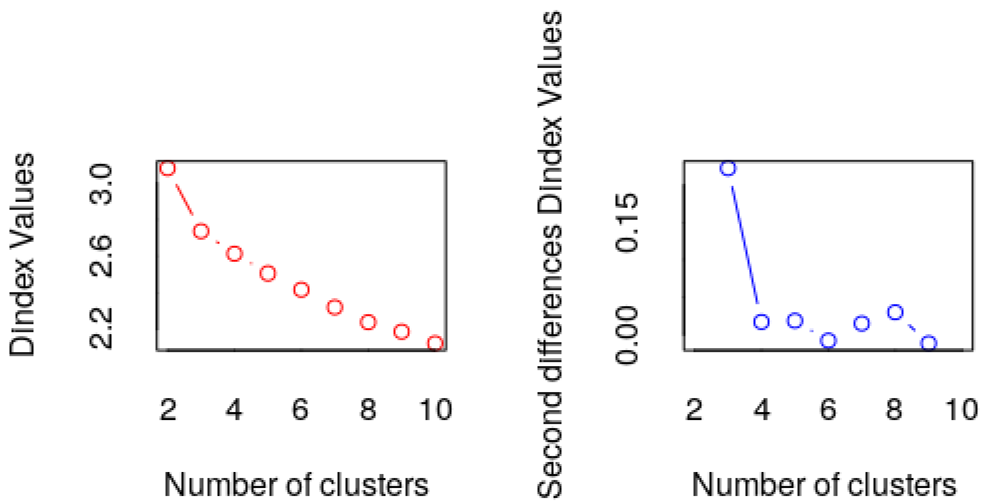


12. Maximum algorithm:

```
## maximum
clusterNo=NbClust(vehicles_scaled,distance="maximum",
min.nc=2,max.nc=15,method="kmeans",index="all")

clusterNo_2=NbClust(vehicles_scaled,distance="maximum",
min.nc=2,max.nc=10,method="kmeans",index="all")
```

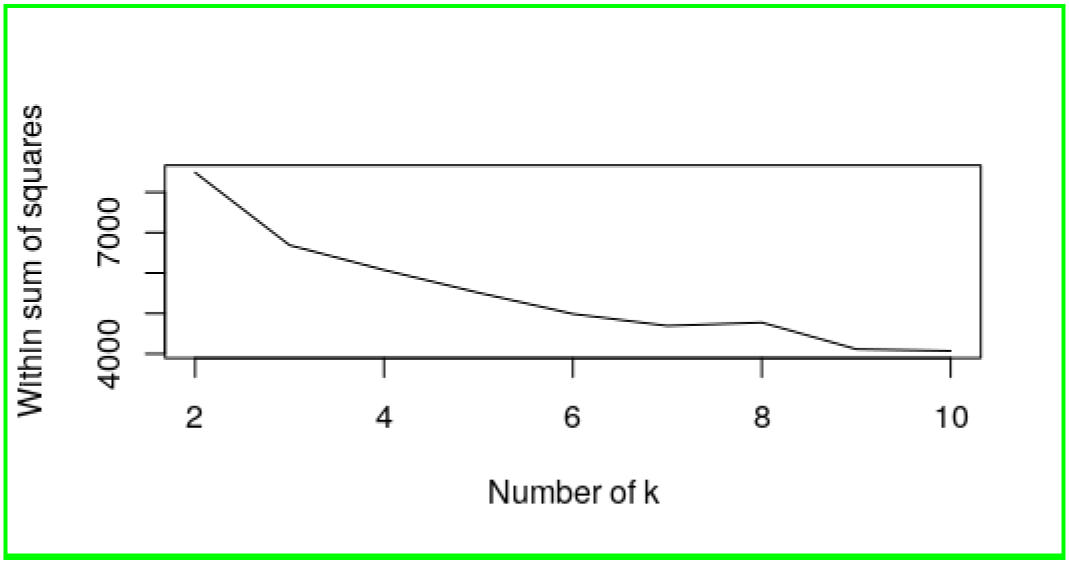
```
*******************************************************************
* Among all indices:
* 9 proposed 2 as the best number of clusters
```

```
* 10 proposed 3 as the best number of clusters
* 1 proposed 7 as the best number of clusters
* 1 proposed 8 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  3
```



13 .Using the elbow method ,refer to the “elbow” folder.

```
##Elbow method
k = 2:10

WSS = sapply(k, function(k) {kmeans(vehicles_scaled, centers=k)$tot.withinss})
plot(k, WSS, type="l", xlab= "Number of k", ylab="Within sum of squares")
```



Output Elbow method suggest to go with k=2 or k=3

- **Manual Process** :

1.Using K =2

```
kc <- kmeans(vehicles_scaled,2)
```

Output:

```
Within cluster sum of squares by cluster:
[1] 5865.445 2619.676
 (between_SS / total_SS =  44.2 %)
```

```
Here first cluster, within cluster distance  -> 5865.445
Second cluster,within cluster distance  -> 2619.676
Between clusters → 44.2 %
```

The issue in the output here is, the first cluster is too high within cluster distance,which is not recommended for the best K clustering.

2.Using K =3

```
kc <- kmeans(vehicles_scaled,3)
```

Output:

```
Within cluster sum of squares by cluster:
[1] 1729.268 2233.286 2724.941
 (between_SS / total_SS =  56.0 %
```

```
Here first cluster  -> 1729.26
Second cluster -> 2233.286
Third cluster -> 2724.941

Between clusters → 56.0 %
```

Between clusters, a high percentage is most preferred in this scenario.

- How to choose the best clusters:

Within the same cluster the distance should be low to each other, but between different clusters distance should be high.

- Final Decision:

As the final decision of the studies,the clusters which applied K=3 is the best, because of the following reasons.

1.This gives 3 clusters that are most suitable,as between the cluster distance is higher and within the clusters the distance is lower than K=2
2.Maximum Algorithm also suggested K=3 clusters ,even others suggested k=2 there is a minor difference between k=2 and k=3 clusters.
3.When analysing the given business scenario it is most understandable that both cars are under one cluster and the other two types(van and bus) are in two different clusters .So totally three clusters are used for data prediction.

- Validating the consistency of the results against the 19 th column and providing relevant discussion.

```
K =2

y        1   2
  van  199   0
  saab 100 117
  bus  161  57
  opel  91 121
```

Here saab and opel can go under the second cluster, at the sametime van and bus should go under the first cluster.

```
k= 3


  y       1   2   3
  van    82   0 117
  saab   39 107  71
  bus    85  50  83
  opel   36 113  63
```

As per above chart saab and opel can go under the second cluster,at the sametime van is under the third cluster and the bus should go under the first cluster.

- Finding the mean of each attribute for the winner cluster(K=3)

```
K-means clustering with 3 clusters of sizes 270, 242, 334

Cluster means:
        comp       circ      d_circ       rad_ra pr_axis_ra   max_l_ra    scat_ra
elong pr_axis_rect max_l_rect
1  1.1512967  1.1807597  1.1994459  1.05054462  0.1930330  0.6866379  1.3059043
-1.2548220    1.3123585  1.1029343
2 -0.9600225 -0.5704828 -0.9150603 -1.15311406 -0.7460693 -0.5561250 -0.8110344
0.8586586   -0.7783015 -0.5333989
3 -0.2351037 -0.5411625 -0.3066042 -0.01375283  0.3845205 -0.1521257 -0.4680354
0.3922352   -0.4969696 -0.5051189
  sc_var_maxis sc_var_maxis_2     ra_gyr    skew_maxis skew_maxis_2  kurt_maxis
kurt_maxis_2    holl_ra
1    1.2669835      1.3281356  1.0857072 -0.0007954144   0.17350992  0.25092754
0.06919444  0.2327185
2   -0.8246845     -0.8229496 -0.4195588  0.8901877226  -0.08478767 -0.24025427
-1.12422792 -1.1579107
3   -0.4266823     -0.4773737 -0.5736758 -0.6443433143  -0.07882952 -0.02876917
0.75862472  0.6508395
```

```
Within cluster sum of squares by cluster:
[1] 2233.286 1729.268 2724.941
 (between_SS / total_SS =  56.0 %)
```

Web urls:
https://github.com/azeemj/Top-10-Machine-Learning-Methods-With-R
https://www.sharpsightlabs.com/blog/mutate-in-r/
https://bookdown.org/aschmi11/RESMHandbook/data-preparation-and-cleaning-in-r.html#
https://rpubs.com/crazyhottommy/reorder-boxplot

Reg no :j20200776
University ID : w1831982

Answer for the 2nd question (Neural network question two -part one)

Scope:
A neural network is a computational system frequently employed in machine learning to create predictions based on existing data

1.Load the necessary libraries .

```
knitr::opts_chunk$set(echo = TRUE)
library(tidyverse)
library(readxl)
library(lubridate)
library(zoo)
library(tidymodels)
library(readxl)
library(neuralnet)
library(knitr)
```

2.Load the CSV and mutating to clear the column names.

```
exchangeGBP <- read_excel("exchangeGBP.xlsx") %>%
  janitor::clean_names() %>%
  mutate(date_in_ymd = ymd(yyyy_mm_dd)) %>%
  select(-1) %>%
  select(date_in_ymd,everything())
```

3.Creating different input variables ,to avoid overfitting model

```
#all the input is in only one dataframe to be able to preserve the testing
and training
#Preparing multiple attributes using the existing single attributes,lag
function can help us to do this.
gbp_exchange_full_data <- exchangeGBP
gbp_exchange_full_data <-
mutate(gbp_exchange_full_data,previous_one_day_set_a =
lag(exchangeGBP$gbp_eur,1))
```

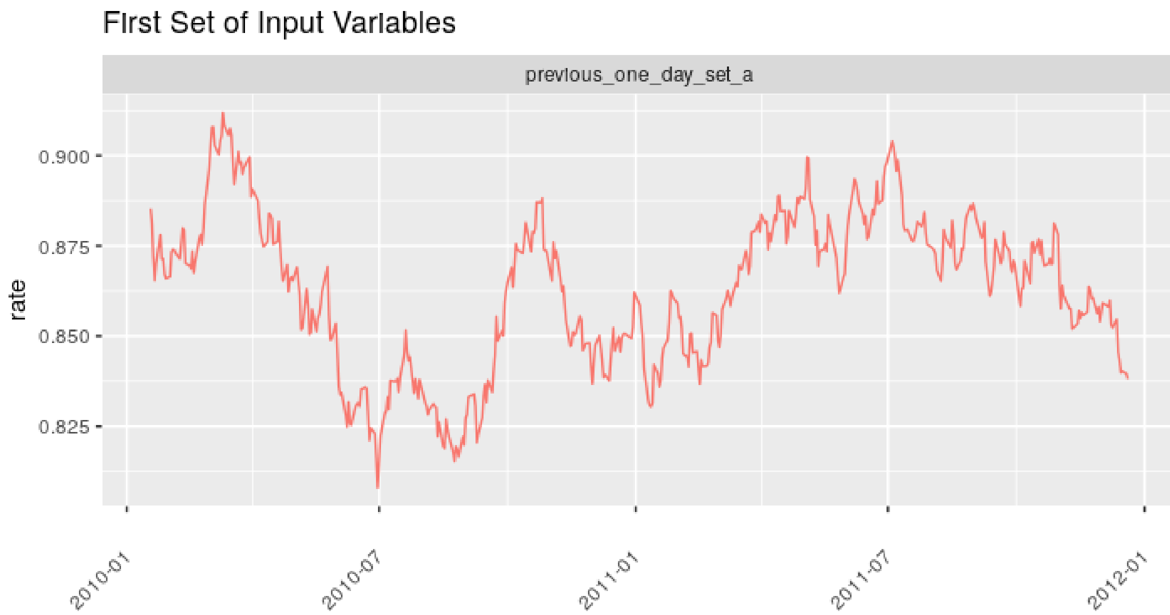
```
gbp_exchange_full_data <-
mutate(gbp_exchange_full_data,previous_two_days_set_a =
lag(exchangeGBP$gbp_eur,2))
gbp_exchange_full_data <-
mutate(gbp_exchange_full_data,previous_three_days_set_a =
lag(exchangeGBP$gbp_eur,3))
gbp_exchange_full_data <-
mutate(gbp_exchange_full_data,previous_four_days_set_a =
lag(exchangeGBP$gbp_eur,4))
gbp_exchange_full_data <- mutate(gbp_exchange_full_data,
previous_five_days_set_a = lag(exchangeGBP$gbp_eur,5))
gbp_exchange_full_data <- mutate(gbp_exchange_full_data,
previous_six_days_set_a = lag(exchangeGBP$gbp_eur,6))
gbp_exchange_full_data <- mutate(gbp_exchange_full_data,
previous_seven_days_set_a = lag(exchangeGBP$gbp_eur,7))
gbp_exchange_full_data <- mutate(gbp_exchange_full_data,
previous_eight_days_set_a = lag(exchangeGBP$gbp_eur,8))
gbp_exchange_full_data <- mutate(gbp_exchange_full_data,
previous_nine_days_set_a = lag(exchangeGBP$gbp_eur,9))
gbp_exchange_full_data <- mutate(gbp_exchange_full_data,
previous_ten_days_set_a = lag(exchangeGBP$gbp_eur,10))
gbp_exchange_full_data <- mutate(gbp_exchange_full_data,five_day_rolling =
rollmean(gbp_eur,5, fill = NA))
gbp_exchange_full_data <- mutate(gbp_exchange_full_data,ten_day_rolling =
rollmean(gbp_eur,10, fill = NA))

#dropping null records
gbp_exchange_full <- drop_na(gbp_exchange_full_data)
summary(gbp_exchange_full)
```

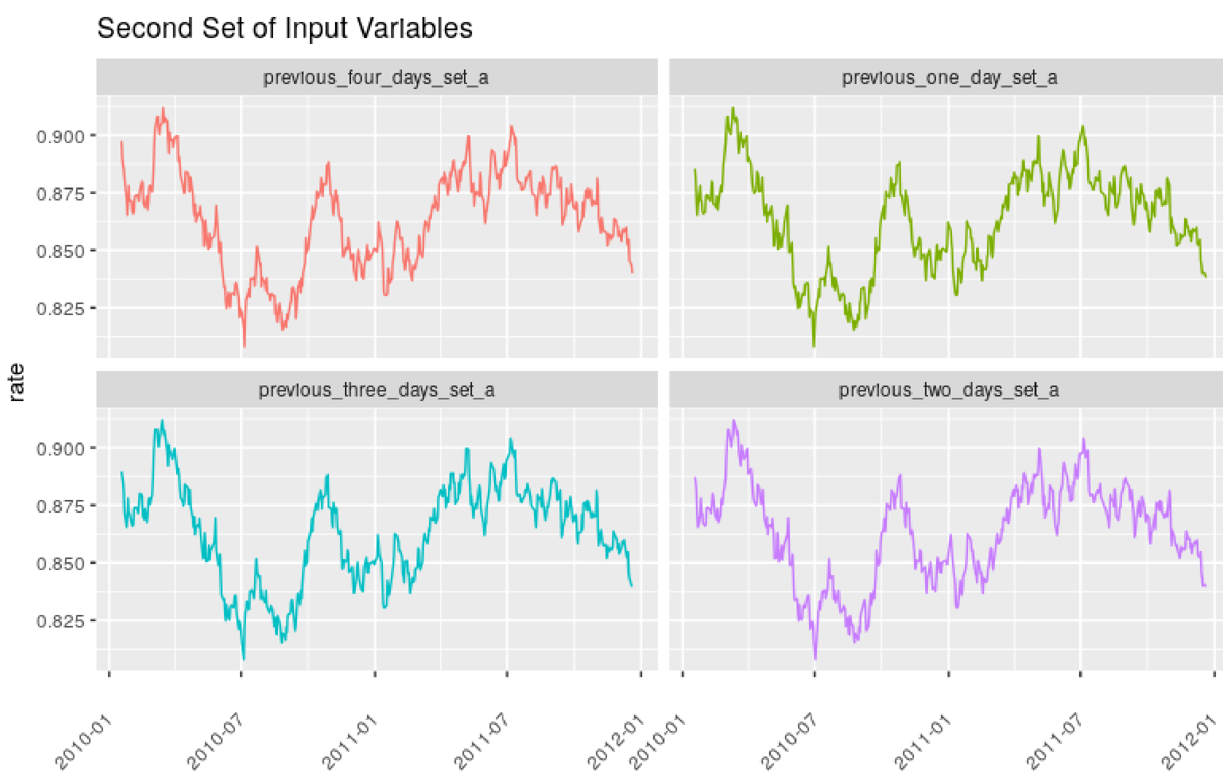
Output:

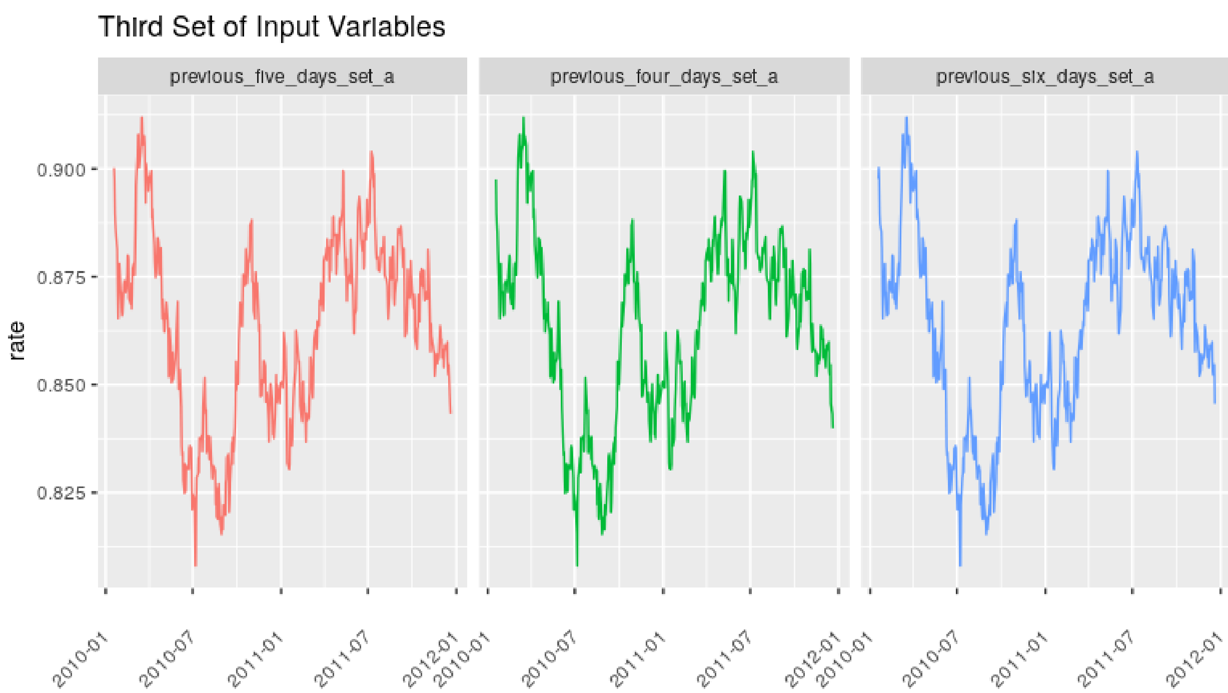
```
summary(gbp_exchange_full)
  date_in_ymd            gbp_eur       previous_one_day_set_a
previous_two_days_set_a previous_three_days_set_a previous_four_days_set_a
 Min.   :2010-01-18   Min.   :0.8080   Min.   :0.8080          Min.
:0.8080          Min.   :0.8080            Min.   :0.8080
 1st Qu.:2010-07-09   1st Qu.:0.8469   1st Qu.:0.8473          1st
Qu.:0.8475          1st Qu.:0.8475           1st Qu.:0.8479
 Median :2011-01-05   Median :0.8656   Median :0.8656          Median
:0.8657          Median :0.8660            Median :0.8662
```

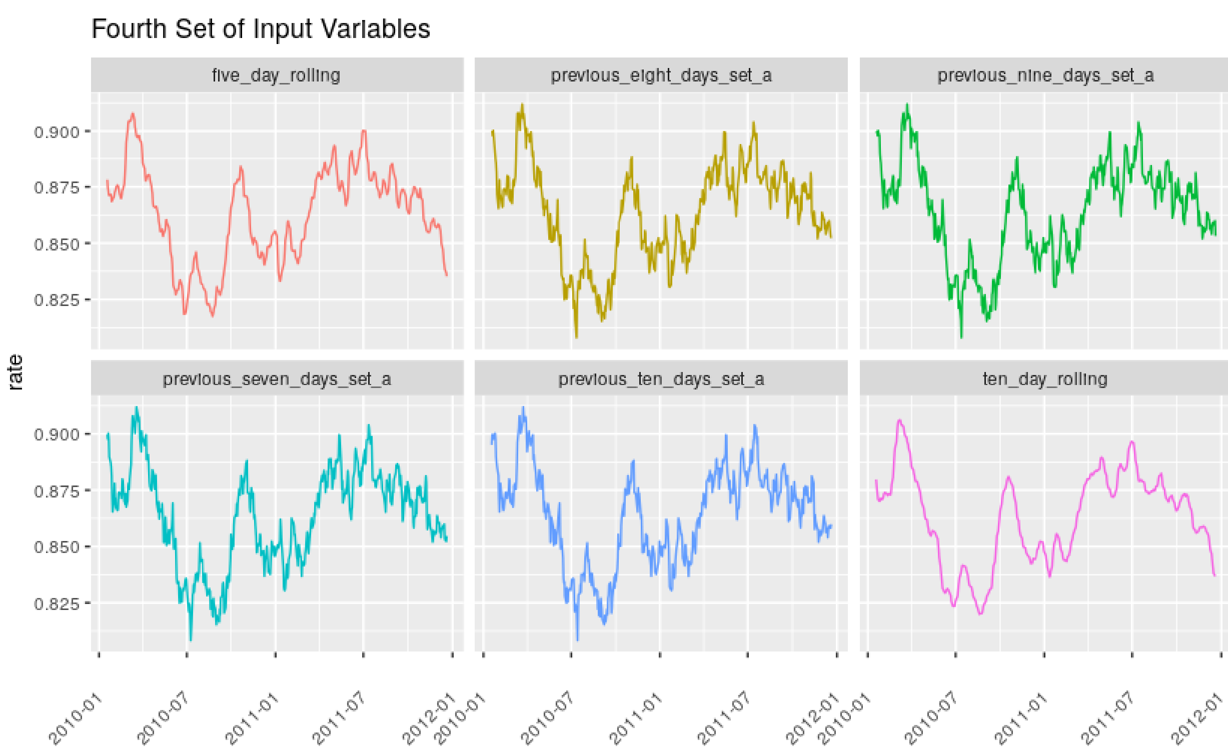
```
 Mean   :2011-01-02   Mean   :0.8628   Mean   :0.8629          Mean   
:0.8630          Mean   :0.8631            Mean   :0.8632          
 3rd Qu.:2011-06-27   3rd Qu.:0.8781   3rd Qu.:0.8784          3rd 
Qu.:0.8784          3rd Qu.:0.8786            3rd Qu.:0.8787          
 Max.   :2011-12-20   Max.   :0.9119   Max.   :0.9119          Max.   
:0.9119          Max.   :0.9119            Max.   :0.9119          
 previous_five_days_set_a previous_six_days_set_a previous_seven_days_set_a
previous_eight_days_set_a previous_nine_days_set_a
 Min.   :0.8080           Min.   :0.8080          Min.   :0.8080           
Min.   :0.8080            Min.   :0.8080          
 1st Qu.:0.8481           1st Qu.:0.8481          1st Qu.:0.8482           
1st Qu.:0.8482            1st Qu.:0.8482          
 Median :0.8664           Median :0.8664          Median :0.8664           
Median :0.8672            Median :0.8672          
 Mean   :0.8633           Mean   :0.8635          Mean   :0.8636           
Mean   :0.8637            Mean   :0.8638          
 3rd Qu.:0.8787           3rd Qu.:0.8787          3rd Qu.:0.8788           
3rd Qu.:0.8789            3rd Qu.:0.8791          
 Max.   :0.9119           Max.   :0.9119          Max.   :0.9119           
Max.   :0.9119            Max.   :0.9119          
 previous_ten_days_set_a five_day_rolling ten_day_rolling
 Min.   :0.8080          Min.   :0.8175   Min.   :0.8199  
 1st Qu.:0.8482          1st Qu.:0.8467   1st Qu.:0.8467  
 Median :0.8673          Median :0.8670   Median :0.8673  
 Mean   :0.8638          Mean   :0.8628   Mean   :0.8628  
 3rd Qu.:0.8792          3rd Qu.:0.8780   3rd Qu.:0.8778  
 Max.   :0.9119          Max.   :0.9078   Max.   :0.9062  
```

4.Before Normalization .


First Set of Input Variables
previous_one_day_set_a
rate
0.900
0.875
0.850
0.825
2010-01
2010-07
2011-01
2011-07
2012-01


Second Set of Input Variables
previous_four_days_set_a
previous_one_day_set_a
previous_three_days_set_a
previous_two_days_set_a
rate
0.900
0.875
0.850
0.825
2010-01
2010-07
2011-01
2011-07
2012-01


Third Set of Input Variables
previous_five_days_set_a
previous_four_days_set_a
previous_six_days_set_a
rate
0.900
0.875
0.850
0.825
2010-01
2010-07
2011-01
2011-07
2012-01


Fourth Set of Input Variables
five_day_rolling
previous_eight_days_set_a
previous_nine_days_set_a
previous_seven_days_set_a
previous_ten_days_set_a
ten_day_rolling
rate
0.900
0.875
0.850
0.825
2010-01
2010-07
2011-01
2011-07
2012-01

5.Normalization ,pre-processing , creating a function to normalize the data from 0 to 1

```
# We can create a function to normalize the data from 0 to 1
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x))) }
```

6.Unnormailizing function ,provided values are not normalized

```
# a function to unnormalize the data=
unnormalize <- function(x, min, max) {
  return( (max - min)*x + min ) }
```

7..All variables are normalized before the training.

```
# All the variables are normalized
normalized_gbp = gbp_exchange_full %>%
  mutate(across(2:14, ~normalize(.x)))
# Look at the data that has been normalized
summary(normalized_gbp)
```

8..Creating normalized training data and testing data.

```
set.seed(123)
gbp_train <- normalized_gbp[1:400,]
gbp_test <- normalized_gbp[401:480,]
```

9.Start with single input to monitor the RMSE.

```
 nn_model_true = neuralnet(gbp_eur ~ previous_one_day_set_a,
data=gbp_train, hidden=c(
    hidden,sec_hidden), linear.output=TRUE)
```

Output:
RMSE 0.0045653

```
|type                |hiddel_layers |input_set |      rmse|       mae|      mape|
|:-----------------|:-------------|:---------|---------:|---------:|---------:|
|Two Hidden Layers |8 and 3       |A         | 0.0045653| 0.0035052| 0.4042473|
|Two Hidden Layers |8 and 5       |A         | 0.0045731| 0.0035173| 0.4056622|
|Two Hidden Layers |4 and 5       |A         | 0.0045909| 0.0035211| 0.4060226|
|Two Hidden Layers |3 and 1       |A         | 0.0045936| 0.0035204| 0.4059596|
|Two Hidden Layers |5 and 5       |A         | 0.0045981| 0.0035312| 0.4072101|
|Two Hidden Layers |7 and 5       |A         | 0.0046043| 0.0035229| 0.4062587|
|Two Hidden Layers |3 and 4       |A         | 0.0046055| 0.0035306| 0.4071513|
|Two Hidden Layers |3 and 2       |A         | 0.0046074| 0.0035387| 0.4080468|
|Two Hidden Layers |6 and 5       |A         | 0.0046084| 0.0035363| 0.4077793|
|Two Hidden Layers |9 and 5       |A         | 0.0046110| 0.0035413| 0.4083698|
```

10. Two attributes as inputs:

```
nn_model_true = neuralnet(gbp_eur ~
previous_one_day_set_a+previous_two_days_set_a, data=gbp_train, hidden=c(
    hidden,sec_hidden), linear.output=TRUE)
```

```
RMSE:0.0045668
```

```
|type
    |hiddel_layers |input_set |      rmse|       mae|      mape|
|:-----------------|:-------------|:---------|---------:|---------:|---------:|
|Two Hidden Layers |1 and 3       |A         | 0.0045668| 0.0035097| 0.4048915|
|Two Hidden Layers |8 and 2       |A         | 0.0045910| 0.0035212|
```

```
0.4061200|
|Two Hidden Layers |10 and 5       |A          | 0.0045986| 0.0035253|
0.4065488|
|Two Hidden Layers |8 and 3        |A          | 0.0045991| 0.0035264|
0.4067053|
|Two Hidden Layers |7 and 4        |A          | 0.0045991| 0.0035261|
0.4066515|
|Two Hidden Layers |5 and 3        |A          | 0.0045997| 0.0035264|
0.4066982|
|Two Hidden Layers |5 and 5        |A          | 0.0046001| 0.0035265|
0.4067173|
|Two Hidden Layers |7 and 1        |A          | 0.0046015| 0.0035282|
0.4068507|
|Two Hidden Layers |8 and 4        |A          | 0.0046017| 0.0035317|
0.4072263|
|Two Hidden Layers |8 and 5        |A          | 0.0046029| 0.0035303|
0.4070994|
```

11.Four attributes:

```
nn_model_true = neuralnet(gbp_eur ~
previous_one_day_set_a+previous_two_days_set_a

+previous_three_days_set_a+previous_four_days_set_a
                            , data=gbp_train, hidden=c(
    hidden,sec_hidden), linear.output=TRUE)
```

```
RMSE:0.0045653
```

```
type              |hiddel_layers |input_set |      rmse|       mae|
mape|
|:-----------------|:-------------|:---------|---------:|---------:|-------
--:|
|Two Hidden Layers |8 and 3        |A          | 0.0045653| 0.0035052|
0.4042473|
|Two Hidden Layers |8 and 5        |A          | 0.0045731| 0.0035173|
0.4056622|
|Two Hidden Layers |4 and 5        |A          | 0.0045909| 0.0035211|
0.4060226|
```

```
|Two Hidden Layers |3 and 1        |A          | 0.0045936| 0.0035204|
0.4059596|
|Two Hidden Layers |5 and 5        |A          | 0.0045981| 0.0035312|
0.4072101|
|Two Hidden Layers |7 and 5        |A          | 0.0046043| 0.0035229|
0.4062587|
|Two Hidden Layers |3 and 4        |A          | 0.0046055| 0.0035306|
0.4071513|
|Two Hidden Layers |3 and 2        |A          | 0.0046074| 0.0035387|
0.4080468|
|Two Hidden Layers |6 and 5        |A          | 0.0046084| 0.0035363|
0.4077793|
|Two Hidden Layers |9 and 5        |A          | 0.0046110| 0.0035413|
0.4083698|
```

12.Input all attributes

```
nn_model_true = neuralnet(gbp_eur ~
previous_one_day_set_a+previous_two_days_set_a

+previous_three_days_set_a+previous_four_days_set_a

+previous_five_days_set_a+previous_six_days_set_a

+previous_seven_days_set_a+previous_eight_days_set_a

+previous_nine_days_set_a+previous_ten_days_set_a
                            +five_day_rolling+ten_day_rolling
                            , data=gbp_train, hidden=c(
    hidden,sec_hidden), linear.output=TRUE)
```

```
RMSE best : 0.0045307
```

```
|type              |hiddel_layers |input_set |      rmse|       mae|
mape|
|:-----------------|:-------------|:---------|---------:|---------:|-------
```

```
--:|
|Two Hidden Layers |7 and 4        |A          | 0.0045307| 0.0033990|
0.3918382|
|Two Hidden Layers |1 and 4        |A          | 0.0045445| 0.0034570|
0.3986186|
|Two Hidden Layers |10 and 1       |A          | 0.0045525| 0.0034605|
0.3989274|
|Two Hidden Layers |9 and 5        |A          | 0.0045566| 0.0034648|
0.3995438|
|Two Hidden Layers |4 and 4        |A          | 0.0045595| 0.0034695|
0.4000530|
|Two Hidden Layers |7 and 3        |A          | 0.0045657| 0.0034740|
0.4004784|
|Two Hidden Layers |8 and 1        |A          | 0.0045669| 0.0034714|
0.4001836|
|Two Hidden Layers |5 and 3        |A          | 0.0045736| 0.0035118|
0.4048969|
|Two Hidden Layers |2 and 1        |A          | 0.0045753| 0.0034941|
0.4029230|
|Two Hidden Layers |5 and 1        |A          | 0.0045812| 0.0034938|
0.4027688|
```

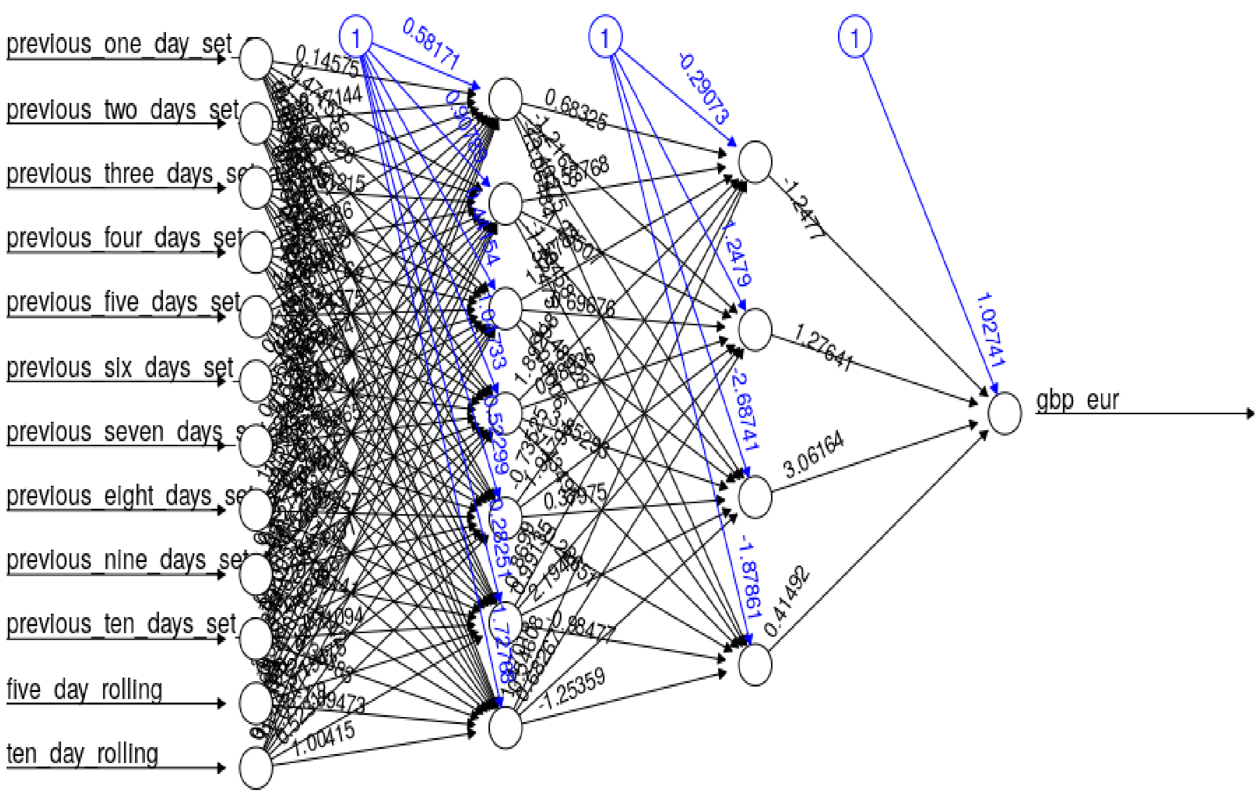
The best RMSE appears when the large number of attributes given to the function and the numbers 7 and 4 passed as the parameters for the hidden layer .

13.As per the above statement the best parameters are used here.

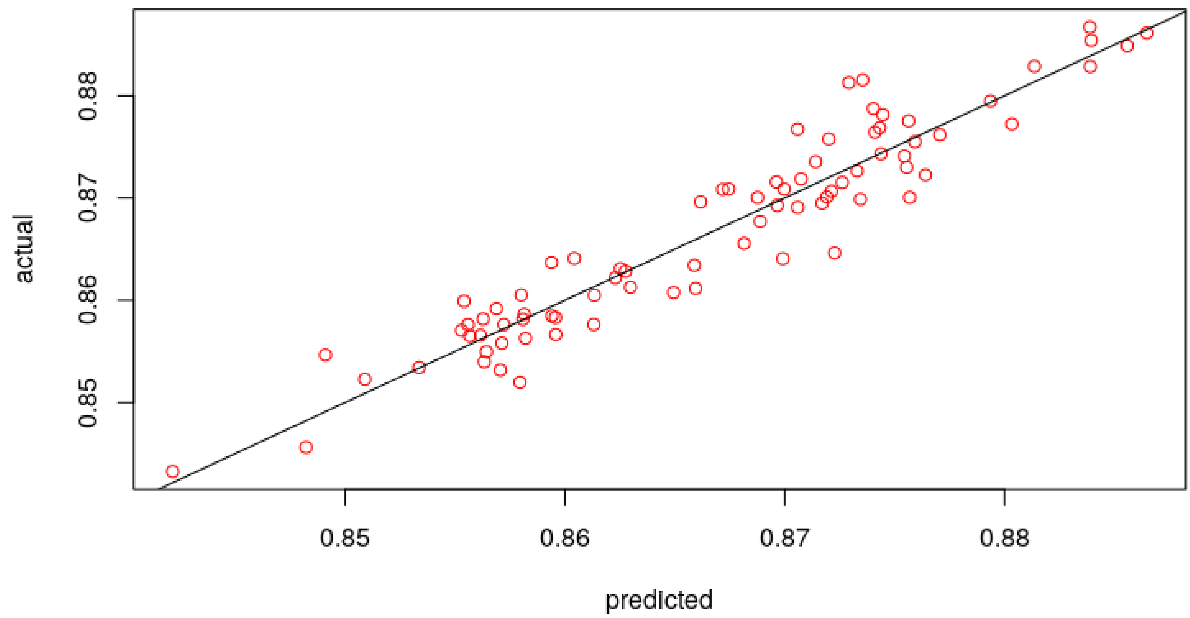
```
###fit the hidden layer as best results
nn_model_true = neuralnet(gbp_eur ~ gbp_eur ~
previous_one_day_set_a+previous_two_days_set_a

+previous_three_days_set_a+previous_four_days_set_a
                          +previous_five_days_set_a+previous_six_days_set_a

+previous_seven_days_set_a+previous_eight_days_set_a
                          +previous_nine_days_set_a+previous_ten_days_set_a
                          +five_day_rolling+ten_day_rolling,
data=gbp_train, hidden=c(
  7,4), linear.output=TRUE)
train_results = compute(nn_model_true,gbp_test[,2:14])
```

```
truthcol = gbp_exchange_full[401:480,2]$gbp_eur
predcol = unnormalize(train_results$net.result,gbp_min_train,
                      gbp_max_train)[,1]
relevant_pred_stat(truthcol,predcol,
                   "Two Hidden Layers") %>%
  mutate(hiddel_layers = paste0(2, " and ",3),
         input_set = "A") %>%
  filter(.metric != "rsq")
plot(nn_model_true)
##truth column vs predict column
plot(predcol,truthcol, col='red',
     xlab="predicted",ylab="actual")
abline(a=0,b=1)
```

Outputs:



Predicted values vs Desired values




```
metric .estimator .estimate type              hiddel_layers input_set
  <chr>    <chr>           <dbl> <chr>             <chr>         <chr>
1 rmse     standard      0.00306 Two Hidden Layers 2 and 3       A
2 mae      standard      0.00240 Two Hidden Layers 2 and 3       A
3 mape     standard      0.276   Two Hidden Layers 2 and 3       A
```

```
results <- data.frame(actual = truthcol, prediction = predcol)
```

```
actual prediction
1  0.87087  0.8699776
2  0.87432  0.8743794
3  0.87409  0.8754448
4  0.87947  0.8793609
5  0.88285  0.8839014
```

```
6  0.88613  0.8864715
7  0.88488  0.8855813
8  0.88671  0.8838834
9  0.88541  0.8839474
10 0.88287  0.8813606
11 0.87722  0.8803334
12 0.87753  0.8756385
13 0.88154  0.8735469
14 0.87065  0.8721256
15 0.86115  0.8659384
16 0.86218  0.8622917
17 0.86555  0.8681542
18 0.86926  0.8696588
19 0.87672  0.8705859
20 0.87225  0.8764025
21 0.87005  0.8756848
22 0.87151  0.8726179
23 0.87874  0.8740336
24 0.87549  0.8759220
25 0.87354  0.8714041
26 0.86908  0.8705910
27 0.86770  0.8688806
28 0.87088  0.8674486
29 0.86961  0.8661713
30 0.86076  0.8649416
31 0.85830  0.8595675
```

Final output:

```
> deviation=((results[,1]-results[,2])/results[,1])
> comparison=data.frame(results[,2],results[,1],deviation)
> accuracy=1-abs(mean(deviation))
> accuracy
[1] 0.9999235
with hidden layers 7 and 4 - all attributes
```

Reg no :j20200776
University ID : w1831982

Answer for the 2nd question (<u>Neural network question two -part two(SVR and SVM)</u>)

Support Vector Regression (SVR) works on similar principles as Support Vector Machine (SVM) classification. It means that SVR is the adapted form of SVM when the dependent variable is numerical rather than categorical.

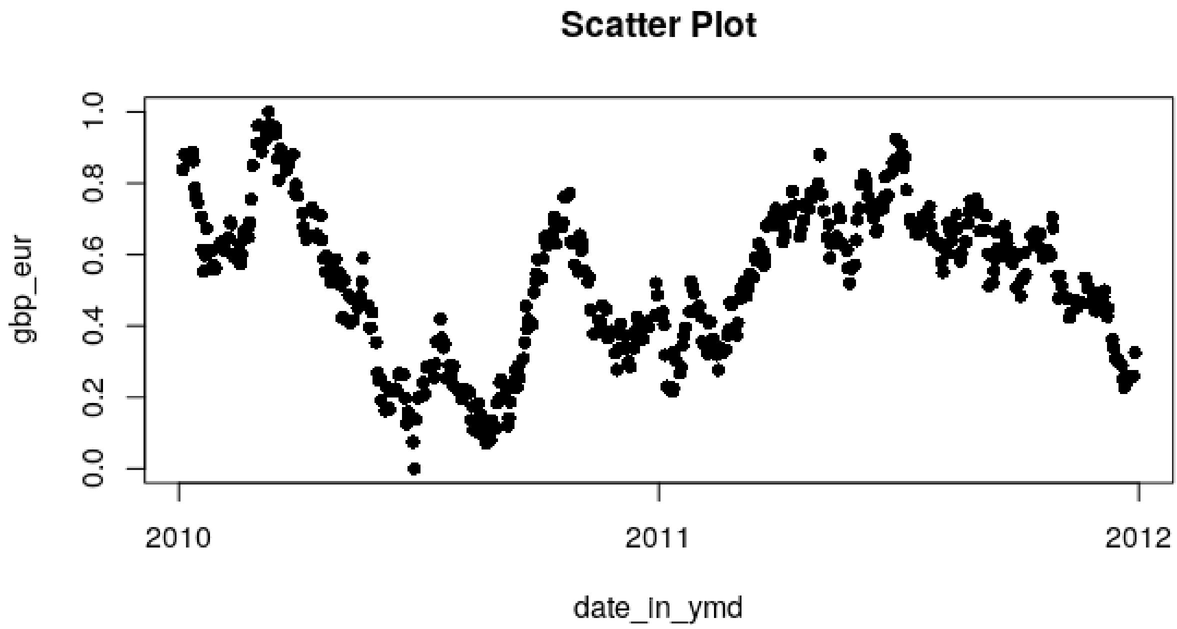
Consider gbp_eur as a dependent variable.

1.In put selection,preprocessing and normalizing the data

```
#load CSV and mutating the fields
data_gb <- read_excel("exchangeGBP.xlsx") %>%
  janitor::clean_names() %>%
  mutate(date_in_ymd = ymd(yyyy_mm_dd)) %>%
  select(-1) %>%  ## removed unwanted columns ,first
  select(date_in_ymd,everything())
head(data_gb)



# We can create a function to normalize the data from 0 to 1
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x))) }

# All the variables are normalized
normalized_gbp = data_gb %>%
  mutate(across(2:2, ~normalize(.x)))
```

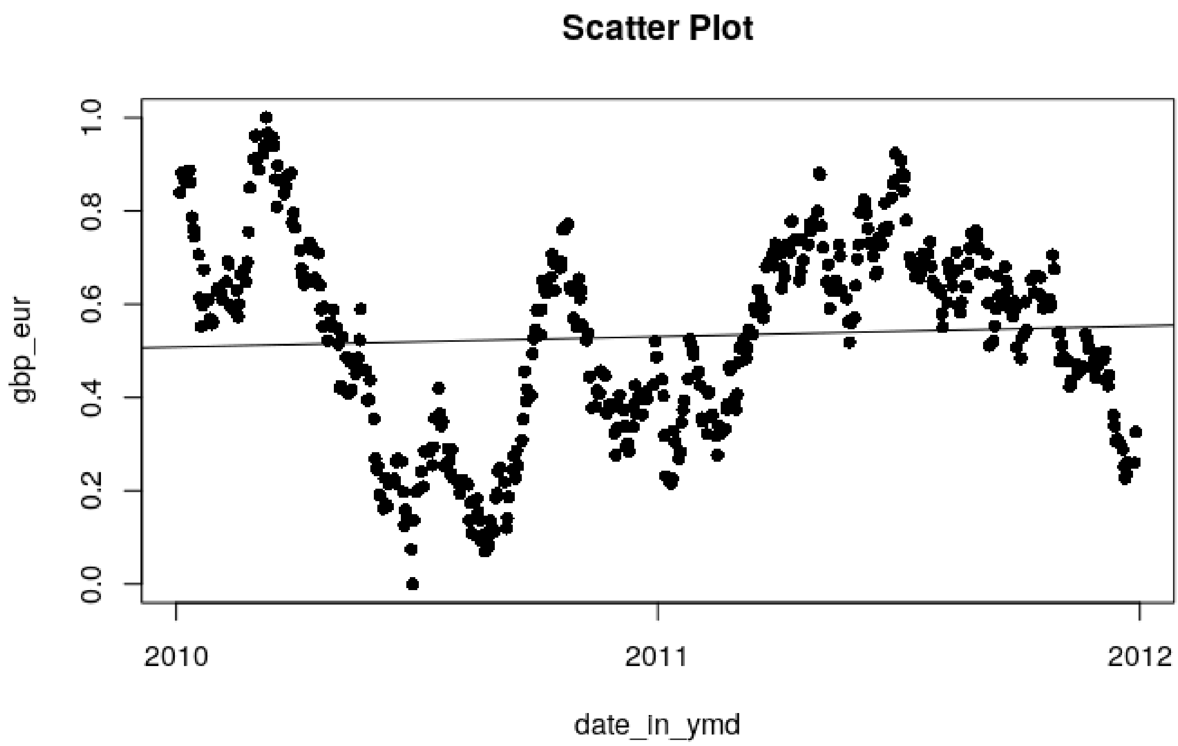
2.Plot

```
#Scatter Plot
plot(normalized_gbp, main ="Scatter Plot", pch=16)
```



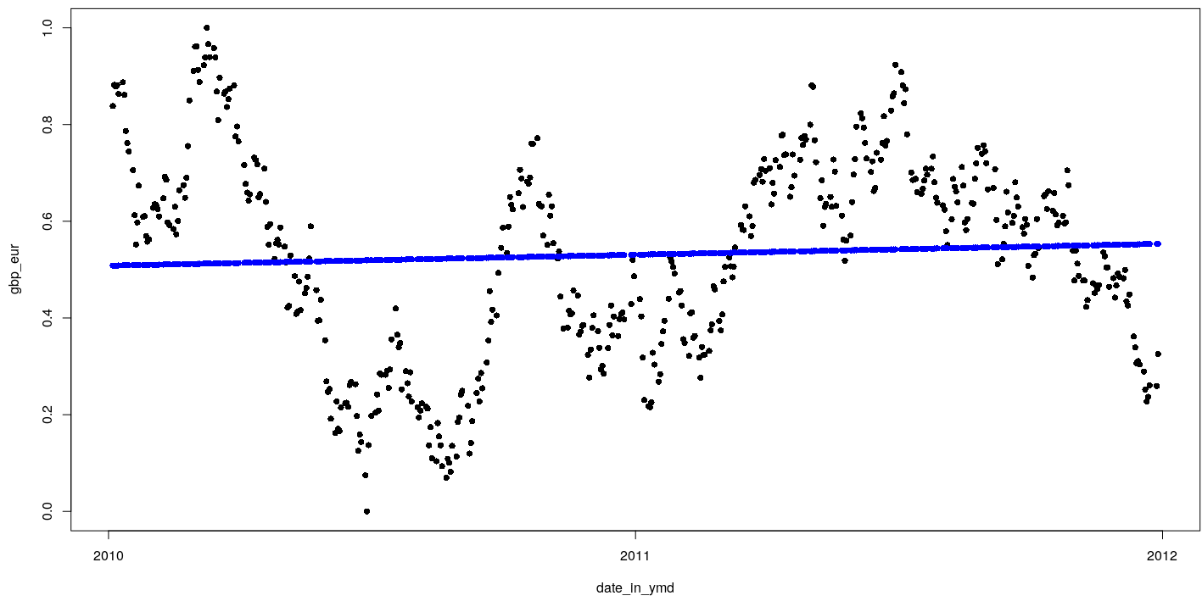
3.First do a simple linear regression, and fitting line on the scatter plot using Ordinary Least Squares (OLS) method.

```
#Fit linear model using OLS
linear_model=lm(gbp_eur~date_in_ymd,normalized_gbp)

#Overlay best-fit line on scatter plot
abline(linear_model)
```



4.Scatter plot displaying actual values and predicted values using the linear model

```
## Scatter plot displaying actual values and predicted values
# make a prediction for each X
predictedY <- predict(linear_model, normalized_gbp)

# display the predictions
points(data_gb$date_in_ymd, predictedY, col = "blue", pch=16)
```



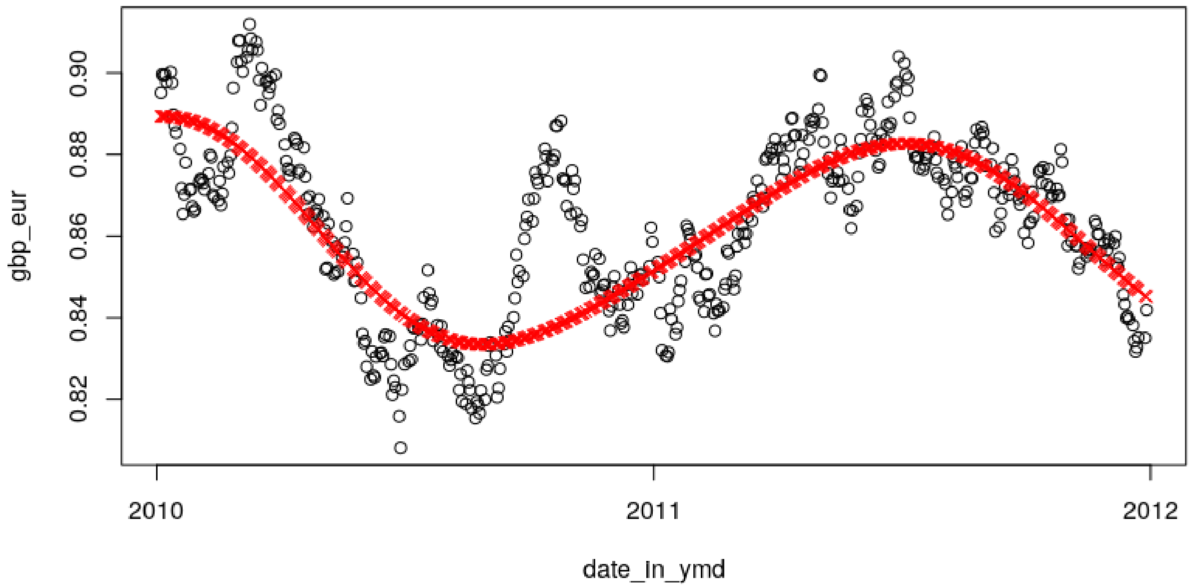
5.RMSE calculation

```
#Calculate RMSE
RMSE=rmse(predictedY,data_gb$gbp_eur)
```

```
> RMSE Linear
[1] 0.3334665
```

6.Using SVR model and visualizing in scatter plot ,predicted vs desired values.

```
#Scatter Plot
plot(normalized_gbp)
#Regression with SVM
modelsvm <- e1071::svm(gbp_eur~date_in_ymd,normalized_gbp)

predictedYSVM <- predict(modelsvm, normalized_gbp)

#Overlay SVM Predictions on Scatter Plot
points(normalized_gbp$date_in_ymd, predictedYSVM, col = "red", pch=4)
```



7.Calculate RMSE

```
RMSEsvm=rmse(predictedYSVM,normalized_gbp$gbp_eur)
```

```
> RMSEsvm
[1] 0.1348122
```

Compare RMSE:

`RMSE Linear`
`[1] 0.3334665`
Against
`> RMSEsvm`
`[1] 0.1348122`

RMSE SVM has much better results

8.Tuning SVR model by varying values of maximum allowable error and cost parameter.

we use the tune method to train models with

$$\epsilon=0,0.1,0.2,...,1$$

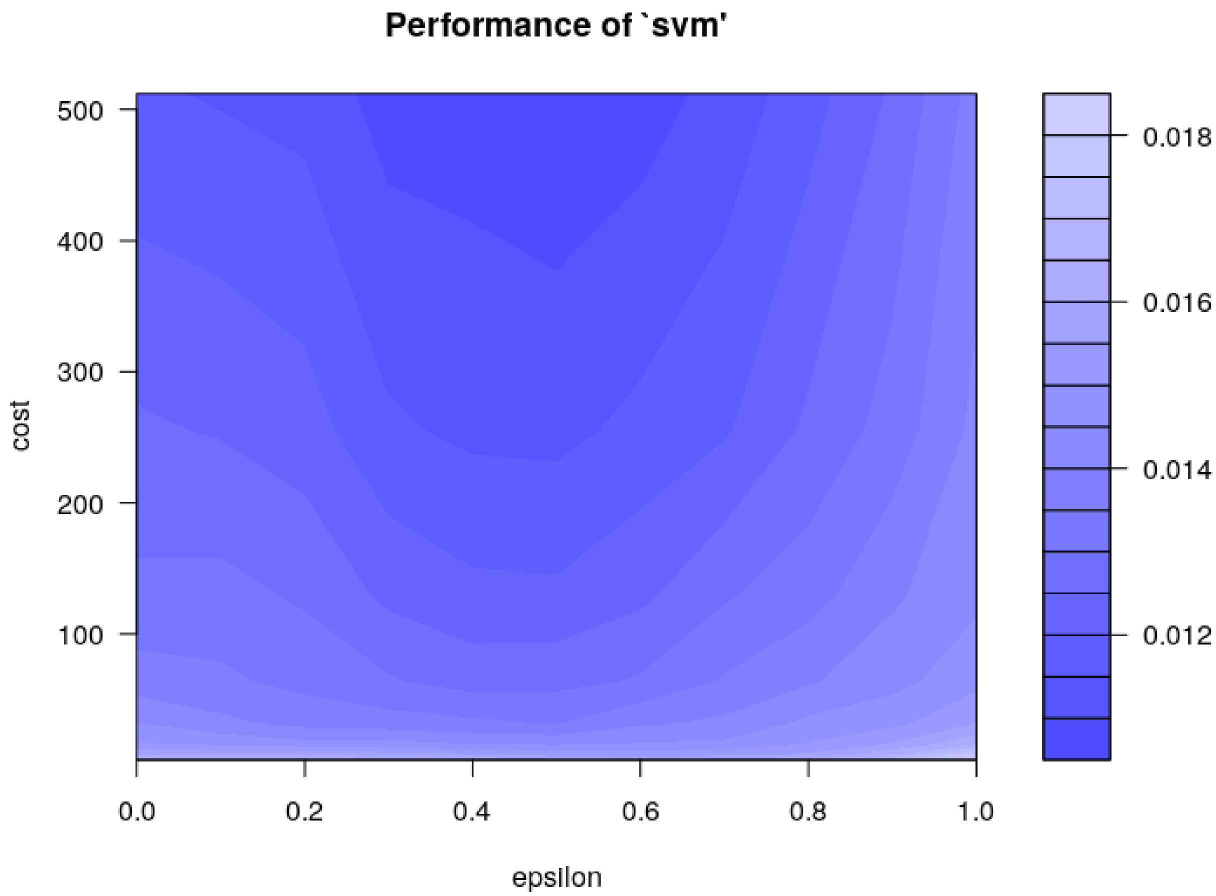
ϵ=0,0.1,0.2,...,1

```
#Tune the SVM model
tuneResult <- tune(svm, gbp_eur ~ date_in_ymd,  data = normalized_gbp,
                   ranges = list(epsilon = seq(0,1,0.1), cost = 2^(2:9))
)

print(tuneResult)
# Draw the tuning graph
plot(tuneResult)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 epsilon cost
     0.5  512

- best performance: 0.01067864 (MSE)
```

```
#RMSE
sqrt(0.01067864);
```

```
RMSE = 0.1033375
```

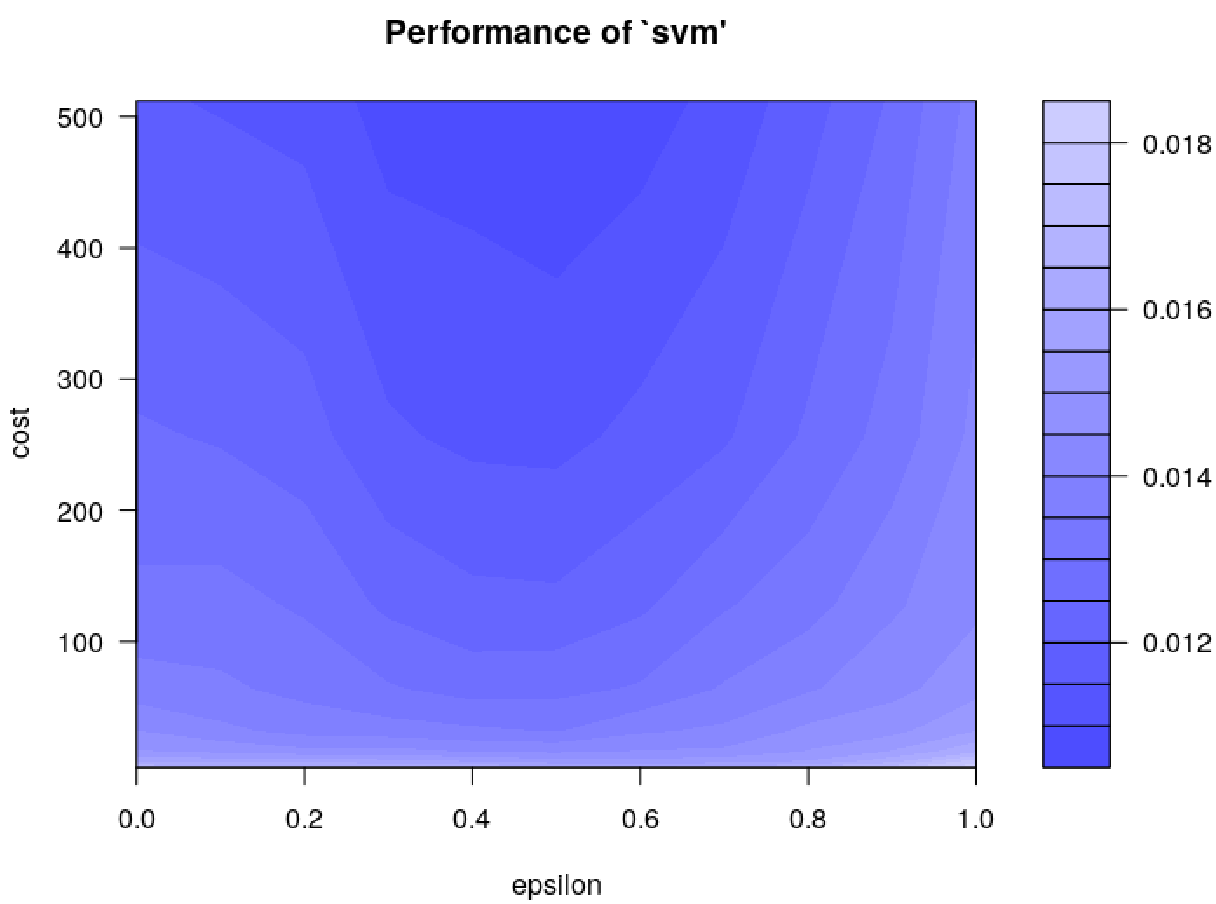


10.Using ϵ values between 0 and 0.2. It does not look like the cost value is having an effect for the moment so keep it as it is to see if it changes.

```
tuneResult <- tune(svm,gbp_eur ~ date_in_ymd,  data = normalized_gbp,
                   ranges = list(epsilon = seq(0,0.2,0.01), cost = 2^(2:9))
)

print(tuneResult)
plot(tuneResult)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 epsilon cost
     0.2  512

- best performance: 0.01115691
```

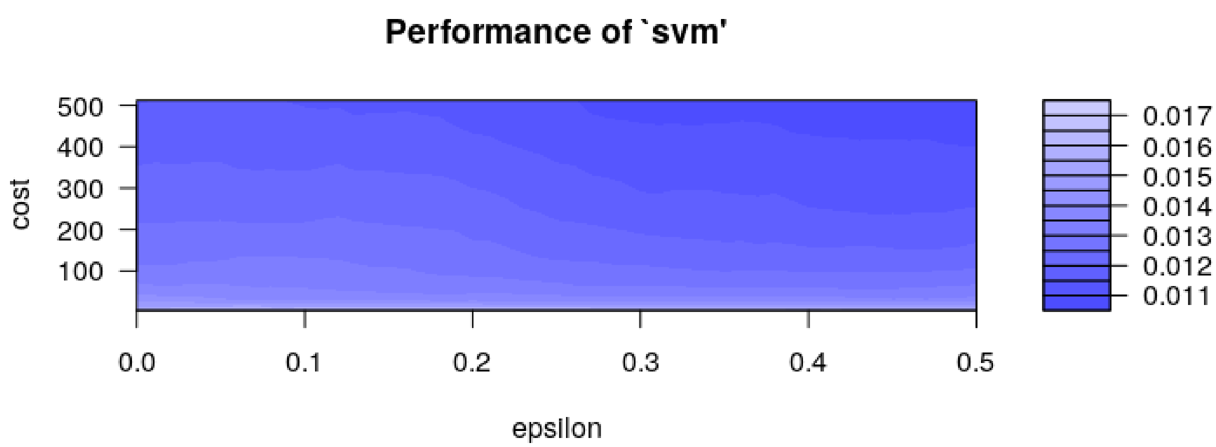
```
#RMSE
> sqrt(0.01115691);
[1] 0.1056263
```



11. It looks like no changes on cost but the best model epsilon `is 0.5 and cost 512.`

- Let see predicted vs real data while using best parameters

```
tuneResult <- tune(svm,gbp_eur ~ date_in_ymd,  data = normalized_gbp,
                   ranges = list(epsilon = seq(0,0.5,0.01), cost = 2^(2:9))
)

print(tuneResult)
plot(tuneResult)
```

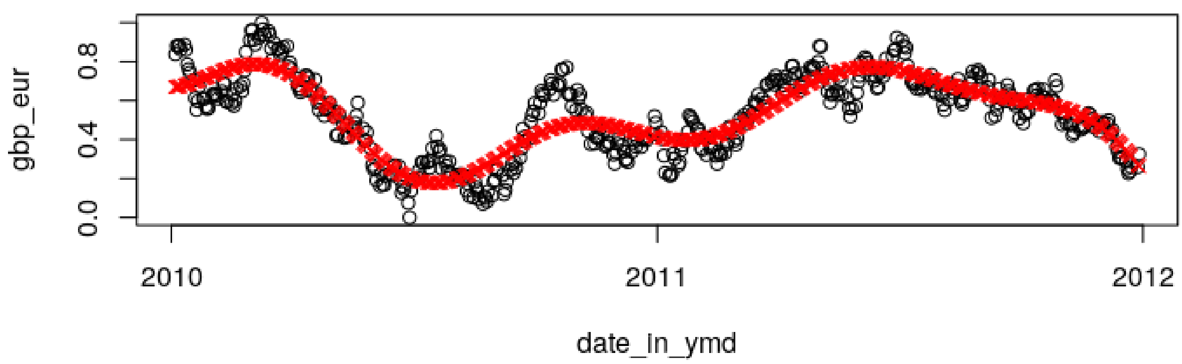



```
best.tune(method = svm, train.x = gbp_eur ~ date_in_ymd, data =
normalized_gbp, ranges = list(epsilon = seq(0,
    0.5, 0.01), cost = 2^(2:9)))


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  512
      gamma:  1
    epsilon:  0.5
```

```
Number of Support Vectors:  170
```

- Tuned up model - predicted vs desired plot.

```
predictedYSVM <- predict(tunedModel, normalized_gbp)

#Overlay SVM Predictions on Scatter Plot
plot(normalized_gbp)
points(normalized_gbp$date_in_ymd, predictedYSVM, col = "red", pch=4)


abline(a=0,b=1)
```



- Final Decision of the above study of the SVM, proves that the epsilon is 0.5 and the cost 512 which is the most suitable parameter to the SVM model for the prediction to the dataset.

References:

https://www.svm-tutorial.com/2014/10/support-vector-regression-r/
https://www.kdnuggets.com/2017/03/building-regression-models-support-vector-regression.html