# Questions

February 4, 2022

## 1 Data Science Challenge

[ ]:
```
# To install packages that are not installed by default, uncomment the last two
 ↪lines
# of this cell and replace <package list> with a list of necessary packages.
# This will ensure the notebook has all the dependencies and works everywhere.

#import sys
#!{sys.executable} -m pip install <package list>
```

[144]:
```
#Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from sklearn.model_selection import cross_validate
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report,
 ↪roc_auc_score, auc, accuracy_score, roc_curve

pd.set_option("display.max_columns", 101)
```

### 1.1 Data Description

| Column | Description |
|---|---|
| id | The unique ID assigned to every hotel. |
| region | The region in which the hotel is located.. |
| latitude | The latitude of the hotel. |
| longitude | The longitude of the hotel. |
| accommodation_type | The type of accommodation offered by the hotel. For example: Private room, Entire house/apt, etc. |

| Column | Description |
| --- | --- |
| cost | The cost of booking the hotel for one night. (in $$) |
| minimum_nights | The minimum number of nights stay required. |
| number_of_reviews | The number of reviews accumulated by the hotel. |
| reviews_per_month | The average number of reviews received by the hotel per month. |
| owner_id | The unique ID assigned to every owner. An owner can own multiple hotels. |
| owned_hotels | The number of hotels owned by the owner. |
| yearly_availability | It indicates if the hotel accepts bookings around the year. Values are 0 (not available for 365 days in a year) and 1 (available for 365 days in a year). |

## 1.2 Data Wrangling & Visualization

[4]:
```python
# Dataset is already loaded below
data = pd.read_csv("train.csv")
```

[5]:
```python
data.head()
```

[5]:
```
      id     region  latitude  longitude accommodation_type  cost  \
0  13232  Manhattan  40.71854  -74.00439    Entire home/apt   170
1    246   Brooklyn  40.64446  -73.95030    Entire home/apt    65
2  19091     Queens  40.78573  -73.81062       Private room    85
3  34305  Manhattan  40.73863  -73.98002       Private room   210
4    444  Manhattan  40.82426  -73.94630        Shared room    75

   minimum_nights  number_of_reviews  reviews_per_month   owner_id  \
0               5                  7               0.56     929983
1               3                238               2.30     281764
2               1                  0                NaN   19923341
3              30                  0                NaN  200380610
4               3                 38               0.42     745069

   owned_hotels  yearly_availability
0             1                    0
1             1                    0
2             1                    1
3            65                    1
4             3                    1
```

[6]:
```python
#Explore columns
data.columns
```

[6]:
```
Index(['id', 'region', 'latitude', 'longitude', 'accommodation_type', 'cost',
       'minimum_nights', 'number_of_reviews', 'reviews_per_month', 'owner_id',
       'owned_hotels', 'yearly_availability'],
      dtype='object')
```

[7]:
```
#Description
data.describe()
```

[7]:
```
                 id     latitude    longitude         cost  minimum_nights  \
count   2870.000000  2870.000000  2870.000000  2870.000000     2870.000000
mean   26760.657143    40.731224   -73.950158   195.943206       11.530314
std    14140.930062     0.054942     0.049745   406.184714       37.972339
min        0.000000    40.507080   -74.242850    10.000000        1.000000
25%    15931.750000    40.692462   -73.984003    75.000000        1.000000
50%    28946.500000    40.728250   -73.956720   120.000000        3.000000
75%    38478.500000    40.762658   -73.934202   200.000000        6.000000
max    48893.000000    40.898730   -73.721730  9999.000000      999.000000

       number_of_reviews  reviews_per_month      owner_id  owned_hotels  \
count        2870.000000        2194.000000  2.870000e+03   2870.000000
mean           16.315331           1.157502  7.202195e+07      8.411498
std            32.481722           1.355028  8.076516e+07     27.105522
min             0.000000           0.010000  2.787000e+03      1.000000
25%             1.000000           0.240000  7.388002e+06      1.000000
50%             4.000000           0.650000  3.352708e+07      1.000000
75%            16.000000           1.530000  1.207625e+08      3.000000
max           395.000000          10.370000  2.738123e+08    327.000000

       yearly_availability
count          2870.000000
mean              0.498606
std               0.500085
min               0.000000
25%               0.000000
50%               0.000000
75%               1.000000
max               1.000000
```

[52]:
```
## Printing the unique regions in the dataset
np.unique(data['region'])
```

[52]:
```
array(['Bronx', 'Brooklyn', 'Manhattan', 'Queens', 'Staten Island'],
      dtype=object)
```

[53]:
```
## Sanity check if all the hotels are unique
len(data), len(np.unique(data['id']))
```

[53]: (2870, 2870)

[54]:
```
## Checking if the owners are unique
len(data), len(np.unique(data['owner_id']))
```

[54]: (2870, 2371)

### 1.2.1 Dataset Insights:

-> Since the latitude and longitude values do not vary much, the data is from a specific region. The names of the regions suggest somewhere close to New York City. We'll keep the latitudes and logitudes to be on the safer side and not lose out on intricate details. -> The cost potentially with the amount of variation could be an important factor for predicting the outcome. At the same time, the maximum value suggest that the data could have some anomalous values, which histograms will help in figuring out -> The reviews per month has 676 missing values. At this point, since the business problem we are trying to solve of predicting if the hotel accepts bookings throughout the year, the average value per month is not important because it does not take into account the skewness of the monthly data, where lets say the hotel does not take bookings in July, we can't figure that out. We can think of dropping this column. Also, statistically the reviews_per_month feature does not have a lot of variation in values, hence intuitively might not contribute to the predictability of our algorithm -> Assuming the train and test data has no overlapsof hotels, we can safely remove the id column -> The mean of the yearly_availability column shows that the data is balanced at this point with a prevalence of almost 50%

[55]:
```
filtered_data=data.drop(columns=['id', 'reviews_per_month'], axis=1)
filtered_data.columns
```

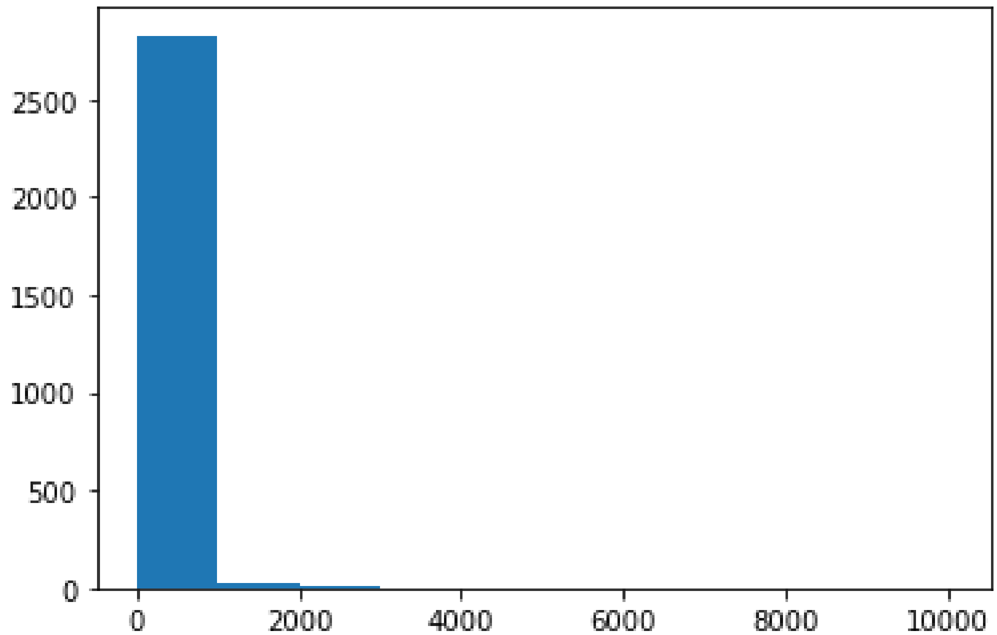
[55]:
```
Index(['region', 'latitude', 'longitude', 'accommodation_type', 'cost',
       'minimum_nights', 'number_of_reviews', 'owner_id', 'owned_hotels',
       'yearly_availability'],
      dtype='object')
```

## 1.3 Analyze cost and minimum_nights

**Analyze Cost First**

[56]:
```
# Plot histogram for cost values
plt.hist(filtered_data['cost'])
```

[56]:
```
(array([2.829e+03, 2.600e+01, 6.000e+00, 1.000e+00, 4.000e+00, 0.000e+00,
        2.000e+00, 1.000e+00, 0.000e+00, 1.000e+00]),
 array([  10. , 1008.9, 2007.8, 3006.7, 4005.6, 5004.5, 6003.4, 7002.3,
        8001.2, 9000.1, 9999. ]),
 <a list of 10 Patch objects>)
```




[57]:
```
## The data looks skewed with majority of the values in the the [0,1000] range
↪from the histogram.
## We know the data is an independent sample of hotels, asssume the population
↪mean of costs would follow a normal distribution,
# let's take two standard deviations to the right of the mean as it showcases
↪in the right-tailed Z-tests.

cost_maximum_value=195.94+2*406.184
print (cost_maximum_value)

count_of_hotels_in_range=0

for hotel_cost in list(filtered_data['cost']):
    if hotel_cost<=value:
        count_of_hotels_in_range+=1

print (len(filtered_data['cost']), count_of_hotels_in_range)
```

```
1008.308
2870 2829
```

[58]:
```
# We now get rid of anomalous values in the dataset by just losing out on 41
↪values.
#Filtering the data accordingly
```

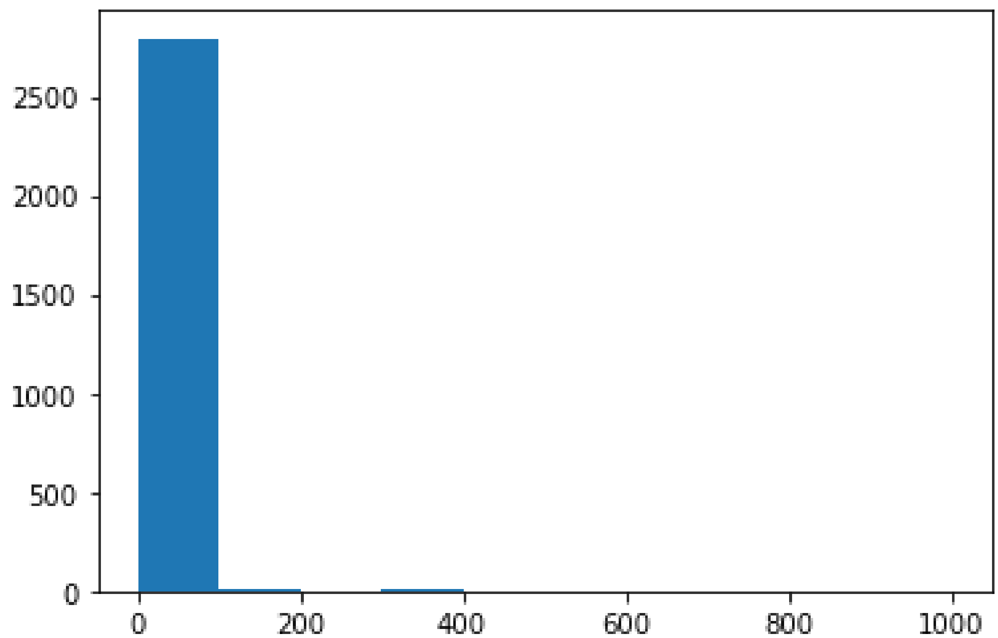
```
filtered_data=filtered_data.loc[filtered_data['cost']<=cost_maximum_value]
print (len(filtered_data), np.min(filtered_data['cost']), np.
 ↪max(filtered_data['cost']))
```

```
2829 10 1002
```

### 1.3.1 Analyze minimum_nights now

[60]:
```
plt.hist(filtered_data['minimum_nights'])
```

[60]:
```
(array([2.794e+03, 1.700e+01, 4.000e+00, 1.100e+01, 1.000e+00, 1.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 1.000e+00]),
 array([  1. , 100.8, 200.6, 300.4, 400.2, 500. , 599.8, 699.6, 799.4,
        899.2, 999. ]),
 <a list of 10 Patch objects>)
```




[61]:
```
np.unique(filtered_data['minimum_nights'])
```

[61]:
```
array([  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,  13,
        14,  15,  16,  19,  20,  21,  25,  26,  27,  28,  29,  30,  31,
        32,  35,  40,  45,  50,  53,  56,  60,  90,  99, 120, 150, 180,
       186, 200, 210, 225, 300, 360, 365, 366, 370, 480, 500, 999])
```

[63]:
```
# Use same normal distribution logic to filter out samples
```

```
minimum_nights_maximum_value=11.53+2*37.97
print (minimum_nights_maximum_value)

count_of_hotels_in_range=0

for hotel_nights in list(filtered_data['minimum_nights']):
    if hotel_nights<=minimum_nights_maximum_value:
        count_of_hotels_in_range+=1

print (len(filtered_data['minimum_nights']), count_of_hotels_in_range)


# We now get rid of anomalous values for minimum nights column too, just losing␣
↪out on 52 datapoints.
# Let's apply this filter on the filtered_data dataframe
```

```
87.47
2829 2777
```

[64]:
```
filtered_data=filtered_data.
↪loc[filtered_data['minimum_nights']<=minimum_nights_maximum_value]
print (len(filtered_data), np.min(filtered_data['minimum_nights']), np.
↪max(filtered_data['minimum_nights']))
```

```
2777 1 60
```

### 1.4 Change string to float for regions & accomodation_type

[162]:
```
def get_region_id_column(filtered_data):
    unique_regions=list(np.unique(filtered_data['region']))
    unique_region_id_mapping={}
    for idx, i in enumerate(unique_regions):
        unique_region_id_mapping[i]=idx

    print (unique_region_id_mapping)

    region_id=[]
    for i in filtered_data['region']:
        region_id.append(unique_region_id_mapping[i])
    filtered_data['region_id']=region_id

    filtered_data=filtered_data.drop(columns=['region'], axis=1)
    filtered_data.columns

    return filtered_data
```

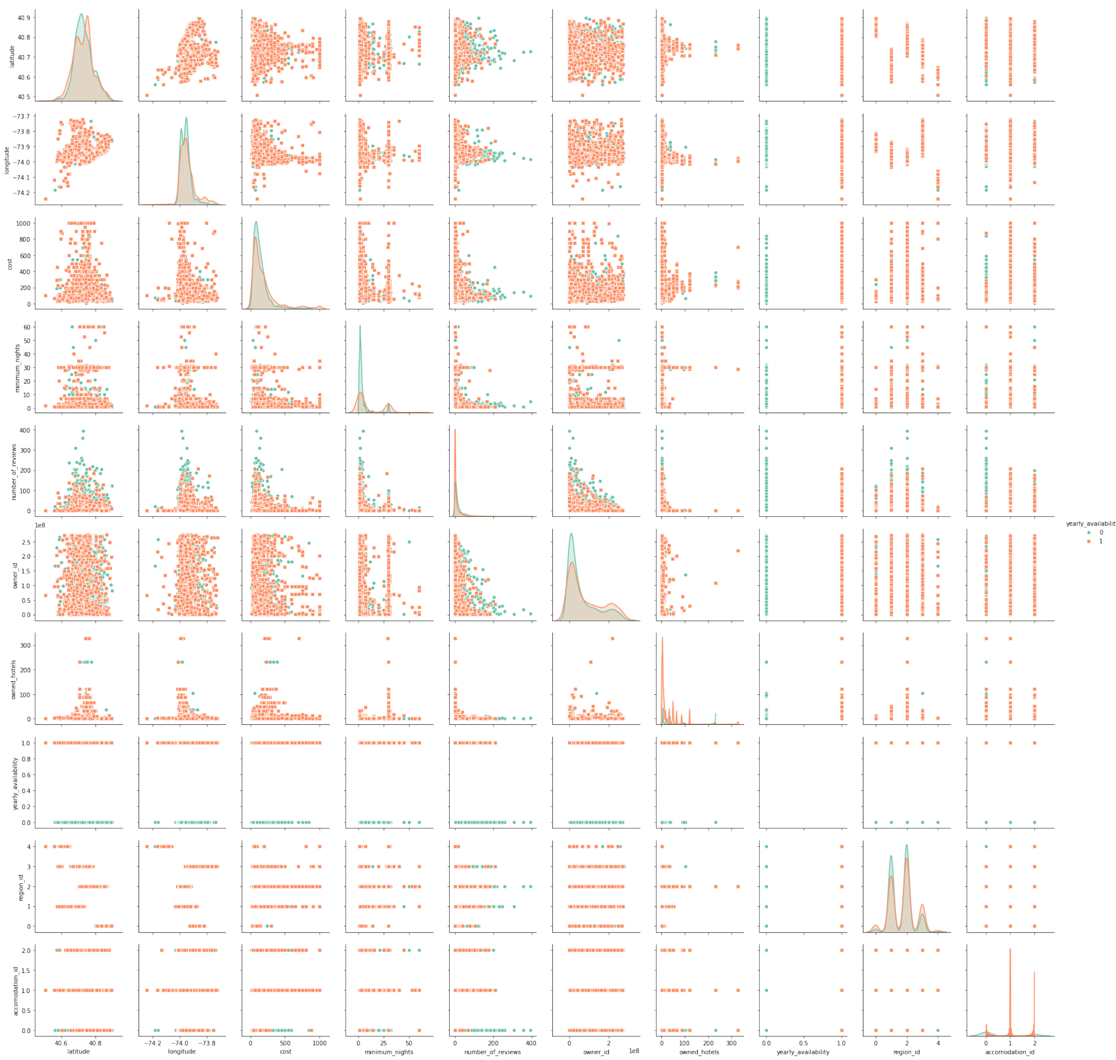
```
filtered_data=get_region_id_column(filtered_data)
```

[163]:
```
def get_accomodation_id_column(filtered_data):
    unique_accomodations=list(np.unique(filtered_data['accommodation_type']))
    unique_accomodation_id_mapping={}
    for idx, i in enumerate(unique_accomodations):
        unique_accomodation_id_mapping[i]=idx

    print (unique_accomodation_id_mapping)

    unique_accomodation_id=[]
    for i in filtered_data['accommodation_type']:
        unique_accomodation_id.append(unique_accomodation_id_mapping[i])
    filtered_data['accomodation_id']=unique_accomodation_id

    filtered_data=filtered_data.drop(columns=['accommodation_type'], axis=1)
    filtered_data.columns

    return filtered_data

filtered_data=get_acccomodation_id_column(filtered_data)
```

[87]:
```
sns.pairplot(filtered_data, kind="scatter", hue="yearly_availability",␣
↪markers=["o", "s"], palette="Set2")
```

```
/opt/conda/lib/python3.7/site-packages/statsmodels/nonparametric/kde.py:487:
RuntimeWarning: invalid value encountered in true_divide
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/opt/conda/lib/python3.7/site-packages/statsmodels/nonparametric/kdetools.py:34:
RuntimeWarning: invalid value encountered in double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
/opt/conda/lib/python3.7/site-packages/statsmodels/nonparametric/kde.py:487:
RuntimeWarning: invalid value encountered in true_divide
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/opt/conda/lib/python3.7/site-packages/statsmodels/nonparametric/kdetools.py:34:
RuntimeWarning: invalid value encountered in double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
```

[87]:
```
<seaborn.axisgrid.PairGrid at 0x7f915f3f1850>
```




[ ]:
```
# When we look at the above pairplot as a whole in majority of feature plots␣
↪there we see two clusters of orange and green depicting our classes
# The data looks more separable at this point.
```

### 1.5 Visualization, Modeling, Machine Learning

Build a model that categorizes hotels on the basis of their yearly availability. Identify how different features influence the decision. Please explain the findings effectively to technical and non-technical audiences using comments and visualizations, if appropriate. - **Build an optimized model that effectively solves the business problem. - The model will be evaluated on the basis of Accuracy. - Read the test.csv file and prepare features for testing.**

[88]:
```
training_data=filtered_data
training_data.head()
```

[89]:
```
print ("Prevalence:", sum(training_data['yearly_availability'])/
↪len(training_data)) #Looks balanced
```

```
Prevalence: 0.4836154123154483
```

[90]:
```
outcome=training_data['yearly_availability']
training_data=training_data.drop(['yearly_availability'], axis=1)
```

### 1.6 Experimentation with hyperparameter grid search using cross-validation

Choosing optimal hyperparamters is an important part of training the machine learning model. What we are doing here is with a set of specified values, the model is running on combinations of all of them and then will output the optimal set of hyperparameters for our dataset

Cross-validation essentially means to divide the dataset into k-sets and hold one out in each new model training iteration. The benefit of doing that is we know if the model is not succeeding by chance on our splits

Reason for choosing Random Forests: -> Ensemble of decision trees (Bagging - End up providing the best combination of trees) -> Reduced overfitting compared to decision trees -> Explainable as compared to paramteric models like Logistic Regression, SVMs, etc. -> Known to have performed quite well on tabular datasets and hence adopted for widespread use in the industry -> Handles missing values and continuous and categorical values quite well

[92]:
```
rfc=RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [100, 1000],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [1,2,3,4,5,6,7,8,9,10],
    'criterion' :['gini', 'entropy']
}

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv=4,␣
↪refit='accuracy')

CV_rfc.fit(training_data, outcome)
print (CV_rfc.best_params_)
```

```
{'criterion': 'entropy', 'max_depth': 8, 'max_features': 'auto', 'n_estimators':
1000}
```

### 1.7 Let's make stratified splits

[96]:
```
x_train, x_val, y_train, y_val = train_test_split(training_data, outcome,␣
↪test_size=0.25, random_state=42)
```

## 1.8 Train model

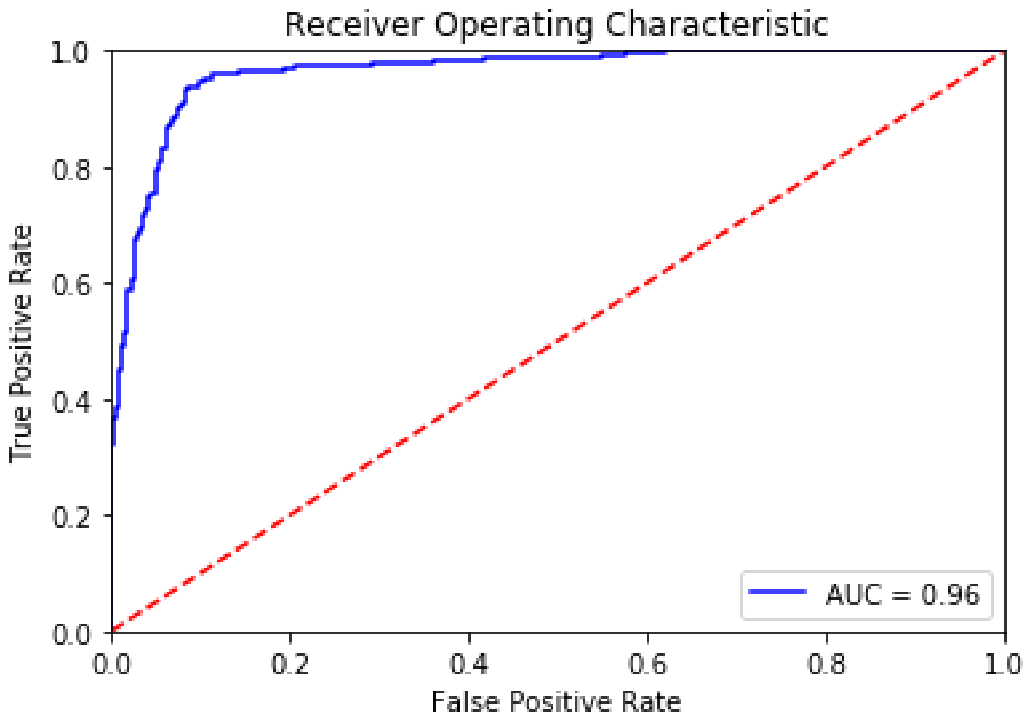
[97]:
```
rfc_model=RandomForestClassifier(random_state=42, criterion='entropy',␣
↪max_depth=8, max_features='auto', n_estimators=1000, class_weight='balanced')
rfc_model.fit(x_train, y_train)
```

[97]:
```
RandomForestClassifier(bootstrap=True, class_weight='balanced',
                       criterion='entropy', max_depth=8, max_features='auto',
                       max_leaf_nodes=None, min_impurity_decrease=0.0,
                       min_impurity_split=None, min_samples_leaf=1,
                       min_samples_split=2, min_weight_fraction_leaf=0.0,
                       n_estimators=1000, n_jobs=None, oob_score=False,
                       random_state=42, verbose=0, warm_start=False)
```

## 1.9 Run Predictions on the Validation set

[141]:
```
predicted_probabilities = rfc_model.predict_proba(x_val)
```

[145]:
```
fpr, tpr, threshold = roc_curve(y_val, predicted_probabilities [:,1])
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




[146]:
```
threshold=0.55    # Threshold chosen on the basis of the ROC curve to get␣
↪maximum accuracy
predictions = (predicted_probabilities [:,1] >= threshold).astype('int')
```

[147]:
```
Y_test=list(y_val)
Y_pred=list(predictions)

print("Confusion Matrix:")
print (confusion_matrix(Y_test,Y_pred))

print("Classification report:")
print(classification_report(Y_test,Y_pred,target_names=['Does not accept 365␣
↪days', 'Accepts 365 days']))

print ("Acuracy:", accuracy_score(Y_test, Y_pred))
```

```
Confusion Matrix:
[[301  35]
 [ 17 342]]
Classification report:
                         precision    recall  f1-score   support

Does not accept 365 days       0.95      0.90      0.92       336
        Accepts 365 days       0.91      0.95      0.93       359
```

```
              accuracy                           0.93       695
             macro avg       0.93      0.92      0.93       695
          weighted avg       0.93      0.93      0.93       695

Acuracy: 0.9251798561151079
```

### 1.10 Predicting on train set to check if the model is overfitting

[148]:
```python
predicted_probabilities = rfc_model.predict_proba(x_train)

predictions = (predicted_probabilities [:,1] >= threshold).astype('int')
```

[149]:
```python
Y_test=list(y_train)
Y_pred=list(predictions)

print("Confusion Matrix:")
print (confusion_matrix(Y_test,Y_pred))

print("Classification report:")
print(classification_report(Y_test,Y_pred,target_names=['Does not accept 365␣
 ↪days', 'Accepts 365 days']))

print ("Acuracy:", accuracy_score(Y_test, Y_pred))
```

```
Confusion Matrix:
[[1026   72]
 [  24  960]]
Classification report:
                          precision    recall  f1-score   support

Does not accept 365 days       0.98      0.93      0.96      1098
        Accepts 365 days       0.93      0.98      0.95       984

                accuracy                           0.95      2082
               macro avg       0.95      0.96      0.95      2082
            weighted avg       0.95      0.95      0.95      2082

Acuracy: 0.9538904899135446
```

**The training and validation sets do not have much of a difference in their metrics. The model seems to have fitted optimally. Additional overfitting can only be checked on an out-of-sample test set**

**Training Accuracy: 0.95**

**Validation Accuracy: 0.93**

[ ]:
```
#Loading Test data
test_data=pd.read_csv('test.csv')
test_data.head()
```

## 2 Clean CSV

[160]:
```
filtered_test_data=test_data.drop(columns=['reviews_per_month'], axis=1)
filtered_test_data.columns
```

[160]:
```
Index(['id', 'region', 'latitude', 'longitude', 'accommodation_type', 'cost',
       'minimum_nights', 'number_of_reviews', 'owner_id', 'owned_hotels'],
      dtype='object')
```

[164]:
```
filtered_test_data=get_region_id_column(filtered_test_data)
filtered_test_data=get_accomodation_id_column(filtered_test_data)
```

```
{'Bronx': 0, 'Brooklyn': 1, 'Manhattan': 2, 'Queens': 3, 'Staten Island': 4}
{'Entire home/apt': 0, 'Private room': 1, 'Shared room': 2}
```

[165]:
```
filtered_test_data.head()
```

[165]:
```
      id  latitude  longitude  cost  minimum_nights  number_of_reviews  \
0  19215  40.70912  -73.94513   135               2                 22
1  36301  40.57646  -73.96641    69               2                  8
2  40566  40.76616  -73.98228   225              30                  0
3  33694  40.77668  -73.94587   125              30                  9
4  28873  40.80279  -73.94450    43               1                 13

    owner_id  owned_hotels  region_id  accomodation_id
0    4360212             1          1                2
1  181356989             2          1                0
2   13773574            12          2                1
3    6788748             1          2                2
4  105061915             2          2                0
```

[171]:
```
test_predicted_probabilities = rfc_model.predict_proba(filtered_test_data.
 ↪drop(['id'], axis=1))
```

[172]:
```
test_predictions = (predicted_probabilities [:,1] >= threshold).astype('int')
```

[175]:
```
positive_predictions=sum(test_predictions)/len(test_predictions)
print (positive_predictions)
```

```
## The positive prediction prevalence atleast matches the prevalence of our␣
↪training dataset.
## This does not ensure accuracy of the model, but atleast tells us the␣
↪predictions are not going completely random
```

0.5027855153203342

**Highlight the most important features of the model for management.**

**Task:**

- **Visualize the top 20 features and their feature importance.**

[150]:
```
importances = list(rfc_model.feature_importances_)

feature_list = training_data.columns

feature_importances = [(feature, round(importance*100, 2)) for feature,␣
↪importance in zip(feature_list, importances)]

feature_importances = sorted(feature_importances, key = lambda x: x[1], reverse␣
↪= True)
```

[153]:
```
ft_imp_mapping = {}
for i in feature_importances:
    ft_imp_mapping[i[0]] = i[1]
```

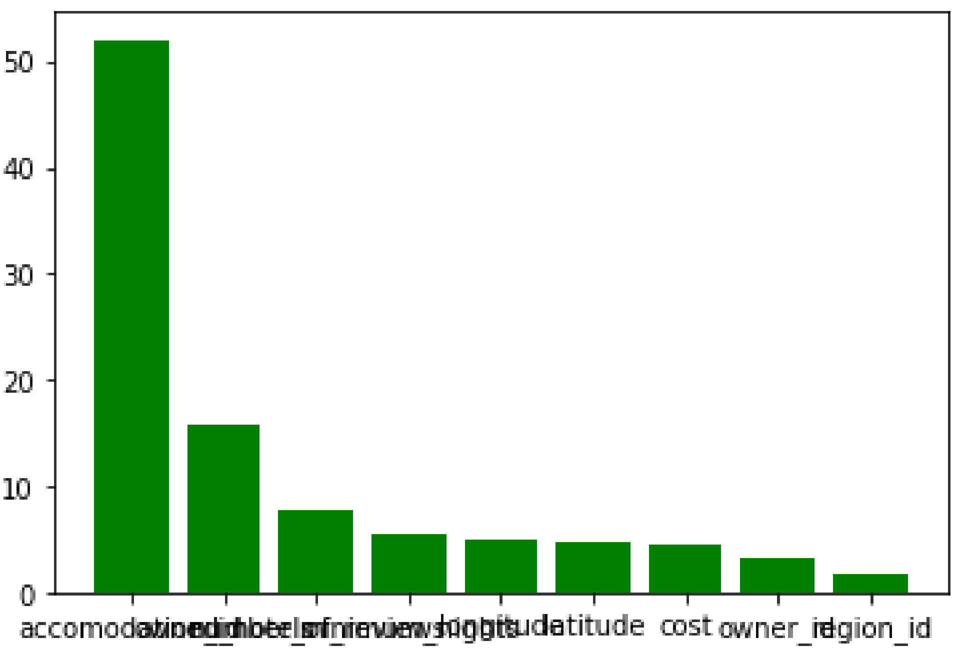
[154]:
```
pd.DataFrame(ft_imp_mapping, index=["Feature Importance (%)"]).T
```

[154]:

| | Feature Importance (%) |
|---|---|
| accomodation_id | 51.99 |
| owned_hotels | 15.69 |
| number_of_reviews | 7.65 |
| minimum_nights | 5.57 |
| longitude | 4.92 |
| latitude | 4.69 |
| cost | 4.58 |
| owner_id | 3.29 |
| region_id | 1.61 |

[159]:
```
plt.bar(ft_imp_mapping.keys(), ft_imp_mapping.values(), color='g')
```

[159]: <BarContainer object of 9 artists>




**As we see on the basis of the importance of features, the accomodation_id corresponding to accomodation_type is the most important feature for determining if the hotel accepts the bookings throughout the year. That feature has 50% of the role to play in each of the decisions. Going on, we see the number of hotels owned by an owner, number of reviews and minimum_nights contribute to decision making. As we suspected during data cleaning the regions are not very far from each other, we see that the geospatial data has less of a role to play in decision making with latitude, longitude, and regions being less contributive to the final decision making.**

**Task:**

- **Submit the predictions on the test dataset using your optimized model** For each record in the test set (`test.csv`), predict the value of the `yearly_availability` variable. Submit a CSV file with a header row and one row per test entry.

The file (`submissions.csv`) should have exactly 2 columns: - **id** - **yearly_availability**

[179]:
```
submission_df=pd.DataFrame()

submission_df['id']=filtered_test_data['id']
submission_df['yearly_availability']=test_predictions
```

[180]:
```
submission_df.head()
```

[180]:

| | id | yearly_availability |
|---|---|---|
| 0 | 19215 | 0 |
| 1 | 36301 | 0 |
| 2 | 40566 | 1 |
| 3 | 33694 | 0 |
| 4 | 28873 | 0 |

[181]:

```
#Submission
submission_df.to_csv('submissions.csv',index=False)
```