CODE

(KAGGLE FIRST ROUND COMPETITION)

WANDERERS (DS21-68)

Highest Kaggle Submission Score: 0.33468

Team members:

Jathurshan Pradeepkumar (Leader)

Mithunjha Anandakumar

Vinith Kugathasan

Github link:
https://github.com/Jathurshan0330/Data_Storm_v2_DS21-68

Code for Best Submission

```
# -*- coding: utf-8 -*-
"""DataStorm2.0 Day 2- Sub1.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1OZeAhIbpit_Joq-_NuDUQFqqMAxVH-Dr

#Import Libraries

##Neptune Ai
"""

! pip install neptune-client==0.4.132

pip install  neptune-contrib neptune-client

import neptune
from neptunecontrib.monitoring.keras import NeptuneMonitor
neptune.init(project_qualified_name='jathurshan0330/DataStorm2-round1', #
change this to your `workspace_name/project_name`

api_token='eyJhcGlfYWRkcmVzcyI6Imh0dHBzOi8vdWkubmVwdHVuZS5haSIsImFwaV91cmwiOi
JodHRwczovL3VpLm5lcHR1bmUuYWkiLCJhcGlfa2V5IjoiZmJkZjYxNGYtMTA0ZC00ZTc1LWJiMTY
tNzczNjgwZWQ3OTUzIn0=', # change this to your api token
            )

"""##other necessary libraries"""

import os
import sys
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.io
import seaborn as sns
from sklearn.model_selection import train_test_split
import tensorflow as tf
from scipy import stats
from tensorflow.keras.models import Sequential,Model
from tensorflow.keras.layers import Dense,
Input,LSTM,Reshape,Conv2D,Flatten,Dropout,BatchNormalization, LeakyReLU,
concatenate, GRU, GlobalMaxPooling1D, GlobalMaxPooling2D, Bidirectional

!pip install scikit-plot

"""#Read Data

##Mount Drive
"""

from google.colab import drive
drive.mount('/content/drive')
```

```
cd '/content/drive/My Drive/Datastorm2.0'

!ls '/content/drive/My Drive/Datastorm2.0'

"""##Data"""

train_data = pd.read_csv('Hotel-A-train.csv')
print(train_data.head())
print(train_data.shape)
val_data = pd.read_csv('Hotel-A-validation.csv')
print(val_data.head())
print(val_data.shape)
test_data = pd.read_csv('Hotel-A-test.csv')
print(test_data.head())
print(test_data.shape)

print(train_data.isna().sum())
print(val_data.isna().sum())
print(test_data.isna().sum())

train_labels = train_data.pop("Reservation_Status")
print(train_labels.head())

for i in range (len(train_labels)):
  if train_labels[i] == 'Check-In':
    train_labels[i] = 1
  if train_labels[i] == 'Canceled':
    train_labels[i] = 2
  if train_labels[i] == 'No-Show':
    train_labels[i] = 3

print(train_labels.head())

val_labels = val_data.pop("Reservation_Status")
print(val_labels.head())

for i in range (len(val_labels)):
  if val_labels[i] == 'Check-In':
    val_labels[i] = 1

  if val_labels[i] == 'Canceled':
    val_labels[i] = 2
  if val_labels[i] == 'No-Show':
    val_labels[i] = 3

print(val_labels.head())

print("No of Check-In in training data : " +str((train_labels == 1).sum()))
print("No of Canceled in training data : " +str((train_labels == 2).sum()))
print("No of No-Show in training data : " +str((train_labels == 3).sum()))
tot=(train_labels == 1).sum()+(train_labels == 2).sum()+(train_labels ==
3).sum()
print("Ratio of Check-In : Canceled : No-Show  in training data = "
+str((train_labels == 1).sum()/tot)+' : '+str((train_labels ==
2).sum()/tot)+' : '+str((train_labels == 3).sum()/tot))
```

```
print("No of Check-In in validation data : " +str((val_labels == 1).sum()))
print("No of Canceled in validation data : " +str((val_labels == 2).sum()))
print("No of No-Show in validation data : " +str((val_labels == 3).sum()))
tot=(val_labels == 1).sum()+(val_labels == 2).sum()+(val_labels == 3).sum()
print("Ratio of Check-In : Canceled : No-Show  in Validation data = "
+str((val_labels == 1).sum()/tot)+' : '+str((val_labels == 2).sum()/tot)+' :
'+str((val_labels == 3).sum()/tot))

"""#Data Preprocessing

##Extracting features from Check in and reservation date
"""

from datetime import datetime
def days(start_date, end_date):
  start_date = datetime.strptime(start_date, "%m/%d/%Y")
  end_date = datetime.strptime(end_date, "%m/%d/%Y")
  #print((end_date - start_date).days)
  return (end_date - start_date).days

def weekday(date):
  year = datetime.strptime(str(date), '%m/%d/%Y').year
  month = datetime.strptime(str(date), '%m/%d/%Y').month
  day = datetime.strptime(str(date), '%m/%d/%Y').day
  #print(year,month,day)
  x = datetime(year,month,day)
  week_no = x.strftime("%w")   #0 = Sunday, 1 = Monday, 2 = Tuesday, 3 =
Wednesday, 4 = Thurday, 5 = Friday, 6 = Saturday
  return int(week_no)

def weekend(start_date, end_date):
  weeklist = [0,1,2,3,4,5,6,0,1,2,3,4,5,6]
  start = weekday(start_date)
  end = weekday(end_date)
  duration = days(start_date, end_date)
  if duration >= 7:
    return 1
  else:
    stayed = weeklist[start : start + duration]
    #print("start day :", start, "; end day :", end)
    #print(start,duration,stayed)
    if (0 or 1) in stayed:
      return 1
    else:
      return 0

#for training data
week_end_train = []
stay_duration = []
reserve_duration = []
a = 0
temp_a = []
b = 0
temp_b = []
for i in range(len(train_labels)):
  checkin = train_data["Expected_checkin"][i]
```

```
  checkout = train_data["Expected_checkout"][i]
  reserve = train_data["Booking_date"][i]
  if days(reserve,checkin ) == 0 and train_labels[i] != 1:
    a+=1
    temp_a.append(i)
  if days(reserve,checkout ) < 0 and train_labels[i] != 1:
    b+=1
    temp_b.append(i)
  stay_duration.append(days(checkin,checkout))
  reserve_duration.append(days(reserve,checkout ))
  week_end_train.append(weekend(checkin, checkout))

stay_duration = pd.DataFrame(stay_duration, columns=['stay_duration'])
reserve_duration = pd.DataFrame(reserve_duration,
columns=['reserve_duration'])
week_end_train = pd.DataFrame(week_end_train, columns=['week_end'])
print(temp_a)
print(temp_b)
#print(train_data["Expected_checkin"][6])
#print(train_data["Booking_date"][6])
print(a)
print(b)
print(stay_duration.head())
print(stay_duration.shape)
print(week_end_train.head())
print(week_end_train.shape)
print(reserve_duration.head())
print(reserve_duration.shape)

#for validation data
week_end_val = []
stay_duration_val = []
reserve_duration_val = []
a = 0
temp_a = []
b = 0
temp_b = []
for i in range(len(val_labels)):
  checkin = val_data["Expected_checkin"][i]
  checkout = val_data["Expected_checkout"][i]
  reserve = val_data["Booking_date"][i]
  if days(reserve,checkin ) == 0 and val_labels[i] != 1:
    a+=1
    temp_a.append(i)
  if days(reserve,checkin ) < 0 and val_labels[i] != 1:
    b+=1
    temp_b.append(i)
  stay_duration_val.append(days(checkin,checkout))
  reserve_duration_val.append(days(reserve,checkout ))
  week_end_val.append(weekend(checkin, checkout))

stay_duration_val = pd.DataFrame(stay_duration_val,
columns=['stay_duration'])
reserve_duration_val = pd.DataFrame(reserve_duration_val,
columns=['reserve_duration'])
week_end_val = pd.DataFrame(week_end_val, columns=['week_end'])
print(temp_a)
```

```
print(temp_b)
#print(train_data["Expected_checkin"][6])
#print(train_data["Booking_date"][6])
print(a)
print(b)
print(stay_duration_val.head())
print(stay_duration_val.shape)
print(week_end_val.head())
print(week_end_val.shape)
print(reserve_duration_val.head())
print(reserve_duration_val.shape)

#for test data
week_end_test = []
stay_duration_test = []
reserve_duration_test = []
a = 0
temp_a = []
b = 0
temp_b = []
for i in range(len(test_data["Expected_checkin"])):
  checkin = test_data["Expected_checkin"][i]
  checkout = test_data["Expected_checkout"][i]
  reserve = test_data["Booking_date"][i]
  if days(reserve,checkin ) == 0:
    a+=1
    temp_a.append(i)
  if days(reserve,checkin ) < 0 :
    b+=1
    temp_b.append(i)
  stay_duration_test.append(days(checkin,checkout))
  reserve_duration_test.append(days(reserve,checkout ))
  week_end_test.append(weekend(checkin, checkout))

stay_duration_test = pd.DataFrame(stay_duration_test,
columns=['stay_duration'])
reserve_duration_test = pd.DataFrame(reserve_duration_test,
columns=['reserve_duration'])
week_end_test = pd.DataFrame(week_end_test, columns=['week_end'])
print(temp_a)
print(temp_b)
#print(train_data["Expected_checkin"][6])
#print(train_data["Booking_date"][6])
print(a)
print(b)
print(stay_duration_test.head())
print(stay_duration_test.shape)
print(week_end_test.head())
print(week_end_test.shape)
print(reserve_duration_test.head())
print(reserve_duration_test.shape)

#concat features

train_data=pd.concat([train_data,week_end_train],axis=1)
train_data=pd.concat([train_data,stay_duration],axis=1)
train_data=pd.concat([train_data,reserve_duration],axis=1)
```

```
val_data=pd.concat([val_data,week_end_val],axis=1)
val_data=pd.concat([val_data,stay_duration_val],axis=1)
val_data=pd.concat([val_data,reserve_duration_val],axis=1)

test_data=pd.concat([test_data,week_end_test],axis=1)
test_data=pd.concat([test_data,stay_duration_test],axis=1)
test_data=pd.concat([test_data,reserve_duration_test],axis=1)

print(train_data.shape)
print(val_data.shape)
print(test_data.shape)


"""##Features from previous visit and cancellation"""

# train data
trust_customer_train = []
not_trust_customer_train = []
for i in range(len(train_labels)):
  if train_data["Visted_Previously"][i] == "Yes" and
train_data["Previous_Cancellations"][i] == "Yes":
    not_trust_customer_train.append(1)
    trust_customer_train.append(0)
  elif train_data["Visted_Previously"][i] == "Yes" and
train_data["Previous_Cancellations"][i] == "No":
    not_trust_customer_train.append(0)
    trust_customer_train.append(1)
  else:
    not_trust_customer_train.append(0)
    trust_customer_train.append(0)

trust_customer_train = pd.DataFrame(trust_customer_train,
columns=['trust_customer'])
not_trust_customer_train = pd.DataFrame(not_trust_customer_train,
columns=['not_trust_customer'])


print(trust_customer_train.head())
print(trust_customer_train.shape)
print(not_trust_customer_train.head())
print(not_trust_customer_train.shape)

# val data
trust_customer_val = []
not_trust_customer_val = []
for i in range(len(val_labels)):
  if val_data["Visted_Previously"][i] == "Yes" and
val_data["Previous_Cancellations"][i] == "Yes":
    not_trust_customer_val.append(1)
    trust_customer_val.append(0)
  elif val_data["Visted_Previously"][i] == "Yes" and
val_data["Previous_Cancellations"][i] == "No":
    not_trust_customer_val.append(0)
    trust_customer_val.append(1)
  else:
    not_trust_customer_val.append(0)
```

```
    trust_customer_val.append(0)

trust_customer_val = pd.DataFrame(trust_customer_val,
columns=['trust_customer'])
not_trust_customer_val = pd.DataFrame(not_trust_customer_val,
columns=['not_trust_customer'])


print(trust_customer_val.head())
print(trust_customer_val.shape)
print(not_trust_customer_val.head())
print(not_trust_customer_val.shape)

# test data
trust_customer_test = []
not_trust_customer_test = []
for i in range(len(test_data["Visted_Previously"])):
  if test_data["Visted_Previously"][i] == "Yes" and
test_data["Previous_Cancellations"][i] == "Yes":
    not_trust_customer_test.append(1)
    trust_customer_test.append(0)
  elif test_data["Visted_Previously"][i] == "Yes" and
test_data["Previous_Cancellations"][i] == "No":
    not_trust_customer_test.append(0)
    trust_customer_test.append(1)
  else:
    not_trust_customer_test.append(0)
    trust_customer_test.append(0)

trust_customer_test = pd.DataFrame(trust_customer_test,
columns=['trust_customer'])
not_trust_customer_test = pd.DataFrame(not_trust_customer_test,
columns=['not_trust_customer'])


print(trust_customer_test.head())
print(trust_customer_test.shape)
print(not_trust_customer_test.head())
print(not_trust_customer_test.shape)

train_data=pd.concat([train_data,trust_customer_train],axis=1)
train_data=pd.concat([train_data,not_trust_customer_train],axis=1)

val_data=pd.concat([val_data,trust_customer_val],axis=1)
val_data=pd.concat([val_data,not_trust_customer_val],axis=1)

test_data=pd.concat([test_data,trust_customer_test],axis=1)
test_data=pd.concat([test_data,not_trust_customer_test],axis=1)

print(train_data.shape)
print(val_data.shape)
print(test_data.shape)

"""## Extracting Number of Rooms required and total cost"""

import math
num_rooms_train = []
```

```
total_cost_train = []
total_cost_dur_train = []

for i in range(len(train_labels)):
  rooms = math.ceil((train_data['Adults'][i] + train_data['Children'][i])/5)
  num_rooms_train.append(rooms)
  total_cost_train.append(rooms*train_data['Room_Rate'][i])

total_cost_dur_train.append(rooms*train_data['Room_Rate'][i]*train_data['stay
_duration'][i])

num_rooms_train = pd.DataFrame(num_rooms_train, columns=['num_rooms'])
total_cost_train = pd.DataFrame(total_cost_train,
columns=['tot_cost_per_day'])
total_cost_dur_train = pd.DataFrame(total_cost_dur_train,
columns=['tot_cost'])

print(train_data['stay_duration'].head())
print(train_data['Adults'].head())
print(train_data['Children'].head())

print(num_rooms_train.head())
print(num_rooms_train.shape)

print(total_cost_train.head())
print(total_cost_train.shape)

print(total_cost_dur_train.head())
print(total_cost_dur_train.shape)

num_rooms_val = []
total_cost_val = []
total_cost_dur_val = []

for i in range(len(val_labels)):
  rooms = math.ceil((val_data['Adults'][i] + val_data['Children'][i])/5)
  num_rooms_val.append(rooms)
  total_cost_val.append(rooms*val_data['Room_Rate'][i])

total_cost_dur_val.append(rooms*val_data['Room_Rate'][i]*val_data['stay_durat
ion'][i])

num_rooms_val = pd.DataFrame(num_rooms_val, columns=['num_rooms'])
total_cost_val = pd.DataFrame(total_cost_val, columns=['tot_cost_per_day'])
total_cost_dur_val = pd.DataFrame(total_cost_dur_val, columns=['tot_cost'])

print(val_data['stay_duration'].head())
print(val_data['Adults'].head())
print(val_data['Children'].head())

print(num_rooms_val.head())
print(num_rooms_val.shape)
print(total_cost_val.head())
print(total_cost_val.shape)
print(total_cost_dur_val.head())
print(total_cost_dur_val.shape)
```

```
num_rooms_test = []
total_cost_test = []
total_cost_dur_test = []

for i in range(len(test_data['Adults'])):
  rooms = math.ceil((test_data['Adults'][i] + test_data['Children'][i])/5)
  num_rooms_test.append(rooms)
  total_cost_test.append(rooms*test_data['Room_Rate'][i])

total_cost_dur_test.append(rooms*test_data['Room_Rate'][i]*test_data['stay_du
ration'][i])

num_rooms_test = pd.DataFrame(num_rooms_test, columns=['num_rooms'])
total_cost_test = pd.DataFrame(total_cost_test, columns=['tot_cost_per_day'])
total_cost_dur_test = pd.DataFrame(total_cost_dur_test, columns=['tot_cost'])

print(test_data['stay_duration'].head())
print(test_data['Adults'].head())
print(test_data['Children'].head())

print(num_rooms_test.head())
print(num_rooms_test.shape)
print(total_cost_test.head())
print(total_cost_test.shape)
print(total_cost_dur_test.head())
print(total_cost_dur_test.shape)

train_data=pd.concat([train_data,num_rooms_train],axis=1)
train_data=pd.concat([train_data,total_cost_train],axis=1)
train_data=pd.concat([train_data,total_cost_dur_train],axis=1)

val_data=pd.concat([val_data,num_rooms_val],axis=1)
val_data=pd.concat([val_data,total_cost_val],axis=1)
val_data=pd.concat([val_data,total_cost_dur_val],axis=1)

test_data=pd.concat([test_data,num_rooms_test],axis=1)
test_data=pd.concat([test_data,total_cost_test],axis=1)
test_data=pd.concat([test_data,total_cost_dur_test],axis=1)

print(train_data.shape)
print(val_data.shape)
print(test_data.shape)

"""##Imbalanced Data Handling

###READ previously saved data
"""

train_data = pd.read_csv('train_data_upsamp_3.csv')          # 3 is best
print(train_data.head())
print(train_data.shape)
train_labels = pd.read_csv('train_labels_upsamp_3.csv')
print(train_labels.head())
print(train_labels.shape)

"""###Upsampling"""
```

```
#Up Sampling   data.iloc [[3, 4], [1, 2]]
temp = train_labels.copy()
for i in range(len(temp)):
  if i%(len(temp)//10)==0:
    print('.',end='')
  if temp[i]==2:
    x = train_data.iloc[i,:]
    x1 = pd.Series(data={'Reservation_Status':2})

    train_data = train_data.append(x, ignore_index = True)
    train_data = train_data.append(x, ignore_index = True)
    train_data = train_data.append(x, ignore_index = True)
    train_data = train_data.append(x, ignore_index = True)


    train_labels = train_labels.append(x1, ignore_index = True)
    train_labels = train_labels.append(x1, ignore_index = True)
    train_labels = train_labels.append(x1, ignore_index = True)
    train_labels = train_labels.append(x1, ignore_index = True)


  if temp[i]==3:
    x=train_data.iloc[i,:]
    x1 = pd.Series(data={'Reservation_Status':3})
    train_data = train_data.append(x, ignore_index = True)
    train_data = train_data.append(x, ignore_index = True)
    train_data = train_data.append(x, ignore_index = True)
    train_data = train_data.append(x, ignore_index = True)
    train_data = train_data.append(x, ignore_index = True)
    train_data = train_data.append(x, ignore_index = True)
    train_data = train_data.append(x, ignore_index = True)
    train_data = train_data.append(x, ignore_index = True)

    train_labels = train_labels.append(x1, ignore_index = True)
    train_labels = train_labels.append(x1, ignore_index = True)
    train_labels = train_labels.append(x1, ignore_index = True)
    train_labels = train_labels.append(x1, ignore_index = True)
    train_labels = train_labels.append(x1, ignore_index = True)
    train_labels = train_labels.append(x1, ignore_index = True)
    train_labels = train_labels.append(x1, ignore_index = True)
    train_labels = train_labels.append(x1, ignore_index = True)
    #break

print(train_data.shape)
print(x)
print(train_data.iloc[-1,:])
print(train_labels.shape)
print(train_labels.iloc[-1])
print(x1)

#Save upsampled data
train_data.to_csv('train_data_upsamp_5.csv',index=False)
train_labels.to_csv('train_labels_upsamp_5.csv',index=False)

from imblearn.over_sampling import SMOTE , RandomOverSampler
from imblearn.under_sampling import EditedNearestNeighbours
sm = EditedNearestNeighbours()
```

```
train_columns = train_data.columns
#sm = SMOTE()
train_data, train_label = sm.fit_resample(train_data,train_label)
train_data = pd.DataFrame(train_data, columns = train_columns)

"""###Checking Ratios

"""

print("No of Check-In in training data : " +str((train_labels == 1).sum()))
print("No of Canceled in training data : " +str((train_labels == 2).sum()))
print("No of No-Show in training data : " +str((train_labels == 3).sum()))
tot=(train_labels == 1).sum()+(train_labels == 2).sum()+(train_labels ==
3).sum()
print("Ratio of Check-In : Canceled : No-Show  in training data = "
+str((train_labels == 1).sum()/tot)+' : '+str((train_labels ==
2).sum()/tot)+' : '+str((train_labels == 3).sum()/tot))


print("No of Check-In in validation data : " +str((val_labels == 1).sum()))
print("No of Canceled in validation data : " +str((val_labels == 2).sum()))
print("No of No-Show in validation data : " +str((val_labels == 3).sum()))
tot=(val_labels == 1).sum()+(val_labels == 2).sum()+(val_labels == 3).sum()
print("Ratio of Check-In : Canceled : No-Show  in Validation data = "
+str((val_labels == 1).sum()/tot)+' : '+str((val_labels == 2).sum()/tot)+' :
'+str((val_labels == 3).sum()/tot))

"""## Encoding Categorical data"""

print(train_data.columns)
print(len(train_data.columns))

print(val_data.columns)
print(len(val_data.columns))

print(test_data.columns)
print(len(test_data.columns))

#train data
gender_dummies=pd.get_dummies(train_data['Gender'],drop_first=True,
prefix='Gen')
gender=train_data.pop('Gender')
train_data=pd.concat([train_data,gender_dummies],axis=1)

eth_dummies=pd.get_dummies(train_data['Ethnicity'],drop_first=False,
prefix='Eth')
eth=train_data.pop('Ethnicity')
train_data=pd.concat([train_data,eth_dummies],axis=1)

edu_dummies=pd.get_dummies(train_data['Educational_Level'],drop_first=False,
prefix='Edu')
edu=train_data.pop('Educational_Level')
train_data=pd.concat([train_data,edu_dummies],axis=1)

in_dummies=pd.get_dummies(train_data['Income'],drop_first=False, prefix='In')
in_dummies=in_dummies.rename(columns={'In_25K --50K': 'In_25K_50K', 'In_50K -
- 100K': 'In_50K_100K', 'In_<25K':'In_below25K', 'In_>100K':'In_above100K' })
```

```
income=train_data.pop('Income')
train_data=pd.concat([train_data,in_dummies],axis=1)

region_dummies=pd.get_dummies(train_data['Country_region'],drop_first=False,
prefix='Coun')
region=train_data.pop('Country_region')
train_data=pd.concat([train_data,region_dummies],axis=1)

hotel_dummies=pd.get_dummies(train_data['Hotel_Type'],drop_first=False,
prefix='Hotel')
hotel=train_data.pop('Hotel_Type')
train_data=pd.concat([train_data,hotel_dummies],axis=1)

meal_dummies=pd.get_dummies(train_data['Meal_Type'],drop_first=False,
prefix='Meal')
meal=train_data.pop('Meal_Type')
train_data=pd.concat([train_data,meal_dummies],axis=1)

visit_prev_dummies=pd.get_dummies(train_data['Visted_Previously'],drop_first=
True, prefix='Visit')
visit_prev=train_data.pop('Visted_Previously')
train_data=pd.concat([train_data,visit_prev_dummies],axis=1)

prev_can_dummies=pd.get_dummies(train_data['Previous_Cancellations'],drop_fir
st=True, prefix='Prev_can')
prev_can=train_data.pop('Previous_Cancellations')
train_data=pd.concat([train_data,prev_can_dummies],axis=1)

dep_dummies=pd.get_dummies(train_data['Deposit_type'],drop_first=False,
prefix='Dep')
dep=train_data.pop('Deposit_type')
train_data=pd.concat([train_data,dep_dummies],axis=1)

book_dummies=pd.get_dummies(train_data['Booking_channel'],drop_first=False,
prefix='Book')
book=train_data.pop('Booking_channel')
train_data=pd.concat([train_data,book_dummies],axis=1)

car_dummies=pd.get_dummies(train_data['Required_Car_Parking'],drop_first=True
, prefix='Car')
car=train_data.pop('Required_Car_Parking')
train_data=pd.concat([train_data,car_dummies],axis=1)

promo_dummies=pd.get_dummies(train_data['Use_Promotion'],drop_first=True,
prefix='Promo')
promo=train_data.pop('Use_Promotion')
train_data=pd.concat([train_data,promo_dummies],axis=1)



print(train_data.columns)
print(train_data.head())

#Val data
gender_dummies=pd.get_dummies(val_data['Gender'],drop_first=True,
prefix='Gen')
gender=val_data.pop('Gender')
```

```
val_data=pd.concat([val_data,gender_dummies],axis=1)

eth_dummies=pd.get_dummies(val_data['Ethnicity'],drop_first=False,
prefix='Eth')
eth=val_data.pop('Ethnicity')
val_data=pd.concat([val_data,eth_dummies],axis=1)

edu_dummies=pd.get_dummies(val_data['Educational_Level'],drop_first=False,
prefix='Edu')
edu=val_data.pop('Educational_Level')
val_data=pd.concat([val_data,edu_dummies],axis=1)

in_dummies=pd.get_dummies(val_data['Income'],drop_first=False, prefix='In')
in_dummies=in_dummies.rename(columns={'In_25K --50K': 'In_25K_50K', 'In_50K -
- 100K': 'In_50K_100K', 'In_<25K':'In_below25K', 'In_>100K':'In_above100K' })
income=val_data.pop('Income')
val_data=pd.concat([val_data,in_dummies],axis=1)

region_dummies=pd.get_dummies(val_data['Country_region'],drop_first=False,
prefix='Coun')
region=val_data.pop('Country_region')
val_data=pd.concat([val_data,region_dummies],axis=1)

hotel_dummies=pd.get_dummies(val_data['Hotel_Type'],drop_first=False,
prefix='Hotel')
hotel=val_data.pop('Hotel_Type')
val_data=pd.concat([val_data,hotel_dummies],axis=1)

meal_dummies=pd.get_dummies(val_data['Meal_Type'],drop_first=False,
prefix='Meal')
meal=val_data.pop('Meal_Type')
val_data=pd.concat([val_data,meal_dummies],axis=1)

visit_prev_dummies=pd.get_dummies(val_data['Visted_Previously'],drop_first=Tr
ue, prefix='Visit')
visit_prev=val_data.pop('Visted_Previously')
val_data=pd.concat([val_data,visit_prev_dummies],axis=1)

prev_can_dummies=pd.get_dummies(val_data['Previous_Cancellations'],drop_first
=True, prefix='Prev_can')
prev_can=val_data.pop('Previous_Cancellations')
val_data=pd.concat([val_data,prev_can_dummies],axis=1)

dep_dummies=pd.get_dummies(val_data['Deposit_type'],drop_first=False,
prefix='Dep')
dep=val_data.pop('Deposit_type')
val_data=pd.concat([val_data,dep_dummies],axis=1)

book_dummies=pd.get_dummies(val_data['Booking_channel'],drop_first=False,
prefix='Book')
book=val_data.pop('Booking_channel')
val_data=pd.concat([val_data,book_dummies],axis=1)

car_dummies=pd.get_dummies(val_data['Required_Car_Parking'],drop_first=True,
prefix='Car')
car=val_data.pop('Required_Car_Parking')
val_data=pd.concat([val_data,car_dummies],axis=1)
```

```
promo_dummies=pd.get_dummies(val_data['Use_Promotion'],drop_first=True,
prefix='Promo')
promo=val_data.pop('Use_Promotion')
val_data=pd.concat([val_data,promo_dummies],axis=1)



print(val_data.columns)
print(val_data.head())

#test data
gender_dummies=pd.get_dummies(test_data['Gender'],drop_first=True,
prefix='Gen')
gender=test_data.pop('Gender')
test_data=pd.concat([test_data,gender_dummies],axis=1)

eth_dummies=pd.get_dummies(test_data['Ethnicity'],drop_first=False,
prefix='Eth')
eth=test_data.pop('Ethnicity')
test_data=pd.concat([test_data,eth_dummies],axis=1)

edu_dummies=pd.get_dummies(test_data['Educational_Level'],drop_first=False,
prefix='Edu')
edu=test_data.pop('Educational_Level')
test_data=pd.concat([test_data,edu_dummies],axis=1)

in_dummies=pd.get_dummies(test_data['Income'],drop_first=False, prefix='In')
in_dummies=in_dummies.rename(columns={'In_25K --50K': 'In_25K_50K', 'In_50K -
- 100K': 'In_50K_100K', 'In_<25K':'In_below25K', 'In_>100K':'In_above100K' })
income=test_data.pop('Income')
test_data=pd.concat([test_data,in_dummies],axis=1)

region_dummies=pd.get_dummies(test_data['Country_region'],drop_first=False,
prefix='Coun')
region=test_data.pop('Country_region')
test_data=pd.concat([test_data,region_dummies],axis=1)

hotel_dummies=pd.get_dummies(test_data['Hotel_Type'],drop_first=False,
prefix='Hotel')
hotel=test_data.pop('Hotel_Type')
test_data=pd.concat([test_data,hotel_dummies],axis=1)

meal_dummies=pd.get_dummies(test_data['Meal_Type'],drop_first=False,
prefix='Meal')
meal=test_data.pop('Meal_Type')
test_data=pd.concat([test_data,meal_dummies],axis=1)

visit_prev_dummies=pd.get_dummies(test_data['Visted_Previously'],drop_first=T
rue, prefix='Visit')
visit_prev=test_data.pop('Visted_Previously')
test_data=pd.concat([test_data,visit_prev_dummies],axis=1)

prev_can_dummies=pd.get_dummies(test_data['Previous_Cancellations'],drop_firs
t=True, prefix='Prev_can')
prev_can=test_data.pop('Previous_Cancellations')
test_data=pd.concat([test_data,prev_can_dummies],axis=1)
```

```
dep_dummies=pd.get_dummies(test_data['Deposit_type'],drop_first=False,
prefix='Dep')
dep=test_data.pop('Deposit_type')
test_data=pd.concat([test_data,dep_dummies],axis=1)

book_dummies=pd.get_dummies(test_data['Booking_channel'],drop_first=False,
prefix='Book')
book=test_data.pop('Booking_channel')
test_data=pd.concat([test_data,book_dummies],axis=1)

car_dummies=pd.get_dummies(test_data['Required_Car_Parking'],drop_first=True,
prefix='Car')
car=test_data.pop('Required_Car_Parking')
test_data=pd.concat([test_data,car_dummies],axis=1)

promo_dummies=pd.get_dummies(test_data['Use_Promotion'],drop_first=True,
prefix='Promo')
promo=test_data.pop('Use_Promotion')
test_data=pd.concat([test_data,promo_dummies],axis=1)



print(test_data.columns)
print(test_data.head())

for i in train_data.columns:
  if i not in test_data.columns:
    print(i)

"""##Label Encoding

"""

from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(train_labels)
print(le.classes_)
train_label=le.transform(train_labels)

le = preprocessing.LabelEncoder()
le.fit(val_labels)
print(le.classes_)
val_label=le.transform(val_labels)

"""##Remove unnecessary columns"""

train_data.pop('Reservation-id')
val_data.pop('Reservation-id')
test_reservation=test_data['Reservation-id']
test_data.pop('Reservation-id')

train_data.pop('Expected_checkin')
val_data.pop('Expected_checkin')
test_data.pop('Expected_checkin')

train_data.pop('Expected_checkout')
```

```
val_data.pop('Expected_checkout')
test_data.pop('Expected_checkout')

train_data.pop('Booking_date')
val_data.pop('Booking_date')
test_data.pop('Booking_date')

"""#Neptune Create Experiment"""

neptune.create_experiment(name = 'Experiment without dates data and testing
on multiple models - upsampled to 0.333 ratio - All features with new 3
features')

"""#Model Building"""

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score,
roc_auc_score, classification_report, plot_confusion_matrix, precision_score,
recall_score
from sklearn import tree, svm
from sklearn.ensemble import RandomForestClassifier
import xgboost
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import NearestNeighbors,KNeighborsClassifier

#train_data=train_data[['Age','Adults','Discount_Rate','Room_Rate','week_end'
,'stay_duration','reserve_duration','Eth_African
American','Eth_caucasian','Coun_East','Hotel_Airport
Hotels','Meal_BB','Meal_FB','Visit_Yes','Prev_can_Yes','Dep_No Deposit']]
#val_data=val_data[['Age','Adults','Discount_Rate','Room_Rate','week_end','st
ay_duration','reserve_duration','Eth_African
American','Eth_caucasian','Coun_East','Hotel_Airport
Hotels','Meal_BB','Meal_FB','Visit_Yes','Prev_can_Yes','Dep_No Deposit']]
#test_data=test_data[['Age','Adults','Discount_Rate','Room_Rate','week_end','
stay_duration','reserve_duration','Eth_African
American','Eth_caucasian','Coun_East','Hotel_Airport
Hotels','Meal_BB','Meal_FB','Visit_Yes','Prev_can_Yes','Dep_No Deposit']]

#train_data = train_data[selected_features]
#val_data = val_data[selected_features]
#test_data = test_data[selected_features]

print(train_data.shape)
print(val_data.shape)
print(test_data.shape)

"""##logistic regression approach"""

model= LogisticRegression(multi_class='multinomial',
solver='saga',max_iter=100)
model.fit(train_data,train_label)
y_predict=model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro',
zero_division=0)))
```

```
print("Recall : "+str(recall_score(val_label,y_predict,average='macro',
zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='
macro', zero_division=0))
#neptune.log_metric('Recall',
recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro',
zero_division=0))

"""##Decision Tree Classifier model approach"""

model = DecisionTreeClassifier(max_depth=20, class_weight = 'balanced'
).fit(train_data,train_label)
y_predict=model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro',
zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='
macro', zero_division=0))
#neptune.log_metric('Recall',
recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro',
zero_division=0))

#Feature Importance in Decision Tree Classifier
print("Feature Importance")
print(model.feature_importances_) #use inbuilt class feature_importances of
tree based classifiers
#plot graph of feature importances for better visualization
plt.figure(figsize=[10,10])
```

```
feat_importances = pd.Series(model.feature_importances_,
index=train_data.columns)
feat_importances.nlargest(40).plot(kind='barh')

plt.show()
#print(feat_importances)
results=pd.DataFrame()
results['columns']=train_data.columns
results['importances'] = model.feature_importances_
results.sort_values(by='importances',ascending=False,inplace=True)

results[:20]

print(results['columns'][:20].tolist())

['Age','Adults','Discount_Rate','Room_Rate','week_end','stay_duration','reser
ve_duration','Eth_African
American','Eth_caucasian','Coun_East','Hotel_Airport
Hotels','Meal_BB','Meal_FB','Visit_Yes','Prev_can_Yes','Dep_No Deposit']

"""##XGB Boost Approach"""

clf = DecisionTreeClassifier(max_depth=50, class_weight = 'balanced')
#model=xgboost.XGBClassifier(base_estimator=clf,max_depth=20,n_estimators=15,
objective='multi:softmax',gamma=4.63,learning_rate=0.2,reg_lambda=1).fit(trai
n_data,train_label) # day 2 second submission model
model=xgboost.XGBClassifier(base_estimator = clf, max_depth = 22,
n_estimators = 15, objective = 'multi:softmax', gamma = 4.5, learning_rate =
0.05, reg_lambda = 3.4).fit(train_data,train_label)    #hypertuned model
y_predict=model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro',
zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='
macro', zero_division=0))
#neptune.log_metric('Recall',
recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro',
zero_division=0))

# Plot non-normalized confusion matrix
titles_options = [("Confusion matrix, without normalization", None),
```

```
                  ("Normalized confusion matrix", 'true')]
class_names = ["Check-In", "Canceled", "No-Show"]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(model, val_data, val_label,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)

#Feature Importance in XGBoost
print("Feature Importance")
print(model.feature_importances_) #use inbuilt class feature_importances of
tree based classifiers
#plot graph of feature importances for better visualization
plt.figure(figsize=[10,10])
feat_importances = pd.Series(model.feature_importances_,
index=train_data.columns)
feat_importances.nlargest(40).plot(kind='barh')

plt.show()
#print(feat_importances)
results=pd.DataFrame()
results['columns']=train_data.columns
results['importances'] = model.feature_importances_
results.sort_values(by='importances',ascending=False,inplace=True)

results[:20]
selected_features = results['columns'][:20].tolist()
print(selected_features)

"""##Support Vector Machine Approach"""

model = svm.SVC(degree=9,decision_function_shape='ovo', class_weight =
'balanced')
model.fit(train_data,train_label)
y_predict=model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro',
zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
```

```
neptune.log_metric('Precision',precision_score(val_label,y_predict,average='m
acro', zero_division=0))
neptune.log_metric('Recall',
recall_score(val_label,y_predict,average='macro', zero_division=0))
neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro',
zero_division=0))

"""##MLP classifier approach"""

from sklearn.neural_network import MLPClassifier
model = MLPClassifier(solver='adam',learning_rate =
'adaptive',learning_rate_init=0.01,activation= 'relu', alpha=1e-6,
hidden_layer_sizes=(150, ), random_state=91,max_iter=400)
model.fit(train_data,train_label)
y_predict=model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro',
zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='
macro', zero_division=0))
#neptune.log_metric('Recall',
recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro',
zero_division=0))

"""##Random Forest approach"""

model = RandomForestClassifier(max_depth=12,n_estimators=75, class_weight =
'balanced' )
model.fit(train_data,train_label)
y_predict=model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro',
zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
```

```
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='
macro', zero_division=0))
#neptune.log_metric('Recall',
recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro',
zero_division=0))

"""##KNN approach"""

model=KNeighborsClassifier(n_neighbors=3,algorithm='auto',weights='distance')
model.fit(train_data,train_label)
y_predict=model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro',
zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='
macro', zero_division=0))
#neptune.log_metric('Recall',
recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro',
zero_division=0))

# Plot non-normalized confusion matrix
titles_options = [("Confusion matrix, without normalization", None),
                  ("Normalized confusion matrix", 'true')]
class_names = ["Check-In", "Canceled", "No-Show"]
for title, normalize in titles_options:
    disp = plot_confusion_matrix(model, val_data, val_label,
                                 display_labels=class_names,
                                 cmap=plt.cm.Blues,
                                 normalize=normalize)
    disp.ax_.set_title(title)

    print(title)
    print(disp.confusion_matrix)
```

```
"""##Ensemble - extra tree classifier approach"""

from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier(max_depth=12,n_estimators=100, class_weight =
'balanced')
model.fit(train_data, train_label)
y_predict= model.predict(val_data)
print("Train accuracy : "+str(model.score(train_data,train_label)))
print("Validation accuracy : "+str(model.score(val_data,val_label)))
print("Precision : "+str(precision_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Recall : "+str(recall_score(val_label,y_predict,average='macro',
zero_division=0)))
print("F1-Score : "+str(f1_score(val_label,y_predict,average='macro',
zero_division=0)))
print("Classification Report")
print(classification_report(val_label,y_predict,zero_division=0))
print("Confusion Matrix")
print(confusion_matrix(val_label,y_predict))
#fig, ax = plt.subplots()
#plot_confusion_matrix(val_label, y_predict, ax=ax)


#neptune.log_metric('Training Accuracy', model.score(train_data,train_label))
#neptune.log_metric('Validation Accuracy', model.score(val_data,val_label))
#neptune.log_metric('Precision',precision_score(val_label,y_predict,average='
macro', zero_division=0))
#neptune.log_metric('Recall',
recall_score(val_label,y_predict,average='macro', zero_division=0))
#neptune.log_metric('F1-Score',f1_score(val_label,y_predict,average='macro',
zero_division=0))

"""##Hypertuning the model"""

#Hypertuning Parameters for Accuracy F1score and AUC score
#max_depth
#learning_rate
#min_child_weight
#gamma 4.63
#colsample_bytree
#scale_pos_weight
#subsample
#reg_lambda
x=np.linspace(0.1,5,num=50,dtype=float)
train_acc = []
val_acc = []
F = []
clf = DecisionTreeClassifier(max_depth=50, class_weight = 'balanced')
for i in x:
     print(i)
     model=xgboost.XGBClassifier(base_estimator = clf, max_depth = 22,
n_estimators = 15, objective = 'multi:softmax', gamma = 4.5, learning_rate =
0.05, reg_lambda = 3.4).fit(train_data,train_label)
     y_pred= model.predict(val_data)
     f=f1_score(val_label,y_pred ,average='macro', zero_division=0)
     F.append(f)
```

```
      #auc=roc_auc_score(val_label,y_pred,average='macro')
      #AUC.append(auc)
      train_acc.append(model.score(train_data,train_label))
      val_acc.append(model.score(val_data,val_label))
      if f > 0.3664:
        print("improvement")
 #ploting hypertuning results

import matplotlib.pyplot as plt
#plt.plot(x,AUC)
#plt.title('AUC Score')
#plt.ylabel('Auc')
#plt.xlabel('Parameters')
#plt.show()
plt.plot(x,F)
plt.title('F1 Score')
plt.ylabel('F1')
plt.xlabel('Parameters')
plt.show()
plt.plot(x,train_acc)
plt.title('Train accuracy')
plt.ylabel('Acc')
plt.xlabel('Parameters')
plt.show()
plt.plot(x,val_acc)
plt.title('Validation Accuracy')
plt.ylabel('Acc')
plt.xlabel('Parameters')
plt.show()

print("Maximum Training Acc : "+str(max(train_acc)))
print(x[train_acc.index(max(train_acc))])

print("Maximum Validation Acc : "+str(max(val_acc)))
print(x[val_acc.index(max(val_acc))])

print("Maximum F1 Score : "+str(max(F)))
print(x[F.index(max(F))])

print(x[F.index(max(F))])

"""##Neptune log order of Models"""

model_list=['logistic regression','Decision Tree Classifier', 'XGB
Boost','MLP classifier', 'Random Forest', 'KNN', 'Ensemble - extra tree
classifier'  ]
for i in model_list:
  neptune.log_text('Model order',i)

neptune.stop()

"""#Prediction For submission"""

y_predict_2= model.predict(test_data)
y_predict_2

y_predict_2=le.inverse_transform(y_predict_2)
```

```
y_predict_2

y_predict_2=pd.DataFrame(y_predict_2,columns=['Reservation_status'] )
y_predict_2

test_reservation=pd.DataFrame(test_reservation)
test_reservation

test_reservation=pd.concat([test_reservation,y_predict_2],axis=1)
test_reservation

#test_reservation.to_csv('submission_XGBoost_upsampled_0.33_0.33_0.33_hypertu
ned_0.3694.csv',index=False)
```