We use Google colab to code.

```
# importing relevant packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import torch

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

# importing datasets
df1=pd.read_csv('/content/drive/My Drive/Datastorm/Hotel-A-train.csv')
df2=pd.read_csv('/content/drive/My
Drive/Datastorm/Hotel-A-validation.csv')
df3=pd.read_csv('/content/drive/My Drive/Datastorm/Hotel-A-test.csv')

# drop rows which has null values
df1.dropna(inplace=True)
df2.dropna(inplace=True)

# concatenate training and validations datasets
df=pd.concat((df1,df2), axis=0)

# drop reservation id column from train and val datasets. Keep ids in test
datasets
df.drop("Reservation-id", axis=1, inplace=True)

id_list = df3["Reservation-id"]
df3.drop("Reservation-id", axis=1, inplace=True)

# check whether the data set is unbalance or not
df["Reservation_Status"].value_counts()
```

```
import datetime

# convert datetime data to number of dates by substracting from now

def no_of_dates(x):
    _x = x.split('/')
    _y = datetime.datetime.now() - datetime.datetime(int(_x[2]),
int(_x[0]), int(_x[1]))
    _y = _y.days
    return _y


df['Booking_date'] = df['Booking_date'].apply(no_of_dates)
df['Expected_checkin'] = df['Expected_checkin'].apply(no_of_dates)
df['Expected_checkout'] = df['Expected_checkout'].apply(no_of_dates)


df3['Booking_date'] = df3['Booking_date'].apply(no_of_dates)
df3['Expected_checkin'] = df3['Expected_checkin'].apply(no_of_dates)
df3['Expected_checkout'] = df3['Expected_checkout'].apply(no_of_dates)

# import encoding packages
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import RobustScaler


categorical_features = ["Ethnicity", "Educational_Level", "Income",
"Country_region", "Adults","Hotel_Type", "Meal_Type", "Visted_Previously",
"Previous_Cancellations",
                        "Deposit_type", "Booking_channel",
"Required_Car_Parking", "Use_Promotion","Gender"]

continuous_features = ['Booking_date','Room_Rate',"Age"]

# replace Reservation_Status column with 1, 2, 3
df.replace(to_replace =["Check-In"], value =1, inplace=True)
df.replace(to_replace =["Canceled"], value =2, inplace=True)
df.replace(to_replace =["No-Show"], value =3, inplace=True)
```

```
for feature in categorical_features:
    lbl_enc = LabelEncoder()
    df[feature]=lbl_enc.fit_transform(df[feature]) + 1
    df3[feature]=lbl_enc.fit_transform(df3[feature]) + 1

# Scaling the features
for feature in continuous_features:
    std_enc = StandardScaler()
    df[feature]=std_enc.fit_transform(df[feature].values.reshape(-1,1))
    df3[feature]=std_enc.fit_transform(df3[feature].values.reshape(-1,1))


df_1 = df[categorical_features]
df_2 = df[continuous_features]
df_3 = df["Reservation_Status"]

df=pd.concat((df_1,df_2,df_3), axis=1)

df_1_t = df3[categorical_features]
df_2_t = df3[continuous_features]

df3=pd.concat((df_1_t,df_2_t), axis=1)


x = df.drop("Reservation_Status",axis=1)
y = df.Reservation_Status

x_final_test = df3


from sklearn.model_selection import train_test_split
# divide the combined dataset to test and validation
X_train,X_test,y_train,y_test=train_test_split(x,y,train_size=0.9)

# importing packages to make the dataset balance
from imblearn.combine import SMOTETomek
from imblearn.under_sampling import NearMiss
from imblearn.over_sampling import RandomOverSampler
```

```
# balancing dataset (Upsampling the dataset)
os=SMOTETomek()
# os=NearMiss()
# os=RandomOverSampler()
X_train_ns,y_train_ns=os.fit_sample(X_train,y_train)

updated_x = pd.DataFrame(X_train_ns, columns = categorical_features +
continuous_features)

updated_y = pd.DataFrame(y_train_ns, columns = ["Reservation_Status"])

from collections import Counter

print("The number of classes before fit {}".format(Counter(y_train)))
print("The number of classes after fit {}".format(Counter(y_train_ns)))

# The number of classes before fit Counter({1: 20584, 2: 4364, 3: 2275})
# The number of classes after fit Counter({3: 20579, 2: 20569, 1: 20564})


# check the importance of each column
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
model=ExtraTreesClassifier()
model.fit(updated_x ,updated_y)

# rank the importance features
ranked_features=pd.Series(model.feature_importances_,index=updated_x.colum
ns)
ranked_features.nlargest(28).plot(kind='barh')
plt.show()

from sklearn.ensemble import RandomForestClassifier
import xgboost
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import ExtraTreesClassifier
```

```
classifier=RandomForestClassifier(n_estimators=1)

classifier = AdaBoostClassifier()


# hyperparameter tuning
n_estimators = [i for i in range(1,101)]
learning_rate=[0.01,0.05,0.1,0.15,0.20]

hyperparameter_grid = {
    'n_estimators': n_estimators,
    'learning_rate':learning_rate,
    }

from sklearn.model_selection import RandomizedSearchCV
random_cv = RandomizedSearchCV(estimator=classifier,
            param_distributions=hyperparameter_grid,
            cv=5, n_iter=50,
            scoring = 'neg_mean_absolute_error',n_jobs = 4,
            verbose = 5,
            return_train_score = True,
            random_state=42)

random_cv.fit(updated_x, updated_y)


random_cv.best_estimator_
# AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None,
learning_rate=0.2,
#                    n_estimators=98, random_state=None)


# make the best model
classifier = AdaBoostClassifier(learning_rate=0.2,n_estimators=98)

# train the model
classifier.fit(updated_x,updated_y)

y_pred=classifier.predict(X_test)
```

```
# check the F1 score
from sklearn.metrics import f1_score
print(f1_score(y_test, y_pred, average='macro'))

y_final = classifier.predict(x_final_test)
result = pd.DataFrame(data={"Reservation-id":id_list.values,
"Reservation_Status":y_final})

result.to_csv('result.csv',index=False)
```

Calculate the revenue loss

```
#import the original test set
Test_df=pd.read_csv('/content/drive/My Drive/Datastorm/Hotel-A-test.csv')

#concatenate the imported test set with the predicted results
output_df = pd.concat((Test_df,y_final), axis=1)

#filtered the result using reservation status 2 (Cancel)
output_df = output_df.loc[output_df['Reservation_Status'] == 2]

#calculated total number of persons
output_df["Total_persons"] = output_df["Adults"] + output_df["Children"]

#function to calculate total rooms
def no_of_rooms(persons):
  rooms = persons//5
  if(persons%5 != 0):
    rooms+=1

  return rooms

output_df['No_of_rooms'] = output_df['Total_persons'].apply(no_of_rooms)

#total price without discount
output_df['Price'] = output_df['No_of_rooms']*output_df["Room_Rate"]
```

```
#final price after reducing the discount
output_df['Final_price'] = output_df['Price'] -
output_df['Price']*(output_df['Discount_Rate']/100.0)

#total loss
expected_revenue_loss = output_df['Final_price'].sum()

print(expected_revenue_loss)
```

The codes that we use for other ML models as mentioned in the report were added to the github.