AMERICAN INTERNATIONAL UNIVERSITY-BANGLADESH

DATA WAREHOUSING AND DATA MINING [B]

Fall (2023-24)

Title: Implementing Naïve Bayes Algorithm for Hotel booking reservation Dataset.

Submitted by-

Name	ID
Nobonita Nonde	20-43819-2

Submitted to-

DR. AKINUL ISLAM JONY
Associate Professor, Department Head
Faculty of science & technology

1.Introduction:

The project resolves an extensive data set that includes multiple features and the status of hotel seat reservations. This dataset, which has around 36,000 records, offers insightful information about the variables affecting reservation status.

In this project, we will delve into the dataset, employing various statistical and visual analysis techniques to explore the relationships and patterns within the data. Our goal is to obtain an accurate prediction for the target class value by investigating the correlations among these variables by using Naïve Bayes algorithm.

2.Dataset description:

For the project we have used dataset about hotel reservation which can be found in the link given below:

https://www.kaggle.com/datasets/779de8cc626fb0c43e6083e957a3a045463962637600765c74dd400a268430d3/code?datasetId=2783627&sortBy=voteCount

The dataset has total **attributes 19** and **36275 instances**. Booking class is the target class in the dataset which could be either cancelled or not cancelled. These details (features we will use to predict) are as follows:

Booking ID: Unique identifier of each booking.

No of adults: Number of adults

No of children: Number of Children

No of weekend nights: Number of weekend nights (Saturday or Sunday) the guest stayed or booked to stay at the hotel.

No of week nights: Number of week nights (Monday to Friday) the guest stayed or booked to stay at the hotel.

Type of meal plan: Type of meal plan booked by the customer.

Required car parking space: Does the customer require a car parking space? (0 - No, 1- Yes)

Room type reserved: Type of room reserved by the customer. The values are ciphered (encoded) by INN Hotels.

Lead time: Number of days between the date of booking and the arrival date.

Arrival year: Year of arrival date.

Arrival month: Month of arrival date.

Arrival date: Date of the month.

Market segment type: Market segment designation.

Repeated guest: Is the customer a repeated guest? (0 - No, 1- Yes)

No of previous cancellations: Number of previous bookings that were canceled by the customer prior to the current booking.

No of previous bookings not canceled: Number of previous bookings not canceled by the customer prior to the current booking.

Average price per room: Average price per day of the reservation; prices of the rooms are dynamic. (in euros)

No of special requests: Total number of special requests made by the customer (e.g. high floor, view from the room, etc)

This will let us determine the target variable which is: Booking status: Flag indicating if the booking was canceled or not.

3.Naive Bayes Algorithm:

Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It uses branch of mathematics known as probability theory to find the most likely of the possible classification. It is mainly used in text classification that includes a highdimensional training dataset.

Advantages:

- Simple to Implement. The conditional probabilities are easy to evaluate.
- Very fast – no iterations since the probabilities can be directly computed. So, this technique is useful where speed of training is important.
- If the conditional Independence assumption holds, it could give great results. **Disadvantages:**
- Conditional Independence Assumption does not always hold. In most situations, the feature show some form of dependency.
- When we encounter words in the test data for a particular class that are not present in the training data, we might end up with zero class probabilities.

4.Data Preprocessing:

4.1 Removing Attributes:

In this case, the attributes that don't affect prediction have been eliminated. For example, arrival date, arrival month, and arrival year are three attributes. We've omitted two of these three because it's not necessary for predicting all three. Likewise, we eliminated the Booking Id attribute as well. After eliminating there are total 16 atrributes.

```
df.drop(["Booking_ID","arrival_year", "arrival_date"], inplace=True,
axis="columns") df.head()  df.shape
```

4.2 Missing values and duplicate values:

(df.isnull().sum() / len(df)) used to find percentage of missing values to the total number of instances for each attribute in the Dataset. In this dataset there were no missing values. But there were duplicate instances. To find out the number of duplicate instances: df.duplicated().sum(). As we can see from output, There are 10477 duplicate instances in total.

```
[110]
(df.isnull().sum() / len(df)) * 100

no_of_adults                          0.0
no_of_children                        0.0
no_of_weekend_nights                  0.0
no_of_week_nights                     0.0
type_of_meal_plan                     0.0
required_car_parking_space            0.0
room_type_reserved                    0.0
lead_time                             0.0
arrival_month                         0.0
market_segment_type                   0.0
repeated_guest                        0.0
no_of_previous_cancellations          0.0
no_of_previous_bookings_not_canceled  0.0
avg_price_per_room                    0.0
no_of_special_requests                0.0
booking_status                        0.0
dtype: float64

[111]
df.duplicated().sum()

10477
```

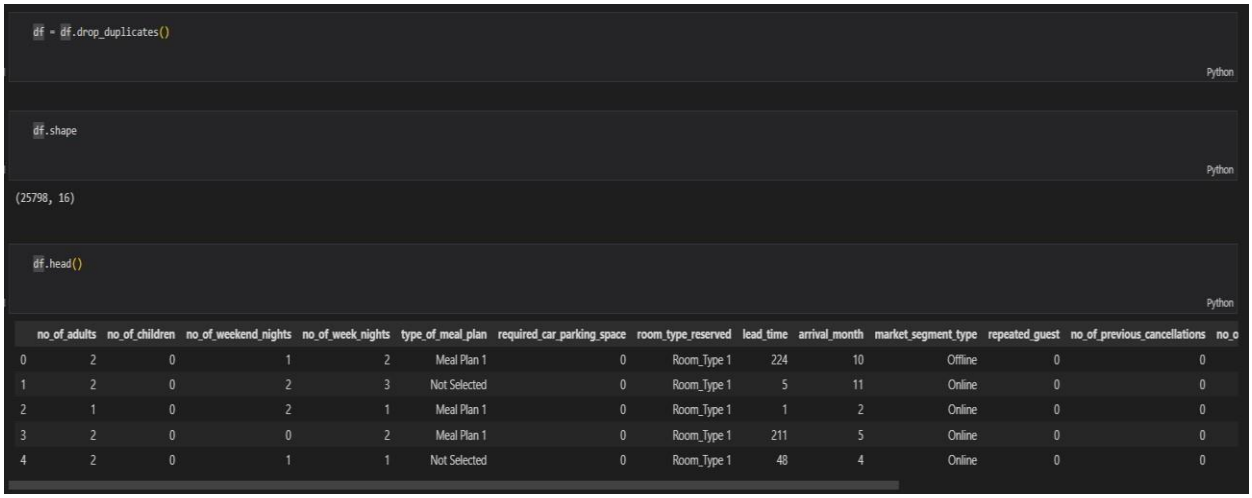
```
df[df.duplicated()]
```

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_reserved	lead_time	arrival_month	market_segment_type	repeated_guest	no_of_previous_cancellations	no_of_previous_bookings_not_canceled	avg_price_per_room	no_of_special_requests	booking_status
20	2	0	2	2	Meal Plan 1	0	Room_Type 1	99	10	Online	0	0	0	65.00	0	Canceled
154	2	0	0	1	Meal Plan 2	0	Room_Type 1	55	4	Offline	0	0	0	104.00	0	Not_Canceled
272	2	0	1	2	Meal Plan 2	0	Room_Type 1	161	3	Online	0	0	0	130.00	0	Canceled
301	1	0	0	2	Meal Plan 1	0	Room_Type 1	188	6	Online	0	0	0	130.00	0	Canceled
319	2	0	0	2	Meal Plan 2	0	Room_Type 1	320	8	Online	0	0	0	115.00	1	Canceled
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
36263	1	0	2	1	Meal Plan 1	0	Room_Type 1	116	2	Online	0	0	0	1.00	0	Not_Canceled
36264	2	0	0	2	Meal Plan 1	0	Room_Type 4	187	7	Online	0	0	0	105.30	0	Canceled
36267	2	0	1	0	Not Selected	0	Room_Type 1	49	7	Online	0	0	0	93.15	0	Canceled
36268	1	0	0	3	Meal Plan 1	0	Room_Type 1	166	11	Offline	0	0	0	110.00	0	Canceled
36274	2	0	1	2	Meal Plan 1	0	Room_Type 1	207	12	Offline	0	0	0	161.67	0	Not_Canceled

10477 rows × 16 columns

4.3 Deleting duplicate instances:

To delete duplicate instance we have used the function drop_duplicates() and then rechecked the shape of dataset. From 36275 instances, there were 25798 instances after duplicate rows were removed.



```
df = df.drop_duplicates()
```

Python

```
df.shape
```

Python

```
(25798, 16)
```

```
df.head()
```

Python

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_reserved	lead_time	arrival_month	market_segment_type	repeated_guest	no_of_previous_cancellations	no_o
0	2	0	1	2	Meal Plan 1	0	Room_Type 1	224	10	Offline	0	0	
1	2	0	2	3	Not Selected	0	Room_Type 1	5	11	Online	0	0	
2	1	0	2	1	Meal Plan 1	0	Room_Type 1	1	2	Online	0	0	
3	2	0	0	2	Meal Plan 1	0	Room_Type 1	211	5	Online	0	0	
4	2	0	1	1	Not Selected	0	Room_Type 1	48	4	Online	0	0	

4.4 Encoding categorical values:

In the dataset some attribute has categorical values. So, before feeding our data to Machine learning algorithms, we have to convert our categorical variables into numerical variables. Type of meal plan, room type reserved, market segment type, booking status; there are category values for these four attributes. In order to transform each attribute's types of values into numeric values, we built an equal number of columns.

```
print(categorical)
df = pd.get_dummies(df, columns=categorical[:-1]) df['booking_status'] =
df['booking_status'].map({"Not_Canceled": 0, "Canceled": 1})
```

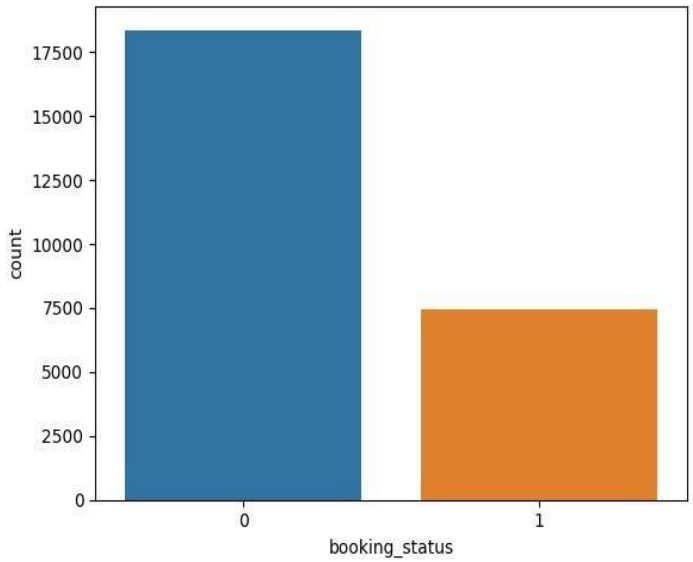
Then, to check if categorical variables were converted to numerical we used “df.dtypes” which returns the data types of each column in the Data frame. It provides information about the type of data stored in each column, such as integers, floats, strings, or categorical data.

```
...    no_of_adults                            int64
       no_of_children                          int64
       no_of_weekend_nights                    int64
       no_of_week_nights                       int64
       required_car_parking_space              int64
       lead_time                               int64
       arrival_month                           int64
       repeated_guest                          int64
       no_of_previous_cancellations            int64
       no_of_previous_bookings_not_canceled    int64
       avg_price_per_room                    float64
       no_of_special_requests                  int64
       booking_status                          int64
       type_of_meal_plan_Meal Plan 1            bool
       type_of_meal_plan_Meal Plan 2            bool
       type_of_meal_plan_Meal Plan 3            bool
       type_of_meal_plan_Not Selected           bool
       room_type_reserved_Room_Type 1           bool
       room_type_reserved_Room_Type 2           bool
       room_type_reserved_Room_Type 3           bool
       room_type_reserved_Room_Type 4           bool
       room_type_reserved_Room_Type 5           bool
       room_type_reserved_Room_Type 6           bool
       room_type_reserved_Room_Type 7           bool
       market_segment_type_Aviation             bool
       market_segment_type_Complementary        bool
       market_segment_type_Corporate            bool
       market_segment_type_Offline              bool
       market_segment_type_Online               bool
       dtype: object
```

4.5 Checking imbalanced dataset and solving:

When a dataset is imbalanced, several issues may arise. Models may exhibit bias toward the majority class, resulting in poor predictions for the minority class. So now we need to check if our target class is balance or not.

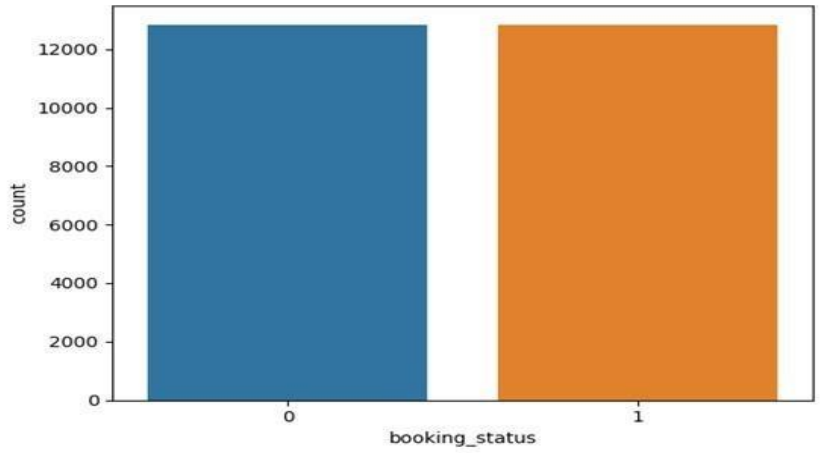
```
df["booking_status"].value_counts() sns.countplot(data=df,
x="booking_status") plt.show()
```



The graph indicates that the numbers for "not cancelled" (0) are significantly higher than those for "cancelled" (1). Thus, there is a possibility that the prediction will be biased toward the majority and focus on the majority. To solve this problem we need to balance the dataset. For balancing dataset here we are

going to use the method “SMOTE”. Synthetic Minority Oversampling Technique (SMOTE) is a statistical technique for increasing the number of cases in your dataset in a balanced way.

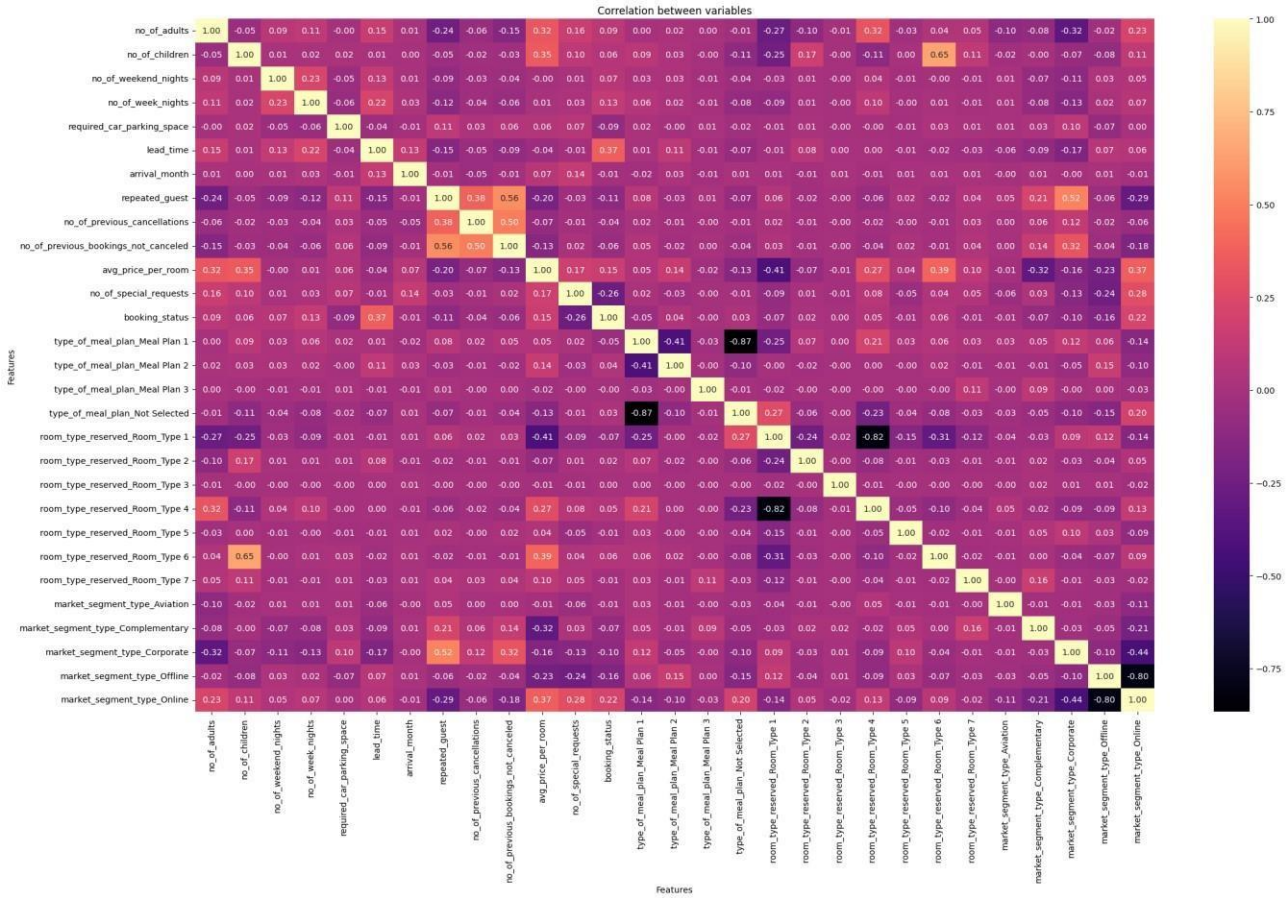
```
smote = SMOTE(sampling_strategy="minority",k_neighbors=5,random_state=42)
x_train_smote, y_train_smote = smote.fit_resample(x_train_pca, y_train)
sns.countplot(data=df, x=y_train_smote) plt.show()
```



4.6 Checking co-relation:

Data correlation is a way to understand the relationship between multiple attributes in the dataset. By using correlation, we can get some insights such as: One or multiple attributes depend on another attribute or a cause for another attribute.

```
fig, ax = plt.subplots(figsize=(25, 15)) sns.heatmap(df.corr(numeric_only=True),
annot=True, cmap="magma", fmt=".2f") plt.gca().set_title("Correlation between
variables") plt.xlabel("Features") plt.ylabel("Features") plt.show()
```



4.7 Train data and test data splitting:

It identifies the target variable (y_train, y_test) and its attributes (x_train, x_test). "booking_status," the target variable, is designated as the data to be predicted. The split is stratified with 30% of the data reserved for testing, while 70% is used for training. The random_state=42 ensures reproducibility by fixing the random seed for the split.

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(     df,
df["booking_status"], test_size=0.3, random_state=42  )
```

4.8 Applying standard scalar:

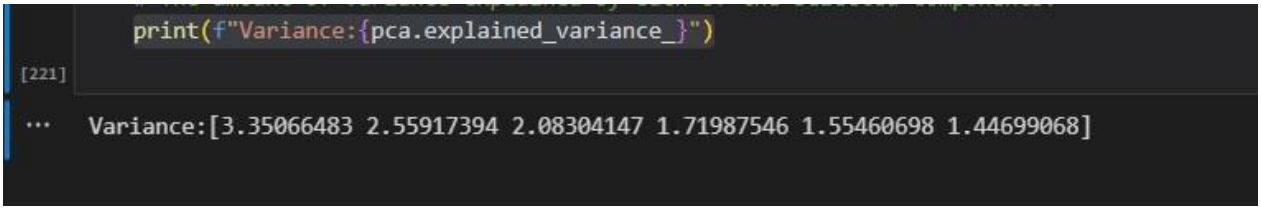
Standard scaler comes into play when the attributes of the dataset measured in different units of measure. Standard scaler removes the mean and scales the data to the unit variance. By guaranteeing a mean of 0 and a standard deviation of 1, standardization improves comparability and makes features more appropriate for machine learning models. This preprocessing stage improves the performance of the model, especially for algorithms that are sensitive to feature scales, and helps the model to train more effectively and with better convergence.

```
sc = StandardScaler() x_train_processed =
sc.fit_transform(x_train)
x_test_processed = sc.transform(x_test)
```

```
from sklearn.preprocessing import StandardScaler
```

4.9 Applying PCA:

```
from sklearn.decomposition import PCA pca=
PCA(n_components=6)
x_train_pca= pca.fit_transform(x_train_processed)
x_test_pca=pca.transform(x_test_processed)
```



5. Creating Naïve bayes:

```
class NaiveBayes: def
__init__(self):
    self.class_probabilities = {}
self.feature_probabilities = {} def fit(self,
X_train, y_train):
self.class_probabilities[0] = np.sum(y_train
== 0) / len(y_train)
self.class_probabilities[1] = np.sum(y_train
== 1) / len(y_train)  for c in [0, 1]:
    X_c = X_train[y_train == c]  mean = np.mean(X_c,
axis=0)  std = np.std(X_c, axis=0)
self.feature_probabilities[c] = {'mean': mean, 'std': std}


  def predict(self, X_test):
predictions = []  for
example in X_test:
likelihood = {}
for c in [0, 1]:  class_prob = np.log(self.class_probabilities[c])  feature_probs =
np.sum(np.log(self.calculate_probability(example[i], self.feature_probabilities[c]['mean'][i],
self.feature_probabilities[c]['std'][i])) for i in range(len(example)))
likelihood[c] = class_prob + feature_probs  predictions.append(max(likelihood, key=likelihood.get))
return predictions
```

```
    def score(self, X_test, y_true):        y_pred =
self.predict(X_test)        correct_predictions = np.sum(y_true
== y_pred)        return
correct_predictions / len(y_true)

    @staticmethod                           def
calculate_probability(x, mean, std):
        exponent = np.exp(-((x - mean) ** 2) / (2 * std ** 2))
return (1 / (np.sqrt(2 * np.pi) * std)) * exponent
```

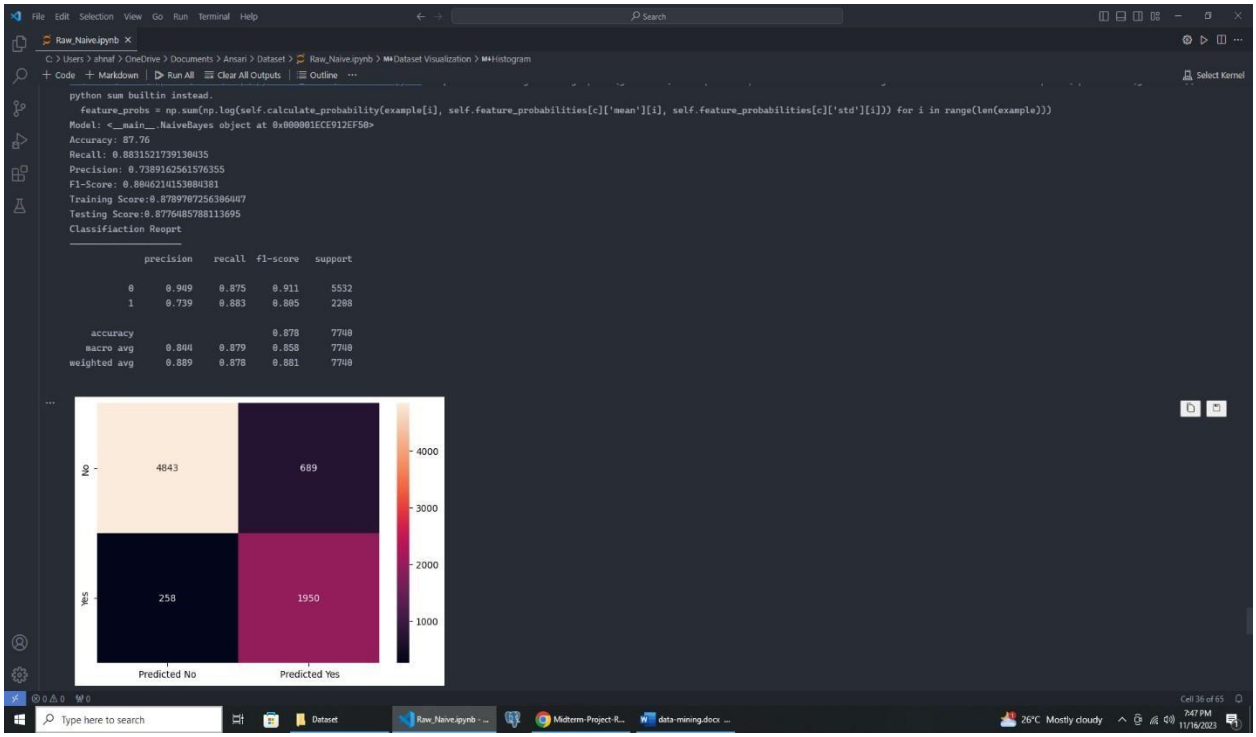
6. Comparing Naïve Bayes Algorithm with Scikit-Learn's existing Naïve Bayes algorithm:

```
from sklearn.naive_bayes import GaussianNB from
sklearn.metrics import * def modelling(x_train,
x_test, y_train, y_test):
    best_result = []
recall_scores = []
precision_scores = []
f1_scores = []     train_score
= []     predict_score = []
classifiers = [
GaussianNB(),
        NaiveBayes()
    ]
    for i in range(len(classifiers)):
model = classifiers[i]        model.fit(x_train,
y_train)        y_pred = model.predict(x_test)

        accuracy = accuracy_score(y_test, y_pred)
best_result.append(accuracy)        recall_scores.append(recall_score(y_test, y_pred))
precision_scores.append(precision_score(y_test, y_pred))
f1_scores.append(f1_score(y_test, y_pred))
train_score.append(model.score(x_train,y_train))
predict_score.append(model.score(x_test,y_test))

        print(f"Model:                          {classifiers[i]}")
print(f"Accuracy: {round(accuracy * 100, 2)}")
        print(f"Recall: {recall_scores[i]}")
print(f"Precision: {precision_scores[i]}")        print(f"F1-Score:
{f1_scores[i]}")        print(f"Training Score:{train_score[i]}")
print(f"Testing
```

```
Score:{predict_score[i]}")        print("Classifiaction Reoprt")        print("----------------------")
print(classification_report(y_test, y_pred, digits=3))        cm=confusion_matrix(y_test, y_pred,
labels=[0, 1])        df_cm = pd.DataFrame(cm, index = [i for i in ["No","Yes"]],                columns = [i
for i in ["Predicted No"," Predicted Yes"]])        plt.figure(figsize = (7,5))        sns.heatmap(df_cm,
annot=True ,fmt='g')        plt.show()        print("------------------------------------------------------------------------
----------------------------------------" )

   model_names = [
"GaussianNB",
"CustomNaiveBayes"
   ]
   result_df = pd.DataFrame(
      {
         "Recall": recall_scores,
         "Precision": precision_scores,
         "F1_Score": f1_scores,
         "Accuracy": best_result,
         "Training_Score": train_score,
         "Testing_Score": predict_score
      },
      index=model_names,
   )
   result_df    =    result_df.sort_values(by="Accuracy",    ascending=False)
return result_df
```

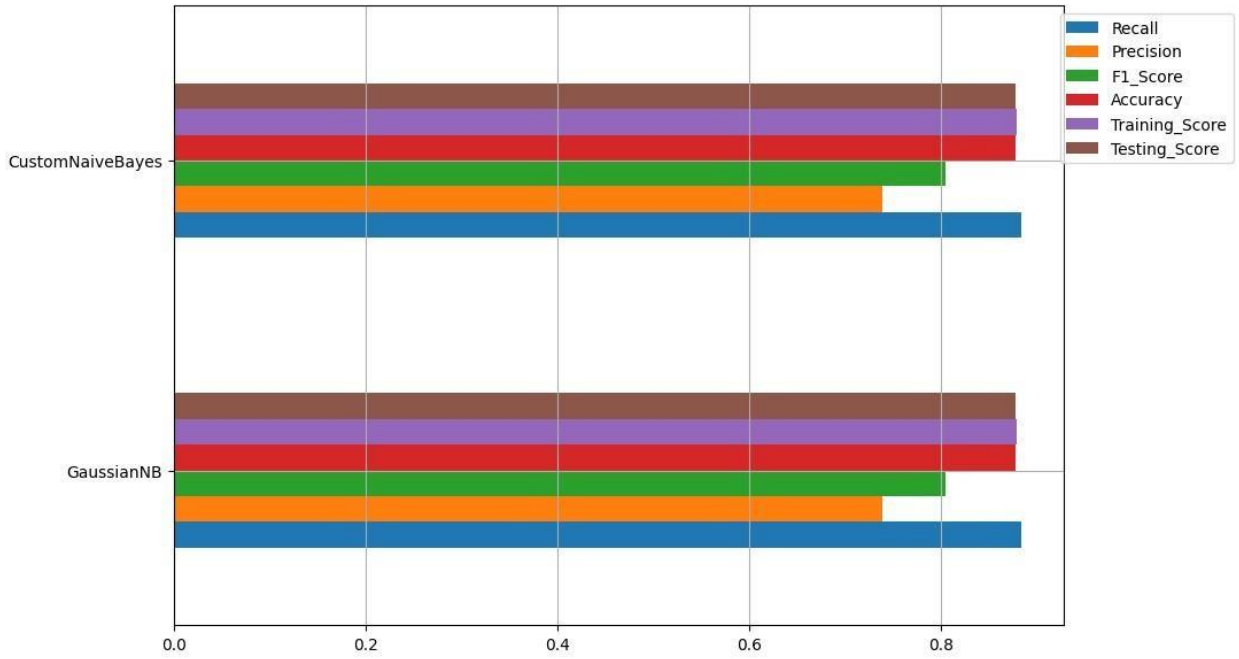
```
Model: GaussianNB()
Accuracy: 87.76
Recall: 0.8831521739130435
Precision: 0.7389162561576355
F1-Score: 0.8046214153084381
Training Score:0.8789707256306447
Testing Score:0.8776485788113695
Classifiaction Reoprt
----------------------
              precision    recall  f1-score   support

           0      0.949     0.875     0.911      5532
           1      0.739     0.883     0.805      2208

    accuracy                          0.878      7740
   macro avg      0.844     0.879     0.858      7740
weighted avg      0.889     0.878     0.881      7740
```

Scikit-Learn's Naïve Bayes:



7. Result:

```
result_df result_df.plot(kind="barh", figsize=(10, 7),
grid=True).legend(bbox_to_anchor=(1.2, 1)) plt.show()
```



After implementing our own naïve bayes algorithm (CustomNaiveBayes) we compared it with the scikit-learn library's naïve bayes (GaussianNB). From the above plot we can see the result is totally same indicating our implemented algorithm (CustomNaiveBayes) is correct. We achieved a very good accuracy of 87.76% for both algorithms. Recall is the Ratio of the correct predictions and the total number of correct items in the dataset which give us score of 88% and precision refers to the number of true positives divided by the total number of positive predictions which is 73% for both algorithms.

8. Conclusion:

The results obtained from implementing the Naïve Bayes algorithm for this project provide valuable insights into the model's performance under different configurations. The project aimed to optimize the accuracy of hotel booking status predictions. The predictions for hotel booking status, as shown for custom naïve bayes, Scikit-Learn's naïve bayes, indicate high percentages of accuracy: 87.76% and 87.76%, respectively. These values suggest that the naïve bayes model performs well in predicting whether a customer will book a hotel room or not. It is seen that the model performs well in recall (88%) and poor while considering precision (73%). Precision is out

of total prediction of a certain class how many of them are right and recall is we know how many true values are there, out of total prediction of a certain class how many of them are correct. F1 score does give us a combined result of precision and recall which is 80%. F1 score assesses the predictive ability of a model by examining its performance on each class individually rather than considering overall performance like accuracy does. We successfully were able to avoid overfitting and underfitting situations by applying some techniques like PCA
(dimensionality reduction), Standard scalar which in essence give us score of training (87%) and testing (87%). In conclusion, this project successfully achieved its objective of developing a robust predictive model using the naïve bayes algorithm.