ELG 5255: Applied Machine Learning

Assignment:2

BY: Group 5

Anas Elbattra

Ahmed Badawy

Esraa Fayad

Part 1→

1-

"Applied Machine Learning"

Part 1:—

* We have → 14 Samples.
→ 3 Features.

Calculating Probabilities

$P(\text{Stolen} = \text{Yes}) = 6/14$

$P(\text{Not Stolen} = \text{No}) = 8/14$

Color	Yes	No
Red	3/6	4/8
Yellow	2/6	2/8
Blue	1/6	2/8

type	Yes	No
Sports	4/6	3/8
SUV	2/6	5/8

origin	Yes	No
Domestic	2/6	5/8
Imported	4/6	3/8

Given New instance:—

NewInstance = (Blue, SUV, Domestic)
" Yes oR No "

$x = \langle$ Color = Blue, type = SUV, origin = Domestic $\rangle$

* $P(\text{color} = \text{Blue} \mid \text{Stolen} = \text{Yes})$

$= \frac{1}{6}$

* $P(\text{color} = \text{Blue} \mid \text{Stolen} = \text{No})$

$= \frac{2}{8}$

* $P(\text{type} = \text{SUV} \mid \text{stolen} = \text{Yes})$

$= \frac{2}{6}$

* $P(\text{type} = \text{SUV} \mid \text{stolen} = \text{No})$

$= \frac{5}{8}$

* $P(\text{origin} = \text{Domestic} \mid \text{stolen} = \text{Yes})$

$= \frac{2}{6}$

* $P(\text{origin} = \text{Domestic} \mid \text{stolen} = \text{No})$

$= \frac{5}{8}$

$P(\text{Yes} \mid X) = [P(\text{Blue} \mid \text{Yes})\, P(\text{SUV} = \text{Yes})\, (\text{Domestic} = \text{Yes})]\, P(\text{stolen} = \text{Yes})$

$= \frac{1}{6} * \frac{2}{6} * \frac{2}{6} * \frac{6}{14}$

$= 0.00793$

$P(\text{No} \mid X) = [P(\text{Blue} \mid \text{No})\, P(\text{SUV} = \text{No})\, (\text{Domestic} = \text{Yes})]\, P(\text{stolen} = \text{Yes})$

$= \left(\frac{2}{8} * \frac{5}{8} * \frac{5}{8}\right)\left(\frac{8}{14}\right)$

$= 0.0558$

Given the fact $P(\text{No} \mid X) > P(\text{Yes} \mid X)$

we will label X to be "No"

2-

Part 1:-

2.

$\lambda_{11} = \lambda_{22} = 0$, $\lambda_{12} = 6$, $\lambda_{21} = 3$

Reject $\longrightarrow \lambda = 2$

$$\rightarrow R(\alpha_1|x) = \lambda_{11} P(C_1|x) + \lambda_{12} P(C_2|x)$$
$$= 0 * P(C_1|x) + 6 P(C_2|x)$$

$$\rightarrow R(\alpha_1|x) = 6 - 6P(C_1|x) \quad \dots \text{①}$$

$$\rightarrow R(\alpha_2|x) = \lambda_{21} P(C_1|x) + \lambda_{22} P(C_2|x)$$
$$= 3P(C_1|x) + 0 * P(C_2|x)$$

$$\rightarrow R(\alpha_2|x) = 3P(C_1|x) \quad \dots \text{②}$$

$$\rightarrow R(\alpha_l|x) = 2 \quad \dots \text{③}$$

* We choose α_1 if

$$\rightarrow R(\alpha_1|x) < 2 \Rightarrow P(C_1|x) > \frac{2}{3}$$

* We choose α_2 if

$$\rightarrow R(\alpha_2|x) < 2 \Rightarrow P(C_1|x) < \frac{2}{3}$$

$\therefore$ We Reject if $\frac{2}{3} < P(C_1|x) < \frac{2}{3}$

Part 2→ Naive Bayes

Function conf_matrix

To preview the confusion matrix and classification report

```python
def conf_matrix(x, y, title, show_report=False):
    cm = confusion_matrix(x, y)
    plt.figure(figsize=(6, 4))

    sns.heatmap(cm, annot=True, fmt='d', cmap="Oranges", cbar=False)
    plt.title(f'Confusion Matrix - {title} Data')
    plt.xlabel('Predicted Class')
    plt.ylabel('True Class')
    plt.show()
    if not show_report:
        print("Accuracy: ", accuracy_score(x, y))

    if show_report:
        report = classification_report(x, y)
        print("Classification Report:")
        print(report)
        plt.show()
```

Read Date from URL

```python
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/spambase.data'


spambase = pd.read_csv(url, header=None)
```

Show The First 5 Rows

```
spambase.head()
```

	0	1	2	3	4	5	6	7	8	9	...	48	49	50	51	52	53	54	55	56	57
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.00	0.000	0.0	0.778	0.000	0.000	3.756	61	278	1
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.00	0.132	0.0	0.372	0.180	0.048	5.114	101	1028	1
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.01	0.143	0.0	0.276	0.184	0.010	9.821	485	2259	1
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	0.137	0.0	0.137	0.000	0.000	3.537	40	191	1
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	0.135	0.0	0.135	0.000	0.000	3.537	40	191	1

5 rows × 58 columns

Show Value Counts for Spam

```
spambase[57].value_counts()
```

```
0    2788
1    1813
Name: 57, dtype: int64
```

Part 2→ (A)

Split the dataset into two parts as training data and test data

- **first 80 percent for training**
- **last 20 percent for test**

```
split_index = int(len(spambase) * 0.8)
first_80_percent= spambase[:split_index]
last_20_percent= spambase[split_index:]
```

split first_80_percent for x_train and y_train and split last_20_percen for x_test and y_test

```
x_train = first_80_percent.drop(first_80_percent.columns[-1], axis=1)
y_train = first_80_percent.iloc[:, -1]
x_test = last_20_percent.drop(first_80_percent.columns[-1], axis=1)
y_test = last_20_percent.iloc[:, -1]
```

Function "Naive_Bayes_Models"

- **this is user defined function whice apply Naive Bayes models**

- **It Takes 4 parameters Naive_Bayes_classifier , X__train , Y__train , X__test**

- **then path X__train and Y__train to fit function then path X__test to predict functoin to calculate classifier_predictions**

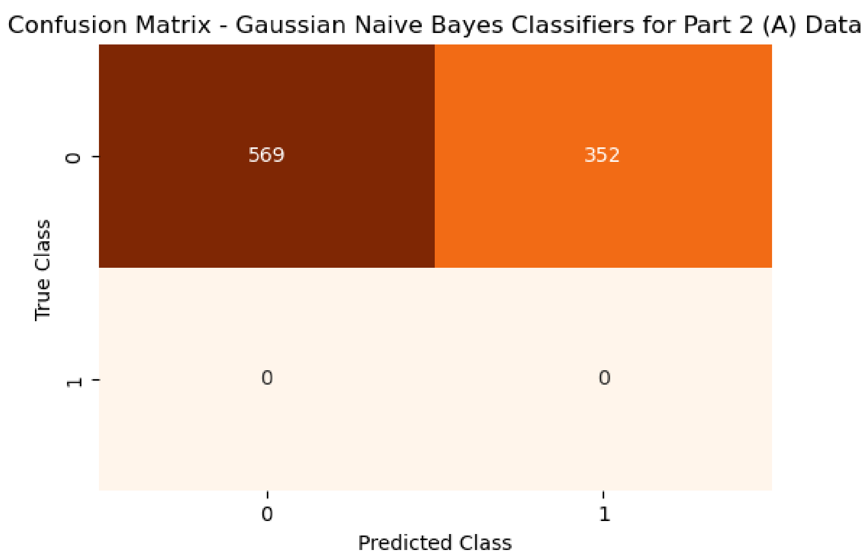
```
def Naive_Bayes_Models(classifier,X__train,Y__train,X__test):
    classifier.fit(X__train, Y__train)
    classifier_predictions = classifier.predict(X__test)
    return classifier_predictions
```

Run "Naive_Bayes_Models" Function using GaussianNB classifier

```
Gaussian_predictions=Naive_Bayes_Models(GaussianNB(),x_train,y_train,x_test)
```

Display the confusion matrix for GaussianNB classifier

```
conf_matrix(y_test,Gaussian_predictions,"Gaussian Naive Bayes Classifiers for Part 2 (A)")
```



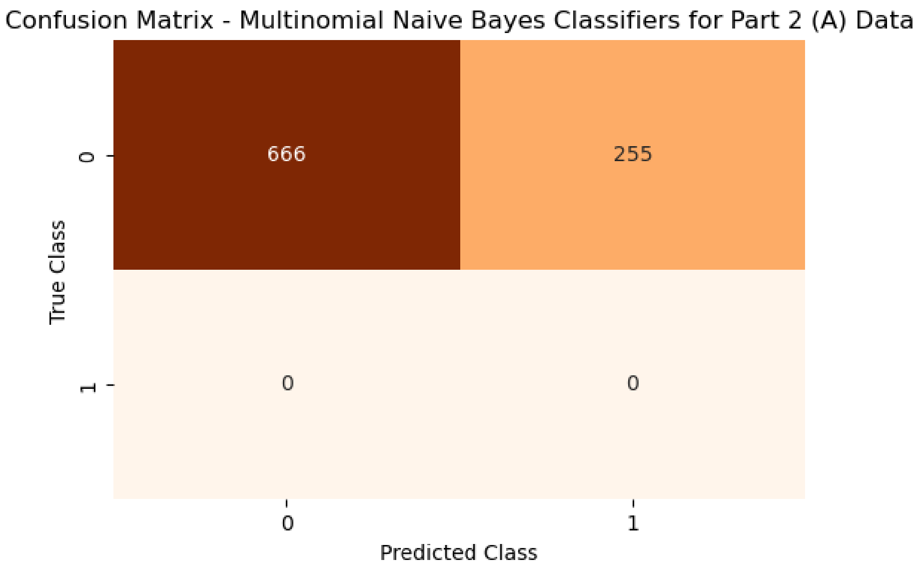

```
Accuracy:  0.6178067318132465
```

Run "Naive_Bayes_Models" Function using MultinomialNB classifier

```
MultinomialNB_predictions=Naive_Bayes_Models(MultinomialNB(),x_train,y_train,x_test)
```

Display the confusion matrix for MultinomialNB classifier

```
conf_matrix(y_test,MultinomialNB_predictions,"Multinomial Naive Bayes Classifiers for Part 2 (A)")
```




```
Accuracy:  0.7231270358306189
```

Part 2→ (B)

Put All columns except the last one in X Put the last column in Y and split X , Y using train_test_split function

```
X = spambase.iloc[:, :-1]  # All columns except the last one
y = spambase.iloc[:, -1]
```

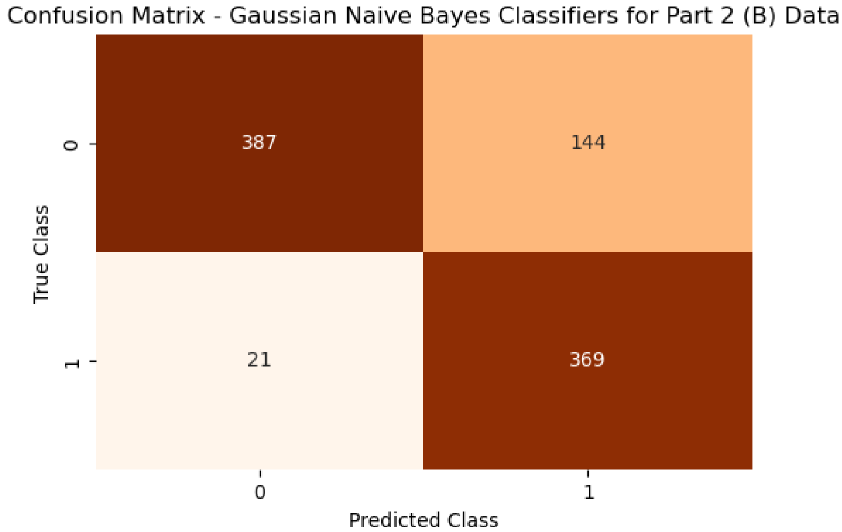
```
X_train_split, X_test_split, y_train_split, y_test_split = train_test_split(X, y, test_size=0.2, random_state=42,shuffle=True)
```

Run "Naive_Bayes_Models" Function using GaussianNB classifier based on train test split function

```
Gaussian_Use_train_split_predictions=Naive_Bayes_Models(GaussianNB(),X_train_split,y_train_split,X_test_split)
```

Display the confusion matrix for GaussianNB classifier

```
conf_matrix(y_test_split,Gaussian_Use_train_split_predictions,"Gaussian Naive Bayes Classifiers for Part 2 (B)")
```



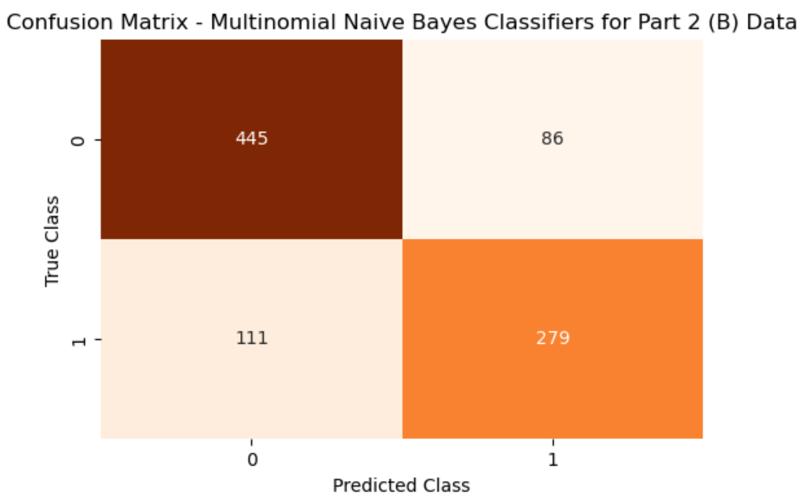

```
Accuracy:  0.8208469055374593
```

Run "Naive_Bayes_Models" Function using MultinomialNB classifier based on train test split function

```
MultinomialNB_Use_train_split_predictions=Naive_Bayes_Models(MultinomialNB(),X_train_split,y_train_split,X_test_split)
```

Display the confusion matrix for MultinomialNB classifier

```
conf_matrix(y_test_split,MultinomialNB_Use_train_split_predictions,"Multinomial Naive Bayes Classifiers for Part 2 (B)")
```




```
Accuracy:  0.7861020629750272
```

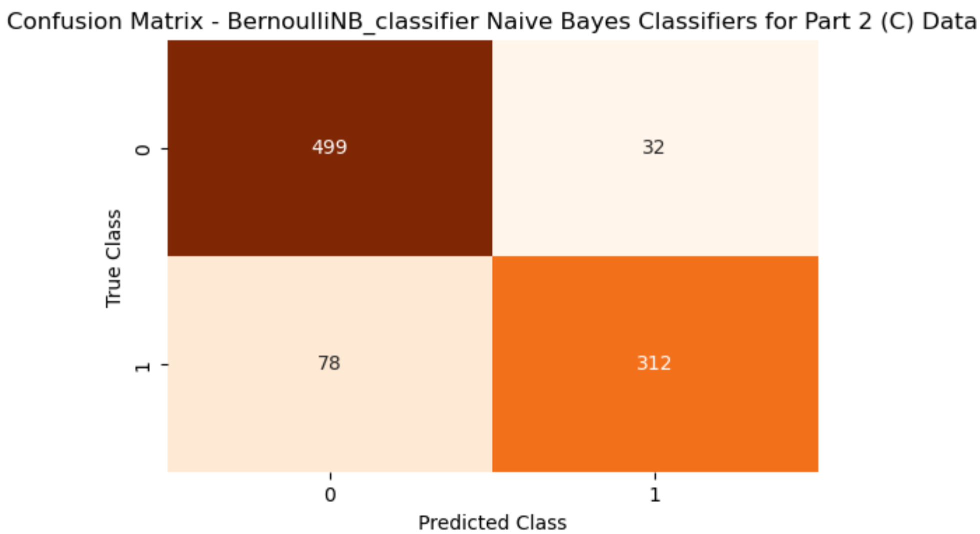
Part 2→ (C)

Run "Naive_Bayes_Models" Function using BernoulliNB classifier based on train test split function

```
BernoulliNB_Use_train_split_predictions=Naive_Bayes_Models(BernoulliNB(),X_train_split,y_train_split,X_test_split)
```

Display the confusion matrix and Classification Report for BernoulliNB classifier

```
conf_matrix(y_test_split,BernoulliNB_Use_train_split_predictions,
            "BernoulliNB_classifier Naive Bayes Classifiers for Part 2 (C)",True)
```




```
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.94      0.90       531
           1       0.91      0.80      0.85       390

    accuracy                           0.88       921
   macro avg       0.89      0.87      0.88       921
weighted avg       0.88      0.88      0.88       921
```

Display Accuracy Score for BernoulliNB classifier

```
print("Accuracy for BernoulliNB classifier : ", accuracy_score(y_test_split, BernoulliNB_Use_train_split_predictions))
```

```
Accuracy for BernoulliNB classifier :  0.8805646036916395
```

I choose BernoulliNB classifier because the Target column (Spam) has two classes (1) or not (0)

I noticed that spam column imbalanced and Class imbalance can have an impact on the performance of classifiers and BernoulliNB is a classifier that can handle imbalanced datasets effectively, particularly when the features are binary. This classifier has the ability to capture the patterns associated with the minority class (spam) while disregarding the patterns linked to the majority class (non-spam).But classifiers like MultinomialNB and GaussianNB might be susceptible to the influence of class imbalance.

Part 2→ (D)

Split the first_80_percent into four equal parts (25%)

```
subset_1=first_80_percent[:int(len(first_80_percent)*0.25)]
subset_2=first_80_percent[int(len(first_80_percent)*0.25):int(len(first_80_percent)*0.50)]
subset_3=first_80_percent[int(len(first_80_percent)*0.50):int(len(first_80_percent)*0.75)]
subset_4=first_80_percent[int(len(first_80_percent)*0.75):]
```

Put the four equal parts into List of 4 subset

```
list_of_subset=[subset_1,subset_2,subset_3,subset_4]
```

loop on list_of_subset to access each sub_set

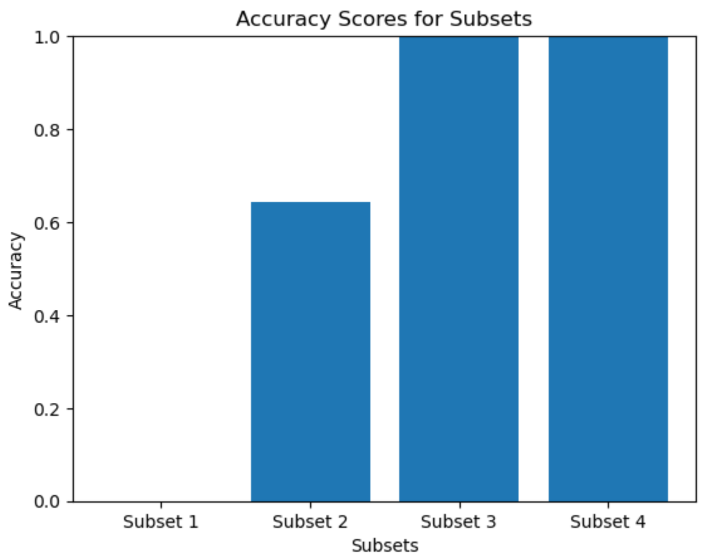
- **Put All columns except the last one in x_train_sub_set**
- **Put the last column in y_train_sub_set**
- **then Run "Naive_Bayes_Models" Function using BernoulliNB classifier for each sub_set**
- **predict the accuracy score by last 20 percent**
- **store the accuracy score for each sub_set in subset_accuracies list**

```
subset_accuracies =[]
for iteration,sub_set in  enumerate(list_of_subset):


    x_train_sub_set = sub_set.drop(sub_set.columns[-1], axis=1)
    y_train_sub_set = sub_set.iloc[:, -1]


    subset_predictions=Naive_Bayes_Models(BernoulliNB(),x_train_sub_set,y_train_sub_set,x_test)



    # Calculate the accuracy score for the subset

    subset_accuracy = accuracy_score(y_test, subset_predictions)
    print(f"Accuracy for subset  {iteration+1} using BernoulliNB classifier : ",subset_accuracy)
    # Store the accuracy score in the List

    subset_accuracies.append(subset_accuracy)
```

```
Accuracy for subset  1 using BernoulliNB classifier :  0.0
Accuracy for subset  2 using BernoulliNB classifier :  0.6438653637350705
Accuracy for subset  3 using BernoulliNB classifier :  1.0
Accuracy for subset  4 using BernoulliNB classifier :  1.0
```

Plot the bar chart to visualize the accuracy scores of the subsets

```
subset_labels = ['Subset 1', 'Subset 2', 'Subset 3', 'Subset 4']
plt.bar(subset_labels, subset_accuracies)
plt.xlabel('Subsets')
plt.ylabel('Accuracy')
plt.title('Accuracy Scores for Subsets')
plt.ylim([0, 1])
plt.show()
```



We got accuracy zero for Subset 1, 64% for Subset 2, 100% for Subset 3, and 100% for Subset 4.

We got accuracy zero for Subset 1 because all the training data in Subset 1 has class (1) only for the 'spam' column, whereas the test data for this Subset has class (0) only for the 'spam' column. Therefore, we obtained zero accuracy because the model was trained on one class and tested on another class for 'spam'

We got an accuracy of 64% for Subset 2 because the training data in Subset 2 consisted of 893 rows for class (1) and 27 rows for class (0) for the 'spam' column. However, the test data for this Subset had only class (0) for the 'spam' column. As a result, we obtained 64% accuracy because the model was trained on insufficient data for class (0) and most of the training focused on class (1) for the 'spam' column, while the test was conducted on class (0) for 'spam'.

We got an 100% accuracy for Subset 3 and Subset 4 because all the training data in both subsets exclusively consisted of class (0) for the 'spam' column. Similarly, the test data for

these subsets only contained class (0) for the 'spam' column. As a result, we obtained 100% accuracy because the model was trained on an adequate amount of data for class (0) in the 'spam' column and tested on the same class (0) for 'spam'.

Show Value Counts and shape for Spam column for each Sub_Set [taining data ,test data]

```
for index,sub_set in  enumerate(list_of_subset):

    print("\t\t\t\t Subset :",index+1,"\n")
    print("\t\t\t **** Training Data ****")

    print("______________________________________________________________________________")
    print(f"shape for subset {index+1} \n ",sub_set.shape)
    print("______________________________________________________________________________")

    print("value counts for Lable column in Training Data\n",sub_set[57].value_counts())
    print("______________________________________________________________________________")

    print("\n\t\t\t **** Test Data ****\n")

    print("value counts for Lable column in Test Data\n",last_20_percent[57].value_counts())
    print("_________________________________________________________________________\n\n")
```

```
                    Subset : 1

              **** Training Data ****
______________________________________________________________
shape for subset 1
  (920, 58)
______________________________________________________________
value counts for Lable column in Training Data
 1    920
Name: 57, dtype: int64
______________________________________________________________

              **** Test Data ****

value counts for Lable column in Test Data
 0    921
Name: 57, dtype: int64
__________________________________________________________


                    Subset : 2

              **** Training Data ****
______________________________________________________________
shape for subset 2
  (920, 58)
______________________________________________________________
value counts for Lable column in Training Data
 1    893
0     27
Name: 57, dtype: int64
______________________________________________________________

              **** Test Data ****

value counts for Lable column in Test Data
 0    921
Name: 57, dtype: int64
__________________________________________________________
```

```
                         Subset : 3

                 **** Training Data ****
______________________________________________________________________________
shape for subset 3
  (920, 58)
______________________________________________________________________________
value counts for Lable column in Training Data
 0    920
Name: 57, dtype: int64
______________________________________________________________________________

                 **** Test Data ****

value counts for Lable column in Test Data
 0    921
Name: 57, dtype: int64
_________________________________________________________________________


                         Subset : 4

                 **** Training Data ****
______________________________________________________________________________
shape for subset 4
  (920, 58)
______________________________________________________________________________
value counts for Lable column in Training Data
 0    920
Name: 57, dtype: int64
______________________________________________________________________________

                 **** Test Data ****

value counts for Lable column in Test Data
 0    921
Name: 57, dtype: int64
_________________________________________________________________________
```