Assessment

Machine Learning and Data Mining

MSc Data Science

Exploring Insights: Classification, Clustering, Text Classification, and Sentiment Analysis

Vyshak Madathil Vattekat

@00687748

08-12-2023

Table of Contents

Part One: Classification Algorithms

Title: Identify if a mushroom is edible to eat or poisonous

1. Introduction

It is essential to distinguish between toxic and edible mushrooms in order to protect those who collect mushrooms for culinary use. Studying this dataset is driven by real-world issues around the gathering and eating of wild mushrooms. A great case study for assessing how well machine learning algorithms perform in categorization tasks is the Mushroom dataset. Based on observable traits, algorithms trained on this dataset can be used to automatically determine if mushrooms are edible.

In the field of machine learning and data mining, the Mushroom dataset is a well-known dataset that is frequently used for classification tasks. This dataset, which was initially taken from the UCI Machine Learning Repository, offers a thorough assortment of features that describe different aspects of mushrooms. Developing classification models that can correctly distinguish between edible and deadly mushrooms based on their observable properties is the main goal of using this dataset.

2. Research Objective

"Can machine learning algorithms effectively predict mushroom edibility based on observable traits, and if yes, which classification model performs best in discriminating between edible and dangerous mushrooms?"

We use at least two classification algorithms and perform a machine learning classification in both Python as well as Microsoft Azure to achieve the above objective. Finally, the results from algorithms are compared using different factors such as accuracy score, analyzing the confusion matrix etc.

3. Overview of dataset

The source of this dataset is https://archive.ics.uci.edu/dataset/73/mushroom. This data set contains descriptions of hypothetical samples belonging to 23 Agaricus and Lepiota Family gilled mushrooms (pp. 500-525). Each species is labeled as either definitely edible, definitely poisonous, or maybe edible but not recommended. This latter category was integrated with the toxic category. The Guide explains unequivocally that there is no easy rule for judging mushroom edibility; no rule like "leaflets three, let it be" for Poisonous Oak and Ivy. The dataset have 8124 instances and 23 features. Below table describes the independent attributes.

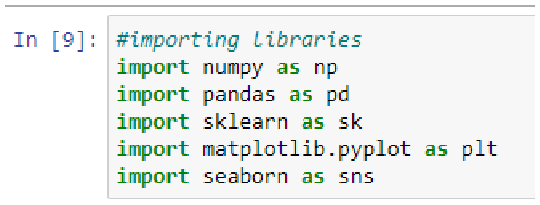
Attributes	**Description**
cap-shape	bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
cap-surface	fibrous=f,grooves=g,scaly=y,smooth=s
cap-color	brown=n,buff=b,cinnamon=c,gray=g,green=r, pink=p,purple=u,red=e,white=w,yellow=y
bruises?	bruises=t,no=f
odor	almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s
Gill-attachment	attached=a,descending=d,free=f,notched=n
gill-spacing	close=c,crowded=w,distant=d
gill-size	broad=b,narrow=n
gill-color	black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,white=w,yellow=y
stalk-shape	enlarging=e,tapering=t
stalk-root	bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,rooted=r,missing=?
stalk-surface-above-ring	fibrous=f,scaly=y,silky=k,smooth=s
stalk-surface-below-ring	fibrous=f,scaly=y,silky=k,smooth=s
stalk-color-above-ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
stalk-color-below-ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
veil-type	partial=p,universal=u
veil-color	brown=n,orange=o,white=w,yellow=y

ring-number	none=n,one=o,two=t
ring-type	cobwebby=c,evanescent=e,flaring=f,large=l, none=n,pendant=p,sheathing=s,zone=z
spore-print-color	black=k,brown=n,buff=b,chocolate=h,green=r, orange=o,purple=u,white=w,yellow=y
population	abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y
habitat	grasses=g,leaves=l,meadows=m,paths=p, urban=u,waste=w,woods=d

The **target feature** is “class” and it has two elements in it edible=e and poisonous=p.

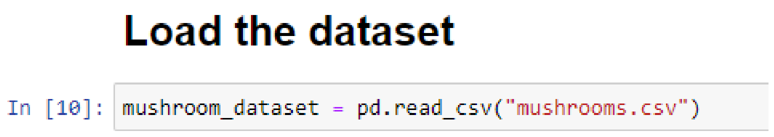
4. Exploratory Data Analysis

Importing necessary libraries: Below figure shows the necessary python libraries that we require to perform our classification task. We will make use some of the popular python libraries like pandas, numpy, sci-kit learn, matplotlib and seaborn. Apart from these we will also require some libraries for our classification which we will see in detail in the later sections.



```
In [9]: #importing libraries
        import numpy as np
        import pandas as pd
        import sklearn as sk
        import matplotlib.pyplot as plt
        import seaborn as sns
```

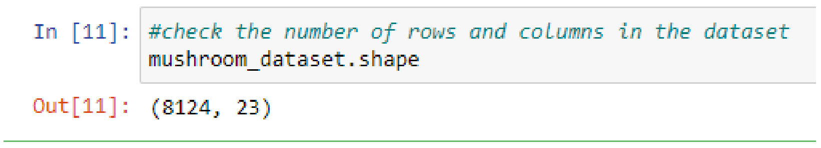
Loading the dataset: using pandas’ read_csv function, we load the dataset of csv format for analysis.



Load the dataset

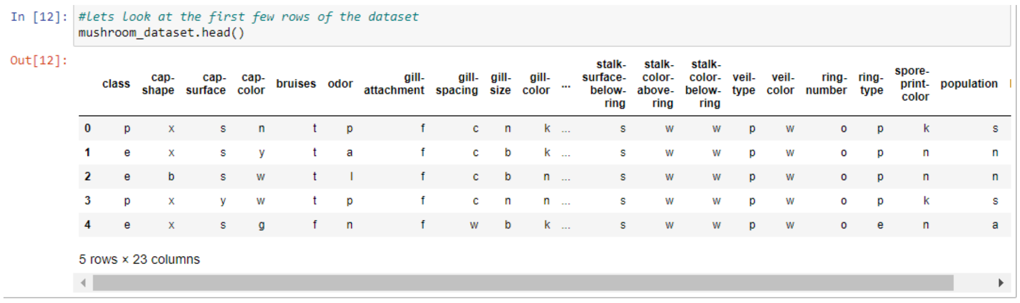
```
In [10]: mushroom_dataset = pd.read_csv("mushrooms.csv")
```

Dimensions of dataset: shape is used to check the dimensions of dataset. This returns a tuple with the number of rows and columns that the dataset has. Our current dataset has 8124 rows and 23 columns.



```
In [11]: #check the number of rows and columns in the dataset
         mushroom_dataset.shape

Out[11]: (8124, 23)
```

Displaying the first few rows of dataset: This is done by using the head() function which print the first 5 rows in the dataset. In the current dataset the target feature column is the first column.



```
In [12]: #lets look at the first few rows of the dataset
         mushroom_dataset.head()
Out[12]:
```

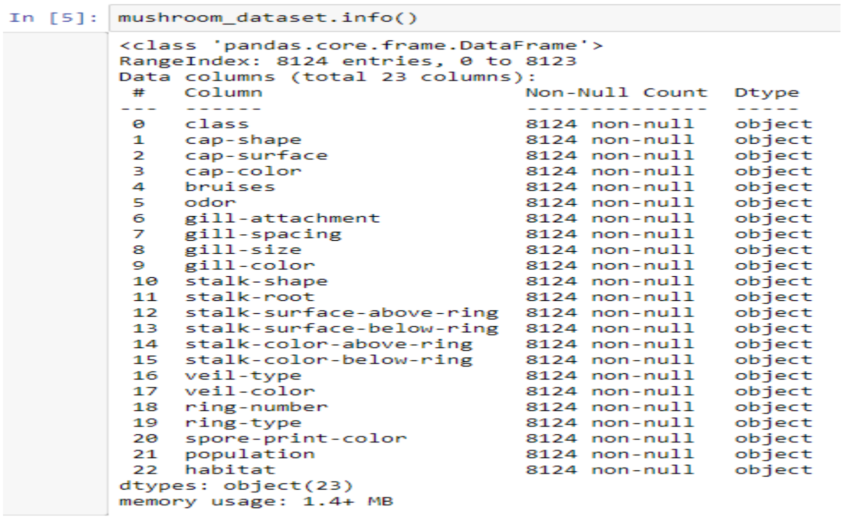
	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a

5 rows × 23 columns

The target feature 'class' is the first column in this dataset.

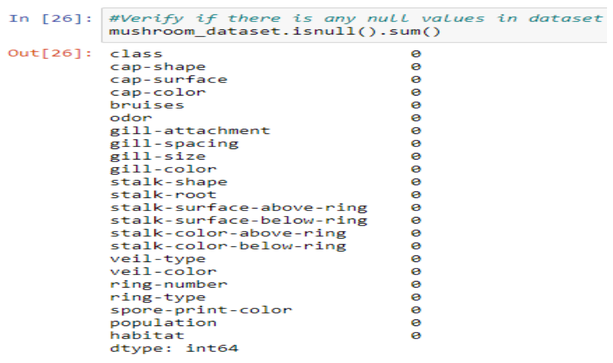
<u>Summary of the dataframe:</u> The info() method is used to print the concise summary of the dataframe. From the below figure we can interpret the following

- All the features in the dataset consists of categorical values
- Since all the feature columns have categorical values in it, we will not run a describe method to know the five-point summary of our dataset.



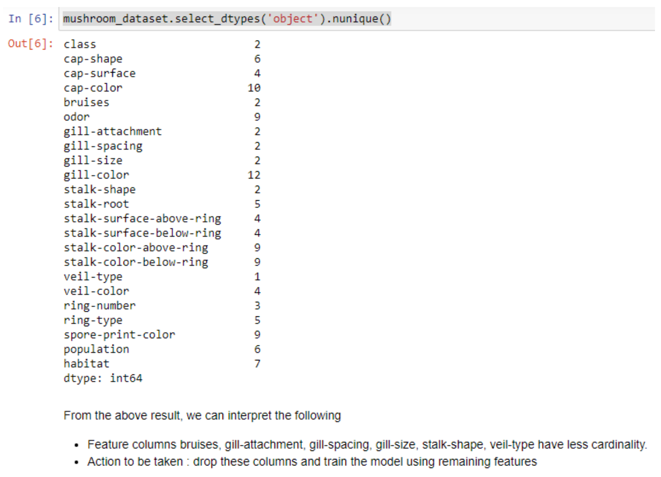
```
In [5]: mushroom_dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   class                     8124 non-null   object
 1   cap-shape                 8124 non-null   object
 2   cap-surface               8124 non-null   object
 3   cap-color                 8124 non-null   object
 4   bruises                   8124 non-null   object
 5   odor                      8124 non-null   object
 6   gill-attachment           8124 non-null   object
 7   gill-spacing              8124 non-null   object
 8   gill-size                 8124 non-null   object
 9   gill-color                8124 non-null   object
 10  stalk-shape               8124 non-null   object
 11  stalk-root                8124 non-null   object
 12  stalk-surface-above-ring  8124 non-null   object
 13  stalk-surface-below-ring  8124 non-null   object
 14  stalk-color-above-ring    8124 non-null   object
 15  stalk-color-below-ring    8124 non-null   object
 16  veil-type                 8124 non-null   object
 17  veil-color                8124 non-null   object
 18  ring-number               8124 non-null   object
 19  ring-type                 8124 non-null   object
 20  spore-print-color         8124 non-null   object
 21  population                8124 non-null   object
 22  habitat                   8124 non-null   object
dtypes: object(23)
memory usage: 1.4+ MB
```

<u>Check for missing values:</u> isnull() is used to check if there is any null values in each of our column. Sum() counts the number of occurrences of null value in each of these column. From the below figure it is clear that there are no null values in our dataset.



```
In [26]: #Verify if there is any null values in dataset
         mushroom_dataset.isnull().sum()

Out[26]: class                       0
         cap-shape                   0
         cap-surface                 0
         cap-color                   0
         bruises                     0
         odor                        0
         gill-attachment             0
         gill-spacing                0
         gill-size                   0
         gill-color                  0
         stalk-shape                 0
         stalk-root                  0
         stalk-surface-above-ring    0
         stalk-surface-below-ring    0
         stalk-color-above-ring      0
         stalk-color-below-ring      0
         veil-type                   0
         veil-color                  0
         ring-number                 0
         ring-type                   0
         spore-print-color           0
         population                  0
         habitat                     0
         dtype: int64
```

Checking for cardinality: The number of distinct values in a column or feature of a dataset is referred to as its cardinality. For the current dataset I will drop all those features with cardinality less than 3 since they won't be of much importance for the analysis. Below figure shows the result of the cardinality.

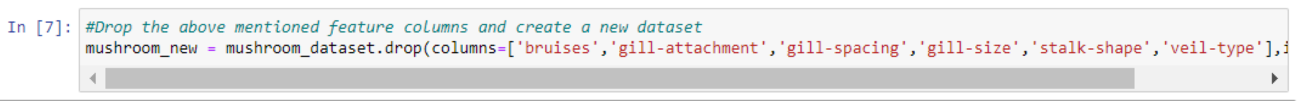


```
In [6]: mushroom_dataset.select_dtypes('object').nunique()

Out[6]: class                        2
        cap-shape                    6
        cap-surface                  4
        cap-color                   10
        bruises                      2
        odor                         9
        gill-attachment              2
        gill-spacing                 2
        gill-size                    2
        gill-color                  12
        stalk-shape                  2
        stalk-root                   5
        stalk-surface-above-ring     4
        stalk-surface-below-ring     4
        stalk-color-above-ring       9
        stalk-color-below-ring       9
        veil-type                    1
        veil-color                   4
        ring-number                  3
        ring-type                    5
        spore-print-color            9
        population                   6
        habitat                      7
        dtype: int64
```

From the above result, we can interpret the following

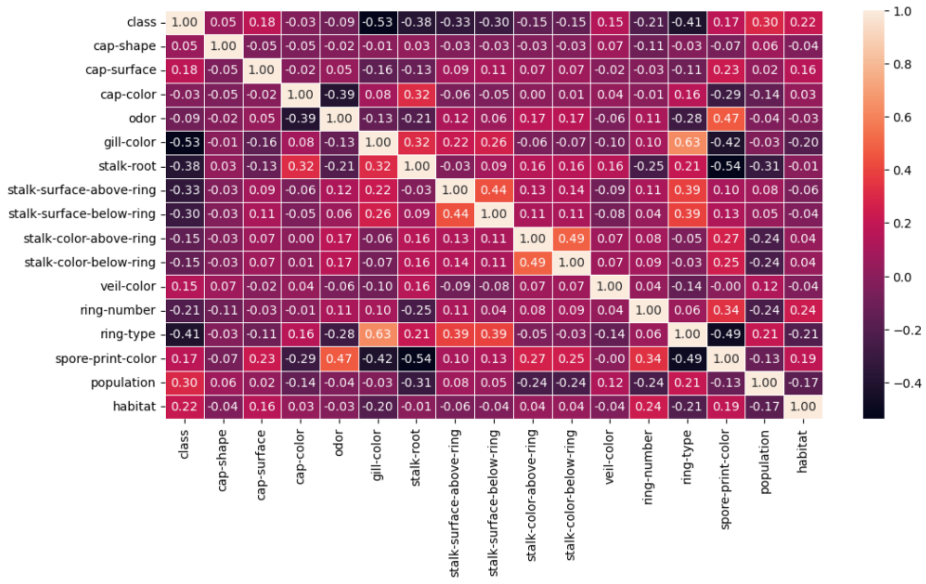
- Feature columns bruises, gill-attachment, gill-spacing, gill-size, stalk-shape, veil-type have less cardinality.
- Action to be taken : drop these columns and train the model using remaining features

Based on the result from above figure we can observe that few features like bruises, gill-attachment, gill-spacing, gill-size, stalk-shape, veil-type have less cardinality. We will drop these features. To achieve this, we will make use of drop(). This method will drop the columns specified inside it. This is shown in below figure. The resultant dataset is stored as a new dataframe, mushroom_new.



```
In [7]: #Drop the above mentioned feature columns and create a new dataset
        mushroom_new = mushroom_dataset.drop(columns=['bruises','gill-attachment','gill-spacing','gill-size','stalk-shape','veil-type'],i
```

Correlation matrix: Figure below shows the correlation matrix visualized using the heatmap. Heatmap is a plotting technique in the seaborn library that is used to interpret the correlation. There is only a moderate correlation between our target variable and other feature variable. However, the negative correlation is bit high when compared to the positive ones. Gill color, stalk root, stalk surface above ring, stalk surface below ring and ring type has a higher negative correlation compared to other negatively correlated ones.

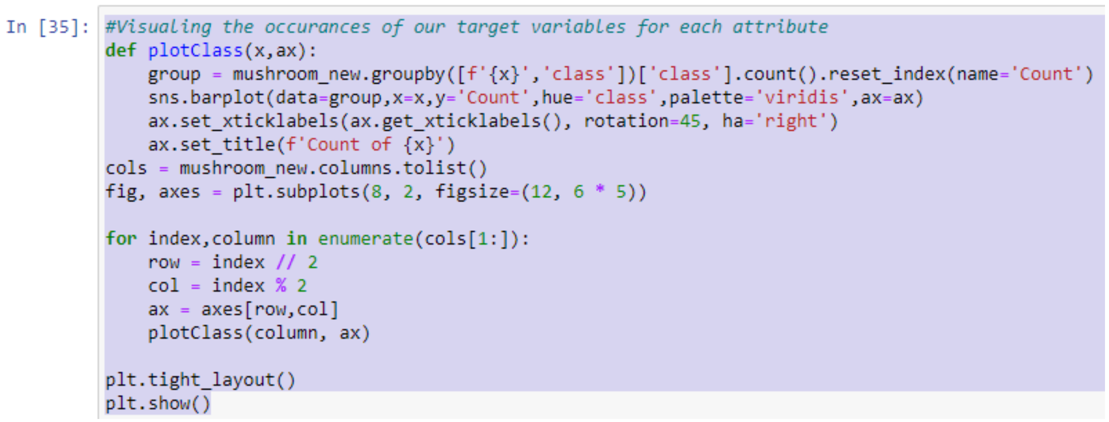


	class	cap-shape	cap-surface	cap-color	odor	gill-color	stalk-root	stalk-surface-above-ring	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-color	ring-number	ring-type	spore-print-color	population	habitat
class	1.00	0.05	0.18	-0.03	-0.09	-0.53	-0.38	-0.33	-0.30	-0.15	-0.15	0.15	-0.21	-0.41	0.17	0.30	0.22
cap-shape	0.05	1.00	-0.05	-0.05	-0.02	-0.01	0.03	-0.03	-0.03	-0.03	-0.03	0.07	-0.11	-0.03	-0.07	0.06	-0.04
cap-surface	0.18	-0.05	1.00	-0.02	0.05	-0.16	-0.13	0.09	0.11	0.07	0.07	-0.02	-0.03	-0.11	0.23	0.02	0.16
cap-color	-0.03	-0.05	-0.02	1.00	-0.39	0.08	0.32	-0.06	-0.05	0.00	0.01	0.04	-0.01	0.16	-0.29	-0.14	0.03
odor	-0.09	-0.02	0.05	-0.39	1.00	-0.13	-0.21	0.12	0.06	0.17	0.17	-0.06	0.11	-0.28	0.47	-0.04	-0.03
gill-color	-0.53	-0.01	-0.16	0.08	-0.13	1.00	0.32	0.22	0.26	-0.06	-0.07	-0.10	0.10	0.63	-0.42	-0.03	-0.20
stalk-root	-0.38	0.03	-0.13	0.32	-0.21	0.32	1.00	-0.03	0.09	0.16	0.16	0.16	-0.25	0.21	-0.54	-0.31	-0.01
stalk-surface-above-ring	-0.33	-0.03	0.09	-0.06	0.12	0.22	-0.03	1.00	0.44	0.13	0.14	-0.09	0.11	0.39	0.10	0.08	-0.06
stalk-surface-below-ring	-0.30	-0.03	0.11	-0.05	0.06	0.26	0.09	0.44	1.00	0.11	0.11	-0.08	0.04	0.39	0.13	0.05	-0.04
stalk-color-above-ring	-0.15	-0.03	0.07	0.00	0.17	-0.06	0.16	0.13	0.11	1.00	0.49	0.07	0.08	-0.05	0.27	-0.24	0.04
stalk-color-below-ring	-0.15	-0.03	0.07	0.01	0.17	-0.07	0.16	0.14	0.11	0.49	1.00	0.07	0.09	-0.03	0.25	-0.24	0.04
veil-color	0.15	0.07	-0.02	0.04	-0.06	-0.10	0.16	-0.09	-0.08	0.07	0.07	1.00	0.04	-0.14	-0.00	0.12	-0.04
ring-number	-0.21	-0.11	-0.03	-0.01	0.11	0.10	-0.25	0.11	0.04	0.08	0.09	0.04	1.00	0.06	0.34	-0.24	0.24
ring-type	-0.41	-0.03	-0.11	0.16	-0.28	0.63	0.21	0.39	0.39	-0.05	-0.03	-0.14	0.06	1.00	-0.49	0.21	-0.21
spore-print-color	0.17	-0.07	0.23	-0.29	0.47	-0.42	-0.54	0.10	0.13	0.27	0.25	-0.00	0.34	-0.49	1.00	-0.13	0.19
population	0.30	0.06	0.02	-0.14	-0.04	-0.03	-0.31	0.08	0.05	-0.24	-0.24	0.12	-0.24	0.21	-0.13	1.00	-0.17
habitat	0.22	-0.04	0.16	0.03	-0.03	-0.20	-0.01	-0.06	-0.04	0.04	0.04	-0.04	0.24	-0.21	0.19	-0.17	1.00

<u>Visualizations:</u>

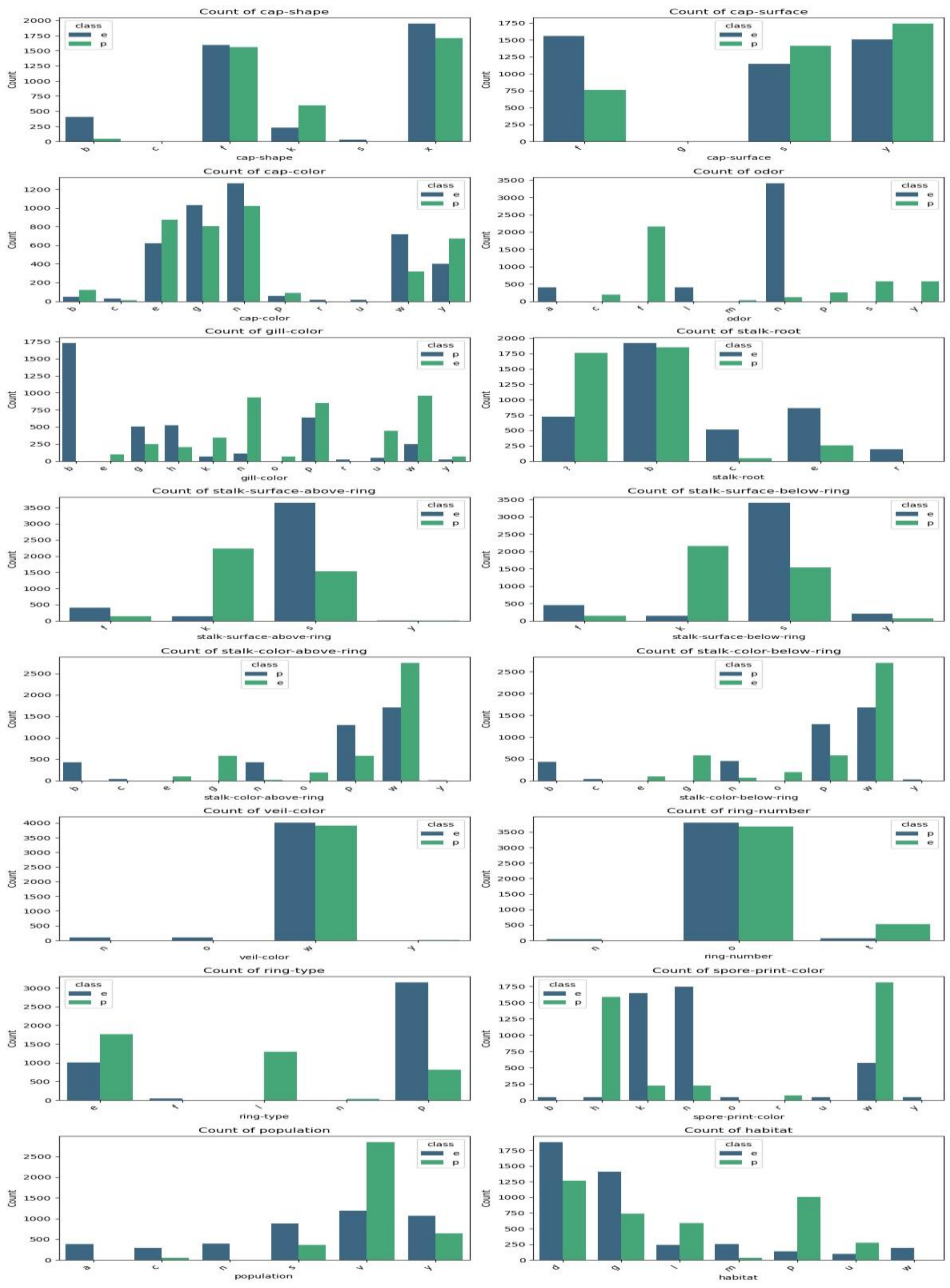
Figure below shows a series of bar graphs that depict the distribution of the target variable ('class' - whether a mushroom is edible or poisonous) for each category feature in the dataset. For this we make use of the seaborn’s barplot function with y axis displaying the count of occurances of the target variables for each element in every feature column.

A for loop is used to iterate through each categorical attribute in the dataframe. For each attribute, it calls the plotClass function that generate a barplot on the corresponding subplot. plt.tight_layout() is used to improve the appearance of the layout by preventing subplot overlaps. Finally, plt.show() is used to display the plots created.



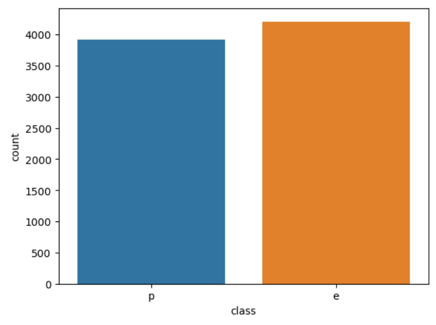
```
In [35]: #Visualing the occurances of our target variables for each attribute
         def plotClass(x,ax):
             group = mushroom_new.groupby([f'{x}','class'])['class'].count().reset_index(name='Count')
             sns.barplot(data=group,x=x,y='Count',hue='class',palette='viridis',ax=ax)
             ax.set_xticklabels(ax.get_xticklabels(), rotation=45, ha='right')
             ax.set_title(f'Count of {x}')
         cols = mushroom_new.columns.tolist()
         fig, axes = plt.subplots(8, 2, figsize=(12, 6 * 5))

         for index,column in enumerate(cols[1:]):
             row = index // 2
             col = index % 2
             ax = axes[row,col]
             plotClass(column, ax)

         plt.tight_layout()
         plt.show()
```

Figure below shows how the distribution of edible and deadly mushrooms vary for each categorical attribute in the dataset, revealing probable correlations between attributes and the target variable. From the plots it can be inferred that most of the mushrooms are edible for consuming.



Check for class imbalance: We use a countplot to visualize and check if the target feature column is balanced or not. From the below figure it can be interpreted that our target class is pretty much balanced since the imbalance is pretty minimal and we can proceed without any oversampling or undersampling methods.

```
In [37]: #To check if there is any imbalances in the target feature column
         sns.countplot(data = mushroom_new, x="class")

Out[37]: <AxesSubplot:xlabel='class', ylabel='count'>
```



5. Model Building

Splitting dataset:

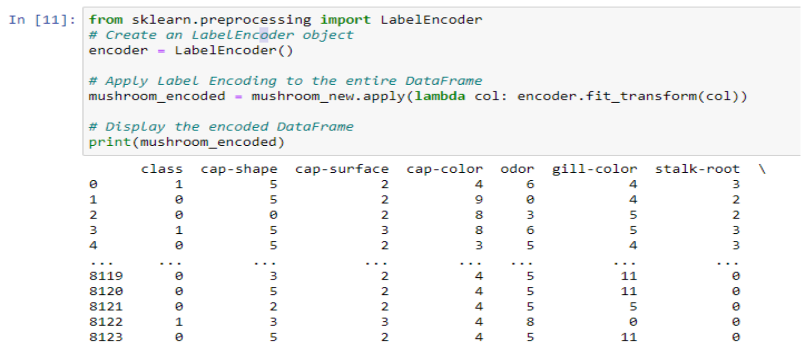
Before splitting the dataset, we need to segregate the independent and target columns. In the below code, we are selecting the features with more negative collinearity as our independent variables. The target column is placed in feature matrix y.

```
In [38]: # Arrange data into independent variables and target variables
         X = mushroom_encoded.loc[: , ['cap-shape','stalk-surface-above-ring', 'habitat', 'spore-print-color']]
         y=mushroom_encoded['class'] #is the target
```

Below code use the train_test_split function of scikit-learn for splitting the datasets to train and test sets. Here 30% of data will be used for testing and rest is training datasets. Value placed in random state parameter will help us ensure that we will get the same split when we use the same value in future. Training set(X_train and y_train) is used to train the machine learning model and Test set(X_test and y_test) is used to evaluate the performance.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=10)
```

Label Encoding: Label encoding is analogous to providing a dictionary to a computer that converts human language into a language that the computer understands. It's a basic yet efficient strategy for making machine learning algorithms more versatile and effective. In our dataset all the columns have categorical variables, so before training the model it is important to convert into numerical so that the algorithm can handle and process the data uniformly throughout the modeling process. mushroom_new.apply(lambda col: encoder.fit_transform(col)): Using the apply function to apply label encoding to each column (col) in the DataFrame mushroom_new. The lambda function uses the LabelEncoder's fit_transform method to each column independently.



```
In [11]: from sklearn.preprocessing import LabelEncoder
         # Create an LabelEncoder object
         encoder = LabelEncoder()

         # Apply Label Encoding to the entire DataFrame
         mushroom_encoded = mushroom_new.apply(lambda col: encoder.fit_transform(col))

         # Display the encoded DataFrame
         print(mushroom_encoded)
```

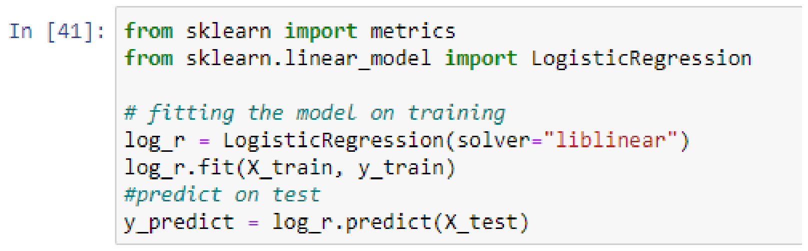
```
      class  cap-shape  cap-surface  cap-color  odor  gill-color  stalk-root  \
0         1          5            2          4     6           4           3
1         0          5            2          9     0           4           2
2         0          0            2          8     3           5           2
3         1          5            3          8     6           5           3
4         0          5            2          3     5           4           3
...     ...        ...          ...        ...   ...         ...         ...
8119      0          3            2          4     5          11           0
8120      0          5            2          4     5          11           0
8121      0          2            2          4     5           5           0
8122      1          3            3          4     8           0           0
8123      0          5            2          4     5          11           0
```

Classification model1: Logistic Regression:

Below code shows the implementation of the logistic regression model using scikit-learn's LogisticRegression library. The solver option in scikit-learn's LogisticRegression class specifies the optimization algorithm to be utilized during logistic regression model training. We are using the liblinear value for solver since this value is basically used for small or mredium datasets.

In machine learning, "Fit (Train) the Model" entails training the model with a dataset to understand the patterns and correlations in the data. Here fit method is called on the logistic regression model with the training data.

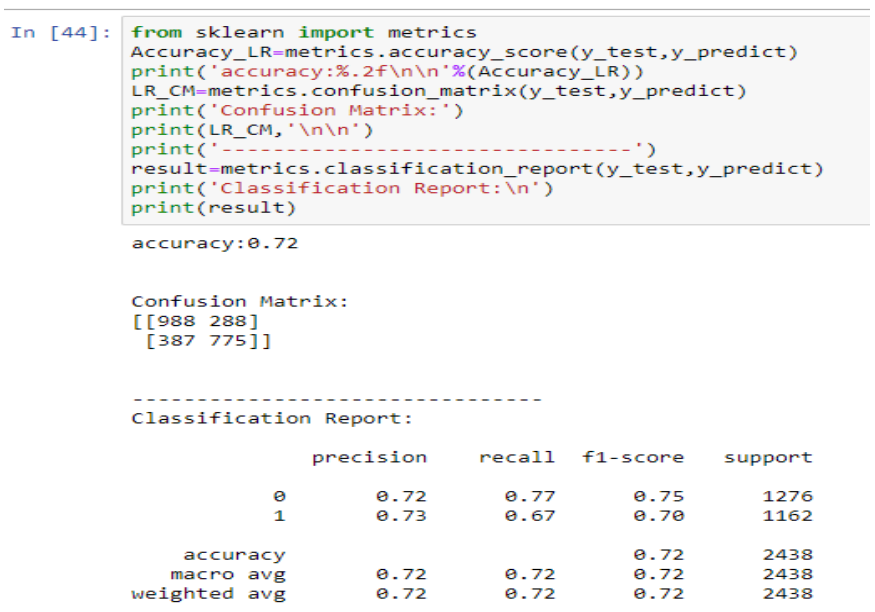
Finally, the trained model is used to make predictions on the test data. This is done by the predict().



```
In [41]: from sklearn import metrics
         from sklearn.linear_model import LogisticRegression

         # fitting the model on training
         log_r = LogisticRegression(solver="liblinear")
         log_r.fit(X_train, y_train)
         #predict on test
         y_predict = log_r.predict(X_test)
```

Below figure shows the code to detect the accuracy of how correctly our model has predicted the actual labels in test data. Then a confusion matrix is drawn to show the number of correct and wrong predictions against the actual data. Finally, it calculates the classification report for each class, which contains precision, recall, F1-score, and support.

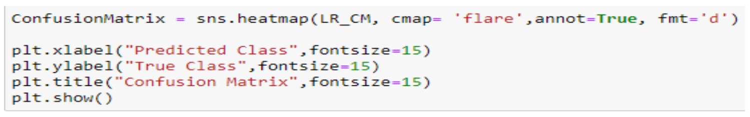
The overall accuracy of the model is 72% which means it correctly predicted the class for 72% of the instances in the test set. From confusion matrix we can interpret that True Positive is 775 i.e., the model predicted edible mushrooms as edible correctly.True Negative is 988 which means the model could predict 988 poisonous mushrooms correctly. Precision for 1 is 72% which means 72% of edible predictions were actually edible. Recall for edible is 67% which means that 67% of edible mushrooms were correctly predicted.



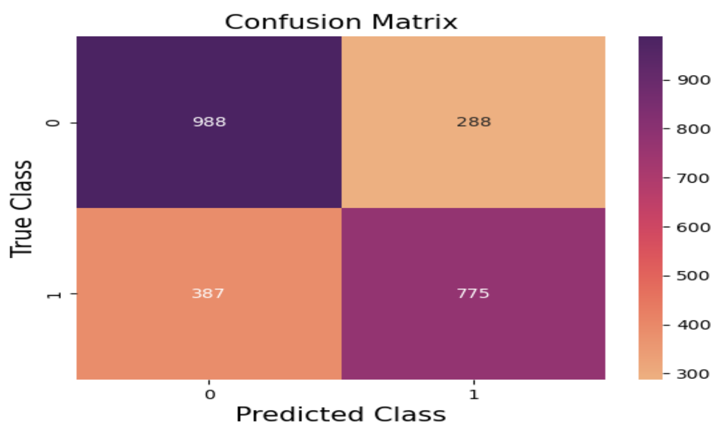
```
In [44]: from sklearn import metrics
         Accuracy_LR=metrics.accuracy_score(y_test,y_predict)
         print('accuracy:%.2f\n\n'%(Accuracy_LR))
         LR_CM=metrics.confusion_matrix(y_test,y_predict)
         print('Confusion Matrix:')
         print(LR_CM,'\n\n')
         print('----------------------------------')
         result=metrics.classification_report(y_test,y_predict)
         print('Classification Report:\n')
         print(result)

         accuracy:0.72


         Confusion Matrix:
         [[988 288]
          [387 775]]


         --------------------------------
         Classification Report:

                       precision    recall  f1-score   support

                    0       0.72      0.77      0.75      1276
                    1       0.73      0.67      0.70      1162

             accuracy                           0.72      2438
            macro avg       0.72      0.72      0.72      2438
         weighted avg       0.72      0.72      0.72      2438
```

Confusion Matrix: Figure below shows the code for plotting the confusion matrix using seaborn's heatmap. From the confusion matrix the following can be interpreted:

- True Positives (TP): 775
- True Negatives (TN): 988
- False Positives (FP): 288
- False Negatives (FN): 387

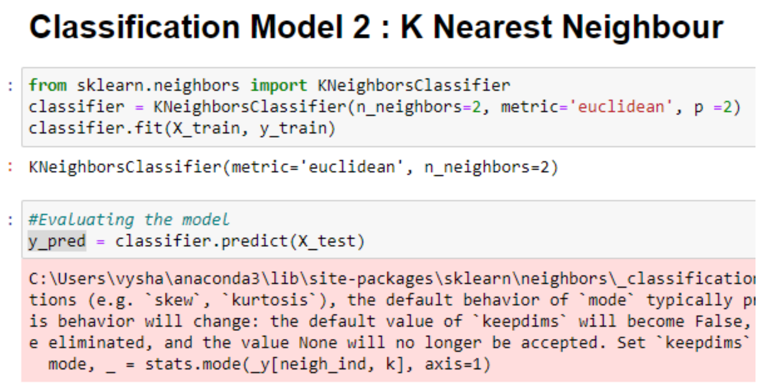


```
ConfusionMatrix = sns.heatmap(LR_CM, cmap= 'flare',annot=True, fmt='d')

plt.xlabel("Predicted Class",fontsize=15)
plt.ylabel("True Class",fontsize=15)
plt.title("Confusion Matrix",fontsize=15)
plt.show()
```



Classification model2: KNeighbours Classifier:

The KNeighborsClassifier is a scikit-learn classification algorithm that implements the k-nearest neighbor's approach. Below code shows the implementation of the KNN model using scikit-learn's KNeighborsClassifier library. In the KNeighborsClassifier method we assign the number of neighbours to 2 which means that the algorithm will consider 2 neighbours while making predictions. The distance metric used here is Euclidean hence the p value is set to 2. Then we fit the model on the training dataset using fit(). Finally, the predict() predicts the labels for a test dataset (X_test) after training and keeps the predictions in y_pred.



Classification Model 2 : K Nearest Neighbour

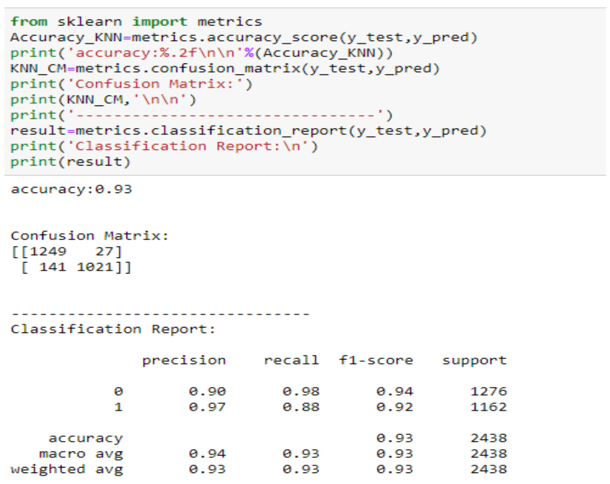
```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=2, metric='euclidean', p =2)
classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier(metric='euclidean', n_neighbors=2)
```

```
#Evaluating the model
y_pred = classifier.predict(X_test)
```

```
C:\Users\vysha\anaconda3\lib\site-packages\sklearn\neighbors\_classificatio
tions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically pr
is behavior will change: the default value of `keepdims` will become False,
e eliminated, and the value None will no longer be accepted. Set `keepdims`
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

Below figure shows the code to detect the accuracy of how correctly our model has predicted the actual labels in test data. Then a confusion matrix is displayed to show the number of correct and wrong predictions against the actual data. Finally, it calculates the classification report for each class, which contains precision, recall, F1-score, and support.
The overall accuracy of the model is 93% which means it correctly predicted the class for 93% of the instances in the test set. From confusion matrix we can interpret that True Positive is 1021 i.e., the model predicted edible mushrooms as edible correctly. True Negative is 1249 which means the model could predict 1249 poisonous mushrooms correctly.
Precision for 1 is 97% which means 97% of edible predictions were actually edible. Recall for edible is 92% which means that 92% of edible mushrooms were correctly predicted.



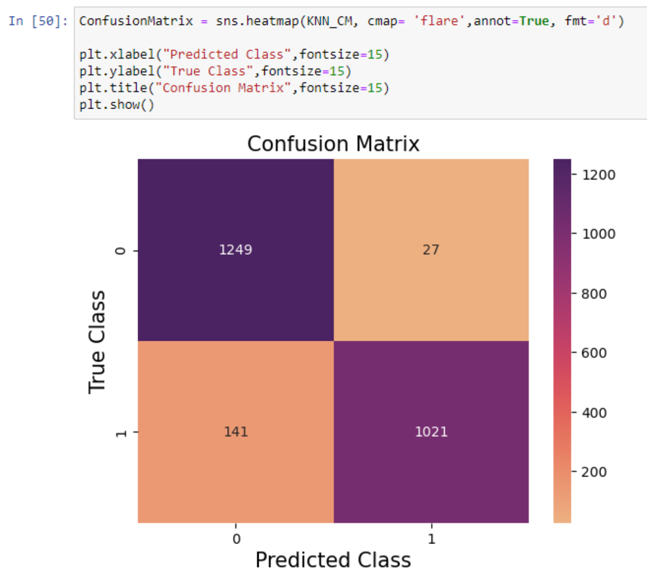
```
from sklearn import metrics
Accuracy_KNN=metrics.accuracy_score(y_test,y_pred)
print('accuracy:%.2f\n\n'%(Accuracy_KNN))
KNN_CM=metrics.confusion_matrix(y_test,y_pred)
print('Confusion Matrix:')
print(KNN_CM,'\n\n')
print('-------------------------------')
result=metrics.classification_report(y_test,y_pred)
print('Classification Report:\n')
print(result)
```

```
accuracy:0.93


Confusion Matrix:
[[1249   27]
 [ 141 1021]]


-------------------------------
Classification Report:

              precision    recall  f1-score   support

           0       0.90      0.98      0.94      1276
           1       0.97      0.88      0.92      1162

    accuracy                           0.93      2438
   macro avg       0.94      0.93      0.93      2438
weighted avg       0.93      0.93      0.93      2438
```

Confusion Matrix: Figure below shows the code for plotting the confusion matrix using seaborn’s heatmap. From the confusion matrix the following can be interpreted:

- True Positives (TP): 1021

- True Negatives (TN): 1249
- False Positives (FP): 27
- False Negatives (FN): 141

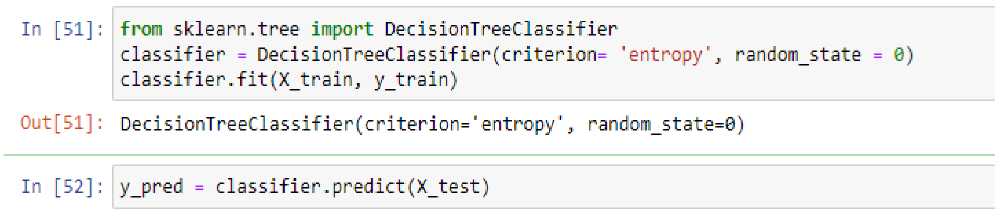


```
In [50]: ConfusionMatrix = sns.heatmap(KNN_CM, cmap= 'flare',annot=True, fmt='d')

         plt.xlabel("Predicted Class",fontsize=15)
         plt.ylabel("True Class",fontsize=15)
         plt.title("Confusion Matrix",fontsize=15)
         plt.show()
```

Classification model2: Decision Tree Classifier:

The DecisionTreeClassifier is a scikit-learn classification algorithm that implements the decision tree algorithm for classification. Below code shows the implementation of the Decision tree model using scikit-learn’s DecisionTreeClassifier library.

The criterion used here is entropy. Entropy is a measure of a node's disorder or impurity. As a result, a node with a more variable composition, such as 2Pass and 2Fail, is thought to have higher Entropy than a node with only pass or only fail. Then we fit the model on the training dataset using fit(). Finally, the predict() predicts the labels for a test dataset (X_test) after training and keeps the predictions in y_pred.



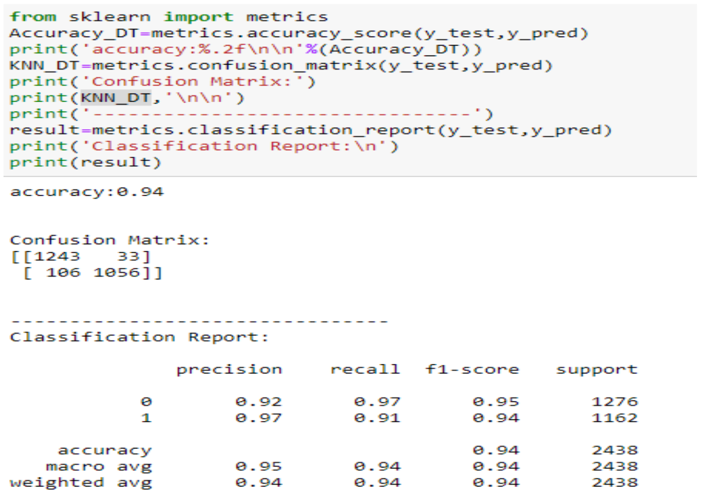
```
In [51]: from sklearn.tree import DecisionTreeClassifier
         classifier = DecisionTreeClassifier(criterion= 'entropy', random_state = 0)
         classifier.fit(X_train, y_train)

Out[51]: DecisionTreeClassifier(criterion='entropy', random_state=0)

In [52]: y_pred = classifier.predict(X_test)
```

Below figure shows the code to detect the accuracy of how correctly our model has predicted the actual labels in test data. Then a confusion matrix is displayed to show the number of correct and wrong predictions against the actual data. Finally, it calculates the classification report for each class, which contains precision, recall, F1-score, and support.

The overall accuracy of the model is 94% which means it correctly predicted the class for 94% of the instances in the test set. From confusion matrix we can interpret that True Positive is 1056 i.e., the model predicted edible mushrooms as edible correctly. True

Negative is 1243 which means the model could predict 1243 poisonous mushrooms correctly.

Precision for 1 is 97% which means 97% of edible predictions were actually edible. Recall for edible is 91% which means that 91% of edible mushrooms were correctly predicted.

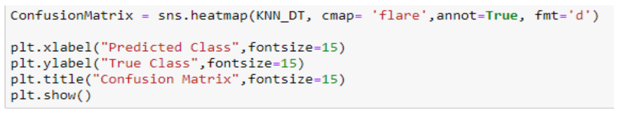


```
from sklearn import metrics
Accuracy_DT=metrics.accuracy_score(y_test,y_pred)
print('accuracy:%.2f\n\n'%(Accuracy_DT))
KNN_DT=metrics.confusion_matrix(y_test,y_pred)
print('Confusion Matrix:')
print(KNN_DT,'\n\n')
print('--------------------------------')
result=metrics.classification_report(y_test,y_pred)
print('Classification Report:\n')
print(result)
```

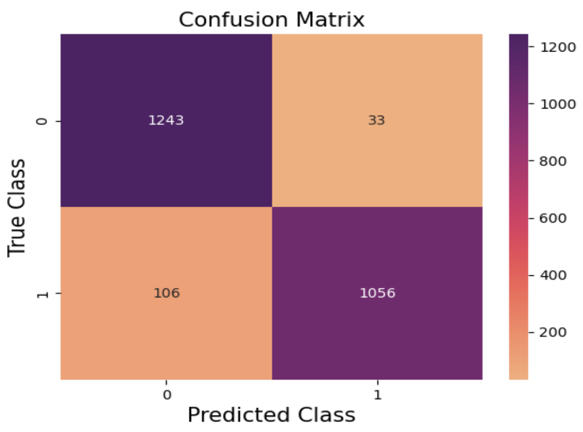
```
accuracy:0.94


Confusion Matrix:
[[1243   33]
 [ 106 1056]]


--------------------------------
Classification Report:

              precision    recall  f1-score   support

           0       0.92      0.97      0.95      1276
           1       0.97      0.91      0.94      1162

    accuracy                           0.94      2438
   macro avg       0.95      0.94      0.94      2438
weighted avg       0.94      0.94      0.94      2438
```

Confusion Matrix: Figure below shows the code for plotting the confusion matrix using seaborn’s heatmap. From the confusion matrix the following can be interpreted:

- True Positives (TP): 1056
- True Negatives (TN): 1243
- False Positives (FP): 33
- False Negatives (FN): 106

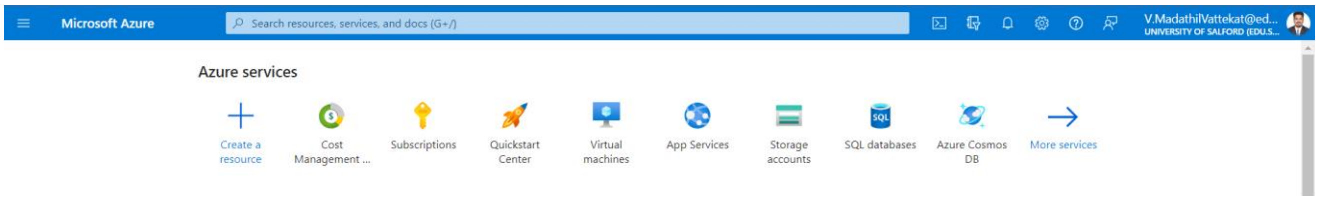


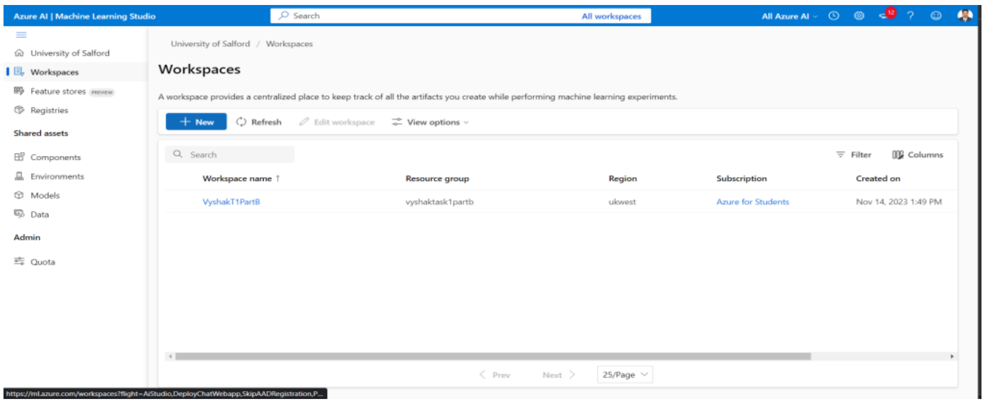
```
ConfusionMatrix = sns.heatmap(KNN_DT, cmap= 'flare',annot=True, fmt='d')

plt.xlabel("Predicted Class",fontsize=15)
plt.ylabel("True Class",fontsize=15)
plt.title("Confusion Matrix",fontsize=15)
plt.show()
```



PART B: Machine Learning using Azure

After creating an Azure Machine Learning Resource, fill in the necessary settings and click on create to get the workspace up and running.





University of Salford / Workspaces

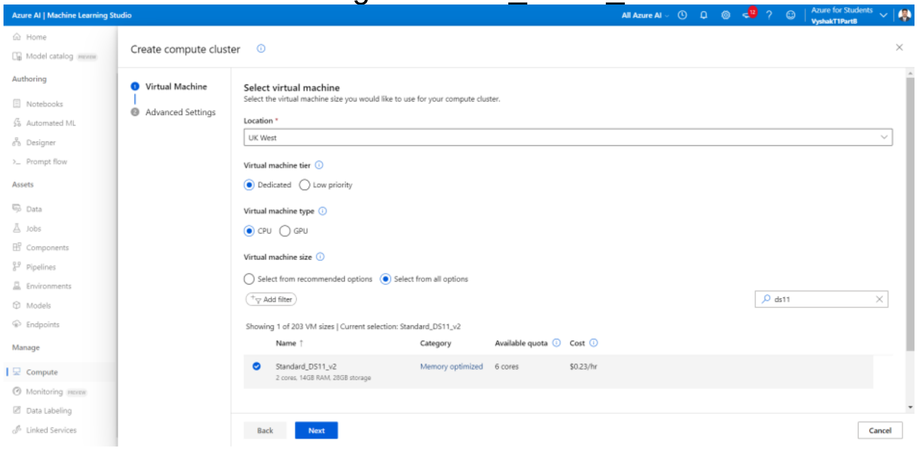
Workspaces

A workspace provides a centralized place to keep track of all the artifacts you create while performing machine learning experiments.

Workspace name ↑	Resource group	Region	Subscription	Created on
VyshakT1PartB	vyshaktask1partb	ukwest	Azure for Students	Nov 14, 2023 1:49 PM

https://ml.azure.com/workspaces?flight=AiStudio,DeployChatWebapp,SkipAADRegistration,P...

A new compute cluster is created using Standard_DS11_v2 virtual machine size



Create compute cluster

Select virtual machine

Select the virtual machine size you would like to use for your compute cluster.

Location *: UK West

Virtual machine tier: [✓ Dedicated] Low priority

Virtual machine type: [✓ CPU] GPU

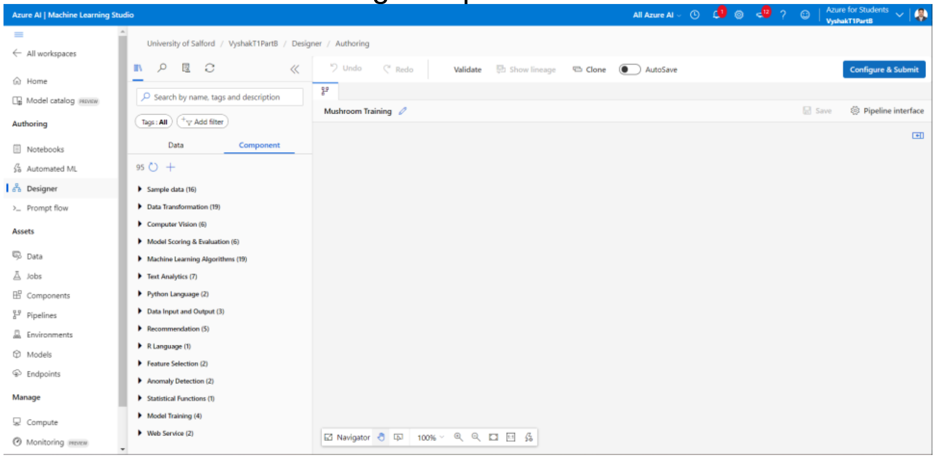
Virtual machine size: Select from recommended options [✓ Select from all options]

ds11

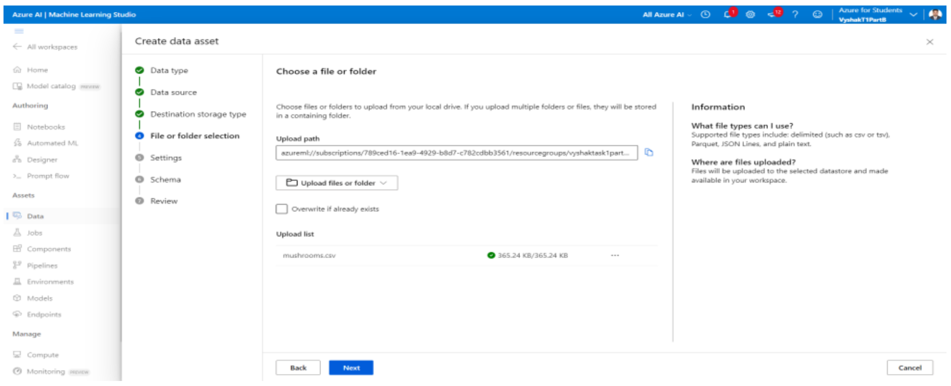
Showing 1 of 203 VM sizes | Current selection: Standard_DS11_v2

Name ↑	Category	Available quota	Cost
✓ Standard_DS11_v2 2 cores, 14GB RAM, 28GB storage	Memory optimized	6 cores	$0.23/hr

A new pipeline is created from the designer option in left menu.



To create a dataset, select data from the Assets menu. After giving the name to the dataset, import the dataset(mushroom.csv) from local files. Destination storage type shall be **workspaceblobstore**. Upload the csv file into Azure. After uploading review the schema and click on create.


Create data asset

- Data type
- Data source
- Destination storage type
- File or folder selection
- Settings
- Schema
- Review

Choose a file or folder

Choose files or folders to upload from your local drive. If you upload multiple folders or files, they will be stored in a containing folder.

Upload path: azureml://subscriptions/789ced16-1ea9-4929-b8d7-c782cdbb3561/resourcegroups/vyshaktask1part...

☐ Overwrite if already exists

Upload list

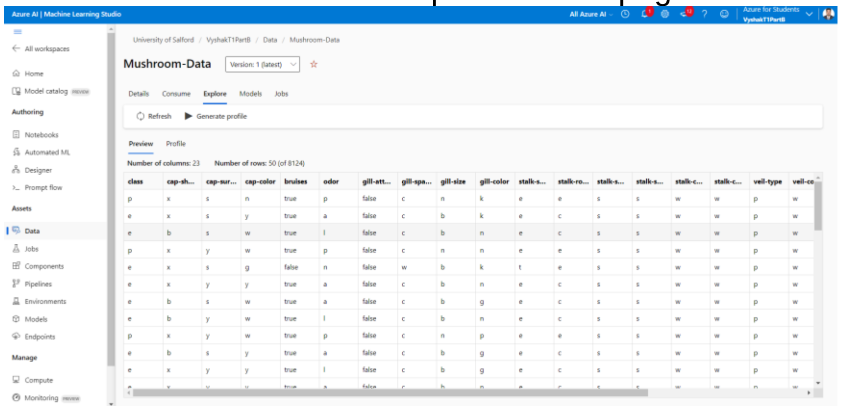
mushrooms.csv	365.24 KB/365.24 KB

Information

What file types can I use?
Supported file types include: delimited (such as csv or tsv), Parquet, JSON Lines, and plain text.

Where are files uploaded?
Files will be uploaded to the selected datastore and made available in your workspace.

The dataset uploaded can be viewed under explore of data page.

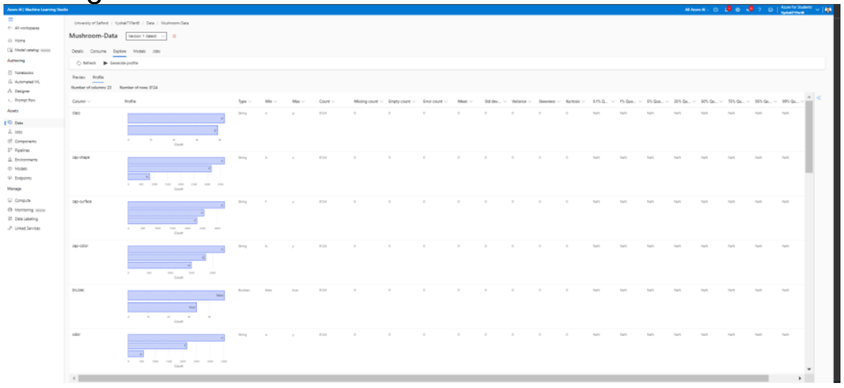

University of Salford / VyshakT1PartB / Data / Mushroom-Data

Mushroom-Data — Version: 1 (latest)

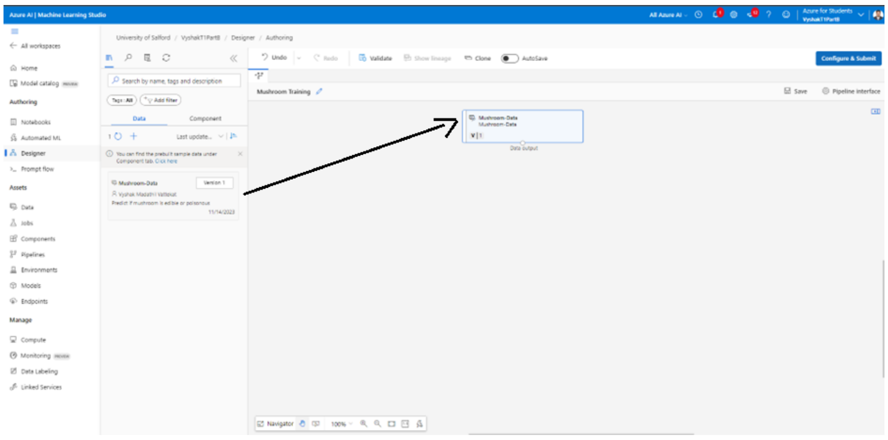
Number of columns: 23 Number of rows: 50 (of 8124)

class	cap-sh...	cap-sur...	cap-color	bruises	odor	gill-att...	gill-spa...	gill-size	gill-color	stalk-s...	stalk-ro...	stalk-s...	stalk-s...	stalk-c...	stalk-c...	veil-type	veil-co
p	x	s	n	true	p	false	c	n	k	e	e	s	s	w	w	p	w
e	x	s	y	true	a	false	c	b	k	e	c	s	s	w	w	p	w
e	b	s	w	true	l	false	c	b	n	e	c	s	s	w	w	p	w
p	x	y	w	true	p	false	c	n	n	e	e	s	s	w	w	p	w
e	x	s	g	false	n	false	w	b	k	t	e	s	s	w	w	p	w
e	x	y	y	true	a	false	c	b	n	e	c	s	s	w	w	p	w
e	b	s	w	true	a	false	c	b	g	e	c	s	s	w	w	p	w
e	b	y	w	true	l	false	c	b	n	e	c	s	s	w	w	p	w
p	x	y	w	true	p	false	c	n	p	e	e	s	s	w	w	p	w
e	b	s	y	true	a	false	c	b	g	e	c	s	s	w	w	p	w
e	x	y	y	true	l	false	c	b	g	e	c	s	s	w	w	p	w

The occurances of target attribute for each feature column can be visualized under profile.

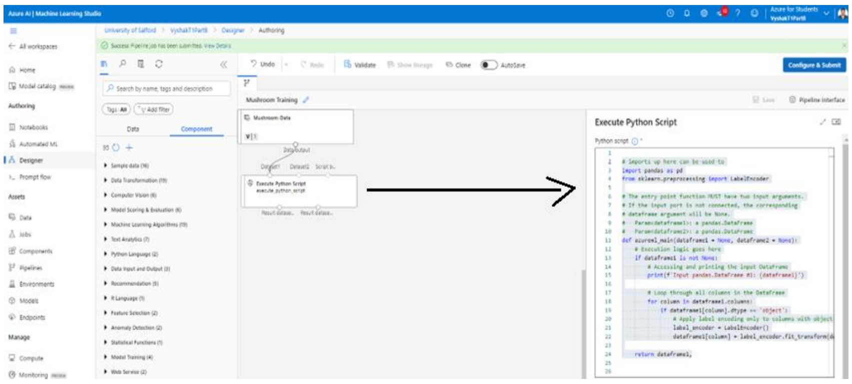


In designer page, load the dataset to canvas by dragging the dataset from data tab to the canvas



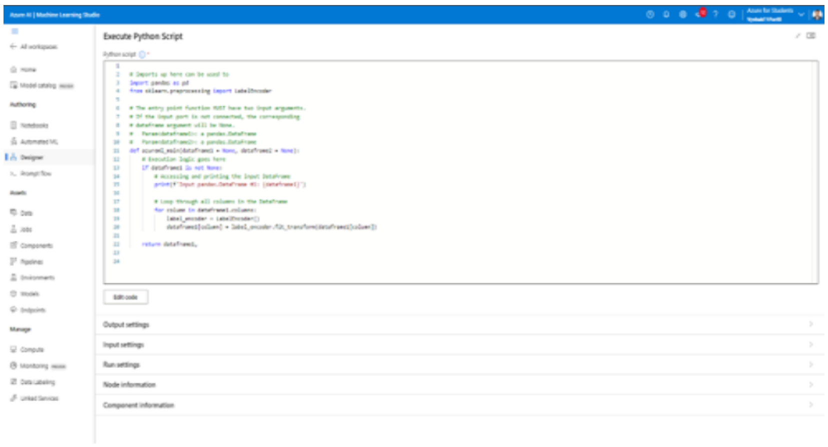
Add Transformations:

1. **Convert to Indicator Values**: Used to convert categorical elements to numerical. We execute the python script using the execute python script component to carry out label encoding.



Execute Python Script

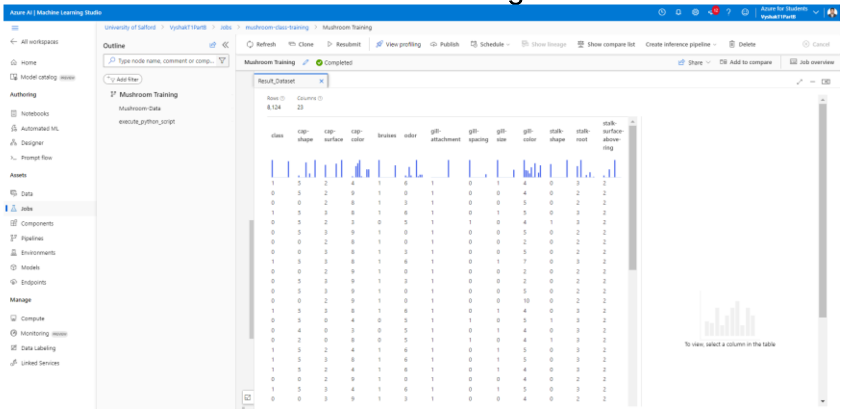
```python

# Imports up here can be used to
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# The entry point function MUST have two input arguments.
# If the input port is not connected, the corresponding
# dataframe argument will be None.
#   Param<dataframe1>: a pandas.DataFrame
#   Param<dataframe2>: a pandas.DataFrame
def azureml_main(dataframe1 = None, dataframe2 = None):
    # Execution logic goes here
    if dataframe1 is not None:
        # Accessing and printing the input DataFrame
        print(f'Input pandas.DataFrame #1: {dataframe1}')

        # Loop through all columns in the DataFrame
        for column in dataframe1.columns:
            if dataframe1[column].dtype == 'object':
                # Apply label encoding only to columns with object
                label_encoder = LabelEncoder()
                dataframe1[column] = label_encoder.fit_transform(d

    return dataframe1,
```



Execute Python Script

```python

# Imports up here can be used to
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# The entry point function MUST have two input arguments.
# If the input port is not connected, the corresponding
# dataframe argument will be None.
#   Param<dataframe1>: a pandas.DataFrame
#   Param<dataframe2>: a pandas.DataFrame
def azureml_main(dataframe1 = None, dataframe2 = None):
    # Execution logic goes here
    if dataframe1 is not None:
        # Accessing and printing the input DataFrame
        print(f"Input pandas.DataFrame #1: {dataframe1}")

        # Loop through all columns in the DataFrame
        for column in dataframe1.columns:
            label_encoder = LabelEncoder()
            dataframe1[column] = label_encoder.fit_transform(dataframe1[column])

    return dataframe1,
```

Below is the resultant dataset. All columns are having the encoded numerical characters.



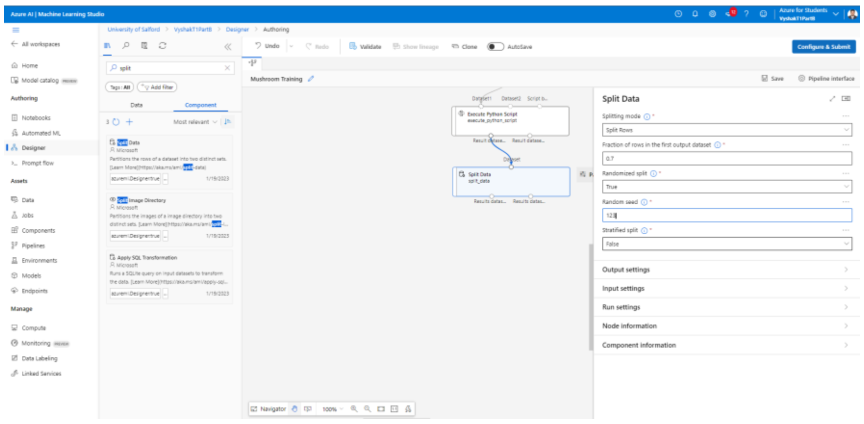
Result_Dataset

Rows 8,124 Columns 23

class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	stalk-root	stalk-surface-above-ring
1	5	2	4	1	6	1	0	1	4	0	3	2
0	5	2	9	1	0	1	0	0	4	0	2	2
0	0	2	8	1	3	1	0	0	5	0	2	2
1	5	3	8	1	6	1	0	1	5	0	3	2
0	5	2	3	0	5	1	1	0	4	1	3	2
0	5	3	9	1	0	1	0	0	5	0	2	2
0	0	2	8	1	0	1	0	0	2	0	2	2
0	0	3	8	1	3	1	0	0	5	0	2	2
1	5	3	8	1	6	1	0	1	7	0	3	2
0	0	2	9	1	0	1	0	0	2	0	2	2
0	5	3	9	1	3	1	0	0	2	0	2	2
0	5	3	9	1	0	1	0	0	5	0	2	2
0	0	2	9	1	0	1	0	0	10	0	2	2
1	5	3	8	1	6	1	0	1	4	0	3	2
0	5	0	4	0	5	1	1	0	5	1	3	2
0	4	0	3	0	5	1	0	1	4	0	3	2
0	2	0	8	0	5	1	1	0	4	1	3	2
1	5	2	4	1	6	1	0	1	5	0	3	2
1	5	3	8	1	6	1	0	1	5	0	3	2
1	5	2	4	1	6	1	0	1	4	0	3	2
0	0	2	9	1	0	1	0	0	4	0	2	2
1	5	3	4	1	6	1	0	1	5	0	3	2
0	0	3	9	1	3	1	0	0	4	0	2	2

To view, select a column in the table

2. **Training the model:** Firstly, we need to split data into train and test data. We Use the split data transformation to achieve this. 70% will be given for test data.


Split Data

Splitting mode: Split Rows

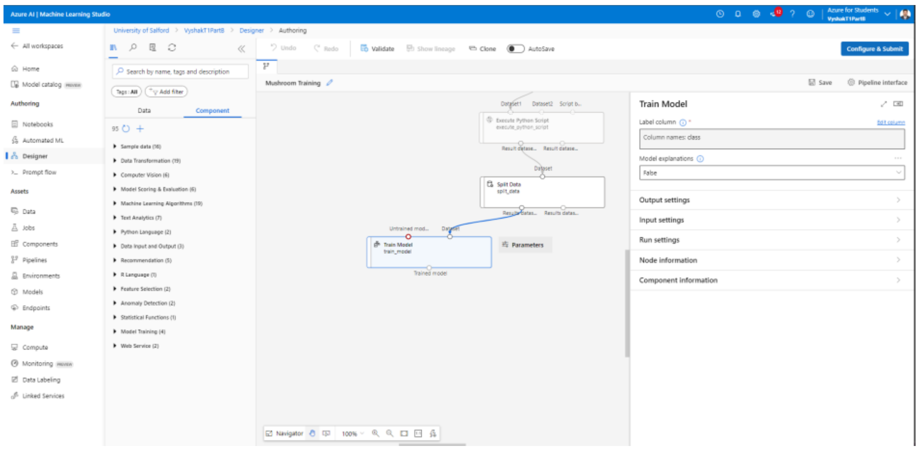
Fraction of rows in the first output dataset: 0.7

Randomized split: True

Random seed: 123

Stratified split: False

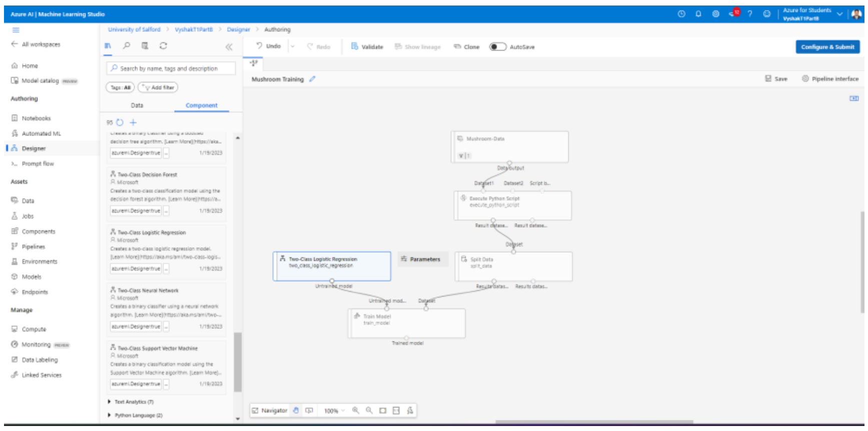
3. Next, we add the train model transformation.Connect split data resultant dataset to train model dataset input(right node). Provide the target variable in the train model.


Train Model

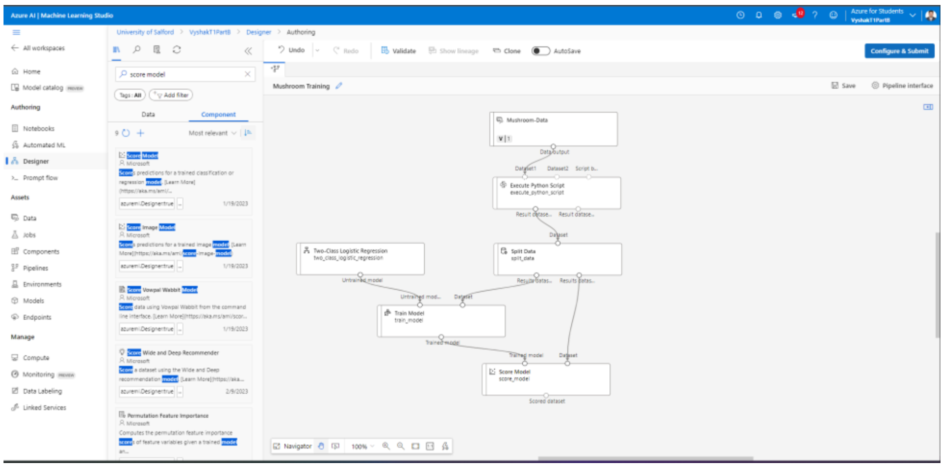
Label column: Column names: class

Model explanations: False

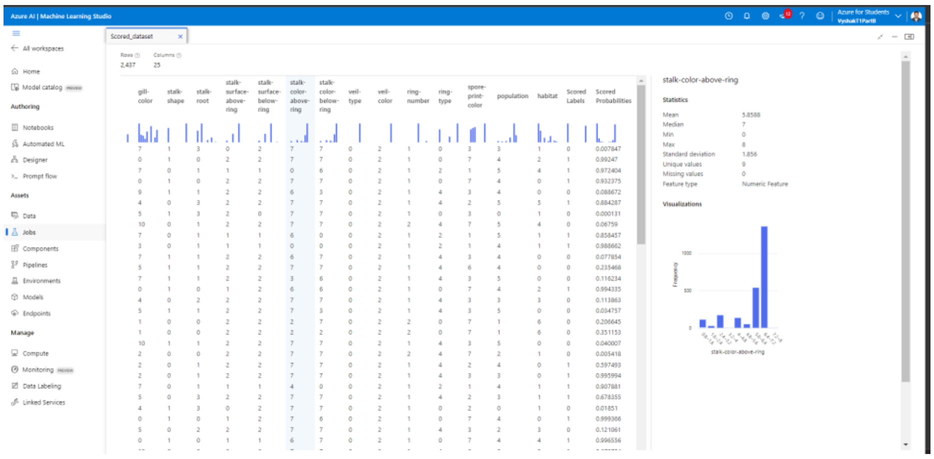
4. **<u>Two Class Logistic Regression :</u>** Add the Two Class Logistic Regression from the machine learning algorithm components. Connect the output of this to the left node of Train model.



5. **<u>Scored Model:</u>** Connect the output of train model to the trained model node input of score model and then connect the resultant dataset of split data to dataset node of scored model.



6. **<u>Run the training pipeline:</u>** Click on configure and submit. In the jobs page select the latest pipeline. Right click on **Scored model** and select Preview data to analyze the result.


Scored_dataset

Rows 2,437 Columns 25

gill-color	stalk-shape	stalk-root	stalk-surface-above-ring	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat	Scored Labels	Scored Probabilities
7	1	3	0	2	7	7	0	2	1	0	3	3	1	0	0.007847
0	1	0	2	2	7	7	0	2	1	0	7	4	2	1	0.99247
7	0	1	1	1	0	6	0	2	1	2	1	5	4	1	0.972404
0	1	0	2	2	7	7	0	2	1	0	7	4	0	1	0.932375
9	1	1	2	2	6	3	0	2	1	4	3	4	0	0	0.088672
4	0	3	2	2	7	7	0	2	1	4	2	5	5	1	0.884287
5	1	3	2	0	7	7	0	2	1	0	3	0	1	0	0.000131
10	0	1	2	2	7	7	0	2	2	4	7	5	4	0	0.06759
7	0	1	1	1	6	0	0	2	1	2	1	5	1	1	0.858457
3	0	1	1	1	0	0	0	2	1	2	1	4	1	1	0.988662
7	1	1	2	2	6	7	0	2	1	4	3	4	0	0	0.077854
5	1	1	2	2	7	7	0	2	1	4	6	4	0	0	0.235468
7	1	1	2	2	3	6	0	2	1	4	3	5	0	0	0.116234
0	1	0	1	2	6	6	0	2	1	0	7	4	2	1	0.994335
4	0	2	2	2	7	7	0	2	1	4	3	3	3	0	0.113863
5	1	1	2	2	7	3	0	2	1	4	3	5	0	0	0.034757
1	0	0	2	2	2	7	0	2	2	0	7	1	6	0	0.206645
1	0	0	2	2	2	2	0	2	2	0	7	1	6	0	0.351153
10	1	1	2	2	7	7	0	2	1	4	3	5	0	0	0.040007
2	0	0	2	2	7	7	0	2	2	4	7	2	1	0	0.005418
2	0	1	2	2	7	7	0	2	1	4	2	4	0	1	0.597493
2	0	1	2	2	7	7	0	2	1	4	3	3	0	1	0.995994
7	0	1	1	1	4	0	0	2	1	2	1	4	1	1	0.907881
5	0	3	2	2	7	7	0	2	1	4	2	3	1	1	0.678355
4	1	3	0	2	7	7	0	2	1	0	2	0	1	0	0.01851
0	1	0	1	2	7	6	0	2	1	0	7	4	0	1	0.999366
5	0	2	2	2	7	7	0	2	1	4	3	2	3	0	0.121061
0	1	0	1	1	6	7	0	2	1	0	7	4	4	1	0.996556

stalk-color-above-ring

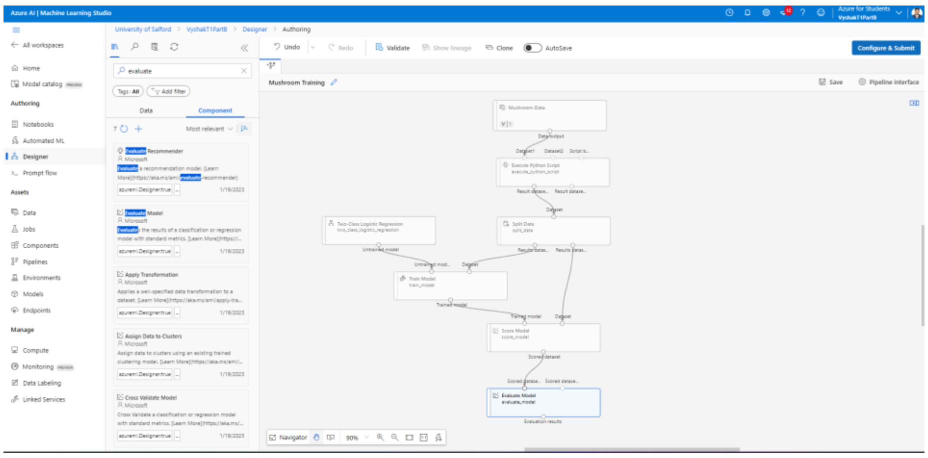
Statistics

Mean	5.8588
Median	7
Min	0
Max	8
Standard deviation	1.856
Unique values	9
Missing values	0
Feature type	Numeric Feature

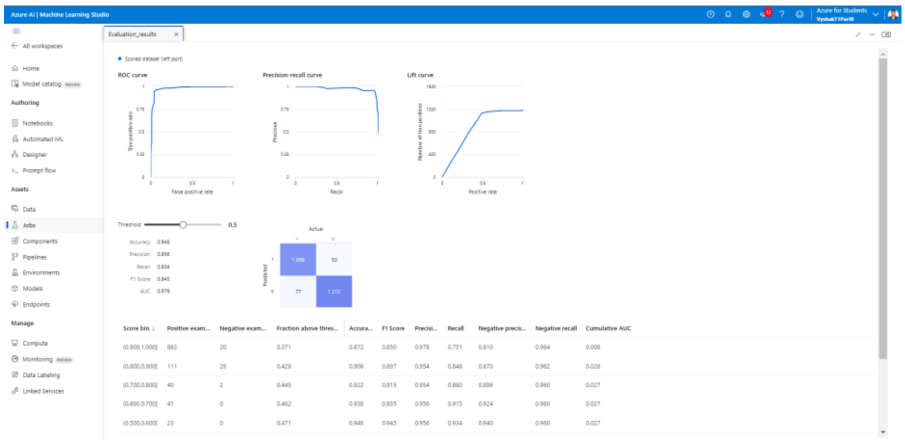
Visualizations

The result will have scored labels and scored probability which displays what is the probability of the predicted result.

7. **<u>Evaluate the model:</u>** Add Evaluate model from components to the canvas. And then click again on configure and submit to run the pipeline.



Once the run is completed go to job and right click on evaluate model to analyze the results.



Threshold 0.5

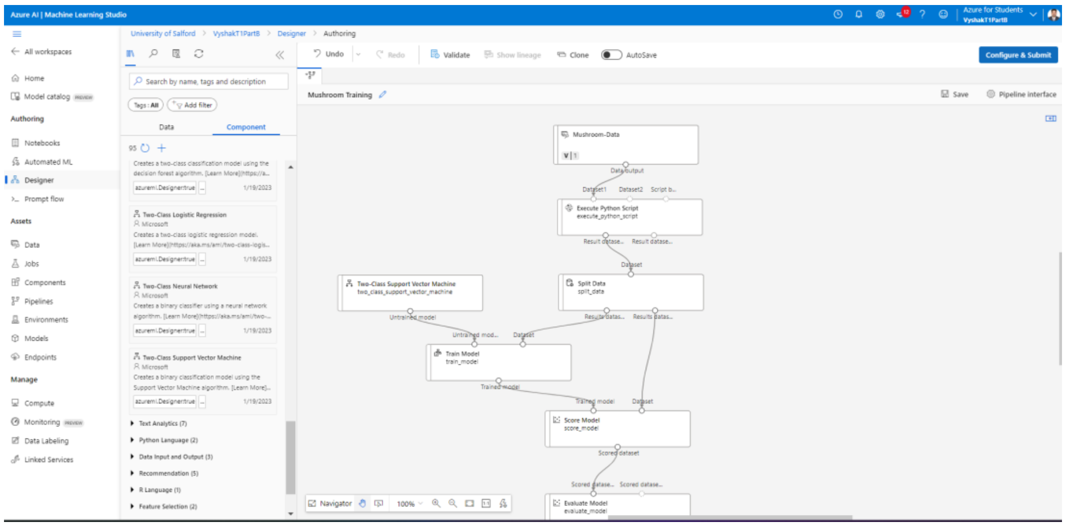
Accuracy	0.948
Precision	0.956
Recall	0.934
F1 Score	0.945
AUC	0.979

Predicted \ Actual	1	0
1	1 098	50
0	77	1 212

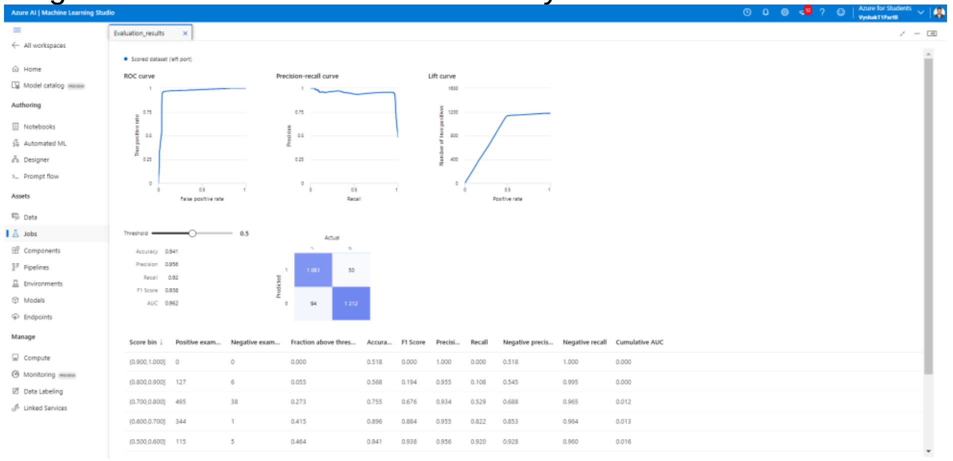
Score bin ↓	Positive exam...	Negative exam...	Fraction above thres...	Accura...	F1 Score	Precisi...	Recall	Negative precis...	Negative recall	Cumulative AUC
(0.900,1.000]	883	20	0.371	0.872	0.850	0.978	0.751	0.810	0.984	0.008
(0.800,0.900]	111	28	0.428	0.906	0.897	0.954	0.846	0.870	0.962	0.026
(0.700,0.800]	40	2	0.445	0.922	0.915	0.954	0.880	0.896	0.960	0.027
(0.600,0.700]	41	0	0.462	0.938	0.935	0.956	0.915	0.924	0.960	0.027
(0.500,0.600]	23	0	0.471	0.948	0.945	0.956	0.934	0.940	0.960	0.027

The two-model logistic regression predicts with an accuracy of 94.8%. True positive is 1098. True Negative is 1212.

Two Class Support Vector Machine:



Once added, click on configure and submit. Once the pipeline run is completed, go to the jobs page and right click on Evaluate model to analyze the result.



Threshold 0.5

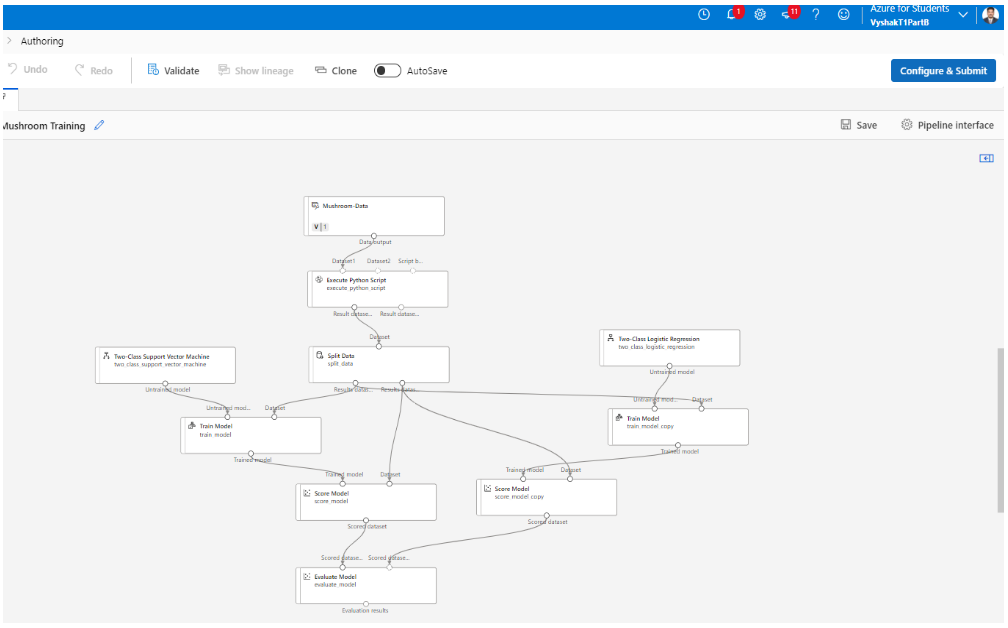
Accuracy	0.941
Precision	0.956
Recall	0.92
F1 Score	0.938
AUC	0.962

Predicted \ Actual	1	0
1	1 081	50
0	94	1 212

Score bin ↓	Positive exam...	Negative exam...	Fraction above thres...	Accura...	F1 Score	Precisi...	Recall	Negative precis...	Negative recall	Cumulative AUC
(0.900,1.000]	0	0	0.000	0.518	0.000	1.000	0.000	0.518	1.000	0.000
(0.800,0.900]	127	6	0.055	0.568	0.194	0.955	0.108	0.545	0.995	0.000
(0.700,0.800]	495	38	0.273	0.755	0.676	0.934	0.529	0.688	0.965	0.012
(0.600,0.700]	344	1	0.415	0.896	0.884	0.955	0.822	0.853	0.964	0.013
(0.500,0.600]	115	5	0.464	0.941	0.938	0.956	0.920	0.928	0.960	0.016

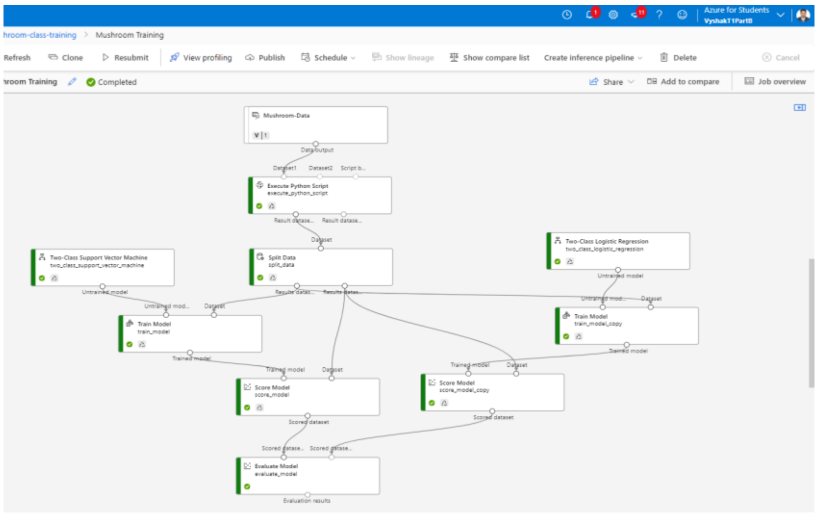
Model predicted at an accuracy of 94.1%. True positives are 1081. True Negatives are 1212.

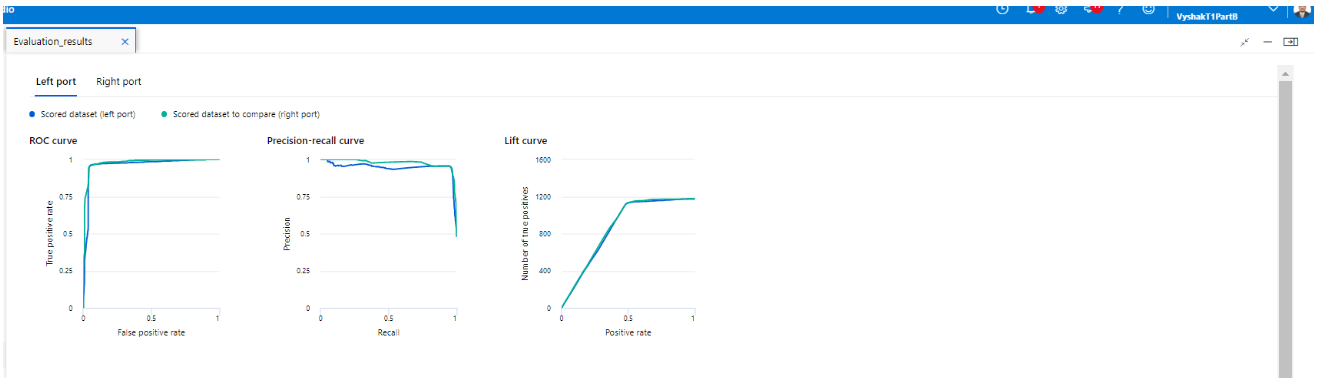
<u>Training Model together</u>: Below figure shows running a pipeline with two models parallelly. We created two train models and score models. Then the machine learning algorithms were connected as explained previously. Then we ran the pipeline.



Once the pipeline run is completed, we will go to the jobs page to evaluate the model.

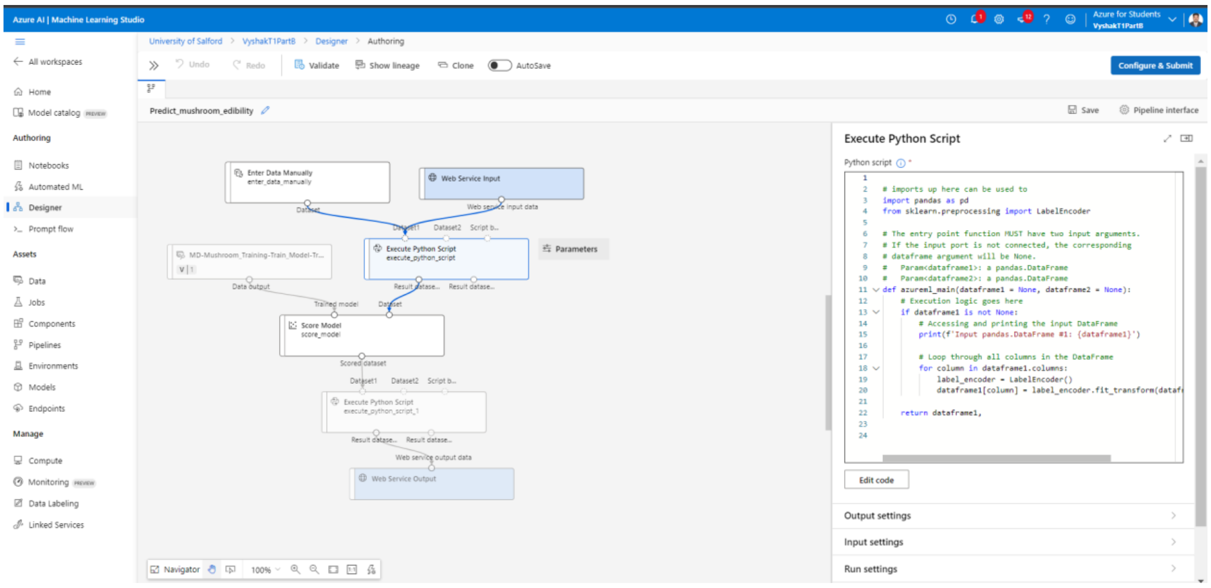
The evaluate model will now have two separate tabs with results of both machine learning models.





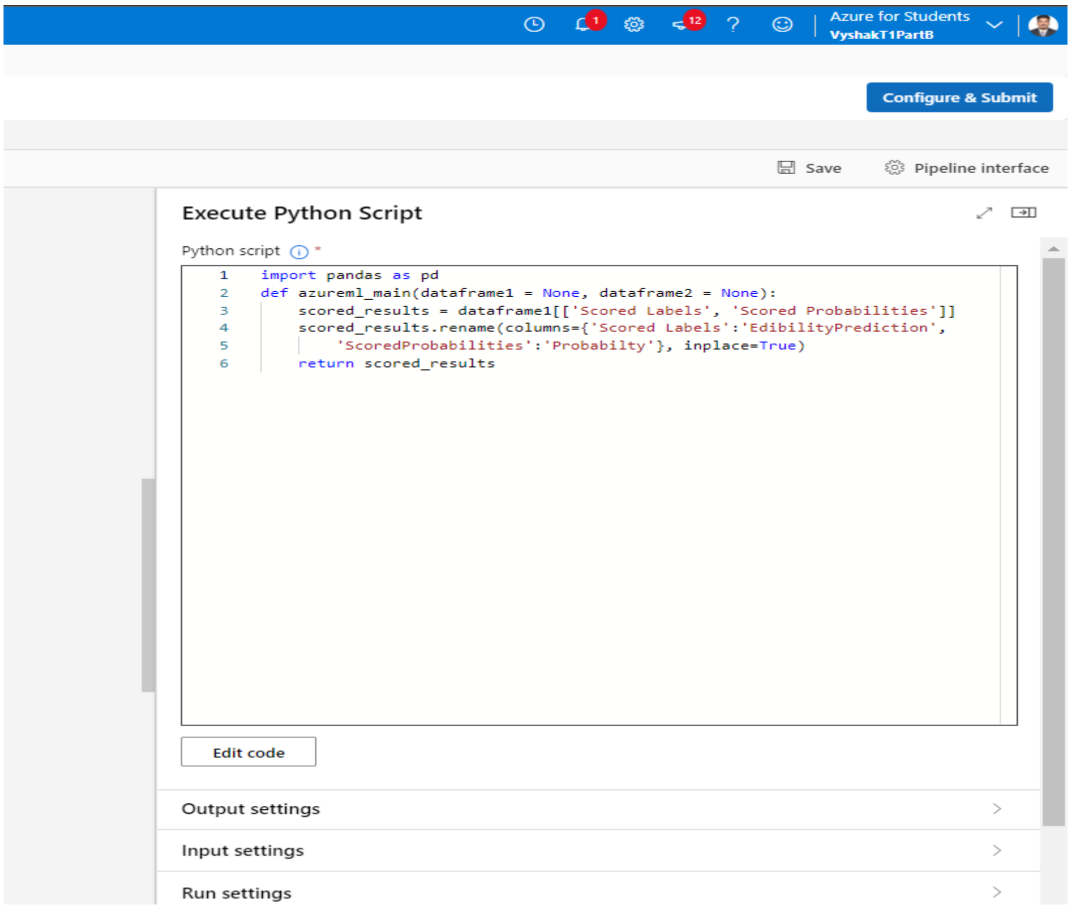
Inference Pipeline:

On jobs page, click on the mushroom training pipeline and create an inference pipeline. The canvas will be populated with few items already in it.



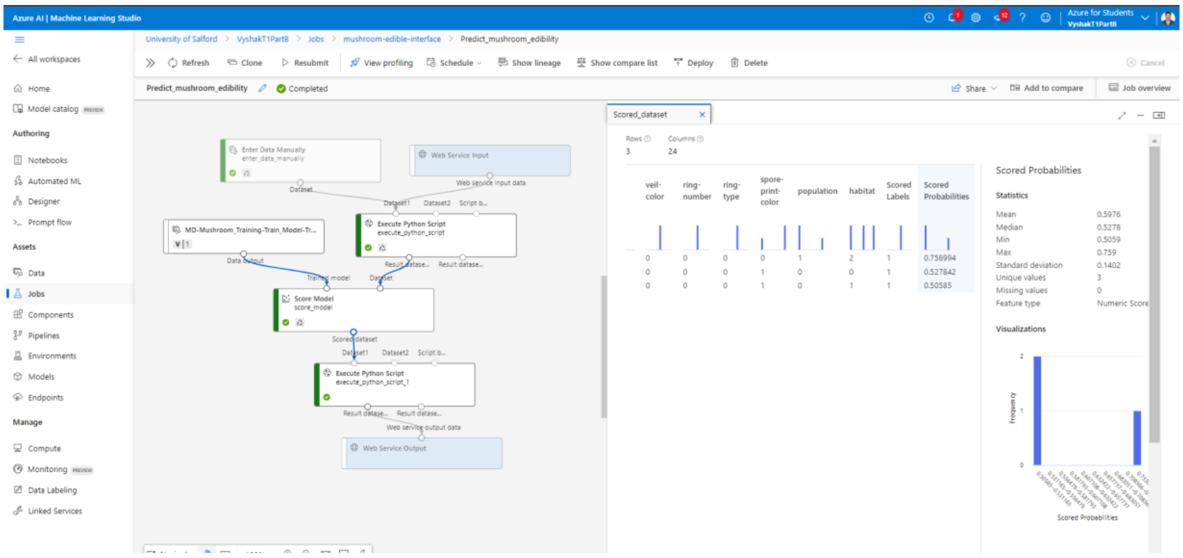
```
# imports up here can be used to
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# The entry point function MUST have two input arguments.
# If the input port is not connected, the corresponding
# dataframe argument will be None.
#   Param<dataframe1>: a pandas.DataFrame
#   Param<dataframe2>: a pandas.DataFrame
def azureml_main(dataframe1 = None, dataframe2 = None):
    # Execution logic goes here
    if dataframe1 is not None:
        # Accessing and printing the input DataFrame
        print(f'Input pandas.DataFrame #1: {dataframe1}')

        # Loop through all columns in the DataFrame
        for column in dataframe1.columns:
            label_encoder = LabelEncoder()
            dataframe1[column] = label_encoder.fit_transform(dataf

    return dataframe1,
```

The first execute python script label will have label encoder script in it. In the second Execute python script, we will have to edit the following:



```
import pandas as pd
def azureml_main(dataframe1 = None, dataframe2 = None):
    scored_results = dataframe1[['Scored Labels', 'Scored Probabilities']]
    scored_results.rename(columns={'Scored Labels':'EdibilityPrediction',
        'ScoredProbabilities':'Probabilty'}, inplace=True)
    return scored_results
```

Run the inference pipeline and once it is completed. Analyze the scored labels and scored probabilities.



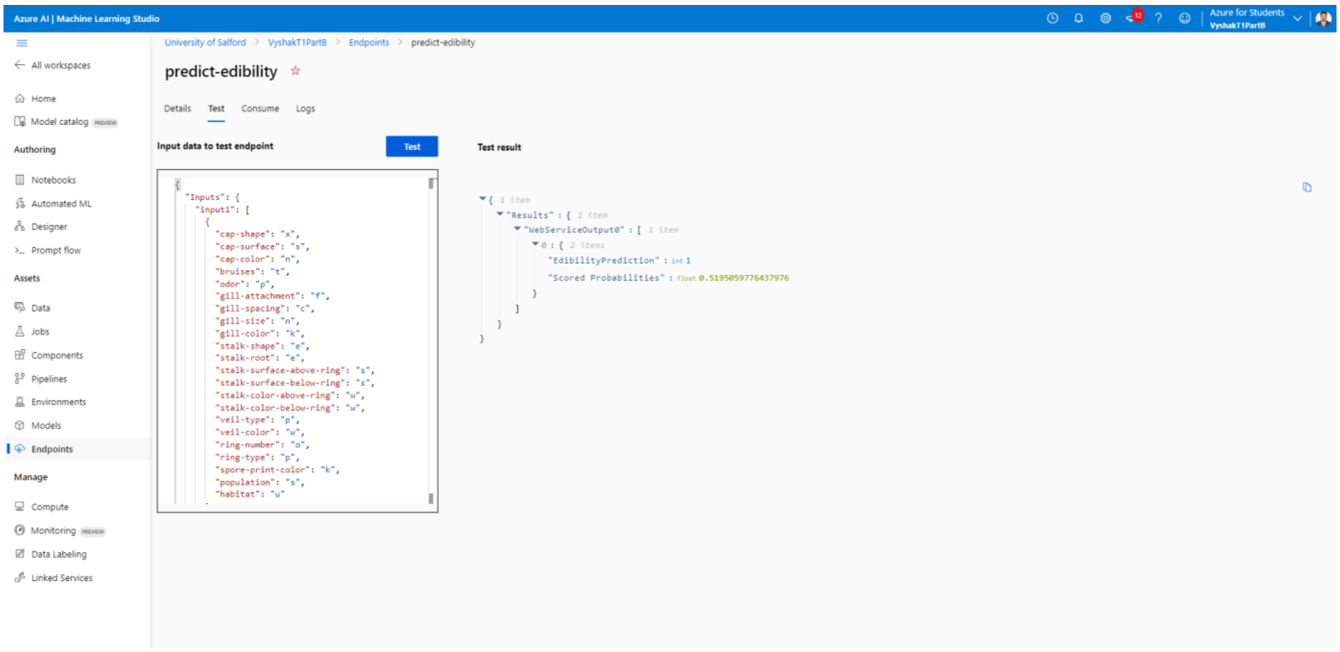
Rows 3 Columns 24

veil-color	ring-number	ring-type	spore-print-color	population	habitat	Scored Labels	Scored Probabilities
0	0	0	0	1	2	1	0.758994
0	0	0	1	0	0	1	0.527842
0	0	0	1	0	1	1	0.50585

Statistics

Mean	0.5976
Median	0.5278
Min	0.5059
Max	0.759
Standard deviation	0.1402
Unique values	3
Missing values	0
Feature type	Numeric Score

Next we will deploy our inference pipeline to test. Once the deployment is done, a test data is fed in json format. The output will be a prediction and the scored probability.



Input data to test endpoint

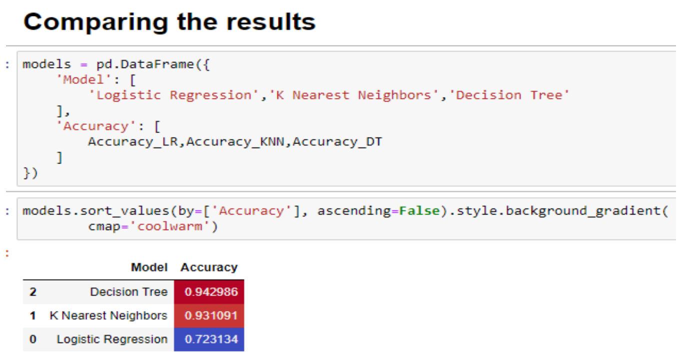
```
{
  "Inputs": {
    "input1": [
      {
        "cap-shape": "x",
        "cap-surface": "s",
        "cap-color": "n",
        "bruises": "t",
        "odor": "p",
        "gill-attachment": "f",
        "gill-spacing": "c",
        "gill-size": "n",
        "gill-color": "k",
        "stalk-shape": "e",
        "stalk-root": "e",
        "stalk-surface-above-ring": "s",
        "stalk-surface-below-ring": "s",
        "stalk-color-above-ring": "w",
        "stalk-color-below-ring": "w",
        "veil-type": "p",
        "veil-color": "w",
        "ring-number": "o",
        "ring-type": "p",
        "spore-print-color": "k",
        "population": "s",
        "habitat": "u"
```

Test result

```
{ 1 item
  "Results" : { 1 item
    "WebServiceOutput0" : [ 1 item
      0 : { 2 items
        "EdibilityPrediction" : int 1
        "Scored Probabilities" : float 0.5195059776437976
      }
    ]
  }
}
```

8. Evaluation of Results:

Python: From the below figure, we can confirm that the best performing model is Decision Tree Classifier which predicted with an accuracy of 94%.



Comparing the results

```
models = pd.DataFrame({
    'Model': [
        'Logistic Regression','K Nearest Neighbors','Decision Tree'
    ],
    'Accuracy': [
        Accuracy_LR,Accuracy_KNN,Accuracy_DT
    ]
})
```

```
models.sort_values(by=['Accuracy'], ascending=False).style.background_gradient(
        cmap='coolwarm')
```

	Model	Accuracy
2	Decision Tree	0.942986
1	K Nearest Neighbors	0.931091
0	Logistic Regression	0.723134

<u>Azure:</u> Two model logistic regression gave good accuracy with 94.8%. the precision and recall metrics—95.6% and 93.4%, respectively—are incredibly high. This indicates that 95.6% of the time the model properly identifies edible mushrooms and 93.4% of the time it correctly identifies poisonous mushrooms.

In Conclusion, both the Python Decision Tree model and the Microsoft Azure logistic regression model showed remarkable accuracy when assessing the classification performance for the mushroom dataset. The Decision Tree demonstrated its strong capacity to distinguish between edible and deadly mushrooms with an accuracy of 94%, along with good precision and recall scores. In contrast, the logistic regression model on Azure performed better, demonstrating its effectiveness in appropriately classifying edible and hazardous mushrooms with an accuracy of 94.8%. With precision and recall metrics of 95.6% and 93.4%, respectively, these numbers demonstrate the model's remarkable categorization accuracy. These findings support the effectiveness of both models in handling the task of predicting the edibility of mushrooms, offering trustworthy and precise insights into their classification.

Part Two: Clustering

1. Introduction

Imagine you have a vast library of user reviews spanning a wide spectrum of establishments, from cozy cafes and bustling marketplaces to serene temples and vibrant art galleries. This is essentially the Google review dataset, a treasure trove of insights into people's preferences and experiences across various venues.

The Google review dataset is a collection of user reviews on a variety of establishments, including churches and resorts, as well as restaurants and beauty spas. Each record in the dataset is distinguished by a unique user ID and average ratings across all sorts of venues, revealing vital information about user preferences and opinions. The dataset provides a rich supply of data for clustering analysis, which aims to group comparable individuals based on their average ratings across many attributes.

Each entry in this dataset is like a unique voice, express ing their opinions and ratings for different types of establishments. Just as a book reviewer might have distinct tastes in genres, a user in this dataset may prefer lively cafes and adventurous restaurants while another might gravitate towards serene temples and intimate art galleries.

This information acts as a social compass, directing us to people who share our tastes and preferences, allowing us to discover new experiences and connections throughout the vast geography of human preferences.

2. Research Objective

“Using clustering methods, investigate patterns in user preferences in the Google review dataset."

The major goal of this clustering study is to find hidden patterns and structures in the Google review dataset. We intend to group users with similar reviewing patterns and preferences using two clustering techniques, potentially revealing distinct subsets of the user community. Finally, we need to decide on which clustering technique can be utilized for future applications.

3. Overview of dataset

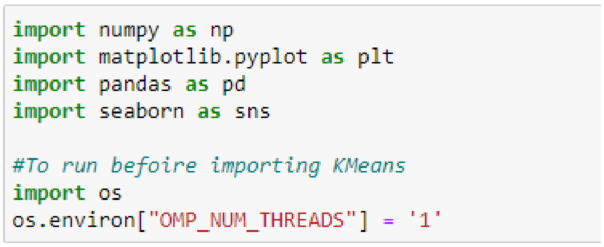
The source of this dataset is https://archive.ics.uci.edu/dataset/485/tarvel+review+ratings User ratings from Google reviews are used to create this data set. Attraction reviews from 24 categories from around Europe are evaluated. Google user ratings vary from 1 to 5, with the average user rating derived for each category. The dataset have 5456 instances and 26 features. Below table describes the independent attributes.

Attributes	Description
User	Unique user id
Category 2	Average ratings on churches
Category 3	Average ratings on beaches
Category 4	bruises=t,no=f
Category 5	Average ratings on parks
Category 6	Average ratings on theatres
Category 7	Average ratings on museums
Category 8	Average ratings on malls
Category 9	Average ratings on zoo
Category 10	Average ratings on restaurants
Category 11	Average ratings on pubs/bars
Category 12	Average ratings on local services
Category 13	Average ratings on burger/pizza shops
Category 14	Average ratings on hotels/other lodgings
Category 15	Average ratings on juice bars
Category 16	Average ratings on art galleries
Category 17	Average ratings on dance clubs
Category 18	Average ratings on swimming pools
Category 19	Average ratings on gyms
Category 20	Average ratings on bakeries
Category 21	Average ratings on beauty & spas
Category 22	Average ratings on cafes
Category 23	Average ratings on view points
Category 24	Average ratings on monuments

Unamed: 25	Average ratings on gardens

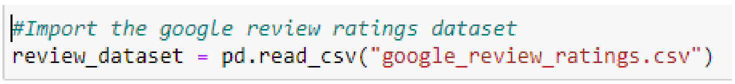
4. Exploratory Data Analysis

Importing necessary libraries: Below figure shows the necessary python libraries that we require to perform our clustering task. We will make use some of the popular python libraries like pandas, numpy, matplotlib and seaborn. Apart from these we will also require some libraries for our clustering which we will see in detail in the later sections. Here we set OMP_NUM_THREADS to 1 to limit the parallelism used by the KMeans algorithm.



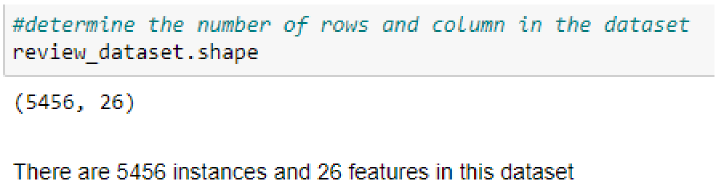
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

#To run befoire importing KMeans
import os
os.environ["OMP_NUM_THREADS"] = '1'
```

Loading the dataset: using pandas' read_csv function, we load the dataset of csv format for analysis.



```
#Import the google review ratings dataset
review_dataset = pd.read_csv("google_review_ratings.csv")
```

Dimensions of dataset: shape is used to check the dimensions of dataset. This returns a tuple with the number of rows and columns that the dataset has. Our current dataset has 5456 rows and 26 columns.

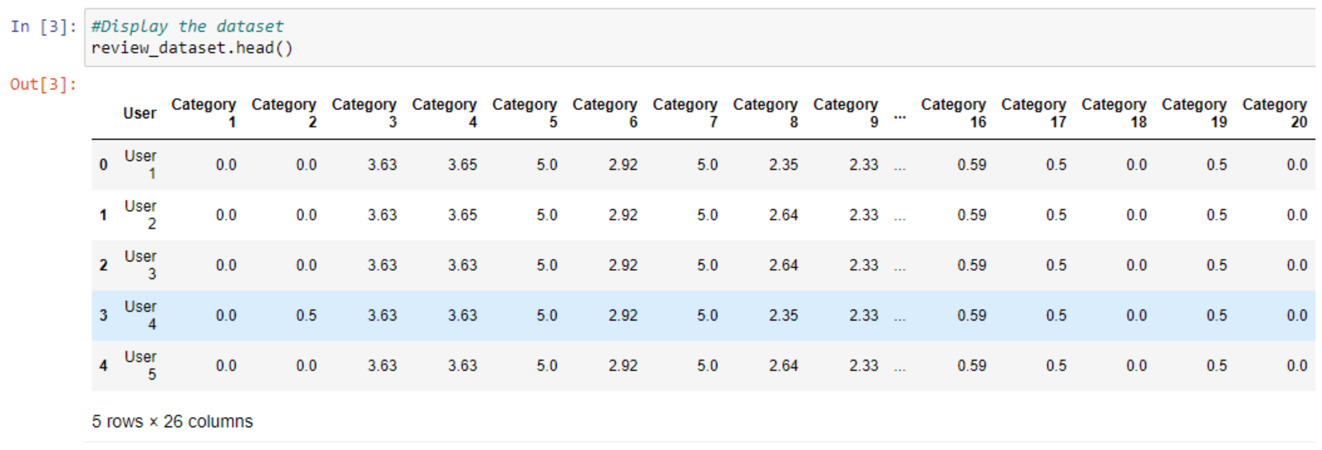


```
#determine the number of rows and column in the dataset
review_dataset.shape
```

```
(5456, 26)
```

There are 5456 instances and 26 features in this dataset

Displaying the first few rows of dataset: This is done by using the head() function which print the first 5 rows in the dataset.



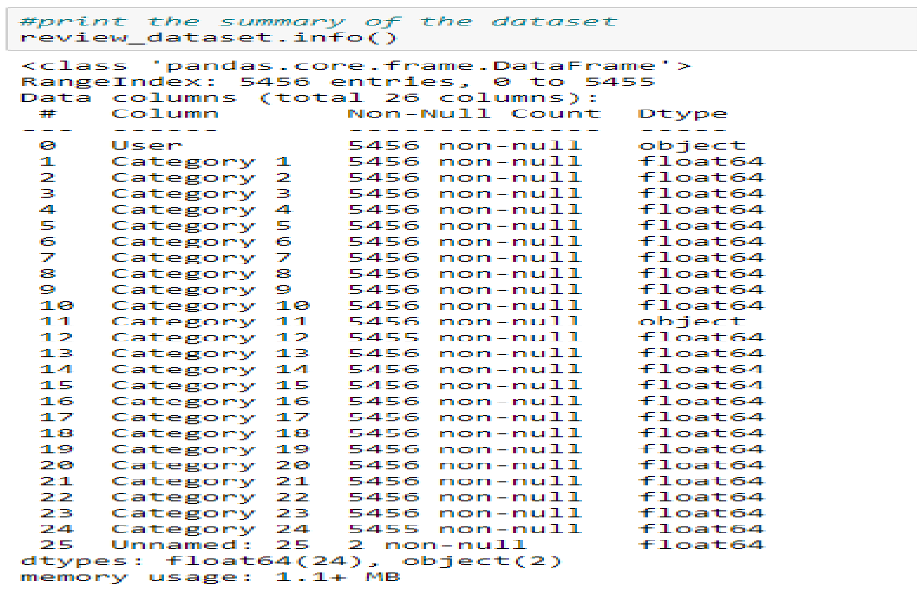
```
In [3]: #Display the dataset
        review_dataset.head()
Out[3]:
```

	User	Category 1	Category 2	Category 3	Category 4	Category 5	Category 6	Category 7	Category 8	Category 9	...	Category 16	Category 17	Category 18	Category 19	Category 20
0	User 1	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.35	2.33	...	0.59	0.5	0.0	0.5	0.0
1	User 2	0.0	0.0	3.63	3.65	5.0	2.92	5.0	2.64	2.33	...	0.59	0.5	0.0	0.5	0.0
2	User 3	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	2.33	...	0.59	0.5	0.0	0.5	0.0
3	User 4	0.0	0.5	3.63	3.63	5.0	2.92	5.0	2.35	2.33	...	0.59	0.5	0.0	0.5	0.0
4	User 5	0.0	0.0	3.63	3.63	5.0	2.92	5.0	2.64	2.33	...	0.59	0.5	0.0	0.5	0.0

5 rows × 26 columns

<u>Summary of the dataframe:</u> The info() method is used to print the concise summary of the dataframe. From the below figure we can interpret the following

- All the features in the dataset consists of categorical values except category 11 which we will convert to numeric in next step.



```
#print the summary of the dataset
review_dataset.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5456 entries, 0 to 5455
Data columns (total 26 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   User         5456 non-null   object
 1   Category 1   5456 non-null   float64
 2   Category 2   5456 non-null   float64
 3   Category 3   5456 non-null   float64
 4   Category 4   5456 non-null   float64
 5   Category 5   5456 non-null   float64
 6   Category 6   5456 non-null   float64
 7   Category 7   5456 non-null   float64
 8   Category 8   5456 non-null   float64
 9   Category 9   5456 non-null   float64
 10  Category 10  5456 non-null   float64
 11  Category 11  5456 non-null   object
 12  Category 12  5455 non-null   float64
 13  Category 13  5456 non-null   float64
 14  Category 14  5456 non-null   float64
 15  Category 15  5456 non-null   float64
 16  Category 16  5456 non-null   float64
 17  Category 17  5456 non-null   float64
 18  Category 18  5456 non-null   float64
 19  Category 19  5456 non-null   float64
 20  Category 20  5456 non-null   float64
 21  Category 21  5456 non-null   float64
 22  Category 22  5456 non-null   float64
 23  Category 23  5456 non-null   float64
 24  Category 24  5455 non-null   float64
 25  Unnamed: 25  2 non-null      float64
dtypes: float64(24), object(2)
memory usage: 1.1+ MB
```

Figure 1

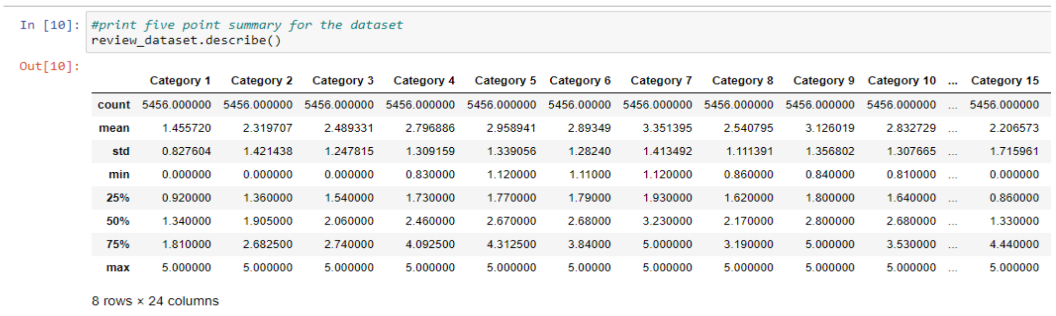
<u>Five Point Summary:</u> Figure below gives the five-point summary of the dataset. Below are some of the observations from that

- The mean ratings across all categories generally range from approximately 1.45 to 3.35. The highest average rating is observed in Category 7, while the lowest mean rating is in Category 18.

- Category22 has the highest SD hence, it has more dispersed ratings. Whereas category1 has the lowest meaning the ratings are consistent.

- A higher median often indicates that a considerable portion of users rated that category significantly higher, indicating a more positive overall mood or experience. A lower median indicates that a significant number of people rated it lower, implying a less pleasant emotion or experience on average.

- Looking at the five point summary, we can assume that there wont be significant outliers.

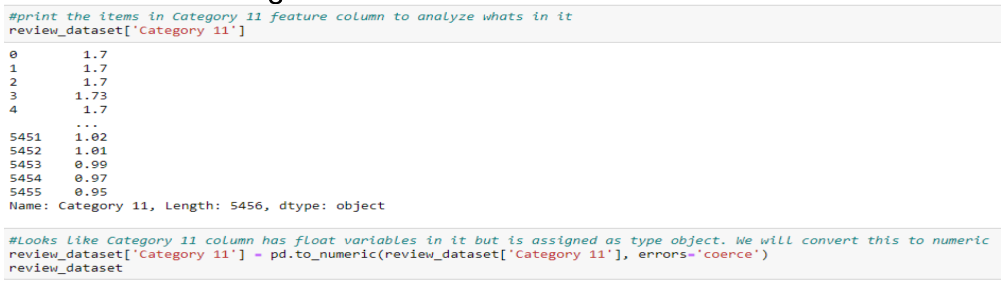


```
In [10]: #print five point summary for the dataset
         review_dataset.describe()
Out[10]:
```

	Category 1	Category 2	Category 3	Category 4	Category 5	Category 6	Category 7	Category 8	Category 9	Category 10	...	Category 15
count	5456.000000	5456.000000	5456.000000	5456.000000	5456.000000	5456.00000	5456.000000	5456.000000	5456.000000	5456.000000	...	5456.000000
mean	1.455720	2.319707	2.489331	2.796886	2.958941	2.89349	3.351395	2.540795	3.126019	2.832729	...	2.206573
std	0.827604	1.421438	1.247815	1.309159	1.339056	1.28240	1.413492	1.111391	1.356802	1.307665	...	1.715961
min	0.000000	0.000000	0.000000	0.830000	1.120000	1.11000	1.120000	0.860000	0.840000	0.810000	...	0.000000
25%	0.920000	1.360000	1.540000	1.730000	1.770000	1.79000	1.930000	1.620000	1.800000	1.640000	...	0.860000
50%	1.340000	1.905000	2.060000	2.460000	2.670000	2.68000	3.230000	2.170000	2.800000	2.680000	...	1.330000
75%	1.810000	2.682500	2.740000	4.092500	4.312500	3.84000	5.000000	3.190000	5.000000	3.530000	...	4.440000
max	5.000000	5.000000	5.000000	5.000000	5.000000	5.00000	5.000000	5.000000	5.000000	5.000000	...	5.000000

8 rows × 24 columns

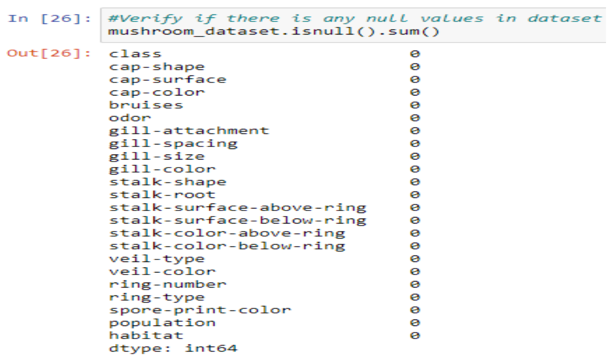
<u>Changing datatype of feature column:</u> One of the columns is a categorical column but has elements of numerical datatype. Hence, we will change the datatype from object to numerical. This is shown in Figure below



```
#print the items in Category 11 feature column to analyze whats in it
review_dataset['Category 11']

0        1.7
1        1.7
2        1.7
3       1.73
4        1.7
        ...
5451    1.02
5452    1.01
5453    0.99
5454    0.97
5455    0.95
Name: Category 11, Length: 5456, dtype: object
```

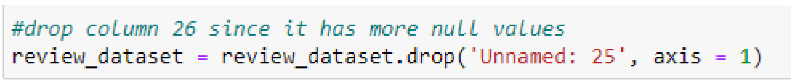
```
#Looks like Category 11 column has float variables in it but is assigned as type object. We will convert this to numeric
review_dataset['Category 11'] = pd.to_numeric(review_dataset['Category 11'], errors='coerce')
review_dataset
```

<u>Check for missing values:</u> isnull() is used to check if there is any null values in each of our column. Sum() counts the number of occurrences of null value in each of these column. From the below figure column 26 have plenty of null values and removing this wont affect much of our analysis. Hence we will remove that in the next step.



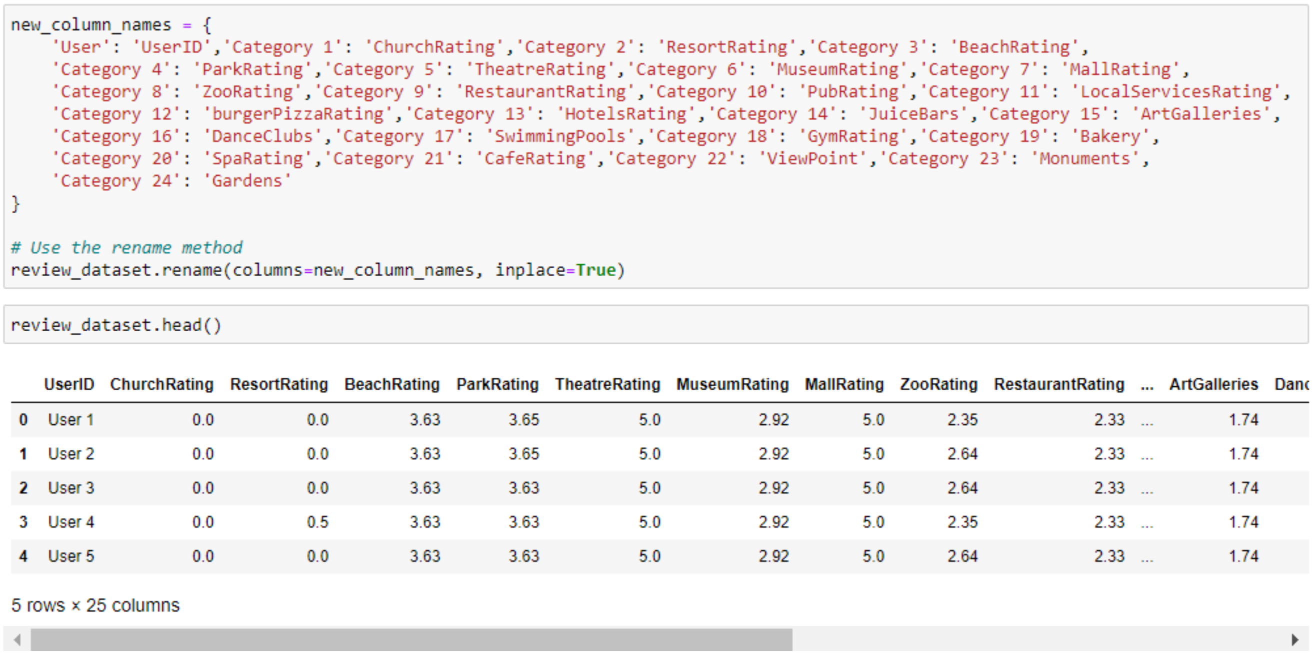
```
In [26]: #Verify if there is any null values in dataset
         mushroom_dataset.isnull().sum()
Out[26]: class                       0
         cap-shape                   0
         cap-surface                 0
         cap-color                   0
         bruises                     0
         odor                        0
         gill-attachment             0
         gill-spacing                0
         gill-size                   0
         gill-color                  0
         stalk-shape                 0
         stalk-root                  0
         stalk-surface-above-ring    0
         stalk-surface-below-ring    0
         stalk-color-above-ring      0
         stalk-color-below-ring      0
         veil-type                   0
         veil-color                  0
         ring-number                 0
         ring-type                   0
         spore-print-color           0
         population                  0
         habitat                     0
         dtype: int64
```

<u>Removing unwanted columns:</u> Column 26 has plenty null values. As discussed above we will remove this column using the drop function.



```
#drop column 26 since it has more null values
review_dataset = review_dataset.drop('Unnamed: 25', axis = 1)
```

Renaming the existing columns: For better readability we are renaming the existing columns as shown in the figure below



```
new_column_names = {
    'User': 'UserID','Category 1': 'ChurchRating','Category 2': 'ResortRating','Category 3': 'BeachRating',
    'Category 4': 'ParkRating','Category 5': 'TheatreRating','Category 6': 'MuseumRating','Category 7': 'MallRating',
    'Category 8': 'ZooRating','Category 9': 'RestaurantRating','Category 10': 'PubRating','Category 11': 'LocalServicesRating',
    'Category 12': 'burgerPizzaRating','Category 13': 'HotelsRating','Category 14': 'JuiceBars','Category 15': 'ArtGalleries',
    'Category 16': 'DanceClubs','Category 17': 'SwimmingPools','Category 18': 'GymRating','Category 19': 'Bakery',
    'Category 20': 'SpaRating','Category 21': 'CafeRating','Category 22': 'ViewPoint','Category 23': 'Monuments',
    'Category 24': 'Gardens'
}

# Use the rename method
review_dataset.rename(columns=new_column_names, inplace=True)
```

```
review_dataset.head()
```

```
   UserID  ChurchRating  ResortRating  BeachRating  ParkRating  TheatreRating  MuseumRating  MallRating  ZooRating  RestaurantRating  ...  ArtGalleries  Danc
0  User 1           0.0           0.0         3.63        3.65            5.0          2.92         5.0       2.35              2.33  ...          1.74
1  User 2           0.0           0.0         3.63        3.65            5.0          2.92         5.0       2.64              2.33  ...          1.74
2  User 3           0.0           0.0         3.63        3.63            5.0          2.92         5.0       2.64              2.33  ...          1.74
3  User 4           0.0           0.5         3.63        3.63            5.0          2.92         5.0       2.35              2.33  ...          1.74
4  User 5           0.0           0.0         3.63        3.63            5.0          2.92         5.0       2.64              2.33  ...          1.74

5 rows × 25 columns
```

5. Clustering

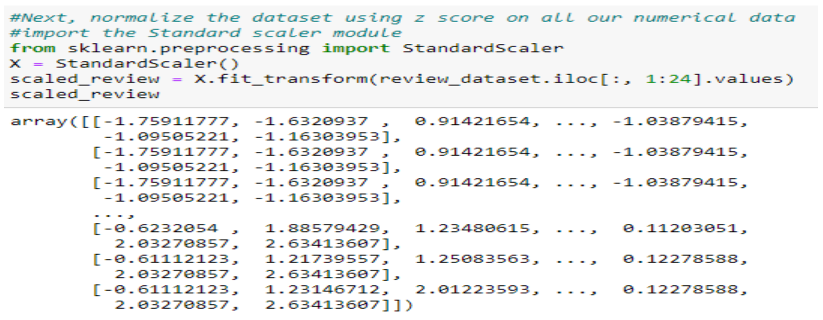
K-Mean Clustering technique

Normalization:

Normalizing the data before to applying K-Means clustering is a best practice for ensuring fair and effective clustering results, which provide a more accurate picture of the underlying patterns in the data. Even though there is no much scale differences, it is always a best practice to do normalization before applying KMeans for the following reasons:

- K-Means is sensitive to feature scale. Normalizing the data guarantees that each feature is assigned equal weight in the clustering process.
- Normalizing the data can improve the K-Means algorithm's convergence speed and efficiency.

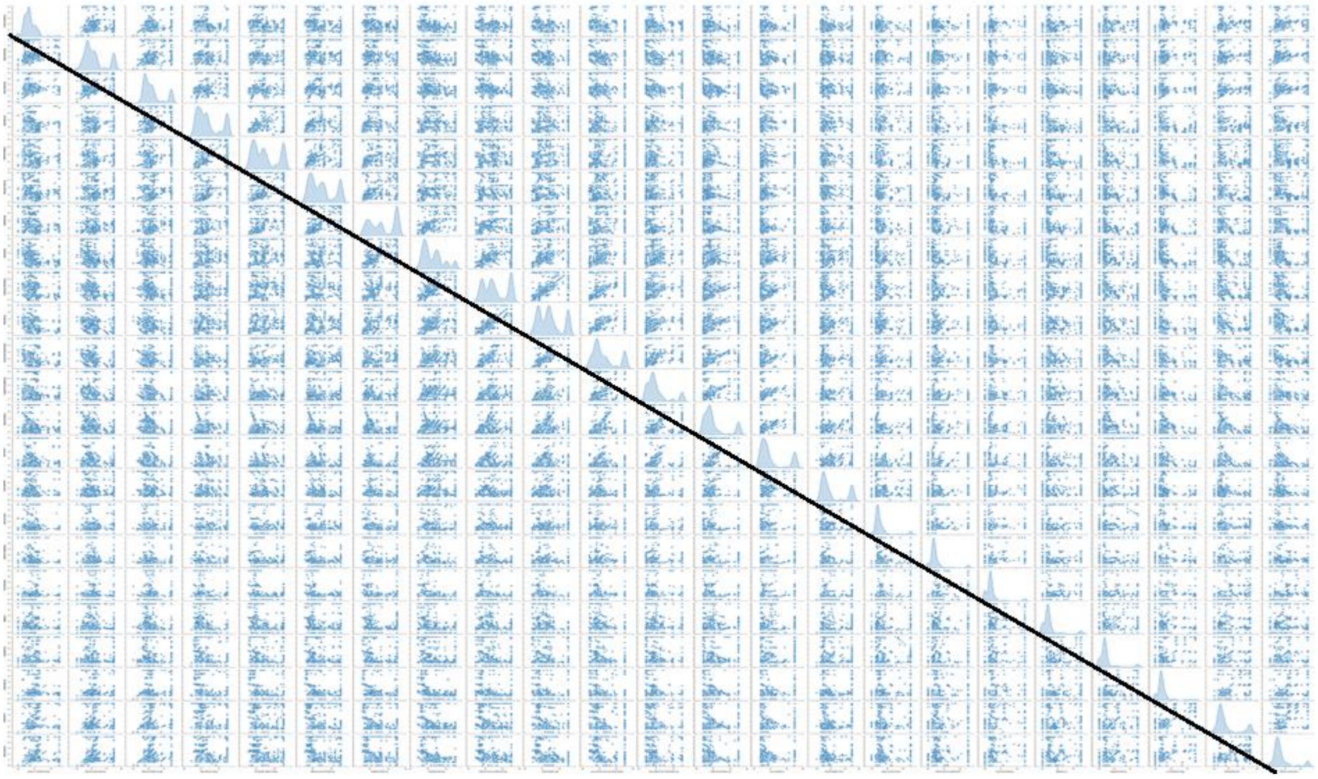
Below figure shows the normalizing of numerical columns using z score normalization which is also known as standardization. We use the **standardscaler** class from scikit-learn to achieve that. In fit_transform(), the mean and standard deviation of each feature are calculated, and the data is then standardized by subtracting the mean and dividing by the standard deviation. Resultant features will have a mean of 0 and SD of 1.



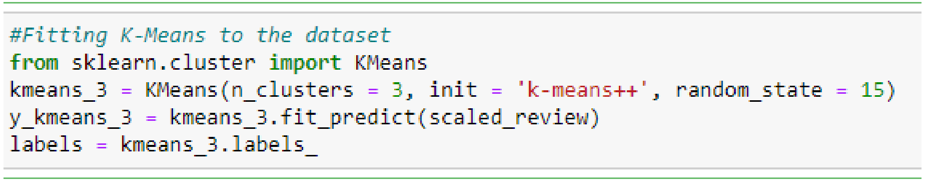
```
#Next, normalize the dataset using z score on all our numerical data
#import the Standard scaler module
from sklearn.preprocessing import StandardScaler
X = StandardScaler()
scaled_review = X.fit_transform(review_dataset.iloc[:, 1:24].values)
scaled_review
```

```
array([[-1.75911777, -1.6320937 ,  0.91421654, ..., -1.03879415,
        -1.09505221, -1.16303953],
       [-1.75911777, -1.6320937 ,  0.91421654, ..., -1.03879415,
        -1.09505221, -1.16303953],
       [-1.75911777, -1.6320937 ,  0.91421654, ..., -1.03879415,
        -1.09505221, -1.16303953],
       ...,
       [-0.6232054 ,  1.88579429,  1.23480615, ...,  0.11203051,
         2.03270857,  2.63413607],
       [-0.61112123,  1.21739557,  1.25083563, ...,  0.12278588,
         2.03270857,  2.63413607],
       [-0.61112123,  1.23146712,  2.01223593, ...,  0.12278588,
         2.03270857,  2.63413607]])
```

Scatter Plot: To get an initial guess on the number of clusters we will plot a scatter plot that shows the pairwise relationship between the variables. The KDE plot along diagonal will provide insights into the distribution which will further give us an initial guess on the number of clusters by analyzing their peaks. From figure below we can interpret most of the variables have a **peak** of 3. So, we will first run our K Means with **3 clusters**.

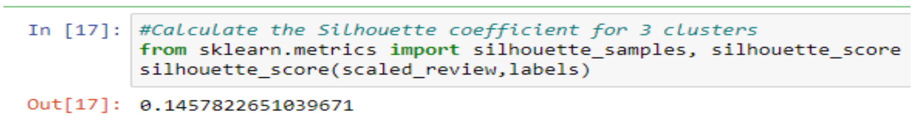


Fitting K-Means to dataset: The below figure shows the code to fit the K-Means clustering algorithm to the scaled dataset. For this we create an instance of KMeans class and pass the number of clusters as 3. K-means++ is a smart initialization method for the centroids. The fit_predict() is used to fit the KMeans model to standardized data and result is stored in y_kmeans_3. Cluster label for each data point is assigned to labels variable.



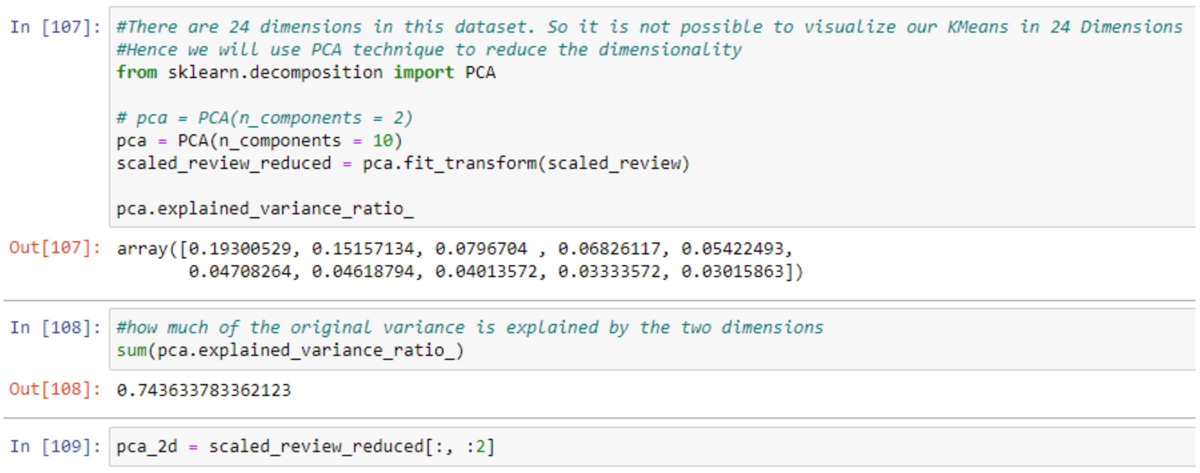
```
#Fitting K-Means to the dataset
from sklearn.cluster import KMeans
kmeans_3 = KMeans(n_clusters = 3, init = 'k-means++', random_state = 15)
y_kmeans_3 = kmeans_3.fit_predict(scaled_review)
labels = kmeans_3.labels_
```

<u>Silhouette Analysis:</u> The ideal number of clusters can be determined by silhouette analysis. The ideal number of clusters is sometimes thought to be the one that optimizes the average silhouette coefficient.Greater isolation and cohesiveness between clusters are indicated by higher silhouette coefficients.Below code shows the Silhoette Coefficient for 3 clusters. **The coefficient score is 0.145**. We will do further analysis with more clusters and determine the best by analysing the silhouette score received.



```
In [17]:  #Calculate the Silhouette coefficient for 3 clusters
          from sklearn.metrics import silhouette_samples, silhouette_score
          silhouette_score(scaled_review,labels)

Out[17]:  0.1457822651039671
```

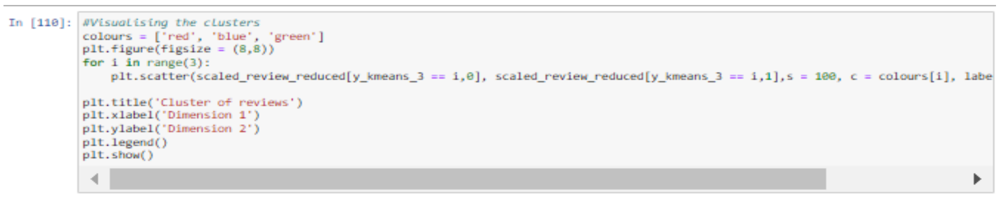
<u>Principal Component Analysis:</u> Since our dataset has 24 components we cannot visualize our clusters in 24 Dimensions, hence we will use PCA technique to reduce the dimensionality. In PCA we assign a value for number of components inorder to retain more variance in our visualization which is set to 10. Then for the purpose of visualizing in 2D space we use a subset of these components. **In the below code we could get a cumulative variance of 74.3%. The interesting fact was with 2 components I could only get 34.4% variance.**



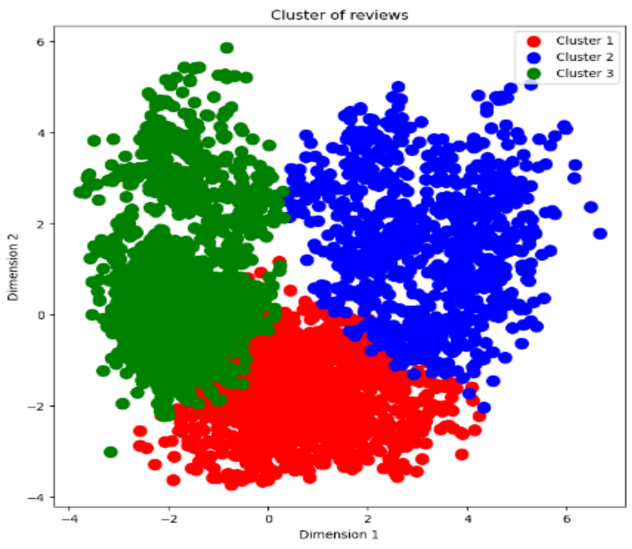
```
In [107]:  #There are 24 dimensions in this dataset. So it is not possible to visualize our KMeans in 24 Dimensions
            #Hence we will use PCA technique to reduce the dimensionality
            from sklearn.decomposition import PCA

            # pca = PCA(n_components = 2)
            pca = PCA(n_components = 10)
            scaled_review_reduced = pca.fit_transform(scaled_review)

            pca.explained_variance_ratio_

Out[107]:  array([0.19300529, 0.15157134, 0.0796704 , 0.06826117, 0.05422493,
                  0.04708264, 0.04618794, 0.04013572, 0.03333572, 0.03015863])

In [108]:  #how much of the original variance is explained by the two dimensions
            sum(pca.explained_variance_ratio_)

Out[108]:  0.743633783362123

In [109]:  pca_2d = scaled_review_reduced[:, :2]
```

<u>Visualising the clusters:</u>

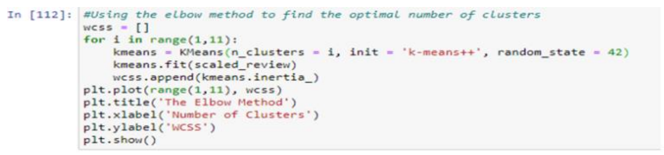
Below figure shows the code for visualizing the clusters on a 2D space. We will be visualizing the 3 clusters below. In the next step we will use a elbow curve to check if we could get more clarity by increasing the clusters.



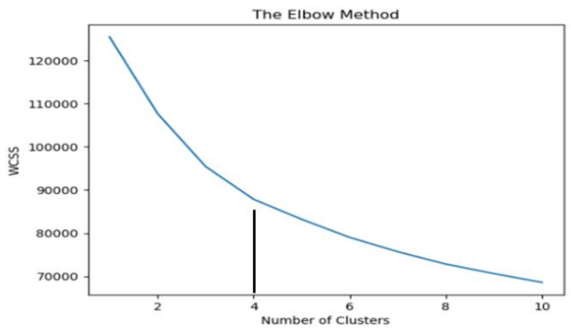
```
In [110]: #Visualising the clusters
          colours = ['red', 'blue', 'green']
          plt.figure(figsize = (8,8))
          for i in range(3):
              plt.scatter(scaled_review_reduced[y_kmeans_3 == i,0], scaled_review_reduced[y_kmeans_3 == i,1],s = 100, c = colours[i], labe

          plt.title('Cluster of reviews')
          plt.xlabel('Dimension 1')
          plt.ylabel('Dimension 2')
          plt.legend()
          plt.show()
```



<u>Elbow Curve:</u>Figure below shows the implementation and output of the elbow method that we are using to determine the number of clusters from the elbow curve. WCSS in Y axis is calculated for each value of k and this decreases as k value increases forming a sharp decrease which is considered as an elbow and we take the value of k at this point. In the output below we can interpret that there is an elbow at k=4. This means we can perform KMeans with 4 clusters.

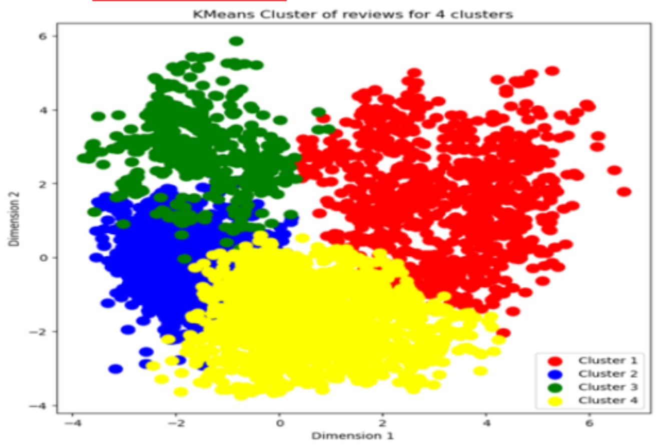


```
In [112]: #Using the elbow method to find the optimal number of clusters
          wcss = []
          for i in range(1,11):
              kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
              kmeans.fit(scaled_review)
              wcss.append(kmeans.inertia_)
          plt.plot(range(1,11), wcss)
          plt.title('The Elbow Method')
          plt.xlabel('Number of Clusters')
          plt.ylabel('WCSS')
          plt.show()
```



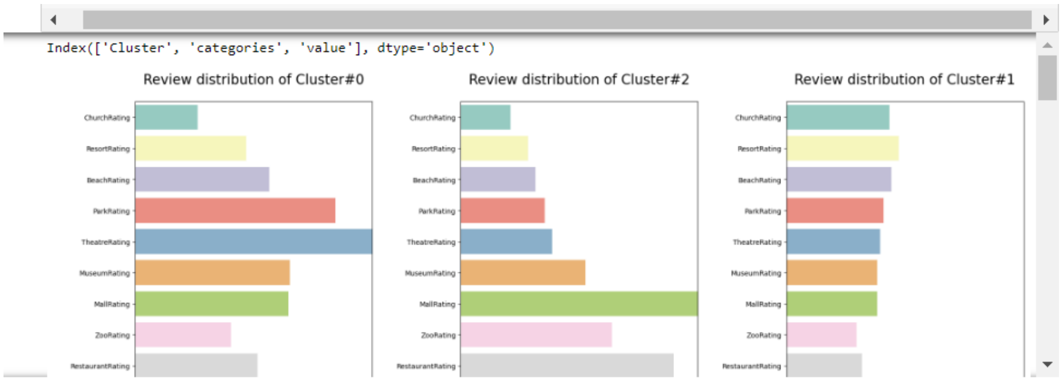
<u>KMeans with 4 clusters :</u> Below figure shows the Silhouette score and scatter plot with 4 clusters. The silhouette score has improved very minimal compared to the one with 3 clusters. However we will visualize and check what features are present in each cluster by taking 4 clusters.

```
In [113]: #Fitting K-Means to the dataset
          from sklearn.cluster import KMeans
          kmeans_4 = KMeans(n_clusters = 4, init = 'k-means++', random_state = 42)
          y_kmeans_4 = kmeans_4.fit_predict(scaled_review)
          labels_4 = kmeans_4.labels_

In [114]: #Calculate the Silhouette coefficient for 3 clusters
          silhouette_score(scaled_review,labels_4)

Out[114]: 0.14780689589404616
```



KMeans Cluster Profiling - Visualizing the features assigned to each cluster:

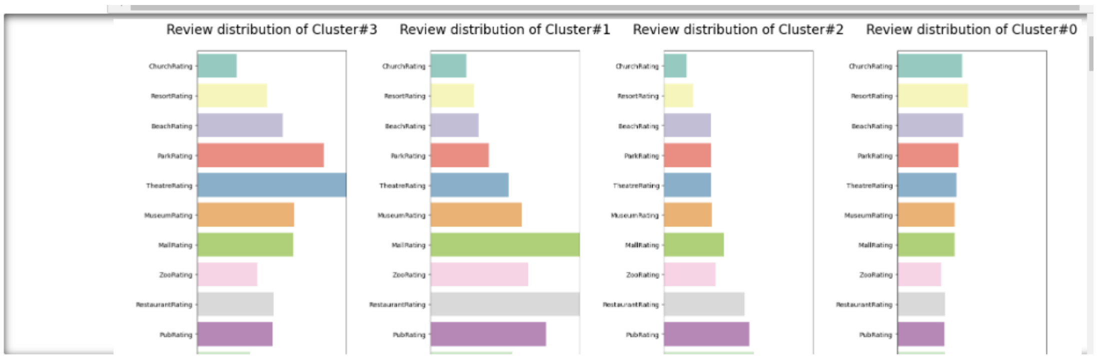
3 Clusters:



```
Index(['Cluster', 'categories', 'value'], dtype='object')
```

```
From above we can Analyze the following
* Cluster 0 : People given positive ratings for Beach,Park, theatre, museum, mall => Interested in ENtertainment
* Cluster 1 : People giving low to moderate ratings in all : least interested
* Cluster 2 : People giving high ratings for Restaurants and pubs => Interested in foods
```

4 Clusters:



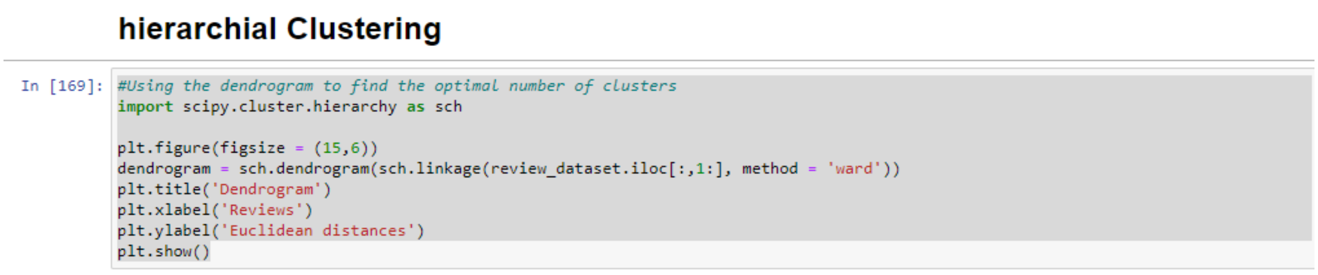
From above we can Analyze the following

- Cluster 1 : People giving low to moderate ratings in all
- Cluster 2 : People giving high ratings for mall restaurants and pubs
- Cluster 3 : People giving high ratings for Burgers, Hotels, Juice Bar, Art Galleries
- Cluster 4 : People giving high ratings for Beach, Park, Museum, Theatre

From the above two figures we can interpret that having 4 clusters help us group more precisely. Hence for K Means 4 cluster will be a good choice.

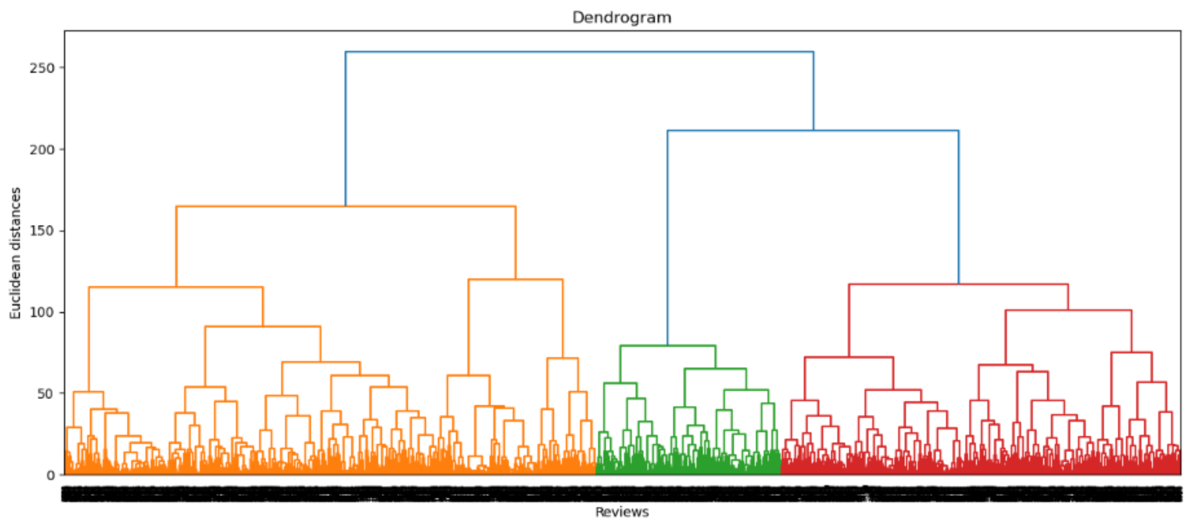
Hierachial Clustering

The below code plots a dendogram which helps us visualize the hierachial clustering. From the output, we can interpret that we can take the number of clusters as 3 if we imagine a vertical line is drawn through the tallest vertical line.

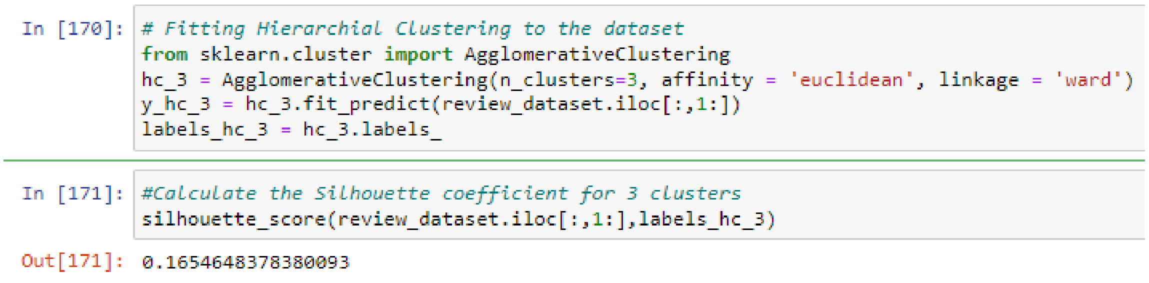


hierarchial Clustering

```
In [169]: #Using the dendrogram to find the optimal number of clusters
          import scipy.cluster.hierarchy as sch

          plt.figure(figsize = (15,6))
          dendrogram = sch.dendrogram(sch.linkage(review_dataset.iloc[:,1:], method = 'ward'))
          plt.title('Dendrogram')
          plt.xlabel('Reviews')
          plt.ylabel('Euclidean distances')
          plt.show()
```



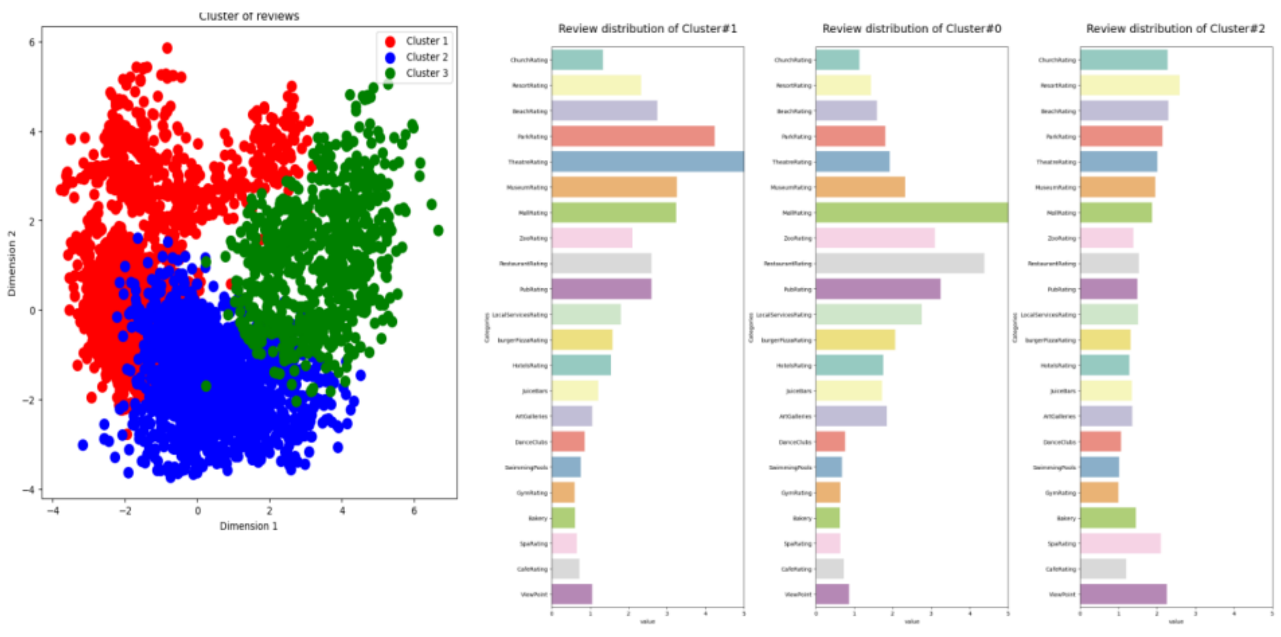
<u>**Performing hierarchical clustering with 3 clusters:**</u> The below code uses scikit-learn's AgglomerativeClustering algorithm to perfrom clustering with 3 clusters. As we can observe the Silhouette score turned out to be 0.165.



```
In [170]: # Fitting Hierarchial Clustering to the dataset
          from sklearn.cluster import AgglomerativeClustering
          hc_3 = AgglomerativeClustering(n_clusters=3, affinity = 'euclidean', linkage = 'ward')
          y_hc_3 = hc_3.fit_predict(review_dataset.iloc[:,1:])
          labels_hc_3 = hc_3.labels_

In [171]: #Calculate the Silhouette coefficient for 3 clusters
          silhouette_score(review_dataset.iloc[:,1:],labels_hc_3)

Out[171]: 0.1654648378380093
```

<u>Hierarchial Cluster Profiling - Visualizing the features assigned to each cluster:</u>

<u>3 Clusters:</u>



From above we can Analyze the following

- Cluster 0 : People given high ratings for Mall, Zoo, Restaurant,Pub, local Service => Family Day Out
- Cluster 1 : People giving High rating for beach, parks, theatres, museum => Short Time Visits
- Cluster 2 : People giving mode rating for all => Neutral

6. Evaluation of Results

- Analyzing the Silhouette score, 3 clusters generated by hierarchial Clustering would be recommended for further Model Building process.
- We were able to categorize the clusters to 3 different categories: Food and Entertainment, Short Time Visits, Neutral.
- These grouping can help to build any recommending application according to the interests of the users.

Finally, for further model building, the Silhouette score strongly suggests using three clusters in hierarchical clustering. The three found clusters (Food and Entertainment, Short Time Visits, and Neutral) provide useful information for developing applications that provide tailored recommendations. These clusters show different user preferences, ranging from a neutral interest group to a desire for short visits and interests in food and entertainment. The outcomes offer a tactical basis for augmenting user involvement, permitting customized offerings, and guiding focused promotional endeavors.

Part Three: Text Classification and Sentiment Analysis

1. Introduction

Envision discovering an immense collection of film critiques, every one conveying its distinct viewpoint and emotion. This is the core of the IMDB dataset, an extensive compilation of movie reviews that provides an insight into the realm of public opinion and cinema criticism.

We will use the capabilities of natural language processing (NLP), namely the Multinomial Naive Bayes (MultinomialNB) approach, to explore this linguistic terrain. Like a proficient language interpreter, our algorithm will assist us in identifying the positive and negative sentiments contained in these reviews.

We aim to evaluate MultinomialNB's performance in this assignment by determining how well it can discern the underlying emotion of movie reviews. Understanding how it performs will help us recognize its advantages and disadvantages in the context of text-based analysis.

2. Research Objective

"To apply Multinomial Naive Bayes for text classification and perform sentiment analysis on the IMDB dataset to discern sentiment patterns and evaluate classification performance."

The goal of this study is to delve into the complexities of sentiment analysis by investigating how Multinomial Naive Bayes, a commonly used text classification algorithm, performs on the IMDB dataset. The project intends to use this classification approach to reveal the underlying sentiment structure in movie reviews, distinguishing between positive and negative thoughts.

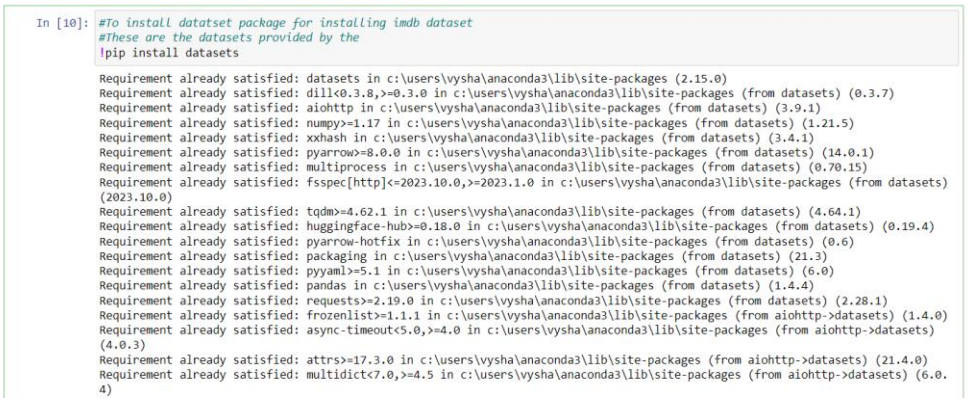
This study is significant because it has the potential to contribute to a better understanding of sentiment analysis applications in the context of movie reviews. Analyzing the sentiment distribution across various reviews provides insights into audience perspectives, perhaps assisting filmmakers, critics, and platforms in measuring movie response. Furthermore, assessing the accuracy and performance metrics of the classification model provides a quantitative layer, confirming the dependability of the chosen approach for sentiment analysis.

3. Overview of dataset

IMDB dataset (Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. 2011, June 1) having 50K movie reviews for natural language processing or Text analytics. This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. We provide a set of 25,000 highly polar movie reviews for training and 25,000 for testing. So, predict the number of positive and negative reviews using either classification or deep learning algorithms.

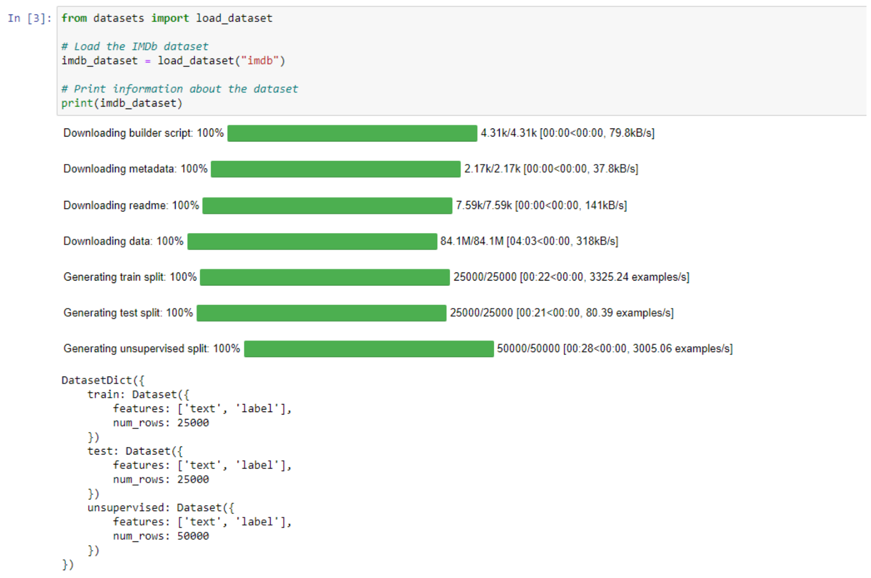
For this task we will be using the unsupervised dataset of imdb review dataset that has 50000 instances in it. To download the dataset, we will follow some of the below steps:

Step1: Installing the dataset via the Python Package Index (PyPi)



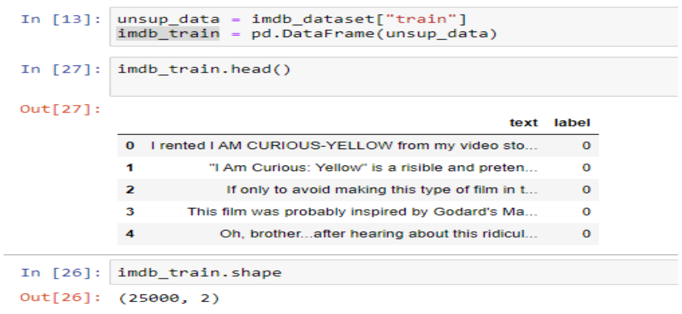
```
In [10]: #To install datatset package for installing imdb dataset
         #These are the datasets provided by the
         !pip install datasets

         Requirement already satisfied: datasets in c:\users\vysha\anaconda3\lib\site-packages (2.15.0)
         Requirement already satisfied: dill<0.3.8,>=0.3.0 in c:\users\vysha\anaconda3\lib\site-packages (from datasets) (0.3.7)
         Requirement already satisfied: aiohttp in c:\users\vysha\anaconda3\lib\site-packages (from datasets) (3.9.1)
         Requirement already satisfied: numpy>=1.17 in c:\users\vysha\anaconda3\lib\site-packages (from datasets) (1.21.5)
         Requirement already satisfied: xxhash in c:\users\vysha\anaconda3\lib\site-packages (from datasets) (3.4.1)
         Requirement already satisfied: pyarrow>=8.0.0 in c:\users\vysha\anaconda3\lib\site-packages (from datasets) (14.0.1)
         Requirement already satisfied: multiprocess in c:\users\vysha\anaconda3\lib\site-packages (from datasets) (0.70.15)
         Requirement already satisfied: fsspec[http]<=2023.10.0,>=2023.1.0 in c:\users\vysha\anaconda3\lib\site-packages (from datasets)
         (2023.10.0)
         Requirement already satisfied: tqdm>=4.62.1 in c:\users\vysha\anaconda3\lib\site-packages (from datasets) (4.64.1)
         Requirement already satisfied: huggingface-hub>=0.18.0 in c:\users\vysha\anaconda3\lib\site-packages (from datasets) (0.19.4)
         Requirement already satisfied: pyarrow-hotfix in c:\users\vysha\anaconda3\lib\site-packages (from datasets) (0.6)
         Requirement already satisfied: packaging in c:\users\vysha\anaconda3\lib\site-packages (from datasets) (21.3)
         Requirement already satisfied: pyyaml>=5.1 in c:\users\vysha\anaconda3\lib\site-packages (from datasets) (6.0)
         Requirement already satisfied: pandas in c:\users\vysha\anaconda3\lib\site-packages (from datasets) (1.4.4)
         Requirement already satisfied: requests>=2.19.0 in c:\users\vysha\anaconda3\lib\site-packages (from datasets) (2.28.1)
         Requirement already satisfied: frozenlist>=1.1.1 in c:\users\vysha\anaconda3\lib\site-packages (from aiohttp->datasets) (1.4.0)
         Requirement already satisfied: async-timeout<5.0,>=4.0 in c:\users\vysha\anaconda3\lib\site-packages (from aiohttp->datasets)
         (4.0.3)
         Requirement already satisfied: attrs>=17.3.0 in c:\users\vysha\anaconda3\lib\site-packages (from aiohttp->datasets) (21.4.0)
         Requirement already satisfied: multidict<7.0,>=4.5 in c:\users\vysha\anaconda3\lib\site-packages (from aiohttp->datasets) (6.0.
         4)
```

Step2: Loading the imdb dataset from the dataset library. Figure below shows that by running the load_dataset module of dataset library, it downloads all the files of imdb dataset. The resultant imdb_dataset is a dictionary with train, test and unsupervised datasets.



```
In [3]: from datasets import load_dataset

        # Load the IMDb dataset
        imdb_dataset = load_dataset("imdb")

        # Print information about the dataset
        print(imdb_dataset)

        Downloading builder script: 100%  4.31k/4.31k [00:00<00:00, 79.8kB/s]
        Downloading metadata: 100%  2.17k/2.17k [00:00<00:00, 37.8kB/s]
        Downloading readme: 100%  7.59k/7.59k [00:00<00:00, 141kB/s]
        Downloading data: 100%  84.1M/84.1M [04:03<00:00, 318kB/s]
        Generating train split: 100%  25000/25000 [00:22<00:00, 3325.24 examples/s]
        Generating test split: 100%  25000/25000 [00:21<00:00, 80.39 examples/s]
        Generating unsupervised split: 100%  50000/50000 [00:28<00:00, 3005.06 examples/s]

        DatasetDict({
            train: Dataset({
                features: ['text', 'label'],
                num_rows: 25000
            })
            test: Dataset({
                features: ['text', 'label'],
                num_rows: 25000
            })
            unsupervised: Dataset({
                features: ['text', 'label'],
                num_rows: 50000
            })
        })
```

Step 3: Accessing the train data from imdb_dataset dictionary. Figure below shows that we are using our train data for our analysis. This data should have 25000 reviews.



```
In [13]: unsup_data = imdb_dataset["train"]
         imdb_train = pd.DataFrame(unsup_data)

In [27]: imdb_train.head()

Out[27]:
```

	text	label
0	I rented I AM CURIOUS-YELLOW from my video sto...	0
1	"I Am Curious: Yellow" is a risible and preten...	0
2	If only to avoid making this type of film in t...	0
3	This film was probably inspired by Godard's Ma...	0
4	Oh, brother...after hearing about this ridicul...	0

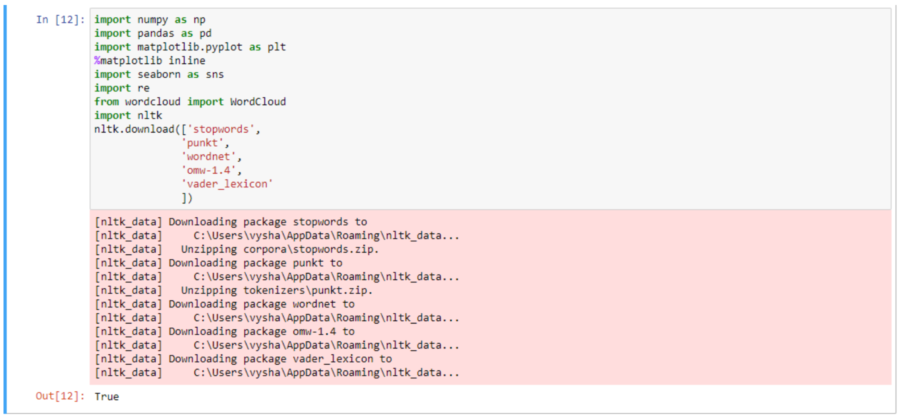
```
In [26]: imdb_train.shape
Out[26]: (25000, 2)
```

Steps to load the dataset is from: https://huggingface.co/docs/datasets/load_hub.

Information about the above source: https://toolbox.google.com/datasetsearch For more dataset information, please go through the following link, http://ai.stanford.edu/~amaas/data/sentiment/

4. Text Classification

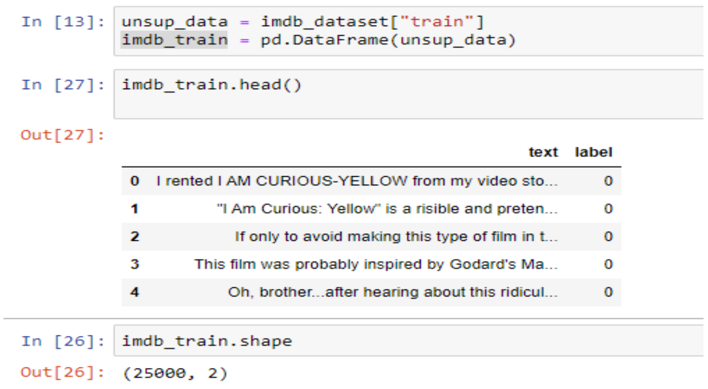
Importing the required libraries : Below figure shows the required libraries. Apart from the previous we will be making use of some nltk libraries like stopwords, wordnet for text processing and word cloud to display the categorized review. We will see this in detail in later sections.



```
In [12]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         %matplotlib inline
         import seaborn as sns
         import re
         from wordcloud import WordCloud
         import nltk
         nltk.download(['stopwords',
                       'punkt',
                       'wordnet',
                       'omw-1.4',
                       'vader_lexicon'
                       ])

         [nltk_data] Downloading package stopwords to
         [nltk_data]     C:\Users\vysha\AppData\Roaming\nltk_data...
         [nltk_data]   Unzipping corpora\stopwords.zip.
         [nltk_data] Downloading package punkt to
         [nltk_data]     C:\Users\vysha\AppData\Roaming\nltk_data...
         [nltk_data]   Unzipping tokenizers\punkt.zip.
         [nltk_data] Downloading package wordnet to
         [nltk_data]     C:\Users\vysha\AppData\Roaming\nltk_data...
         [nltk_data] Downloading package omw-1.4 to
         [nltk_data]     C:\Users\vysha\AppData\Roaming\nltk_data...
         [nltk_data] Downloading package vader_lexicon to
         [nltk_data]     C:\Users\vysha\AppData\Roaming\nltk_data...

Out[12]: True
```

Loading the dataset and checking the dimensions

In the below screenshot, the train data is loaded and a dataframe is created using the dataframe module of the pandas library. The used dataset have 25000 rows and 2 columns.



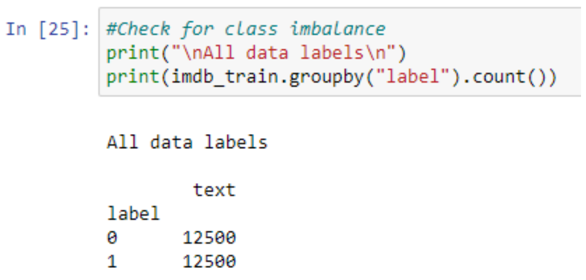
```
In [13]: unsup_data = imdb_dataset["train"]
         imdb_train = pd.DataFrame(unsup_data)

In [27]: imdb_train.head()

Out[27]:
```

	text	label
0	I rented I AM CURIOUS-YELLOW from my video sto...	0
1	"I Am Curious: Yellow" is a risible and preten...	0
2	If only to avoid making this type of film in t...	0
3	This film was probably inspired by Godard's Ma...	0
4	Oh, brother...after hearing about this ridicul...	0

```
In [26]: imdb_train.shape

Out[26]: (25000, 2)
```

Check for class imbalance

The figure below shows the result of the class imbalance. For this we use target column label and count the unique occurrences. We can observe that the class is clearly balanced with equal number of 0 and 1 results.



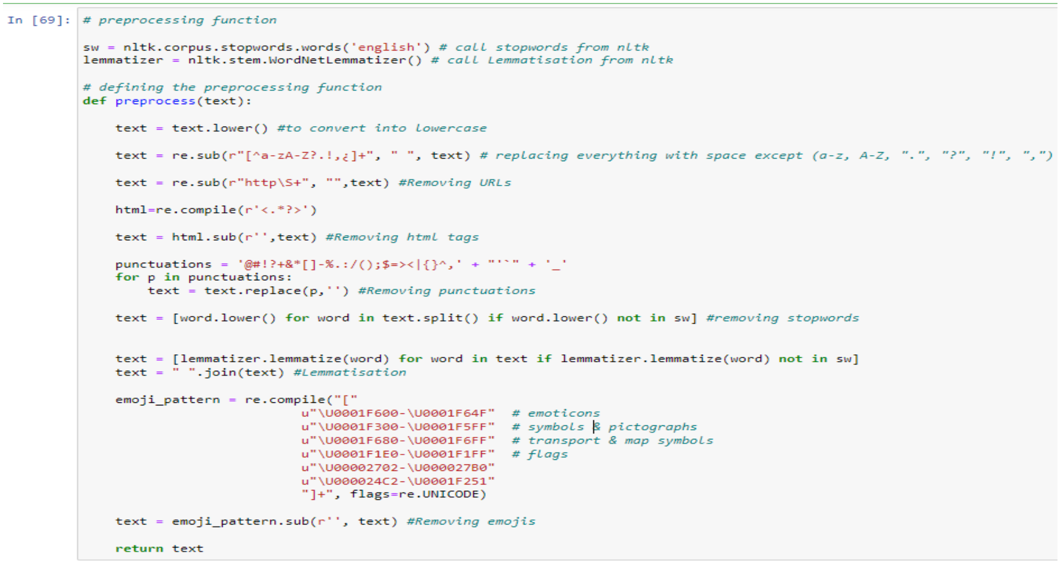
```
In [25]: #Check for class imbalance
         print("\nAll data labels\n")
         print(imdb_train.groupby("label").count())

         All data labels

                 text
         label
         0      12500
         1      12500
```

Text Preprocessing:

The preprocessing function incorporates various steps to clean and prepare the text data for analysis. The following key aspects are covered:

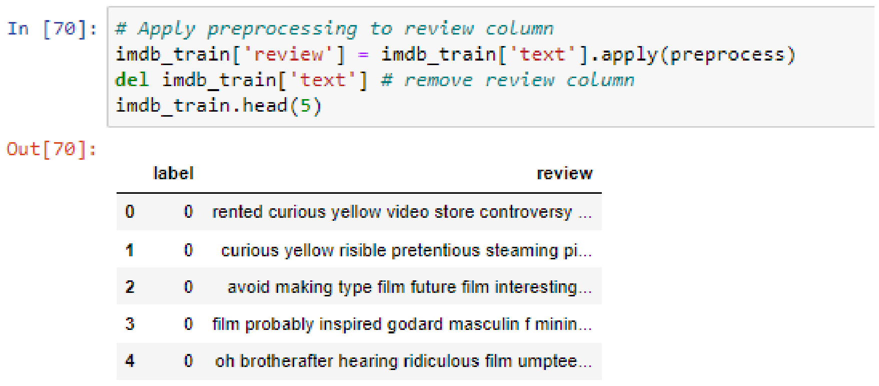
- Lowercasing: Consistent casing for every words to reduce dimensionality.
- Removing special characters: Special symbols, URLs and HTML tags donot contribute much to the sentiment analysis. Hence, we will remove those.
- Removing Punctuations.
- Removing Stopwords of English language.
- Lemitization to reduce words to their root form to achieve standardization.
- Removing emojis as they don't contribute enough for sentiment analysis.

Below figure shows the function defined to achieve this



```
In [69]: # preprocessing function

         sw = nltk.corpus.stopwords.words('english') # call stopwords from nltk
         lemmatizer = nltk.stem.WordNetLemmatizer() # call Lemmatisation from nltk

         # defining the preprocessing function
         def preprocess(text):

             text = text.lower() #to convert into lowercase

             text = re.sub(r"[^a-zA-Z?.!,¿]+", " ", text) # replacing everything with space except (a-z, A-Z, ".", "?", "!", ",")

             text = re.sub(r"http\S+", "",text) #Removing URLs

             html=re.compile(r'<.*?>')

             text = html.sub(r'',text) #Removing html tags

             punctuations = '@#!?+&*[]-%.:/();$=><|{}^,' + "'`" + '_'
             for p in punctuations:
                 text = text.replace(p,'') #Removing punctuations

             text = [word.lower() for word in text.split() if word.lower() not in sw] #removing stopwords


             text = [lemmatizer.lemmatize(word) for word in text if lemmatizer.lemmatize(word) not in sw]
             text = " ".join(text) #Lemmatisation

             emoji_pattern = re.compile("["
                                    u"\U0001F600-\U0001F64F"  # emoticons
                                    u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                                    u"\U0001F680-\U0001F6FF"  # transport & map symbols
                                    u"\U0001F1E0-\U0001F1FF"  # flags
                                    u"\U00002702-\U000027B0"
                                    u"\U000024C2-\U0001F251"
                                    "]+", flags=re.UNICODE)

             text = emoji_pattern.sub(r'', text) #Removing emojis

             return text
```

Next, as shown below we take all the elements in the text column which holds all the review and call the preprocess function to eliminate all those which are not required for our analysis. This will return a new column 'review' that will have the preprocessed text which will look something like this

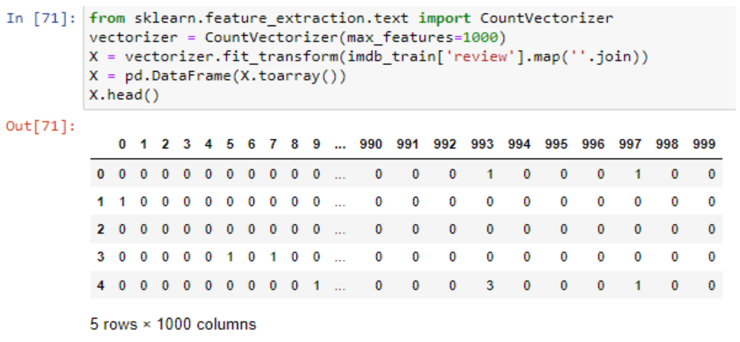


```
In [70]: # Apply preprocessing to review column
         imdb_train['review'] = imdb_train['text'].apply(preprocess)
         del imdb_train['text'] # remove review column
         imdb_train.head(5)
```

Out[70]:

	label	review
0	0	rented curious yellow video store controversy ...
1	0	curious yellow risible pretentious steaming pi...
2	0	avoid making type film future film interesting...
3	0	film probably inspired godard masculin f minin...
4	0	oh brotherafter hearing ridiculous film umptee...

<u>Text Vectorization:</u> We create a matrix of token counts from a series of text documents using scikit-learn's CountVectorizer. The maximum number of words is limited to 1000 by setting the max_feature option to 1000. This is necessary since, if the parameter is left unset, our dataset generates an enormous matrix of tokens and could result in a memory usage problem. Fit_transform tokenizes a document, each row representing a word, and returns a space matrix. This matrix is finally transformed into a dataframe called "X."



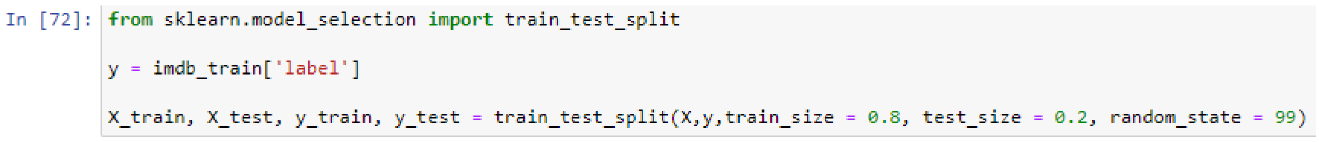
```
In [71]: from sklearn.feature_extraction.text import CountVectorizer
         vectorizer = CountVectorizer(max_features=1000)
         X = vectorizer.fit_transform(imdb_train['review'].map(''.join))
         X = pd.DataFrame(X.toarray())
         X.head()
```

Out[71]:

	0	1	2	3	4	5	6	7	8	9	...	990	991	992	993	994	995	996	997	998	999
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	1	...	0	0	0	3	0	0	0	1	0	0

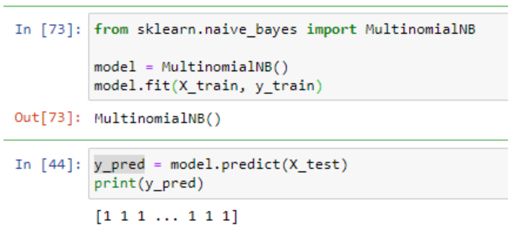
5 rows × 1000 columns

Model Building: Below figure shows assigning of target variable and splitting the dataset to train and test data.



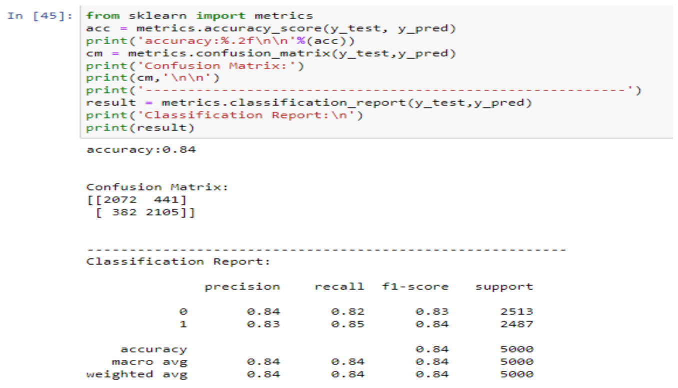
```
In [72]: from sklearn.model_selection import train_test_split

         y = imdb_train['label']

         X_train, X_test, y_train, y_test = train_test_split(X,y,train_size = 0.8, test_size = 0.2, random_state = 99)
```

In the below figure we make use of Multinomial Naive Bayes, a popular text classification task. Finally the model.fit() will fit the model to the training data.



```
In [73]: from sklearn.naive_bayes import MultinomialNB

         model = MultinomialNB()
         model.fit(X_train, y_train)

Out[73]: MultinomialNB()

In [44]: y_pred = model.predict(X_test)
         print(y_pred)

         [1 1 1 ... 1 1 1]
```

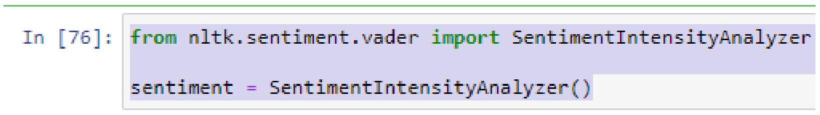
Evaluation of result: The model predicts with an accuracy of 84% with 2105 TP and 2072 TN.



```
In [45]: from sklearn import metrics
         acc = metrics.accuracy_score(y_test, y_pred)
         print('accuracy:%.2f\n\n'%(acc))
         cm = metrics.confusion_matrix(y_test,y_pred)
         print('Confusion Matrix:')
         print(cm,'\n\n')
         print('-----------------------------------------------------------')
         result = metrics.classification_report(y_test,y_pred)
         print('Classification Report:\n')
         print(result)

         accuracy:0.84


         Confusion Matrix:
         [[2072  441]
          [ 382 2105]]


         -----------------------------------------------------------
         Classification Report:

                       precision    recall  f1-score   support

                    0       0.84      0.82      0.83      2513
                    1       0.83      0.85      0.84      2487

             accuracy                           0.84      5000
            macro avg       0.84      0.84      0.84      5000
         weighted avg       0.84      0.84      0.84      5000
```

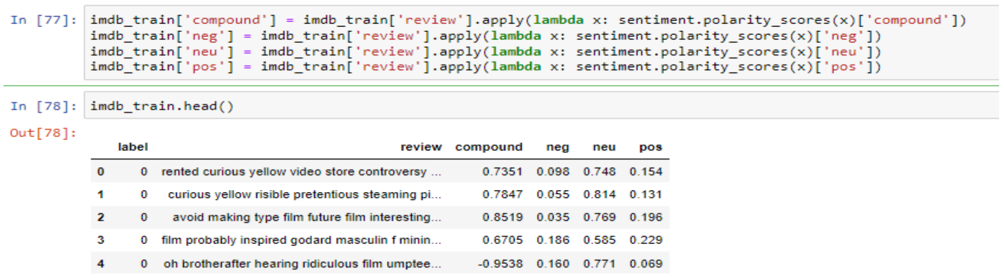
5. Sentiment Analysis

SentimentIntensityAnalyzer is a part of NLTK designed for sentiment analysis using VADER analysis tool. It returns a dictionary sentiment with values like neg, compound, pos and neu.



```
In [76]: from nltk.sentiment.vader import SentimentIntensityAnalyzer

         sentiment = SentimentIntensityAnalyzer()
```

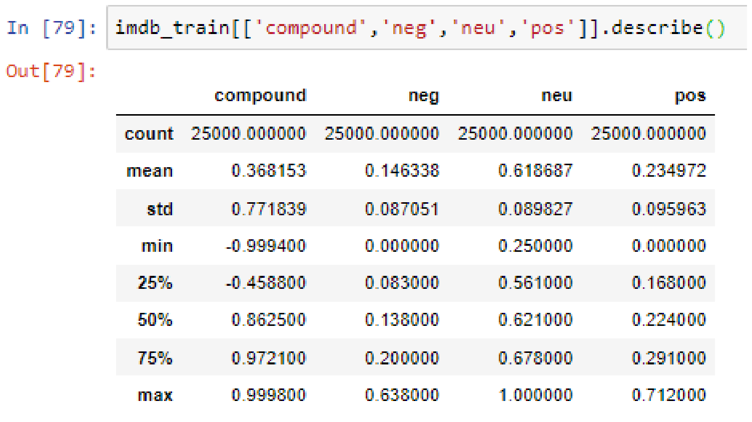
Below code runs a lambda function for every values and create additional columns in the existing dataframe with sentiment score for each value.



```
In [77]: imdb_train['compound'] = imdb_train['review'].apply(lambda x: sentiment.polarity_scores(x)['compound'])
         imdb_train['neg'] = imdb_train['review'].apply(lambda x: sentiment.polarity_scores(x)['neg'])
         imdb_train['neu'] = imdb_train['review'].apply(lambda x: sentiment.polarity_scores(x)['neu'])
         imdb_train['pos'] = imdb_train['review'].apply(lambda x: sentiment.polarity_scores(x)['pos'])

In [78]: imdb_train.head()

Out[78]:
```

	label	review	compound	neg	neu	pos
0	0	rented curious yellow video store controversy ...	0.7351	0.098	0.748	0.154
1	0	curious yellow risible pretentious steaming pi...	0.7847	0.055	0.814	0.131
2	0	avoid making type film future film interesting...	0.8519	0.035	0.769	0.196
3	0	film probably inspired godard masculin f minin...	0.6705	0.186	0.585	0.229
4	0	oh brotherafter hearing ridiculous film umptee...	-0.9538	0.160	0.771	0.069

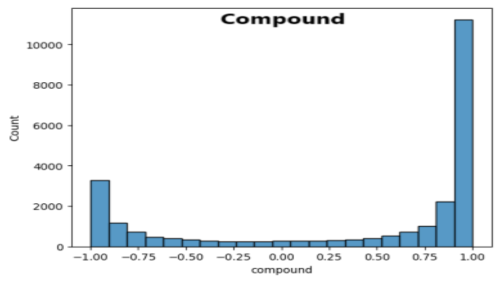
The below five point summary for the compound variable have a positive mean and median value suggesting an overall positive sentiment with less negative sentiment. The maximum value suggest that a very high scoring text have a positive sentiment.

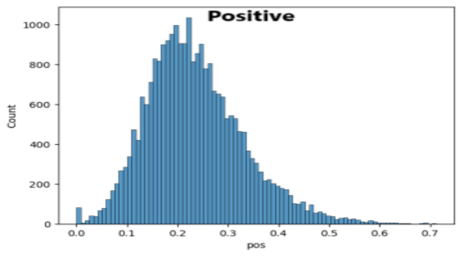


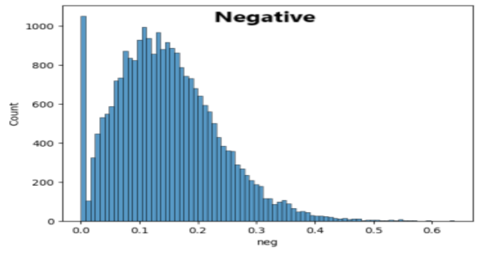
```
In [79]: imdb_train[['compound','neg','neu','pos']].describe()

Out[79]:
```

	compound	neg	neu	pos
count	25000.000000	25000.000000	25000.000000	25000.000000
mean	0.368153	0.146338	0.618687	0.234972
std	0.771839	0.087051	0.089827	0.095963
min	-0.999400	0.000000	0.250000	0.000000
25%	-0.458800	0.083000	0.561000	0.168000
50%	0.862500	0.138000	0.621000	0.224000
75%	0.972100	0.200000	0.678000	0.291000
max	0.999800	0.638000	1.000000	0.712000

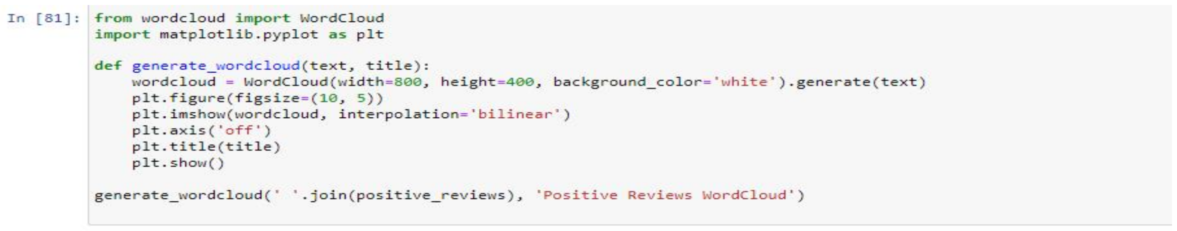
Below graphs show the distribution of compound, positive and negative scores. Positive looks like a normal distribution and negative seems to be right skewed.



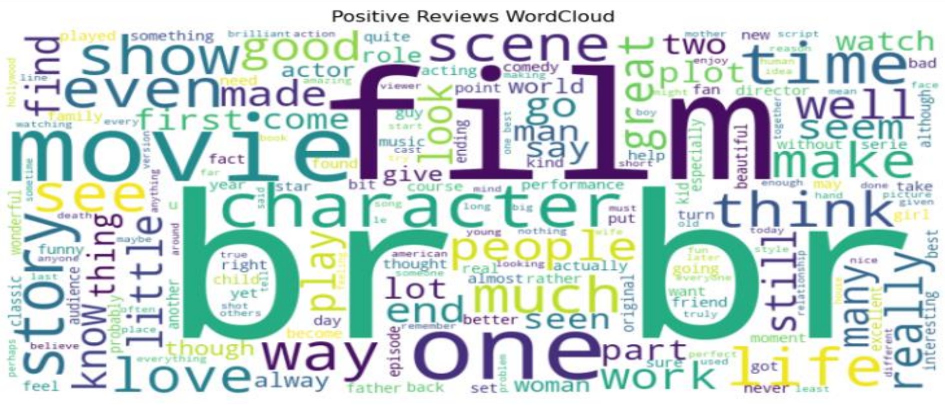


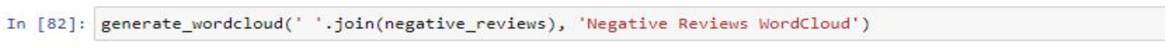


6. Evaluation of Result

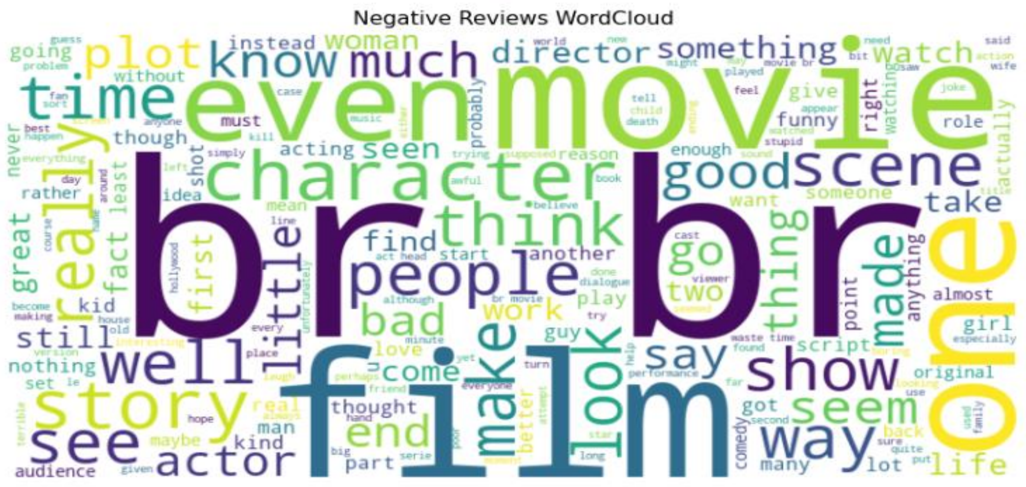


```
In [81]: from wordcloud import WordCloud
         import matplotlib.pyplot as plt

         def generate_wordcloud(text, title):
             wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)
             plt.figure(figsize=(10, 5))
             plt.imshow(wordcloud, interpolation='bilinear')
             plt.axis('off')
             plt.title(title)
             plt.show()

         generate_wordcloud(' '.join(positive_reviews), 'Positive Reviews WordCloud')
```





```
In [82]: generate_wordcloud(' '.join(negative_reviews), 'Negative Reviews WordCloud')
```



In conclusion, we successfully distinguished between positive and negative reviews with an amazing 84% accuracy rate in our text classification and sentiment analysis on the IMDb review dataset using the Multinomial Naive Bayes (MNB) model. The model's strong performance is demonstrated by the confusion matrix, which has 2072 true negatives, 2105 true positives, 441 erroneous positives, and 382 false negatives. The model's accuracy in making predictions is supported by its balanced performance for both classes as demonstrated by measures like as precision, recall, and F1-score.

Additionally, the sentiment analysis word cloud provides a visual depiction of the most influential terms in the dataset and includes both positive and negative ratings. This qualitative investigation offers insightful information on the common opinions voiced by reviewers. terms like “Good, brilliant action, excellent, interesting, best, beautiful, funny” are indicative of positive sentiments, whereas terms like “mean, poor, unfortunately, waste time, awful, never, problem” are indicative of negative sentiments.

Part Four: References and Appendices

Mushroom Dataset: https://archive.ics.uci.edu/dataset/73/mushroom

Google Review Dataset: https://archive.ics.uci.edu/dataset/485/tarvel+review+ratings

IMDB Review Dataset: https://ai.stanford.edu/~amaas/data/sentiment/

IMDB Dataset: Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., & Potts, C. (2011, June 1). *Learning Word Vectors for Sentiment Analysis*. ACLWeb; Association for Computational Linguistics. http://www.aclweb.org/anthology/P11-1015