MA616 :Elements of Data Science Lab Project

Use LDA and Decision Tree to predict whether a customer will buy the caravan insurance policy

Keshav Kishore
2018eeb1158

Submission Date: 10/05/2021

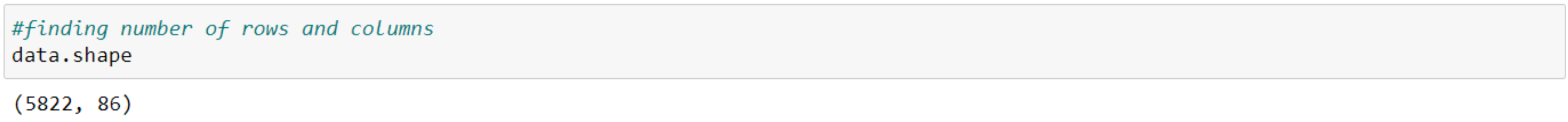
Objectives:

Do exploratory data analysis on the data. Use LDA and decision tree to predict whether a customer will buy a Caravan Insurance policy. Compare the findings from different methods

Components/Tools Required: Jupyter notebook, Gdocs

Exploratory Data Analysis:

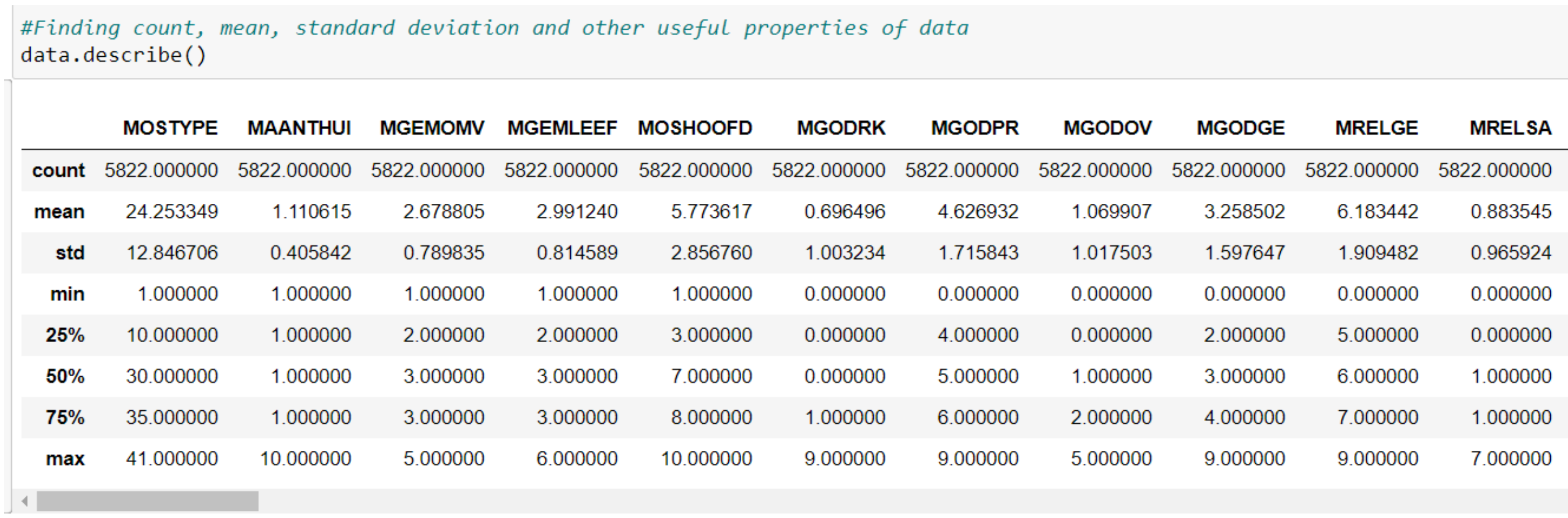
- There are 5822 rows and 86 columns.



```
#finding number of rows and columns
data.shape
```

```
(5822, 86)
```

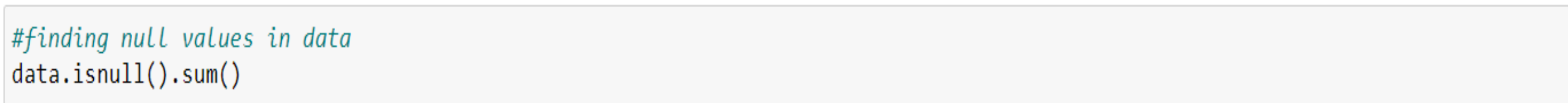
- From the below table we can see the mean, standard deviation, minimum and maximum values of each column. Also, we can see the first and third quartile(Q1 and Q3) of each column. It is useful in summarizing the given data.



```
#Finding count, mean, standard deviation and other useful properties of data
data.describe()
```

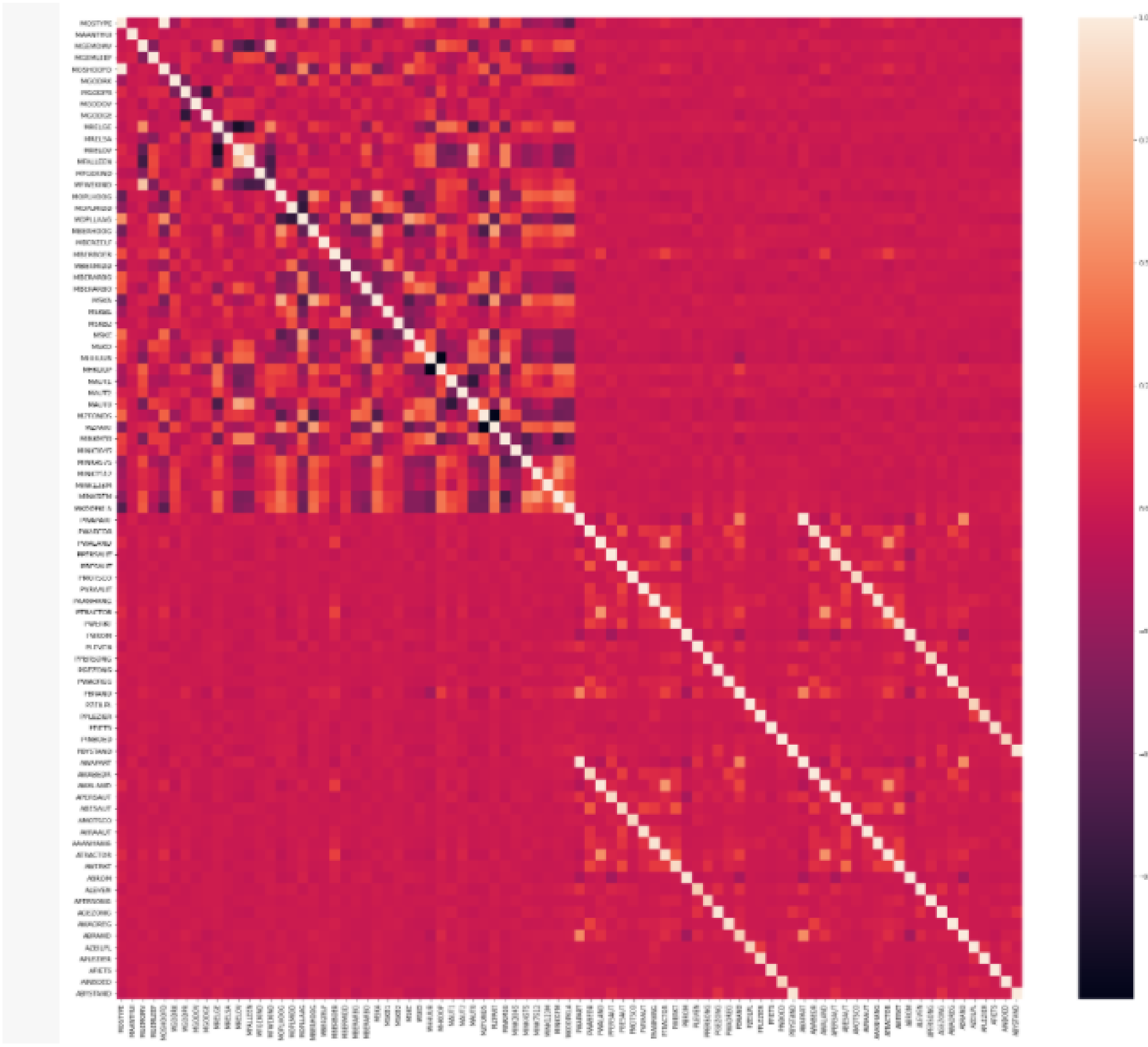
	MOSTYPE	MAANTHUI	MGEMOMV	MGEMLEEF	MOSHOOFD	MGODRK	MGODPR	MGODOV	MGODGE	MRELGE	MRELSA
count	5822.000000	5822.000000	5822.000000	5822.000000	5822.000000	5822.000000	5822.000000	5822.000000	5822.000000	5822.000000	5822.000000
mean	24.253349	1.110615	2.678805	2.991240	5.773617	0.696496	4.626932	1.069907	3.258502	6.183442	0.883545
std	12.846706	0.405842	0.789835	0.814589	2.856760	1.003234	1.715843	1.017503	1.597647	1.909482	0.965924
min	1.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	10.000000	1.000000	2.000000	2.000000	3.000000	0.000000	4.000000	0.000000	2.000000	5.000000	0.000000
50%	30.000000	1.000000	3.000000	3.000000	7.000000	0.000000	5.000000	1.000000	3.000000	6.000000	1.000000
75%	35.000000	1.000000	3.000000	3.000000	8.000000	1.000000	6.000000	2.000000	4.000000	7.000000	1.000000
max	41.000000	10.000000	5.000000	6.000000	10.000000	9.000000	9.000000	5.000000	9.000000	9.000000	7.000000

- There are no null values in the dataset observed from the below command.



```
#finding null values in data
data.isnull().sum()
```

- Below is the correlation heatmap.



From the above heatmap, the features which were perfectly positively correlated(i.e having correlation 1) are:

PWAPART	AWAPART
PWABEDR	AWABEDR
PWALAND	AWALAND
PPERSAUT	APERSAUT
PBESAUT	ABESAUT
PMOTSCO	AMOTSCO
PVRAAUT	AVRAAUT
PAANHANG	AAANHANG
PTRACTOR	ATRACTOR
PWERKT	AWERKT
PBROM	ABROM
PLEVEN	ALEVEN
PPERSONG	APERSONG
PGEZONG	AGEZONG
PWAOREG	AWAOREG
PBRAND	ABRAND
PZEILPL	AZEILPL
PPLEZIER	APLEZIER
PFIETS	AFIETS
PINBOED	AINBOED
PBYSTAND	ABYSTAND

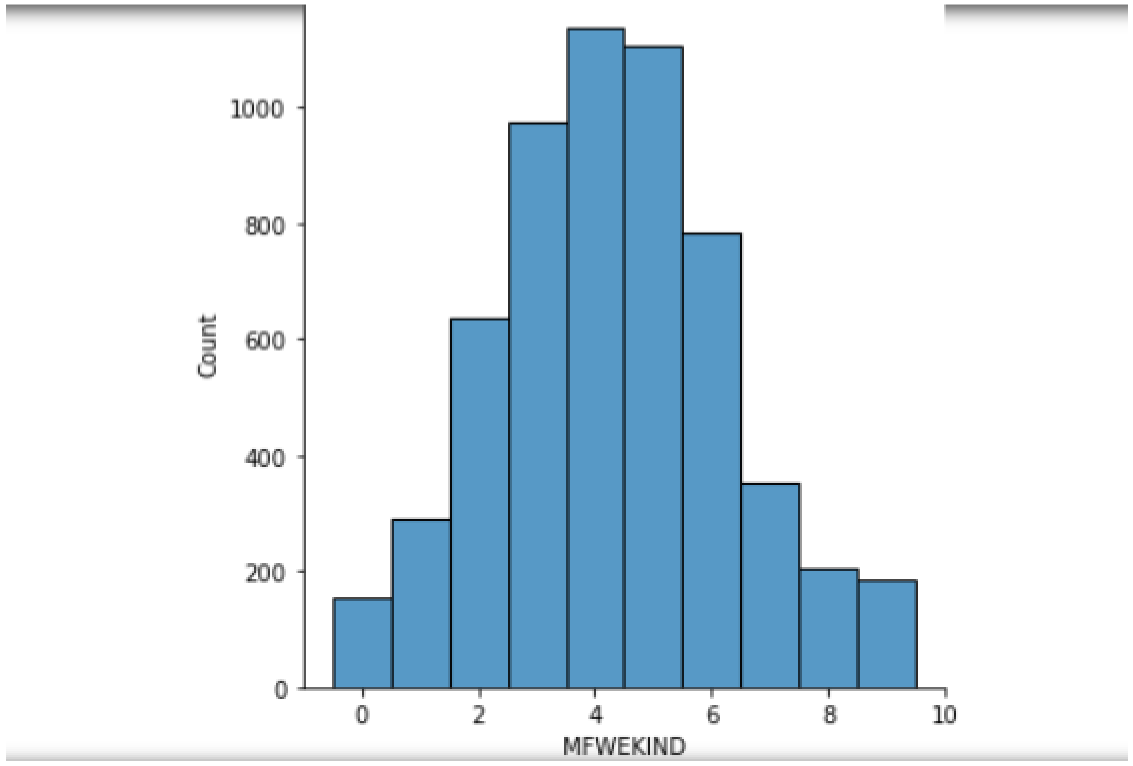
Columns having perfectly negative correlation(-1):

MHKOOP	MHHUUR
MZPART	MZFONDS

Analysis of input columns from histogram and box whisker plot.

MFWEKIND

Histogram :



From the histogram, we can observe that data is similar to normal distribution thus the mean will be closer to 5(actual - 4.3). Also, there should not be any outliers.

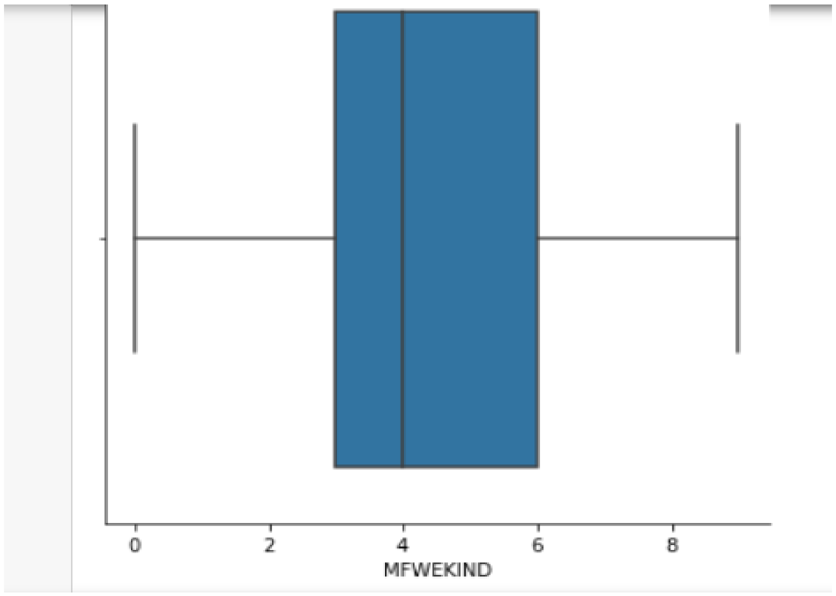
From data.describe() we got :

Q1 - 3

Q2 - 4

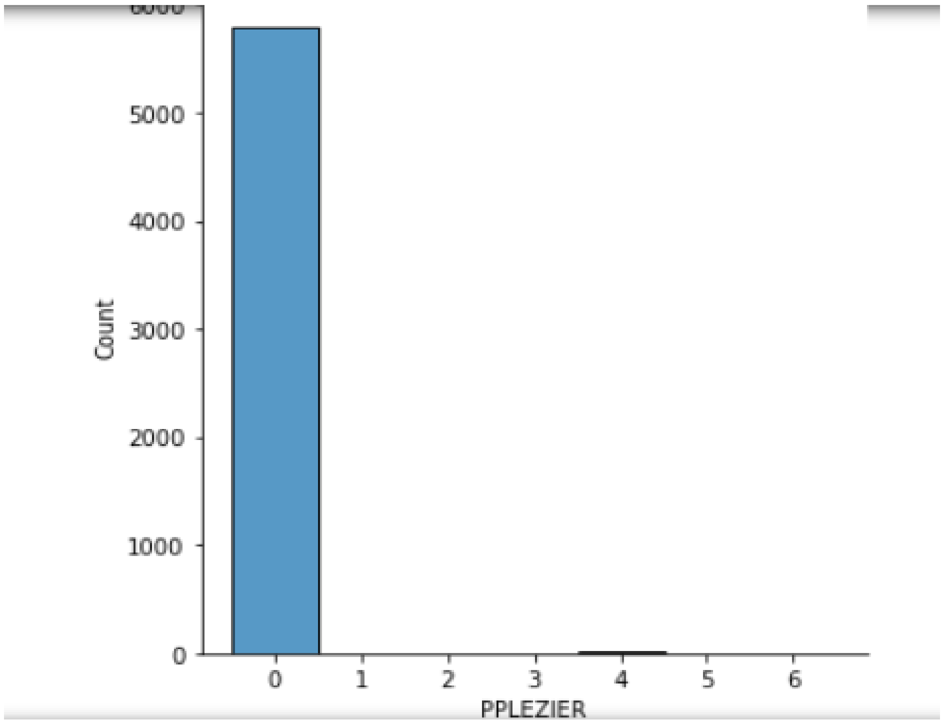
Q3 - 6

The same can be verified from below box plot.



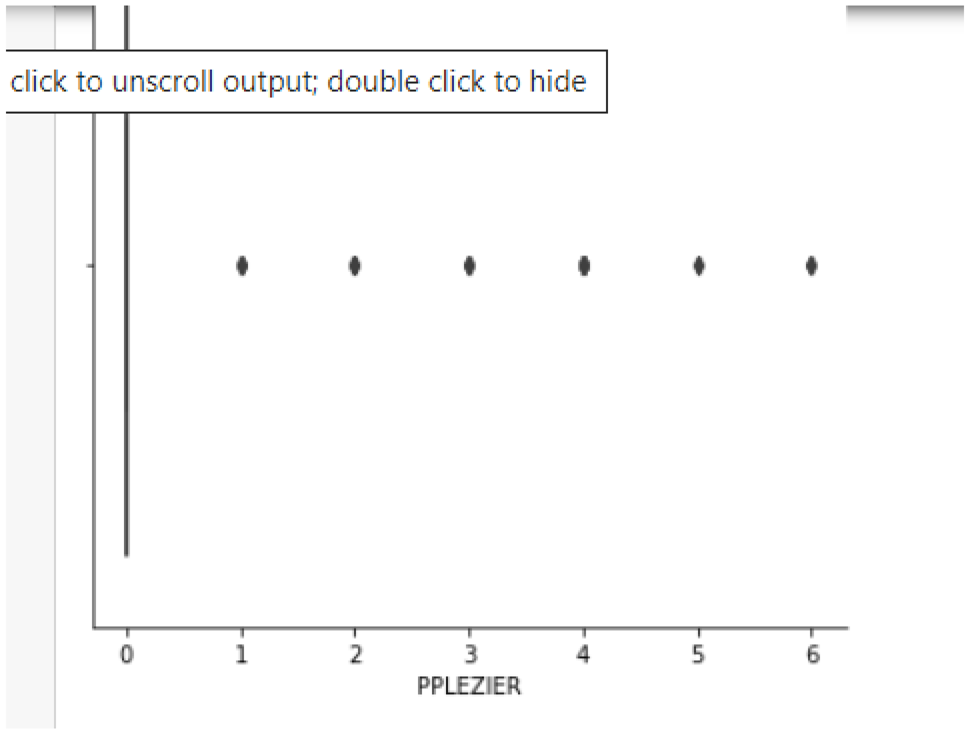
PPLEZIER

Histogram:



From the histogram, we can see that data is very unevenly distributed where more than 95% of data is 0. This column will definitely have outliers.

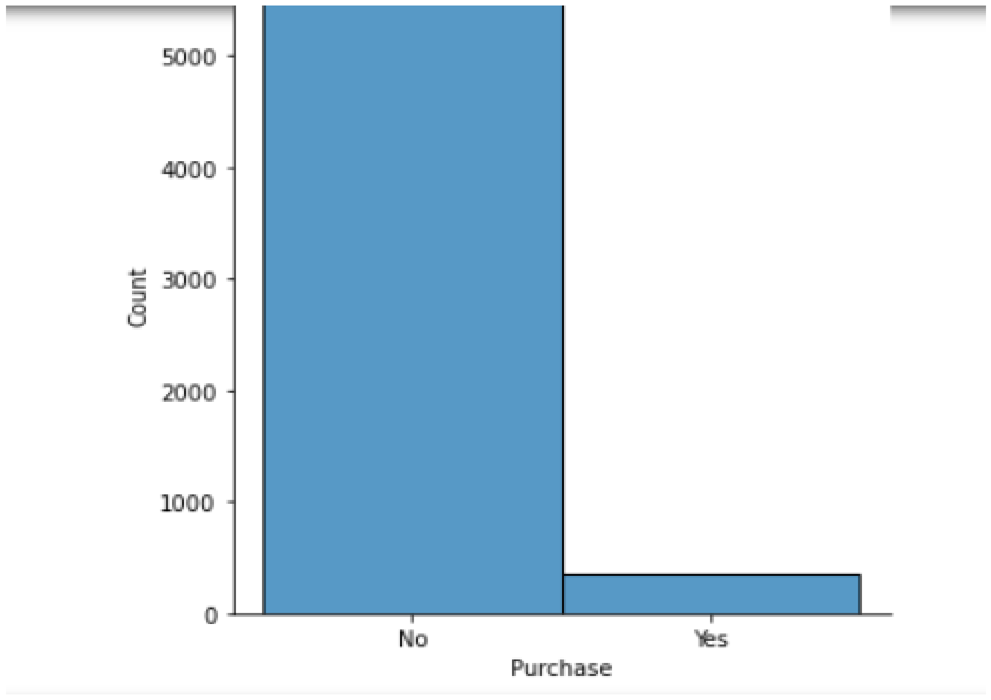
From data.describe():
Q1- 0
Q2-0
Q3-0
But the maximum value is 6.



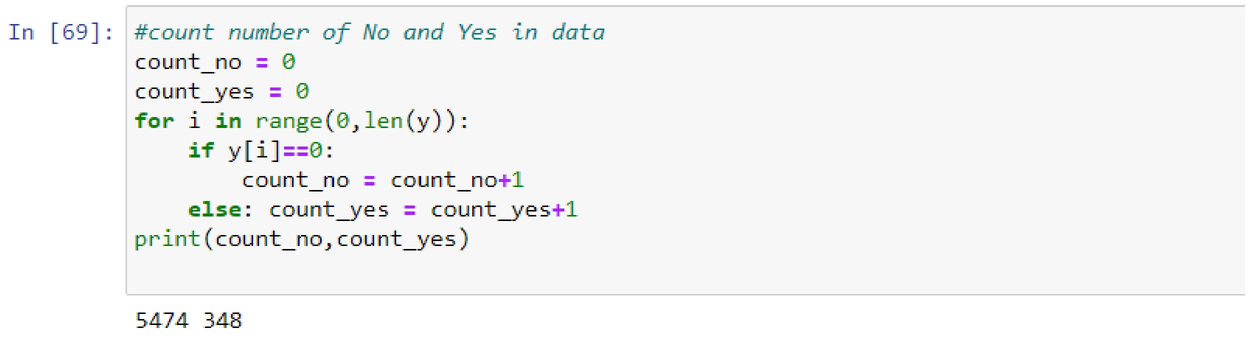
From the box plot we can see data points 1,2,3,4,5,6 comes under outliers denoted by dots.

Analysis of output column:

Histogram :



We can see that Data is highly imbalanced as more than 90 percent belongs to class-1(No) and very few data points belong to class-2(Yes).



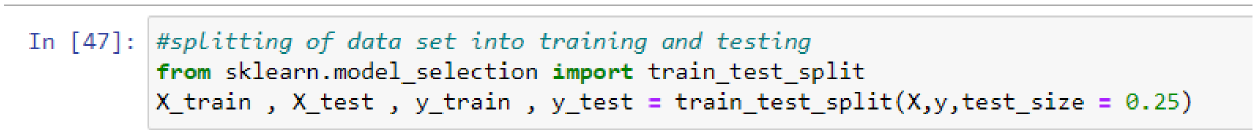
```
In [69]: #count number of No and Yes in data
         count_no = 0
         count_yes = 0
         for i in range(0,len(y)):
             if y[i]==0:
                 count_no = count_no+1
             else: count_yes = count_yes+1
         print(count_no,count_yes)

         5474 348
```

From the above snippet, we can see 5474 - NO and 348 - YES.

Assumption - For LDA and Decision tree classification no feature scaling is needed.

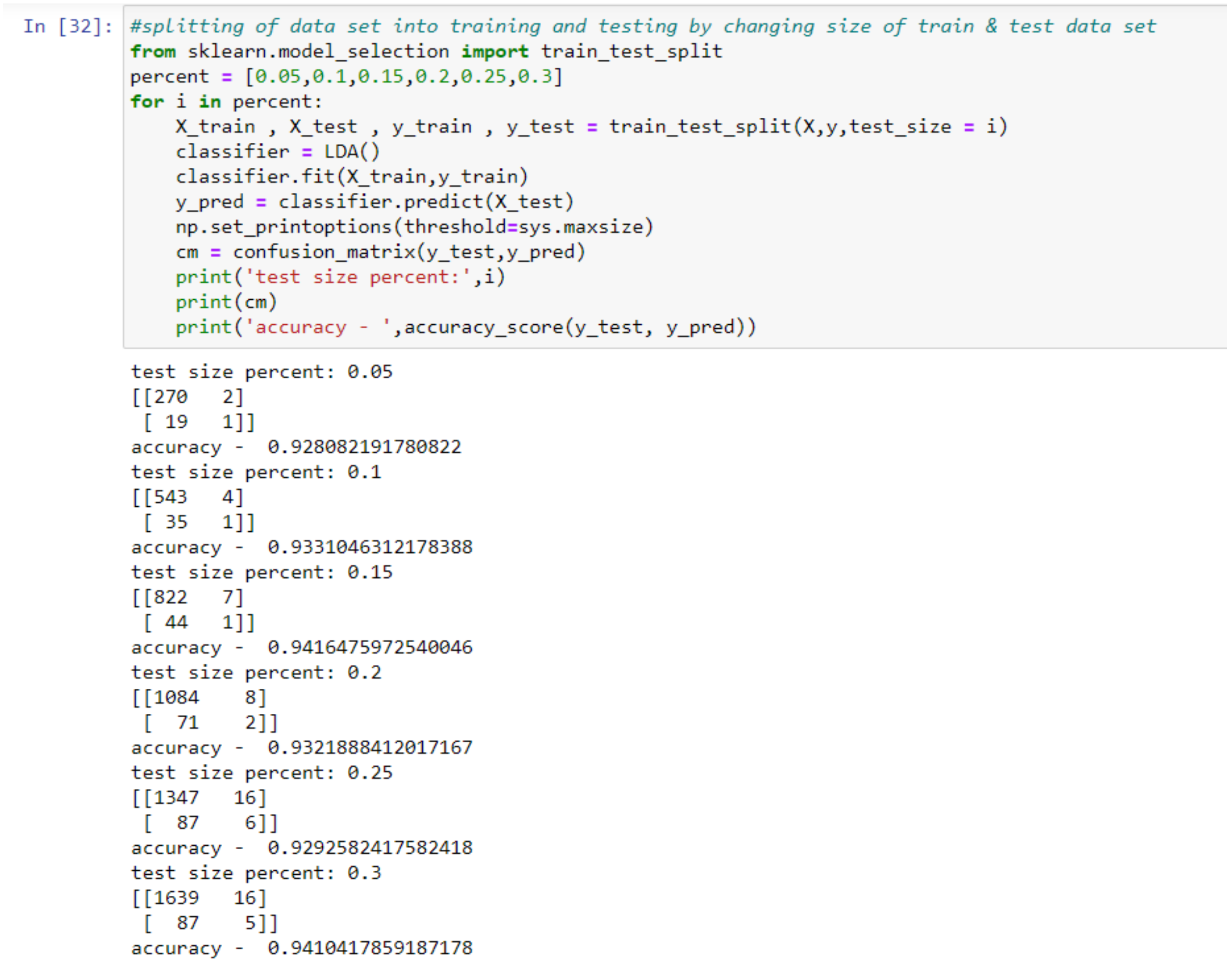
Linear Discriminant analysis:

- Splitting the train and test data(75% and 25% respectively).



```
In [47]: #splitting of data set into training and testing
         from sklearn.model_selection import train_test_split
         X_train , X_test , y_train , y_test = train_test_split(X,y,test_size = 0.25)
```

- Accuracy from our LDA analysis will vary every time we run the train_test_split command as it divides the dataset randomly.

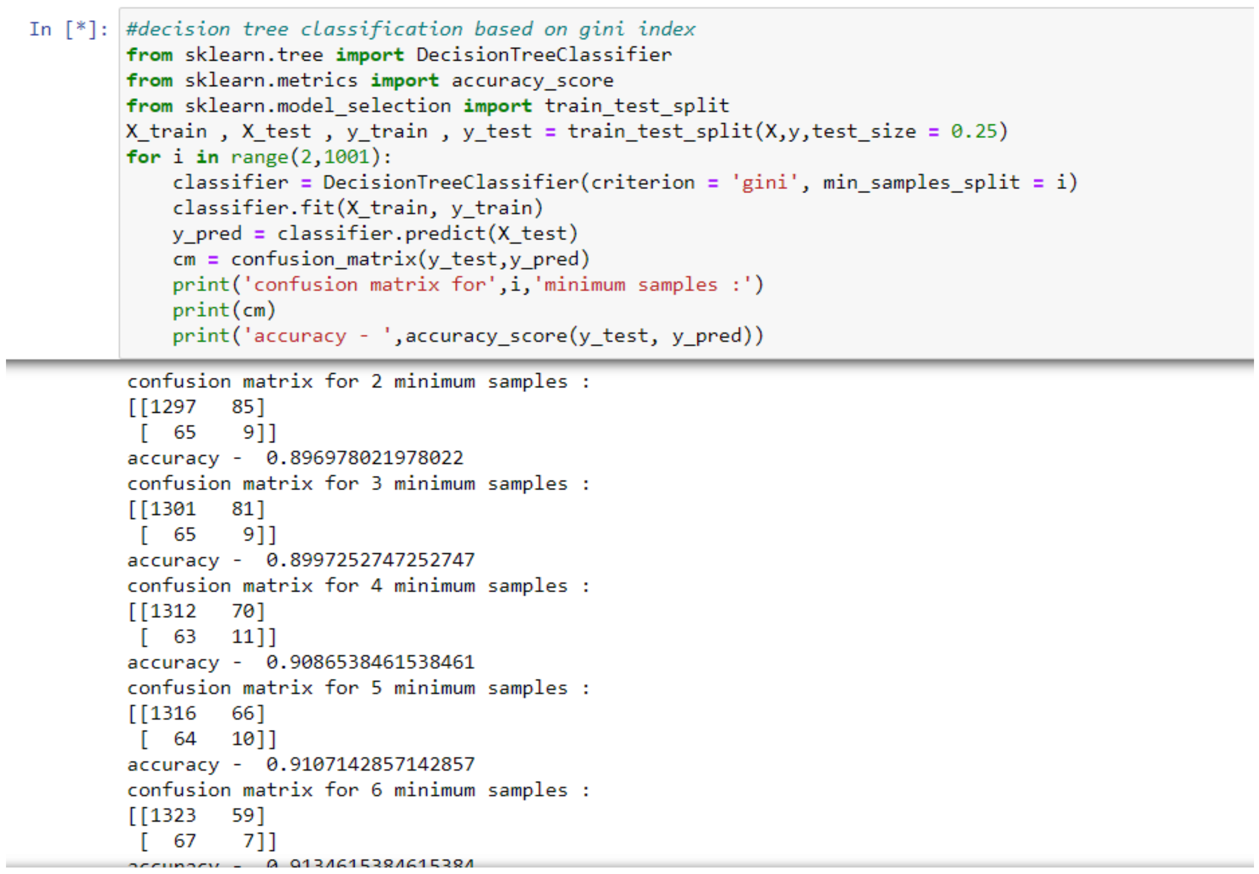


```
In [32]: #splitting of data set into training and testing by changing size of train & test data set
         from sklearn.model_selection import train_test_split
         percent = [0.05,0.1,0.15,0.2,0.25,0.3]
         for i in percent:
             X_train , X_test , y_train , y_test = train_test_split(X,y,test_size = i)
             classifier = LDA()
             classifier.fit(X_train,y_train)
             y_pred = classifier.predict(X_test)
             np.set_printoptions(threshold=sys.maxsize)
             cm = confusion_matrix(y_test,y_pred)
             print('test size percent:',i)
             print(cm)
             print('accuracy - ',accuracy_score(y_test, y_pred))

         test size percent: 0.05
         [[270   2]
          [ 19   1]]
         accuracy -  0.928082191780822
         test size percent: 0.1
         [[543   4]
          [ 35   1]]
         accuracy -  0.9331046312178388
         test size percent: 0.15
         [[822   7]
          [ 44   1]]
         accuracy -  0.9416475972540046
         test size percent: 0.2
         [[1084    8]
          [  71    2]]
         accuracy -  0.9321888412017167
         test size percent: 0.25
         [[1347   16]
          [  87    6]]
         accuracy -  0.9292582417582418
         test size percent: 0.3
         [[1639   16]
          [  87    5]]
         accuracy -  0.9410417859187178
```

- We can see that overall accuracy is more than 90 percent for each case as most of the data belongs to the 'NO' category.
- If we see the accuracy of only the 'YES' category, 0.25% has the highest accuracy of 0.06%

Decision Tree classification based on GINI index

- Different models can be made by deciding the minimum number of samples required for splitting fixing test size to be 25%.



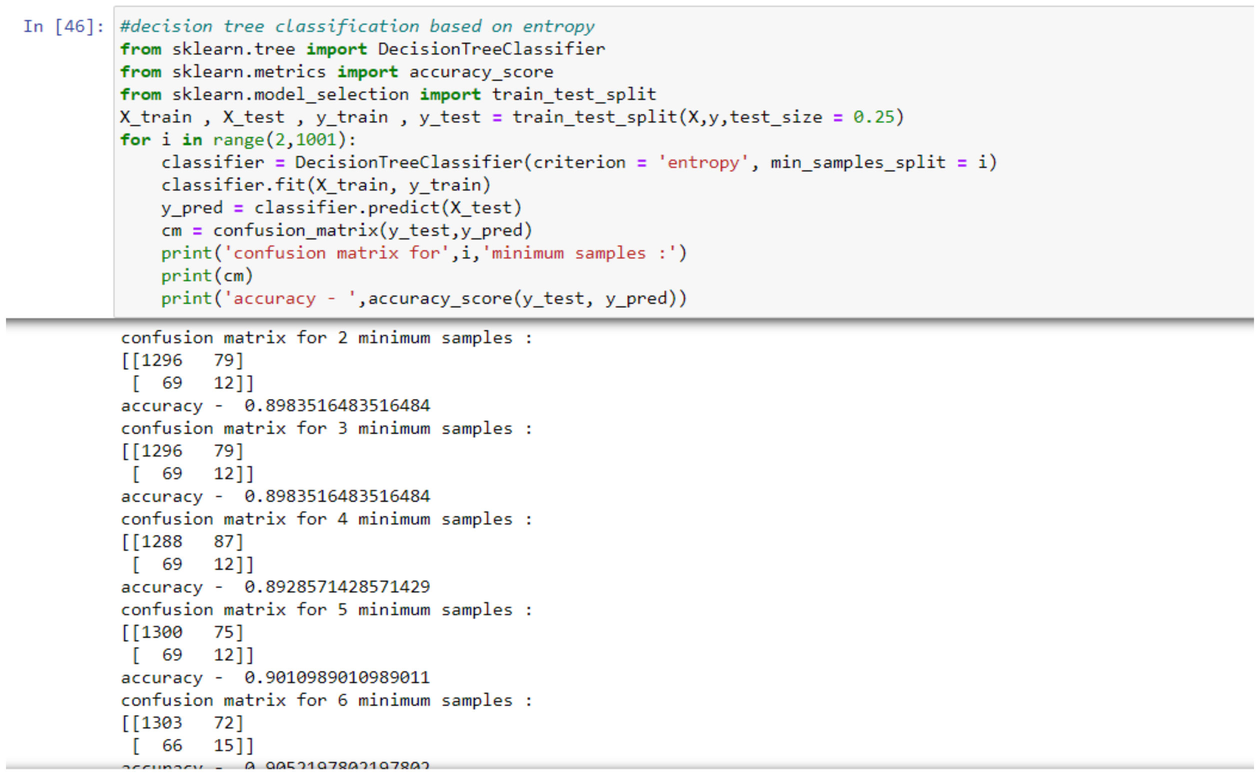
```
In [*]: #decision tree classification based on gini index
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.metrics import accuracy_score
        from sklearn.model_selection import train_test_split
        X_train , X_test , y_train , y_test = train_test_split(X,y,test_size = 0.25)
        for i in range(2,1001):
            classifier = DecisionTreeClassifier(criterion = 'gini', min_samples_split = i)
            classifier.fit(X_train, y_train)
            y_pred = classifier.predict(X_test)
            cm = confusion_matrix(y_test,y_pred)
            print('confusion matrix for',i,'minimum samples :')
            print(cm)
            print('accuracy - ',accuracy_score(y_test, y_pred))
```

```
confusion matrix for 2 minimum samples :
[[1297   85]
 [  65    9]]
accuracy -  0.896978021978022
confusion matrix for 3 minimum samples :
[[1301   81]
 [  65    9]]
accuracy -  0.8997252747252747
confusion matrix for 4 minimum samples :
[[1312   70]
 [  63   11]]
accuracy -  0.9086538461538461
confusion matrix for 5 minimum samples :
[[1316   66]
 [  64   10]]
accuracy -  0.9107142857142857
confusion matrix for 6 minimum samples :
[[1323   59]
 [  67    7]]
accuracy -  0.9134615384615384
```

- We can observe that as the data is so imbalanced, maximum accuracy is achieved when our model is always returning the predicted value in the 'No' category. Thus, we should use class-specific accuracy or assign more weights to small classes for a better model.
- Maximum accuracy achieved is 0.949%
- When the minimum number of samples for splitting is 4, we are getting the highest accuracy of the 'YES' class(0.15%).

Decision Tree classification based on entropy.

- Different models can be made by deciding the minimum number of samples required for splitting fixing test size to be 25%.



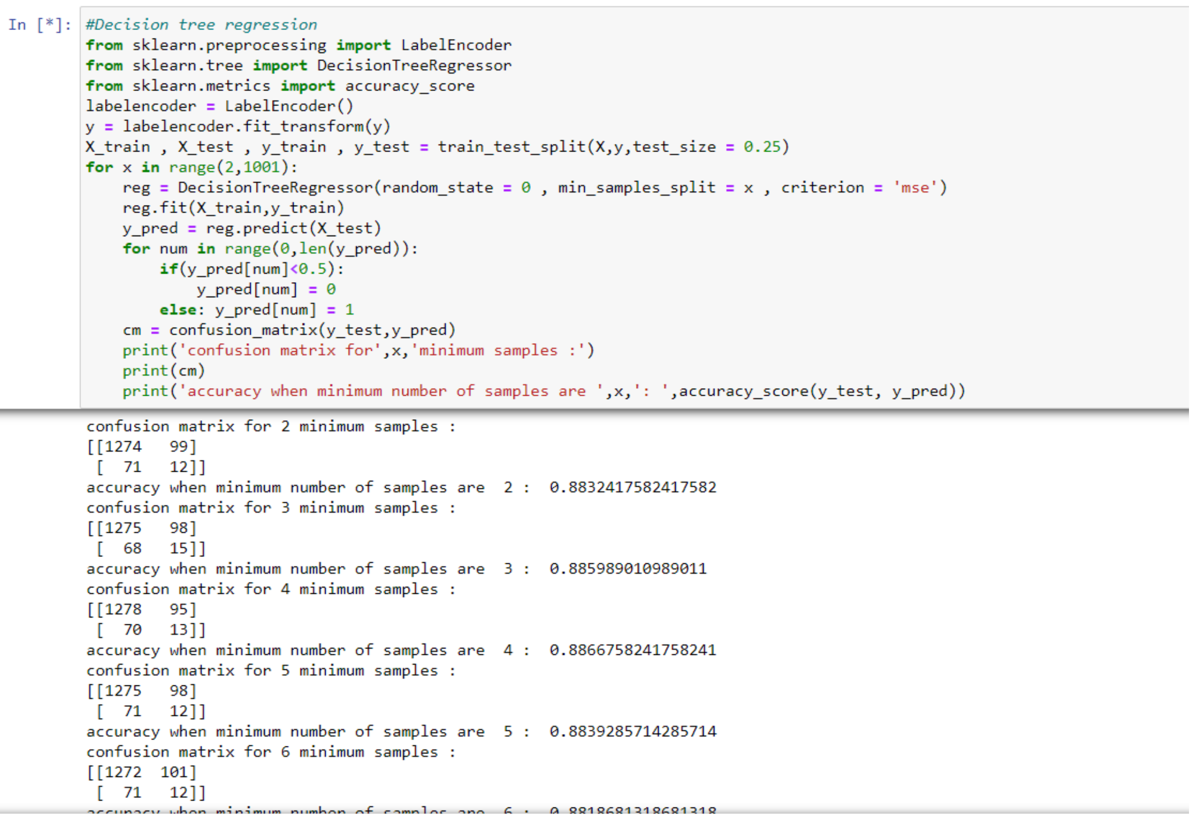
```
In [46]: #decision tree classification based on entropy
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.metrics import accuracy_score
         from sklearn.model_selection import train_test_split
         X_train , X_test , y_train , y_test = train_test_split(X,y,test_size = 0.25)
         for i in range(2,1001):
             classifier = DecisionTreeClassifier(criterion = 'entropy', min_samples_split = i)
             classifier.fit(X_train, y_train)
             y_pred = classifier.predict(X_test)
             cm = confusion_matrix(y_test,y_pred)
             print('confusion matrix for',i,'minimum samples :')
             print(cm)
             print('accuracy - ',accuracy_score(y_test, y_pred))
```

```
confusion matrix for 2 minimum samples :
[[1296   79]
 [  69   12]]
accuracy -  0.8983516483516484
confusion matrix for 3 minimum samples :
[[1296   79]
 [  69   12]]
accuracy -  0.8983516483516484
confusion matrix for 4 minimum samples :
[[1288   87]
 [  69   12]]
accuracy -  0.8928571428571429
confusion matrix for 5 minimum samples :
[[1300   75]
 [  69   12]]
accuracy -  0.9010989010989011
confusion matrix for 6 minimum samples :
[[1303   72]
 [  66   15]]
accuracy -  0.9052197802197802
```

- Similar observations can be made for this decision tree. Maximum accuracy is achieved when the predicted value is almost always 'No'.
- The maximum accuracy achieved is 0.944%.
- When the minimum number of samples for splitting is 6, we are getting the highest accuracy of the 'YES' class(0.18%).

Decision Tree Regression

- Different models can be made by deciding the minimum number of samples required for splitting and by fixing the test size to be 25%.



```
In [*]: #Decision tree regression
        from sklearn.preprocessing import LabelEncoder
        from sklearn.tree import DecisionTreeRegressor
        from sklearn.metrics import accuracy_score
        labelencoder = LabelEncoder()
        y = labelencoder.fit_transform(y)
        X_train , X_test , y_train , y_test = train_test_split(X,y,test_size = 0.25)
        for x in range(2,1001):
            reg = DecisionTreeRegressor(random_state = 0 , min_samples_split = x , criterion = 'mse')
            reg.fit(X_train,y_train)
            y_pred = reg.predict(X_test)
            for num in range(0,len(y_pred)):
                if(y_pred[num]<0.5):
                    y_pred[num] = 0
                else: y_pred[num] = 1
            cm = confusion_matrix(y_test,y_pred)
            print('confusion matrix for',x,'minimum samples :')
            print(cm)
            print('accuracy when minimum number of samples are ',x,': ',accuracy_score(y_test, y_pred))
```

```
confusion matrix for 2 minimum samples :
[[1274   99]
 [  71   12]]
accuracy when minimum number of samples are  2 :  0.8832417582417582
confusion matrix for 3 minimum samples :
[[1275   98]
 [  68   15]]
accuracy when minimum number of samples are  3 :  0.885989010989011
confusion matrix for 4 minimum samples :
[[1278   95]
 [  70   13]]
accuracy when minimum number of samples are  4 :  0.8866758241758241
confusion matrix for 5 minimum samples :
[[1275   98]
 [  71   12]]
accuracy when minimum number of samples are  5 :  0.8839285714285714
confusion matrix for 6 minimum samples :
[[1272  101]
 [  71   12]]
accuracy when minimum number of samples are  6 :  0.8818681318681318
```

- If the regression model predicts a value less than 0.5, it is considered in the 'NO' category else 'YES' category.
- The maximum accuracy achieved is 0.9423%.
- When the minimum number of samples for splitting is 3, we are getting the highest accuracy of the 'YES' class(0.18%).

Note: For all of the above methods, data will change whenever we run the python command as train_test_split() will split the data randomly.