

CSE 587 DATA INTENSIVE COMPUTING

PROJECT PHASE- 2 REPORT

SAI TEJA SANKARNENI AND NISHANT VARMA INDUKURI – CSE 587

PROBLEM STATEMENT

(A)

The Online Hotel Reservation system has changed the landscape of hotel booking. Customers no longer need to worry about the availability of rooms and services until they reach the front desk. They can check and plan accordingly beforehand through Online Reservation Systems. However, this has also led to a few more challenges for the hotels beyond meeting customer expectations. It has led to a significant increase in cancellations and no-shows.

Although cancellations are common occurrences and could be due to various reasons like a change of plans, weather, scheduling conflicts, etc., they have caused negative implications for the hotels. These hotels started to overbook or under book in some instances, which ultimately led to an impact on the Hotels' revenue.

So, to deal with these kinds of situations, we decided to come up with a model that can help predict whether the customer will cancel the reservation or not. The contents of the table are given below:

- Booking_ID: Unique booking code for each booking
- No_of_adults: Number of adults
- No_of_children: Number of children
- No_of_weekend_nights: Number of Saturday/Sunday nights the guests stayed
- No_of_week_nights: Number of weekday nights the guests stayed
- Type_of_meal_plan: Meal plan chosen by the guest
- Required_car_parking_space: Whether the guest require car parking or not
- Room_type_reserved: Room type selected by the customer
- Lead_time: Number of days between the date of booking and date of arrival
- Arrival_year: Year of the arrival date
- Arrival_month: Month of the arrival date
- Arrival_date: Date of the month of arrival
- Market_segment_type: Designation of market segment
- Repeated_guest: Is the guest a frequent or returning customer
- No_of_previous_cancellations: Number of previous bookings cancelled before arrival date

- No_of_previous_bookings_not_cancelled: Number of previous bookings attended by the customer.
- Avg_price_per_room: Average price of room per day of reservation
- No_of_special_requests: Number of special requests made by the customer
- Booking_status: Booking indication whether it is cancelled or not

(B)

This project has the potential to revolutionize how hotels manage their booking, increase revenue, and enhance customer satisfaction. With the help of this model, we can use historical data on hotel bookings to optimize prices, predict future demand, and suggest choices and services to customers. Through this, hotels could be well-functioning and aim for profits.

DATA SOURCES

The data file is also attached in the folder as Hotel_reservations_data.csv

And has been taken from Kaggle. It consists of 36275 rows and 19 columns.

<https://www.kaggle.com/datasets/ahsan81/hotel-reservations-classification-dataset?datasetId=2783627&sortBy=voteCount>

DATA INITIALIZATION

Firstly, we import all the required libraries and also read the csv file.

```
In [6]: # Importing all required libraries
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
```

```
In [8]: # Reading the HotelReservationsDataset and displaying top 10 rows of the dataset
df = pd.read_csv("DIC.csv")
print("Total Number of Rows:", df.shape[0])
print("Total Number of Columns:", df.shape[1])
df
```

Total Number of Rows: 36275
Total Number of Columns: 19

```
Out[8]:
```

	Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_reserve
0	INN00001	2	0	1	2	Meal Plan 1	0	Room_Type
1	INN00002	2	0	2	3	Not Selected	0	Room_Type
2	INN00003	1	0	2	1	Meal Plan 1	0	Room_Type
3	INN00004	2	0	0	2	Meal Plan 1	0	Room_Type
4	INN00005	2	0	1	1	Not Selected	0	Room_Type
...
36270	INN36271	3	0	2	6	Meal Plan 1	0	Room_Type
36271	INN36272	2	0	1	3	Meal Plan 1	0	Room_Type
36272	INN36273	2	0	2	6	NaN	0	Room_Type
36273	INN36274	2	0	0	3	Not Selected	0	Room_Type
36274	INN36275	2	0	1	2	Meal Plan 1	0	Room_Type

MACHINE LEARNING MODELS

We have chosen the following Machine Learning Models:

➔ LOGISTIC REGRESSION

➔ NAÏVE BAYES

➔ SUPPORT VECTOR MACHINES

➔ DECISION TREE CLASSIFIER

➔ LINEAR REGRESSION

MODEL 1: LOGISTIC REGRESSION

Logistic regression is used to analyse the relationship between a dependent variable and a minimum of one independent variable. It is mainly used for classification of the data where the output variable is a binary outcome.

In this data set the output variable which is the booking outcome is a binary outcome which answers whether the customer has cancelled the booking or not. So, we have decided to apply logistic regression to the model.

At first the data is split into X and Y where X contains all the columns which are the feature variables and Y contains the binary response variable.

```
In [67]: Y_data = df_lr['booking_status']
X_data = df_lr.drop(labels=['booking_status', 'arrival_date'], axis=1)
```

We have decided to put the booking status as the response variable and all other predictors except arrival date as feature variables.

```
In [68]: X_data
```

Out[68]:

	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
0	2	0	2	3	3	0	0	5
1	1	0	2	1	0	0	0	1
2	2	0	0	2	0	0	0	211
3	2	0	1	1	3	0	0	48
4	2	0	1	3	0	0	0	34
...
24330	2	0	0	2	0	0	3	187
24331	2	0	1	3	0	0	0	15
24332	2	0	2	2	0	0	1	8
24333	2	2	0	1	0	0	5	0
24334	2	0	0	3	3	0	0	63

24335 rows x 14 columns

```
In [69]: Y_data
```

Out[69]:

```
0      1
1      0
2      0
3      0
4      1
..
24330  0
24331  1
24332  0
24333  0
24334  0
Name: booking_status, Length: 24335, dtype: category
Categories (2, int64): [0, 1]
```

Now, we split the data into test and train data. (Training Data:80%, Testing Data: 20%)

```
In [70]: # Splitting the data into training and testing (80% of data for training and 20% for testing)
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X_data,Y_data,test_size=0.2)
print(len(X_train))
print(len(X_test))
print(len(y_train))
print(len(y_test))

19468
4867
19468
4867
```

Now, we apply Logistic regression on this training dataset from the Sklearn.linear_model and also importing accuracy and other metrics from sklearn.metrics to analyze our model performance.

```
In [71]: # Training the Logistic regression model on Train Data

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

model_logR = LogisticRegression(max_iter=10000)
model_logR.fit(X_train, y_train)

Out[71]: LogisticRegression(max_iter=10000)
```

Then, we apply the learning from the training data to the test data and compare it with the target variable and compute different metrics achieved through this model.

```
In [49]: # Using the trained model on Test Data and Calculating different Metrics

pred_lr = model_logR.predict(X_test)
print("Model: Logistic Regression")
print("Different Metrics of the Model on Testing Data:")
print(f"Accuracy: {accuracy_score(pred_lr, y_test)*100}%")
print(f"Precision Score: {precision_score(pred_lr, y_test)*100}%")
print(f"f1 Score: {f1_score(pred_lr, y_test)*100}%")
print(f"Recall_Score: {recall_score(pred_lr, y_test)*100}%")

Model: Logistic Regression
Different Metrics of the Model on Testing Data:
Accuracy: 81.52866242038218%
Precision Score: 92.31194690265487%
f1 Score: 88.13201320132012%
Recall_Score: 84.31422076281889%
```

Through these results we have concluded that the accuracy of prediction made by logistic regression is **81.52%**

While the precision score is **92.311%**.

➔ **VISUALIZATIONS:**

CONFUSION MATRIX:

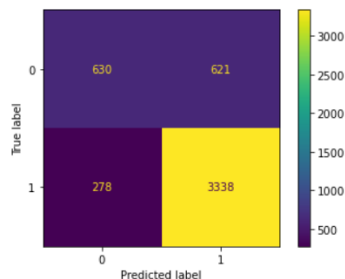
As it is a classification model, we use confusion matrix to determine and visualise the findings. We can deduce that the accuracy is good as the correctly predicted positive values and correctly predicted negative values are high.

In [50]: # Confusion Matrix

```
cm_lr = confusion_matrix(y_test, pred_lr)
print("Confusion Matrix:")
print(cm_lr)
ConfusionMatrixDisplay(cm_lr).plot()
plt.show()
```

Confusion Matrix:

```
[[ 630  621]
 [ 278 3338]]
```



ROC – AUC CURVE:

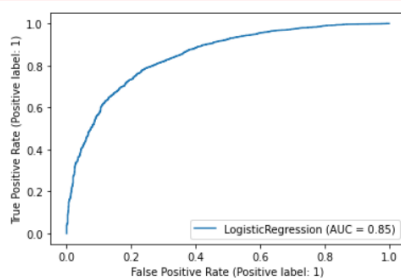
ROC curves plot the true positive rate (TPR) against the false positive rate (FPR) for different threshold values of the classifier while AUC is the area under the ROC curve, which provides a single scalar value that summarizes the performance of the classifier across all possible threshold values.

In [51]: # ROC - AUC Curve

```
print("ROC - AUC Curve:")
plot_roc_curve(model_logR, X_test, y_test)
plt.show()
```

decimal numbers if dtype='numeric'. This behavior is deprecated in 0.24 and will be removed in 1.1 (renaming of 0.26). Please convert your data to numeric values explicitly instead.

```
X = check_array(X, **check_params)
```



PRECISION – RECALL CURVE:

The PR curve is a plot of precision against recall at different threshold values of the classifier.

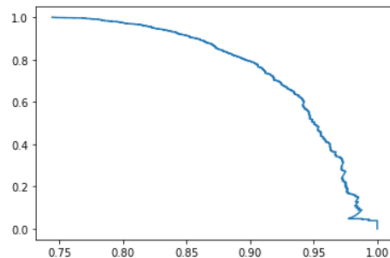
```
In [52]: # Precision - Recall Curve
```

```
probs_logR = model_logR.decision_function(X_test)
p_logR, r_logR, _ = precision_recall_curve(y_test, probs_logR)
print("Precision - Recall Curve:")
plt.plot(p_logR, r_logR)
plt.show()
```

Precision - Recall Curve:

C:\Users\inish\anaconda3\lib\site-packages\sklearn\base.py:566: FutureWarning: Arrays of bytes/strings is being converted to decimal numbers if dtype='numeric'. This behavior is deprecated in 0.24 and will be removed in 1.1 (renaming of 0.26). Please convert your data to numeric values explicitly instead.

```
X = check_array(X, **check_params)
```



MODEL 2: NAIVE BAYES

Naive bayes is a probabilistic classification algorithm which makes predictions by calculating the probability of each class given input features.

In our dataset, we have categorical data as well as independent data which means that all the columns may not be correlated to each other. Hence, we have chosen to perform naïve bayes as it might work well on the chosen dataset One other feature is that our data is balanced which means that all the classes have similar number of samples as others.

We, first import the Gaussian naïve bayes from sklearn and then perform it on the training data.

```
In [53]: # Training the Naive Bayes model on Train Data
```

```
from sklearn.naive_bayes import GaussianNB
```

```
model_nb = GaussianNB()
model_nb.fit(X_train, y_train)
```

C:\Users\inish\anaconda3\lib\site-packages\sklearn\utils\validation.py:964: FutureWarning: Arrays of bytes/strings is being converted to decimal numbers if dtype='numeric'. This behavior is deprecated in 0.24 and will be removed in 1.1 (renaming of 0.26). Please convert your data to numeric values explicitly instead.

```
X = check_array(X, **check_params)
```

```
Out[53]: GaussianNB()
```

The trained model is then used on the test data, the predictions are compared, and different metrics are calculated.

```
In [54]: # Using the trained model on Test Data and Calculating different Metrics
pred_nb = model_nb.predict(X_test)

print("Model: Naive Bayes")
print("Different Metrics of the Model on Testing Data:")
print(f"Accuracy: {accuracy_score(pred_nb, y_test)*100}%")
print(f"Precision Score: {precision_score(pred_nb, y_test)*100}%")
print(f"f1 Score: {f1_score(pred_nb, y_test)*100}%")
print(f"Recall Score: {recall_score(pred_nb, y_test)*100}%")

Model: Naive Bayes
Different Metrics of the Model on Testing Data:
Accuracy: 36.901582083418944%
Precision Score: 16.261061946902654%
f1 Score: 27.690134212385214%
Recall Score: 93.18541996830429%
```

The accuracy we get is 36.90% and the precision score is 16.26%.

This is not high accuracy for Naive Bayes, as it is an extremely simple classifier and we should not expect it to be strong, even more data probably won't help finding good accuracy. Our gaussian estimators are probably already very good, **simply Naive assumptions of the model maybe causing this problem. To overcome this problem, we are using strong models like SVM and Decision trees in our next steps.**

➔ VISUALIZATIONS:

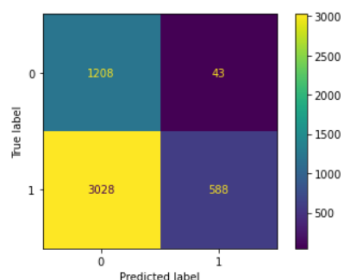
CONFUSION MATRIX:

As it is a classification model, we use confusion matrix to determine and visualise the findings. We can deduce that the accuracy is low as the false positives are high while using naïve bayes classification

```
In [55]: # Confusion Matrix

cm_nb = confusion_matrix(y_test, pred_nb)
print("Confusion Matrix:")
print(cm_nb)
ConfusionMatrixDisplay(cm_nb).plot()
plt.show()

Confusion Matrix:
[[1208  43]
 [3028 588]]
```



ROC – AUC CURVE:

ROC curves plot the true positive rate (TPR) against the false positive rate (FPR) for different threshold values of the classifier while AUC is the area under the ROC

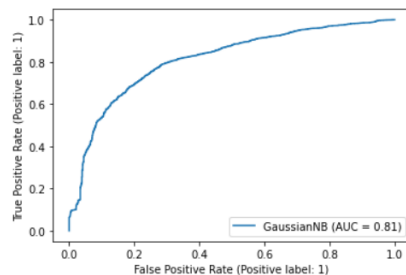
curve, which provides a single scalar value that summarizes the performance of the classifier across all possible threshold values.

In [56]: # ROC - AUC Curve

```
print("ROC - AUC Curve:")
plot_roc_curve(model_nb, X_test, y_test)
plt.show()
```

ROC - AUC Curve:

C:\Users\inish\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:'plot_roc_curve' is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:'sklearn.metrics.RocCurveDisplay.from_predictions' or :meth:'sklearn.metrics.RocCurveDisplay.from_estimator'.
warnings.warn(msg, category=FutureWarning)
C:\Users\inish\anaconda3\lib\site-packages\sklearn\base.py:566: FutureWarning: Arrays of bytes/strings is being converted to decimal numbers if dtype='numeric'. This behavior is deprecated in 0.24 and will be removed in 1.1 (renaming of 0.26). Please convert your data to numeric values explicitly instead.
X = check_array(X, **check_params)



PRECISION – RECALL CURVE:

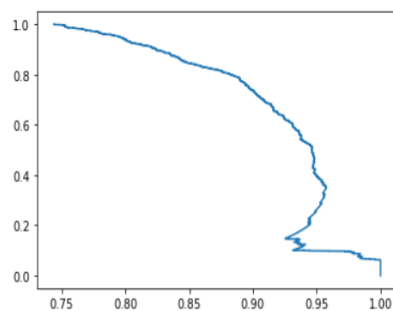
The PR curve is a plot of precision against recall at different threshold values of the classifier.

In [57]: # Precision - Recall Curve

```
probs_nb = model_nb.predict_proba(X_test)
probs_nb = probs_nb[:, 1]
p_nb, r_nb, _ = precision_recall_curve(y_test, probs_nb)
print("Precision - Recall Curve:")
plt.plot(p_nb, r_nb)
plt.show()
```

Precision - Recall Curve:

C:\Users\inish\anaconda3\lib\site-packages\sklearn\base.py:566: FutureWarning: Arrays of bytes/strings is being converted to decimal numbers if dtype='numeric'. This behavior is deprecated in 0.24 and will be removed in 1.1 (renaming of 0.26). Please convert your data to numeric values explicitly instead.
X = check_array(X, **check_params)



MODEL 3: SUPPORT VECTOR MACHINES [SVM]

Support vector machines uses the principle of finding out a hyperplane in N- dimensional space which can distinctly classify the data points. N being the number of features.

We have decided to use SVM for regression as we felt that the data has a large number of features and SVM would be optimal to handle these high dimensional data and to generalize the results in such cases.

SVM is also less sensitive to outliers and a strong model compared to Naïve Bayes Model so can perform better than Naïve Bayes. So, we are choosing Support Vector Machines model for performing Binary classification.

The model is initially trained on the training data.

```
In [58]: # Training the Support Vector Machine model on Train Data
from sklearn.svm import SVC

model_svm = SVC()
model_svm.fit(X_train, y_train)

Out[58]: SVC()
```

Then we use the trained model to perform the SVM classification on the test data and calculate various metrics.

```
In [59]: # Using the trained model on Test Data and Calculating different Metrics

pred_svm = model_svm.predict(X_test)
print("Model: Support Vector Machines(SVM)")
print("Different Metrics of the Model on Testing Data:")
print(f"Accuracy: {accuracy_score(pred_svm, y_test)*100}%")
print(f"Precision Score: {precision_score(pred_svm, y_test)*100}%")
print(f"f1 Score: {f1_score(pred_svm, y_test)*100}%")
print(f"Recall_Score: {recall_score(pred_svm, y_test)*100}%")

Model: Support Vector Machines(SVM)
Different Metrics of the Model on Testing Data:
Accuracy: 78.42613519621943%
Precision Score: 97.42809734513274%
f1 Score: 87.03063241106719%
Recall_Score: 78.63839285714286%
```

Accuracy achieved is 78.42% while the precision achieved is 97.42%.

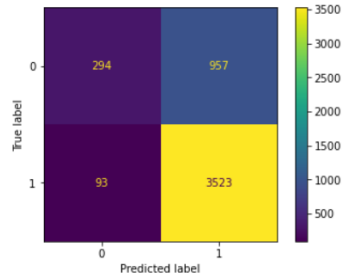
➔ VISUALIZATIONS:

CONFUSION MATRIX:

As it is a classification model, we use confusion matrix to determine and visualise the findings. We can deduce that the accuracy is high as the correctly predicted positive values are comparatively high while using SVM classification.

```
In [60]: # Confusion Matrix
cm_svm = confusion_matrix(y_test, pred_svm)
print("Confusion Matrix:")
print(cm_svm)
ConfusionMatrixDisplay(cm_svm).plot()
plt.show()
```

```
Confusion Matrix:
[[ 294  957]
 [   93 3523]]
```



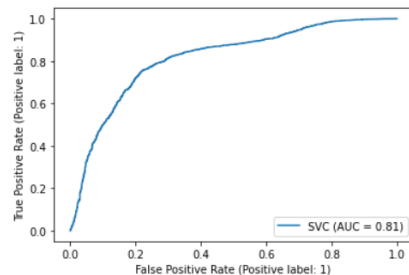
ROC – AUC CURVE:

ROC curves plot the true positive rate (TPR) against the false positive rate (FPR) for different threshold values of the classifier while AUC is the area under the ROC curve, which provides a single scalar value that summarizes the performance of the classifier across all possible threshold values.

```
In [61]: # ROC - AUC Curve
```

```
print("ROC - AUC Curve:")
plot_roc_curve(model_svm, X_test, y_test)
plt.show()
```

Warning: Function `plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: `sklearn.metrics.RocCurveDisplay.from_predictions` or `sklearn.metrics.RocCurveDisplay.from_estimator`.



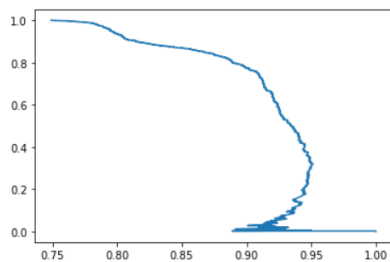
PRECISION – RECALL CURVE:

The PR curve is a plot of precision against recall at different threshold values of the classifier.

```
In [62]: # Precision - Recall Curve

probs_svm = model_svm.decision_function(X_test)
p_svm, r_svm, _ = precision_recall_curve(y_test, probs_svm)
print("Precision - Recall Curve:")
plt.plot(p_svm, r_svm)
plt.show()
```

Precision - Recall Curve:



MODEL 4: DECISION TREE CLASSIFIER

A decision tree is a machine learning algorithm that constructs a tree structure to create classification or regression models. The dataset is divided into smaller subsets while a decision tree is incrementally developed, resulting in a tree with decision nodes and leaf nodes.

Decision tree works well on data that is a combination of both numerical and categorical. Since our data is of the above type, we have decided to use decision tree classifier on our dataset.

We perform Decision tree classifier on the training data.

```
In [63]: # Training the Random Forest model on Train Data

from sklearn.tree import DecisionTreeClassifier

model_dt = DecisionTreeClassifier(max_depth=3, random_state=42)
model_dt.fit(X_train, y_train)

Out[63]: DecisionTreeClassifier(max_depth=3, random_state=42)
```

Then we use the trained model to perform the classification on the test data and calculated various metrics

```
In [64]: # Using the trained model on Test Data and Calculating different Metrics
```

```
pred_dt = model_dt.predict(X_test)
print("Model: Decision Tree Classifier")
print("Different Metrics of the Model on Testing Data:")
print(f"Accuracy: {accuracy_score(pred_dt, y_test)*100}%")
print(f"Precision Score: {precision_score(pred_dt, y_test)*100}%")
print(f"f1 Score: {f1_score(pred_dt, y_test)*100}%")
print(f"Recall Score: {recall_score(pred_dt, y_test)*100}%")
```

```
Model: Decision Tree Classifier
Different Metrics of the Model on Testing Data:
Accuracy: 79.55619478117937%
Precision Score: 82.8816371681416%
f1 Score: 85.76334239519245%
Recall Score: 88.8526534242514%
```

Accuracy achieved is 79.55% while the precision achieved is 82.88%.

➔ VISUALIZATIONS:

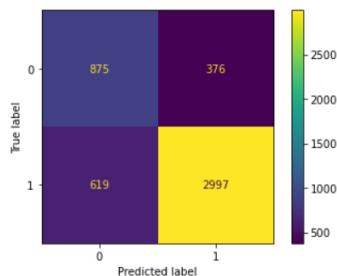
CONFUSION MATRIX:

As it is a classification model, we use confusion matrix to determine and visualise the findings. We can deduce that the accuracy is high as the correctly predicted positive values and correctly predicted negative values are comparatively high while using Decision tree classifier.

```
In [65]: # Confusion Matrix
```

```
cm_dt = confusion_matrix(y_test, pred_dt)
print("Confusion Matrix:")
print(cm_dt)
ConfusionMatrixDisplay(cm_dt).plot()
plt.show()
```

```
Confusion Matrix:
[[ 875  376]
 [ 619 2997]]
```



ROC – AUC CURVE:

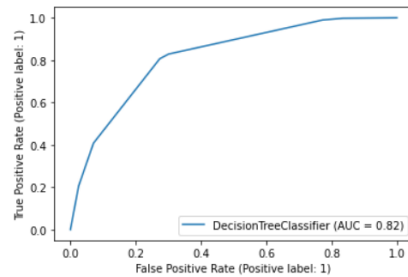
ROC curves plot the true positive rate (TPR) against the false positive rate (FPR) for different threshold values of the classifier while AUC is the area under the ROC curve, which provides a single scalar value that summarizes the performance of the classifier across all possible threshold values.

In [66]: `# ROC - AUC Curve`

```
print("ROC - AUC Curve:")
plot_roc_curve(model_dt, X_test, y_test)
plt.show()
```

ROC - AUC Curve:

C:\Users\inish\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metrics.RocCurveDisplay.from_predictions` or :meth:`sklearn.metrics.RocCurveDisplay.from_estimator`.
warnings.warn(msg, category=FutureWarning)



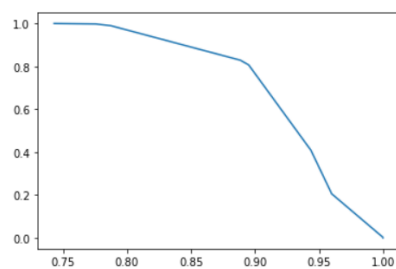
PRECISION – RECALL CURVE:

The PR curve is a plot of precision against recall at different threshold values of the classifier.

In [67]: `# Precision - Recall Curve`

```
probs_dt = model_dt.predict_proba(X_test)
probs_dt = probs_dt[:, 1]
p_dt, r_dt, _ = precision_recall_curve(y_test, probs_dt)
print("Precision - Recall Curve:")
plt.plot(p_dt, r_dt)
plt.show()
```

Precision - Recall Curve:



MODEL 5: LINEAR REGRESSION

Linear regression is used to find the relationship between a dependent variable and more than one independent variable used to predict a continuous outcome.

We have used linear regression to predict the prices of the hotels that may be shown at a particular date. As our problem statement states that we even try to find out the optimal prices that the hotel may consider charging as the day of arrival of guest and the booking date are intersecting. So, for this part we chose linear regression to predict the Average price per room.

Creating a copy of the dataset.

```
In [68]: df_linearR = df.copy()
df_linearR
```

Out[68]:

	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
0	2	0	2	3	3	0	0	5
1	1	0	2	1	0	0	0	1
2	2	0	0	2	0	0	0	211
3	2	0	1	1	3	0	0	48
4	2	0	1	3	0	0	0	34
...
24330	2	0	0	2	0	0	3	187
24331	2	0	1	3	0	0	0	15
24332	2	0	2	2	0	0	1	8
24333	2	2	0	1	0	0	5	0
24334	2	0	0	3	3	0	0	63

24335 rows × 9 columns

Looking for various datatypes in the dataset.

```
In [69]: df_linearR.dtypes
```

Out[69]:

no_of_adults	int64
no_of_children	int64
no_of_weekend_nights	int64
no_of_week_nights	int64
type_of_meal_plan	category
required_car_parking_space	category
room_type_reserved	category
lead_time	int64
arrival_date	datetime64[ns]
market_segment_type	category
repeated_guest	category
no_of_previous_cancellations	int64
no_of_previous_bookings_not_canceled	int64
avg_price_per_room	float64
no_of_special_requests	int64
booking_status	category
dtype:	object

Describing the dataset.

```
In [70]: df_linearR.describe()
```

Out[70]:

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	lead_time	no_of_previous_cancellations	no_of_previous_bookings_not_canceled
count	24335.000000	24335.000000	24335.000000	24335.000000	24335.000000	24335.000000	24335.000000
mean	1.891720	0.139306	0.838463	2.138155	57.714732	0.029587	0.218
std	0.529389	0.446890	0.832628	1.274219	55.249023	0.418931	2.053
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	2.000000	0.000000	0.000000	1.000000	11.000000	0.000000	0.000
50%	2.000000	0.000000	1.000000	2.000000	40.000000	0.000000	0.000
75%	2.000000	0.000000	2.000000	3.000000	90.000000	0.000000	0.000
max	4.000000	2.000000	4.000000	5.000000	219.000000	13.000000	58.000

We then normalize the data set as it would be optimal to do that as linear regression assumes that the dataset is in normality.

```
In [71]: # Normalizing the Numerical columns for better performance

noncat_cols_lst = ["no_of_adults", "no_of_children", "no_of_weekend_nights", "no_of_week_nights", "lead_time",
                   "no_of_previous_cancellations", "no_of_previous_bookings_not_canceled", "avg_price_per_room",
                   "no_of_special_requests"]

for c in noncat_cols_lst:
    print(f"{c}")
    print(f"Max: {df_linearR[c].loc[df_linearR[c].idxmax()}], Min: {df_linearR[c].loc[df_linearR[c].idxmin()]}")
    df_linearR[c] = df_linearR[c]/df_linearR[c].abs().max()
df_linearR
```

no_of_adults
Max: 4, Min: 0
no_of_children
Max: 2, Min: 0
no_of_weekend_nights
Max: 4, Min: 0
no_of_week_nights
Max: 5, Min: 0
lead_time
Max: 219, Min: 0
no_of_previous_cancellations
Max: 13, Min: 0
no_of_previous_bookings_not_canceled
Max: 58, Min: 0
avg_price_per_room
Max: 540.0, Min: 0.0
no_of_special_requests
Max: 5, Min: 0

We then split the data into test and training dataset.

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

Y_data = df_linearR['avg_price_per_room']
X_data = df_linearR.drop(labels=['avg_price_per_room', 'arrival_date'], axis=1)

# Splitting the data into training and testing (80% of data for training and 20% for testing)
X_train, X_test, y_train, y_test = train_test_split(X_data, Y_data, test_size=0.2)
print(len(X_train))
print(len(X_test))
print(len(y_train))
print(len(y_test))

19468
4867
19468
4867
```


We then perform linear regression on the training data

```
In [73]: import numpy as np
from sklearn.linear_model import LinearRegression

model_linearR = LinearRegression()
model_linearR.fit(X_train, y_train)

C:\Users\inish\anaconda3\lib\site-packages\sklearn\utils\validation.py:964: FutureWarning: Arrays of bytes/strings is being converted to decimal numbers if dtype='numeric'. This behavior is deprecated in 0.24 and will be removed in 1.1 (renaming of 0.26). Please convert your data to numeric values explicitly instead.
  X = check_array(

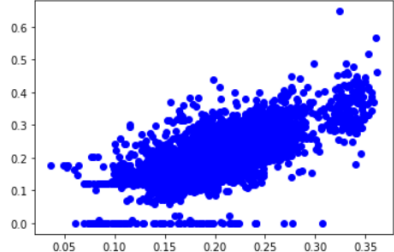
Out[73]: LinearRegression()
```

Plotting a scatterplot of the predicted values and actual response variable values.

```
In [74]: print("Scattel Plot:")
pred_linearR = model_linearR.predict(X_test)
plt.scatter(pred_linearR, y_test, color='b')
plt.show()

Scattel Plot:

C:\Users\inish\anaconda3\lib\site-packages\sklearn\base.py:566: FutureWarning: Arrays of bytes/strings is being converted to decimal numbers if dtype='numeric'. This behavior is deprecated in 0.24 and will be removed in 1.1 (renaming of 0.26). Please convert your data to numeric values explicitly instead.
  X = check_array(X, **check_params)
```



The trained model is then used to perform the linear regression on the test data and print out various metrics.

```
In [75]: # Using the trained model on Test Data and Calculating the Metrics
from sklearn.metrics import mean_squared_error, mean_absolute_error

print("Model: Linear Regression")
print("Different Metrics of the Model on Testing Data:")
mse = mean_squared_error(y_test, pred_linearR)
print(f"Mean Squared Error: {mse}")
rmse = mean_squared_error(y_test, pred_linearR, squared=False)
print(f"Root Mean Squared Error: {rmse}")
mae = mean_absolute_error(y_test, pred_linearR)
print(f"Mean Absolute Error: {mae}")
r2_score = model_linearR.score(X_test, y_test)
print(f"R2 Score: {r2_score}")

Model: Linear Regression
Different Metrics of the Model on Testing Data:
Mean Squared Error: 0.0025767555479280857
Root Mean Squared Error: 0.05076175280590777
Mean Absolute Error: 0.037410217666010756
R2 Score: 0.4609596122928279
```

The mean squared error is 0.0025 and the root mean squared error is 0.050. If we derive the Mean absolute error, it is also returned as 0.037.

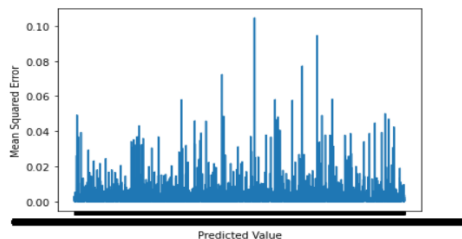
```
In [76]: print("Plot: Predicted Values VS Mean Squared Errors")

err_lst = []
y_test_lst = list(y_test)
pred_lst = list(pred_linearR)

for i in range(0, len(y_test)):
    e = (y_test_lst[i] - pred_lst[i])**2
    err_lst.append(e)

plt.plot(err_lst)
plt.xticks(ticks=[i for i in range(len(err_lst))], labels=pred_lst)
plt.xlabel('Predicted Value')
plt.ylabel('Mean Squared Error')
plt.show()
```

Plot: Predicted Values VS Mean Squared Errors



SUMMARY

For Classification we have performed 4 models that are: Logistic regression, Naïve bayes, Support Vector Machines and Decision Tree. Out of them Logistic Regression and Naïve Bayes were taught in class. Out of all the four machine learning algorithms Logistic Regression performed the best with an Accuracy of 81.52%.

For predicting the Average price per room based on all the other features we have used Linear Regression Model which was taught in class. We obtained a very low Mean Squared error value of 0.0025 indicating it performed great when we used it to make predictions on the testing data.

REFERENCES

Logistic Regression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

Naïve Bayes: <https://heartbeat.comet.ml/naive-bayes-classifier-in-python-using-scikit-learn-13c4deb83bcf>

Support Vector Machines (SVM): <https://blog.paperspace.com/implementing-support-vector-machine-in-python-using-sklearn/>

Decision Trees: <https://towardsdatascience.com/introduction-to-decision-tree-classifiers-from-scikit-learn-32cd5d23f4d>

Linear Regression: <https://realpython.com/linear-regression-in-python/>

Visualizations: https://scikit-learn.org/stable/auto_examples/miscellaneous/plot_display_object_visualization.html#sphx-glr-auto-examples-miscellaneous-plot-display-object-visualization-py