

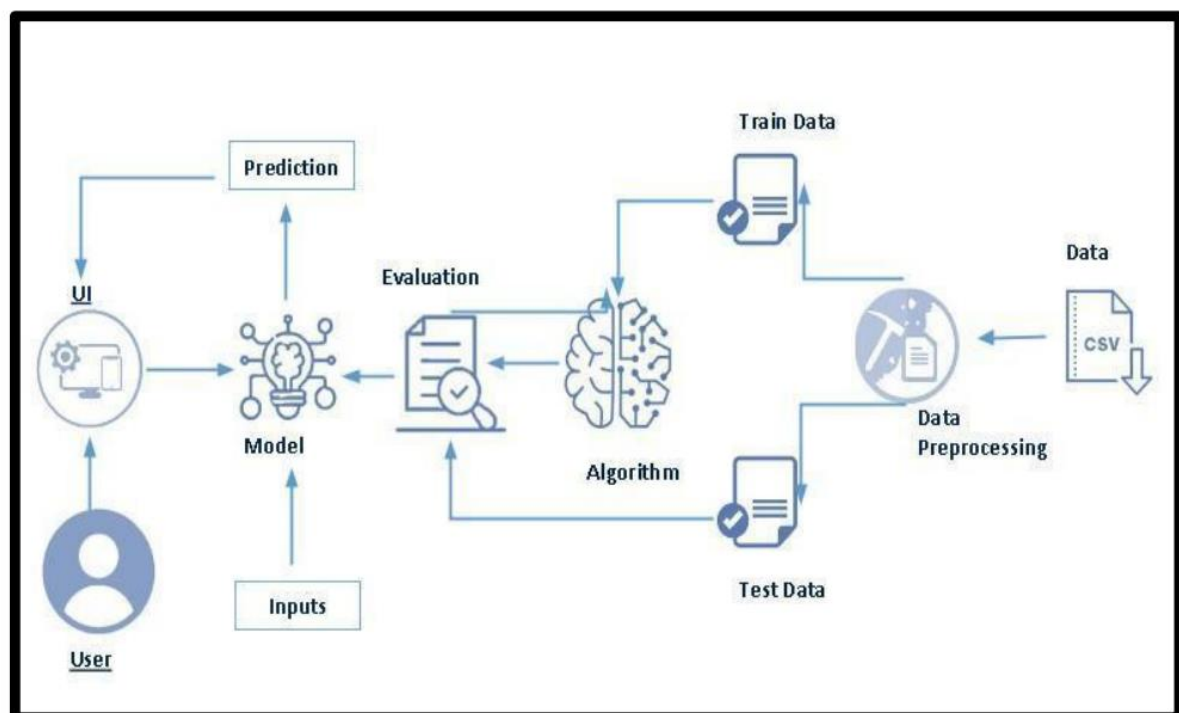
# Reservation Cancellation Prediction

In today's world, people take high amounts of responsibilities. These high responsibilities lead to stress and wear out. To relax everybody thinks of going on a holiday or vacation. Reservations are made for hotels at some destination. It is not necessary that once the reservation is made, the person who made the reservation will show up at the day of reservation. They might have to cancel it due to unforeseen circumstances. These reservation cancellations are a huge problem faced by the hotel owners or managers as they are at a loss because they cannot take another booking on the same day.

Catering to all the problems stated above, we have developed a model which can predict whether the made reservation will be cancelled. This model can be used by hotel owners and managers for predicting the booking cancellations. This model does not ask for personalised data such as name, age, gender, religion, address, etc. You can use this web application anytime you receive a reservation, and the model will the output for the reservation cancellation.

The model is right 92 out of 100 times on an average. This model should be used for reducing losses by the hotel owners and should not be the only factor for making important decisions.

## Technical Architecture:



## Project Flow:

- User is shown the Home page. The user will browse through Home page and click on the Predict button.
- After clicking the Predict button the user will be directed to the Details page where the user will input the details of the reservation and click on the Predict button.
- User will be redirected to the Results page. The model will analyse the inputs given by the user and showcase the prediction of the by the model whether the reservation will be cancelled.

To accomplish this, we have to complete all the activities listed below:

- Define problem / Problem understanding
  - Specify the business problem
  - Business Requirements
  - Literature Survey
  - Social or Business Impact
- Data Collection
  - Collect the dataset
  - Data Preparation
- Exploratory Data Analysis
  - Descriptive statistical
  - Checking Unique Values
  - Visual Analysis
- Data Pre-processing
  - Splitting features and target
  - Balancing the data
  - Splitting into training and validation data
- Model Building
  - Creating a function for evaluation
  - Training and testing the Models using multiple algorithms
- Performance Testing & Hyperparameter Tuning
  - Testing model with multiple evaluation metrics
  - Comparing model accuracy for different number of features.
  - Comparing model accuracy before & after applying hyperparameter tuning
  - Checking the Predictions
- Model Deployment
  - Save the best model
  - Integrate with Web Framework

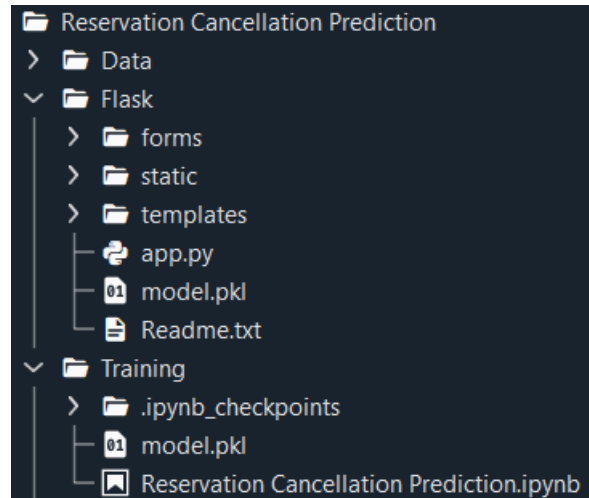
## Prior Knowledge:

You must have the prior knowledge of the following topics to complete this project.

- ML Concepts:
  - Supervised learning: <https://www.javatpoint.com/supervised-machine-learning>
  - K Nearest Neighbours: <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning>
  - SVM: <https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm>
  - Decision tree: <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>
  - Random forest: <https://www.javatpoint.com/machine-learning-random-forest-algorithm>
  - Naïve Bayes: [https://scikit-learn.org/stable/modules/naive\\_bayes.html](https://scikit-learn.org/stable/modules/naive_bayes.html)
  - Logistic Regression: [https://scikitlearn.org/stable/modules/generated/sklearn.linear\\_model.LogisticRegression.html](https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)
  - Gradient Boost: <https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html>
  - Evaluation metrics: <https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/>
- Flask Basics: [https://www.youtube.com/watch?v=lj4I\\_CvBnt0](https://www.youtube.com/watch?v=lj4I_CvBnt0)

## Project Structure:

Create project folder which contains files as shown below:



- The data obtained is in two csv files, one for training and another for testing.
- We are building a Flask application which will require the html files to be stored in the templates folder.
- The css files should be stored in the static folder.
- App.py file is used for routing purposes using scripting.
- Model.pkl is the saved model. This will further be used in the Flask integration.
- Training folder contains a model training file.

## **Milestone 1: Define Problem/ Problem Understanding**

### **Activity 1: Specify the Business Problem**

Reservation Cancellation Prediction is a Binary Classification problem. Reservation Cancellation Prediction involves identifying reservations which can be cancelled based on number of factors like number of adults, number of children, type of booking, parking allotment, etc. Predictive analytics and machine learning techniques can be used to analyse large amounts of data to identify patterns of reservation cancellation. Reservation Cancellation Prediction involves use of various algorithms to analyse large datasets and identify patterns and tell apart the reservations that are going to be cancelled.

### **Activity 2: Business Requirements**

A reservation cancellation prediction problem can have a variety of business requirements, depending on the specific goals and objectives of the project. Some potential requirements may include:

- **Accurate and reliable information:**  
The case of reservation cancellation prediction is critical and no false information can be tolerated since large amount of capital is involved. Also the right inputs should be mapped to the right outputs in the training data provided.
- **Trust:**  
Trust needs to be developed for the users to use the model. It is difficult to create trust among hotel users as they are in the same field for many years.
- **Compliance:**  
The model should be fit with all the relevant laws and regulations of the reservation and booking websites and with the tourism units of states.
- **User friendly interface:**  
The interface should be easy to use and understand by the user. The model should not ask inputs for which the user does not have answers.

### **Activity 3: Literature Survey**

A literature survey for reservation cancellation prediction project would involve researching and reviewing existing studies, articles, and other publications on the topic of reservations. The survey would aim to gather information on current classification systems, their strengths and weaknesses, and any gaps in knowledge that the project could address. The literature survey would also look at the methods and techniques used in previous reservation cancellation prediction projects, and any relevant data or findings that could inform the design and implementation of the current project.

## **Activity 4: Social or Business Impact.**

### **Social Impact:**

This system will keep help identify users who frequently cancel their booking and their IDs can be blocked by the reservation websites.

### **Business Impact:**

The model will help the hotel managers and owners to check the availability of their hotel and predict the available rooms for any particular day.

## **Milestone 2: Data Collection and Preparation:**

Machine Learning depends heavily on data. It is the most crucial part aspect that makes algorithm training possible. So, this section guides on how to download dataset.

### **Activity 1: Collect the dataset**

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: <https://www.kaggle.com/datasets/gauravduttakiit/reservation-cancellation-prediction>

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

**Note:** There are several techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

### **Activity 1.1: Importing the Libraries**

Import the necessary libraries as shown in the image. Some of them are optional and can be skipped according to your usage.

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score

from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
```

### **Activity 1.2: Read the Dataset**

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read\_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

```
In [2]: train = pd.read_csv("C:/Users/hp/SB/Reservation Cancellation Prediction/Data/train__dataset.csv")
```

```
In [3]: train.head()
```

```
Out[3]:
```

	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	arri
0	2	0	1	4	0	0	0	118	
1	2	1	0	2	0	0	0	17	
2	1	0	1	5	0	0	0	349	
3	1	0	2	4	0	0	0	69	
4	2	0	0	4	1	0	0	11	

```
In [4]: train.shape
```

```
Out[4]: (18137, 18)
```

```
In [5]: test = pd.read_csv("C:/Users/hp/SB/Reservation Cancellation Prediction/Data/test__dataset.csv")
```

```
In [6]: test.head()
```

```
Out[6]:
```

	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	arri
0	2	0	1	2	0	1	0	65	
1	2	0	0	2	0	0	0	322	
2	1	0	2	3	0	0	0	115	
3	2	0	2	0	2	0	0	386	
4	2	0	1	4	0	0	1	51	

As we have two datasets, one for training and other for testing we will import both the csv files.

## Activity 2: Data Preparation

As we have understood how the data is, let us pre-process the collected data.

The Machine Learning model cannot be trained on the imported data directly. The dataset might have randomness, we might have to clean the dataset and bring it in the right form. This activity involves the following steps:

- Removing Redundant Columns
- Handling Missing Values

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

### Activity 2.1: Removing Redundant Columns

There are no redundant columns present in the data. All the columns look useful. However, if we wish to drop any columns based on their importance for machine learning models we can drop them later.

We can skip this step.



## Activity 2.2: Handling Missing Values

```
In [8]: train.isnull().any()
```

```
Out[8]: no_of_adults           False
        no_of_children        False
        no_of_weekend_nights   False
        no_of_week_nights     False
        type_of_meal_plan      False
        required_car_parking_space False
        room_type_reserved     False
        lead_time              False
        arrival_year           False
        arrival_month          False
        arrival_date           False
        market_segment_type    False
        repeated_guest         False
        no_of_previous_cancellations False
        no_of_previous_bookings_not_canceled False
        avg_price_per_room     False
        no_of_special_requests False
        booking_status         False
        dtype: bool
```

There are no missing values in the dataset. That is why we can skip this step.

## Milestone 3: Exploratory Data Analysis

### Activity 1: Descriptive Statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top, and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
In [10]: train.describe()
```

Out[10]:

	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
count	18137.000000	18137.000000	18137.000000	18137.000000	18137.000000	18137.000000	18137.000000	18137.000000
mean	1.846777	0.107515	0.811104	2.208965	0.318465	0.031648	0.336770	85.3774
std	0.516020	0.408901	0.873470	1.426365	0.629140	0.175066	0.772865	86.6111
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	17.000000
50%	2.000000	0.000000	1.000000	2.000000	0.000000	0.000000	0.000000	57.000000
75%	2.000000	0.000000	2.000000	3.000000	0.000000	0.000000	0.000000	127.000000
max	4.000000	9.000000	7.000000	17.000000	3.000000	1.000000	6.000000	443.000000

### Activity 2: Checking Unique values in each column

```
In [11]: train.columns
```

Out[11]: Index(['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\_year', 'arrival\_month',  
'arrival\_date', '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'],  
dtype='object')

```
In [12]: for i in train.columns:  
         print(i)  
         print(train[i].value_counts())
```

```
no_of_adults  
2    13104  
1     3809  
3     1150  
0         67  
4          7  
Name: no_of_adults, dtype: int64  
no_of_children  
0    16767  
1     814  
2     544  
3       10  
9         2  
Name: no_of_children, dtype: int64  
no_of_weekend_nights  
0    8444  
1    4985  
2    4535  
3     224
```

Here we have seen the unique values for all the columns. It has come to our observation that there are some categorical columns, and some columns have continuous values.

## Activity 3: Visual Analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

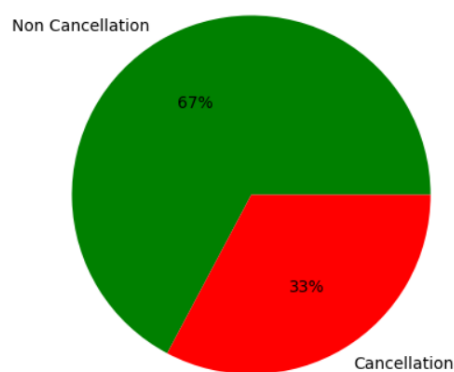
### Activity 3.1: Univariate Analysis:

In simple words, univariate analysis is understanding the data with a single feature. We have displayed three different types of graphs and plots.

For simple visualizations we can use the matplotlib.pyplot library.

```
In [16]: plt.figure(figsize = (5,5))
a = train['booking_status'].value_counts()
plt.pie(x = a, data = train, labels= ['Non Cancellation','Cancellation'], autopct='%0f%%',colors = 'gr')
plt.title("Pie chart showing the distribution of Booking Status into number of Cancellation/Non Cancellation ")
Out[16]: Text(0.5, 1.0, 'Pie chart showing the distribution of Booking Status into number of Cancellation/Non Cancellation ')
```

Pie chart showing the distribution of Booking Status into number of Cancellation/Non Cancellation



Here the plt.figure() command is used to determine the size of the plot.

The pie plot shows the different values distribution in the Booking Status column which is our target variable. It shows that there are 67% observations where the booking status has value 0 and there are 33% observations where the booking status has value 1.

```
In [17]: train.hist(bins=5, figsize=(18,18))
```

```
Out[17]: array([[<Axes: title={'center': 'no_of_adults'}>,
<Axes: title={'center': 'no_of_children'}>,
<Axes: title={'center': 'no_of_weekend_nights'}>,
<Axes: title={'center': 'no_of_week_nights'}>],
[<Axes: title={'center': 'type_of_meal_plan'}>,
<Axes: title={'center': 'required_car_parking_space'}>,
<Axes: title={'center': 'room_type_reserved'}>,
<Axes: title={'center': 'lead_time'}>],
[<Axes: title={'center': 'arrival_year'}>,
<Axes: title={'center': 'arrival_month'}>,
<Axes: title={'center': 'arrival_date'}>,
<Axes: title={'center': 'market_segment_type'}>],
[<Axes: title={'center': 'repeated_guest'}>,
<Axes: title={'center': 'no_of_previous_cancellations'}>,
<Axes: title={'center': 'no_of_previous_bookings_not_cancelled'}>,
<Axes: title={'center': 'avg_price_per_room'}>],
[<Axes: title={'center': 'no_of_special_requests'}>,
<Axes: title={'center': 'booking_status'}>],
dtype=object])
```



Here we have plotted histograms for all the columns which show us the distribution of the values in each column. Some columns clearly show that there are only binary values in them. Eg. Arrival Year, No\_of\_previous\_bookings\_not\_cancelled, etc.

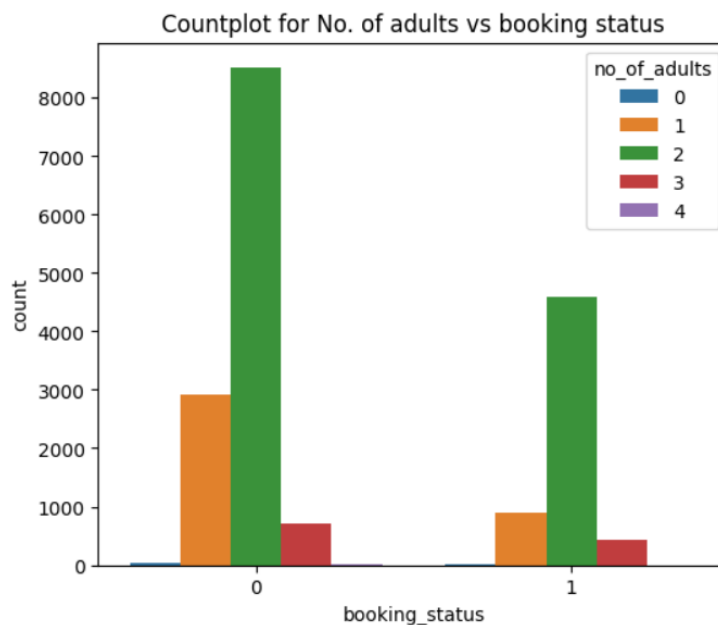
We can also see that the arrival\_date column is continuous.

### Activity 3.2: Bivariate Analysis:

To find the relation between two features we use bivariate analysis. Here we are visualising the relationship between booking status and the no\_of\_adults column. We have observed that the distribution of number of adults is similar in both cases of cancellation and non cancellation of reservation.

```
In [18]: def countplot_of_2(x,hue,title=None,figsize=(6,5)):
          plt.figure(figsize=figsize)
          sns.countplot(data=train[[x,hue]],x=x,hue=hue)
          plt.title(title)
          plt.show()
```

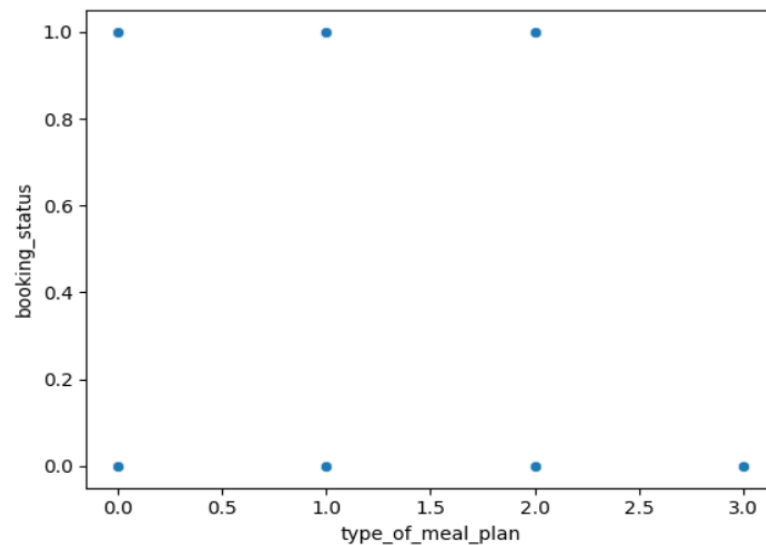
```
In [20]: countplot_of_2('booking_status','no_of_adults', 'Countplot for No. of adults vs booking status')
```



Here we create a function by the name of countplot\_of\_2 where we can give different inputs. Many more bivariate visualizations can easily be created with the help of this function. Here hue represents the colour of the categorical variable which is no\_of\_adults in this case.

We have also plotted a scatterplot of type\_of\_meal\_plan column with the booking status column.

```
In [21]: sns.scatterplot(data = train, x = "type_of_meal_plan", y = "booking_status")
plt.show()
```



From the scatter plot we can observe that when the type of meal plan is 3, we can be certain that the booking will not be cancelled.

### Activity 3.3: Multivariate Analysis

In multivariate analysis we try to find the relation between multiple features. This can be done primarily with the help of Correlation matrix.

```
In [23]: corr = train.corr()
corr.style.background_gradient('coolwarm')
```

Out[23]:

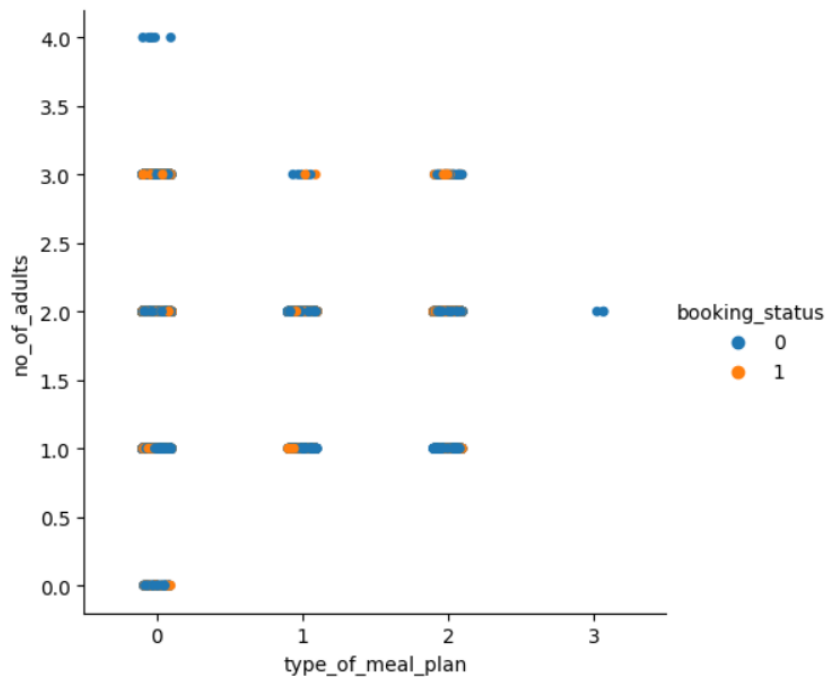
	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space
no_of_adults	1.000000	-0.017565	0.098241	0.103135	0.025139	0.010956
no_of_children	-0.017565	1.000000	0.026761	0.018957	-0.066445	0.026409
no_of_weekend_nights	0.098241	0.026761	1.000000	0.188796	-0.048357	-0.029054
no_of_week_nights	0.103135	0.018957	0.188796	1.000000	-0.076067	-0.054087
type_of_meal_plan	0.025139	-0.066445	-0.048357	-0.076067	1.000000	-0.021426
required_car_parking_space	0.010956	0.026409	-0.029054	-0.054087	-0.021426	1.000000
room_type_reserved	0.163542	0.484223	0.044741	0.066756	-0.146645	0.031255
lead_time	0.098254	-0.051646	0.047559	0.148348	0.134350	-0.076608
arrival_year	0.077915	0.044164	0.054914	0.037240	-0.083469	0.006502
arrival_month	0.018254	0.008244	-0.010060	0.044170	0.019607	-0.018159
arrival_date	0.028433	0.020804	0.024026	-0.007471	0.019243	-0.008735
market_segment_type	-0.098518	0.078264	-0.011626	-0.060650	-0.156242	0.109336
repeated_guest	-0.198334	-0.037868	-0.058549	-0.094936	-0.068313	0.116001
no_of_previous_cancellations	-0.043494	-0.015580	-0.015543	-0.024761	-0.018854	0.027336
no_of_previous_bookings_not_canceled	-0.119703	-0.021495	-0.018829	-0.041108	-0.042258	0.062674
avg_price_per_room	0.292910	0.342425	-0.005837	0.029326	0.039359	0.062199
no_of_special_requests	0.179970	0.128325	0.057814	0.037413	-0.054258	0.083360
booking_status	0.094354	0.035093	0.061117	0.096472	0.076685	-0.092640

As we have 18 columns which have numerical values the correlation matrix is of dimensions 18 X 18. These many features can only be parsed by scrolling. From the correlation matrix we try to remove the values which are highly correlated with each other. When 2 values are highly correlated with each other, we can only remove one of them. We remove columns

where the correlation between the columns is above 0.9. Here we have observed that no columns have correlation above 0.9, so we can skip this step.

Next we plot a catplot where we compare the type of meal plan and the number of adults with the booking status.

```
In [22]: sns.catplot(data = train, x = "type_of_meal_plan", y = "no_of_adults", hue='booking_status')  
plt.show()
```



Here the booking status 0 is represented with blue colour and booking status 1 is represented with orange colour. We have observed that when the number of adults is 4, the booking is not cancelled. Also when the meal type is 3, the booking will not be cancelled.

## Milestone 4: Data Pre-processing

### Activity 1: Splitting features and target

```
In [24]: x = train.drop('booking_status', axis = 1)
         y = train['booking_status']
```

Here we separate the features and the target variable. The features are stored in X and the target variable is stored in y.

### Activity 2: Balancing of Data

The observations where the booking status is 0 are considerably more than the observations where the booking status is 1. This can cause our model to give only one type of output.

```
In [9]: train['booking_status'].value_counts()
Out[9]: 0    12195
        1     5942
        Name: booking_status, dtype: int64
```

We will use the over sampling technique for balancing the data.

```
In [25]: from imblearn import over_sampling
```

```
In [26]: os = over_sampling.RandomOverSampler()
```

```
In [27]: X, y = os.fit_resample(X,y)
```

```
In [28]: y.value_counts()
```

```
Out[28]: 0    12195
        1    12195
        Name: booking_status, dtype: int64
```

Over\_sampling is imported from the imblearn library. Then Random over sampler is used for balancing the data. This over sampler randomly creates replicas of the observations with booking status as 1. We can see that the data is balanced now.



### Activity 3: Splitting into training and testing data:

We have testing data given separately. But there are no labels assigned to it. Due to this we cannot use the testing data.

We split the training data into training and validation data.

```
In [29]: x_train, x_val, y_train, y_val = train_test_split(X,y,test_size=0.2)
```

We have kept 80 % data for training and 20% is used for validation.

## Milestone 5: Model Building

### Activity 1: Creating a function for model evaluation

We will be creating multiple models and then testing them. It will reduce our monotonous task if we directly write a function for model evaluation.

```
In [30]: def model_evaluation(classifier):
          cm = confusion_matrix(y_val, classifier.predict(X_val))
          counts = [value for value in cm.flatten()]
          labels = [f'{v1}' for v1 in counts]
          labels = np.asarray(labels).reshape(2,2)
          sns.heatmap(cm, annot = labels, cmap = 'Greens', fmt = '')
          y_pred = classifier.predict(X_val)
          yt_pred = classifier.predict(X_train)
          print('The Training Accuracy of the algorithm is ', accuracy_score(y_train, yt_pred))
          print('The Validation Accuracy of the algorithm is ', accuracy_score(y_val, y_pred))
          return [(accuracy_score(y_train * 100, yt_pred * 100) * 100), (accuracy_score(y_val * 100, y_pred * 100) * 100), f1_score(y
```

This function will show the accuracies of prediction of model for training, and validation data. It will also return those accuracies along with the model's F1 score.

### Activity 2: Training and testing the models using multiple algorithms

Now that we have clean data, a function for evaluation, it is time to build models to train the data. For this project, we will be using 7 different classification algorithms to build our models. The best model will be used for prediction.

#### Activity 2.1: K Nearest Neighbors Model

A variable is created with name `knn` which has `KNeighborsClassifier()` algorithm initialised in it. The `knn` model is trained using the `.fit()` function. The model is trained on the `X_train` and `y_train` data, that is the training features and training target variables. This model is then given to the `model_evaluation` function to check its performance.

```
In [31]: knn = KNeighborsClassifier()
          knn.fit(X_train, y_train)
```

```
Out[31]: KNeighborsClassifier
          KNeighborsClassifier()
```

```
In [32]: knn_r = model_evaluation(knn)
```

The Training Accuracy of the algorithm is 0.8671586715867159  
The Validation Accuracy of the algorithm is 0.7984829848298483



Here we can see that the model has achieved 86.7% accuracy on training and has achieved 79.8% accuracy on validation data. We have plotted confusion matrix to check the predictions. The results are stored in a variable named knn\_r.

## Activity 2.2: SVM Model

A variable is created with name svm which has SVC() algorithm initialised in it. The svm model is trained using the .fit() function. The model is trained on the X\_train and y\_train data that is the training features and training target variables. This model is then given to the model\_evaluation function to check its performance.

```
In [33]: svm = SVC()  
svm.fit(X_train, y_train)
```

```
Out[33]: 

▼ SVC

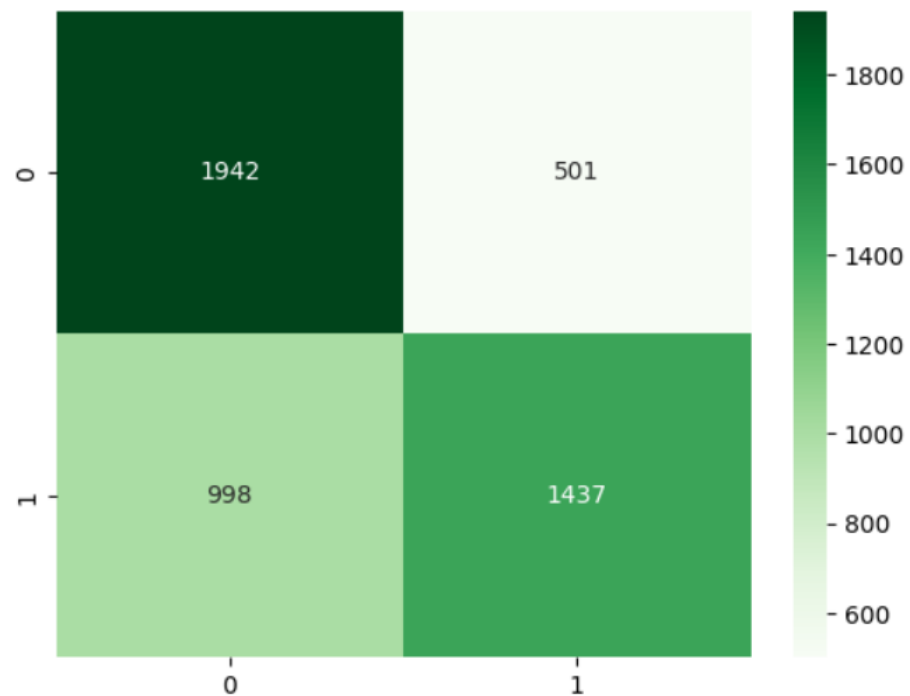


SVC()


```

```
In [34]: svm_r = model_evaluation(svm)
```

```
The Training Accuracy of the algorithm is 0.6929069290692907  
The Validation Accuracy of the algorithm is 0.6927019270192702
```



Here we can see that the model has achieved 69.2% accuracy on training and has achieved 69.2% accuracy on validation data. We have plotted confusion matrix to check the predictions. The results are stored in a variable named `svm_r`.

### Activity 2.3: Naïve Bayes Model

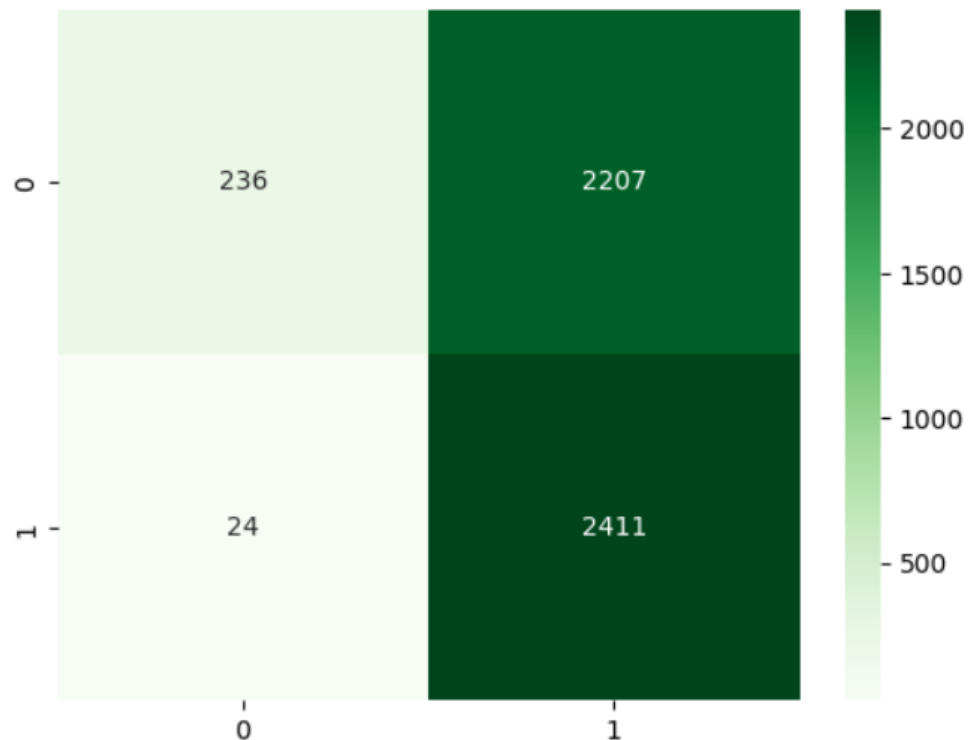
A variable is created with name `gnb` which has `GaussianNB()` algorithm initialised in it. The `gnb` model is trained using the `.fit()` function. The model is trained on the `X_train` and `y_train` data that is the training features and training target variables. This model is then given to the `model_evaluation` function to check its performance.

```
In [35]: gnb = GaussianNB()  
gnb.fit(X_train, y_train)
```

```
Out[35]: GaussianNB  
GaussianNB()
```

```
In [36]: gnb_r = model_evaluation(gnb)
```

```
The Training Accuracy of the algorithm is 0.5427429274292743  
The Validation Accuracy of the algorithm is 0.542640426404264
```



Here we can see that the model has achieved 54.2% accuracy on training and has achieved 54.2% accuracy on validation data. We have plotted confusion matrix to check the predictions. The results are stored in a variable named gnb\_r.

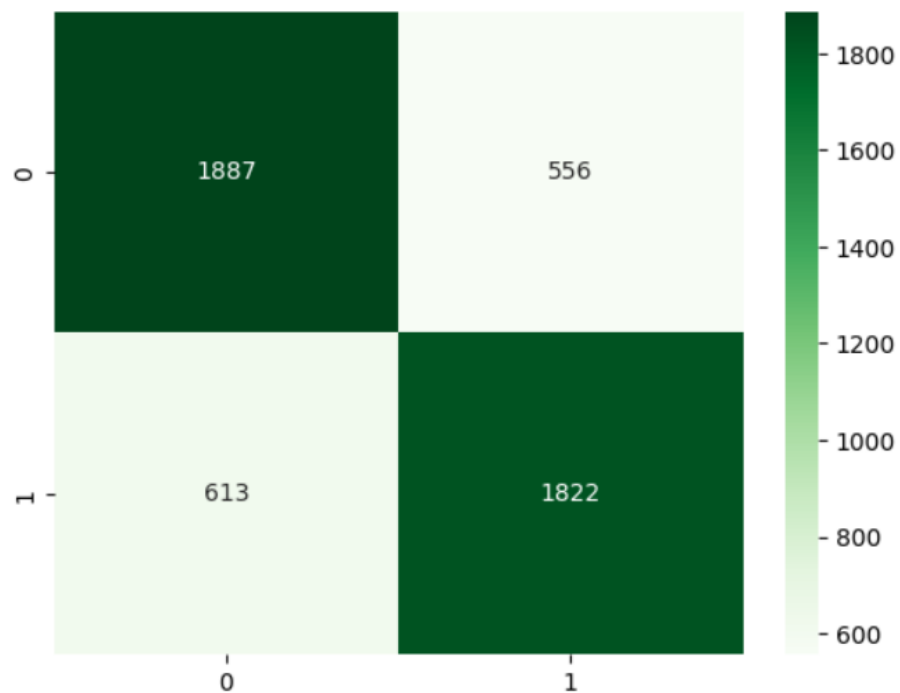
### Activity 2.4: Logistic Regression Model

A variable is created with name lr which has LogisticRegression() algorithm initialised in it. The lr model is trained using the .fit() function. The model is trained on the X\_train and y\_train data that is the training features and training target variables. This model is then given to the model\_evaluation function to check its performance.

```
In [37]: lr = LogisticRegression()  
         lr.fit(X_train, y_train)
```

```
In [38]: lr_r = model_evaluation(lr)
```

The Training Accuracy of the algorithm is 0.7657339073390734  
The Validation Accuracy of the algorithm is 0.7603526035260353



Here we can see that the model has achieved 76.5% accuracy on training and has achieved 76.0% accuracy on validation data. We have plotted confusion matrix to check the predictions. The results are stored in a variable named lr\_r.

### Activity 2.5: Decision Tree Model

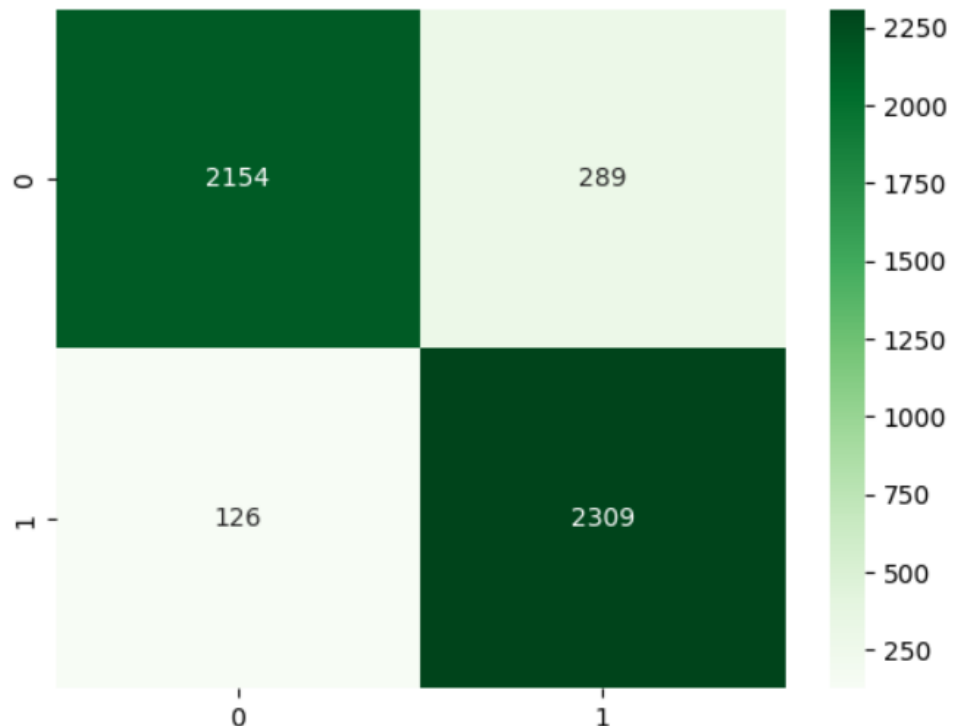
A variable is created with name dt which has DecisionTreeClassifier() algorithm initialised in it. The dt model is trained using the .fit() function. The model is trained on the X\_train and y\_train data that is the training features and training target variables. This model is then given to the model\_evaluation function to check its performance.

```
In [39]: dt = DecisionTreeClassifier()  
dt.fit(X_train, y_train)
```

```
Out[39]: DecisionTreeClassifier  
DecisionTreeClassifier()
```

```
In [40]: dt_r = model_evaluation(dt)
```

```
The Training Accuracy of the algorithm is 0.9957462074620747  
The Validation Accuracy of the algorithm is 0.9149241492414925
```



Here we can see that the model has achieved 99.5% accuracy on training and has achieved 91.4% accuracy on validation data which suggests overfitting. We have plotted confusion matrix to check the predictions. The results are stored in a variable named dt\_r.

## Activity 2.6: Random Forest Model

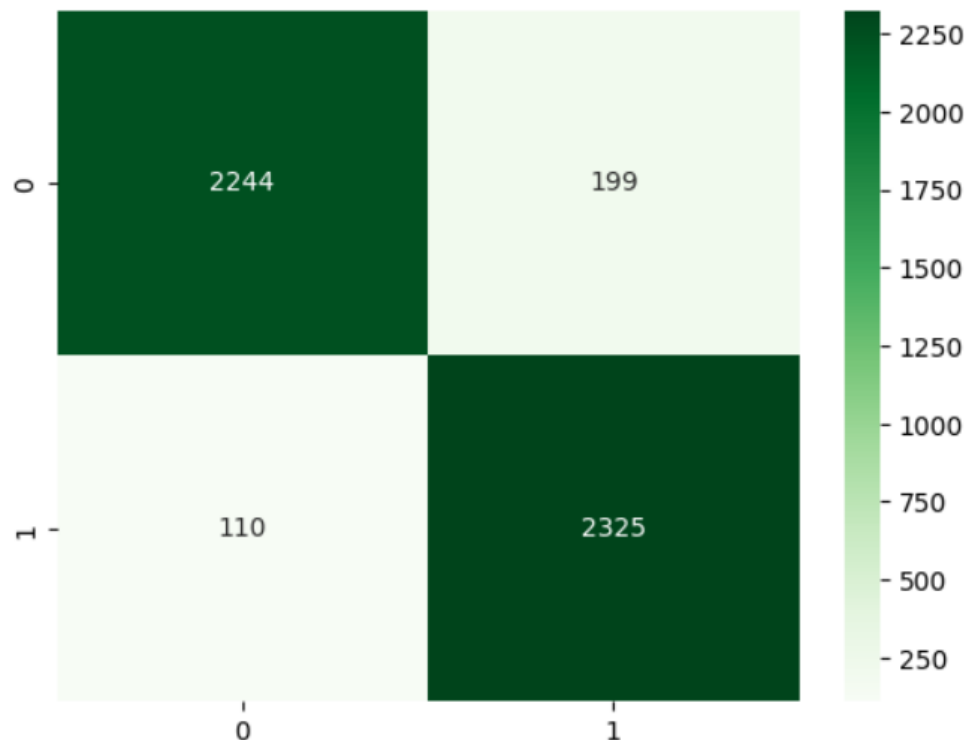
Random Forest Classifier is a Bagging model which utilises multiple decision trees and takes their aggregate to give a prediction. A variable is created with name rf which has RandomForestClassifier() algorithm initialised in it. The rf model is trained using the .fit() function. The model is trained on the X\_train and y\_train data that is the training features and training target variables. This model is then given to the model\_evaluation function to check its performance.

```
In [41]: rf = RandomForestClassifier()  
         rf.fit(X_train, y_train)
```

```
Out[41]: ▾ RandomForestClassifier  
         RandomForestClassifier()
```

```
In [42]: rf_r = model_evaluation(rf)
```

The Training Accuracy of the algorithm is 0.9957462074620747  
The Validation Accuracy of the algorithm is 0.9366543665436654



Here we can see that the model has achieved 99.5% accuracy on training and has achieved 93.6% accuracy on validation data which suggests overfitting. We have plotted confusion matrix to check the predictions. The results are stored in a variable named `rf_r`.

### Activity 2.7: Gradient Boosting Model

A variable is created with name `gb` which has `GradientBoostingClassifier()` algorithm initialised in it. The `gb` model is trained using the `.fit()` function. The model is trained on the `X_train` and `y_train` data that is the training features and training target variables. This model is then given to the `model_evaluation` function to check its performance.

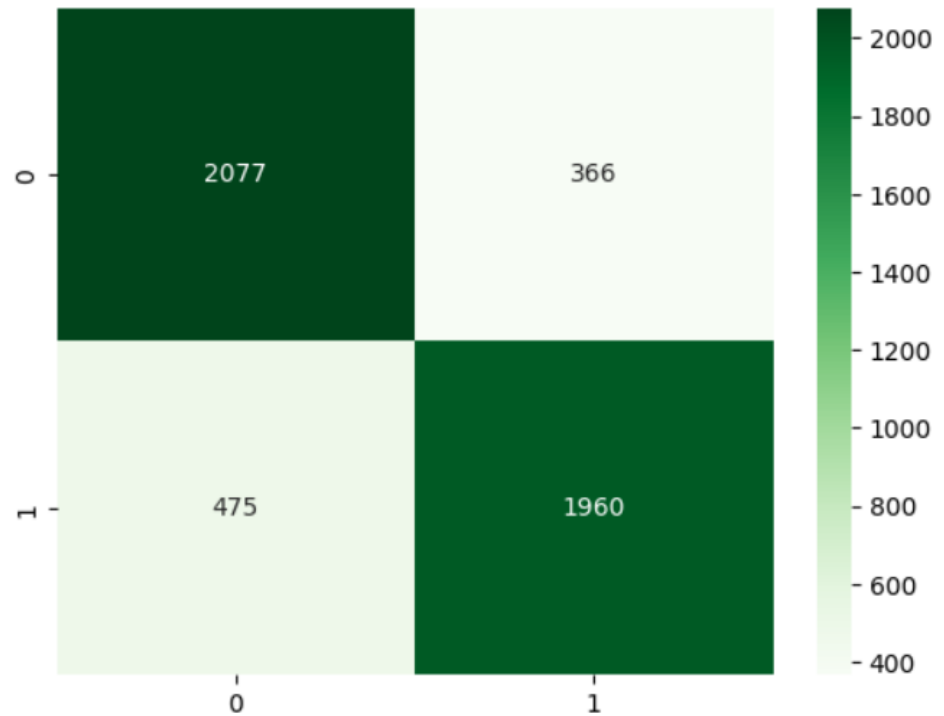
```
In [43]: gb = GradientBoostingClassifier()
         gb.fit(X_train, y_train)
```

```
Out[43]: ▾ GradientBoostingClassifier
         GradientBoostingClassifier()
```



```
In [44]: gb_r = model_evaluation(gb)
```

The Training Accuracy of the algorithm is 0.8293357933579336  
The Validation Accuracy of the algorithm is 0.8275932759327593



Here we can see that the model has achieved 82.9% accuracy on training and has achieved 82.7% accuracy on validation data. We have plotted confusion matrix to check the predictions. The results are stored in a variable named `gb_r`.

## Milestone 6: Performance Testing & Hyperparameter Tuning

### Activity 1: Testing model with Multiple Evaluation metrics

We can check accuracy to test the model. We have already values of the training, and validation accuracies of various models. We can put them in a table and then check for the best model.

```
In [45]: results = pd.DataFrame(data = [knn_r, svm_r, gnb_r, lr_r, dt_r, rf_r, gb_r],  
                                columns = ['Training Accuracy', 'Validation Accuracy', 'F1 Score'],  
                                index = ['K Nearest Neighbors', 'Support Vector Machines',  
                                         'Naive Bayes', 'Logistic Regression',  
                                         'Decision Tree', 'Random Forest', 'Gradient Boost'])
```

```
In [46]: results
```

```
Out[46]:
```

	Training Accuracy	Validation Accuracy	F1 Score
K Nearest Neighbors	86.715867	79.848298	0.802967
Support Vector Machines	69.290693	69.270193	0.657215
Naive Bayes	54.274293	54.264043	0.683681
Logistic Regression	76.573391	76.035260	0.757116
Decision Tree	99.574621	91.492415	0.917544
Random Forest	99.574621	93.665437	0.937689
Gradient Boost	82.933579	82.759328	0.823356

From the table we can see that Random Forest Classifier and Decision Tree Classifier have high accuracies along with a high F1 score. But we will choose Random Forest Classifier to avoid overfitting as it is an aggregate model of multiple decision trees.

### Activity 2: Comparing Model accuracy for different number of features.

Currently the training data has 17 features, which are a high number. If we need to reduce the number of features, we need to check the accuracies for various number of features.

We can check the feature importance using the Random Forest Classifier model.

In the figure below we have created a dictionary with the column names as indexes and the values as their feature importance. Much importance is not assigned to any feature. It is distributed among all the features.

We will keep some number of features for training and check for the accuracy. This process will be repeated a number of times.

```
In [47]: fi = rf.feature_importances_
```

```
In [48]: col = train.columns
```

```
In [49]: feat_imp = {}  
for i, j in zip(fi,col):  
    feat_imp[j] = i
```

```
In [50]: feat_imp
```

```
Out[50]: {'no_of_adults': 0.02451589510890438,  
          'no_of_children': 0.007860620740177576,  
          'no_of_weekend_nights': 0.03853559411143688,  
          'no_of_week_nights': 0.05185374686394164,  
          'type_of_meal_plan': 0.0207749336794491,  
          'required_car_parking_space': 0.008765835562143448,  
          'room_type_reserved': 0.01617138934303456,  
          'lead_time': 0.3095095259226672,  
          'arrival_year': 0.029251040004139677,  
          'arrival_month': 0.08517839262180074,  
          'arrival_date': 0.08870916665908744,  
          'market_segment_type': 0.0554798564141468,  
          'repeated_guest': 0.0028544439844667892,  
          'no_of_previous_cancellations': 0.0001926760710848964,  
          'no_of_previous_bookings_not_canceled': 0.002942865603882197,  
          'avg_price_per_room': 0.14955876581731695,  
          'no_of_special_requests': 0.10784525149231991}
```

We get a dictionary named feat\_imp with 17 column names and their feature importance.

We will drop columns which have very less feature importance.

Let us create a for loop which will train the model and give out the accuracy.

```
In [52]: rfc_results = []
```

```
In [53]: for main in [0.03,0.025,0.020,0.014, 0.008]:  
    to_drop = []  
    for i,j in zip(feat_imp.keys(),feat_imp.values()):  
        if j < main:  
            to_drop.append(i)  
  
    X_new = X.drop(to_drop,axis = 1)  
    y_new = y  
    X1_train, X1_val, y1_train, y1_val = train_test_split(X_new, y_new, test_size=0.2)  
    rfc_new = RandomForestClassifier()  
    rfc_new.fit(X1_train, y1_train)  
    temp1 = model_evaluation1(X1_train.shape[1], rfc_new)  
    rfc_results.append(temp1)
```

The for loop will iterate over values given in the list one by one. The first value will be 0.030 and the last will be 0.008

There is a to\_drop list created.

If the feature\_importance is below threshold then the column name will be added to the to\_drop list.

The columns whose name is in the to\_drop list will be dropped.

The new data will be split into features and target variable. Further they will be split into training, and validation data.

Random Forest Classifier model will be trained and its accuracy will be stored in the list.

This process will go on till all the values for i are iterated.

We then plot a table using the number of features and accuracies.

```
In [55]: randomf = pd.DataFrame(data = rfc_results, columns=['Number of features', 'Training Accuracy', 'Validation Accuracy'])
randomf
```

```
Out[55]:
```

	Number of features	Training Accuracy	Validation Accuracy
0	8	0.995695	0.499795
1	9	0.995439	0.491185
2	11	0.995746	0.502870
3	12	0.995644	0.502665
4	13	0.995695	0.498565

This is the table for random forest Classifier for various features. We can see that as the number of features go on decreasing the overfitting increases. Due to the overfitting we will not reduce the number of features. We will keep them as they are.

### Activity 3: Comparing model accuracy before and after applying hyperparameter tuning

As the accuracies are already so high, we need not do hyperparameter tuning for the models but there is some amount of overfitting in the Random Forest model. So we do hyperparameter tuning for the same.

```
In [56]: from sklearn.model_selection import GridSearchCV
```

```
In [57]: parameters = {
            "max_features": [None, 6, 7, 8],
            "max_depth": [None, 13, 15]
        }
```

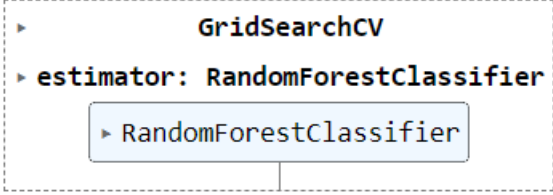
```
In [58]: from sklearn.model_selection import KFold
gdcv = GridSearchCV(estimator=rf, param_grid=parameters)
```

We import GridSearchCV and we give multiple values which we want to check in parameters dictionary.

```
In [58]: from sklearn.model_selection import KFold  
gdcv = GridSearchCV(estimator=rf,param_grid=parameters)
```

```
In [59]: gdcv.fit(X_train, y_train)
```

```
Out[59]:
```



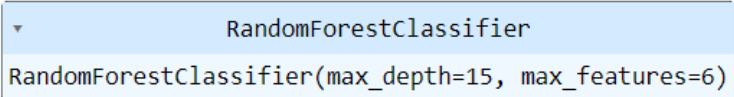
```
In [60]: gdcv.best_params_
```

```
Out[60]: {'max_depth': None, 'max_features': 7}
```

Grid Search CV suggests us to use max\_depth as None and max\_features as 7. We will try these values for Random Forest Classifier model.

```
In [61]: rf_new = RandomForestClassifier(max_depth=15, max_features = 6)  
rf_new.fit(X_train, y_train)
```

```
Out[61]:
```



```
In [62]: model_evaluation(rf_new)
```

```
The Training Accuracy of the algorithm is  0.9445469454694547  
The Validation Accuracy of the algorithm is  0.9114391143911439
```

```
Out[62]: [94.45469454694548, 91.14391143911439, 0.911728647323253]
```

We manually test different values to reduce overfitting and end up with max\_depth as 15 and max\_features as 6. There is very less overfitting now. This is our final model stored in rf\_new variable.

## Activity 4: Checking the Predictions

We need to make sure that the predictions we are getting are correct and we should not rely on accuracy alone.

```
In [64]: pred = rf_new.predict(X_val)
```

```
In [65]: pred
```

```
Out[65]: array([1, 1, 0, ..., 0, 1, 1], dtype=int64)
```

```
In [66]: pd.DataFrame({'Original': y_val, 'Predicted': pred})
```

```
Out[66]:
```

	Original	Predicted
23419	1	1
23480	1	1
6115	0	0
21894	1	1
24140	1	1
...	...	...
16581	0	0
7582	0	0
2327	0	0
19137	1	1
10686	0	1

4878 rows × 2 columns

Here we check the predictions for the validation data. Most of the predictions of our model are correct.

## **Milestone 7: Model Deployment**

### **Activity 1: Save the best model**

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance and saving its weights and configuration. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

After checking the performance, we decide to save the `rf_new` model built with `max_depth` 15 and `max_features` 6.

```
In [63]: pickle.dump(rf_new, open('model.pkl', 'wb'))
```

We save the model using the pickle library into a file named `model.pkl`

### **Activity 2: Integrate with Web Framework**

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the users where he has to enter the values for predictions. The entered values are given to the saved model and prediction is showcased on the UI. This section has the following tasks

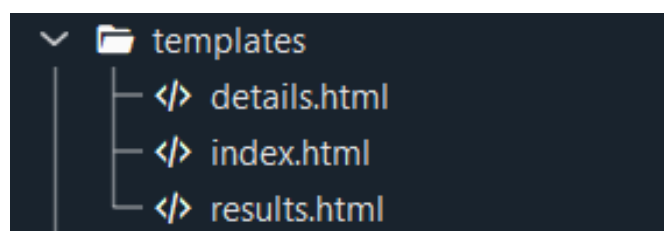
- Building HTML Pages
- Building server-side script
- Run the web application

#### **Activity 2.1: Building HTML pages:**

For this project we create three HTML files namely

- `Index.html`
- `Details.html`
- `Results.html`

And we will save them in the templates folder.



## Activity 2.2: Build Python code

Create a new app.py file which will be store in the Flask folder.

Import the necessary Libraries.

```
from flask import Flask, render_template, request
import pandas as pd
import pickle
```

```
model = pickle.load(open('model.pkl', 'rb'))
app = Flask(__name__)
```

This code first loads the saved Random Forest model from the "model.pkl" file using the "pickle.load()" method. The "rb" parameter indicates that the file should be opened in binary mode to read data from it.

After loading the model, the code creates a new Flask web application object named "app" using the Flask constructor. The "name" argument tells Flask to use the current module as the name for the application.

```
@app.route("/")
def home():
    return render_template('index.html')
```

This code sets up a new route for the Flask web application using the "@app.route()" decorator. The route in this case is the root route "/", which is the default route when the website is accessed.

The function "home()" is then associated with this route. When a user accesses the root route of the website, this function is called.

The "render\_template()" method is used to render an HTML template named "index.html". The "index.html" is the home page.

```
@app.route('/details')
def pred():
    return render_template('predict.html')
```

The route in this case is "/details". When a user accesses the "/predict" route of the website, this function is "index()" called. The "render\_template()" method is used to render an HTML template named "details.html".



```

@app.route('/predict', methods = ['GET', 'POST'])
def predict():
    no_of_adults = request.form['no_of_adults']
    no_of_children = request.form['no_of_children']
    no_of_weekend_nights = request.form['no_of_weekend_nights']
    no_of_week_nights = request.form['no_of_week_nights']
    type_of_meal_plan = request.form['type_of_meal_plan']
    required_car_parking_space = request.form['required_car_parking_space']
    room_type_reserved = request.form['room_type_reserved']
    lead_time = request.form['lead_time']
    arrival_year = request.form['arrival_year']
    arrival_month = request.form['arrival_month']
    arrival_date = request.form['arrival_date']
    market_segment_type = request.form['market_segment_type']
    repeated_guest = request.form['repeated_guest']
    no_of_previous_cancellations = request.form['no_of_previous_cancellations']
    no_of_previous_bookings_not_canceled = request.form['no_of_previous_bookings_not_canceled']
    avg_price_per_room = request.form['avg_price_per_room']
    no_of_special_requests = request.form['no_of_special_requests']

    total = [[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_year, arrival_month, arrival_date, market_segment_type,
              repeated_guest, no_of_previous_cancellations, no_of_previous_bookings_not_canceled,
              avg_price_per_room, no_of_special_requests]]
    d1 = pd.DataFrame(data = total, columns = ['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_year', 'arrival_month',
                                              'arrival_date', 'market_segment_type', 'repeated_guest',
                                              'no_of_previous_cancellations', 'no_of_previous_bookings_not_canceled',
                                              'avg_price_per_room', 'no_of_special_requests'])

    prediction = model.predict(d1)
    prediction = prediction[0]
    if prediction == 0:
        return render_template('results.html', prediction_text = "The Reservation will not be cancelled")
    else:
        return render_template('results.html', prediction_text = "The Reservation will be cancelled")

```

This code sets up another route for the Flask web application using the "@app.route()" decorator. The route in this case is "/predict", and the method is set to GET and POST.

The function "predict()" is then associated with this route. In this function we create a dataframe which has all the 17 column names that we have used in our model. In the details.html page we are going to take inputs from the user, which will be in the form of dropdowns and text.

The values are stored in request.form.values()

These values are taken out one after other and stored in their respective variable names. These variables are then used to create a 2D list named total. This list is then used to create a DataFrame named d1.

This d1 is given for prediction to our model.

This prediction is returned to the results.html page using render\_template()

## Main Function:

This code sets the entry point of the Flask application. The function "app.run()" is called, which starts the Flask deployment server.

```

if __name__ == "__main__":
    app.run(debug= True)

```

## Activity 2.3: Run the Web Application

When you run the “app.py” file this window will open in the console or output terminal. Copy the URL given in the form <http://127.0.0.1:5000> and paste it in the browser.

```
In [1]: runfile('C:/Users/hp/SB/Reservation Cancellation Prediction/Flask/app.py', wdir='C:/Users/hp/SB/Reservation Cancellation Prediction/Flask')
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

When we paste the URL in a web browser, our index.html page will open.



It contains various sections in the header bar such as Home, About, Contact and also has a Predict button. There is some information given on the web page about our model.

If you click on the Predict button on home page or in the header bar you will be redirected to the details.html page.

Our Details.html looks as shown below.

+91 5589 55488 55
hotel@domain.com

RESERVATION CANCELLATION PREDICTION
Home
About
Contact

Input the values for Prediction
Home / Details

Number of Adults:
0

Number of Children:
0

Number of Weekend Nights:
0

Number of Weekday Nights:
0 to 17 in whole numbers

Type of Meal Plan:
0

Parking Space Requirement:
0

Room Type:
0

Lead Time:
0 to 443 in whole numbers

Year of Arrival:
2017

Month of Arrival:
1

Date of Arrival:
0 to 31 in whole numbers

Market Segment Type:
0

Month of Arrival:
1

Date of Arrival:
0 to 31 in whole numbers

Market Segment Type:
0

Repeated Guest:
0

Number of Previous Cancellations:
0

Previous Bookings NOT Cancelled:
0 to 58 in whole numbers

Average Price per room:
0 to 540 in single decimal

Number of Special Requests:
0

Predict

RESERVATION CANCELLATION PREDICTION

A108 Adam Street  
MH 41048, India

Phone: +91 5589 55488 55  
Email: hotel@domain.com

Useful Links

- Home
- About us
- Terms of service
- Privacy policy

Our Services

- Data Analytics
- Web Development
- Product Management
- Machine Learning

Our Newsletter

Subscribe to our newsletter to stay updated.

Subscribe

We are provided with 17 input fields. Some of them are drop downs and others are input boxes. The input boxes have information such as range and decimal values given in them.

We will input some inputs according to the instructions.

We will be redirected to the Results.html page once we click the Predict button.

+91 5589 55488 55
hotel@domain.com

RESERVATION CANCELLATION PREDICTION
Home
About
Contact
PREDICT

Results
Home / Results

Based on the inputs the model predicts that

The Reservation will not be cancelled

RESERVATION CANCELLATION PREDICTION

A108 Adam Street  
MH 41048, India

Phone: +91 5589 55488 55  
Email: hotel@domain.com

Useful Links

- Home
- About us
- Terms of service
- Privacy policy

Our Services

- Data Analytics
- Web Development
- Product Management
- Machine Learning

Our Newsletter

Subscribe to our newsletter to stay updated.

Subscribe

The results say that the Reservation will not be cancelled. This means our model has given output 0.