Last Name: Pandey
First Name: Aditya
Student ID: 1001405034
Course: Data Mining CSE 5334 Fall 2016
Topic: Advanced Programming Assignment

<u>Step 1: Description of the Dataset</u>

<u>Name:</u>

Fertility Data Set

<u>Attribute Information:</u>

Season in which the analysis was performed. 1) winter, 2) spring, 3) Summer, 4) fall. (-1, -0.33, 0.33, 1)

Age at the time of analysis. 18-36 (0, 1)

Childish diseases (ie , chicken pox, measles, mumps, polio) 1) yes, 2) no. (0, 1)

Accident or serious trauma 1) yes, 2) no. (0, 1)

Surgical intervention 1) yes, 2) no. (0, 1)

High fevers in the last year 1) less than three months ago, 2) more than three months ago, 3) no. (-1, 0, 1)

Frequency of alcohol consumption 1) several times a day, 2) every day, 3) several times a week, 4) once a week, 5) hardly ever or never (0, 1)

Smoking habit 1) never, 2) occasional 3) daily. (-1, 0, 1)

Number of hours spent sitting per day ene-16 (0, 1)

Output: Diagnosis normal (N), altered (O)

Step 2: Initializing the datasets

Figure 1: Summary of the dataset

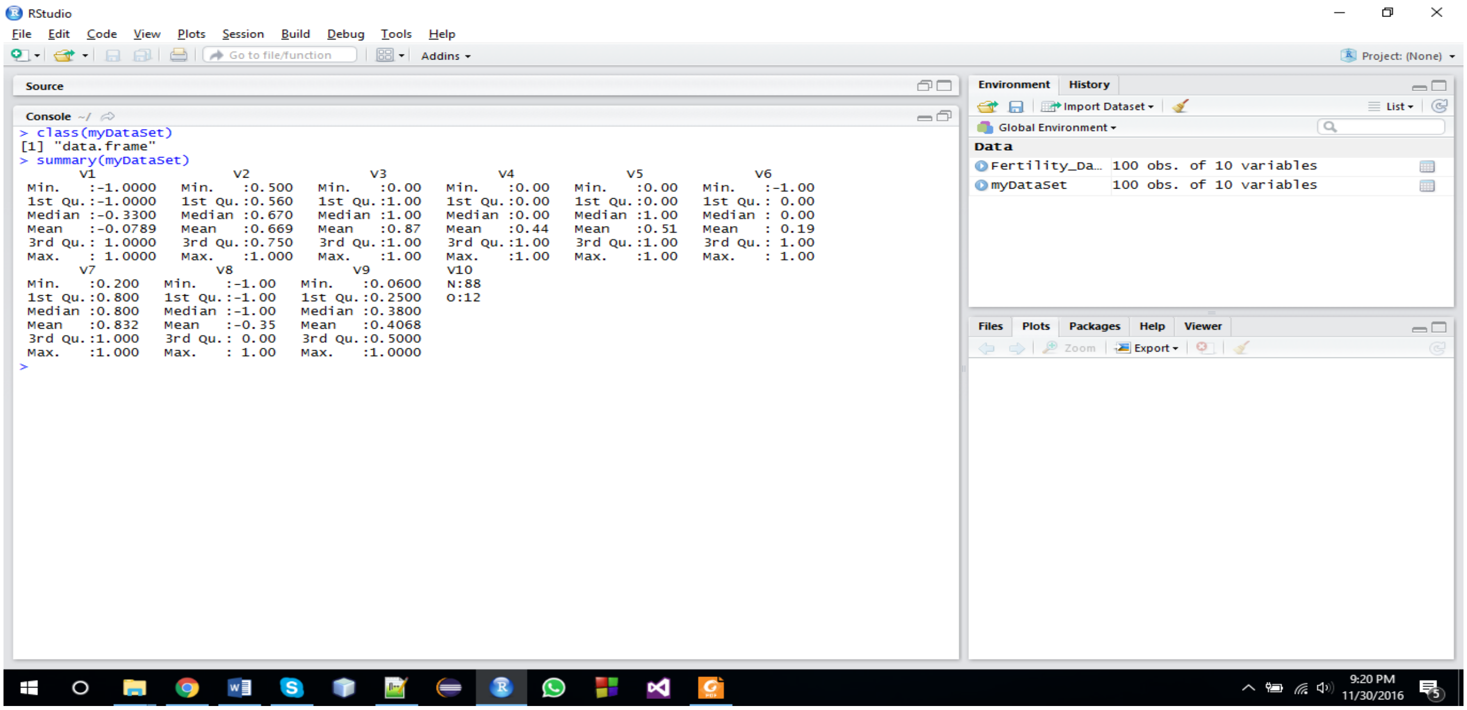


Figure 2: head(myDataSet) > Display first 5 tuples of myDataSet

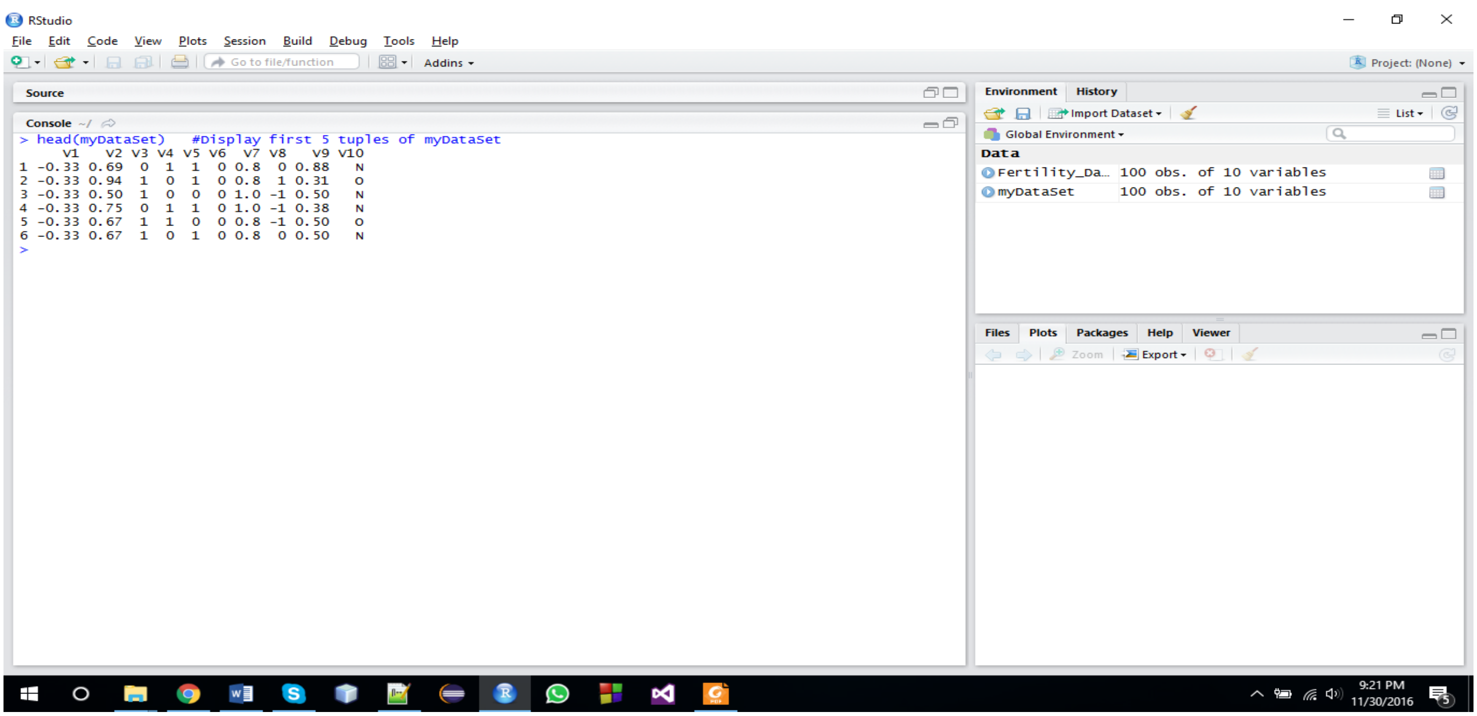
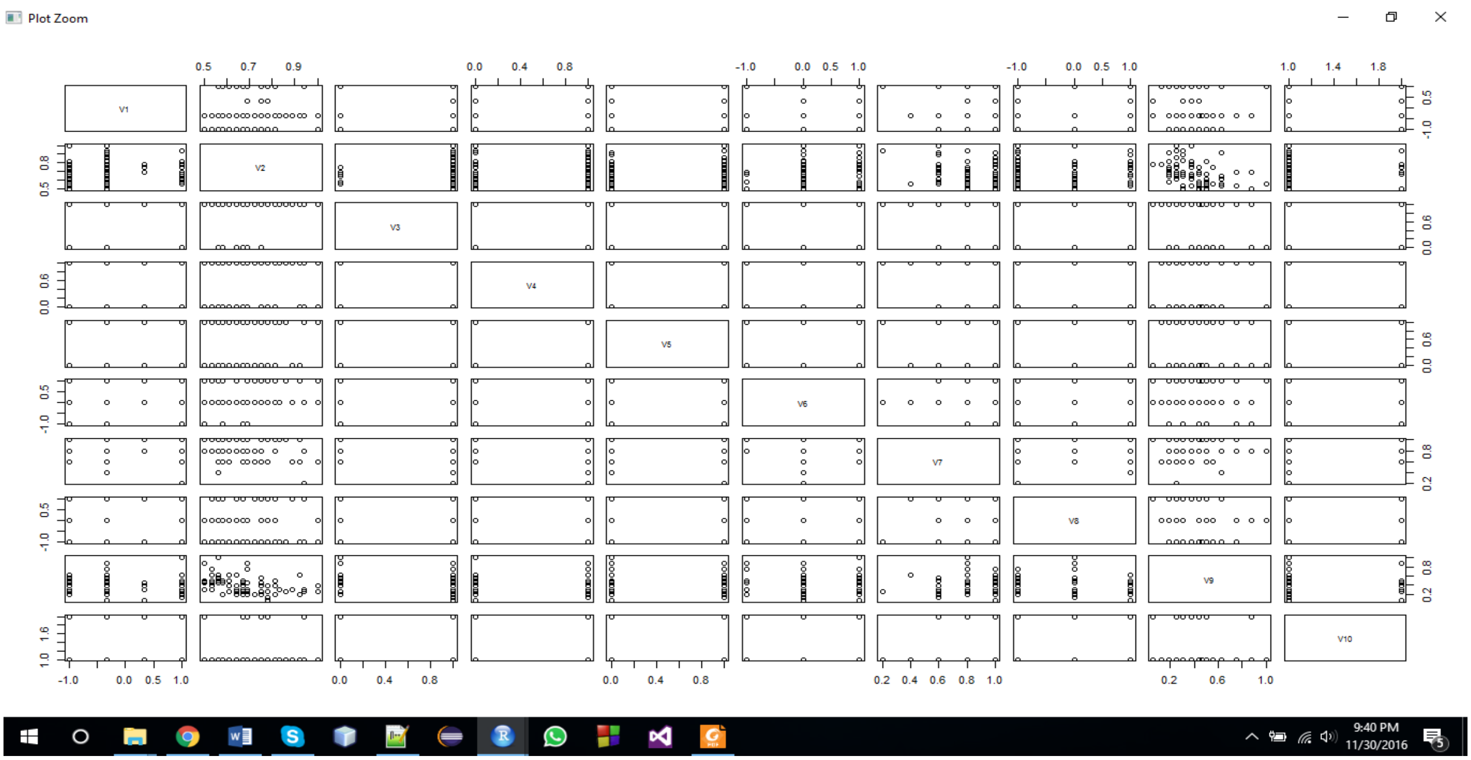


Figure 3: Plotting the data



Step 3: KNN Classifier Implementation

Figure 4: Defining Normalize function

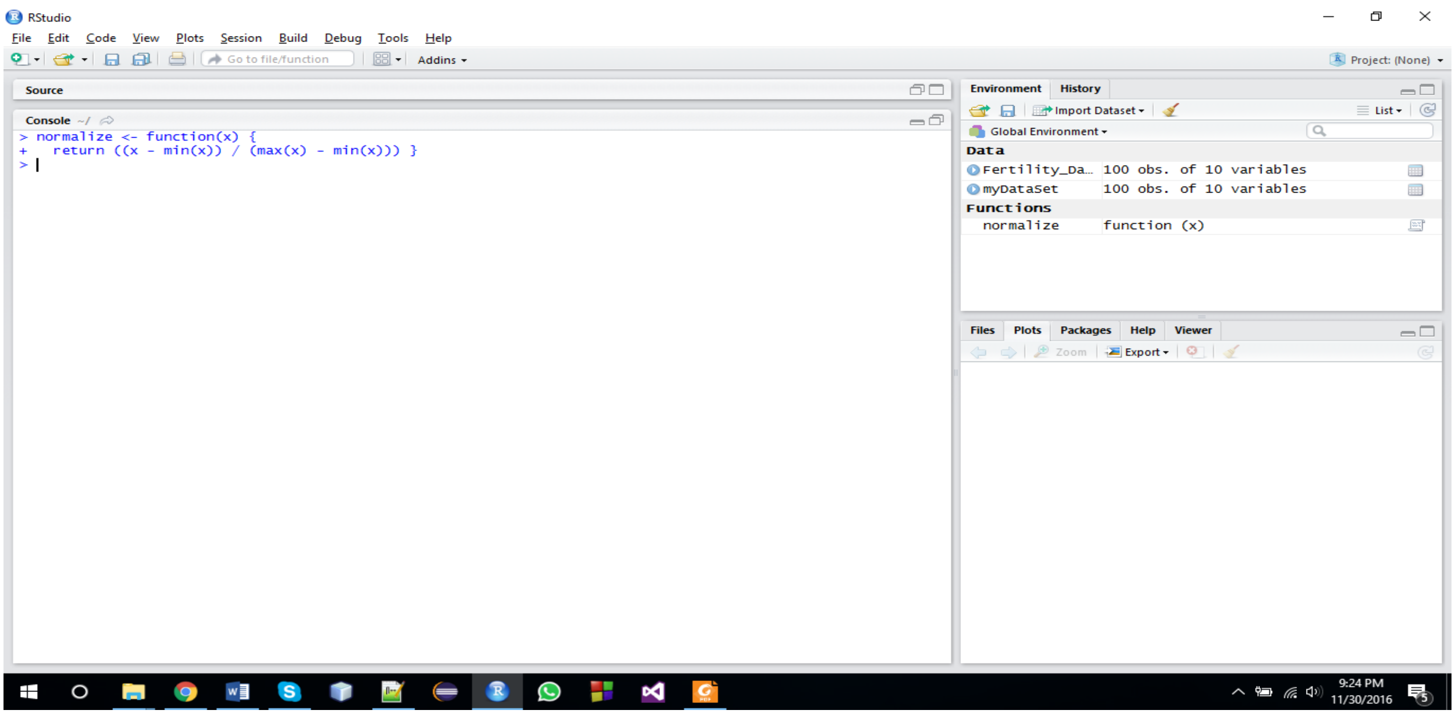
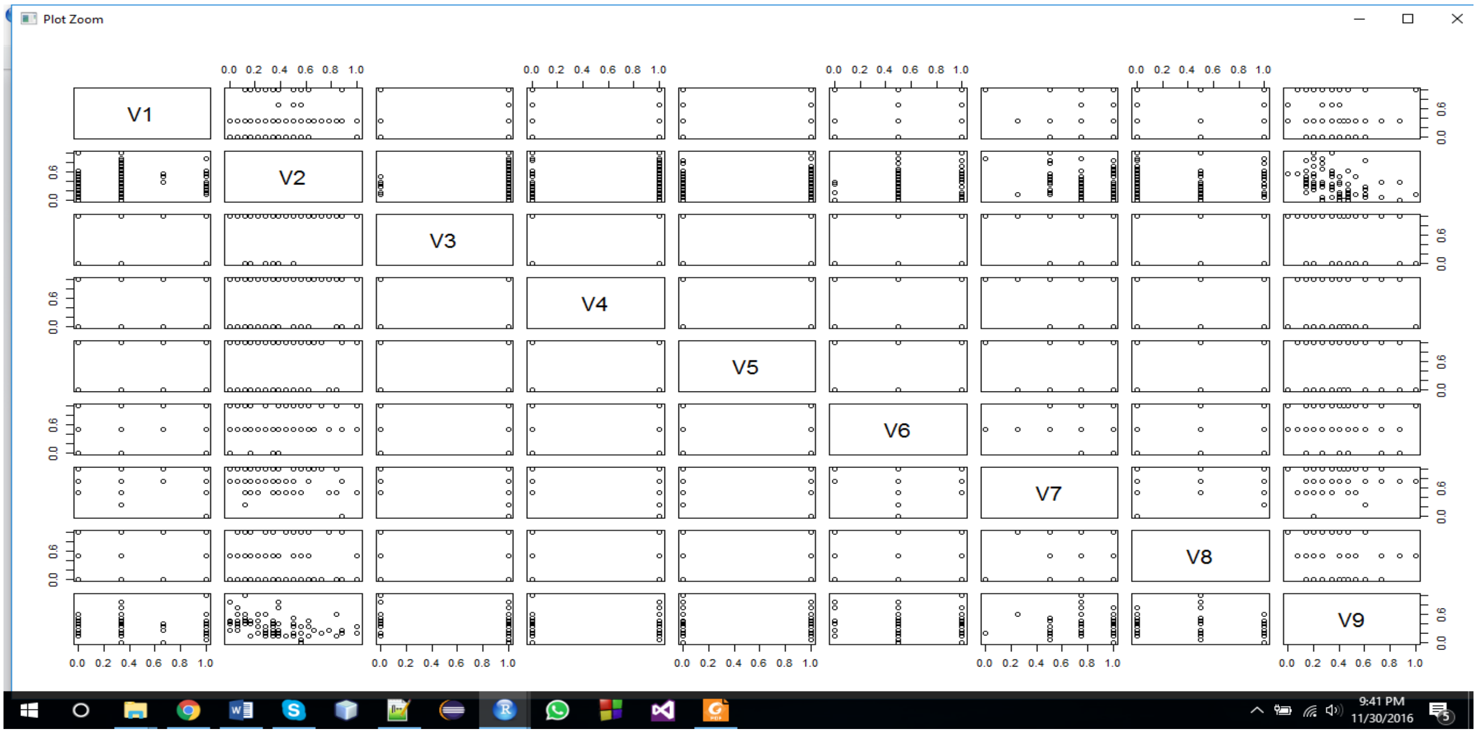
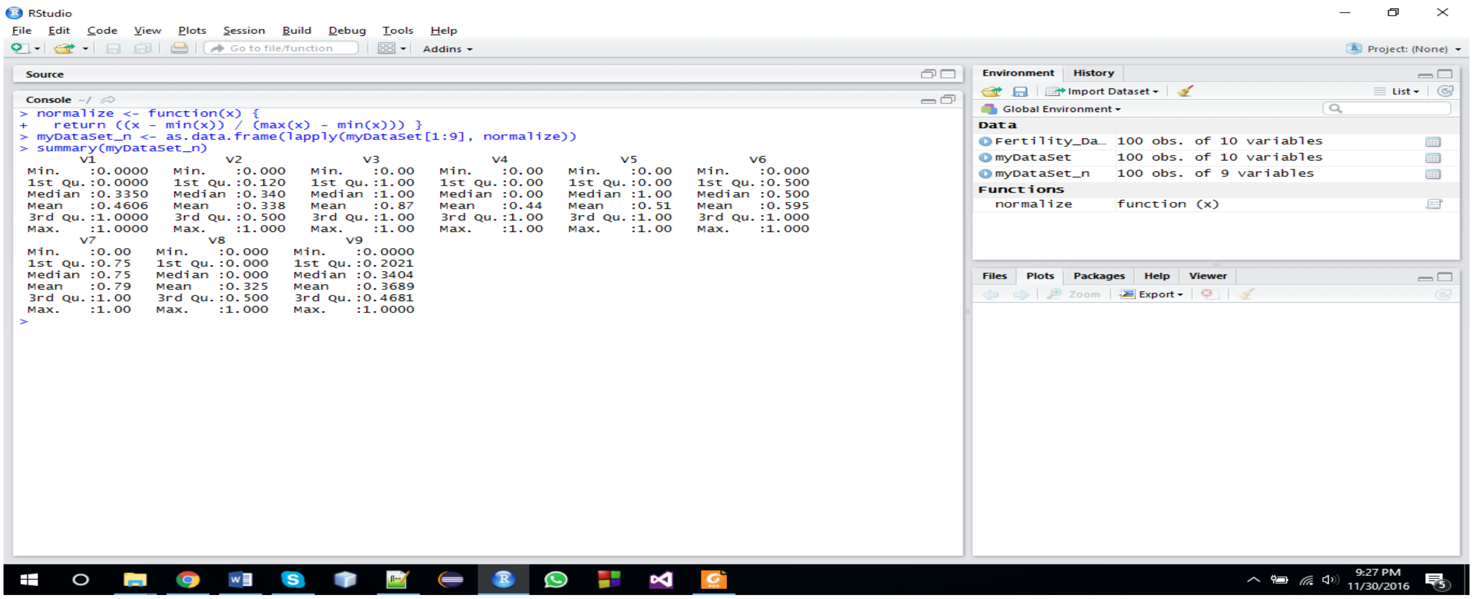


Figure 5: Plotting the output of normalized variables



This is a function, which normalizes the input value between 0 and 1. Then returns it to the caller
Now using this statement ‘myDataSet_n <- as.data.frame(lapply(myDataSet[1:9], normalize))’ we This will create a table with the normalized value from the attributes 1 to 9, Since 10 is a string and the deciding factor

Figure 6: Output of the summary of the normalized values



trainData <- myDataSet_n[1:81,] : This will be used to train the data on 1-81 rows

Figure 7: Output of the Training dataset

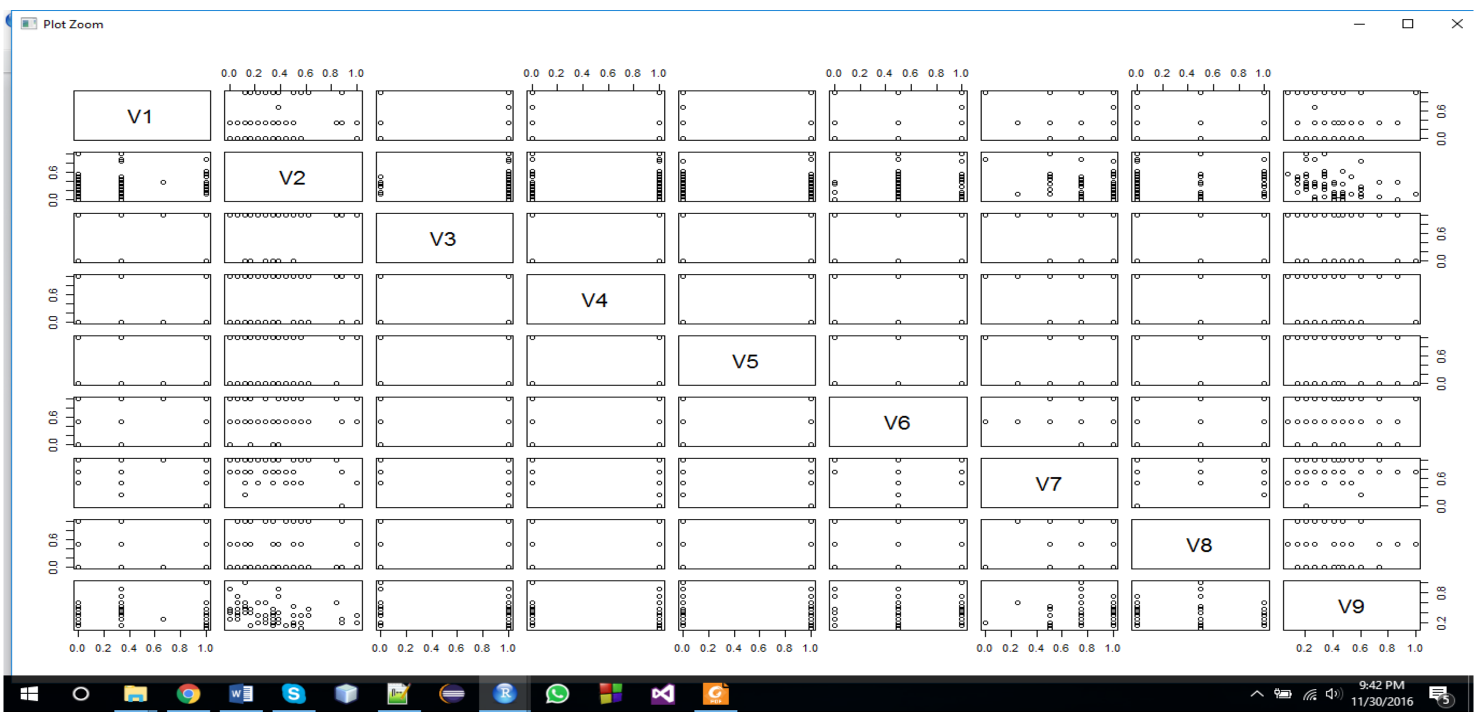
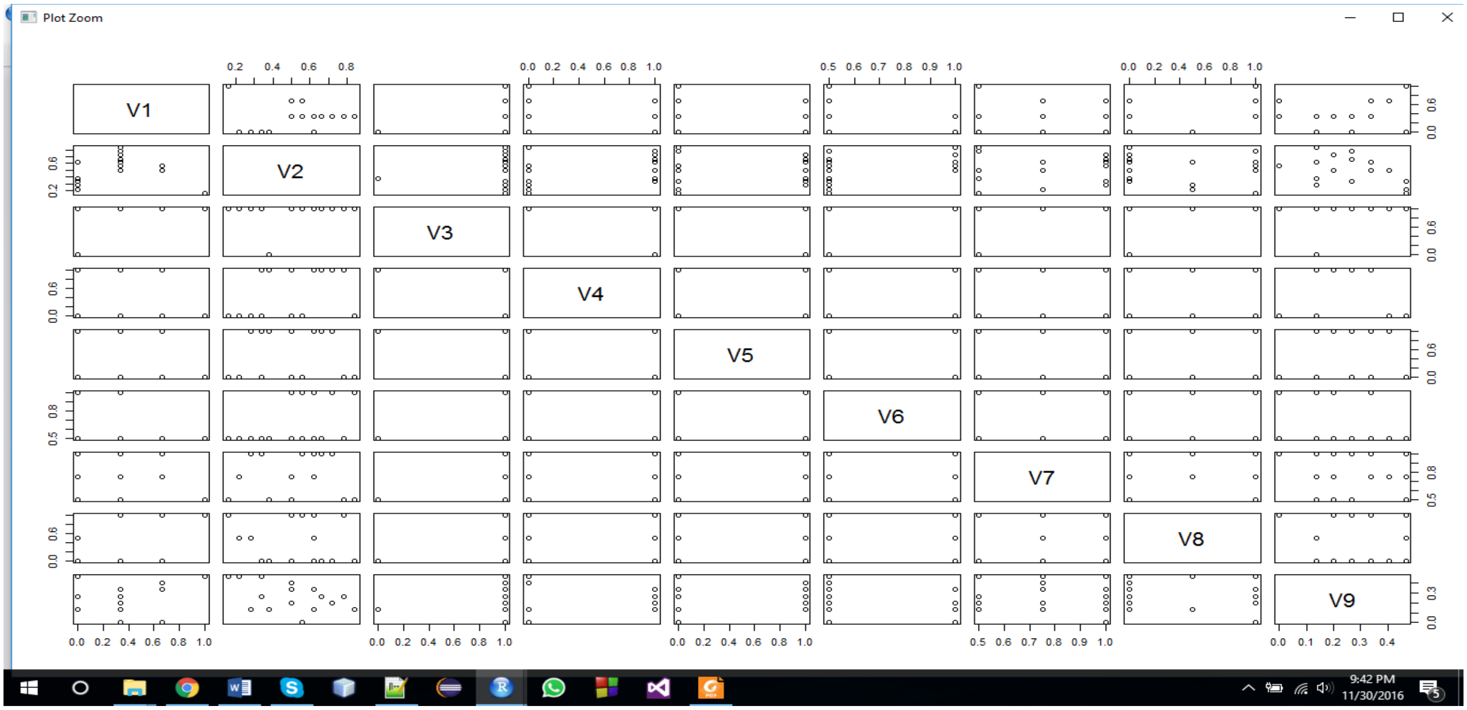


Figure 8: Output of the Testing dataset

testData <- myDataSet_n[82:100,] : Test data will hold the remaining rows, 19



new_train_labels <- myDataSet[1:81,1]   : We create a training class lable, which will be used as a class argument in the knn function

Figure 9: We can now see that we have 5 environmental variables data initialized, one value and one function

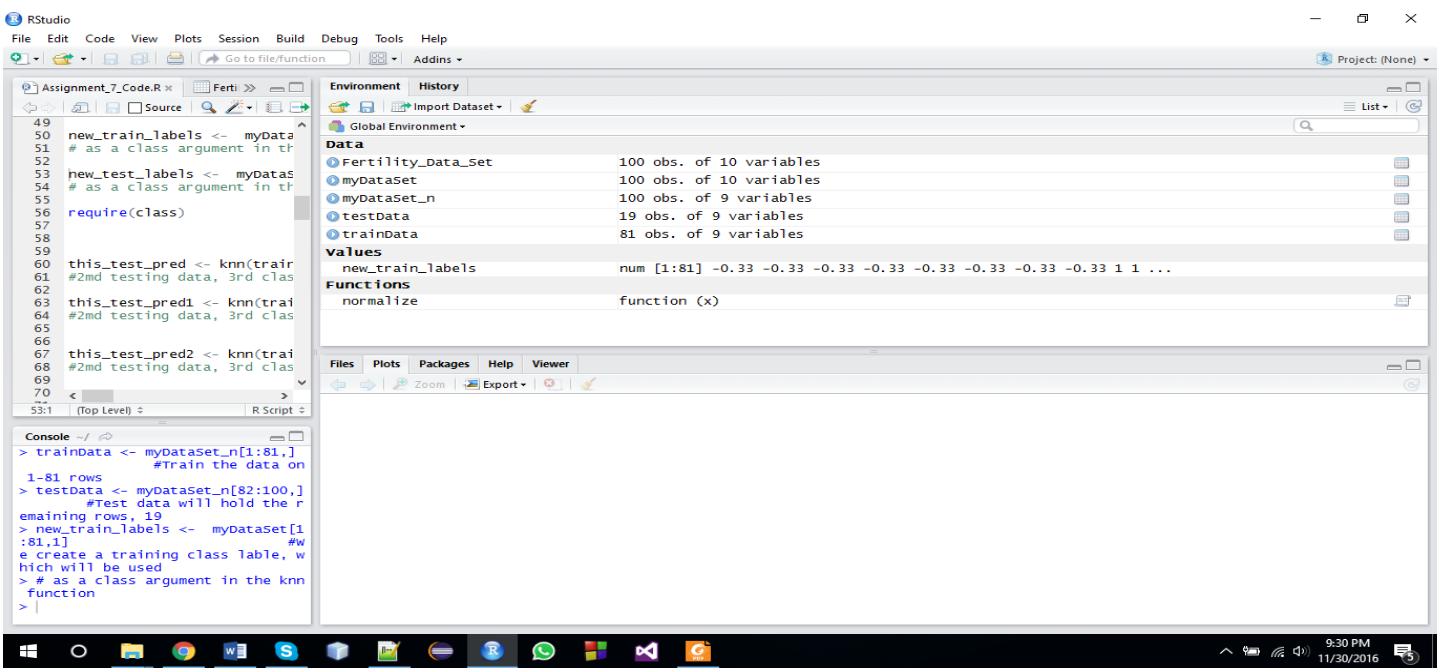


Figure 10: Now we print this using the confusion matrix

We now use the knn function from the class, ‘class’ : this_test_pred <- knn(trainData, testData , new_train_labels ,10)
It does the prediction, has 4 arguments. 1st = training data,
2md testing data, 3rd class, 4th = number of neighbours considered, which is 10 in this case.

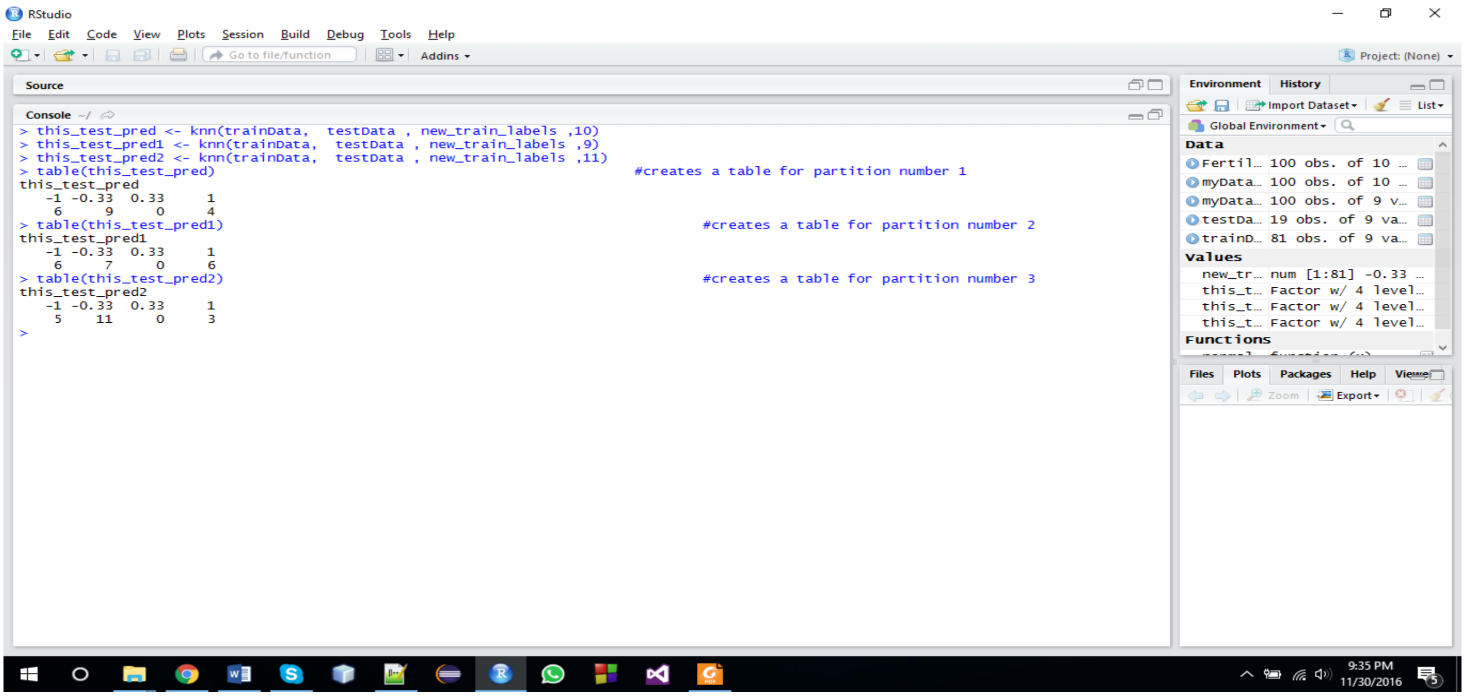


Figure 11: Prediction for which we consider 10 nearest neighbors(k=10)

Figure 12: Prediction for which we consider 9 neighbors(k=9)

Figure 13: Prediction for which we consider 11 neighbors(k=11)

Step 4: SVM Classifier Implementation

For this classifier, we use the above dataset.

We initialize 2 variables:
x <- subset(myDataSet, select=-V10): It creates a subset of the data without the 10th attribute in it
y <- V10 : it holds the 10th attribute data

We import the e1071 library using the following command, library(e1071)
We now use the svm function in the library e1071 and store the result in the variable svm_model
svm_model <- svm(V10 ~ ., data=myDataSet): 1st is the decision attribute argument and 2nd is the dataset

Now we use the summary command and output the result stored in svm_model
Notice that we have 2 classes here: N and O.
And the number of initial support vectors are 45: 33 and 12 respectively.

Figure 14: This is without the tuned version of svm_model

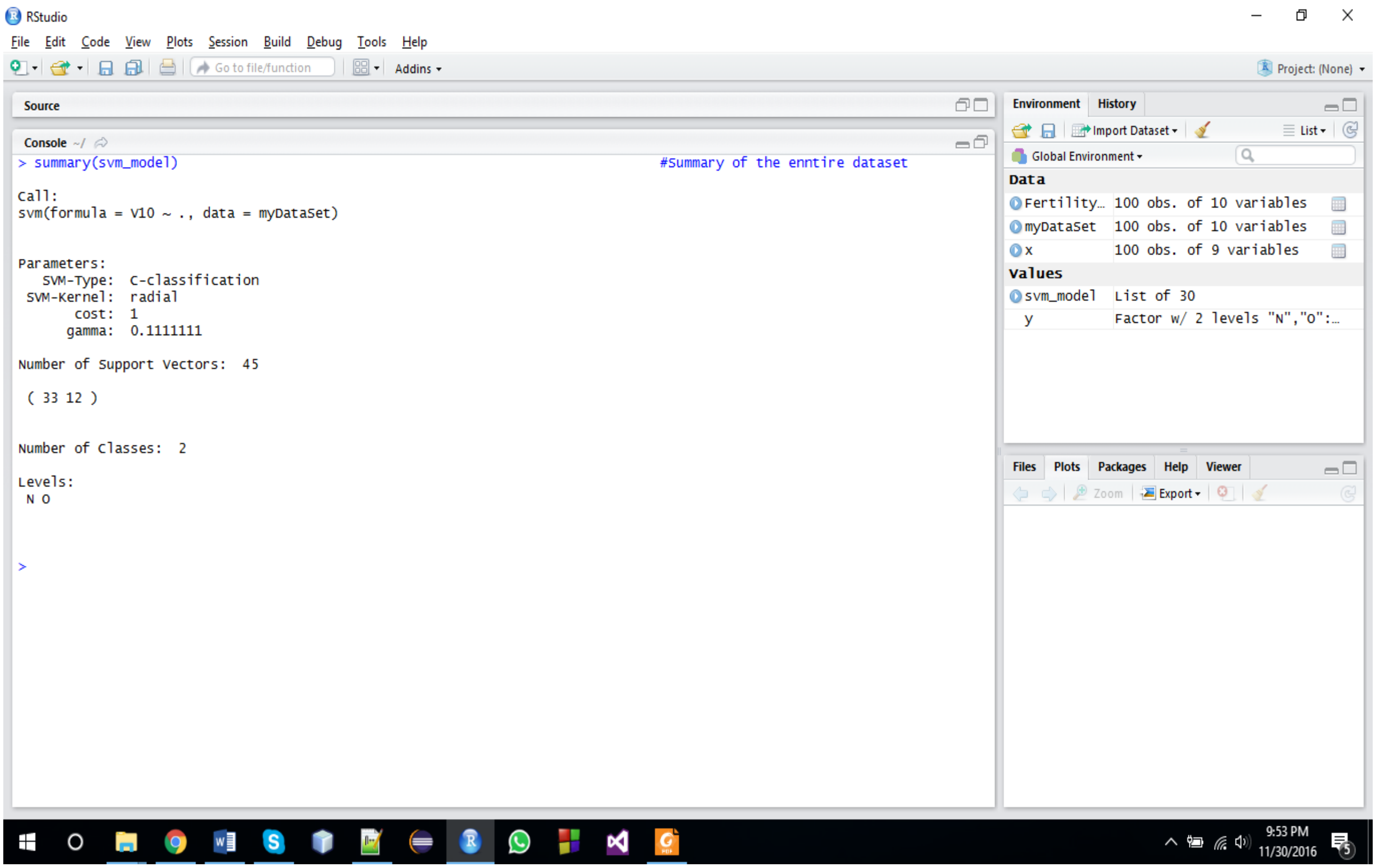


Figure 15: Now we predict our svm_model with the subset of the data(x).

And calculate how much time it takes to predict

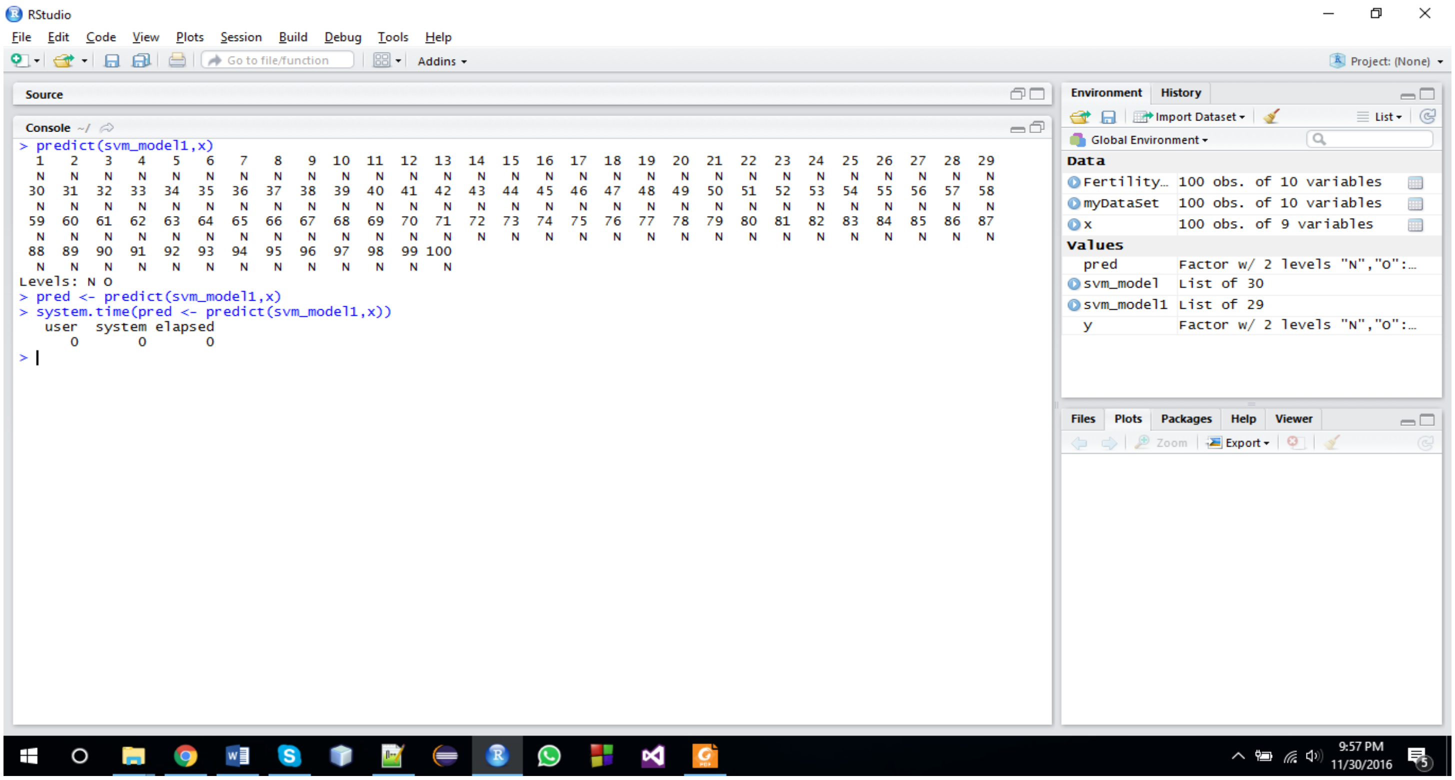


Figure 16: We now plot the svm model:

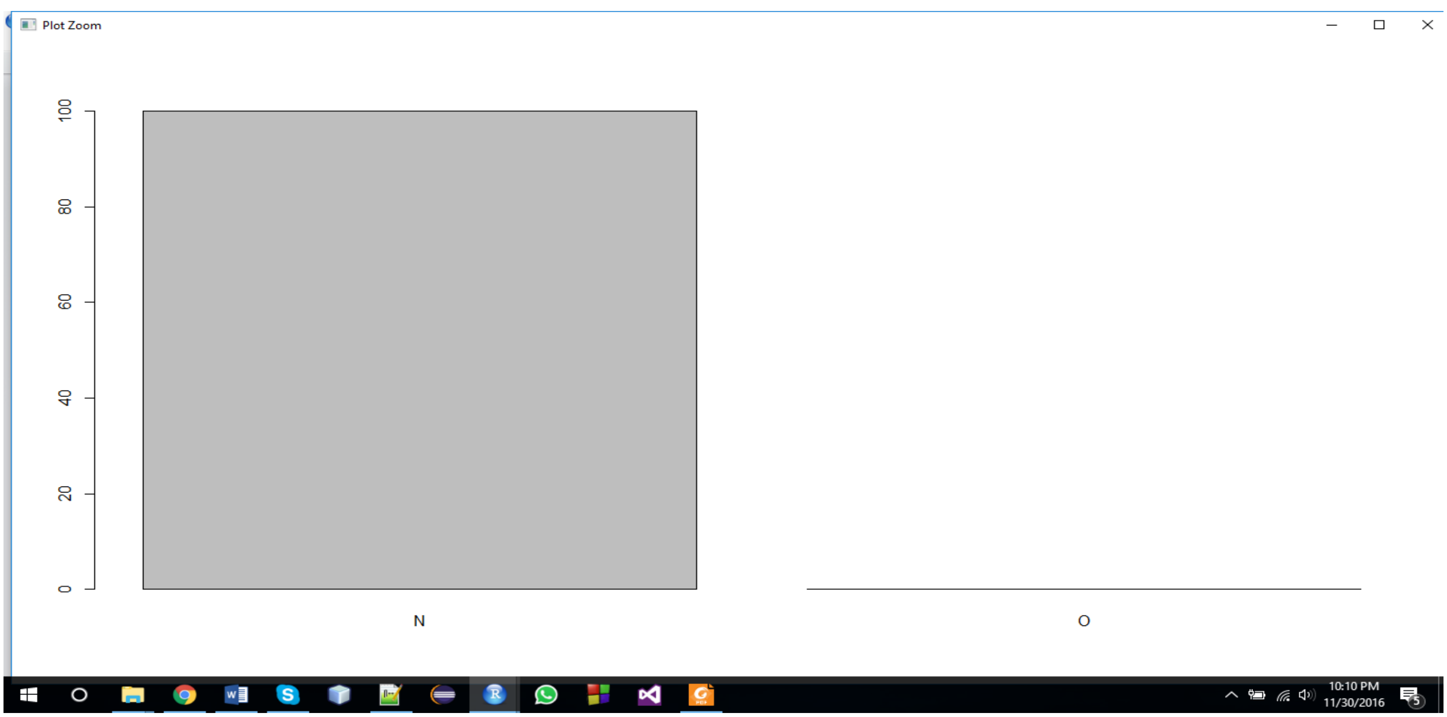


Figure 17: We display the initial confusion matrix without tuning the svm model:

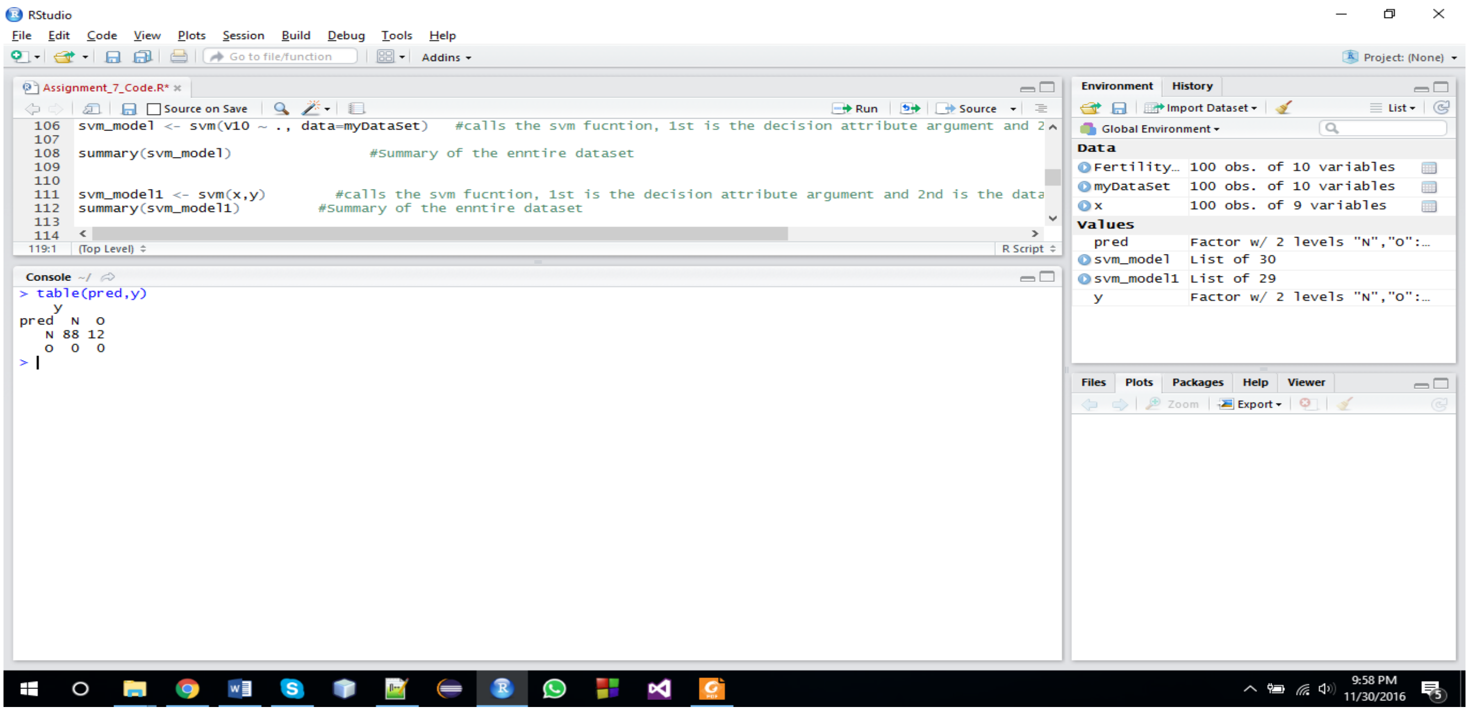


Figure 18: Now we tune our svm_model with the tune command and the print it:

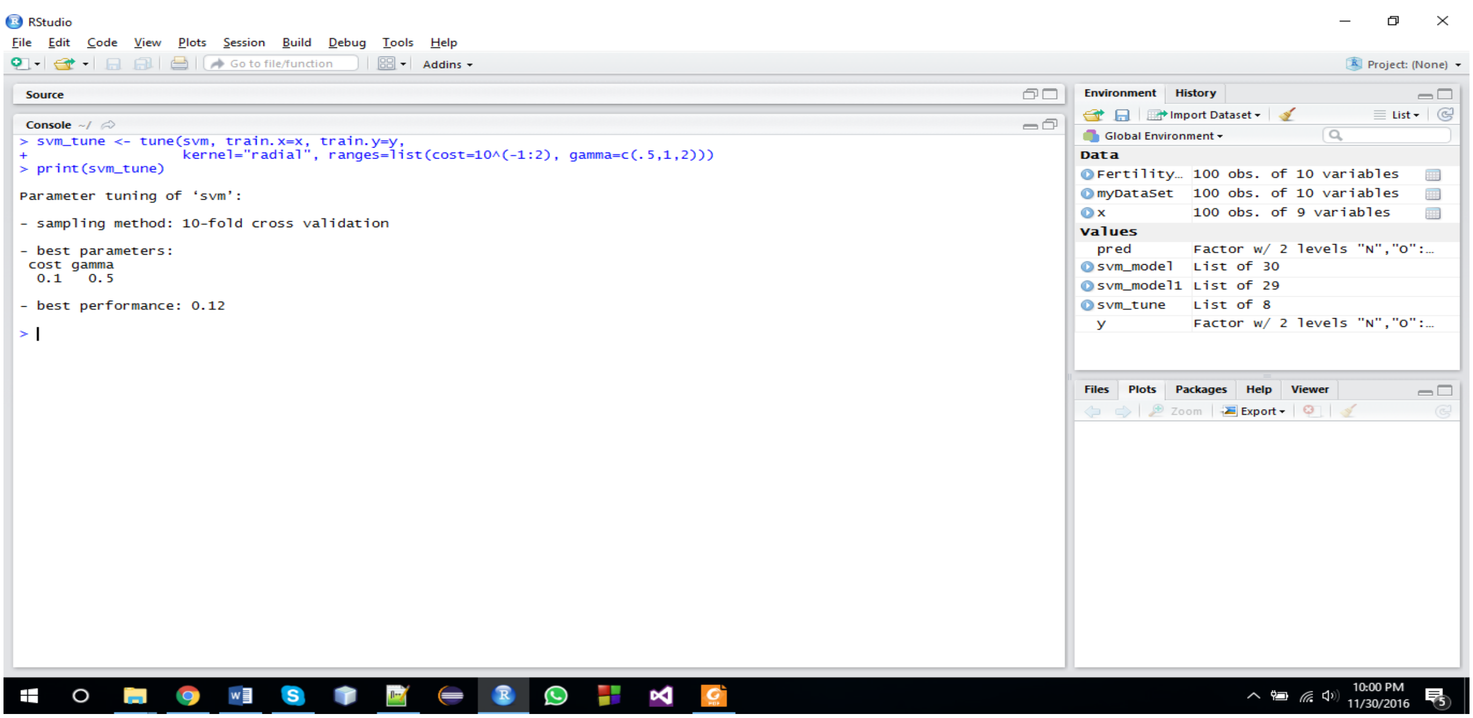
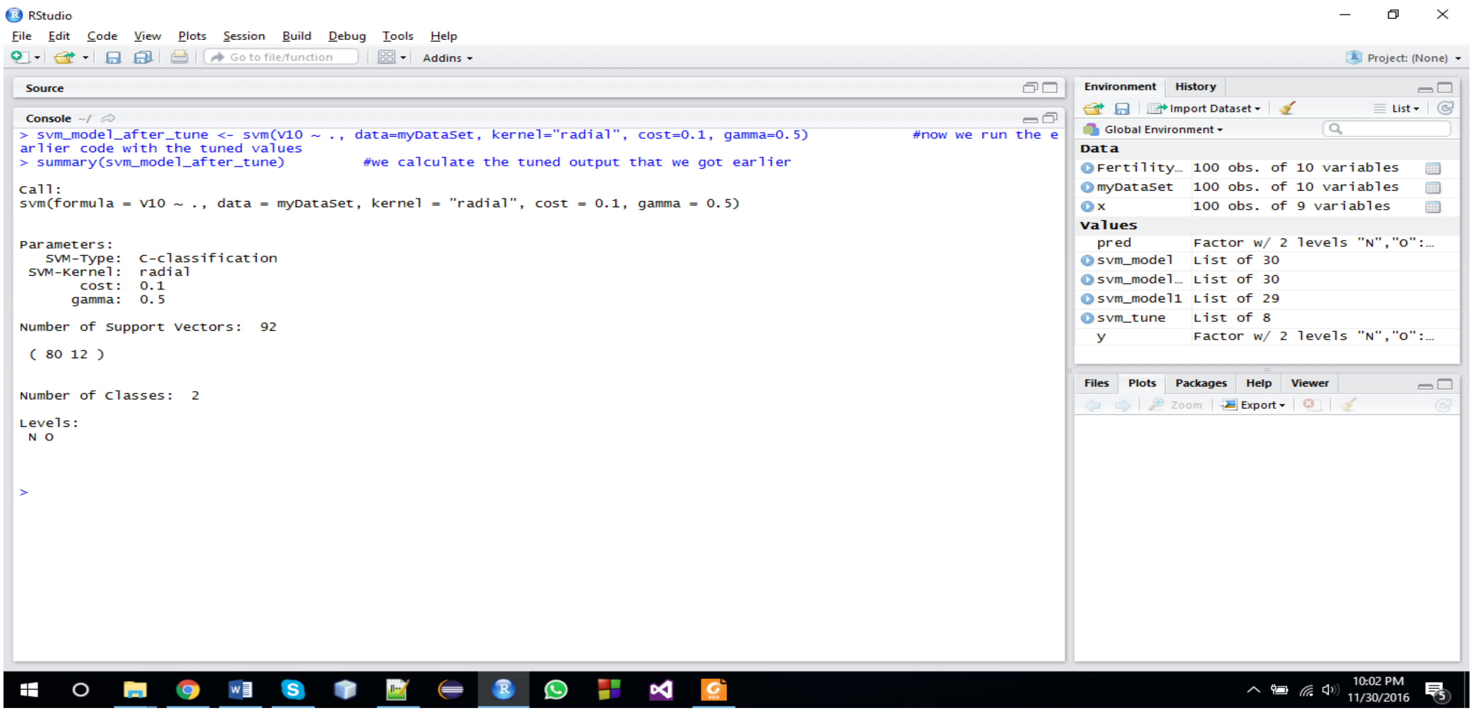


Figure 19: We get the cost and 0.1 and gamma as 0.5

We now use this to get our tuned version of svm model:



Notice that, we now have 92 support vectors: 80 and 12. Earlier we had 45. This is the tuned version of our svm_model.

Figure 20: We now predict the new value after tuning the svm model and print it. Also, we calculate the system time it took to compute this:

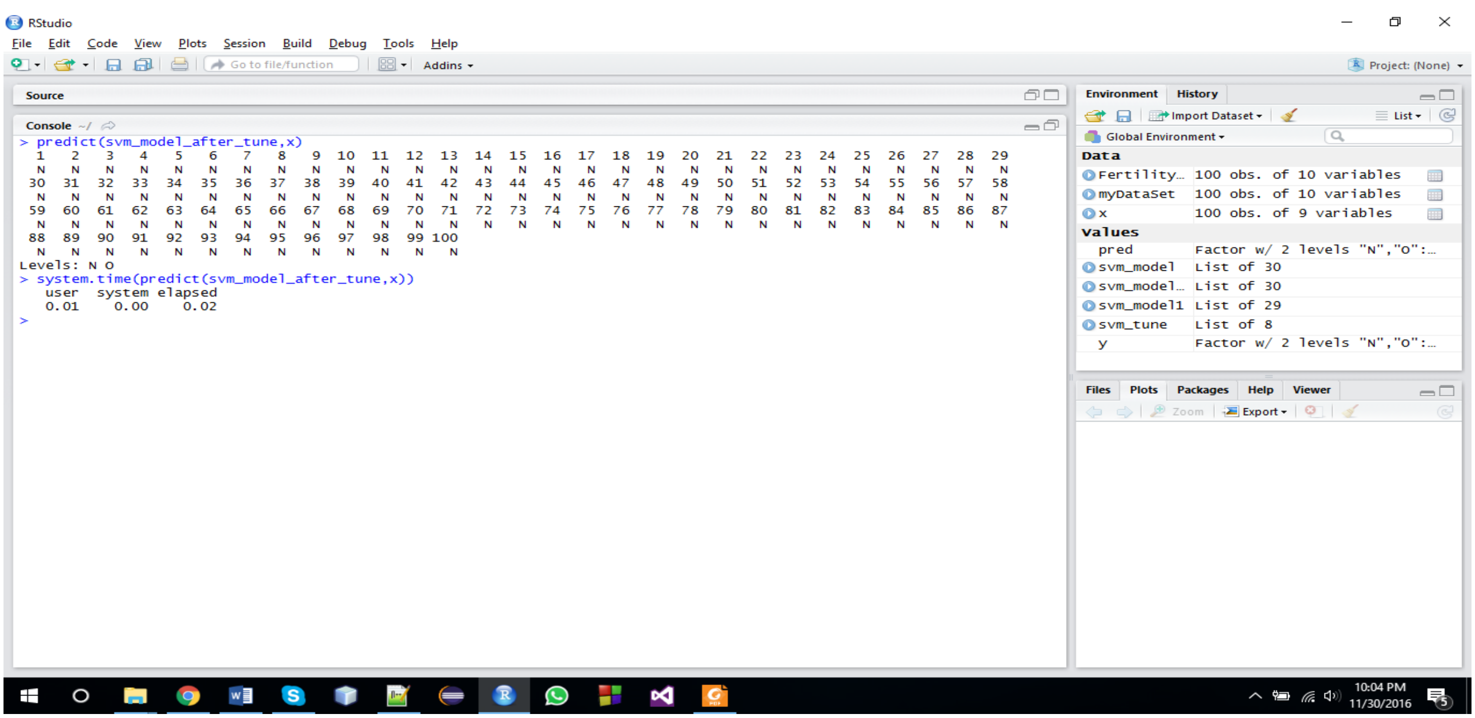
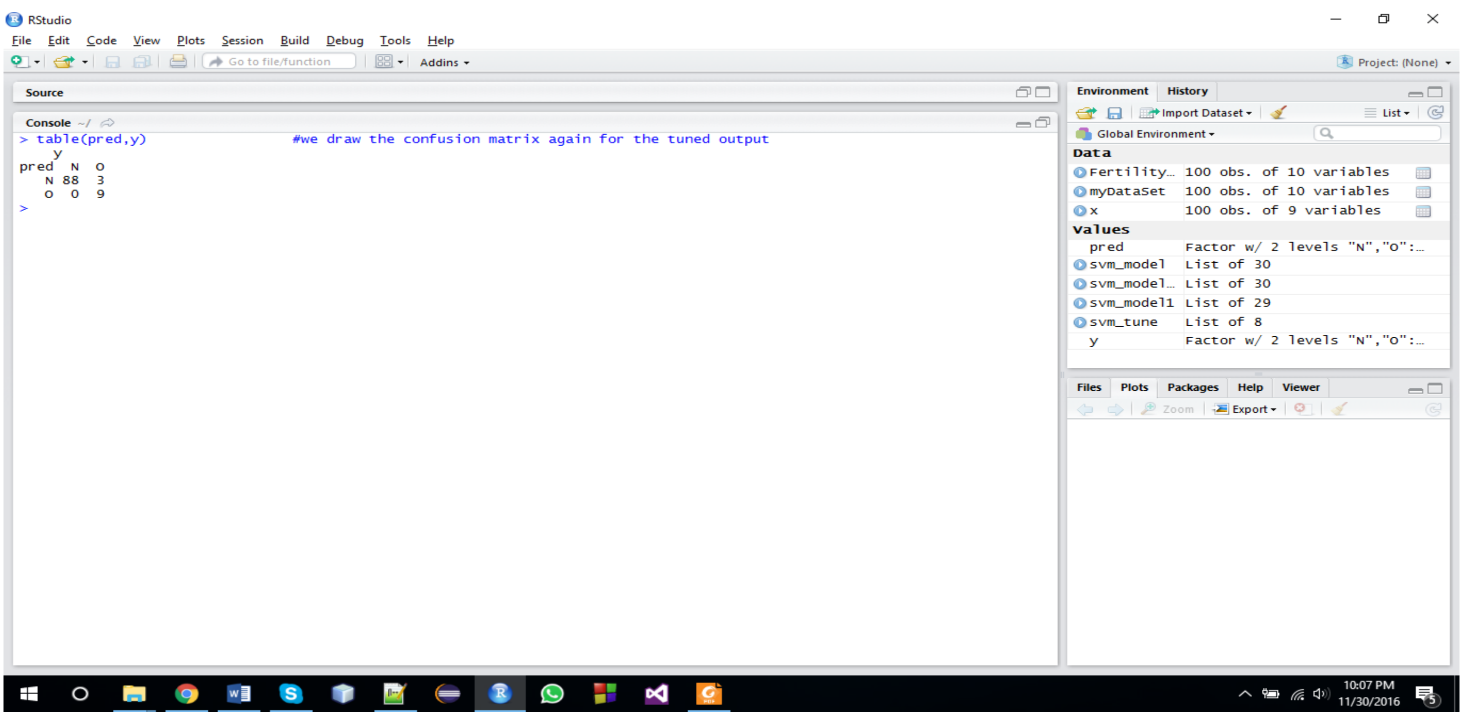


Figure 21: We now plot the result of the tuned svm model:

Figure 22: We now predict again with the tuned version and get the confusion matrix:



Notice that it is a different confusion matrix that the previous one

Step 5: Naive Bayes Classifier Implementation

For this classifier, we use the above dataset.

Figure 23: We distribute the data into two partitions randomly:

Training data of 75% and Testing data of 25%
We output the first 5 elements of testing data and training data
We also use nrow formula to calculate the number of rows in each dataset
Testing data:77 rows
Training data:23 rows

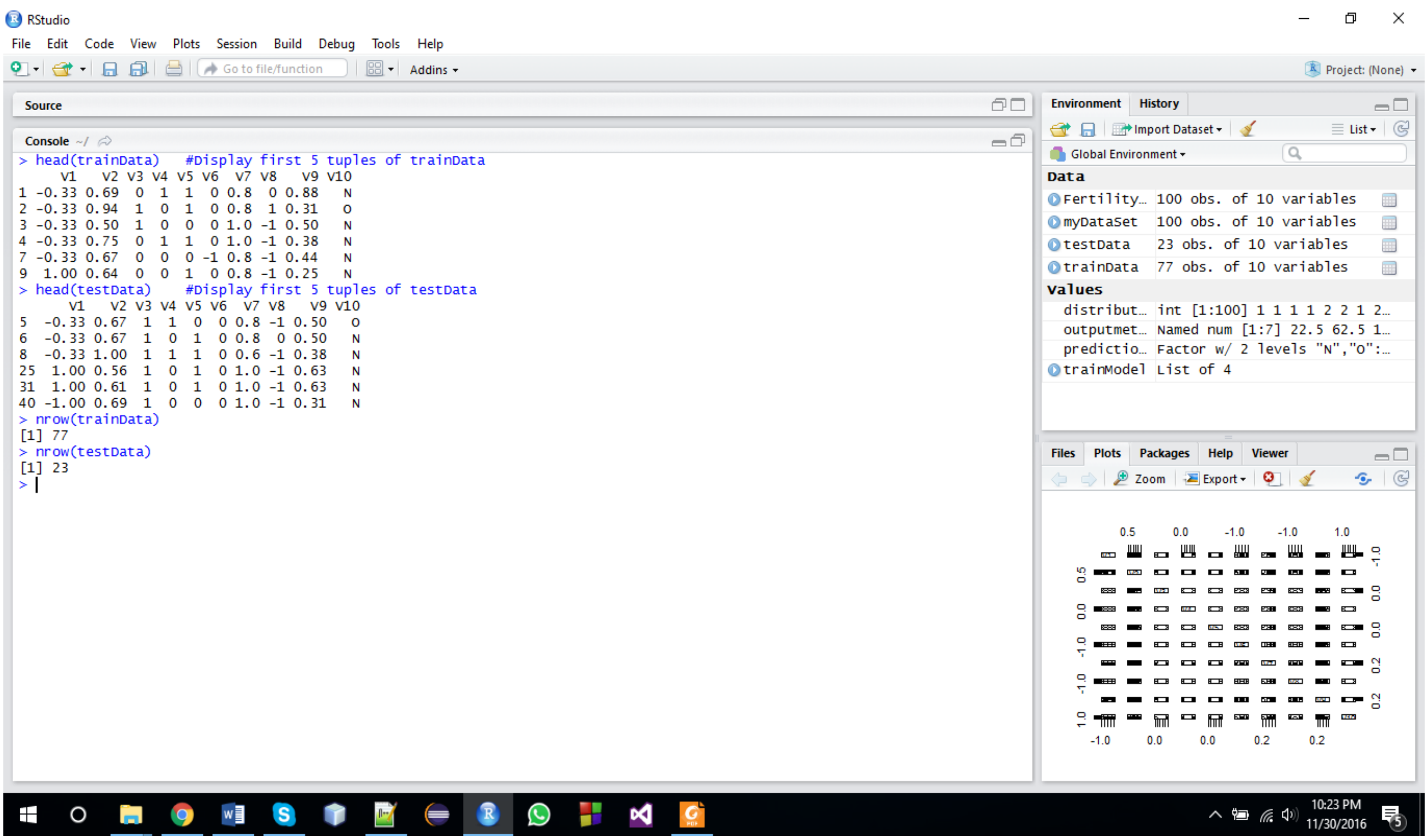


Figure 24: We plot the Testing data:

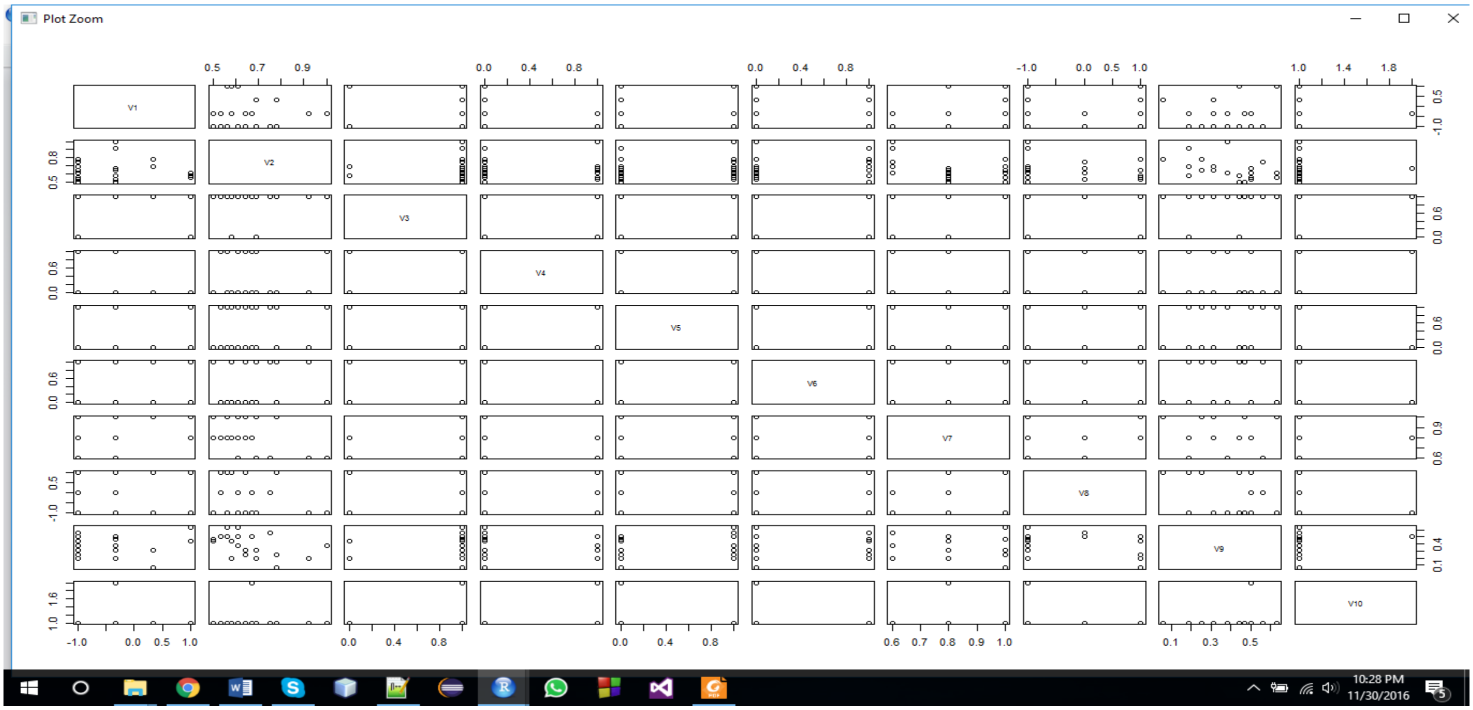


Figure 25: We plot the training data:

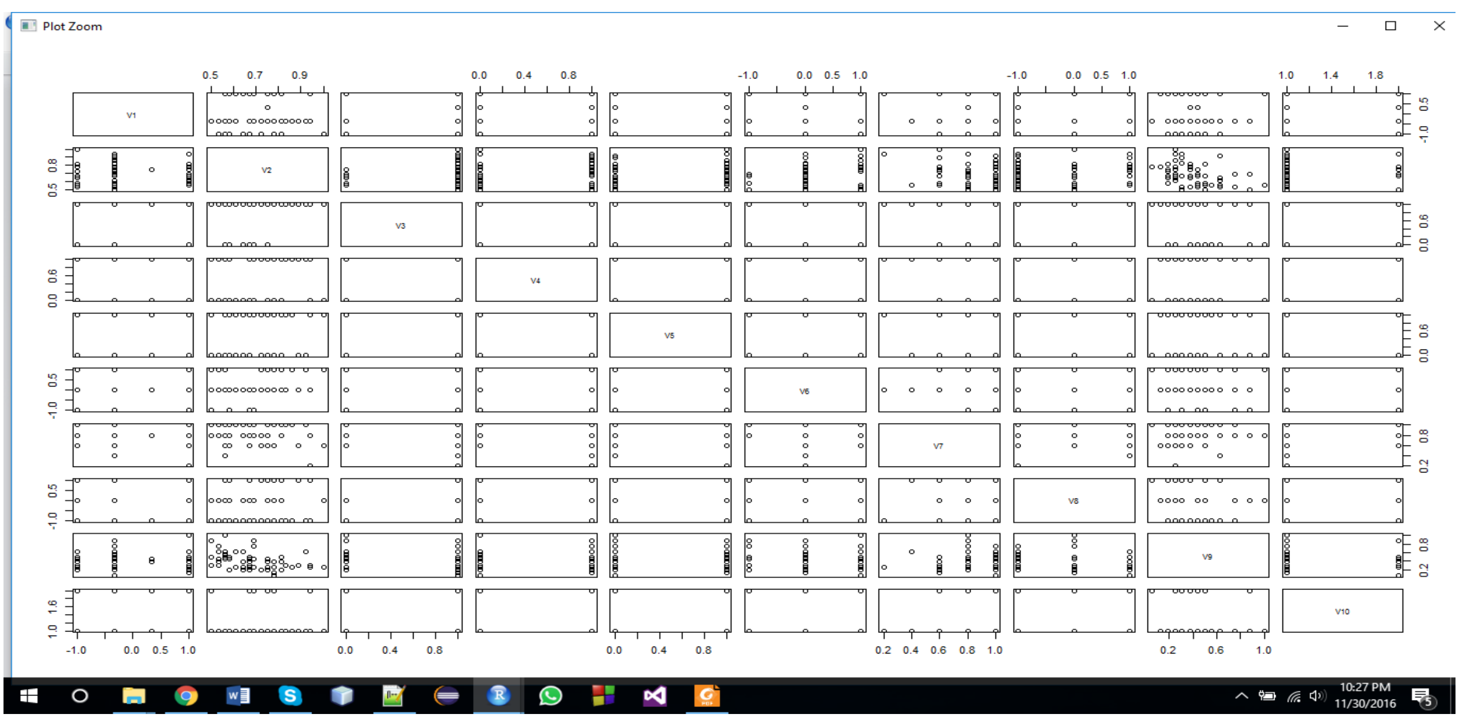
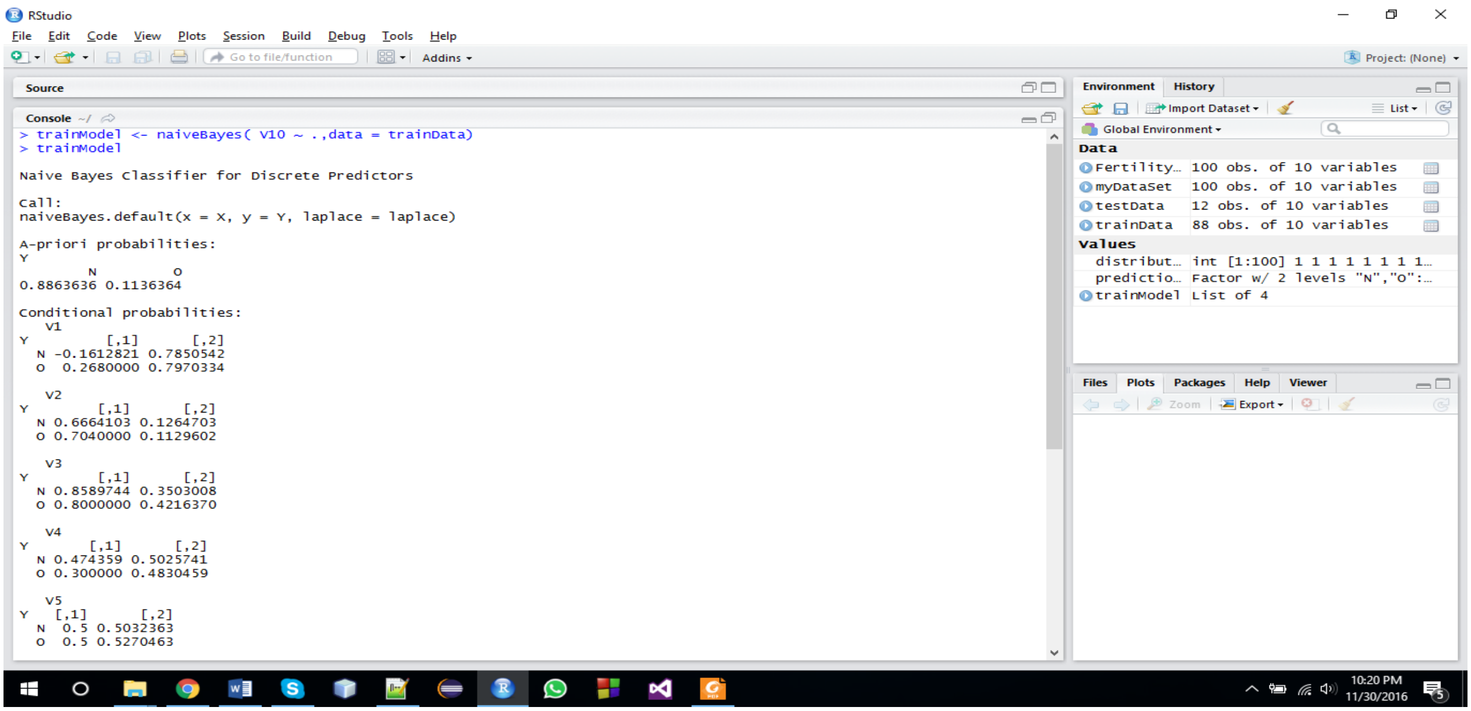


Figure 26 and 27: We now call the Naïve Bayes function and pass on the training data with the target variable as attribute 10 and display the output here:



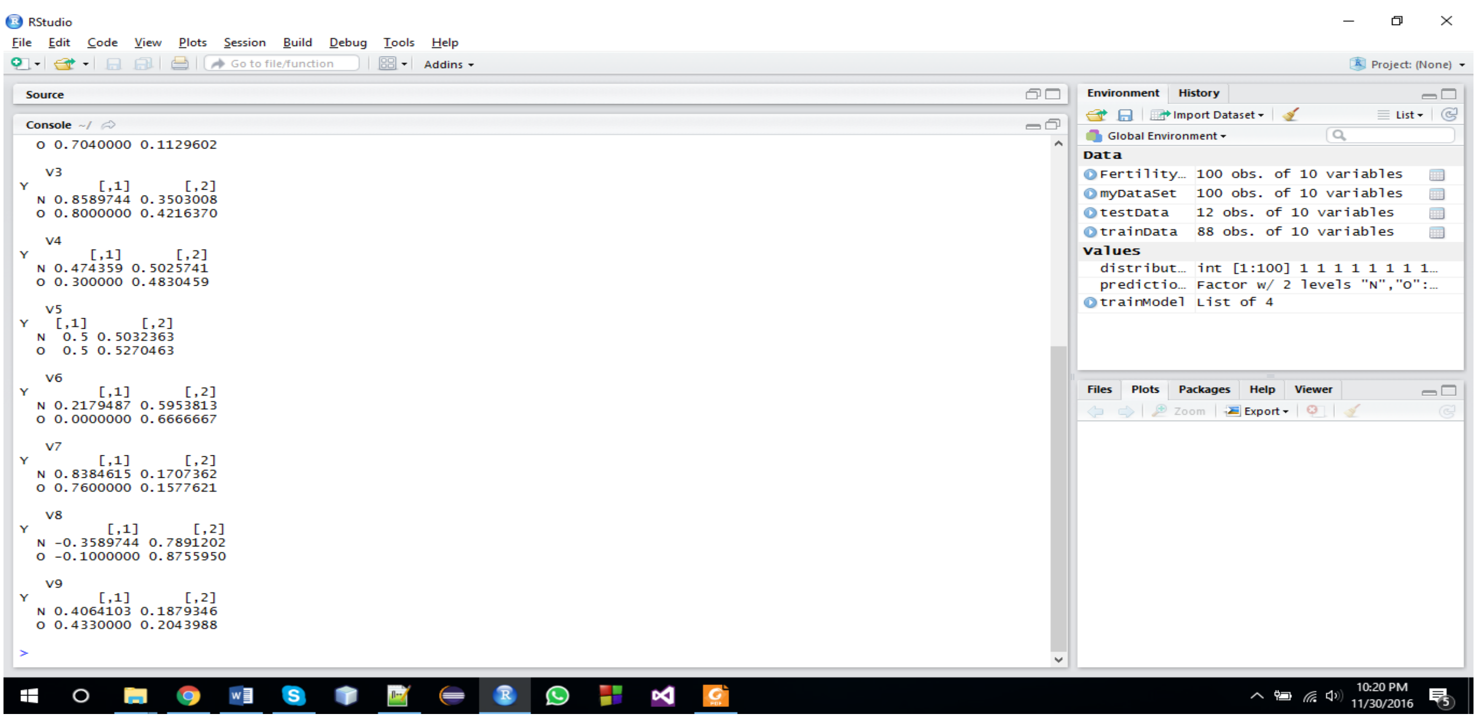
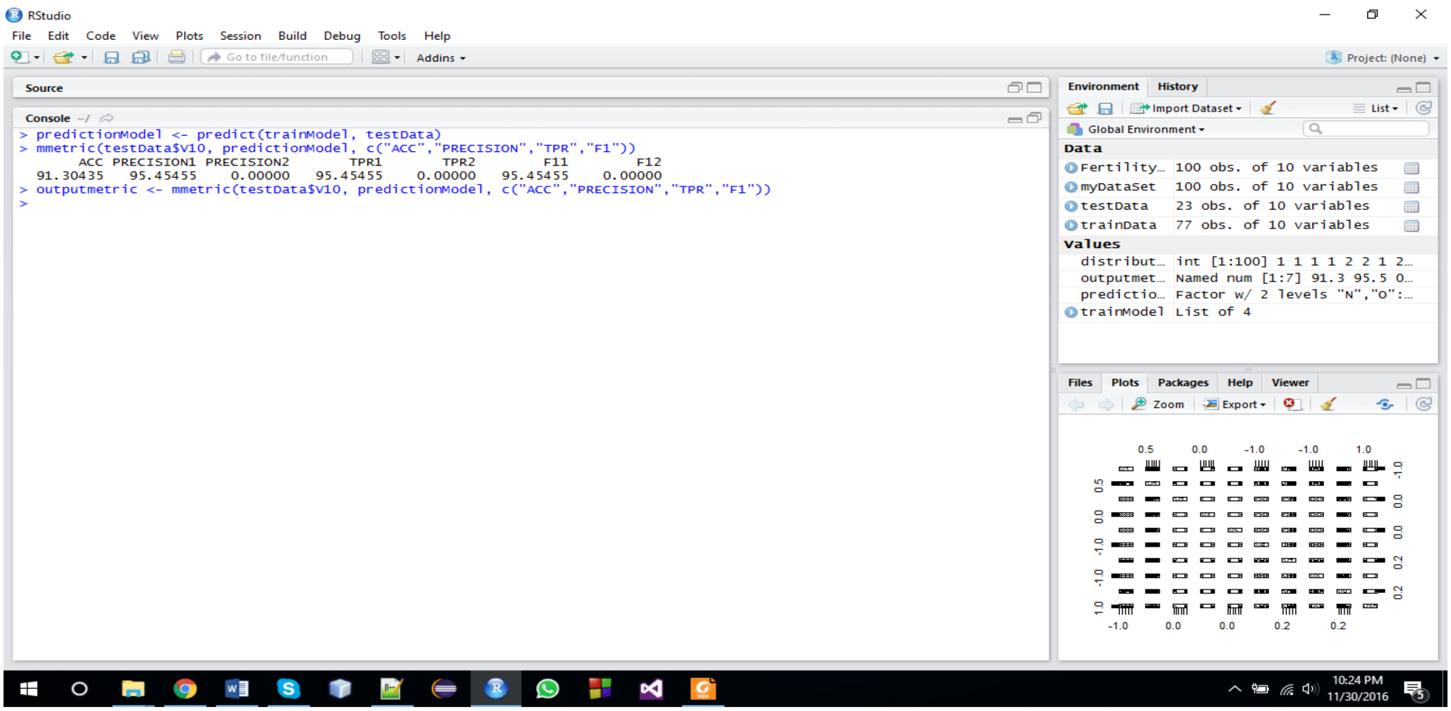


Figure 28: We now use the predict function and predict the value from the training model that we created and the testing data:



The figure shows:

ACC	PRECISION1	PRECISION2	TPR1	TPR2	F11	F12
91.30435	95.45455	0.00000	95.45455	0.00000	95.45455	0.00000

Here, accuracy is 91.30435, precision is 95.45455.

Figure 29: We now plot the prediction model:

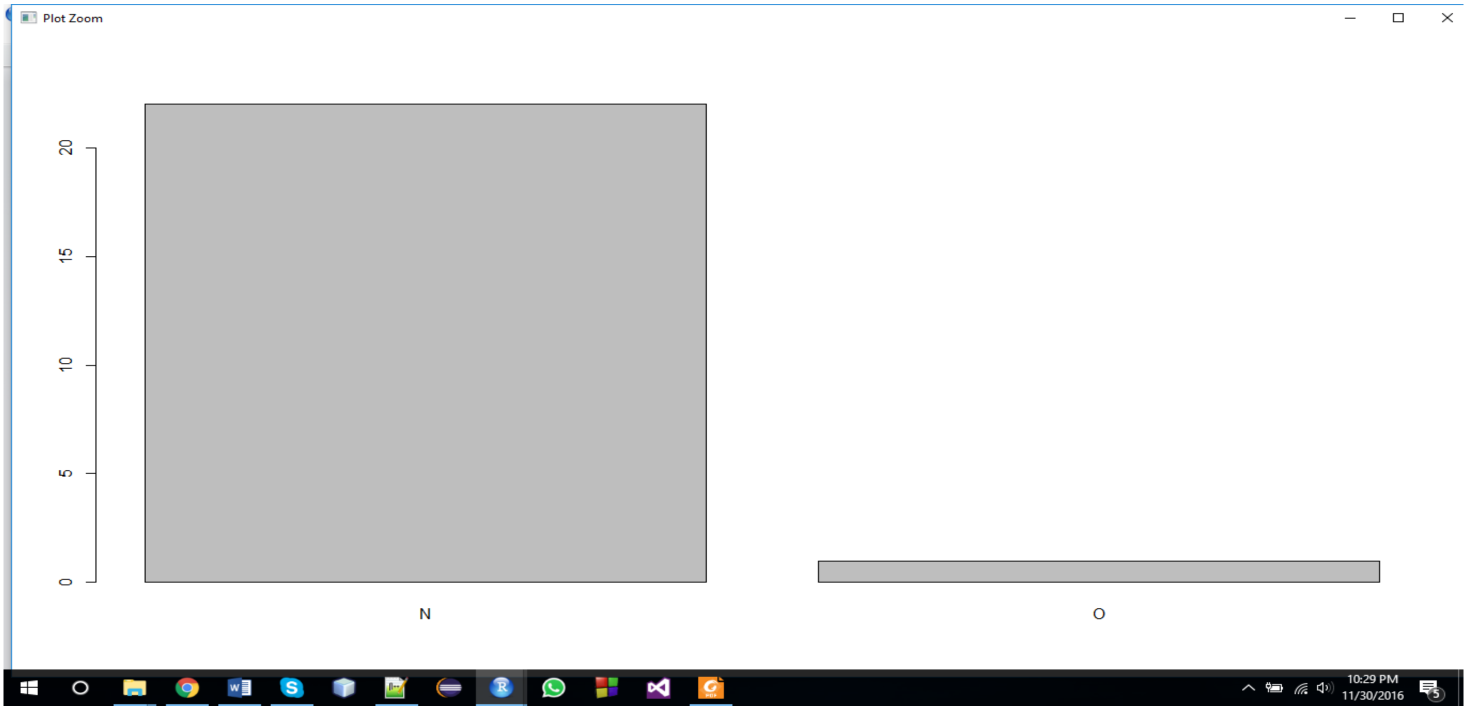
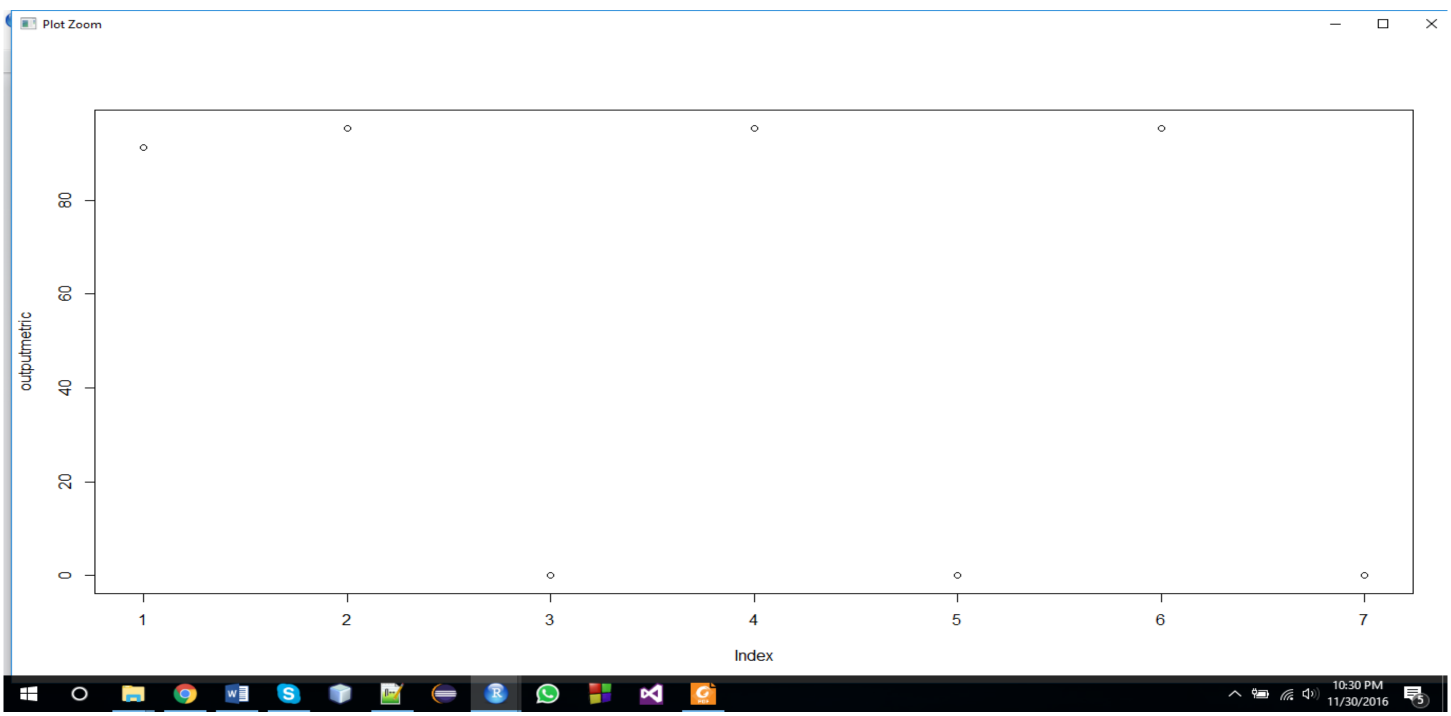


Figure 30: We now plot the output of the mmetric function:



REFERENCES:

[1] http://www.rdatamining.com/resources/data
[2] www.quora.com
[3] www.stackoverflow.com
[4] www.youtube.com
[5] www.Wikipedia.com
[6] www.cran.r-project.org