CAPSTONE PROJECT ON

AIRLINE REFERRAL PREDICTION

CHAPTER 1- OBJECTIVES AND INTRODUCTION

**1.1 INTRODUCTION**



Any child who sees a plane in the sky fantasises about riding in it. Traveling by plane has become a craze, and as an adult, one will consider taking care of the experience. As a result, the opinions of those who have travelled have become increasingly important. This improved people's positive travel experiences, as well as the airlines' ability to understand what their customers are feeling and this increased the amount of room for improvement that airlines can do.

Skytrax <https://www.airlinequality.com/> is one of the leading air transport organizations. It is an international air transport rating organisation headquartered in the United Kingdom whose goal is to boost airline and airport customer experiences around the world. Person feedback left by confirmed customers of most of the world's major airlines make up the dataset.

**1.2 OBJECTIVE :**

The main objective of this project is to predict whether passengers will refer the airline to their friends. In this project we deployed multiple machine learning models to see the performance.

The next chapters have the following sections:

Section 1 - Understanding data

Section 2 - Data preparation

Section 3 - Exploratory data analysis

Section 4- Imputation of NaN values

Section 4 - Feature Engineering

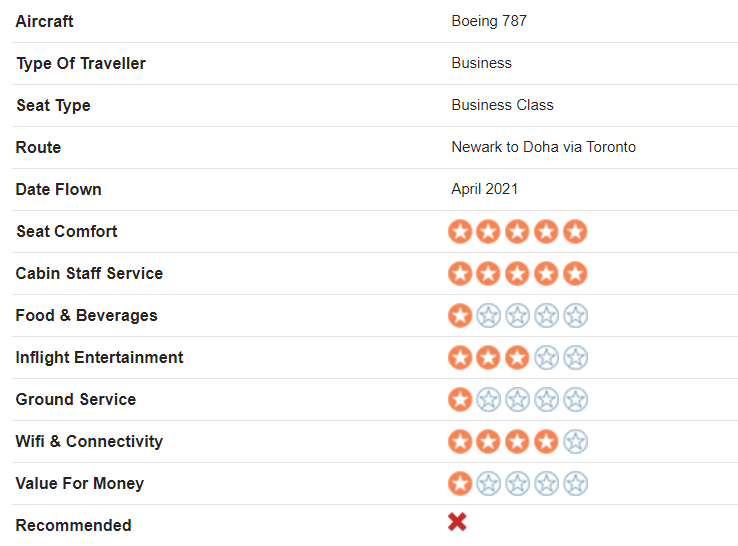
Section 5 - Working different models

Section 6 - Evaluating model

Section 7 - Conclusions

CHAPTER 2 - UNDERSTANDING DATA AND DATA PREPARATION

**2.1 UNDERSTANDING DATA:**



This snapshot is taken from the skytrax website. The customer who travelled can give reviews and can also add review text.

**Description of features :**

airline: Name of the airline.

overall: Overall point is given to the trip between 1 to 10.

author: Author of the trip

review date: Date of the Review customer review: Review of the customers in free text format

aircraft: Type of the aircraft

traveller type: Type of traveler (e.g. business, leisure)

cabin: Cabin at the flight date flown: Flight date

seat comfort: Rated between 1-5

cabin service: Rated between 1-5

foodbev: Rated between 1-5 entertainment: Rated between 1-5

groundservice: Rated between 1-5

value for money: Rated between 1-5

recommended: Binary, target variable.

Mount the drive and load the file

# Mounting drive

from google.colab import drive

drive.mount('/content/drive')

missing\_values = ['N/a', 'na', 'np-nan', ‘None’, ‘none’]

After mounting the drive the next step is to import the required libraries. Python has a wide number of libraries which makes the work easier. Here pandas, numpy, matplotlib, seaborn, math, nltk, sklearn etc., libraries are used.

# importing libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

sns.set\_palette('Set2')

from sklearn.feature\_extraction.text import CountVectorizer

import nltk

from nltk import tokenize, pos\_tag

from nltk.corpus import stopwords, wordnet

from nltk.sentiment.vader import SentimentIntensityAnalyzer

from nltk.stem import WordNetLemmatizer

import datetime as dt

import dateutil

import importlib

#Reading data

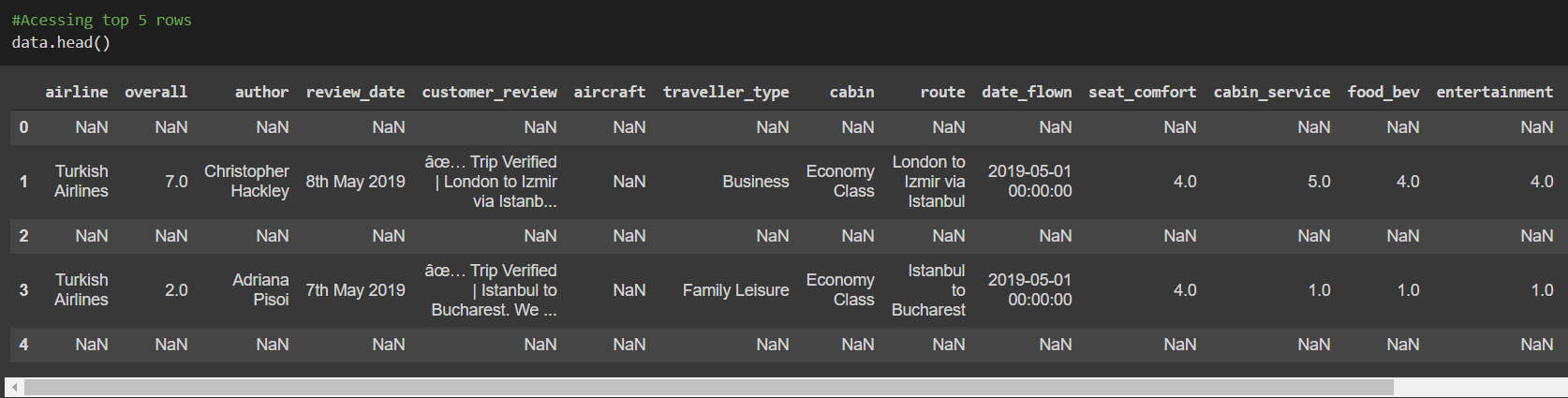
data = pd.read\_excel("/content/drive/MyDrive/Almabetter/Cohort Nilgiri/capstone projects/capstone-3/Copy of data\_airline\_reviews.xlsx", na\_values= missing\_values)

The file is in .xlsx format. So read\_excel is used.

#Shape of dataset

data.shape

The shape of data is (131895,17)



The next step after seeing this dataset is to check the number of null values.

# Checking for null values

data.isnull().sum()

The data set contains a higher number of null values. Data cleaning has become highly essential.

**2.2 PREPARING DATA**

In this step the first foremost thing we are interested in is cleaning data.

1. Dropping rows having entire row as Nan

# Dropping rows if the entire row is null

data.dropna(how = 'all',inplace = True)

Looks like all the odd rows are left empty in excel so the no.of rows having complete Nan in the row are more.

1. Dropping columns which don't add value for the analysis

# Removing columns that are not required

data.drop(columns = ['aircraft','author'],inplace = True)

The Reason behind dropping aircraft are

1. This column has null values more than 80%

2. There are more than 2000 distinct values. As a result we cannot draw any value out of it.

To make the columns names understandable they are renamed

# Renaming columns

data.rename(columns={'overall':'review\_score', 'customer\_review':'review\_text'}, inplace=True)

1. Dropping duplicates

#Dropping the duplicates by keeping the first occurence

data = data.drop\_duplicates(keep= 'first')

Now we need to check for Nan’s again

data.isnull().sum()

airline 0

review\_score 1782

review\_date 0

review\_text 0

traveller\_type 23643

cabin 2478

route 23670

date\_flown 23749

seat\_comfort 4972

cabin\_service 4943

food\_bev 12842

entertainment 20953

ground\_service 24014

value\_for\_money 1856

recommended 1422

dtype: int64

Shape - (61183,15)

To impute these null values we need to do some plotting. To understand the data more. In the next chapter that is performed.

CHAPTER 3 - EXPLORATORY DATA ANALYSIS

The primary goal of EDA is to support the analysis of data prior to making any conclusions. It may aid in the detection of apparent errors, as well as a deeper understanding of data patterns, the detection of outliers or anomalous events, and the discovery of interesting relationships between variables.

Note : Before plotting is performed, review features are scaled which are ranged from 1 to 5 are now made to 1 to 10 so that we can visualize patterns and make conclusions out of them.

def scaled\_feature(feature\_to\_be\_scaled):

'''scaling entire column by multiplying by 2 so that all ratings are given out of 10'''

airline\_data[feature\_to\_be\_scaled] = airline\_data[feature\_to\_be\_scaled]\*2

**3.1 STACKED PLOTS**

#Stacked plot of rating features

def stacked\_plot(feat):

''' Stacked plot of rating features'''

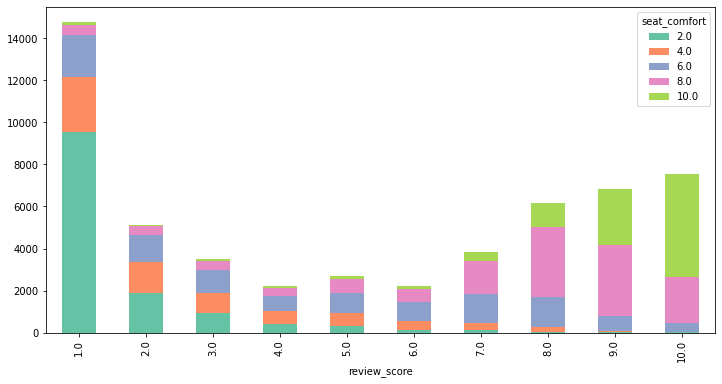
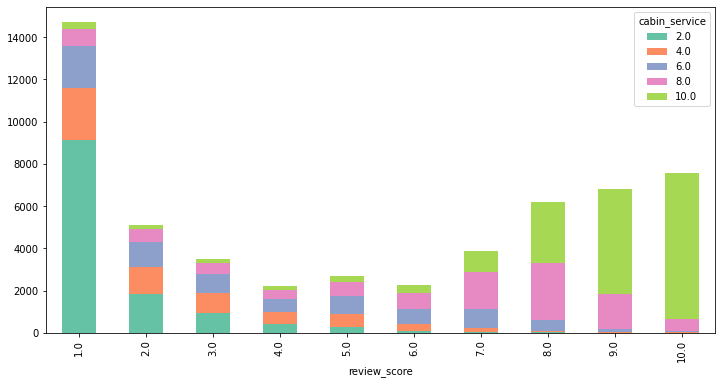
x = airline\_data.groupby([airline\_data['review\_score']])

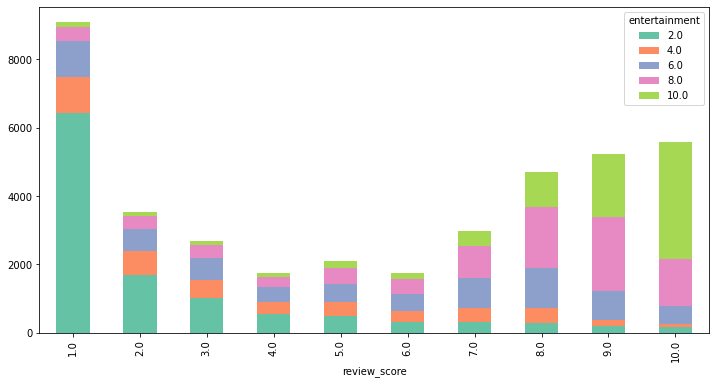
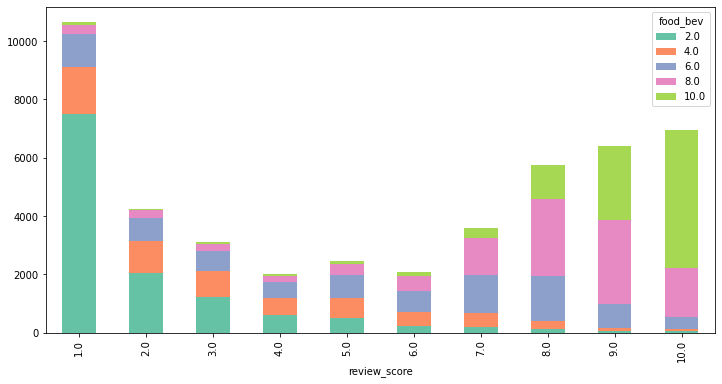
x[feat].value\_counts().unstack().plot(kind= 'bar',stacked = True, figsize=(12,6))

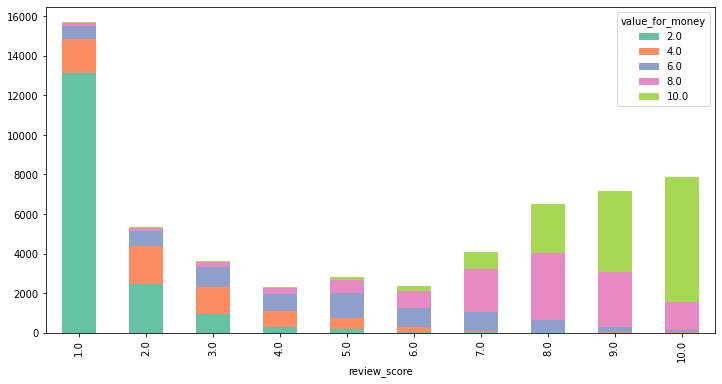
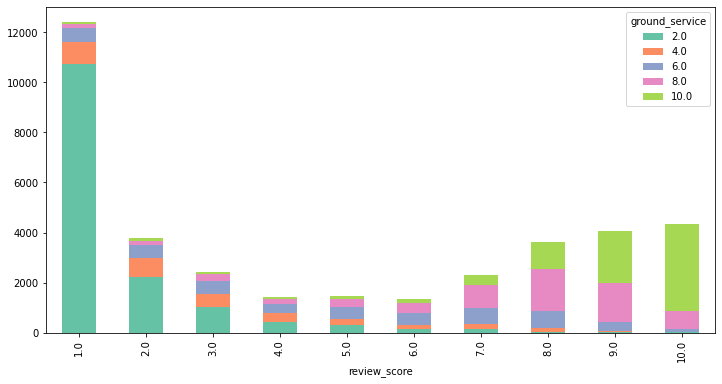
review\_features = ['seat\_comfort','cabin\_service','food\_bev','entertainment', 'ground\_service', 'value\_for\_money']

for feat in review\_features:

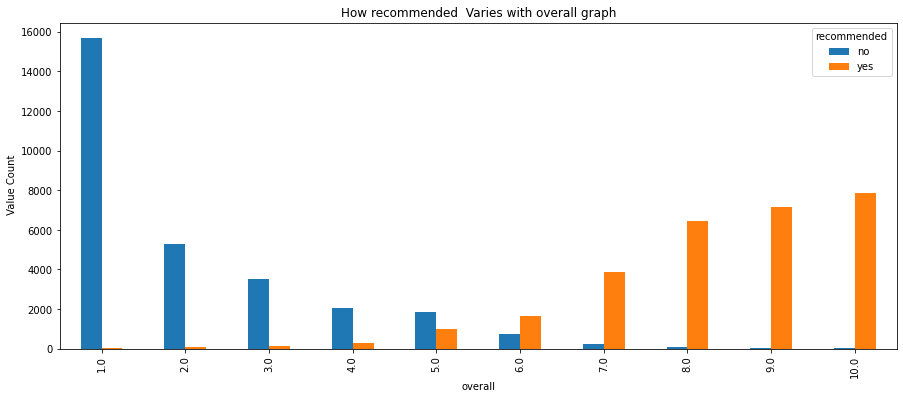
stacked\_plot(feat)





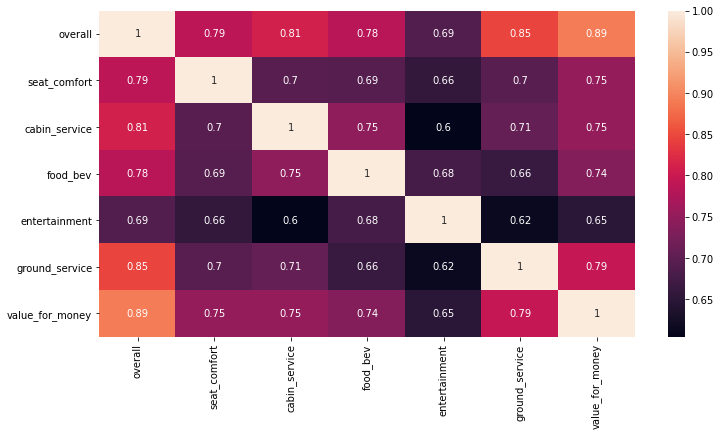


From these graphs we are able to see that these features are related to each other.

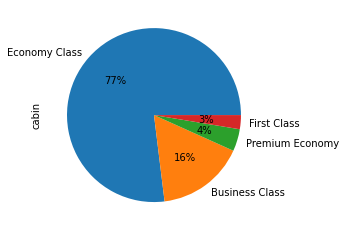


Let us check it with a correlation heat map.

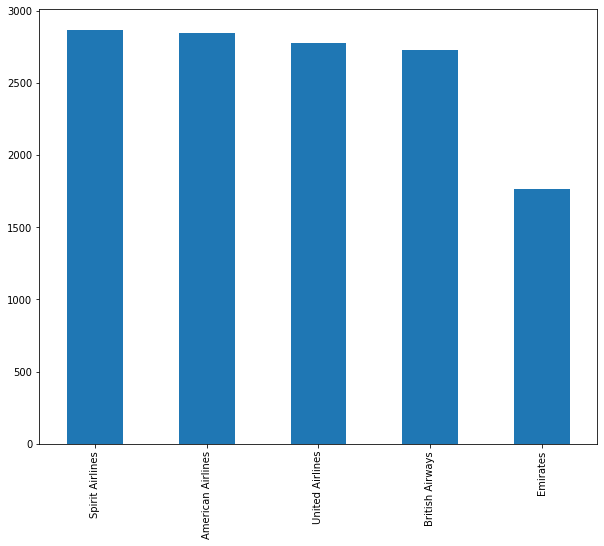
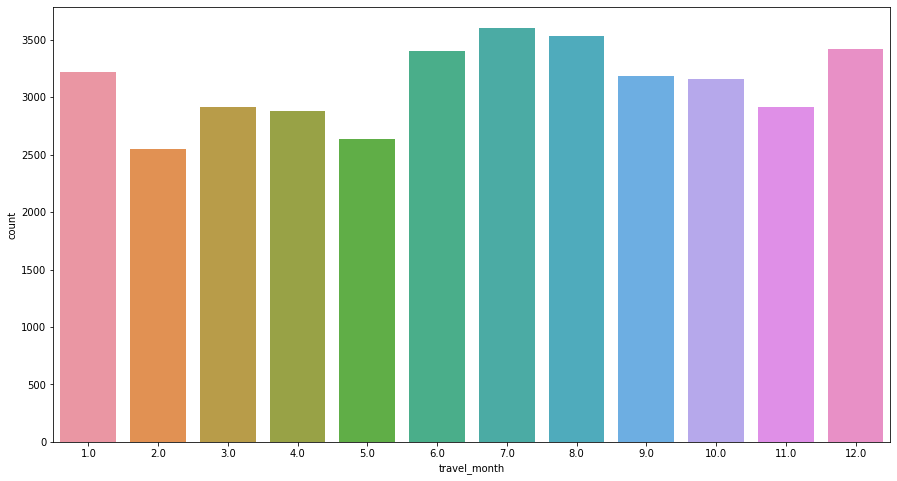
**3.2 CORRELATION MAP**



**3.3 PIE CHART**



**3.4 BAR PLOTS**



**3.5 CONCLUSION POINTS FROM THE ABOVE PLOTS**

Points that can be taken from the above plots

1. Review features are having high correlation with review score
2. Recommended is having high dependence on review score.
3. From correlation it can be observed that rating columns are having highly positive correlation with dependent variable.
4. Since economy class fares are less expensive, 77 percent of passengers opted to travel in this class.
5. Highest travel history is present in the month of july
6. The top 5 airlines preferred are Spirit, American, United, British, Emirates Airlines.

For imputing independent variables we can draw conclusions from these graphs.

CHAPTER 4 : IMPUTATION OF NAN VALUES AND FEATURE ENGINEERING

**4.1 IMPUTATION OF NAN VALUES OF INDEPENDENT VARIABLES :**

There are NaN values in Review\_features, review\_score, travller\_type, cabin\_type and recommended(dependent variable).

The first part here is dealing with NaN values in independent variables.

**4.1.1 Filling review\_score :**  To fill this take the average of the review\_ features and round off to get a rating from 1-10.

# Filling review\_score

y = airline\_data.drop(columns = 'review\_score')

airline\_data['avg'] = round(y.mean(axis=1))

airline\_data['review\_score'].fillna(value= airline\_data['avg'],inplace = True)

**4.1.2 Filling review\_features :** If the row is having review\_score then take that value to impute other review\_features

airline\_data['seat\_comfort'].fillna(value= airline\_data['review\_score'],inplace = True)

airline\_data['cabin\_service'].fillna(value= airline\_data['review\_score'],inplace = True)

airline\_data['food\_bev'].fillna(value= airline\_data['review\_score'],inplace = True)

airline\_data['entertainment'].fillna(value= airline\_data['review\_score'],inplace = True)

airline\_data['ground\_service'].fillna(value= airline\_data['review\_score'],inplace = True)

airline\_data['value\_for\_money'].fillna(value= airline\_data['review\_score'],inplace = True)

Now we have rows which don't have any filled columns of rating columns. There were 143 rows . For them we choose to drop. As all the features are empty there is no way we can take value from them.

# re-ordering the index as rows are removed

airline\_data.reset\_index(drop=True,inplace = True)

**4.1.3 One-hot encoding :** With the help of one hot encoding can fill the NaNs in travel type and cabin type. This encoding creates dummy variables for each unique value present in the feature and fills them with 1 and 0 based on the presence.

# Filling travel\_type and cabin type

airline\_data= pd.concat([airline\_data,pd.get\_dummies(airline\_data['traveller\_type'])],axis=1)

airline\_data= pd.concat([airline\_data,pd.get\_dummies(airline\_data['cabin'])],axis=1)

airline\_data.drop(columns=['traveller\_type','date\_flown','route','cabin'],inplace= True)

**4.2 IMPUTATION OF NAN VALUES OF DEPENDENT VARIABLES :**

To impute NaN values in the recommended column the method used is using the review\_text and based on the sentiment we can impute the values. For this a model is developed.

**Model - Naive Bayes classifier**

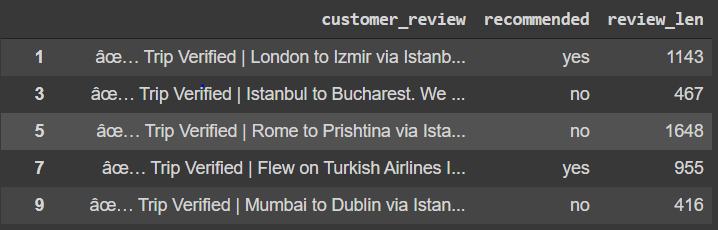
# creating a separate dataset to perform

# data for naive bayes model

text\_df= airline\_data[['customer\_review','recommended']]

# Adding a feature in text df based on string length of the review text

text\_df['review\_len']= text\_df['customer\_review'].str.len()



# doing groupby to plot bar graphs on bases of yes and no

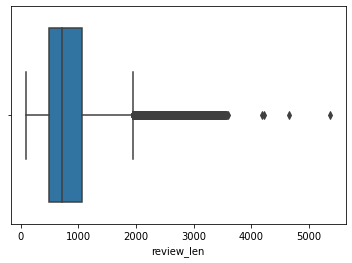
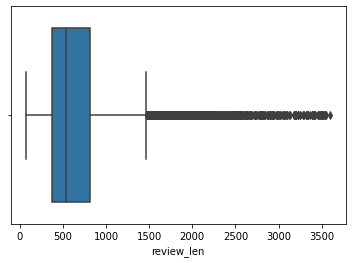
GN= text\_df.groupby('recommended')

for name , name\_df in GN:

print(name)

sns.boxplot(x='review\_len',data= name\_df)

plt.show()



We can draw the conclusion from the above boxplot that as the length of the text increases, the recommended value will be no.

**TEXT PROCESSING**

# import re for regularExpression

# importing natural language toolkit

import re

import nltk

# importing stopwords from nltk corpus

from nltk.corpus import stopwords

# downloading all stopwords

nltk.download('stopwords')

nltk.download('wordnet')

stop\_words=stopwords.words('english')

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

def text\_cleaning(data):

#remove all special character

processed\_feature = re.sub(r'\W', ' ', str(data))

# remove all single characters

processed\_feature= re.sub(r'\s+[a-zA-Z]\s+', ' ', processed\_feature)

# Remove single characters from the start

processed\_feature = re.sub(r'\^[a-zA-Z]\s+', ' ', processed\_feature)

# Substituting multiple spaces with single space

processed\_feature = re.sub(r'\s+', ' ', processed\_feature, flags=re.I)

# Removing prefixed 'b'

processed\_feature = re.sub(r'^b\s+', '', processed\_feature)

# Converting to Lowercase

processed\_feature = processed\_feature.lower()

# removing stopword

processed\_feature = processed\_feature.split(' ')

processed\_feature = [lemmatizer.lemmatize(i) for i in processed\_feature]

processed\_feature = ' '.join([i for i in processed\_feature if i not in stop\_words])

return processed\_feature

text\_df['tokenized\_mess'] = text\_df['customer\_review'].apply(text\_cleaning)

After text cleaning to deploy a model first thing is to separate test and train set

# New dataframes

text\_df\_1 = text\_df.dropna()

text\_df\_2 = text\_df[text\_df['recommended'].isna()]

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.model\_selection import train\_test\_split

#Creating testing and training dataset

X\_train,X\_test,y\_train,y\_test = train\_test\_split(text\_df\_1['tokenized\_mess'],text\_df\_1['recommended'],test\_size=0.25)

**TF-idf vectorization -**

# there are more than 10K features

# setting max\_features to 7500 for system performance

vectorization = TfidfVectorizer(max\_features=7500,min\_df=7,max\_df=0.8)

X\_train = vectorization.fit\_transform(X\_train).toarray()

X\_test = vectorization.transform(X\_test).toarray()

from sklearn.naive\_bayes import GaussianNB

from sklearn.naive\_bayes import MultinomialNB

from sklearn.naive\_bayes import BernoulliNB

# Importing Classification report

from sklearn.metrics import classification\_report

**GaussianNB**

#Applying Gaussian naive bayes

GNB = GaussianNB().fit(X\_train,y\_train)

y\_train\_pred\_gnb = GNB.predict(X\_train)

y\_test\_pred\_gnb = GNB.predict(X\_test)

**MultinomialNB**

#Applying Multinomial Naive bayes

MNB = MultinomialNB().fit(X\_train,y\_train)

y\_train\_pred\_mnb = MNB.predict(X\_train)

y\_test\_pred\_mnb = MNB.predict(X\_test)

**BernoulliNB**

#Applying Bernoulli Naive bayes

BNB = BernoulliNB().fit(X\_train,y\_train)

y\_train\_pred\_bnb = BNB.predict(X\_train)

y\_test\_pred\_bnb = BNB.predict(X\_test)

With the help of classification report we see that multinomial is performing best in all naive bayes classifiers. Therefore now we have a model with 86% accuracy. We use this model to fill the dependent column.

Imputing missing values of recommended column

recommended\_nan = airline\_data['recommended'].isna()

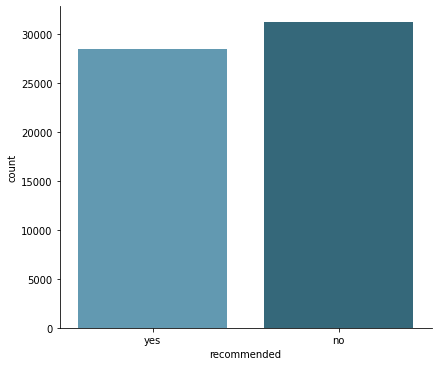
# imputation of dependent variable with prediction MNB model

main\_df.loc[recommended\_nan,'recommended'] = text\_df\_2['recommended']

# replacing yes =1 and no =0 in recommended column

main\_df['recommended'].replace({'yes':1,'no':0},inplace=True)

Checking for imbalances -



Now our data is free from NaN values.

**4. 3 ADDING NEW FEATURE BASED ON REVIEW SCORE**

The review score column ranges from 1to 10. Classifying the reviews into positive and negative sentiments.

# classifying the review score into one of the 3 categories: Positive, Negative

def classify\_review\_score(df):

"""Return:

- 'pos' if the review score is positive (>5),

- 'neg' if the review score is negative (<=5). """

pos\_neg= None

if (df['review\_score'] <= 5):

pos\_neg = 0

else:

pos\_neg = 1

return pos\_neg

# Adding a feature

airline\_data['pos\_neg'] = airline\_data.apply(lambda x: classify\_review\_score(x),axis=1)

**4.4 HANDLING DATES**

# Function to convert the column to datetime object

def date\_timestamp(df\_ , date\_col):

if (isinstance(df\_[date\_col],dt.datetime)):

date\_timestamp = df\_[date\_col]

else:

date\_timestamp = dateutil.parser.parse(df\_[date\_col])

return date\_timestamp

# Changing review\_date object type

airline\_data['review\_date'] = airline\_data.apply(lambda x: date\_timestamp(x, 'review\_date'), axis=1)

Creating a function to separate columns for date, month, year and fetch them from the review\_date

# Add other augmented features

airline\_data['review\_date\_day'] = airline\_data.apply(lambda x: get\_review\_date\_day(x),axis=1)

airline\_data['review\_date\_month'] = airline\_data.apply(lambda x: get\_review\_date\_month(x),axis=1)

airline\_data['review\_date\_year'] = airline\_data.apply(lambda x: get\_review\_date\_year(x),axis=1)

**4.5 VADER (Valence Aware Dictionary for Sentiment Reasoning) :**

**4.5.1 Introduction and Using VADER :**

To do sentimental analysis we will use NLTK (Natural Language Toolkit), more specifically we use a tool named VADER.

In essence, it analyses a text and returns a dictionary with four keys. The letters 'neg', 'neu', and 'pos' stand for 'Negative,"Neutral,' and Positive,' respectively.

The final key is called compound, and it is a mix of the previous three here it is termed as polarity.

To see the source code of VADER <https://www.nltk.org/_modules/nltk/sentiment/vader.html> here.

Using VADER to do sentimental analysis on review text and to extract sentiment of positive and negative out of it.

#Copying data to sent\_analysis

sent\_analysis = airline\_data.copy()

We need to download and instal additional data for NLTK to use VADER; in fact, several of its tools require a second download step to get the requisite collection of data (typically coded lexicons) to function properly.

# Downloading packages

nltk.download('punkt')

nltk.download('vader\_lexicon')

Initializing sentiment Intensity analyzer.

# Initiating

sid = SentimentIntensityAnalyzer()

Copy all the review texts to a list named review\_list (here)

# copy review text to review list

reviews\_list = sent\_analysis['review\_text'].copy()

Adding polarity score to the dataset

# Augment the dataset with the overall polarity score of the review, as obtained using VADER on the review level.

reviews\_polarity = []

for i\_review, review in enumerate(reviews\_list):

review\_polarity\_scores = sid.polarity\_scores(review)

review\_polarity\_score\_compound = review\_polarity\_scores['compound']

reviews\_polarity.append(review\_polarity\_score\_compound)

# Adding polarity feature into sent\_analysis data frame

sent\_analysis['polarity'] = reviews\_polarity

# Mapping recommended column. Replacing yes with 1 and no with 0

sent\_analysis['recommended'] = sent\_analysis['recommended'].map({'yes':1 ,'no':0})

**4.5.2 Plots based on polarity score :**

corr\_values = sent\_analysis[['polarity','pos\_neg','recommended']].dropna(axis=0,how='any').corr()

# Get heatmap of correlation matrix on the dataset

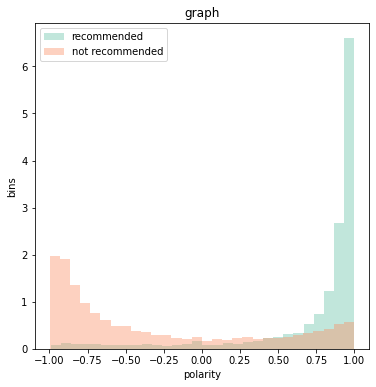
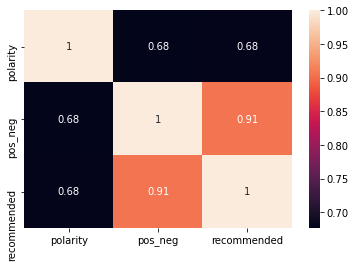
plt.figure(figsize=(6,4))

sns.heatmap(corr\_values,annot = True)

plt.figure(figsize=(6,6))

sns.distplot(sent\_analysis[sent\_analysis['recommended']== 1]['polarity'],hist=True,norm\_hist=True,kde=False,label='recommended',bins=30)

sns.distplot(sent\_analysis[sent\_analysis['recommended']== 0]['polarity'],hist=True,norm\_hist=True,kde=False,label='not recommended',bins=30)



We can see that the polarity ranged from -1 to +1 based on the. Polarity values greater than 0.7 are typically considered to be recommended; otherwise, they are not.

Now we'll create an auxiliary function that we'll use to make the code tidy and readable. It simply converts the compound score into one of three states: 'Negative,' 'Neutral,' or 'Positive,' based on a threshold. The score is 1 for pure positive sentiment, 0 for pure neutral feeling, and -1 for pure negative attitude.

**4.5.3 Adding a new feature rec\_nonrec:**

rec\_nonrec - A polarity scoring rate feature. If the score is less than 0.7, the sentiment is negative and is filled with zero; if the score is greater than 0.7, the sentiment is positive and is filled with one.

# classifying the polarity into one of the 2 categories: Positive, Negative

def classify\_polarity\_score(df):

rec\_nonrec = None

if (df['polarity'] <= 0.7):

rec\_nonrec = 0

else:

rec\_nonrec = 1

return rec\_nonrec

sent\_analysis['rec\_nonrec'] = sent\_analysis.apply(lambda x: classify\_polarity\_score(x),axis=1)

# Plot showing relation between pos\_neg and recommended feature

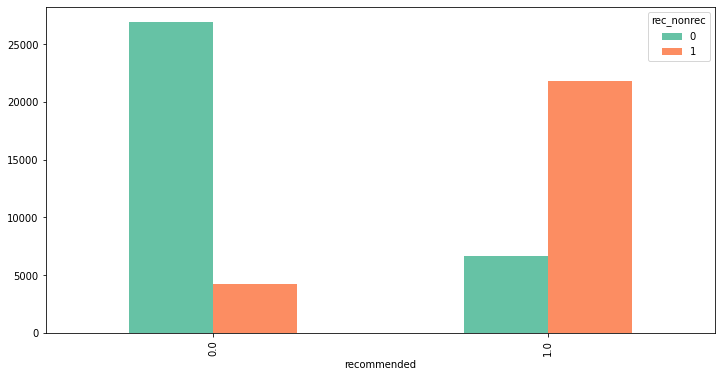
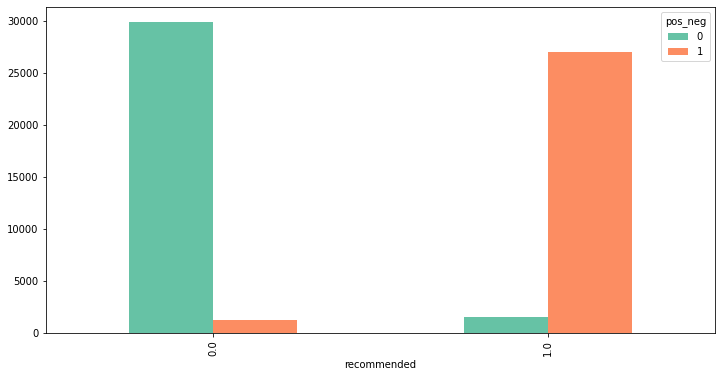
x = sent\_analysis.groupby([sent\_analysis['recommended']])

x['pos\_neg'].value\_counts().unstack().plot(kind= 'bar', figsize=(12,6))

# Plot showing relation between rec\_nonrec and recommended feature

x = sent\_analysis.groupby([sent\_analysis['recommended']])

x['rec\_nonrec'].value\_counts().unstack().plot(kind= 'bar', figsize=(12,6))



We can infer from the above two plots how many classes would be cross-filled if the suggested column is filled with certain features.

Percentages of masking differences between these features

There are 18.47% different values between review\_score sentiment and text\_review sentiment.

There are 6.80% different values between review\_score sentiment and recommended.

There are 20.87% different values between recommended and text\_review sentiment.

**4.5 Word Cloud**

# Total Reviews word cloud

# Import all necessary libraries

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

# Get stopwords from wordcloud library

stopwords = set(STOPWORDS)

# Add some extra words ad hoc for our purpose

words\_ = ['even','given','flight','found','asked','will','now','got','although','one']

stopwords.update(words\_)

# join all reviews

text = " ".join(review for review in sent\_analysis.review\_text)

# Generate the image

wordcloud = WordCloud(stopwords=stopwords, background\_color="white", max\_words=50).generate(text)

# visualize the image

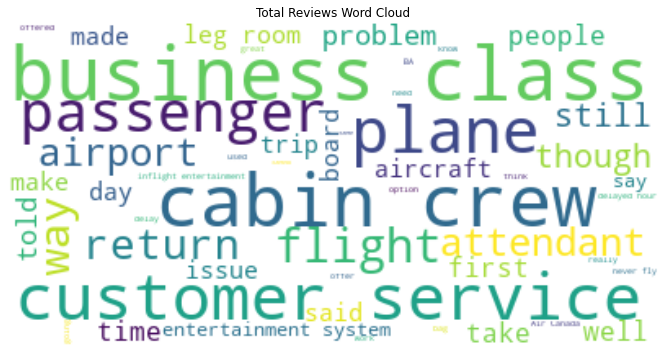
fig=plt.figure(figsize=(15, 8))

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis("off")

plt.title('Total Reviews Word Cloud')

plt.show()



From the word cloud we are able to see that the customer in the review is talking about cabin crew, customer service, business class, issues and problems.

CHAPTER 5 : WORKING WITH DIFFERENT MODELS

Models used are

1. Logistic Regression
2. DecisionTree
3. Random Forest
4. Gradient Boosting
5. XG Boost
6. SVM

**5.1 LOGISTIC REGRESSION**

Logistic regression is a classification technique that predicts the likelihood of a single-valued result (i.e. a dichotomy). A logistic regression yields a logistic curve with values only ranging from 0 to 1. The likelihood that each input belongs to a specific category is modelled using logistic regression. Logistic regression is a fantastic tool to have in your toolbox for classification purposes.

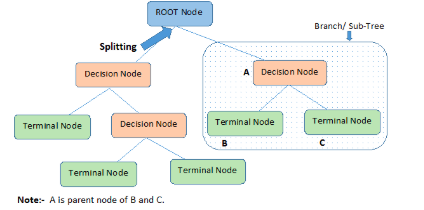
For classification situations, where the output value we want to predict only takes on a small number of discrete values, logistic regression is an important technique to know.

The logistic function offers a number of appealing characteristics. The probability is represented by the y-value, which is always confined between 0 and 1, which is exactly what we wanted for probabilities. A 0.5 probability is obtained for an x value of 0. A higher likelihood is also associated with a higher positive x value, while a lower probability is associated with a greater negative x value.

In logistic regression to learn the coefficients of features in order to maximize the probability of correctly classifying the classes. For this maximum likelihood concept is used.

**5.2 DECISION TREE**

A decision tree is a supervised learning technique used to solve categorization problems. Both categorical and continuous input and output variables are supported.



The decision to make strategic splits has a significant impact on a tree's accuracy. The decision criteria for classification and regression trees are different.

To decide whether to break a node into two or more sub-nodes, decision trees employ a variety of techniques. The homogeneity of the generated sub-nodes improves with the generation of sub-nodes. To put it another way, the purity of the node improves as the target variable grows. The decision tree separates the nodes into sub-nodes based on all available variables, then chooses the split that produces the most homogenous sub-nodes.

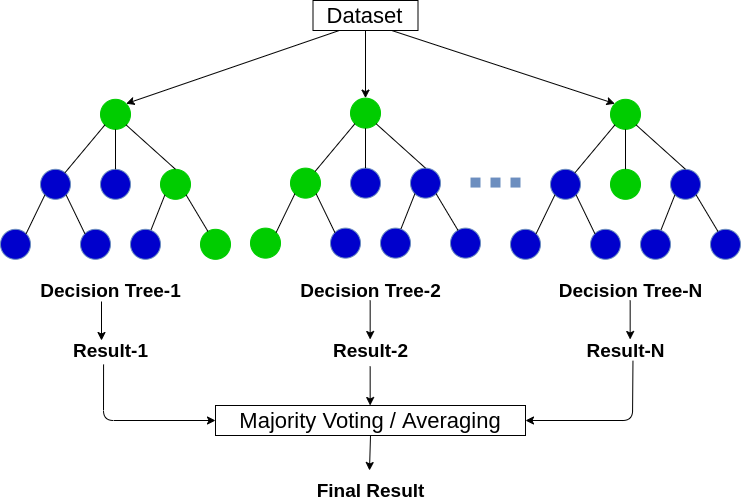
**5.3 RANDOM FOREST**

We create several trees in the Random Forest model rather than a single tree in the CART model.

From the subsets of the original dataset, we create trees. These subsets can contain a small number of columns and rows.

Each tree assigns a categorization to a new object based on attributes, and we say that the tree "votes" for that class.

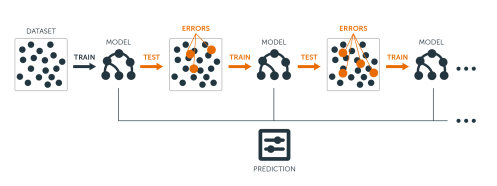
The classification with the highest votes is chosen by the forest.



**5.4 GRADIENT BOOSTING**

The primary idea behind boosting is to incrementally add additional models to the ensemble. In general, boosting approaches the bias-variance tradeoff by starting with a weak model (e.g., a decision tree with only a few splits) and incrementally improving its performance by building new trees, with each new tree in the series attempting to correct where the previous one made the most errors (i.e., each new tree in the series will concentrate on the training rows with the highest prediction errors from the preceding tree).

Since this approach may be generalised to loss functions other than SSE, it is called a gradient boosting machine.



Gradient boosting can be thought of as a type of gradient descent technique. Gradient descent is a fairly general optimization process that may identify the best solutions to a wide variety of problems.

The basic principle behind gradient descent is to iteratively change parameter(s) in order to minimise a cost function. Assume you're a downhill skier competing against a friend. Taking the path with the steepest slope is an excellent way to beat your friend to the bottom.

**5.5 XG BOOST**

XGBoost is a distributed gradient boosting library that has been optimised for performance, flexibility, and portability. It uses the Gradient Boosting paradigm to implement machine learning algorithms. XGBoost is a parallel tree boosting (also known as GBDT, GBM) algorithm that solves a variety of data science problems quickly and accurately.

<https://xgboost.readthedocs.io/en/latest/python/index.html>

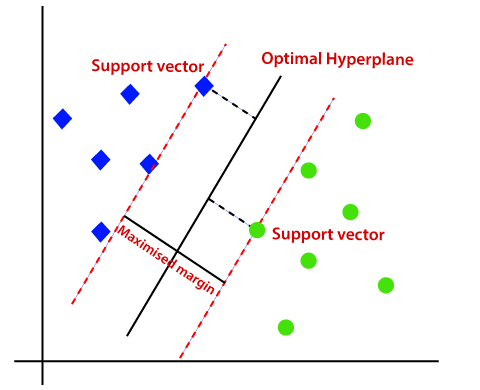
Extreme Gradient Boosting (XGBoost) is just an extension of gradient boosting with the following added advantages:

1. Regularization: Standard GBM implementation has no regularization like XGBoost, therefore it also helps to reduce overfitting. In fact, XGBoost is also known as ‘regularized boosting‘ technique.
2. Parallel Processing: XGBoost implements parallel processing and is blazingly faster as compared to GBM. But hang on, we know that boosting is a sequential process so how can it be parallelized? We know that each tree can be built only after the previous one, but to make a tree it uses all the cores of the system. XGBoost also supports implementation on Hadoop.
3. High Flexibility: XGBoost allows users to define custom optimization objectives and evaluation criteria. This adds a whole new dimension to the model and there is no limit to what we can do.
4. Handling Missing Values: XGBoost has an in-built routine to handle missing values. User is required to supply a different value than other observations and pass that as a parameter. XGBoost tries different things as it encounters a missing value on each node and learns which path to take for missing values in future.
5. Tree Pruning: A GBM would stop splitting a node when it encounters a negative loss in the split. Thus it is more of a greedy algorithm. XGBoost on the other hand makes splits up to the max\_depth specified and then starts pruning the tree backwards and removes splits beyond which there is no positive gain. Another advantage is that sometimes a split of negative loss say -2 may be followed by a split of positive loss +10. GBM would stop as it encounters -2. But XGBoost will go deeper and it will see a combined effect of +8 of the split and keep both.
6. Built-in Cross-Validation: XGBoost allows users to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run. This is unlike GBM where we have to run a grid-search and only a limited value can be tested.
7. Continue on Existing Model: Users can start training an XGBoost model from its last iteration of previous run. This can be of significant advantage in certain specific applications. GBM implementation of sklearn also has this feature so they are even on this point.

**5.6 SVM(Support Vector Machine)**

SVMs take a direct approach to binary classification by attempting to find a hyperplane in a feature space that "best" separates the two classes. In practise, however, finding a hyperplane that completely separates the classes using only the original features is challenging (if not impossible). SVMs get around this by expanding the idea of separating hyperplanes in two different ways.

(1)Expand the feature space to the point where perfect separation of classes is (more) likely, and(2) apply the so-called kernel trick to extend the feature space.



Support Vector - the dividing line between two sets of points that maximises the margin between them A number of the training sites are nearly on the edge of the margin, as represented by the black circles in this diagram. The support vectors are the pivotal elements of this fit, and they are known as the key aspects of this fit.

**5.7 MODELLING ON DATASET**

# importing all models from sklearn

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import GradientBoostingClassifier

from xgboost import XGBClassifier

from sklearn.svm import LinearSVC

# importing metrics for evaluation

from sklearn.metrics import accuracy\_score,confusion\_matrix

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import roc\_auc\_score

from sklearn.metrics import roc\_curve

from sklearn.metrics import auc

#declare the models

lr\_model=LogisticRegression()

dt\_model=DecisionTreeClassifier()

rf\_model=RandomForestClassifier()

gbc\_model=GradientBoostingClassifier()

xgb\_model=XGBClassifier()

svc\_model=LinearSVC()

Mnb\_model=MultinomialNB()

#create a list of models

models=[lr\_model,svc\_model,Mnb\_model,dt\_model,rf\_model,gbc\_model,xgb\_model]

#creating dictionary for storing the confusion matrix

dct\_train={}

dct\_test={}

Lst\_imp=[]

# function for calculation the evaluation matrix

def score\_model(X\_train,y\_train,X\_test,y\_test):

df\_columns=[]

df=pd.DataFrame(columns=df\_columns)

i=0

#read model one by one

for model in models:

model.fit(X\_train,y\_train)

y\_pred=model.predict(X\_test)

y\_pred\_train=model.predict(X\_train)

#compute metrics

train\_accuracy=accuracy\_score(y\_train,y\_pred\_train)

test\_accuracy=accuracy\_score(y\_test,y\_pred)

p\_score\_train=precision\_score(y\_train,y\_pred\_train)

p\_score=precision\_score(y\_test,y\_pred)

r\_score\_train=recall\_score(y\_train,y\_pred\_train)

r\_score=recall\_score(y\_test,y\_pred)

train\_auc = roc\_auc\_score(y\_train,y\_pred\_train)

test\_auc = roc\_auc\_score(y\_test,y\_pred)

fp, tp, th = roc\_curve(y\_test, y\_pred)

#insert in dataframe

df.loc[i,"Model\_Name"]=model.\_\_class\_\_.\_\_name\_\_

df.loc[i,"Train\_Accuracy"]=round(train\_accuracy\*100,2)

df.loc[i,"Test\_Accuracy"]=round(test\_accuracy\*100,2)

df.loc[i,"Precision\_Train"]=round(p\_score\_train\*100,2)

df.loc[i,"Precision\_Test"]=round(p\_score\*100,2)

df.loc[i,"Recall\_Train"]=round(r\_score\_train\*100,2)

df.loc[i,"Recall\_test"]=round(r\_score\*100,2)

df.loc[i,"ROC\_AUC\_Train"]=round(train\_auc\*100,2)

df.loc[i,"ROC\_AUC\_Test"]=round(test\_auc\*100,2)

df.loc[i,'AUC'] = auc(fp, tp)

#inserted in dictionary

dct\_train[model.\_\_class\_\_.\_\_name\_\_]=confusion\_matrix(y\_train,y\_pred\_train)

dct\_test[model.\_\_class\_\_.\_\_name\_\_]=confusion\_matrix(y\_test,y\_pred)

i+=1

# Return the data frame and dictionary

return df,dct\_train,dct\_test

The performance is exceptionally good but we saw a scope of improvement where we can detect anomalies and replace the recommended column with the correct one.

All models are working great on this dataset and getting a good range of accuracies around 95%, which is pretty good. But to make sure our model is not in an overfitting condition performing cross validation techniques would help.

**Cross Validation techniques** used are K-fold and Repeated K-fold. At every fold accuracy is 95% only this means that the models are actually working well on models.

**5.8 WORKING WITH ANOMALIES :**

We can find some suspicious reviews in the data set where the overall score is 9 or 10 but the recommended column is blank, or where the overall score is 1 or 2 but the recommended column is blank. These are anomalies in our data set.

We swapped them out for the right ones. On both the test and train sets, the results have improved by nearly 1%.

CHAPTER 6 : RESULTS AND CONCLUSIONS

**6.1 FEATURE IMPORTANCE**

# Get feature importance

features = X\_train.columns

importances = rf\_optimal\_model.feature\_importances\_

indices = np.argsort(importances)

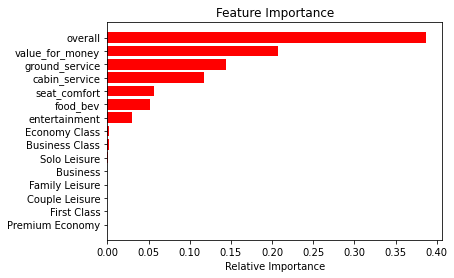
# Plot feature importance graph

plt.title('Feature Importance')

plt.barh(range(len(indices)), importances[indices], color='red', align='center')

plt.yticks(range(len(indices)), [features[i] for i in indices])

plt.xlabel('Relative Importance')



# Get the Shap summary of important features on test data to analyze how each feature contributes in the insurance decisioning process.

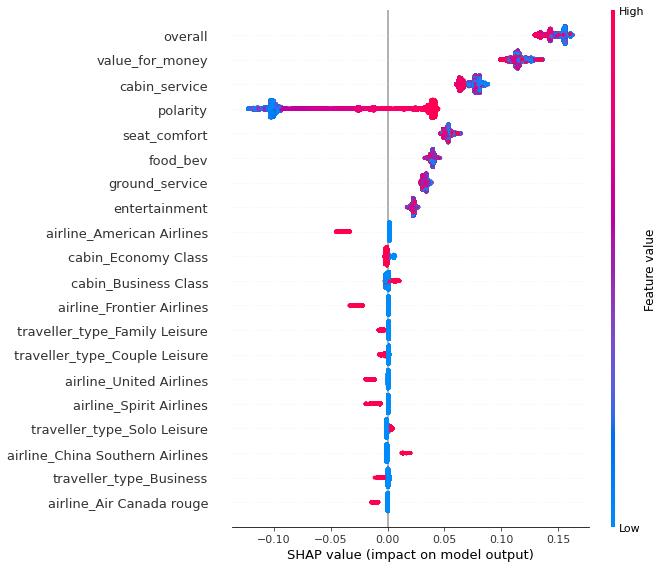
import shap

X\_shap=X\_test

explainer = shap.TreeExplainer(rf\_optimal\_model)

shap\_values = explainer.shap\_values(X\_shap)

shap.summary\_plot(shap\_values[1], X\_shap, plot\_type="dot")



**6.2 RESULTS OF MODELS**

# Getting a data frame with all results of models

result\_df,dct\_train,dct\_test=score\_model(X\_train,y\_train,X\_test,y\_test)

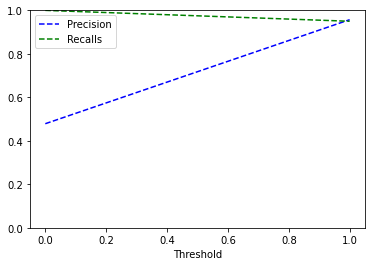
Model Results on test data set

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S.No | MODEL | ACCURACY | PRECISION | RECALL | ROC\_AUC | TIME |
| 1 | Logistic regression | 96.01 | 95.78 | 95.88 | 96.00 | 1.630 |
| 2 | Linear SVC | 96.02 | 95.80 | 95.88 | 96.02 | 0.360 |
| 3 | MultinomialNB | 86.94 | 82.44 | 92.32 | 87.17 | 0.022 |
| 4 | Decision Tree | 100 | 100 | 100 | 100 | 0.618 |
| 5 | Random Forest | 98.99 | 98.94 | 98.94 | 98.98 | 7.604 |
| 6 | Gradient Boosting | 96.05 | 96.11 | 95.61 | 96.03 | 25.075 |
| 7 | XG Boost | 95.93 | 95.89 | 95.59 | 95.92 | 10.345 |

Model Results on train data set

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| S.No | MODEL | ACCURACY | PRECISION | RECALL | ROC\_AUC | TIME |
| 1 | Logistic regression | 95.83 | 95.72 | 95.52 | 95.82 | 1.630 |
| 2 | Linear SVC | 95.85 | 95.72 | 95.55 | 95.84 | 0.360 |
| 3 | MultinomialNB | 86.61 | 82.11 | 91.88 | 86.85 | 0.022 |
| 4 | Decision Tree | 93.99 | 93.76 | 93.60 | 93.97 | 0.618 |
| 5 | Random Forest | 95.79 | 96.08 | 95.04 | 95.76 | 7.604 |
| 6 | Gradient Boosting | 95.78 | 96.00 | 95.11 | 95.75 | 25.075 |
| 7 | XG Boost | 95.82 | 95.87 | 95.33 | 95.80 | 10.345 |

Note : Time is total time by that model.



#confusion matrix of train and test sets

for key,value in dct\_train.items():

print(f'Confusion matrix for {key}')

print(value)

for key,value in dct\_test.items():

print(f' Confusion matrix for {key} ')

print(value)

**6.3 CONCLUSIONS**

* We have built classifier models using 7 different types of classifiers and all these are able to give accuracy of more than 95%.
* The most important features are Overall rating and Value for money that contribute to a model's prediction.
* The classifier model developed will enable airlines ability to identify impactful passengers who can help in bringing more revenue.