REPORT

(KAGGLE FIRST ROUND COMPETITION)

WANDERERS (DS21-68)

Highest Kaggle Submission Score: 0.33468

Team members : Jathurshan Pradeepkumar (Leader)

Mithunjha Anandakumar

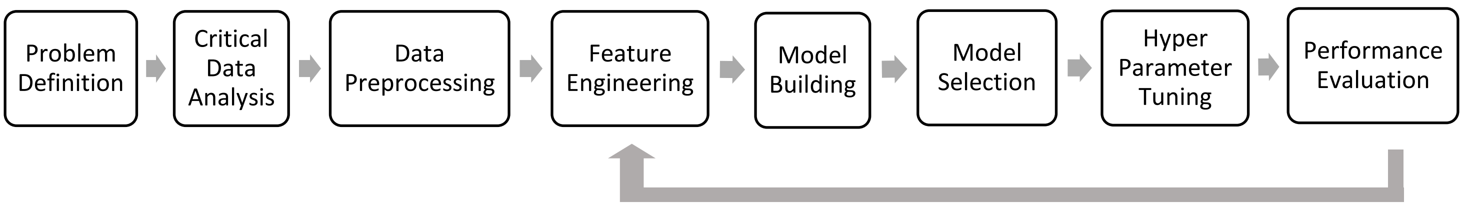
Vinith Kugathasan

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

Introduction

In recent years, the cancellation rate for booking has become quite high. By predicting the possibilities of a traveler cancelling or not showing in advance, hotels can develop interventions to minimize these circumstances & loss. The task given is to predict whether a given traveler will cancel or not show for their booking reservation. For this purpose, 15 months of booking reservations of 3 hotel types (City hotels, Airport Hotel, Resorts) of Hotel chain A are provided. The developed model should predict '1' if the traveler would check-in, '2' if the traveler would cancel, and '3' if the travel would not show. This is a multi-class classification problem. This prediction is very important for hotels because option to cancel the service puts the risk on the hotel, as the hotel has to guarantee rooms to customers to honor their bookings but, at the same time has to bear with the opportunity cost of vacant capacity when a customer cancels a booking or does not show up.

Approaches



We started with a brief literature review on this business problem, because good background knowledge and understanding on the business problem is important to tackle it efficiently. After the literature review, data analysis was conducted by doing data processing and visualization to understand the given dataset collected by Hotel Chain A over 15 months of data from their three hotel types. We were able to get a good understanding of the features in the dataset through data analysis. Brainstorming session was conducted to identify best features and new features which can be extracted from the data that would have more effect on the decision of the traveler. The problem was tackled by using feedback mechanism consisting of data analysis, data preprocessing, feature engineering, Model building, model selection, hyperparameter tuning and performance evaluation.

The dataset consists of three classes which are quantitatively imbalanced. Prediction of cancellation and no-show classes is important to the business, so that they can take necessary steps to overcome potential revenue loss. Macro F1 Score was considered as the main evaluation metric for the classification model, because to identify the ability of the model to predict cancellation and no-show. Along with Macro-F1 Score, several other metrics (recall, precision, confusion matrix, validation accuracy) were also used to evaluate the performance of the model.

Tools utilized.

Google Colaboratory was utilized to build our experiments and it was selected because it is easy to share, and it provides GPU and TPU features. The following libraries were utilized in our approach:

- Pandas
- NumPy

- Matplotlib
- Seaborn
- Imblearn
- Sklearn
- TensorFlow
- Keras

Tracking the results of all conducted experiments is hard and time consuming. We utilized Neptune.ai to store and maintain all the necessary performance metrics of experiments to track the experiments efficiently.

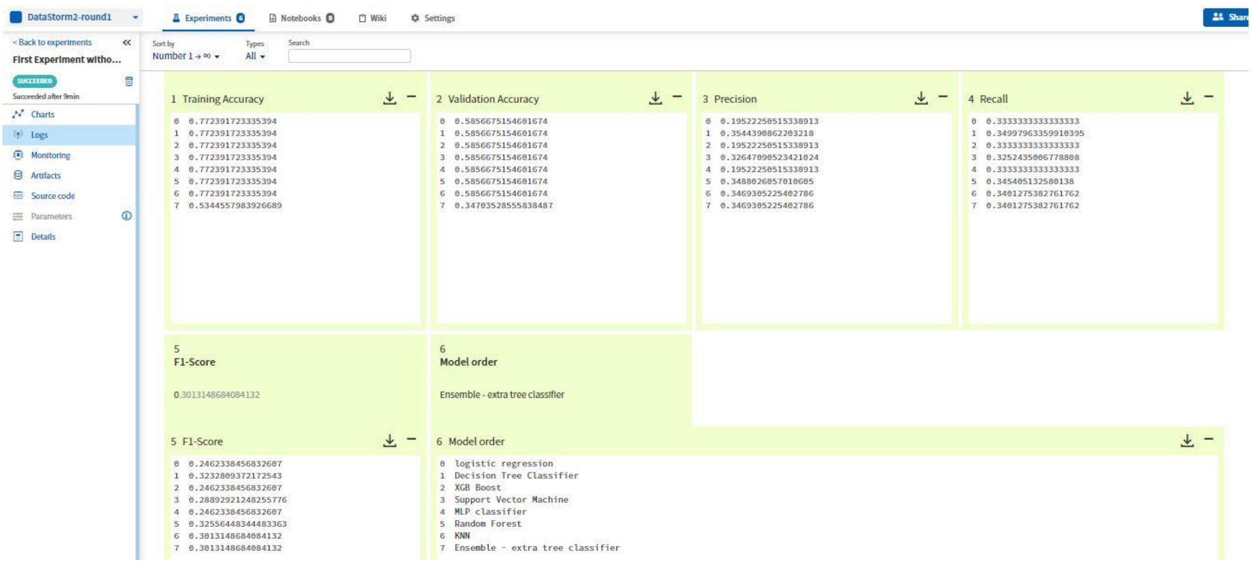


FIGURE 1 : NEPTUNE.AI FOR EXPERIMENT TRACKING

Data Preprocessing

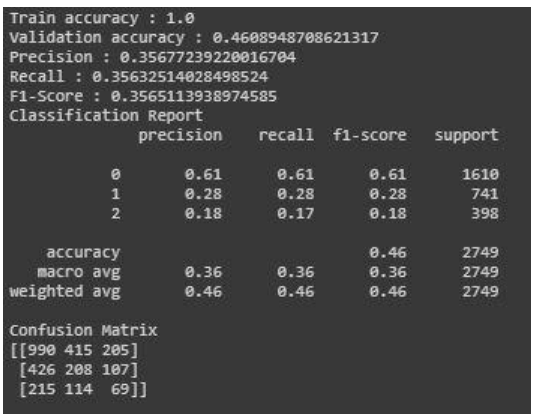
- First, we checked the dataset for N/A values using (isna()) but found none.
- Then the Nominal data fields in the string format such as gender, ethnicity, educational level, income, country region, hotel type, meal type, visited previously, previous cancellation, deposit type, booking channel, required car parking and Use promotion were converted into binary variable using the one hot encoding.
- The dataset provided has imbalance issue and it was dealt by upsampling the minority classes.

Feature Engineering

Through our literature review and data analysis, we identified several factors that can cause the booking cancellations, such as booking method, income level, meals, room rate etc. Knowledge through literature review was utilized to identify suitable features from the dataset to predict cancellation or no show.

- Initially we processed all the features and converted them to the suitable format to feed to the model. In our first experiment we excluded features in date format, which are Expected_checkin, Expected_checkout and Booking_date. Then we used all the other features and trained several models to select the best model with the best F1 score and accuracy. However, the F1 score was in the range of 0.24 – 0.3 because of imbalanced data.

- Several imbalanced data handling techniques such as oversampling (Random Over Sampling, SMOTE, ADASYN) and under sampling (Random Under Sampling, Near-Miss, Edited Nearest Neighbors) were experimented. Along with the available techniques, an algorithm was implemented to duplicate cancellation and no-show classes to create a balanced dataset. Balanced dataset had better performance with respect to F1-Score in most of the trained models. K Nearest Neighbor classifier had the best F1-Score compared to other models and its prediction on test data was submitted **(Submission Score: 0.33335).**



```
Train accuracy : 1.0
Validation accuracy : 0.4608948708621317
Precision : 0.35677239220016704
Recall : 0.35632514028498524
F1-Score : 0.3565113938974585
Classification Report
              precision    recall  f1-score   support

           0       0.61      0.61      0.61      1610
           1       0.28      0.28      0.28       741
           2       0.18      0.17      0.18       398

    accuracy                           0.46      2749
   macro avg       0.36      0.36      0.36      2749
weighted avg       0.46      0.46      0.46      2749

Confusion Matrix
[[990 415 205]
 [426 208 107]
 [215 114  69]]
```

FIGURE 3 : PERFORMANCE OF KNN WITH UP SAMPLING & WITHOUT DATE FEATURES (KAGGLE SCORE: 0.3335)

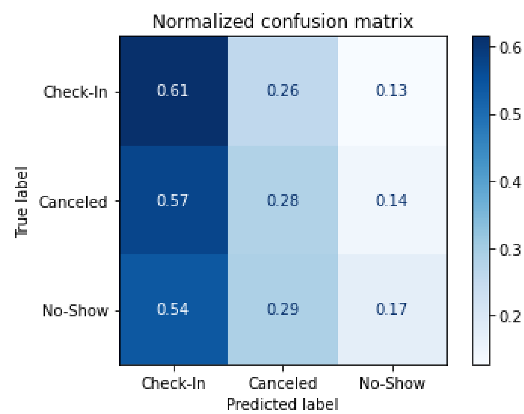
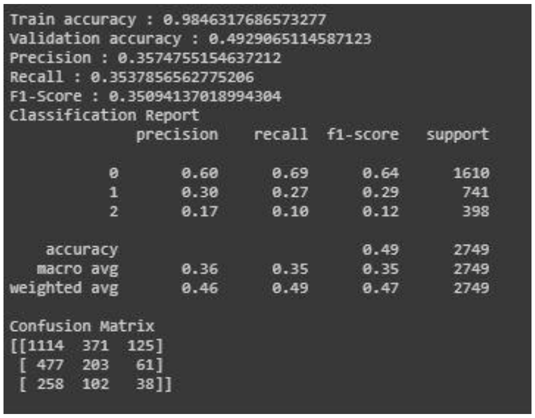


FIGURE 2 : NORMALIZED CONFUSION MATRIX OF KNN CLASSIFIER

- We identified that most of the features are not making a big effect on the prediction. Then we removed features with low correlation and feature importance. We also created new features combing some of the available features.
- In our third experiment we extracted new features from Expected_checkin, Expected_checkout and Booking_date. The Features we extracted are as follows:
    - **week_end:** Whether the expected stay of the traveler includes weekend or not (1 or 0). This feature is extracted using Expected_checkin and Expected_checkout features.
    - **stay_duration:** Number of days the traveler expected to stay, which is extracted using Expected_checkin and Expected_checkout.
    - **reserve_duration:** Number of days between Booking_date and Expected_checkin. This feature includes negative values as well because some bookings were done after the expected check-in date, in training, validation and test dataset.

- Sixteen features were selected from the pool of features including newly extracted features based on the feature importance from the decision tree classifier, and several models were trained to select the best model with the best F1 score and validation accuracy. XGBoost classifier had the best F1-Score compared to other models and its prediction on test data was submitted **(Submission Score: 0.33028).**

- XGBoost classifier was hyper tuned to improve its performance on validation accuracy and F1-score. The prediction on test data using above classifier was submitted, which gave the best results **(Best Kaggle Score: 0.33468).** Figures 5 and 7 indicate that the prediction of cancellation and no-show classes have increased compared to the previous methods.



```
Train accuracy : 0.9846317686573277
Validation accuracy : 0.4929065114587123
Precision : 0.3574755154637212
Recall : 0.3537856562775206
F1-Score : 0.35094137018994304
Classification Report
              precision    recall  f1-score   support

           0       0.60      0.69      0.64      1610
           1       0.30      0.27      0.29       741
           2       0.17      0.10      0.12       398

    accuracy                           0.49      2749
   macro avg       0.36      0.35      0.35      2749
weighted avg       0.46      0.49      0.47      2749

Confusion Matrix
[[1114  371  125]
 [ 477  203   61]
 [ 258  102   38]]
```

FIGURE 4 : PERFORMANCE OF XGBOOST CLASSIFIER WITH UP SAMPLING & WITH SELECTED 16 FEATURES INCLUDING NEW FEATURES. (KAGGLE SCORE : 0.33028)

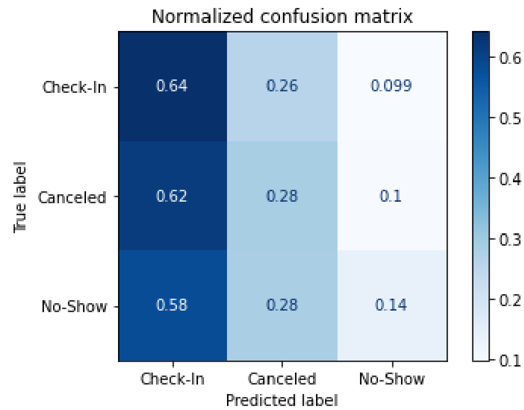
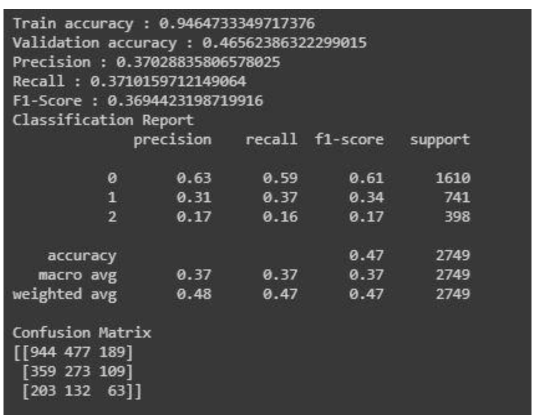


FIGURE 5 : NORMALIZED CONFUSION MATRIX OF XGBOOST CLASSIFIER



```
Train accuracy : 0.9464733349717376
Validation accuracy : 0.46562386322299015
Precision : 0.37028835806578025
Recall : 0.3710159712149064
F1-Score : 0.3694423198719916
Classification Report
              precision    recall  f1-score   support

           0       0.63      0.59      0.61      1610
           1       0.31      0.37      0.34       741
           2       0.17      0.16      0.17       398

    accuracy                           0.47      2749
   macro avg       0.37      0.37      0.37      2749
weighted avg       0.48      0.47      0.47      2749

Confusion Matrix
[[944 477 189]
 [359 273 109]
 [203 132  63]]
```

FIGURE 6 : PERFORMANCE OF HYPER TUNED XGBOOST CLASSIFIER WITH UP SAMPLING & WITH SELECTED 16 FEATURES INCLUDING NEW FEATURES.
(BEST KAGGLE SCORE : 0.33468)

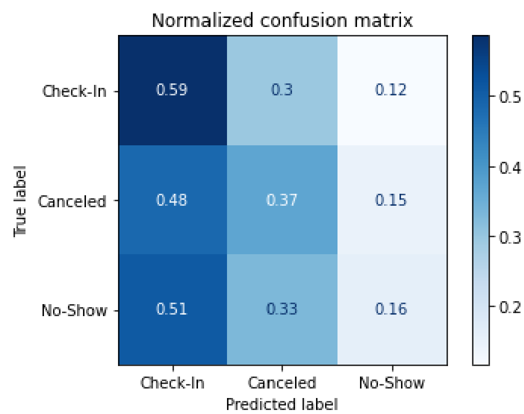
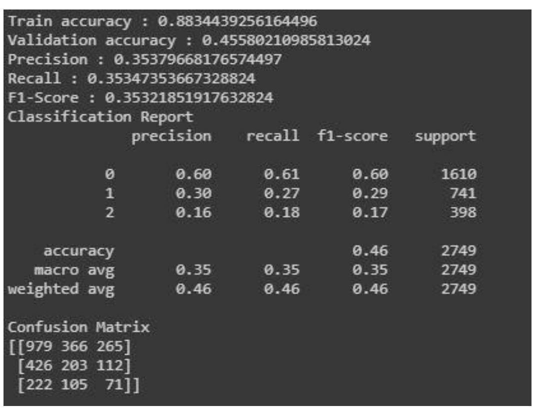


FIGURE 7 : NORMALIZED CONFUSION MATRIX OF HYPER TUNED XGBOOST CLASSIFIER

- In our next experiment, additional new features were extracted from the dataset. Visted_Previously and Previous_Cancellations features were utilized to identify trusted travelers in the dataset. Adults, Children, Room_Rate and stay_duration features were used to extract number of rooms required and total cost features. Extracted new features are as follows:
  - **trust_customer:** Whether the traveler has previously checked in for a reservation or not (Boolean).

- **not_trust_customer:** Whether the traveler has previously cancelled a reservation or not (Boolean). If above both features are False it indicates that the traveler has never made a reservation before.
- **num_rooms:** The number of rooms required by the traveler considering that one room can accommodate up to 5 people including children (Except babies). This feature was extracted using number of adults and children.
- **tot_cost_per_day:** The total cost for a day in the hotel for the traveler and it was calculated using the number of rooms required and room rate.
- **tot_cost:** The total cost for the stay in the hotel for the traveler and it was calculated using the number of rooms required, room rate and stay duration.

- Multiple models were trained using all the extracted features and the best model was selected based on their F1 score, validation accuracy and confusion matrix. Decision tree classifier had the best performance on this dataset and its parameters were hyper tuned to improve its performance. The prediction of the Decision tree classifier on test data was submitted and its results were much closer to our best submission score **(Submission Score: 0.33415).**
- Best 28 features were selected using feature importance and heat map and, were used to train and hyper tune Decision tree classifier. This model gave the best result among all the models in terms of Macro F1-score, validation accuracy and better prediction of cancellation and no-class in validation data. However, its submission score was not the best **(Submission Score: 0.32727).**



```
Train accuracy : 0.8834439256164496
Validation accuracy : 0.45580210985813024
Precision : 0.35379668176574497
Recall : 0.35347353667328824
F1-Score : 0.35321851917632824
Classification Report
              precision    recall  f1-score   support

           0       0.60      0.61      0.60      1610
           1       0.30      0.27      0.29       741
           2       0.16      0.18      0.17       398

    accuracy                           0.46      2749
   macro avg       0.35      0.35      0.35      2749
weighted avg       0.46      0.46      0.46      2749

Confusion Matrix
[[979 366 265]
 [426 203 112]
 [222 105  71]]
```

FIGURE 8 : PERFORMANCE OF HYPER TUNED DECISION TREE CLASSIFIER WITH UP SAMPLING & WITH ALL FEATURES INCLUDING NEW FEATURES.
(KAGGLE SCORE : 0.33415)

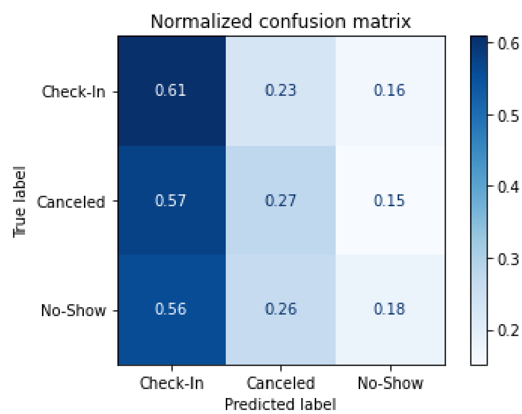
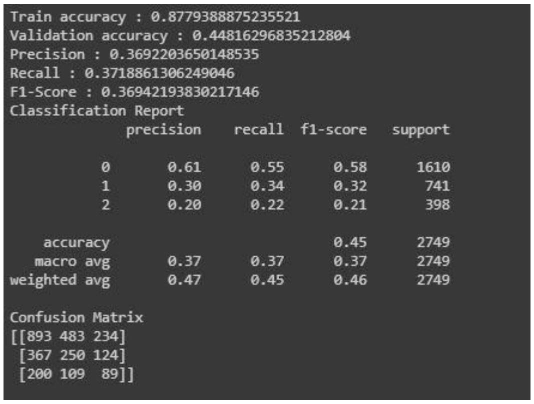


FIGURE 9 : NORMALIZED CONFUSION MATRIX OF HYPER TUNED DECISION TREE CLASSIFIER WITH ALL FEATURES.



```
Train accuracy : 0.8779388875235521
Validation accuracy : 0.44816296835212804
Precision : 0.3692203650148535
Recall : 0.3718861306249046
F1-Score : 0.36942193830217146
Classification Report
              precision    recall  f1-score   support

           0       0.61      0.55      0.58      1610
           1       0.30      0.34      0.32       741
           2       0.20      0.22      0.21       398

    accuracy                           0.45      2749
   macro avg       0.37      0.37      0.37      2749
weighted avg       0.47      0.45      0.46      2749

Confusion Matrix
[[893 483 234]
 [367 250 124]
 [200 109  89]]
```

Figure 4 : Performance of hyper tuned Decision Tree classifier with up sampling & with selected 28 features including new features.
(Kaggle Score : 0.32727)

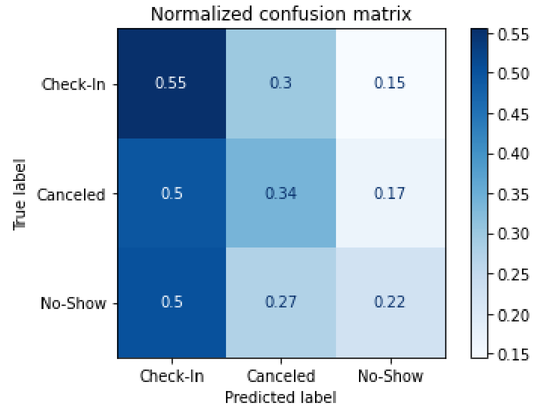


Figure 5 : Normalized Confusion matrix of hyper tuned Decision Tree Classifier with 28 features.

- Along with above methods, voting classifier method was implemented using one vs all classification method and was tested using 3 different models to improve the prediction of each classes. The technique was unable to improve the overall macro F1 score and validation accuracy.
- Feature importance on Decision tree classifier of all the extracted features and selected best 28 features are as follows:

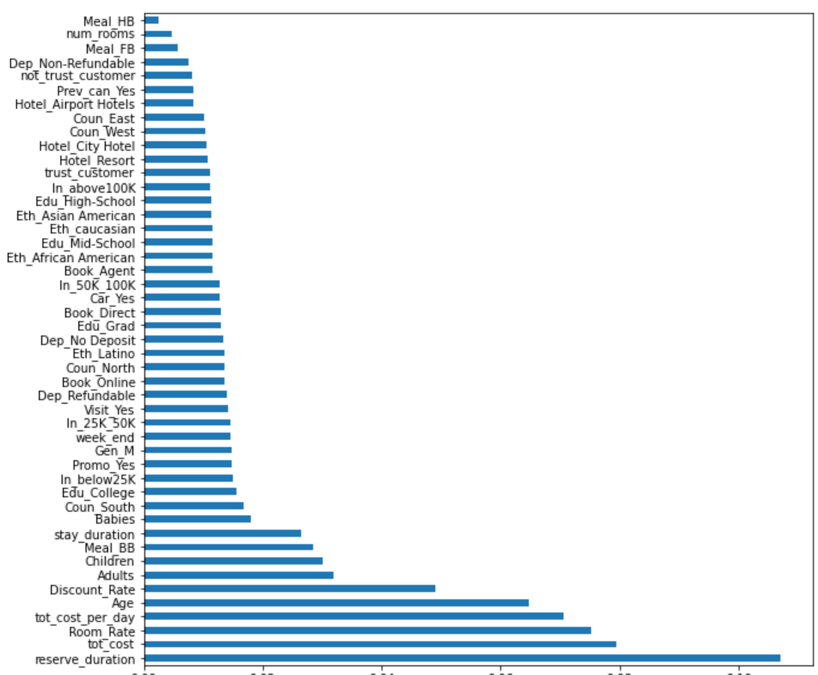


Figure 6 : feature importance of all features

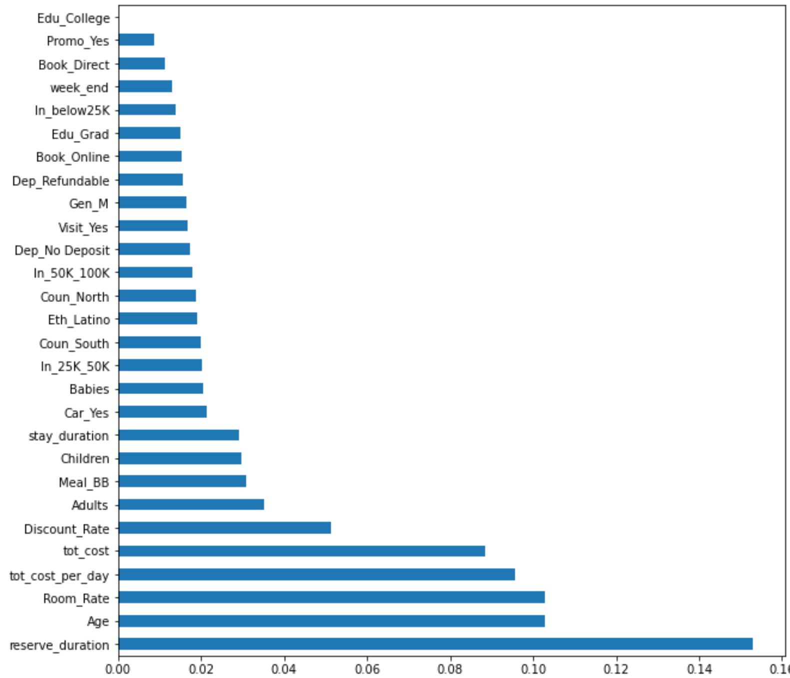


FIGURE 7 : FEATURE IMPORTANCE OF SELECTED 28 FEATURES

Model Building, Model Selection and Performance Evaluation

- Extracted features after each experiment was used to train different type of classifiers to identify the best suitable model for the business problem. The classifiers considered during model building are as follows:
    - Logistic Regression
    - Decision Tree Classifier
    - XGBoost Classifier
    - Support Vector Classifier
    - Random Forest Classifier
    - Multi-Level Perceptron Classifier
    - Neural Networks
    - Extra Tree Classifier (ensemble approach)
- Along with the above methods, voting classifier method was implemented using one vs all classification method and was tested using Decision Tree Classifier, Random Forest Classifier and XGBoost Classifier.
- **Best Model Architecture**
    1. xgboost.XGBClassifier(base_estimator=clf,max_depth=20,n_estimators=15,objective='multi:softmax',gamma = 4.63,learning_rate = 0.2,reg_lambda = 1)
    2. DecisionTreeClassifier(max_depth = 20, class_weight = 'balanced', max_features = 'log2', random_state = 8)

- Hyper-tuning the parameters of each classifier was conducted during each experiment and the best models were selected based on following performance metrics:
    - Macro F1 score
    - Validation accuracy
    - Precision score
    - Recall score.
    - Normalized confusion matrix.
- XGBoost Classifier, Decision Tree Classifier and Random Forest Classifier were selected as best models suitable for the problem.
- After hyper parameter tuning to improve F-score, we went back to data pre-processing and continued the process.
- Performance metrics of our submissions are summarized in the following table:

Classifier	No of Features	Validation Accuracy	Precision Score	Recall Score	Macro F1 Score (Validation)	Kaggle Submission Score
KNN	39	0.4609	0.3568	0.3563	0.3565	0.33335
XGBoost	16	**0.4929**	0.3574	0.3538	0.3509	0.33028
XGBoost (Hyper tuned)	16	0.4656	**0.3703**	0.3710	**0.3694**	**0.33468**
Decision Tree	47	0.4558	0.3538	0.3535	0.3532	0.33415
Decision Tree	28	0.4482	0.3692	**0.3718**	**0.3694**	0.32727

Business Insights

- According to the dataset we obtained, the lead indicators of hotel booking cancellation and not-show classes are **Age, Gender, Income, Ethnicity, Room rate, Discount rate, Adults, Children, Babies, Meal type, Country region, Educational level, Car parking, Booking channel, Deposit type, Visited previously and Use promotion** which are the given features and **reserved duration, total cost, total cost per day, stay duration, weekend** which are the features synthesized using the given features.
- The hotel's focus is to increase their revenue, which will get affected by the cancellation or no show of traveler for their reservations. Thus, the model they should employ should focus on accurately detecting the cancellation and no-show classes than the accurate prediction of check-in class. This can be achieved with the aid of the above-mentioned lead indicators hyper tuned for higher recall values under both cancellation and no-show classes. Such model will help to identify potential travelers' reservations which may result in cancellation

or no-show. However, for the sake of the competition, we also were cautious of the F1 score, which also considers the precision values.

- Cancellation or no-show for the reservations can be reduced among the travelers by predicting the possibilities of cancelling or not-showing up in advance and applying several interventions such as overbooking or cancellation policies, pricing policies and added discount rates such that the impact of cancellation or not-show is reduced.

Questions

(a) What is the expected revenue loss due to booking cancellations?

Loss of Revenue was calculated using the given formula. Travelers except Check-in and Refundable deposit were considered for the calculation.

Revenue Loss = No. of Rooms x Room Rate x Stay Duration x (100 – Discount Rate) / 100

No. of Rooms is estimated by dividing the summation of adults and children by 5 since it is mentioned that one room can only accommodate 5 occupants, apart from the babies. Although it might not represent the exact value in all the cases, for rough estimation we've followed this approach.

No. of Rooms = (Adults + Children) / 5

Discount Rate is considered only if the status of 'Use_Promotion' field in the dataset is Yes.

Stay Duration is calculated from the expected check-in and checkout dates.

Stay Duration = Expected checkout date – Expected check-in date

According to the above formula, Revenue loss was calculated for all the three datasets – train, validation and test.

Train Dataset : Expected Revenue Loss = 1,859,208.50 monetary units

Validation Dataset : Expected Revenue Loss = 336,425.80 monetary units

Test Dataset : Expected Revenue Loss = 1,293,818.30 monetary units

(b) Discuss any additional attributes that hotel management should collect

Purpose of Visit : Because cancellation pattern differs for different purpose of visits. i.e., if a room is booked for business meetings, the chance of it getting cancelled might be very rare whereas family vacation bookings can get cancelled more often.

Cancellation Reason : This is not the specific reason for cancellation, but occurrence of any unfriendly situation that causes the cancellation. For e.g., in the context of Sri Lanka, cancellations due to situations such as Easter attack and Covid-19 should not be considered for forecasting the future cancellations, since situations like this doesn't indicate the customer's real behavior.

Number of Families : This might not be an easily acquirable data, but if this dataset could be obtained, it may serve as a good indicator. The logical backing is that bookings with many families have a lower chance to get cancelled due to interdependency, but bookings made by single families have a comparatively higher probability of cancelling the bookings.

Country : In the dataset, there was a field termed as 'Country Region'. However, it is not explicitly stated whether this defines the local travelers' location within the country or foreign travelers' location in the world map. If it's for foreign travelers, country-wise cancellation patterns can address air-travel issues (E.g., airport issues due to bad climate in European countries) and their native holiday patterns (summer holiday patterns) more accurately. If the dataset is based on local travelers, then the respective province/state can be acquired instead.

History of previous visits and cancellations : In the dataset, these field were given qualitatively whereas quantitative values would be more useful. For e.g., if a person visits eight times and cancels for just one time, inclusion of qualitative measures (yes/no) makes the model consider him/her as a person with a higher chance of cancelling the booking. However, in the real scenario he /she can be trusted more than a customer who hasn't carried out any previous visits.

(c) List down several interventions that the management team can take to reduce the revenue loss.

Proper cancellation policies & overbooking policies : As a way to manage the risk associated to booking cancellations, hotels can implement a combination of overbooking and cancellation policies. However, both overbooking and cancellation policies can be prejudicial to the hotel if they are not implemented properly. Overbooking, by not allowing the customer to check in at the hotel he/she previously booked, forces the hotel to deny service provision to the customer, which can be a terrible experience for the customer. This experience can have a negative effect on both the hotel's reputation and immediate revenue, not to mention the potential loss of future revenue from discontent customers who will not book again to stay at the hotel. Cancellation policies, especially nonrefundable policies, have the potential not to only reduce the number of bookings, but to also to diminish revenue due to their significant discounts on price. However, hotel managers can deduce the value of their demand calculating their net demand with the help of cancellation forecasting. When provided with a more accurate model, derived value of the net demand will also be more accurate and hotel managers can develop better overbooking and cancellation policies that would result in fewer costs and decreased risk. For e.g., if a hotel has more cancellations and comparatively lesser recognition, it can target the overbooking policies whereas if the hotel has less cancellations and a better recognition, it can target the cancellation policies. However, the execution must be flexible enough such that if a long term loyal customer demands any changes in that for his/her visit, the company should be mindful enough to have personalized adjustments.

Special policies for group booking : It is important to make sure that there should be a group contract which should protect you against both overbooking and cancellations by the groups under group bookings. If they originally wanted ten rooms and you offered them a special discounted rate based on a ten-room piece of business, they shouldn't end up either booking twenty rooms at that special rate or cancelling all of them.

Discount rate : Decide on how much of discount prices should be given in order to ensure that adequate revenue is generated during low peak periods.

Proper pricing mechanism : Increase the room rates during peak periods in order to counter any revenue losses during low peak periods. Promotions/Discounts during seasonal times can also boost the income, but hotel management should be mindful about how much of discount provided vs how much of revenue can be earned from it.

Appendix A (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='eyJhcGlfYWRkcmVzcyI6Imh0dHBzOi8vdWkubmVwdHVuZS5haSIsImFwaV91cmwiOiJodHRwczovL3VpLm5lcHR1bmUuYWkiLCJhcGlfa2V5IjoiZmJkZjYxNGYtMTA0ZC00ZTc1LWJiMTYtNzczNjgwZWQ3OTUzIn0=', # 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"""
```

```python
#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)))
```

```python
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)
```