HOTEL RESERVATION CANCELLATION PREDICTION

Vanindra

Data Description

The online hotel reservation channels have dramatically changed booking possibilities and customers' behavior. A significant number of hotel reservations are called-off due to cancellations or no-shows. The typical reasons for cancellations include change of plans, scheduling conflicts, etc. This is often made easier by the option to do so free of charge or preferably at a low cost which is beneficial to hotel guests but it is a less desirable and possibly revenue-diminishing factor for hotels to deal with.

The Goal

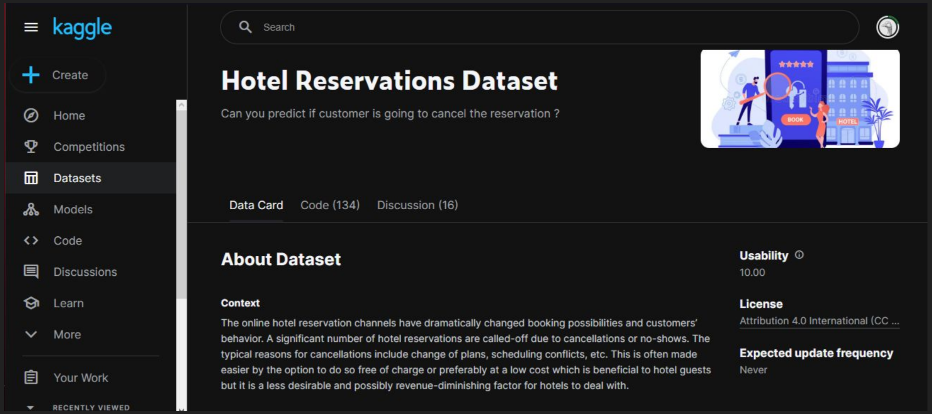
Predicting if the customer is going to honor the reservation or cancel it

Vanindra

SOURCE OF DATA

The dataset used for this project is taken from Kaggle with the name 'Hotel Reservations Dataset' which can be accessed through this link:

https://www.kaggle.com/datasets/ahsan81/hotel-reservations-classification-dataset



IMPORTING PACKAGES AND DATASET

```
In [1]: import numpy as np
        import pandas as pd

In [2]: import matplotlib.pyplot as plt
        import seaborn as sns
```

PACKAGES USED

```
data = pd.read_csv('Hotel Reservations.csv')
data.head()
```

	Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	requ
0	INN00001	2	0	1	2	Meal Plan 1	0
1	INN00002	2	0	2	3	Not Selected	0
2	INN00003	1	0	2	1	Meal Plan 1	0
3	INN00004	2	0	0	2	Meal Plan 1	0
4	INN00005	2	0	1	1	Not Selected	0

IMPORTING DATASET

Vanindra

DATA PROFILING

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 36275 entries, 0 to 36274
Data columns (total 19 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   Booking_ID                            36275 non-null  object
 1   no_of_adults                          36275 non-null  int64
 2   no_of_children                        36275 non-null  int64
 3   no_of_weekend_nights                  36275 non-null  int64
 4   no_of_week_nights                     36275 non-null  int64
 5   type_of_meal_plan                     36275 non-null  object
 6   required_car_parking_space            36275 non-null  int64
 7   room_type_reserved                    36275 non-null  object
 8   lead_time                             36275 non-null  int64
 9   arrival_year                          36275 non-null  int64
 10  arrival_month                         36275 non-null  int64
 11  arrival_date                          36275 non-null  int64
 12  market_segment_type                   36275 non-null  object
 13  repeated_guest                        36275 non-null  int64
 14  no_of_previous_cancellations          36275 non-null  int64
 15  no_of_previous_bookings_not_canceled  36275 non-null  int64
 16  avg_price_per_room                    36275 non-null  float64
 17  no_of_special_requests                36275 non-null  int64
 18  booking_status                        36275 non-null  object
dtypes: float64(1), int64(13), object(5)
memory usage: 5.3+ MB
```

This dataset consists of 36275 rows and 19 columns.

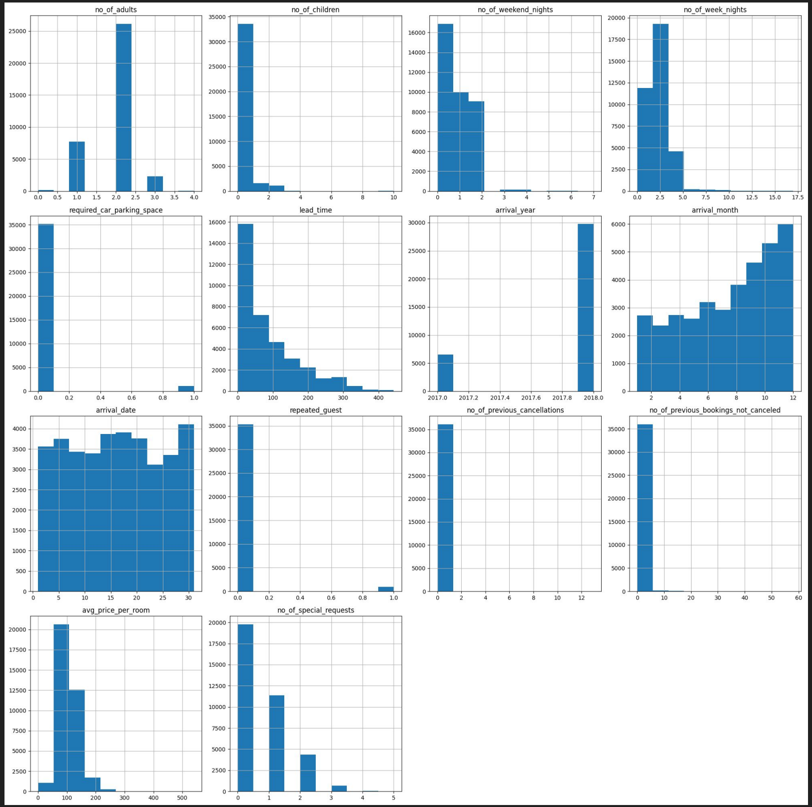
```
# Checking for missing values
data.isnull().sum()
```

```
Booking_ID                              0
no_of_adults                            0
no_of_children                          0
no_of_weekend_nights                    0
no_of_week_nights                       0
type_of_meal_plan                       0
required_car_parking_space              0
room_type_reserved                      0
lead_time                               0
arrival_year                            0
arrival_month                           0
arrival_date                            0
market_segment_type                     0
repeated_guest                          0
no_of_previous_cancellations            0
no_of_previous_bookings_not_canceled    0
avg_price_per_room                      0
no_of_special_requests                  0
booking_status                          0
dtype: int64
```

There are no missing values in this dataset.

PROFILING DESCRIPTIVE STATISTICS

```
# Distribution of each features
data.hist()
plt.show()
```



PROFILING DESCRIPTIVE STATISTICS

	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	required_car_parking_space	lead_time	arrival_year	arrival_month	arrival_date	repeated_guest	no_of_previous_cancellations	no_of_previous_bookings_not_canceled	avg_price_per_room	no_of_special_requests
count	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000	36275.000000
mean	1.844962	0.105279	0.810724	2.204300	0.030986	85.232557	2017.820427	7.423653	15.596995	0.025637	0.023349	0.153411	103.423539	0.619655
std	0.518715	0.402648	0.870644	1.410905	0.173281	85.930817	0.383836	3.069894	8.740447	0.158053	0.368331	1.754171	35.089424	0.786236
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2017.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	2.000000	0.000000	0.000000	1.000000	0.000000	17.000000	2018.000000	5.000000	8.000000	0.000000	0.000000	0.000000	80.300000	0.000000
50%	2.000000	0.000000	1.000000	2.000000	0.000000	57.000000	2018.000000	8.000000	16.000000	0.000000	0.000000	0.000000	99.450000	0.000000
75%	2.000000	0.000000	2.000000	3.000000	0.000000	126.000000	2018.000000	10.000000	23.000000	0.000000	0.000000	0.000000	120.000000	1.000000
max	4.000000	10.000000	7.000000	17.000000	1.000000	443.000000	2018.000000	12.000000	31.000000	1.000000	13.000000	58.000000	540.000000	5.000000

A hotel will not rent out their rooms for free, unless there is a special offer in place. However, there is no indication that the hotels are giving special offers, thus making this number impossible. Therefore, 0 in average price per room will be treated as missing value.

PROFILING DESCRIPTIVE STATISTICS

```python
# Finding and treating missing values
data.loc[data['avg_price_per_room']==0]
```

545 rows × 19 columns

Treatment:

```python
data['avg_price_per_room'] = data['avg_price_per_room'].replace([0],data['avg_price_per_room'].median())
```

```python
data['avg_price_per_room'].describe()
```

```
count    36275.000000
mean       104.917688
std         32.688889
min          0.500000
25%         81.000000
50%         99.450000
75%        120.000000
max        540.000000
Name: avg_price_per_room, dtype: float64
```

Vanindra

PROFILING DESCRIPTIVE STATISTICS

```
#Object data profiling
data.describe(include='O')
```

	Booking_ID	type_of_meal_plan	room_type_reserved	market_segment_type	booking_status
count	36275	36275	36275	36275	36275
unique	36275	4	7	5	2
top	INN00001	Meal Plan 1	Room_Type 1	Online	Not_Canceled
freq	1	27835	28130	23214	24390

CHECKING DUPLICATED DATA

```
data[data.duplicated()]
```

Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week_nights	type_of_meal_plan	required_car_parking_space	room_type_reserved

There is no duplicated entry.

Vanindra

CHECKING THE DATA BALANCE

```
data.groupby('booking_status').size()
```

```
booking_status
Canceled        11885
Not_Canceled    24390
dtype: int64
```

```
data['booking_status'].value_counts(normalize=True)
```

```
Not_Canceled    0.672364
Canceled        0.327636
Name: booking_status, dtype: float64
```

The data is not balance. 67% did not cancel, and 33% canceled.

Vanindra

SIMPLIFYING FEATURES NAME

```
data.rename(columns={'no_of_adults':'adults'},inplace=True)
data.rename(columns={'no_of_children':'children'},inplace=True)
data.rename(columns={'no_of_weekend_nights':'weekend_nights'},inplace=True)
data.rename(columns={'no_of_week_nights':'weekday_nights'},inplace=True)
data.rename(columns={'type_of_meal_plan':'meal_plan'},inplace=True)
data.rename(columns={'no_of_weekend_nights':'weekend_nights'},inplace=True)
data.rename(columns={'no_of_special_requests':'special_requests'},inplace=True)
data.rename(columns={'market_segment_type':'market_type'},inplace=True)
data.rename(columns={'special_requests':'request_num'},inplace=True)
```

```
data.rename(columns={'required_car_parking_space':'parking_space'},inplace=True)
```

```
data.rename(columns={'room_type_reserved':'room_type'},inplace=True)
data.rename(columns={'avg_price_per_room':'avg_room_price'},inplace=True)
```

SIMPLIFYING FEATURES NAME

```
#   Column
--- ------
0   Booking_ID
1   no_of_adults
2   no_of_children
3   no_of_weekend_nights
4   no_of_week_nights
5   type_of_meal_plan
6   required_car_parking_space
7   room_type_reserved
8   lead_time
9   arrival_year
10  arrival_month
11  arrival_date
12  market_segment_type
13  repeated_guest
14  no_of_previous_cancellations
15  no_of_previous_bookings_not_canceled
16  avg_price_per_room
17  no_of_special_requests
18  booking_status
```

Before vs After

```
#   Column
--- ------
0   Booking_ID
1   adults
2   children
3   weekend_nights
4   weekday_nights
5   meal_plan
6   parking_space
7   room_type
8   lead_time
9   arrival_year
10  arrival_month
11  arrival_date
12  market_type
13  repeated_guest
14  no_of_previous_cancellations
15  no_of_previous_bookings_not_canceled
16  avg_room_price
17  request_num
18  booking_status
```

CHANGING THE TARGET INTO NUMERIC

```
data['booking_status'] = data['booking_status'].replace(['Not_Canceled','Canceled'],[0,1])
```

booking_status
Not_Canceled
Not_Canceled
Canceled
Canceled
Canceled

booking_status
0
0
1
1
1

FEATURE SCALLING AND ENCODING

```python
#Importing Scaler and Encoder
from sklearn.preprocessing import RobustScaler
from sklearn.preprocessing import OneHotEncoder
```

Robust Scaler is used as we leave the outliers in the data. One Hot Encoder itself is used because all the categorical feature only serve as category without hierarchy.

```python
rscaler = RobustScaler()
```

```python
ohc = OneHotEncoder(handle_unknown = 'ignore')
```

Vanindra

FEATURE SCALLING AND ENCODING

```
data.drop(columns=['Booking_ID'], axis=1, inplace=True)
```

Booking ID is dropped as it will not be used in Machine Learning.

```
# Copying the dataset for scalling
data_scaled = data.copy()
```

```
data_scaled.head()
```

	adults	children	weekend_nights	weekday_nights	meal_plan	parking_space	room_type	lead_time
0	2	0	1	2	Meal Plan 1	0	Room_Type 1	224
1	2	0	2	3	Not Selected	0	Room_Type 1	5
2	1	0	2	1	Meal Plan 1	0	Room_Type 1	1
3	2	0	0	2	Meal Plan 1	0	Room_Type 1	211
4	2	0	1	1	Not Selected	0	Room_Type 1	48

FEATURE SCALLING

```
# SCalling the numerical features
data_scaled['adults'] = rscaler.fit_transform(data_scaled[['adults']])
data_scaled['children'] = rscaler.fit_transform(data_scaled[['children']])
data_scaled['weekend_nights'] = rscaler.fit_transform(data_scaled[['weekend_nights']])
data_scaled['weekday_nights'] = rscaler.fit_transform(data_scaled[['weekday_nights']])
data_scaled['lead_time'] = rscaler.fit_transform(data_scaled[['lead_time']])
data_scaled['no_of_previous_cancellations'] = rscaler.fit_transform(data_scaled[['no_of_previous_cancellations']])
data_scaled['no_of_previous_bookings_not_canceled'] = rscaler.fit_transform(data_scaled[['no_of_previous_bookings_not_canceled']])
data_scaled['avg_room_price'] = rscaler.fit_transform(data_scaled[['avg_room_price']])
data_scaled['request_num'] = rscaler.fit_transform(data_scaled[['request_num']])
```

```
data_scaled.head()
```

	adults	children	weekend_nights	weekday_nights	meal_plan	parking_space	room_type	lead_time
0	0.0	0.0	0.0	0.0	Meal Plan 1	0	Room_Type 1	1.532110
1	0.0	0.0	0.5	0.5	Not Selected	0	Room_Type 1	-0.477064
2	-1.0	0.0	0.5	-0.5	Meal Plan 1	0	Room_Type 1	-0.513761
3	0.0	0.0	-0.5	0.0	Meal Plan 1	0	Room_Type 1	1.412844
4	0.0	0.0	0.0	-0.5	Not Selected	0	Room_Type 1	-0.082569

Vanindra

FEATURE ENCODING

```
col = sorted(data['meal_plan'].unique().tolist()) + sorted(data['room_type'].unique().tolist()) + sorted(data['market_type'].unique().tolist())
```

```
#Encoding
enc_df = pd.DataFrame(ohc.fit_transform(data_scaled[['meal_plan', 'room_type', 'market_type']]).toarray(), columns=col)
```

```
enc_df.head()
```

	Meal Plan 1	Meal Plan 2	Meal Plan 3	Not Selected	Room_Type 1	Room_Type 2	Room_Type 3	Room_Type 4	Room_Type 5	Room_Type 6	Room_Type 7	Aviation	Complementary	Corporate	Offline	Online
0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
1	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
2	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
4	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

FEATURE SCALLING AND ENCODING

```
#Joining the scaled dataframe with encoded dataframe
data_scaled = data_scaled.join(enc_df)
data_scaled.head()
```

	adults	children	weekend_nights	weekday_nights	meal_plan	parking_space	room_type	lead_time	arrival_year	arrival_month	...	Room_Type 3	Room_Type 4	Room_Type 5	Room_Type 6	Room_Type 7	Aviation
0	0.0	0.0	0.0	0.0	Meal Plan 1	0	Room_Type 1	1.532110	2017	10	...	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.5	0.5	Not Selected	0	Room_Type 1	-0.477064	2018	11	...	0.0	0.0	0.0	0.0	0.0	0.0
2	-1.0	0.0	0.5	-0.5	Meal Plan 1	0	Room_Type 1	-0.513761	2018	2	...	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	-0.5	0.0	Meal Plan 1	0	Room_Type 1	1.412844	2018	5	...	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	-0.5	Not Selected	0	Room_Type 1	-0.082569	2018	4	...	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 34 columns

Vanindra

FEATURE SCALLING AND ENCODING

```
In [298]:
# Dropping the unused features
data_scaled.drop(columns=['meal_plan','room_type','market_type', 'arrival_year','arrival_month','arrival_date'],axis=1,inplace=True)
```

```
In [301]:
# joining dummies to the dataframe
final_data=data_scaled.copy()
```

```
 0   adults                                36275 non-null  float64
 1   children                              36275 non-null  float64
 2   weekend_nights                        36275 non-null  float64
 3   weekday_nights                        36275 non-null  float64
 4   parking_space                         36275 non-null  int64
 5   lead_time                             36275 non-null  float64
 6   repeated_guest                        36275 non-null  int64
 7   no_of_previous_cancellations          36275 non-null  float64
 8   no_of_previous_bookings_not_canceled  36275 non-null  float64
 9   avg_room_price                        36275 non-null  float64
 10  request_num                           36275 non-null  float64
 11  booking_status                        36275 non-null  int64
 12  Meal Plan 1                           36275 non-null  float64
 13  Meal Plan 2                           36275 non-null  float64
 14  Meal Plan 3                           36275 non-null  float64
 15  Not Selected                          36275 non-null  float64
 16  Room_Type 1                           36275 non-null  float64
 17  Room_Type 2                           36275 non-null  float64
 18  Room_Type 3                           36275 non-null  float64
 19  Room_Type 4                           36275 non-null  float64
 20  Room_Type 5                           36275 non-null  float64
 21  Room_Type 6                           36275 non-null  float64
 22  Room_Type 7                           36275 non-null  float64
 23  Aviation                              36275 non-null  float64
 24  Complementary                         36275 non-null  float64
 25  Corporate                             36275 non-null  float64
 26  Offline                               36275 non-null  float64
 27  Online                                36275 non-null  float64
```

All the features is either integer or float.

TRAIN TEST SPLIT

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from collections import Counter
```

```
#Splitting X and Y
X = final_data.drop(columns=['booking_status'],axis=1)
y = final_data['booking_status']
```

```
#split train-rest data
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=0)
```

```
X_train.shape
```

```
(27206, 28)
```

```
y_train.shape
```

```
(27206,)
```

Vanindra

BALANCING DATA

```
print('Before Oversampling: ',Counter(y_train))
#defining smote
SMOTE = SMOTE(random_state=0)

#fit and apply the transform
X_train_smote, y_train_smote = SMOTE.fit_resample(X_train,y_train)

#summarize class distribution
print('After Oversampling: ',Counter(y_train_smote))
```

```
Before Oversampling:  Counter({0: 18328, 1: 8878})
After Oversampling:  Counter({0: 18328, 1: 18328})
```

Vanindra

MACHINE LEARNING MODELING

(Decision Tree Classifier)

```
# Decision Tree without resampling
dt = DecisionTreeClassifier(random_state=0)
dt.fit(X_train, y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_
state=0)
```

```
# Decision Tree with resampling
dt2 = DecisionTreeClassifier(random_state=0)
dt2.fit(X_train_smote, y_train_smote)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(random_
state=0)
```

```
y_pred_dt = dt.predict(X_test)
print(classification_report(y_test, y_pred_dt))
```

	precision	recall	f1-score	support
0	0.89	0.90	0.89	6062
1	0.79	0.77	0.78	3007
accuracy			0.85	9069
macro avg	0.84	0.83	0.83	9069
weighted avg	0.85	0.85	0.85	9069

```
y_pred_dt2 = dt2.predict(X_test)
print(classification_report(y_test, y_pred_dt2))
```

	precision	recall	f1-score	support
0	0.89	0.87	0.88	6062
1	0.75	0.78	0.77	3007
accuracy			0.84	9069
macro avg	0.82	0.83	0.83	9069
weighted avg	0.85	0.84	0.84	9069

MACHINE LEARNING MODELING
(Random Forest Classifier)

```
# Random Forest without resampling
rf1 = RandomForestClassifier(random_state=0)
rf1.fit(X_train,y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_
state=0)
```

```
y_pred_rf1 = rf1.predict(X_test)
print(classification_report(y_test, y_pred_rf1))
```

```
              precision    recall  f1-score   support

           0       0.89      0.94      0.92      6062
           1       0.86      0.78      0.82      3007

    accuracy                           0.88      9069
   macro avg       0.88      0.86      0.87      9069
weighted avg       0.88      0.88      0.88      9069
```

```
# Random Forest with resampling
rf2 = RandomForestClassifier(random_state=0)
rf2.fit(X_train_smote,y_train_smote)
```

```
RandomForestClassifier
RandomForestClassifier(random_
state=0)
```

```
y_pred_rf2 = rf2.predict(X_test)
print(classification_report(y_test, y_pred_rf2))
```

```
              precision    recall  f1-score   support

           0       0.90      0.91      0.91      6062
           1       0.82      0.80      0.81      3007

    accuracy                           0.88      9069
   macro avg       0.86      0.86      0.86      9069
weighted avg       0.88      0.88      0.88      9069
```

MACHINE LEARNING MODELING
(KNN)

```python
# KNN without resampling
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

```python
y_pred_knn = knn.predict(X_test)
print(classification_report(y_test, y_pred_knn))
```

	precision	recall	f1-score	support
0	0.88	0.91	0.89	6062
1	0.80	0.74	0.77	3007
accuracy			0.85	9069
macro avg	0.84	0.82	0.83	9069
weighted avg	0.85	0.85	0.85	9069

```python
# KNN with resampling
knn2 = KNeighborsClassifier()
knn2.fit(X_train_smote, y_train_smote)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

```python
y_pred_knn2 = knn2.predict(X_test)
print(classification_report(y_test, y_pred_knn2))
```

	precision	recall	f1-score	support
0	0.90	0.85	0.87	6062
1	0.72	0.81	0.76	3007
accuracy			0.83	9069
macro avg	0.81	0.83	0.82	9069
weighted avg	0.84	0.83	0.84	9069

MACHINE LEARNING MODELING

(Support Vector Classifier, Linear Kernel)

```
# SVCL without resampling
svc = SVC(kernel='linear',random_state=0)
svc.fit(X_train,y_train)
```

```
SVC
SVC(kernel='linear', random_s
tate=0)
```

```
y_pred_svc = svc.predict(X_test)
print(classification_report(y_test, y_pred_svc))
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86      6062
           1       0.75      0.59      0.66      3007

    accuracy                           0.80      9069
   macro avg       0.78      0.75      0.76      9069
weighted avg       0.79      0.80      0.79      9069
```

```
# SVCL with resampling
svc2 = SVC(kernel='linear',random_state=0)
svc2.fit(X_train_smote,y_train_smote)
```

```
SVC
SVC(kernel='linear', random_s
tate=0)
```

```
y_pred_svc2 = svc2.predict(X_test)
print(classification_report(y_test, y_pred_svc2))
```

```
              precision    recall  f1-score   support

           0       0.87      0.77      0.82      6062
           1       0.63      0.77      0.69      3007

    accuracy                           0.77      9069
   macro avg       0.75      0.77      0.76      9069
weighted avg       0.79      0.77      0.78      9069
```

MACHINE LEARNING MODELING
(Support Vector Classifier, RBF Kernel)

```
# SVCR without resampling
svc3 = SVC(kernel='rbf',random_state=0)
svc3.fit(X_train,y_train)
```

```
SVC
SVC(random_state=0)
```

```
y_pred_svc3 = svc3.predict(X_test)
print(classification_report(y_test, y_pred_svc3))
```

	precision	recall	f1-score	support
0	0.83	0.94	0.88	6062
1	0.82	0.61	0.70	3007
accuracy			0.83	9069
macro avg	0.83	0.77	0.79	9069
weighted avg	0.83	0.83	0.82	9069

```
# SVCR with resampling
svc4 = SVC(kernel='rbf',random_state=0)
svc4.fit(X_train_smote,y_train_smote)
```

```
SVC
SVC(random_state=0)
```

```
y_pred_svc4 = svc4.predict(X_test)
print(classification_report(y_test, y_pred_svc4))
```

	precision	recall	f1-score	support
0	0.89	0.83	0.86	6062
1	0.69	0.80	0.74	3007
accuracy			0.82	9069
macro avg	0.79	0.81	0.80	9069
weighted avg	0.83	0.82	0.82	9069

MACHINE LEARNING MODELING
(Best Performing Model)

```
y_pred_rf1 = rf1.predict(X_test)
print(classification_report(y_test, y_pred_rf1))
```

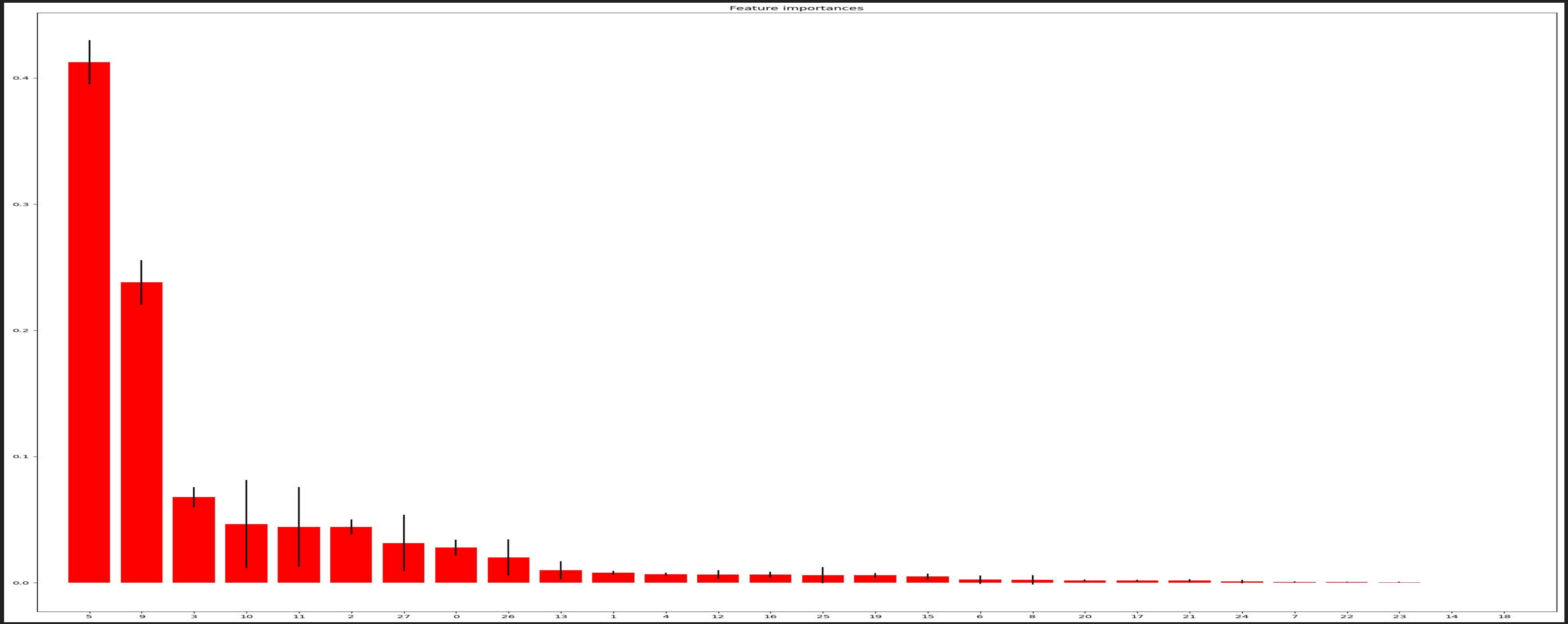
```
              precision    recall  f1-score   support

           0       0.89      0.94      0.92      6062
           1       0.86      0.78      0.82      3007

    accuracy                           0.88      9069
   macro avg       0.88      0.86      0.87      9069
weighted avg       0.88      0.88      0.88      9069
```

Random Forest Classifier without resampling

Vanindra

CHECKING FEATURE IMPORTANCE

In the model earlier, it still uses a lot of features. I aim to check the importance of each features so I can reduce it thus making the model work more efficiently.

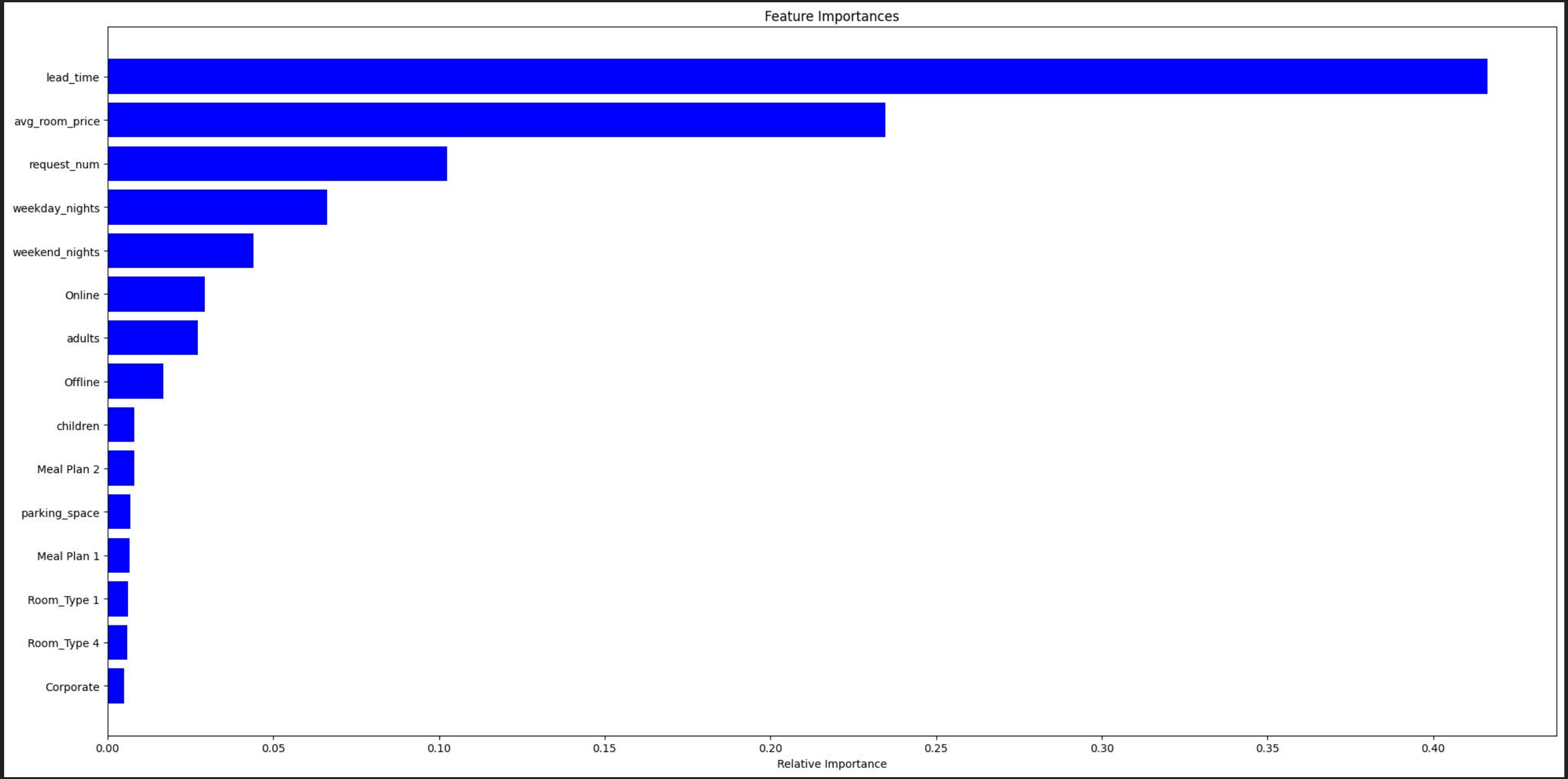


CHECKING FEATURE IMPORTANCE

```python
# Checking Feature Importance
features = X.columns
importances = rf1.feature_importances_
indices = np.argsort(importances)[-15:]
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

Vanindra

CHECKING FEATURE IMPORTANCE



RETRAINING AFTER CHECKING FEATURE IMPORTANCE

```
#Splitting X and Y
# X = final_data['adults','children','lead_time','avg_room_price','request_num','weekday_nights','weekend_nights','Online','Offline']
X = final_data.drop(columns=['parking_space','repeated_guest','no_of_previous_cancellations','no_of_previous_bookings_not_canceled',
                            'Meal Plan 1','Meal Plan 2','Meal Plan 3','Not Selected','Room_Type 1','Room_Type 2','Room_Type 3','Room_Type 4',
                            'Room_Type 5','Room_Type 6','Room_Type 7','Aviation','Complementary','Corporate','booking_status'], axis=1)
y = final_data['booking_status']
```

```
#split train-rest data
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=0)
```

Vanindra

RETRAINING AFTER CHECKING FEATURE IMPORTANCE

```
y_pred_rf1 = rf1.predict(X_test)
print(classification_report(y_test, y_pred_rf1))
```

	precision	recall	f1-score	support
0	0.89	0.94	0.92	6062
1	0.86	0.78	0.82	3007
accuracy			0.88	9069
macro avg	0.88	0.86	0.87	9069
weighted avg	0.88	0.88	0.88	9069

BEFORE

```
y_pred_rf3 = rf3.predict(X_test)
print(classification_report(y_test, y_pred_rf3))
```

	precision	recall	f1-score	support
0	0.89	0.93	0.91	6062
1	0.85	0.76	0.80	3007
accuracy			0.88	9069
macro avg	0.87	0.85	0.86	9069
weighted avg	0.87	0.88	0.87	9069

AFTER

There is not much difference after we reduce the number of features used. This performance can be improved with hyperparameter tuning. In this project, I am using the Grid Search CV.

HYPERPARAMETER TUNING

```
from sklearn.model_selection import GridSearchCV
```

```
# Define the parameter grid to search over
param_grid = {
    'n_estimators': [10, 50, 100, 200],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}


# Create the GridSearchCV object
grid_search = GridSearchCV(rf3, param_grid=param_grid, cv=5)

# Fit the GridSearchCV object to the data
grid_search.fit(X, y)

# Print the best hyperparameters and the corresponding score
print("Best hyperparameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_)
```

```
Best hyperparameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
Best score: 0.8831426602343211
```

Vanindra

HYPERPARAMETER TUNING

```
grid_search.best_params_
```

```
{'max_depth': 20,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 200}
```

```
grid_search.best_estimator_
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=20, n_estimators=200, r
andom_state=0)
```

Vanindra

HYPERPARAMETER TUNING RESULT

```
y_pred_rf3 = rf3.predict(X_test)
print(classification_report(y_test, y_pred_rf3))
```

	precision	recall	f1-score	support
0	0.89	0.93	0.91	6062
1	0.85	0.76	0.80	3007
accuracy			0.88	9069
macro avg	0.87	0.85	0.86	9069
weighted avg	0.87	0.88	0.87	9069

BEFORE

```
best_grid = grid_search.best_estimator_
```

```
y_pred_grid = best_grid.predict(X_test)
print(classification_report(y_test, y_pred_grid))
```

	precision	recall	f1-score	support
0	0.96	0.99	0.97	6062
1	0.97	0.92	0.95	3007
accuracy			0.97	9069
macro avg	0.97	0.95	0.96	9069
weighted avg	0.97	0.97	0.97	9069

AFTER

After tuning, we can see that the performance of the model is significantly improved.

Vanindra

THANK YOU