PA2 Final Project

😀

Yiqing Cheng, Hanwei Hu, Jiusi Li, Xingyu Wang

Business Problem:

Guest room scheduling is a key area in maintaining hotel operation efficiency. When a customer cancels their reservation unexpectedly, the hotel needs to spend extra effort in staff planning and extra cost on maintaining vacant rooms. A predictive model is built to help hotel owners understand whether a customer would cancel their reservation.
After receiving the cancellation prediction result from the model, the hotel can take proactive measures to reduce costs. One such measure is to offer incentives to the customer in order to dissuade them from canceling their reservation. This can include discounts on room rates or providing complimentary services or amenities, such as a free spa treatment. The hotel can also attempt to rebook the canceled room with another customer. By doing so, the hotel can minimize the impact of the cancellation on its occupancy and revenue.

Data Description:

The dataset consists of 36275 unique hotel booking records and 19 columns. Each row represents a booking record of the hotel. The booking_status column indicates if the booking is canceled or not, and it will be the predicted variable of the model.

The explanation of the predictor variables as following:

- Booking_ID: unique identifier of each booking
- No_of_adults: Number of adults
- No_of_children: Number of Children
- no_of_weekend_night: Number of weekend nights (Saturday or Sunday) the guest stayed or booked to stay at the hotel
- no_of_week_nights: Number of week nights (Monday to Friday) the guest stayed or booked to stay at the hotel
- type_of_meal_plan: Type of meal plan booked by the customer
- required_car_parking_space: Does the customer require a car parking space?
- room_type_reserved: Type of room reserved by the customer
- lead_time: Number of days between the date of booking and the arrival date
    - arrival_year: Year of arrival date
    - arrival_month: Month of arrival date
    - arrival_date: Date of the month
    - market_segment_type: Market segment designation
- repeated_guest: Is the customer a repeated guest?

- no_of_previous_cancellations: Number of previous bookings that were canceled by the customer prior to the current booking
- no_of_previous_bookings_not_canceled: Number of previous bookings not canceled by the customer prior to the current booking
- avg_price_per_room: Average price per day of the reservation; prices of the rooms are dynamic. (in euros)
- no_of_special_requests: Total number of special requests made by the customer (e.g. high floor, view from the room, etc)

EDA

Preprocessing:

1. **Feature selection:**
   Feature selection is an important step in building a predictive model. In this regard, we have excluded columns with all unique values, such as Booking_ID, as they do not provide any useful information for our model.

   On the other hand, we have considered other features such as reservation details, previous booking history, and special requests as potentially informative and relevant in building the predictive model. These features will be retained and further preprocessed to ensure they are appropriate for model fitting.
2. **Categorical variables encoding** ①
   - **Transform to Ordinal encoding (From code EDA and Data Transformation)**
     - It is beneficial to change categorical variables. First, it may improve the accuracy of modeling since it can capture some latent relationships. Second, it can result in a simple model since the dimension of the data reduces. Third, the new variables can be more interpretable than categorical variables as the underlying relationship is captured.
   - **OneHot Encoding (From code EDA and Data Transformation)**
     - Only two categorical variables, type_of_meal_plan and market_segment_type, are one hot encoded because we believe there is no apparent ordinal relationship between different market segments and different meal plans.
3. **Dimensional Reduction** ②
   - **PCA (From code PCA):** This is a method to reduce dimensions of data by linear combinations. We tried to use it, but it seems PCA does not improve the performance of models, and we do not have a large number of features. We finally discard the PCA in our final models.

4. **Anomaly detection(From code EDA and Data Transformation)**
    - We applied the IsolationTree model, which is a tree-based anomaly detection algorithm for identifying outliers or anomalies in a dataset. It is based on the concept of randomly partitioning data into subsets, and then isolating anomalies by detecting observations that require fewer partitions to isolate. It would work well since the correlation of features is not high. By using this algorithm, we drop about 4000 rows, and the distribution becomes better.

Methods (From code Cross Validation on All Models): ③

For this project, a wide variety of machine learning models have been used to predict whether a customer would cancel their reservation. The models include probabilistic, linear, non-linear, tree, and ensemble models. Each of these models has its own strengths and weaknesses, making them suitable for different types of data and problems. In this section, we will focus on introducing some of the important models.

The baseline model is a probabilistic model based on Bayes' theorem called naive bayesian. It is a computational efficient algorithm that can run quickly in large data. However, it requires the assumption of feature independence, which may not hold true in most cases. We used GaussianNB from sklearn to train the naive bayesian model Another baseline model is SVM. It is a classification algorithm which tries to find a decision boundary that separates the data with distance as far as possible. It can also apply to non-linear data by transformed data into a higher dimensional space(kernel trick)

A neural network classification model is a type of machine learning model that uses a neural network to classify data into different categories or classes. It works by analyzing a set of input data and using a series of interconnected nodes to identify patterns and relationships within the data. It can handle non-linear relationships and interactions between variables, making it well-suited for dataset with categorical variables and thus possible complex interactions between features. Neural network is also robust to noise and outliers in data and can automatically extract relevant features from raw data, reducing the need for manual feature engineering. The code can be found in note book named Tunning Logistic and NN model.

The best performing tree based model is random forest and XGboost. Random Forest is a machine learning algorithm that combines multiple decision trees to make predictions. It would create subtrees using bootstrap sample and random subset of features. It is highly accurate and robust. The code can be found in note book named Tune_XGboostModel and Tune Random Forest.

Xgboost is an ensemble tree-based model that uses gradient boosting to iteratively add trees to correct the error. It is highly accurate and efficient. It has used techniques such as parallel computing and weighted quantile sketch to improve the model performance and efficiency. The package we used is called xgboost.

Model Performance (From code Cross Validation on All Models)

Model Name	ROC-AUC Score
Naive Bayes	0.793
DecisionTreeClassifier	0.822
RidgeClassifierCV	0.855
Logistic Regression	0.855
svc	0.882
Neural_Network	0.893
XGBoost	0.933
Random Forest	0.937

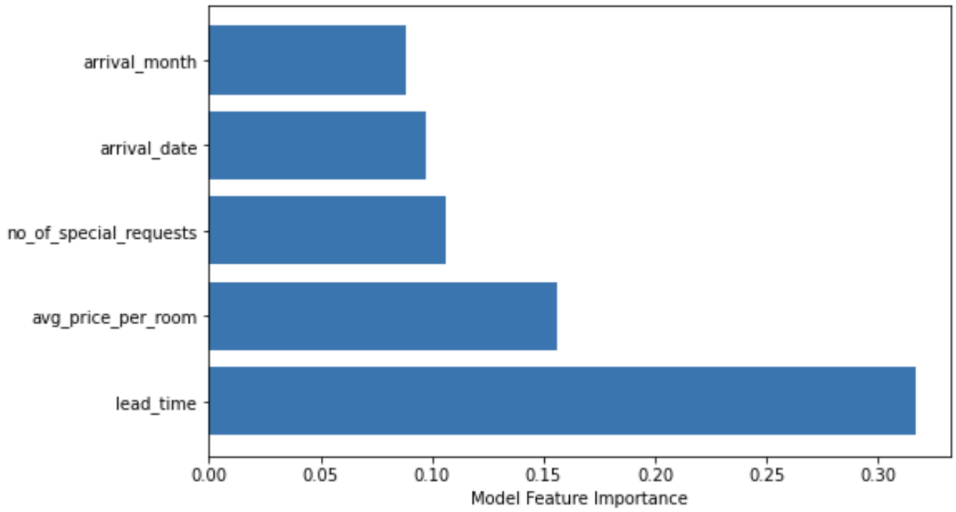
Our analysis of the modeling performance reveals that the tree-based model surpasses other models in terms of performance. This is largely attributed to the significant presence of categorical features in the data, which are handled well by tree-based models.

However, the neural network model also demonstrated satisfactory performance, owing to its ability to capture non-linear relationships effectively. On the other hand, linear models were found to be less effective, as they cannot capture non-linear relationships.

Final Model (From code Cross Validation on All Models):

The best tuned model is a random forest model with 1600 trees and a maximum depth of 30. Each tree is built with a relatively deeper size to capture non-linearities, and we built sufficient trees to smooth out the fitted line. This model has an AUC score of 0.937, outperforming all other models.

Feature Importance (From code Cross Validation on All Models):



Based on the feature importance analysis for random forest model, variable lead_time is the most important among 18 predictors. Avg_price_per_room and number of special requests are the second and third variables that have the most influence on predicted outcomes. As the number of days between the date of booking and the arrival date gets larger, there could be higher uncertainties of customers changing their plans, which increases the risk of cancellations. In real life scenarios, hotels' cancellation policies generally allow customers to cancel the bookings several days ahead of the arrival date without penalty fees, which potentially makes customers with longer lead time more likely to cancel than customers with shorter lead time. Demands usually are sensitive to the price. People who reserved expensive rooms may have higher sensitivity to other external factors, such as weather and finding other cheaper hotels, and thus are more likely to cancel the room.

Business Result/Solution

Based on the cancellation prediction generated by the model, the hotel can proactively take the following actions to lower its expenses:

- The hotel can provide incentives to the customers to discourage them from canceling their reservation: providing discount on their room rates, giving free spa session, or serving free dinner at the hotel restaurant
- The hotel can fill the canceled room by reserving it for another guest. It can set 95% as a threshold of rebooking the room. Since the model precision is 86.3%, the model precision for at least one room will be canceled is 98.1%. The hotel can therefore rebook one room for every two customers predicted to cancel order by the model.

EDA&Anomaly_Detection

March 7, 2023

1 EDA and Data Transformation

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

[117]:
```
df = pd.read_csv("../Hotel Reservations.csv")
```

[118]:
```
df.duplicated().sum()
```

[118]:
```
0
```

[119]:
```
df.dtypes
```

[119]:
```
Booking_ID                               object
no_of_adults                              int64
no_of_children                            int64
no_of_weekend_nights                      int64
no_of_week_nights                         int64
type_of_meal_plan                        object
required_car_parking_space                int64
room_type_reserved                       object
lead_time                                 int64
arrival_year                              int64
arrival_month                             int64
arrival_date                              int64
market_segment_type                      object
repeated_guest                            int64
no_of_previous_cancellations              int64
no_of_previous_bookings_not_canceled      int64
avg_price_per_room                      float64
no_of_special_requests                    int64
booking_status                           object
dtype: object
```

[120]:
```
df.isna().sum()
```

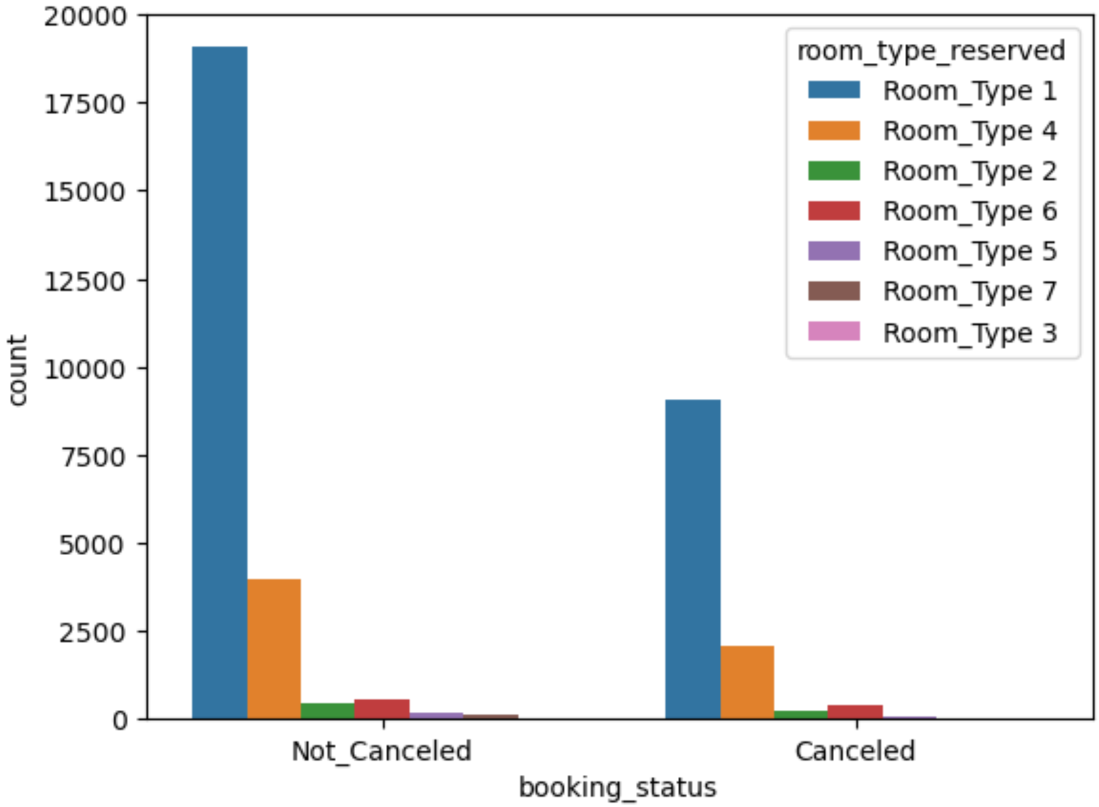
[120]:
```
Booking_ID                              0
no_of_adults                            0
no_of_children                          0
no_of_weekend_nights                    0
no_of_week_nights                       0
type_of_meal_plan                       0
required_car_parking_space              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
```

2 Transform some variables to Nominal

[121]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
```

[122]:
```python
sns.countplot(data=df, x="booking_status", hue="room_type_reserved")
```

[122]:
```
<AxesSubplot: xlabel='booking_status', ylabel='count'>
```



[ ]:
```
ax = sns.countplot(x="booking_status", data=df, hue="room_type_reserved")
total = len(df["booking_status"])
for p in ax.patches:
    percentage = '{:.1f}%'.format(100 * p.get_height()/total)
    x = p.get_x() + p.get_width()/2 - 0.1
    y = p.get_height() + 3
    ax.annotate(percentage, (x, y), size=12)

# Add labels and title
ax.set_xlabel("Room type")
ax.set_ylabel("Count")
ax.set_title("Count Plot with Percentage")

# Show plot
plt.show()
```

It seems that the room could be ordinal.

[123]:
```
df.groupby(by="room_type_reserved")["avg_price_per_room"].mean().sort_values()
```

[123]:
```
room_type_reserved
Room_Type 3     73.678571
Room_Type 2     87.848555
Room_Type 1     95.918532
Room_Type 5    123.733623
Room_Type 4    125.287317
Room_Type 7    155.198291
Room_Type 6    182.212836
Name: avg_price_per_room, dtype: float64
```

[125]:
```
df.drop(["Booking_ID"],axis=1, inplace=True)
```

[126]:
```
df.booking_status.replace({"Not_Canceled": 0, "Canceled": 1},inplace=True)
```

[128]:
```
ordinal_mapping = {'Room_Type 1': 3, 'Room_Type 2':2, 'Room_Type 3': 1,
                    'Room_Type 4': 5, 'Room_Type 5':4, 'Room_Type 6': 7,
                   'Room_Type 7':6}

# Map the room type to ordinal values
df['room_type_reserved'] = df['room_type_reserved'].map(ordinal_mapping)
```

[129]:
```
df_dummy = pd.get_dummies(df,drop_first=True)
```

[131]:
```
Feb_wrong2018 = df[(df["arrival_year"] == 2018) & (df["arrival_month"] == 2) &␣
 ↪(df["arrival_date"] >= 29)]
dropped_index = Feb_wrong2018.index
df_dummy.drop(dropped_index, axis = 0, inplace = True)
```

[134]:
```
from sklearn.ensemble import IsolationForest
dat_iso = df_dummy
model = IsolationForest(n_estimators = 150,max_samples ='auto',contamination =␣
 ↪"auto", max_features = 1.0)
model.fit(dat_iso)
scores = model.decision_function(dat_iso)
anomaly = model.predict(dat_iso)
dat_iso['scores'] = scores
dat_iso['anomaly'] = anomaly
anomaly = dat_iso.loc[dat_iso['anomaly'] == -1]
anomaly_index = list(dat_iso.index)
dat_iso_drop = dat_iso[dat_iso.anomaly == 1]
dat_iso_drop.head()
dat_iso_drop.drop(columns=['scores', 'anomaly'], inplace= True)
dat_iso_drop.head()
```

```
/var/folders/sv/npxlc_k53696tn8hryg5dx5w0000gn/T/ipykernel_51795/1179662965.py:1
3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dat_iso_drop.drop(columns=['scores', 'anomaly'], inplace= True)
```

[134]:

```
   no_of_adults  no_of_children  no_of_weekend_nights  no_of_week_nights  \
0             2               0                     1                  2
1             2               0                     2                  3
2             1               0                     2                  1
3             2               0                     0                  2
4             2               0                     1                  1

   required_car_parking_space  room_type_reserved  lead_time  arrival_year  \
0                           0                   3        224          2017
1                           0                   3          5          2018
2                           0                   3          1          2018
3                           0                   3        211          2018
4                           0                   3         48          2018

   arrival_month  arrival_date  …  avg_price_per_room  \
0             10             2  …               65.00
1             11             6  …              106.68
2              2            28  …               60.00
3              5            20  …              100.00
4              4            11  …               94.50

   no_of_special_requests  booking_status  type_of_meal_plan_Meal Plan 2  \
0                       0               0                              0
1                       1               0                              0
2                       0               1                              0
3                       0               1                              0
4                       0               1                              0

   type_of_meal_plan_Meal Plan 3  type_of_meal_plan_Not Selected  \
0                              0                               0
1                              0                               1
2                              0                               0
3                              0                               0
4                              0                               1

   market_segment_type_Complementary  market_segment_type_Corporate  \
0                                  0                              0
1                                  0                              0
2                                  0                              0
3                                  0                              0
4                                  0                              0
```

```
   market_segment_type_Offline  market_segment_type_Online
0                            1                           0
1                            0                           1
2                            0                           1
3                            0                           1
4                            0                           1

[5 rows x 23 columns]
```

[136]:

```
dat_iso.drop(["scores",'anomaly'],axis=1, inplace=True)
```

[138]:

```
dat_iso.to_csv("with_anomaly.csv",index=False)
dat_iso_drop.to_csv("without_anomaly.csv",index=False)
```

PCA

March 8, 2023

1 PCA

[1]:
```
import numpy as np
import pandas as pd
from sklearn import datasets, metrics
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.ticker import MaxNLocator
import numpy as np
import math
from sklearn.decomposition import PCA
import requests
```

[2]:
```
# loading the data
url="https://raw.githubusercontent.com/KelvinYQC/msia420PA_project/
↪6bb5fe63050105194819f99c90476486076e5227/Data/Hotel%20Reservations.csv"
df=pd.read_csv(url)
df.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
```

[3]:
```
# drop the response variable
df.drop(['Booking_ID', 'booking_status'], axis=1, inplace=True)
dropping = []
for col in df:
    if df[col].dtypes == "object":
        dropping.append(col)
dropping
```

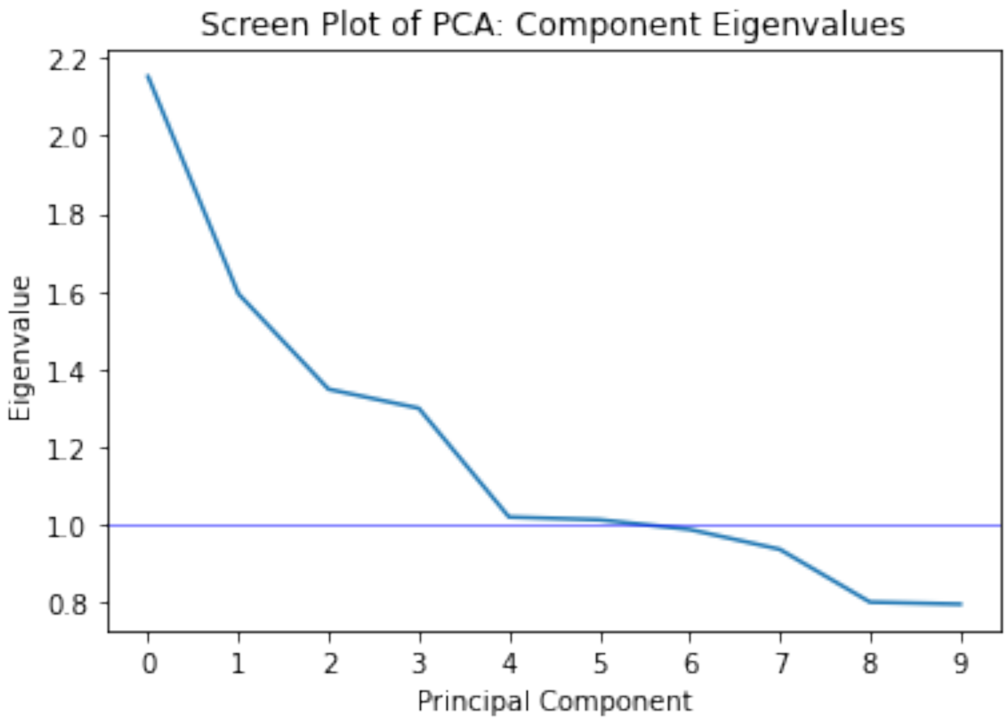
[3]:
```
['type_of_meal_plan', 'room_type_reserved', 'market_segment_type']
```

[4]:
```
df= df.drop(dropping,axis=1)
df = pd.get_dummies(df)
```

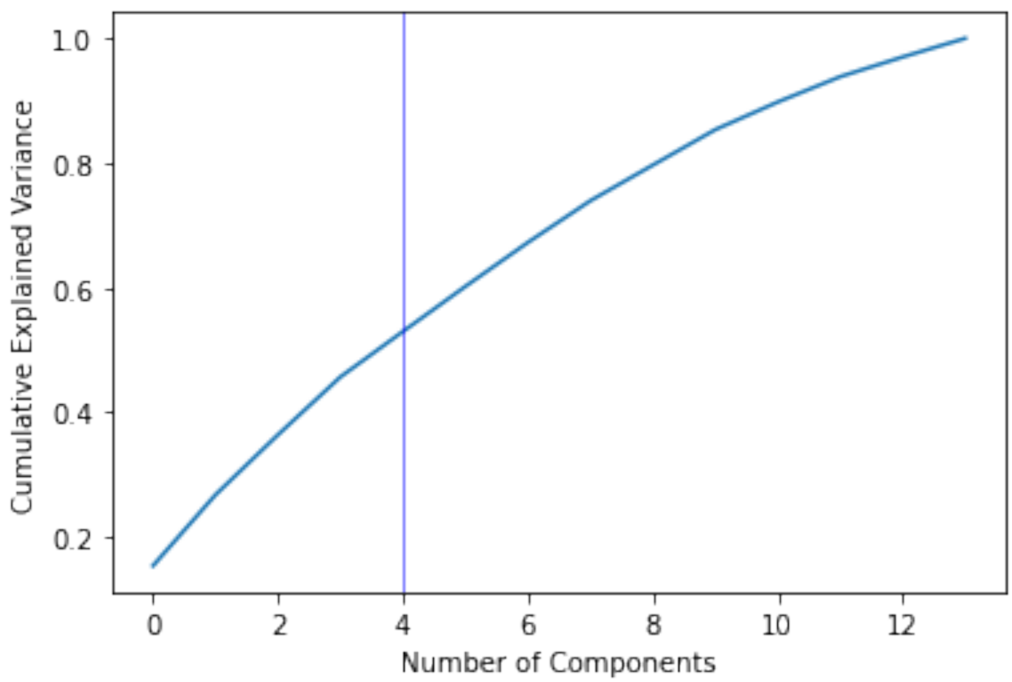
[5]:
```
from sklearn.preprocessing import StandardScaler
scalar = StandardScaler()
# fitting
scalar.fit(df)
df_std = scalar.transform(df)
```

[6]:
```
# run PCA
n_components = 10
pca = PCA(n_components=n_components)
pca_fit = pca.fit_transform(df_std)
df_pca = pd.DataFrame(data = pca_fit
                      , columns = ['PC '+ str(i+1) for i in␣
 ↪range(n_components)])
```

[7]:
```
# check the Screen plot
ax = plt.figure().gca()
ax.plot(pca.explained_variance_)
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.xlabel('Principal Component')
plt.ylabel('Eigenvalue')
plt.axhline(y=1, linewidth=1, color='b', alpha=0.5)
plt.title('Screen Plot of PCA: Component Eigenvalues')
plt.show()
```



[8]:
```
# check the variance explained from PCA
ax = plt.figure().gca()
pca = PCA().fit(df_std)
ax.plot(np.cumsum(pca.explained_variance_ratio_))
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.axvline(x=4, linewidth=1, color='b', alpha=0.5)
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()
```



[9]:
```
# check the loading for PCA elements
loadings = pca.components_
results = pd.DataFrame(loadings)
results.index=df.columns
print(results[0].sort_values(ascending=True))
```

```
lead_time                              -0.432807
no_of_previous_bookings_not_canceled   -0.429218
no_of_adults                           -0.293649
avg_price_per_room                     -0.120011
required_car_parking_space             -0.114405
arrival_year                           -0.111549
no_of_weekend_nights                   -0.006604
repeated_guest                          0.033542
no_of_previous_cancellations            0.154484
no_of_special_requests                  0.238295
no_of_week_nights                       0.248083
no_of_children                          0.283716
arrival_month                           0.353457
arrival_date                            0.391786
Name: 0, dtype: float64
```

[10]:
```
# run PCA with 1 component
from sklearn.decomposition import PCA
pca = PCA(n_components=1)
```

```
pca.fit(df_std)
x_pca = pca.transform(df_std)
```

[11]:
```
# output the csv file
x_pca_df = pd.DataFrame(x_pca)
x_pca_df.to_csv("PCA_dat.csv")
```

[ ]:
```
```

③

Tune_Logistic and NN model

March 3, 2023

1 Tunning Logistic and NN model

[36]:
```
# load package
import pandas as pd
import numpy as np
from sklearn.metrics import roc_auc_score
import warnings
warnings.filterwarnings('ignore')
```

[37]:
```
# load the data
anomoly  = False
# load data either by anomoly or not
if anomoly:
    X=pd.read_csv("https://raw.githubusercontent.com/KelvinYQC/
 ↪msia420PA_project/main/Data/with_anomaly.csv")
else:
    X=pd.read_csv("https://raw.githubusercontent.com/KelvinYQC/
 ↪msia420PA_project/main/Data/without_anomaly.csv")
y = X['booking_status']
X.drop(['booking_status'], axis = 1, inplace = True)
```

2 Baseline model– Logistic Regression

2.1 Modeling

[38]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=42)
X_train.shape
```

[38]:
```
(21708, 22)
```

[39]:
```
LR_model = LogisticRegression(random_state=0).fit(X_train, y_train)
```

[40]:
```
# score = correct predictions / total number of data
score = LR_model.score(X_test, y_test)
print(score)
```

```
0.8015524174693724
```

[41]:
```
y_pred = LR_model.predict(X_test)
# y_pred_prob = LR_model.predict_proba(X_test)
# y_pred_prob
# print(roc_auc_score(y, LR_model.predict_proba(X_test)[:, 1]))
```

[42]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.89      0.86      7287
           1       0.72      0.62      0.66      3406

    accuracy                           0.80     10693
   macro avg       0.78      0.75      0.76     10693
weighted avg       0.80      0.80      0.80     10693
```

2.2 Logistic regression with ridge

[43]:
```
from sklearn.linear_model import RidgeClassifierCV
LR_ridge = RidgeClassifierCV(alphas=[1e-3, 1e-2, 1e-1, 1]).fit(X_train, y_train)
LR_ridge.score(X_test, y_test)
```

[43]:
```
0.8013653792200505
```

[44]:
```
y_pred_ridge = LR_ridge.predict(X_test)
y_pred_ridge
```

[44]:
```
array([0, 0, 1, …, 0, 0, 0])
```

[45]:
```
print(classification_report(y_test, y_pred_ridge))
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86      7287
           1       0.73      0.59      0.65      3406

    accuracy                           0.80     10693
   macro avg       0.78      0.75      0.76     10693
weighted avg       0.80      0.80      0.80     10693
```

3 Second Model: Neural Network

[46]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train_standardize = scaler.fit_transform(X_train)
X_test_standardize = scaler.fit_transform(X_test)
```

[47]:
```
nn1 = MLPClassifier(solver='lbfgs',
                    # alpha=1e-5,
                    # hidden_layer_sizes=(6,),
                    random_state=123)
nn1.fit(X_train_standardize,y_train)
y_predNN = nn1.predict(X_test_standardize)
```

[48]:
```
print(classification_report(y_test,y_predNN))
```

```
              precision    recall  f1-score   support

           0       0.89      0.90      0.89      7287
           1       0.78      0.76      0.77      3406

    accuracy                           0.86     10693
   macro avg       0.83      0.83      0.83     10693
weighted avg       0.85      0.86      0.85     10693
```

[63]:
```
params = {'hidden_layer_sizes': [(30,),(50,),(70,),(100,)],
         'learning_rate_init': [0.0001, 0.001,0.01, 0.1,1]}
nn_model = MLPClassifier()
```

[64]:
```
from sklearn.experimental import enable_halving_search_cv # noqa
from sklearn.model_selection import HalvingRandomSearchCV

gs_nn1 = HalvingRandomSearchCV(
    nn_model, params, scoring="roc_auc", n_jobs=-1, factor=4, cv = 10
)
# gs_nn1 = GridSearchCV(nn_model,
#                       param_grid=params,
#                       scoring='roc_auc',
#                       cv=10)
```

[65]:
```
gs_nn1.fit(X_train_standardize,y_train)
print(gs_nn1.best_params_)
```

/opt/homebrew/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_perceptron.py:702:

```
packages/sklearn/neural_network/_multilayer_perceptron.py:702:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/lib/python3.10/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:702:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/lib/python3.10/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:702:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/lib/python3.10/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:702:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

```
{'learning_rate_init': 0.001, 'hidden_layer_sizes': (30,)}
```

[66]:
```
gs_knn_pred = gs_nn1.predict(X_test_standardize)
```

[67]:
```
print(classification_report(y_test,gs_knn_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      0.90      0.89      7287
           1       0.77      0.74      0.75      3406

    accuracy                           0.85     10693
   macro avg       0.82      0.82      0.82     10693
weighted avg       0.84      0.85      0.85     10693
```

[68]:
```
print('The final auc score for NN is: ')
print(round(roc_auc_score(y_test, gs_nn1.predict_proba(X_test_standardize)[:,
 ↪1]), 3))
```

```
The final auc score for NN is:
0.909
```

Tune_XGboostModel

March 3, 2023

1 XGboost model

[1]:
```
import xgboost as xgb
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV

from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import roc_auc_score
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
anomoly  = False
# load data either by anomoly or not
if anomoly:
    X=pd.read_csv("https://raw.githubusercontent.com/KelvinYQC/
 ↪msia420PA_project/main/Data/with_anomaly.csv")
else:
    X=pd.read_csv("https://raw.githubusercontent.com/KelvinYQC/
 ↪msia420PA_project/main/Data/without_anomaly.csv")
y = X['booking_status']
X.drop(['booking_status'], axis = 1, inplace = True)
```

[3]:
```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=1,␣
 ↪test_size=None, train_size=0.3)
```

[7]:
```
from sklearn.pipeline import Pipeline
from sklearn.experimental import enable_halving_search_cv # noqa
from sklearn.model_selection import HalvingRandomSearchCV

pipeline = Pipeline(steps=[("scaler", StandardScaler()), ("clf", xgb.
 ↪XGBClassifier())])
```

```
parameters = {
    'clf__learning_rate': [0.001, 0.01, 0.05, 0.1, None],
    'clf__max_depth': [2,5,7,10,15,30, None]
}

clf = HalvingRandomSearchCV(
    pipeline, parameters, scoring="roc_auc", n_jobs=-1, factor=4, cv = 10
)

# clf = RandomizedSearchCV(estimator=pipeline,
#                          param_distributions = parameters,
#                          n_iter=20,
#                          verbose=1,
#                          cv = 5,
#                          scoring = 'roc_auc')
clf_xgb = clf.fit(X_train, y_train)
print("Best parameters:", clf.best_params_)
```

```
Best parameters: {'clf__max_depth': 15, 'clf__learning_rate': None}
```

[8]:
```
from sklearn.metrics import roc_auc_score

y_pred = clf_xgb.predict(X_test)
classifier_score_xgb = roc_auc_score(y_test, y_pred)

print("The resulting roc auc score: ")
print(round(classifier_score_xgb,3))
```

```
The resulting roc auc score:
0.854
```

The AUC from XGboost model is 0.854.

Random_Forset_tuning

March 3, 2023

1 Random Forest Tunning

[16]:
```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
```

1.1 Random Forest

[17]:
```
# Tune model without anomoly
anomoly  = False

# load data either by anomoly or not
if anomoly:
    X=pd.read_csv("https://raw.githubusercontent.com/KelvinYQC/
 ↪msia420PA_project/main/Data/with_anomaly.csv")
else:
    X=pd.read_csv("https://raw.githubusercontent.com/KelvinYQC/
 ↪msia420PA_project/main/Data/without_anomaly.csv")
y = X['booking_status']
X.drop(['booking_status'], axis = 1, inplace = True)
```

[19]:
```
X.shape
```

[19]: (32401, 22)

[13]:
```
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
max_features = ['auto', 'sqrt']
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
min_samples_split = [2, 5, 10]
min_samples_leaf = [1, 2, 4]
bootstrap = [True]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
```

```python
                'min_samples_leaf': min_samples_leaf,
                'bootstrap': bootstrap}
rf = RandomForestClassifier()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions =␣
↪random_grid, n_iter = 100, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
rf_random.fit(X, np.ravel(y))
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
[CV] END bootstrap=True, max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=800; total time=  16.6s
[CV] END bootstrap=True, max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=800; total time=  16.9s
[CV] END bootstrap=True, max_depth=70, max_features=auto, min_samples_leaf=2,
min_samples_split=2, n_estimators=800; total time=  15.0s
[CV] END bootstrap=True, max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=800; total time=  16.6s
[CV] END bootstrap=True, max_depth=40, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=1800; total time=  33.7s
[CV] END bootstrap=True, max_depth=40, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=1800; total time=  34.0s
[CV] END bootstrap=True, max_depth=40, max_features=sqrt, min_samples_leaf=2,
min_samples_split=2, n_estimators=1800; total time=  34.1s
[CV] END bootstrap=True, max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=2000; total time=  42.1s
[CV] END bootstrap=True, max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=2000; total time=  42.5s
[CV] END bootstrap=True, max_depth=None, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=2000; total time=  42.6s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=4,
min_samples_split=5, n_estimators=200; total time=   3.3s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=4,
min_samples_split=5, n_estimators=200; total time=   3.3s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=4,
min_samples_split=5, n_estimators=200; total time=   3.4s
[CV] END bootstrap=True, max_depth=70, max_features=auto, min_samples_leaf=2,
min_samples_split=2, n_estimators=800; total time=  15.3s
[CV] END bootstrap=True, max_depth=70, max_features=auto, min_samples_leaf=2,
min_samples_split=2, n_estimators=800; total time=  15.0s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=2,
min_samples_split=10, n_estimators=600; total time=  11.4s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=2,
min_samples_split=10, n_estimators=600; total time=  11.8s
[CV] END bootstrap=True, max_depth=100, max_features=auto, min_samples_leaf=2,
min_samples_split=10, n_estimators=600; total time=  11.7s
```

```
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=1200; total time=  31.9s
[CV] END bootstrap=True, max_depth=110, max_features=auto, min_samples_leaf=2,
min_samples_split=5, n_estimators=400; total time=  10.7s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=1,
min_samples_split=10, n_estimators=200; total time=   5.0s
[CV] END bootstrap=True, max_depth=110, max_features=auto, min_samples_leaf=2,
min_samples_split=5, n_estimators=400; total time=  10.3s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=1,
min_samples_split=10, n_estimators=200; total time=   5.3s
[CV] END bootstrap=True, max_depth=110, max_features=auto, min_samples_leaf=2,
min_samples_split=5, n_estimators=400; total time=  10.4s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=1400; total time=  40.5s
[CV] END bootstrap=True, max_depth=100, max_features=sqrt, min_samples_leaf=1,
min_samples_split=10, n_estimators=200; total time=   5.3s
[CV] END bootstrap=True, max_depth=40, max_features=sqrt, min_samples_leaf=1,
min_samples_split=10, n_estimators=200; total time=   5.0s
[CV] END bootstrap=True, max_depth=40, max_features=sqrt, min_samples_leaf=1,
min_samples_split=10, n_estimators=200; total time=   5.0s
[CV] END bootstrap=True, max_depth=20, max_features=sqrt, min_samples_leaf=1,
min_samples_split=2, n_estimators=1200; total time=  32.0s
[CV] END bootstrap=True, max_depth=40, max_features=sqrt, min_samples_leaf=1,
min_samples_split=10, n_estimators=200; total time=   5.0s
[CV] END bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=5, n_estimators=1400; total time=  35.4s
[CV] END bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=5, n_estimators=1400; total time=  35.5s
[CV] END bootstrap=True, max_depth=20, max_features=auto, min_samples_leaf=1,
min_samples_split=5, n_estimators=1400; total time=  35.2s
[CV] END bootstrap=True, max_depth=None, max_features=auto, min_samples_leaf=4,
min_samples_split=10, n_estimators=1000; total time=  22.5s
[CV] END bootstrap=True, max_depth=None, max_features=auto, min_samples_leaf=4,
min_samples_split=10, n_estimators=1000; total time=  22.0s
[CV] END bootstrap=True, max_depth=30, max_features=auto, min_samples_leaf=1,
min_samples_split=2, n_estimators=1600; total time=  42.2s
[CV] END bootstrap=True, max_depth=30, max_features=auto, min_samples_leaf=1,
min_samples_split=2, n_estimators=1600; total time=  42.6s
[CV] END bootstrap=True, max_depth=30, max_features=auto, min_samples_leaf=1,
min_samples_split=2, n_estimators=1600; total time=  42.6s
[CV] END bootstrap=True, max_depth=None, max_features=auto, min_samples_leaf=4,
min_samples_split=10, n_estimators=1000; total time=  15.9s
```

[13]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=100,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True],
                                        'max_depth': [10, 20, 30, 40, 50, 60,
```

```
                                                      70, 80, 90, 100, 110,
                                                      None],
                                        'max_features': ['auto', 'sqrt'],
                                        'min_samples_leaf': [1, 2, 4],
                                        'min_samples_split': [2, 5, 10],
                                        'n_estimators': [200, 400, 600, 800,
                                                         1000, 1200, 1400, 1600,
                                                         1800, 2000]},
                   random_state=42, verbose=2)
```

[14]:

```
rf_random.best_params_
```

[14]:

```
{'n_estimators': 1600,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'auto',
 'max_depth': 30,
 'bootstrap': True}
```

[ ]:

```
```

CV_Model_Selection

March 3, 2023

1 Cross Validation on All Models

[1]:
```
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import roc_auc_score
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression, RidgeClassifierCV
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RepeatedStratifiedKFold
```

[2]:
```
anomoly  = False

# load data either by anomoly or not
if anomoly:
    X=pd.read_csv("https://raw.githubusercontent.com/KelvinYQC/
 ↪msia420PA_project/main/Data/with_anomaly.csv")
else:
    X=pd.read_csv("https://raw.githubusercontent.com/KelvinYQC/
 ↪msia420PA_project/main/Data/without_anomaly.csv")
y = X['booking_status']
X.drop(['booking_status'], axis = 1, inplace = True)
```

[3]:
```
# train test split
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=1,␣
 ↪test_size=None, train_size=0.3)
```

1.1 Stratified K-fold CV

Since we have imbalanced dataset, we would use stratified K-fold cross validation to test the model. In such manner, we would ensure each fold would have a representative proportion of both the majority class and minority class.

[4]:
```
# Run CV for 5 folds
models = {
    # Baseline model
    "Naive Bayes" : GaussianNB(),
    # Linear model
    "Logistic Regression": LogisticRegression(),
    "RidgeClassifierCV": RidgeClassifierCV(alphas=[1e-3, 1e-2, 1e-1, 1]),
    # Non-linear model
    "Neural_Network": MLPClassifier(hidden_layer_sizes = 30,learning_rate_init␣
 ↪= 0.001),
    "svc": SVC(C = 10, gamma = 0.1, kernel = 'poly'),
    # Tree based model
    "DecisionTreeClassifier":DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(n_estimators =␣
 ↪800,min_samples_split= 5,min_samples_leaf=1,max_features="sqrt", max_depth=␣
 ↪100),
    "XGBoost": xgb.XGBClassifier(max_depth=15, learning_rate=None),
}

# Create folds
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=10)

# Run for k folds to find best model.
cv_scores = {}
for name, model in models.items():
    pipeline = Pipeline(steps=[("scaler", StandardScaler()), ("clf", model)])
    cur_score = np.mean(cross_val_score(pipeline, X_train, y_train, cv=cv,␣
 ↪scoring = "roc_auc"))
    cv_scores[name] = cur_score
print(cv_scores)
```

```
{'Naive Bayes': 0.7934784789992383, 'Logistic Regression': 0.8554749144324799,
'RidgeClassifierCV': 0.8553588347600939, 'Neural_Network': 0.8930518188193989,
'svc': 0.8825549709734914, 'DecisionTreeClassifier': 0.8224114514980566, 'Random
Forest': 0.9373128209043726, 'XGBoost': 0.9337680717445344}
```

[6]:
```
for i , v in sorted(cv_scores.items(), key = lambda x:x[1]):
    print((i,v))
```

```
('Naive Bayes', 0.7934784789992383)
('DecisionTreeClassifier', 0.8224114514980566)
('RidgeClassifierCV', 0.8553588347600939)
```

```
('Logistic Regression', 0.8554749144324799)
('svc', 0.8825549709734914)
('Neural_Network', 0.8930518188193989)
('XGBoost', 0.9337680717445344)
('Random Forest', 0.9373128209043726)
```

1.2 Final model

1.2.1 Fit the final model

[4]:
```
final_model = RandomForestClassifier(n_estimators = 800,min_samples_split=
 ↪5,min_samples_leaf=1,max_features="sqrt", max_depth= 100)
final_pip = Pipeline(steps=[("scaler", StandardScaler()), ("clf", final_model)])

final_pip.fit(X_train, y_train)
final_model = final_pip.fit(X_train, y_train)
y_pred = final_model.predict(X_test)
classifier_score_xgb = roc_auc_score(y_test, y_pred)

print("The final roc auc score: ")
print(round(classifier_score_xgb,3))
```

```
The final roc auc score:
0.859
```

[7]:
```
from sklearn.metrics import precision_score
pre_score = precision_score(y_test, y_pred)
print("The final precision score: ")
print(round(pre_score,3))
```

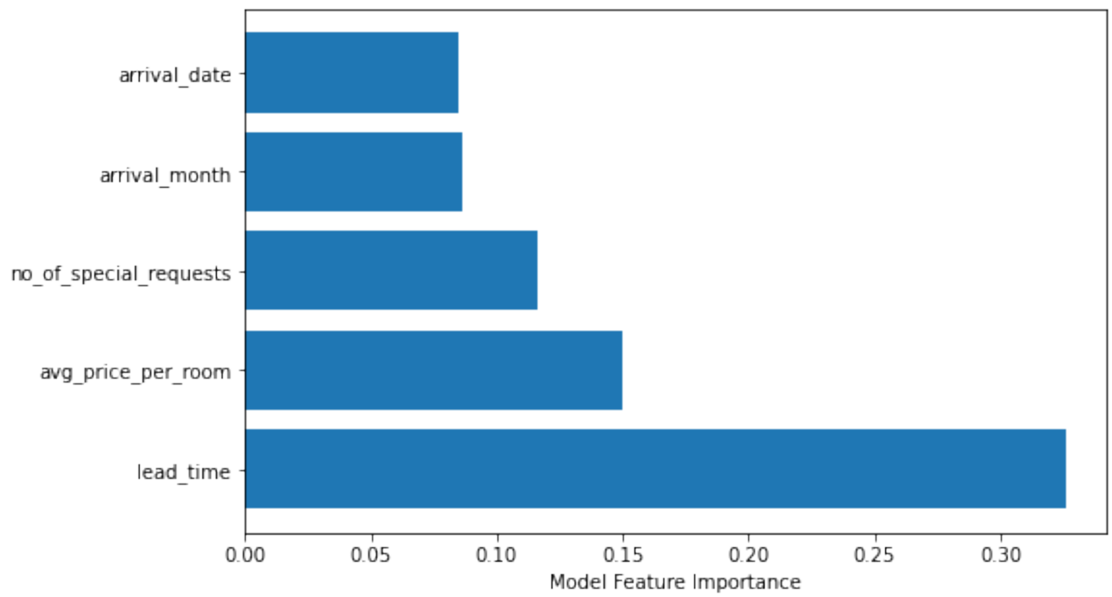
```
The final precision score:
0.863
```

1.2.2 Variable Importance Plot

[9]:
```
from xgboost import plot_importance
from matplotlib import pyplot as plt
model = final_model["clf"]
sorted_idx = model.feature_importances_.argsort()[::-1]
top_n = 5  # Specify the number of top variables you want to display

plt.figure(figsize=(8, 5))  # Set the figure size
plt.barh(range(top_n), model.feature_importances_[sorted_idx][:top_n],
 ↪align='center')
plt.yticks(range(top_n), X_test.columns[sorted_idx][:top_n])
plt.xlabel("Model Feature Importance")

plt.show()
```


arrival_date
arrival_month
no_of_special_requests
avg_price_per_room
lead_time
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Model Feature Importance

[ ]: