import warnings

warnings.filterwarnings('ignore')

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.preprocessing import LabelEncoder

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import f1\_score, accuracy\_score, recall\_score, confusion\_matrix,classification\_report, precision\_score, roc\_auc\_score

from sklearn.model\_selection import train\_test\_split, GridSearchCV, RandomizedSearchCV

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import OneHotEncoder, LabelEncoder, MinMaxScaler, StandardScaler, RobustScaler

from sklearn.compose import ColumnTransformer

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

import joblib

import warnings

warnings.filterwarnings('ignore')

pd.set\_option('display.max\_columns', None)

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, GridSearchCV, KFold, StratifiedKFold

from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoostClassifier

from sklearn.preprocessing import OneHotEncoder

from sklearn.metrics import confusion\_matrix, f1\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn import metrics

from sklearn.feature\_selection import SelectFromModel, SelectPercentile

from sklearn.dummy import DummyClassifier

from sklearn.preprocessing import LabelEncoder, StandardScaler, RobustScaler

import random

import joblib

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, ExtraTreesClassifier, AdaBoostClassifier

from imblearn.over\_sampling import SMOTE, RandomOverSampler

#reading the hotel demand datset downloaded from Kaggle

df = pd.read\_csv(r'C:\Users \hotel\_bookings.csv')

#UNDERSTANDING THE DATA

df.head()

#resulting first five rows in the data set

df.info()

#The data set consists of 32 columns on a whole

df.duplicated().any()

#To check if there are any duplicate values in rows of the data set

df.duplicated().sum()

#To result the number of rows which has duplicate values

df.drop\_duplicates(inplace=False)

#Using pandas built-in method to drop duplicate rows

#By default, this method returns new DataFrame with duplicate rows removed

#We can set the argument(inplace = true) to remove duplicates from original DataFrame.

df.isna()

#Checking if the data set has any missing values

#isna() method is used to find missing values in the data set.

#We can also use isnull() method but it will internally call isna() method and it is an alias of isna() method.

#This method returns TRUE when there is a missing value and FALSE when there is no missing value.

df.isna().sum()

#The above command gives an integer value of all the missing values in each column

#There are two ways to handle missing values - Deleting or Imputing the missing values

#Deleting is a bad approach as it may affect other features.

#Children, Country, Agent and Company are the features which has missing values.

df['children'].mean()

#To replace the missing values in Children column, trying to find mean, median and mode to check which is better.

#mean value is a floating point value but children can never be float

#So not using the mean value to replace missing values.

df['children'].median()

df['children'].mode()

#Mode and median are same which is 0. To check with which to replace with, using one more method below

df['children'].value\_counts()

#value\_counts method returns the most frequent value in that column and the count of it

#respectively for all values in that column.

#For children 0 is the most frequest value which is mode and median.

#So, replacing the missing values of children with 0

df['children'] = df['children'].fillna(0)

df['country'].value\_counts()

#Imputing the missing values in column country with most frequest country.

df['country'] = df['country'].fillna(df['country'].value\_counts().index[0])

df =df.drop(columns = ['agent', 'company'])

#Company column has more than 90% of missing values, so dropping it as it is not needed

#In the agent column the most frequent value is very less but the actual missing values are much high.

#Imputing with mode will not be optimal and as most of the values are NaN, we can impute 0 or drop the agent column

df.isna().sum()

#Below is the missing values after handling all the columns

df['adr'].unique()

df.describe()

#Checking for outliers

#Outlier is value that lies in an abnormal distance from other values.

#If we observe the mean and min or max values of all columns, adr column has a negative value, and the minimum value is

#extremely far from mean value, which means the minimum value is an outlier.

sns.boxplot(x=df['adr'])

#We can visualize outliers using boxplot

df[df['adr'] <0]

#Resulting any value of adr column which has negative values

df = df.drop(df[df.adr < 0].index)

df[df['adr'] < 0]

#Dropping the negative value row and printing to check if there are any more negative values

#We get 0 rows that means no more negative values.

df[df['adr']> 5000]

#If we observe the boxplot above, there is only one value above 5000 which is at extreme compared to others

df = df.drop(df[df.adr > 5000].index)

df[df['adr']> 5000]

#Dropping the row with 5000

df[df['adults']+df['babies']+df['children'] == 0].shape

#There are some rows which has adults,babies,children equal to 0.

#No booking cab be made with adult/children 0. So, check if there are any rows of the above kind.

#There are 180 rows of suck kind, Our algorithm will not learn anything from these rows.

#Dropping all the rows which satisfy above conditiondf1.drop(df1[df1['adults']+df1['babies']+df1['children'] == 0].index, inplace = True)

df.drop(df[df['adults']+df['babies']+df['children'] == 0].index, inplace = False)

df['is\_canceled'].value\_counts()

df['is\_canceled'].value\_counts().plot(kind = 'bar')

#If we look at the bar plot of target variable there is approximately 2:1 ratio between majority and minority class.

#When the data is imbalanced there is a chance that the model will be biased towards majority class.

#Synthetic Minority Oversampling Technique or SMOTE is another technique to oversample the minority class.

#Simply adding duplicate records of minority class often don’t add any new information to the model.

#In SMOTE new instances are synthesized from the existing data. If we explain it in simple words, SMOTE looks into

#minority class instances and use k nearest neighbor to select a random nearest neighbor, and a synthetic instance is

#created randomly in feature space.

#We still haven't balanced the data set as SMOTE tends to create a large no. of noisy data points in feature space.df1['total\_people'] = df1['adults'] + df1['babies'] + df1['children']

df['total\_people'] = df['adults'] + df['babies'] + df['children']

df['total\_stay'] = df['stays\_in\_weekend\_nights'] + df['stays\_in\_week\_nights']

#Adding two new columns

#Combining adults, babies,children into total\_people

#Similarly added total\_stay column

#Data Visualization

df['hotel'].value\_counts().plot.pie(explode=[0.05, 0.05], autopct='%1.1f%%',shadow=True, figsize=(6,4),fontsize=15)

plt.title('Pie Chart for Most Preffered Hotel')

#Pie chart to visualize the ratio between Resort hotel and City hotel

plt.subplots(figsize=(5, 3))

sns.countplot(x='arrival\_date\_year', hue='hotel', data=df);

#The percetage of booking each year?

#More than double bookings were made in the year 2016 than previous year

#In which month most bookings happened?

# groupby arrival\_date\_month and taking the hotel count

bookings\_by\_months\_df=df.groupby(['arrival\_date\_month'])['hotel'].count().reset\_index().rename(columns={'hotel':"Counts"})

# Create list of months in order

months = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December']

# creating df which will map the order of above months list without changing its values.

bookings\_by\_months\_df['arrival\_date\_month']=pd.Categorical(bookings\_by\_months\_df['arrival\_date\_month'],categories=months,ordered=True)

# sorting by arrival\_date\_month

bookings\_by\_months\_df=bookings\_by\_months\_df.sort\_values('arrival\_date\_month')

bookings\_by\_months\_df

plt.figure(figsize=(20,8))

#pltting lineplot on x- months & y- booking counts

sns.lineplot(x=bookings\_by\_months\_df['arrival\_date\_month'],y=bookings\_by\_months\_df['Counts'])

# set title for the plot

plt.title('Number of bookings across each month')

#set x label

plt.xlabel('Month')

#set y label

plt.ylabel('Number of bookings')

#BIVARIATE AND MULTIVARIATE ANALYSIS

#Which hotel type has the highest ADR

#grouping by hotel adr

grup\_by\_hotel=df.groupby('hotel')

highest\_adr=grup\_by\_hotel['adr'].mean().reset\_index()

#set plot size

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

# set labels

plt.xlabel('Hotel type')

plt.ylabel('ADR')

plt.title("Avg ADR of each Hotel type")

#plot the graph

sns.barplot(x=highest\_adr['hotel'],y=highest\_adr['adr'])

#City Hotel has the highest ADR that means city hotels are generating more revenue than resort hotels

#Which hotel has the highest cancellation rate

# booking canceled=1

# booking not canceled= 0

# creating new DataFrame where bookings are cancelled.

canceled\_df=df[df['is\_canceled']==1]

# Grouping by hotel

canceled\_df=canceled\_df.groupby('hotel').size().reset\_index().rename(columns={0: "no\_of\_cancelled\_bookings"})

# adding 'total booking column for calculating the percentage.

canceled\_df['total\_booikngs']=df.groupby('hotel').size().reset\_index().rename(columns={0:"total\_bookings"}).drop('hotel',axis=1)

canceled\_df

#plotting the barchat

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

sns.barplot(x=canceled\_df['hotel'],y=canceled\_df['no\_of\_cancelled\_bookings']\*100/canceled\_df['total\_booikngs'])

#set labels

plt.xlabel('Hotel type')

plt.ylabel('Percentage(%)')

plt.title("Percentage of booking cancellation")

#Percentage of booking cancellation is high in City hotel

#Which distribution channel has the highest cancellation rate

canceled\_df=df[df['is\_canceled']==1] # 1= canceled

#group by distribution channel

canceled\_df=canceled\_df.groupby(['distribution\_channel','hotel']).size().reset\_index().rename(columns={0:'Counts'})

# canceled\_df['Percentage']=canceled\_df['Counts']\*100/df1[df1['is\_canceled']==1][0]

canceled\_df

#set plot size and plot barchart

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

sns.barplot(x='distribution\_channel',y='Counts',hue="hotel",data=canceled\_df)

# set labels

plt.xlabel('Distribution channel')

plt.ylabel('counts')

plt.title('Cancellation Rate Vs Distribution channel')

#Effect of lead\_time on cancellation using scatter plot

# group data for lead\_time:

lead\_cancel\_data = df.groupby("lead\_time")["is\_canceled"].describe()

# use only lead\_times wih more than 10 bookings for graph:

lead\_cancel\_data\_10 = lead\_cancel\_data.loc[lead\_cancel\_data["count"] >= 10]

#show figure:

plt.figure(figsize=(8, 6))

sns.regplot(x=lead\_cancel\_data\_10.index, y=lead\_cancel\_data\_10["mean"].values \* 100)

plt.title("Effect of lead time on cancelation", fontsize=16)

plt.xlabel("Lead time", fontsize=16)

plt.ylabel("Cancelations [%]", fontsize=16)

# plt.xlim(0,365)

plt.show()

#Bookings made few days before are rarely canceled, whereas bookings made one year in advance are canceled very often

hotel\_booking\_df = df.copy()

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

correlation = sns.heatmap(hotel\_booking\_df.corr(), vmin=-1, vmax=1, annot=True, linewidths=1, linecolor='black', cmap = "viridis")

correlation.set\_title('Correlation Matrix of the Hotel Booking', fontdict={'fontsize': 24})