RATING PREDICTION PROJECT

SUBMITTED BY:-

NAINCY JOSHI

PROBLEM STATEMENT:-

E-commerce, the online app, has reached out to you to help them to predict how good or bad a product will turn out in the future. So that, they can take a decision to include the product in their app or remove it.

 It contains the details about the products and what rating it achieved finally through many customers.

My task is to create a machine learning model which can predict the Rating of a Product based on its Review from the customers.

In below case study I will discuss the step by step approach to create a Machine Learning predictive model in such scenarios.

You can use this flow as a template to solve any supervised ML Regression problem!

The flow of the case study is as below:

- Reading the data in python
- Defining the problem statement
- Identifying the Target variable
- Looking at the distribution of Target variable
- Basic Data exploration
- Feature Engineering
- Rejecting useless columns
- Visual Exploratory Data Analysis for data distribution (Histogram and Barcharts)
- Feature Selection based on data distribution
- Outlier treatment
- Missing Values treatment
- Visual correlation analysis
- Statistical correlation analysis (Feature Selection)
- Converting data to numeric for ML
- Sampling and K-fold cross validation
- Trying multiple Regression algorithms

- Selecting the best Model
- Deploying the best model in production

Create a Predictive model which can predict the future Rating of a product from any E-commerce websites.

- Target Variable: Rating
- Predictors: reviews of the customers.

- Rating=1 Worst
- Rating=5 Best

Determining the type of Machine Learning

Based on the problem statement you can understand that we need to create a **supervised ML Regression model**, as the target variable is Continuous.

Basic Data Exploration

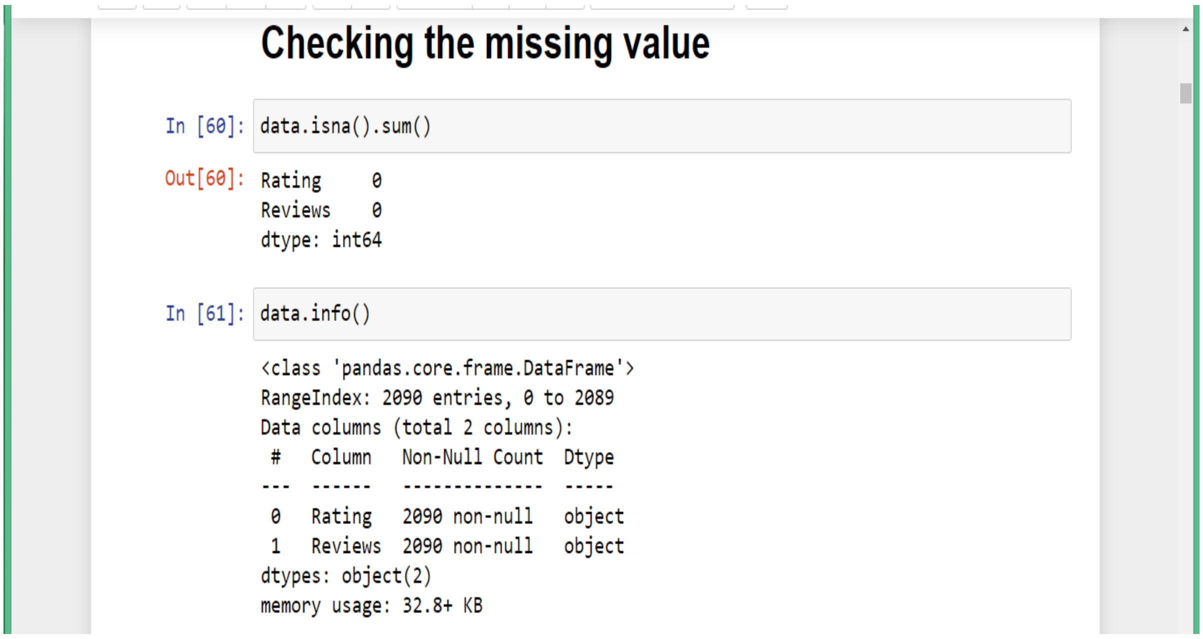
This step is performed to guage the overall data. The volume of data, the types of columns present in the data. Initial assessment of the data should be done to identify which columns are Quantitative, Categorical or Qualitative.

This step helps to start the column rejection process. You must look at each column carefully and ask, does this column affect the values of the

Target variable? For example in this case study, we have only one column i.e Review column based upon the review we will classify the rating.

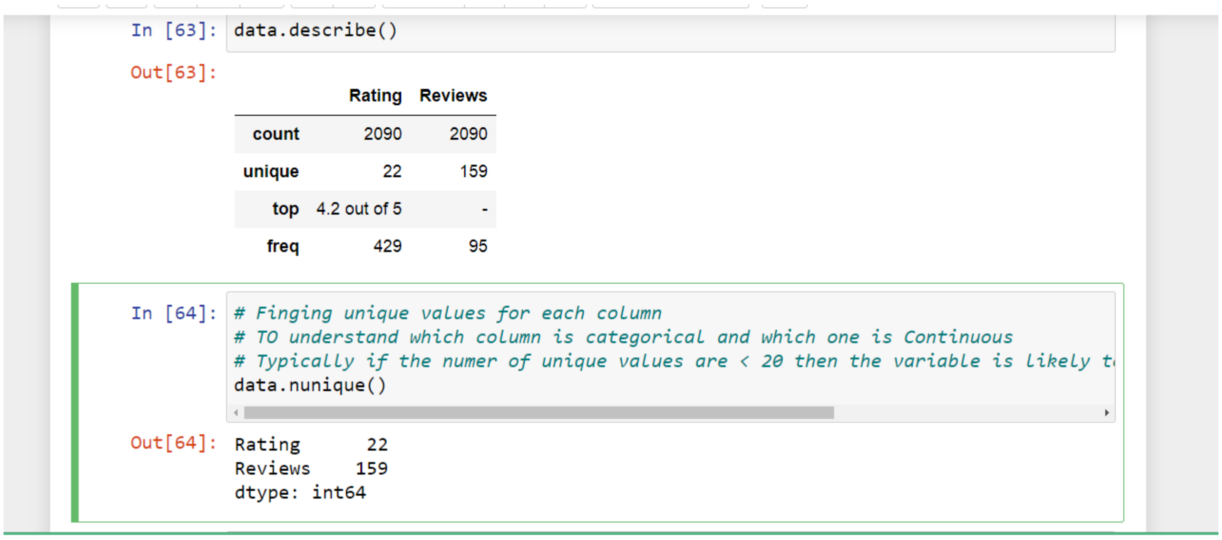
There are four commands which are used for Basic data exploration in Python

- **head()** : This helps to see a few sample rows of the data
- **info()** : This provides the summarized information of the data
- **describe()** : This provides the descriptive statistical details of the data
- **nunique()**: This helps us to identify if a column is categorical or continuous
- **Isna()**.sum(): This provide information about the missing value in the dataset.



Checking the missing value

```
In [60]: data.isna().sum()
Out[60]: Rating     0
         Reviews    0
         dtype: int64

In [61]: data.info()

         <class 'pandas.core.frame.DataFrame'>
         RangeIndex: 2090 entries, 0 to 2089
         Data columns (total 2 columns):
          #   Column   Non-Null Count  Dtype 
         ---  ------   --------------  ----- 
          0   Rating   2090 non-null   object
          1   Reviews  2090 non-null   object
         dtypes: object(2)
         memory usage: 32.8+ KB
```



```
In [63]: data.describe()
```

Out[63]:

	Rating	Reviews
count	2090	2090
unique	22	159
top	4.2 out of 5	-
freq	429	95

```
In [64]: # Finging unique values for each column
         # TO understand which column is categorical and which one is Continuous
         # Typically if the numer of unique values are < 20 then the variable is likely t
         data.nunique()
```

```
Out[64]: Rating      22
         Reviews    159
         dtype: int64
```

Basic Data Exploration Results

Based on the basic exploration above, you can now create a simple report of the data, noting down your observations regarding each column. Hence, creating an initial roadmap for further analysis.

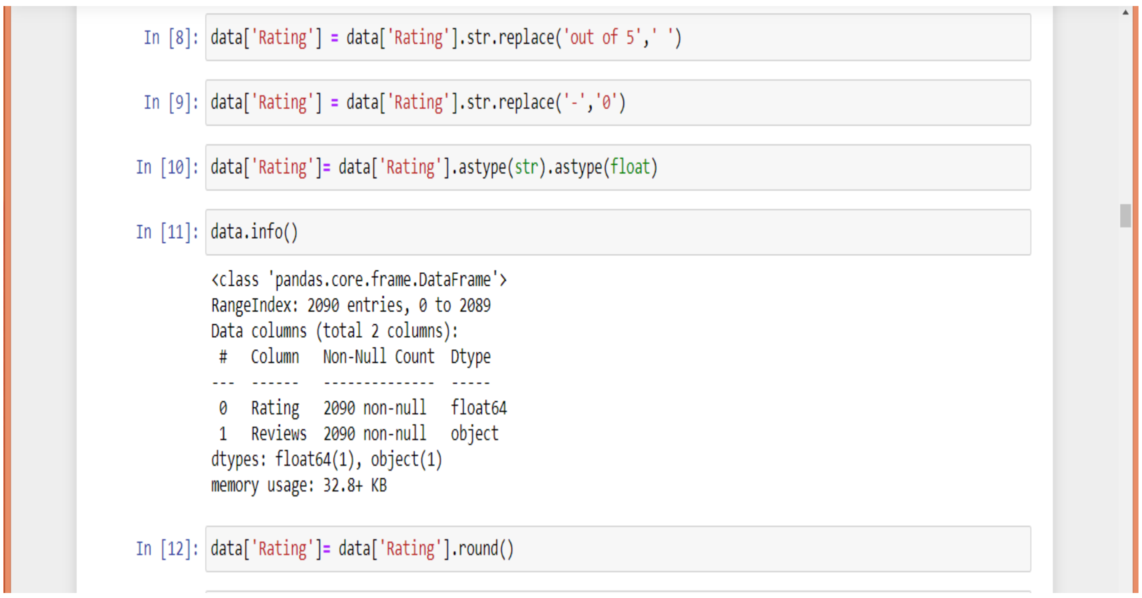
- **Reviews**: Continuous. Selected.
- **Rating**: Continuous. Selected. This is the **Target Variable!**

Feature Engineering

Some of the columns cannot be used directly for machine learning like Dates, addresses, etc. because these are qualitative in nature. Hence every row has a different string value, Hence the ML algorithms cannot learn anything from them because each row has a different description. In simple terms, no general rules can be created from such columns.

However, we can extract some information from these columns, which can be used in ML. Like from a date column we can take out month, week, quarter etc, which becomes categorical feature. If it is given in the dataset.

The data which are used are of object type. The rating column also incude out 0f 5 that has to be replaced with the blank space. The review column contain text so we have to convert into machine language.



```
In [8]: data['Rating'] = data['Rating'].str.replace('out of 5',' ')

In [9]: data['Rating'] = data['Rating'].str.replace('-','0')

In [10]: data['Rating']= data['Rating'].astype(str).astype(float)

In [11]: data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2090 entries, 0 to 2089
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Rating   2090 non-null   float64
 1   Reviews  2090 non-null   object
dtypes: float64(1), object(1)
memory usage: 32.8+ KB

In [12]: data['Rating']= data['Rating'].round()
```

Removing useless columns from the data

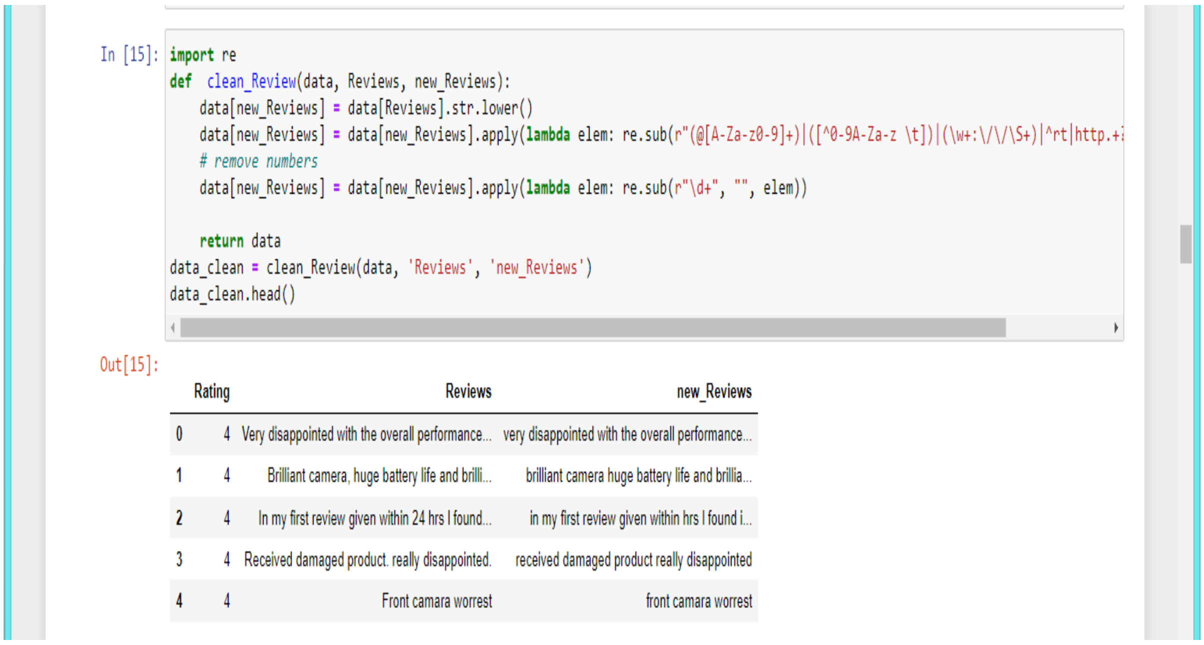
As we have only 2 data in the dataset , so we don’t have any column to remove

Visual Exploratory Data Analysis

As we have only one target variable and one feature that is Review and both are in “Object” datatype. The review are in the text form which has to be converted into machine learning language.

Feature Engineering

- We will convert all the text into lower case.
- In the review column, we will remove the the numbers or any other symbolic expression which has no meaning to the computer language.
- Commas and semicolons are to be removed



```
In [15]: import re
         def  clean_Review(data, Reviews, new_Reviews):
             data[new_Reviews] = data[Reviews].str.lower()
             data[new_Reviews] = data[new_Reviews].apply(lambda elem: re.sub(r"(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)|^rt|http.+
             # remove numbers
             data[new_Reviews] = data[new_Reviews].apply(lambda elem: re.sub(r"\d+", "", elem))

             return data
         data_clean = clean_Review(data, 'Reviews', 'new_Reviews')
         data_clean.head()
```

Out[15]:

	Rating	Reviews	new_Reviews
0	4	Very disappointed with the overall performance...	very disappointed with the overall performance...
1	4	Brilliant camera, huge battery life and brilli...	brilliant camera huge battery life and brillia...
2	4	In my first review given within 24 hrs I found...	in my first review given within hrs I found i...
3	4	Received damaged product. really disappointed.	received damaged product really disappointed
4	4	Front camara worrest	front camara worrest

- The NLTK library is one of the oldest and most commonly used Python libraries for Natural Language Processing. NLTK supports stop word removal, and you can find the list of stop words in the corpus module.
- Text may contain stop words like ‘the’, ‘is’, ‘are’. Stop words can be filtered from the text to be processed. There is no universal list of stop words in nlp (Natural Language Process) research, however the nltk module contains a list of stop words.
- Stop words are frequently used words that carry very little meaning. Stop words are words that are so common they are basically ignored by typical tokenizers.
- By default, NLTK (Natural Language Toolkit) includes a list of 40 stop words, including: “a”, “an”, “the”, “of”, “in”, etc.
- TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency.
- This is very common algorithm to transform text into a meaningful representation of numbers which is used to fit machine algorithm for prediction.
- Let’s take sample example and explore two different spicy sparse matrix before go into deep explanation . It gives overall view what I am trying to explain below .Simple basic example of TF-IDF.
- **Train Document Set:**

  d1: The sky is blue.

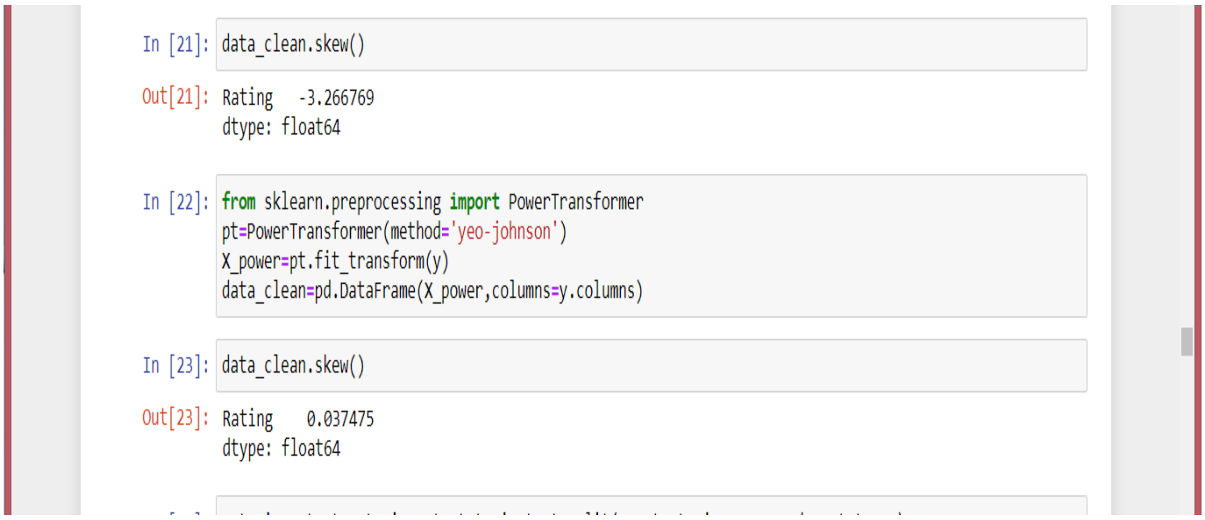
d2: The sun is bright.

Test Document Set:

d3: The sun **in** the sky is bright.

d4: We can see the shining sun, the bright sun.

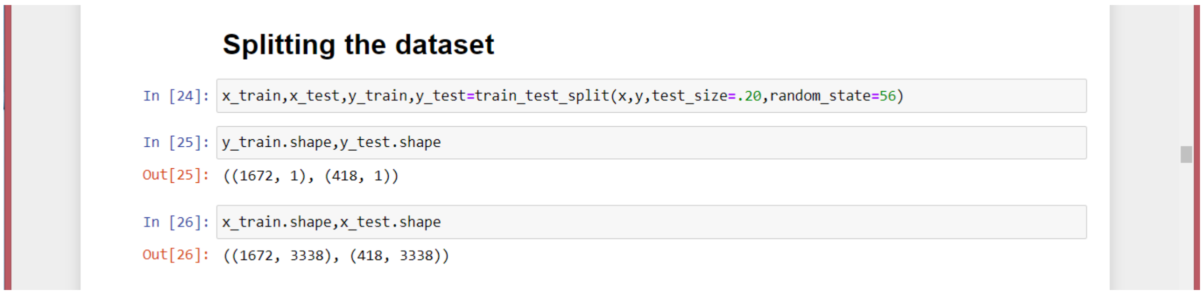
Removing the skewness:



```
In [21]: data_clean.skew()

Out[21]: Rating   -3.266769
         dtype: float64

In [22]: from sklearn.preprocessing import PowerTransformer
         pt=PowerTransformer(method='yeo-johnson')
         X_power=pt.fit_transform(y)
         data_clean=pd.DataFrame(X_power,columns=y.columns)

In [23]: data_clean.skew()

Out[23]: Rating    0.037475
         dtype: float64
```

- Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like.
- Currently, Power Transformer supports the Box-Cox transform and the Yeo-Johnson transform. The optimal parameter for stabilizing variance and minimizing skewness is estimated through maximum likelihood.
- Box-Cox requires input data to be strictly positive, while Yeo-Johnson supports both positive or negative data.
- We have removed the skewness using power transformer

Machine Learning: Splitting the data into Training and Testing sample

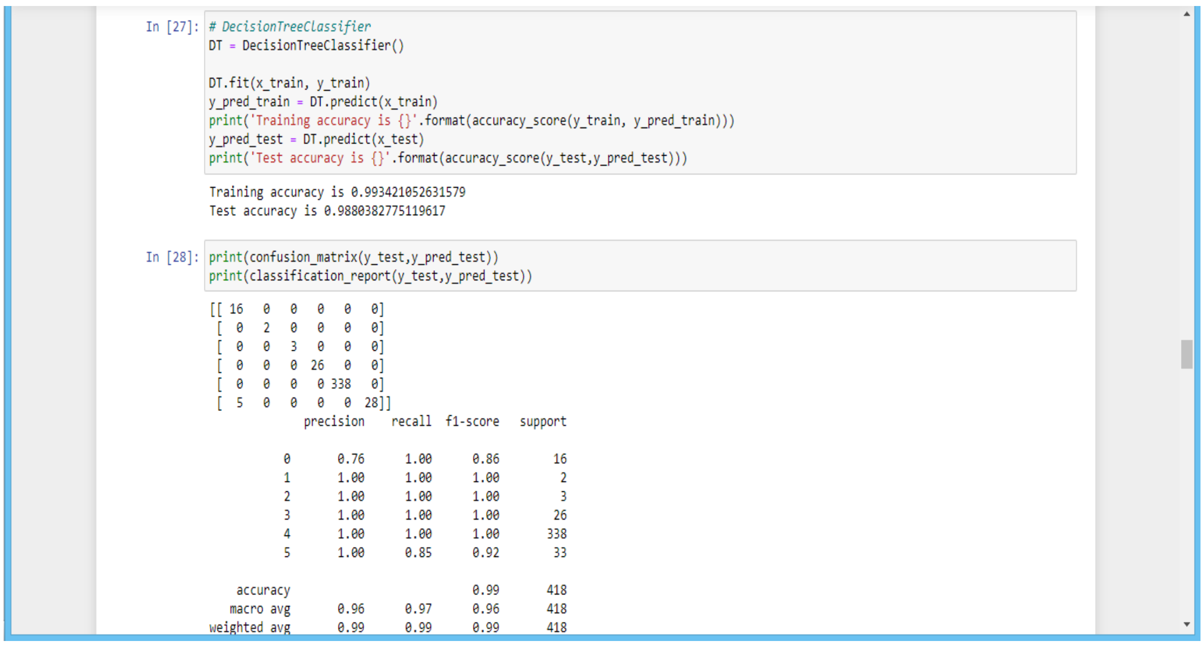


Splitting the dataset

```
In [24]: x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.20,random_state=56)

In [25]: y_train.shape,y_test.shape
Out[25]: ((1672, 1), (418, 1))

In [26]: x_train.shape,x_test.shape
Out[26]: ((1672, 3338), (418, 3338))
```

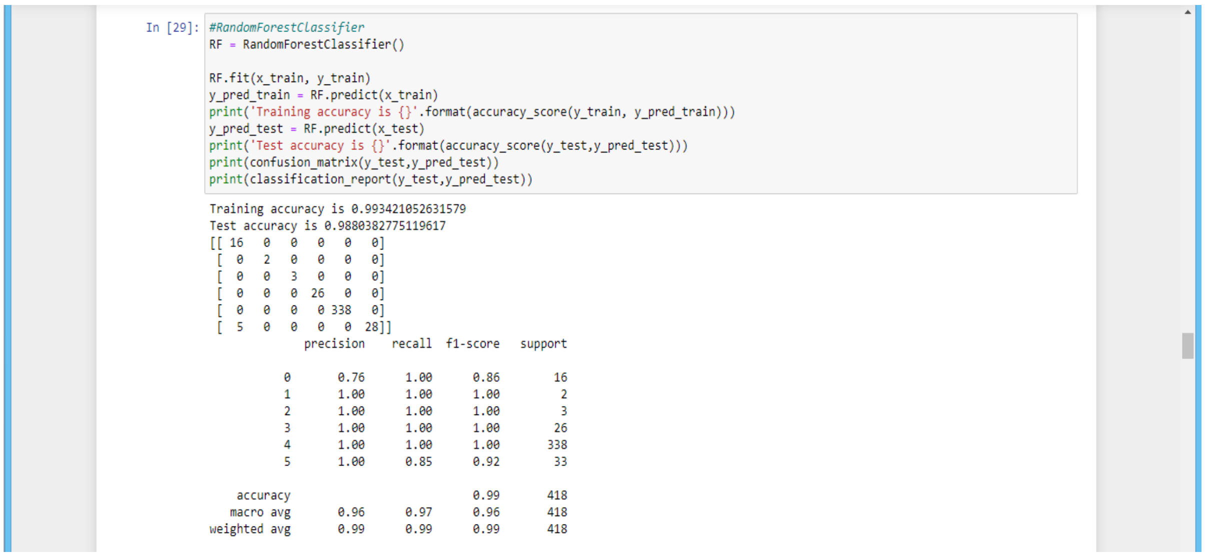
Building the model

1. Decision Tree Classifier



```
In [27]: # DecisionTreeClassifier
         DT = DecisionTreeClassifier()

         DT.fit(x_train, y_train)
         y_pred_train = DT.predict(x_train)
         print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
         y_pred_test = DT.predict(x_test)
         print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))

         Training accuracy is 0.993421052631579
         Test accuracy is 0.9880382775119617

In [28]: print(confusion_matrix(y_test,y_pred_test))
         print(classification_report(y_test,y_pred_test))

         [[ 16   0   0   0   0   0]
          [  0   2   0   0   0   0]
          [  0   0   3   0   0   0]
          [  0   0   0  26   0   0]
          [  0   0   0   0 338   0]
          [  5   0   0   0   0  28]]
                       precision    recall  f1-score   support

                    0       0.76      1.00      0.86        16
                    1       1.00      1.00      1.00         2
                    2       1.00      1.00      1.00         3
                    3       1.00      1.00      1.00        26
                    4       1.00      1.00      1.00       338
                    5       1.00      0.85      0.92        33

             accuracy                           0.99       418
            macro avg       0.96      0.97      0.96       418
         weighted avg       0.99      0.99      0.99       418
```

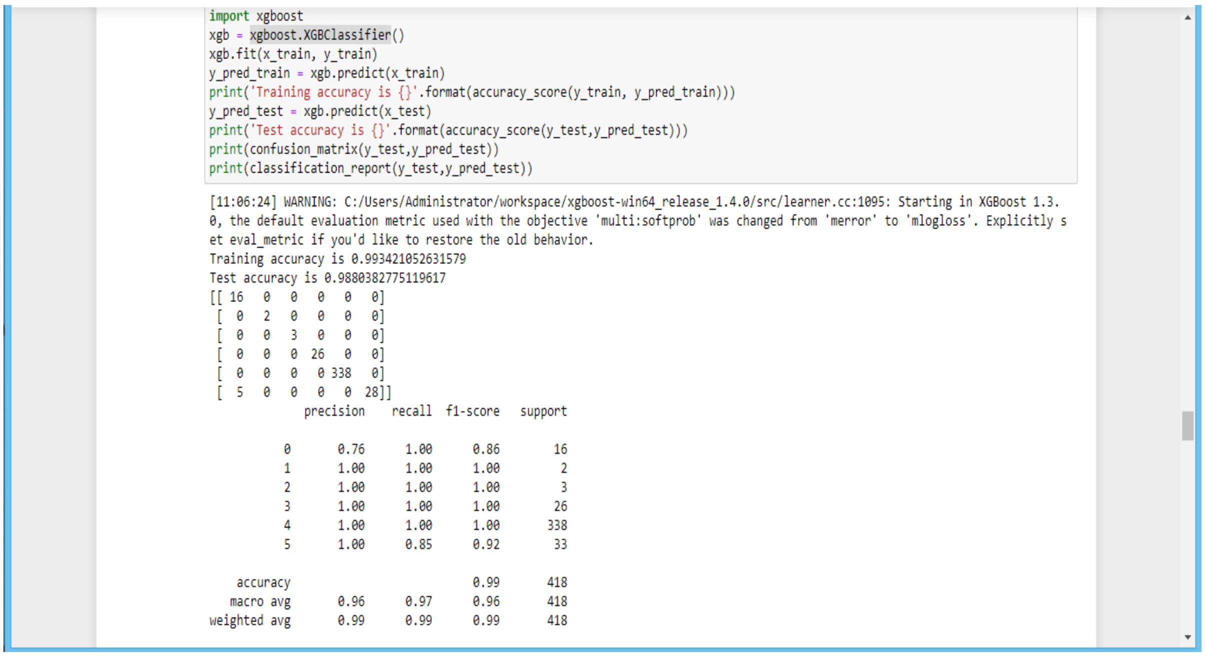
Random Forest classifier



```
In [29]: #RandomForestClassifier
         RF = RandomForestClassifier()

         RF.fit(x_train, y_train)
         y_pred_train = RF.predict(x_train)
         print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
         y_pred_test = RF.predict(x_test)
         print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
         print(confusion_matrix(y_test,y_pred_test))
         print(classification_report(y_test,y_pred_test))
```

```
Training accuracy is 0.993421052631579
Test accuracy is 0.9880382775119617
[[ 16   0   0   0   0   0]
 [  0   2   0   0   0   0]
 [  0   0   3   0   0   0]
 [  0   0   0  26   0   0]
 [  0   0   0   0 338   0]
 [  5   0   0   0   0  28]]
              precision    recall  f1-score   support

           0       0.76      1.00      0.86        16
           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00         3
           3       1.00      1.00      1.00        26
           4       1.00      1.00      1.00       338
           5       1.00      0.85      0.92        33

    accuracy                           0.99       418
   macro avg       0.96      0.97      0.96       418
weighted avg       0.99      0.99      0.99       418
```

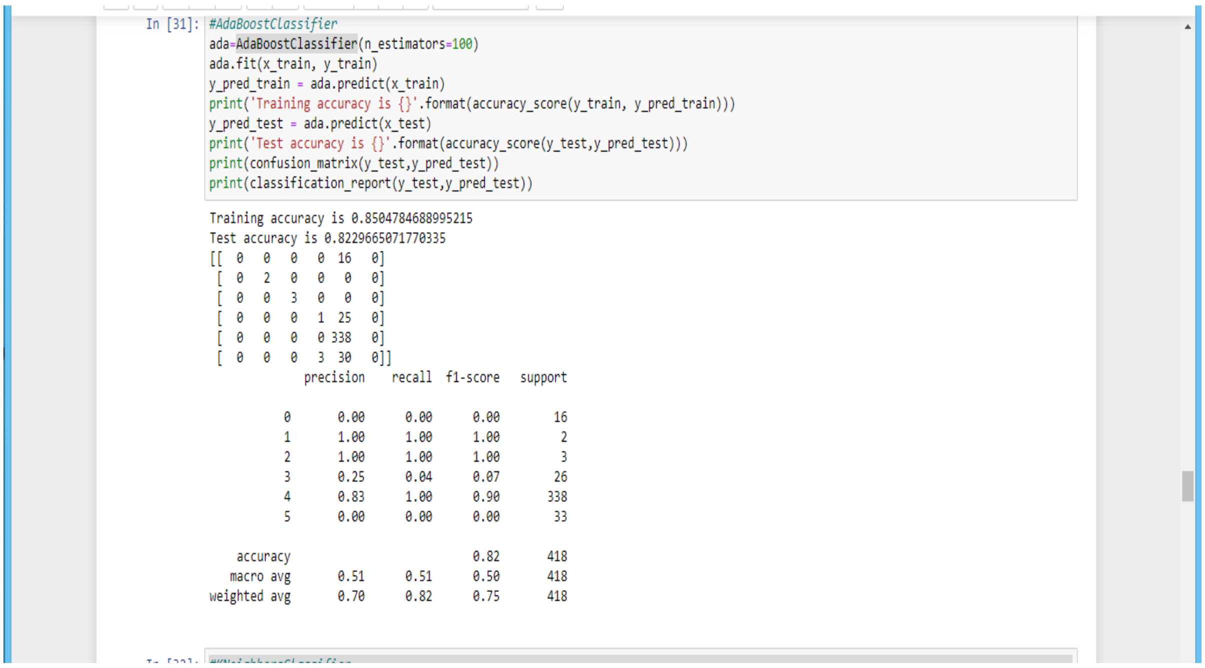
XGB Classifier



```
import xgboost
xgb = xgboost.XGBClassifier()
xgb.fit(x_train, y_train)
y_pred_train = xgb.predict(x_train)
print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
y_pred_test = xgb.predict(x_test)
print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
print(confusion_matrix(y_test,y_pred_test))
print(classification_report(y_test,y_pred_test))
```

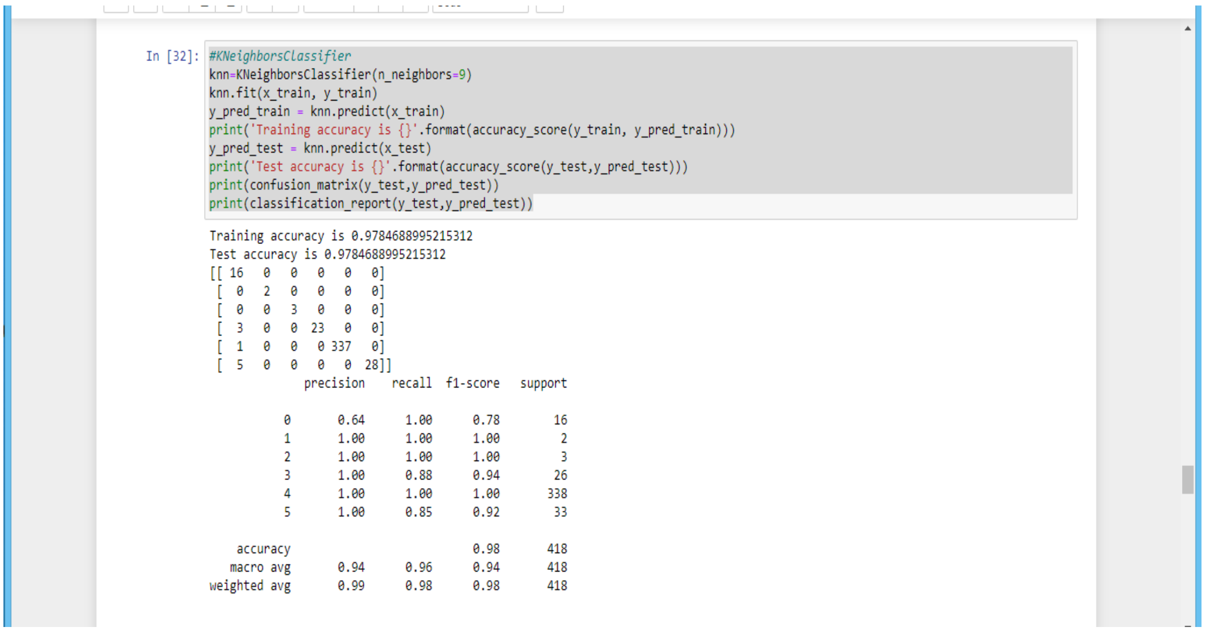
```
[11:06:24] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/learner.cc:1095: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly s
et eval_metric if you'd like to restore the old behavior.
Training accuracy is 0.993421052631579
Test accuracy is 0.9880382775119617
[[ 16   0   0   0   0   0]
 [  0   2   0   0   0   0]
 [  0   0   3   0   0   0]
 [  0   0   0  26   0   0]
 [  0   0   0   0 338   0]
 [  5   0   0   0   0  28]]
              precision    recall  f1-score   support

           0       0.76      1.00      0.86        16
           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00         3
           3       1.00      1.00      1.00        26
           4       1.00      1.00      1.00       338
           5       1.00      0.85      0.92        33

    accuracy                           0.99       418
   macro avg       0.96      0.97      0.96       418
weighted avg       0.99      0.99      0.99       418
```

AdaBoost Classifier



```
In [31]: #AdaBoostClassifier
         ada=AdaBoostClassifier(n_estimators=100)
         ada.fit(x_train, y_train)
         y_pred_train = ada.predict(x_train)
         print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
         y_pred_test = ada.predict(x_test)
         print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
         print(confusion_matrix(y_test,y_pred_test))
         print(classification_report(y_test,y_pred_test))

         Training accuracy is 0.8504784688995215
         Test accuracy is 0.8229665071770335
         [[  0   0   0   0  16   0]
          [  0   2   0   0   0   0]
          [  0   0   3   0   0   0]
          [  0   0   0   1  25   0]
          [  0   0   0   0 338   0]
          [  0   0   0   3  30   0]]
                       precision    recall  f1-score   support

                    0       0.00      0.00      0.00        16
                    1       1.00      1.00      1.00         2
                    2       1.00      1.00      1.00         3
                    3       0.25      0.04      0.07        26
                    4       0.83      1.00      0.90       338
                    5       0.00      0.00      0.00        33

             accuracy                           0.82       418
            macro avg       0.51      0.51      0.50       418
         weighted avg       0.70      0.82      0.75       418
```

K-Nearest Neighbor



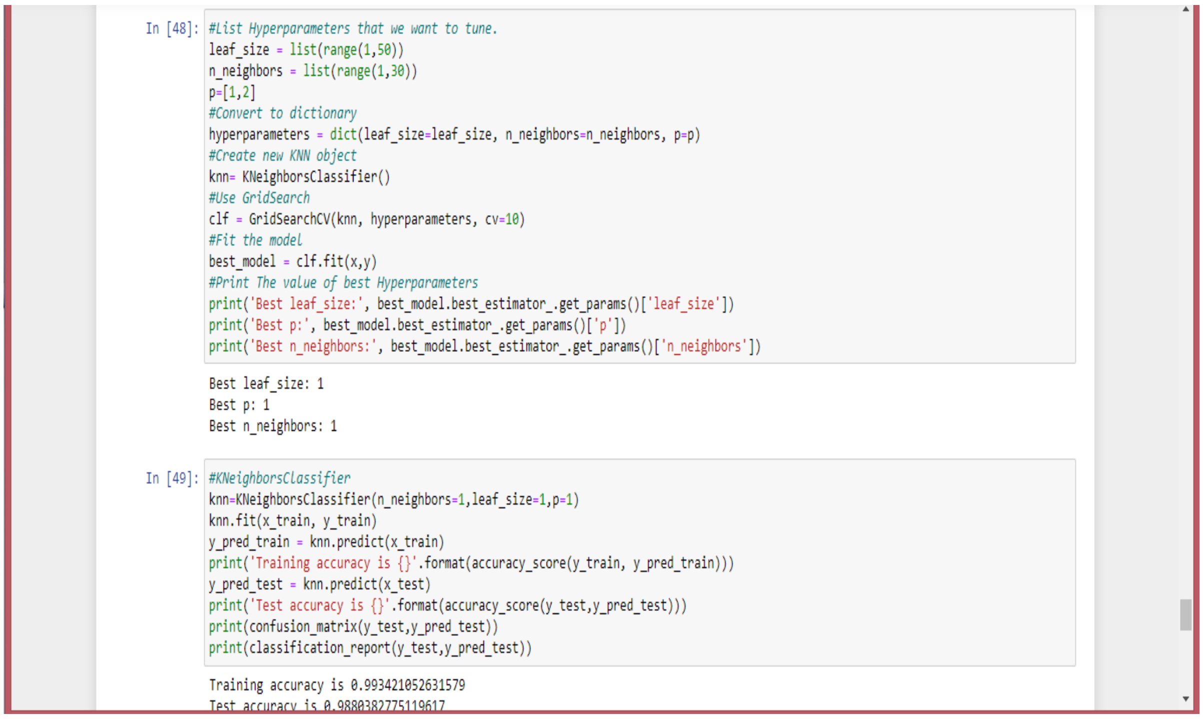
```
In [32]: #KNeighborsClassifier
         knn=KNeighborsClassifier(n_neighbors=9)
         knn.fit(x_train, y_train)
         y_pred_train = knn.predict(x_train)
         print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
         y_pred_test = knn.predict(x_test)
         print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
         print(confusion_matrix(y_test,y_pred_test))
         print(classification_report(y_test,y_pred_test))

         Training accuracy is 0.9784688995215312
         Test accuracy is 0.9784688995215312
         [[ 16   0   0   0   0   0]
          [  0   2   0   0   0   0]
          [  0   0   3   0   0   0]
          [  3   0   0  23   0   0]
          [  1   0   0   0 337   0]
          [  5   0   0   0   0  28]]
                       precision    recall  f1-score   support

                    0       0.64      1.00      0.78        16
                    1       1.00      1.00      1.00         2
                    2       1.00      1.00      1.00         3
                    3       1.00      0.88      0.94        26
                    4       1.00      1.00      1.00       338
                    5       1.00      0.85      0.92        33

             accuracy                           0.98       418
            macro avg       0.94      0.96      0.94       418
         weighted avg       0.99      0.98      0.98       418
```

- This step is actually quite simple. Once we decide which model to apply on the data, we can create an object of its corresponding class, and fit the object on our training set, considering X_train as the input and y_train as the output.

- After this step we will improve the model accuracy or its performance using hyper-parameter function.
- In this model we will take the K-nearest neighbor because the difference between the train accuracy and test accuracy is less.
- The performance of a classifier can be assessed by the parameters of accuracy, precision, recall, and f1-score. These values can be seen using a method known as classification_report (). It can also be viewed as a confusion matrix that helps us to know how many of which category of data have been classified correctly.

Hyper-Parameter Tunning

- *The k-nearest neighbors algorithm (KNN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression.*
- In the case of k-NN, we can tune *k*, the number of nearest neighbors.
- We can also tune our distance i.e P metric/similarity function as well.

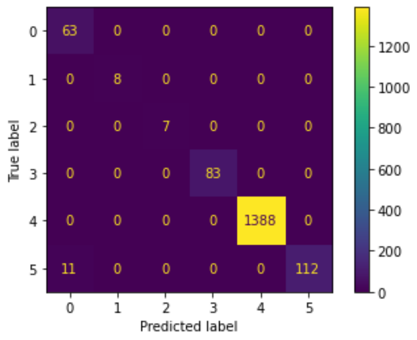


```
In [48]: #List Hyperparameters that we want to tune.
         leaf_size = list(range(1,50))
         n_neighbors = list(range(1,30))
         p=[1,2]
         #Convert to dictionary
         hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)
         #Create new KNN object
         knn= KNeighborsClassifier()
         #Use GridSearch
         clf = GridSearchCV(knn, hyperparameters, cv=10)
         #Fit the model
         best_model = clf.fit(x,y)
         #Print The value of best Hyperparameters
         print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
         print('Best p:', best_model.best_estimator_.get_params()['p'])
         print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])

         Best leaf_size: 1
         Best p: 1
         Best n_neighbors: 1

In [49]: #KNeighborsClassifier
         knn=KNeighborsClassifier(n_neighbors=1,leaf_size=1,p=1)
         knn.fit(x_train, y_train)
         y_pred_train = knn.predict(x_train)
         print('Training accuracy is {}'.format(accuracy_score(y_train, y_pred_train)))
         y_pred_test = knn.predict(x_test)
         print('Test accuracy is {}'.format(accuracy_score(y_test,y_pred_test)))
         print(confusion_matrix(y_test,y_pred_test))
         print(classification_report(y_test,y_pred_test))

         Training accuracy is 0.993421052631579
         Test accuracy is 0.9880382775119617
```

- WE got the the K-nearest number as 1 and the distance P is also 1.
- Hyperparameter tuning is done to increase the efficiency of a model by tuning the parameters of the neural network.
- We have increased the model accuracy from 97% to 99%.

Saving the Model

```
Out[55]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0xadb4faac40>
```



We have improved our accuracy of the model from 97% to 99% parameter tunning

Saving the model

```
In [ ]: import joblib
        joblib.dump(knn,"Rating_Prediction.pkl")
```