Questions

February 8, 2022

1 Data Science Challenge

[466]:
```
# 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 xgboost
!{sys.executable} -m pip install plot-metric
```

```
Requirement already satisfied: xgboost in /opt/conda/lib/python3.7/site-packages
(1.5.2)
Requirement already satisfied: numpy in /opt/conda/lib/python3.7/site-packages
(from xgboost) (1.19.2)
Requirement already satisfied: scipy in /opt/conda/lib/python3.7/site-packages
(from xgboost) (1.3.3)
Collecting plot-metric
  Downloading plot_metric-0.0.6-py3-none-any.whl (13 kB)
Requirement already satisfied: matplotlib>=3.0.2 in
/opt/conda/lib/python3.7/site-packages (from plot-metric) (3.1.3)
Requirement already satisfied: pandas>=0.23.4 in /opt/conda/lib/python3.7/site-
packages (from plot-metric) (0.25.3)
Requirement already satisfied: scipy>=1.1.0 in /opt/conda/lib/python3.7/site-
packages (from plot-metric) (1.3.3)
Requirement already satisfied: numpy>=1.15.4 in /opt/conda/lib/python3.7/site-
packages (from plot-metric) (1.19.2)
Collecting colorlover>=0.3.0
  Downloading colorlover-0.3.0-py3-none-any.whl (8.9 kB)
Requirement already satisfied: scikit-learn>=0.21.2 in
/opt/conda/lib/python3.7/site-packages (from plot-metric) (1.0.2)
Requirement already satisfied: seaborn>=0.9.0 in /opt/conda/lib/python3.7/site-
packages (from plot-metric) (0.9.0)
Requirement already satisfied: cycler>=0.10 in /opt/conda/lib/python3.7/site-
packages (from matplotlib>=3.0.2->plot-metric) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=3.0.2->plot-metric)
```

```
(2.4.7)
Requirement already satisfied: python-dateutil>=2.1 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=3.0.2->plot-metric)
(2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/opt/conda/lib/python3.7/site-packages (from matplotlib>=3.0.2->plot-metric)
(1.3.0)
Requirement already satisfied: pytz>=2017.2 in /opt/conda/lib/python3.7/site-
packages (from pandas>=0.23.4->plot-metric) (2020.1)
Requirement already satisfied: joblib>=0.11 in /opt/conda/lib/python3.7/site-
packages (from scikit-learn>=0.21.2->plot-metric) (0.17.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/conda/lib/python3.7/site-packages (from scikit-learn>=0.21.2->plot-metric)
(3.1.0)
Requirement already satisfied: six in /opt/conda/lib/python3.7/site-packages
(from cycler>=0.10->matplotlib>=3.0.2->plot-metric) (1.15.0)
Installing collected packages: colorlover, plot-metric
Successfully installed colorlover-0.3.0 plot-metric-0.0.6
```

[469]:
```python
#Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier, plot_importance
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from plot_metric.functions import BinaryClassification
from sklearn.metrics import roc_auc_score
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.

Column	Description
accommodation_type	The type of accommodation offered by the hotel. For example: Private room, Entire house/apt, etc.
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

[646]:
```
# Dataset is already loaded below
data = pd.read_csv("train.csv")
print(len(data))
```

```
2870
```

There is high correlation between owner_id and id, and number_of_reviews and reviews_per_month. One of these columns will be removed as explained below.

[647]:
```
cor_plot = data.corr()
cor_plot.style.background_gradient(cmap = 'coolwarm')
```

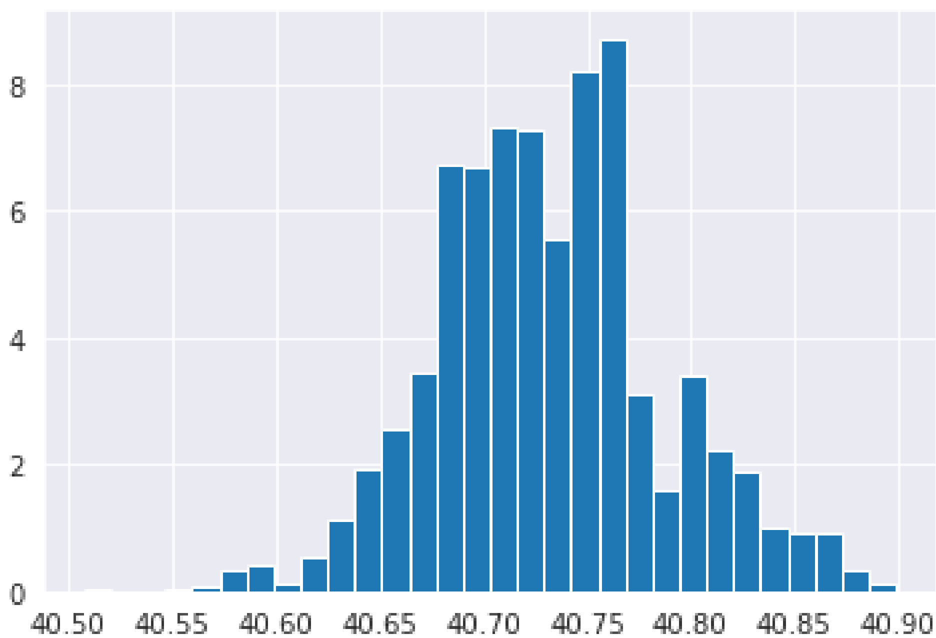
[647]:
```
<pandas.io.formats.style.Styler at 0x7f3d4021a790>
```

2 Histogram plot of numerical columns

[648]:
```
plt.hist(data['latitude'] , density=True, bins = 30)
```

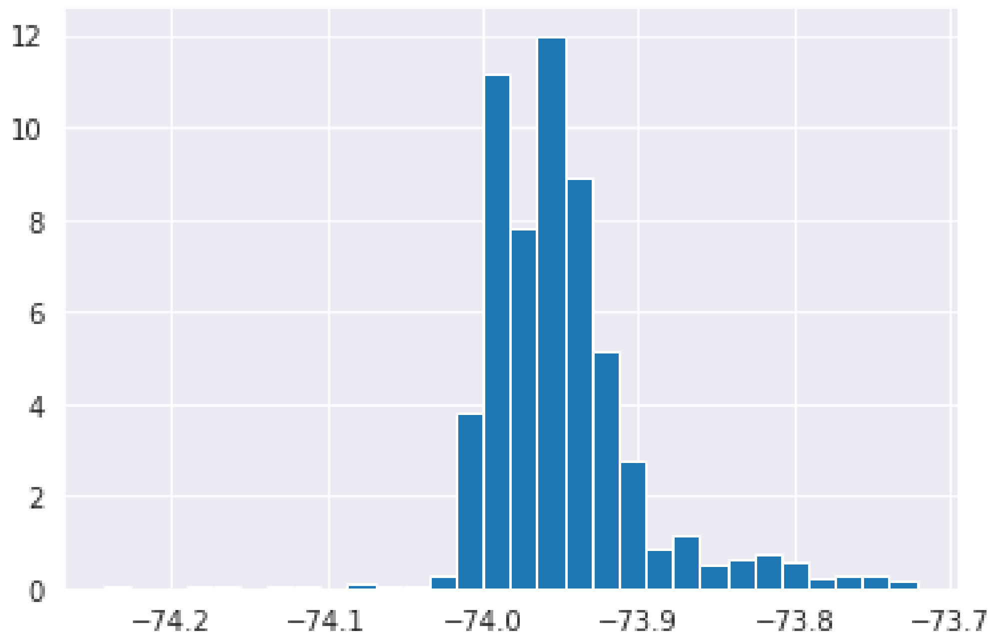
[648]:
```
(array([0.02668955, 0.        , 0.        , 0.02668955, 0.08006865,
        0.34696413, 0.40034323, 0.13344774, 0.53379097, 1.12096104,
        1.94833704, 2.56219666, 3.44295176, 6.72576622, 6.69907668,
        7.33962584, 7.28624674, 5.55142609, 8.19369139, 8.72748236,
        3.12267718, 1.57468336, 3.38957266, 2.24192207, 1.8682684 ,
```

```
        1.01420284, 0.90744465, 0.90744465, 0.32027458, 0.10675819]),
 array([40.50708 , 40.520135, 40.53319 , 40.546245, 40.5593  , 40.572355,
        40.58541 , 40.598465, 40.61152 , 40.624575, 40.63763 , 40.650685,
        40.66374 , 40.676795, 40.68985 , 40.702905, 40.71596 , 40.729015,
        40.74207 , 40.755125, 40.76818 , 40.781235, 40.79429 , 40.807345,
        40.8204  , 40.833455, 40.84651 , 40.859565, 40.87262 , 40.885675,
        40.89873 ]),
 <a list of 30 Patch objects>)
```



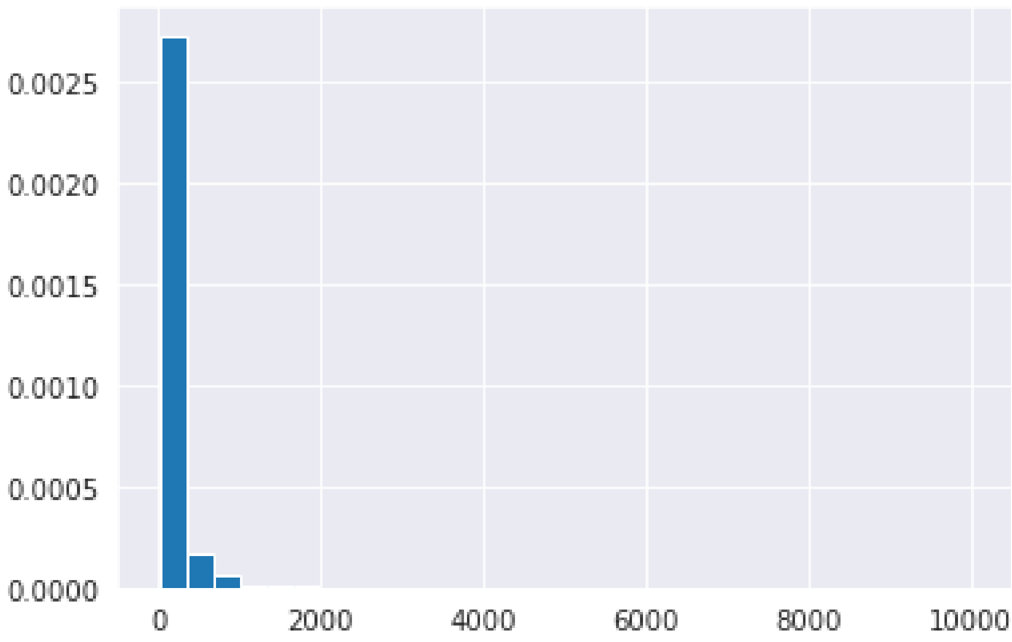
[649]:
```
plt.hist(data['longitude'] , density=True, bins = 30)
```

[649]:
```
(array([ 0.02005865,  0.        ,  0.        ,  0.02005865,  0.06017594,
         0.        ,  0.04011729,  0.04011729,  0.        ,  0.08023458,
         0.02005865,  0.02005865,  0.2607624 ,  3.83120141, 11.19272454,
         7.7827547 , 11.99507039,  8.92609753,  5.11495476,  2.74803452,
         0.88258043,  1.14334283,  0.5215248 ,  0.60175938,  0.76222855,
         0.58170074,  0.22064511,  0.2607624 ,  0.28082105,  0.16046917]),
 array([-74.24285   , -74.22547933, -74.20810867, -74.190738  ,
        -74.17336733, -74.15599667, -74.138626  , -74.12125533,
        -74.10388467, -74.086514  , -74.06914333, -74.05177267,
        -74.034402  , -74.01703133, -73.99966067, -73.98229   ,
        -73.96491933, -73.94754867, -73.930178  , -73.91280733,
        -73.89543667, -73.878066  , -73.86069533, -73.84332467,
        -73.825954  , -73.80858333, -73.79121267, -73.773842  ,
        -73.75647133, -73.73910067, -73.72173   ]),
 <a list of 30 Patch objects>)
```



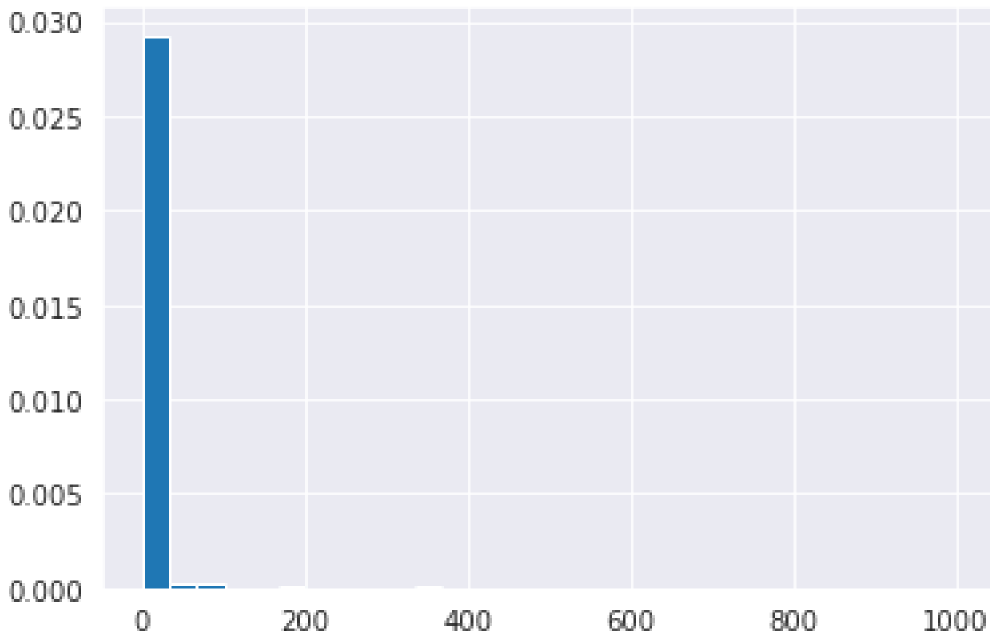
[650]:
```
plt.hist(data['cost'] , density=True, bins = 30)
```

[650]:
```
(array([2.72913445e-03, 1.63245772e-04, 6.80190719e-05, 1.04644726e-05,
        7.32513081e-06, 9.41802533e-06, 1.04644726e-06, 2.09289452e-06,
        3.13934178e-06, 1.04644726e-06, 0.00000000e+00, 0.00000000e+00,
        1.04644726e-06, 2.09289452e-06, 1.04644726e-06, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 1.04644726e-06,
        1.04644726e-06, 0.00000000e+00, 0.00000000e+00, 1.04644726e-06,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 1.04644726e-06]),
 array([  10.        ,  342.96666667,  675.93333333, 1008.9       ,
        1341.86666667, 1674.83333333, 2007.8       , 2340.76666667,
        2673.73333333, 3006.7       , 3339.66666667, 3672.63333333,
        4005.6       , 4338.56666667, 4671.53333333, 5004.5       ,
        5337.46666667, 5670.43333333, 6003.4       , 6336.36666667,
        6669.33333333, 7002.3       , 7335.26666667, 7668.23333333,
        8001.2       , 8334.16666667, 8667.13333333, 9000.1       ,
        9333.06666667, 9666.03333333, 9999.        ]),
 <a list of 30 Patch objects>)
```



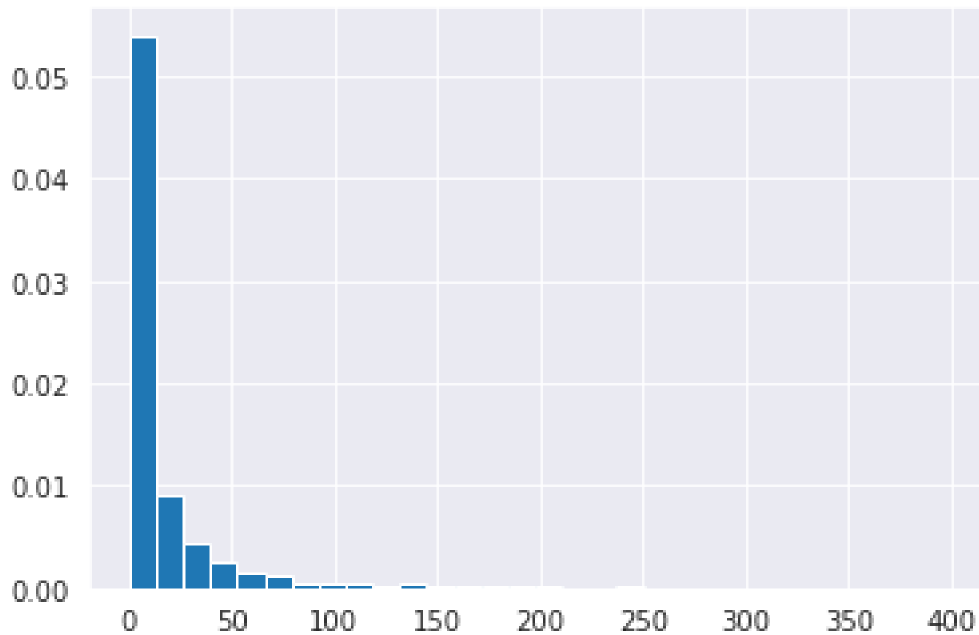
[651]:
```
plt.hist(data['minimum_nights'] , density=True, bins = 30)
```

[651]:
```
(array([2.92850509e-02, 1.99004280e-04, 1.88530371e-04, 2.09478190e-05,
        1.04739095e-05, 1.57108642e-04, 2.09478190e-05, 0.00000000e+00,
        2.09478190e-05, 0.00000000e+00, 1.15213004e-04, 1.04739095e-05,
        0.00000000e+00, 0.00000000e+00, 1.04739095e-05, 1.04739095e-05,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
        0.00000000e+00, 1.04739095e-05]),
 array([  1.        ,  34.26666667,  67.53333333, 100.8       ,
        134.06666667, 167.33333333, 200.6       , 233.86666667,
        267.13333333, 300.4       , 333.66666667, 366.93333333,
        400.2       , 433.46666667, 466.73333333, 500.        ,
        533.26666667, 566.53333333, 599.8       , 633.06666667,
        666.33333333, 699.6       , 732.86666667, 766.13333333,
        799.4       , 832.66666667, 865.93333333, 899.2       ,
        932.46666667, 965.73333333, 999.        ]),
 <a list of 30 Patch objects>)
```



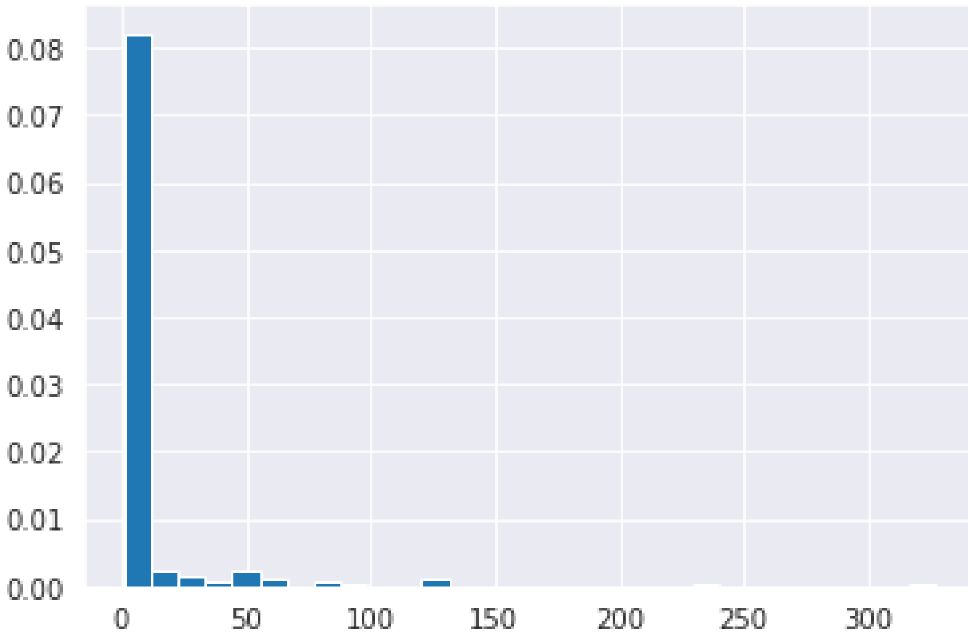
[652]:
```
plt.hist(data['number_of_reviews'] , density=True, bins = 30)
```

[652]:
```
(array([5.39849160e-02, 9.05041238e-03, 4.26057425e-03, 2.61985622e-03,
        1.56132845e-03, 1.16438054e-03, 5.82190270e-04, 5.29263882e-04,
        3.96947912e-04, 2.64631941e-04, 3.44021523e-04, 2.64631941e-04,
        1.58779165e-04, 2.11705553e-04, 7.93895823e-05, 1.85242359e-04,
        5.29263882e-05, 5.29263882e-05, 7.93895823e-05, 2.64631941e-05,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 2.64631941e-05,
        0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 2.64631941e-05,
        0.00000000e+00, 2.64631941e-05]),
 array([  0.        ,  13.16666667,  26.33333333,  39.5       ,
         52.66666667,  65.83333333,  79.        ,  92.16666667,
        105.33333333, 118.5       , 131.66666667, 144.83333333,
        158.        , 171.16666667, 184.33333333, 197.5       ,
        210.66666667, 223.83333333, 237.        , 250.16666667,
        263.33333333, 276.5       , 289.66666667, 302.83333333,
        316.        , 329.16666667, 342.33333333, 355.5       ,
        368.66666667, 381.83333333, 395.        ]),
 <a list of 30 Patch objects>)
```




```
[653]: plt.hist(data['owned_hotels'] , density=True, bins = 30)
```

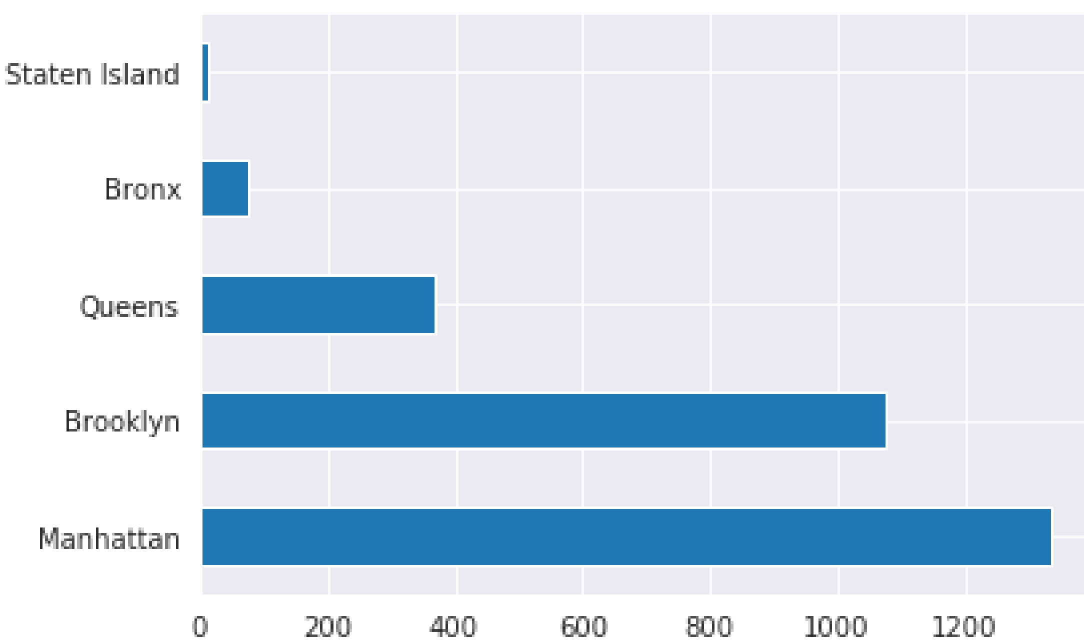
```
[653]: (array([8.21487356e-02, 2.30862957e-03, 1.44289348e-03, 6.41285992e-04,
                2.11624377e-03, 9.29864689e-04, 0.00000000e+00, 6.41285992e-04,
                2.56514397e-04, 3.20642996e-05, 0.00000000e+00, 9.93993288e-04,
                0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
                0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
                0.00000000e+00, 2.88578696e-04, 0.00000000e+00, 0.00000000e+00,
                0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00,
                0.00000000e+00, 2.24450097e-04]),
         array([  1.        ,  11.86666667,  22.73333333,  33.6       ,
                 44.46666667,  55.33333333,  66.2       ,  77.06666667,
                 87.93333333,  98.8       , 109.66666667, 120.53333333,
                131.4       , 142.26666667, 153.13333333, 164.        ,
                174.86666667, 185.73333333, 196.6       , 207.46666667,
                218.33333333, 229.2       , 240.06666667, 250.93333333,
                261.8       , 272.66666667, 283.53333333, 294.4       ,
                305.26666667, 316.13333333, 327.        ]),
         <a list of 30 Patch objects>)
```



Region wise count

[654]:
```
data['region'].value_counts().plot(kind = 'barh')
```

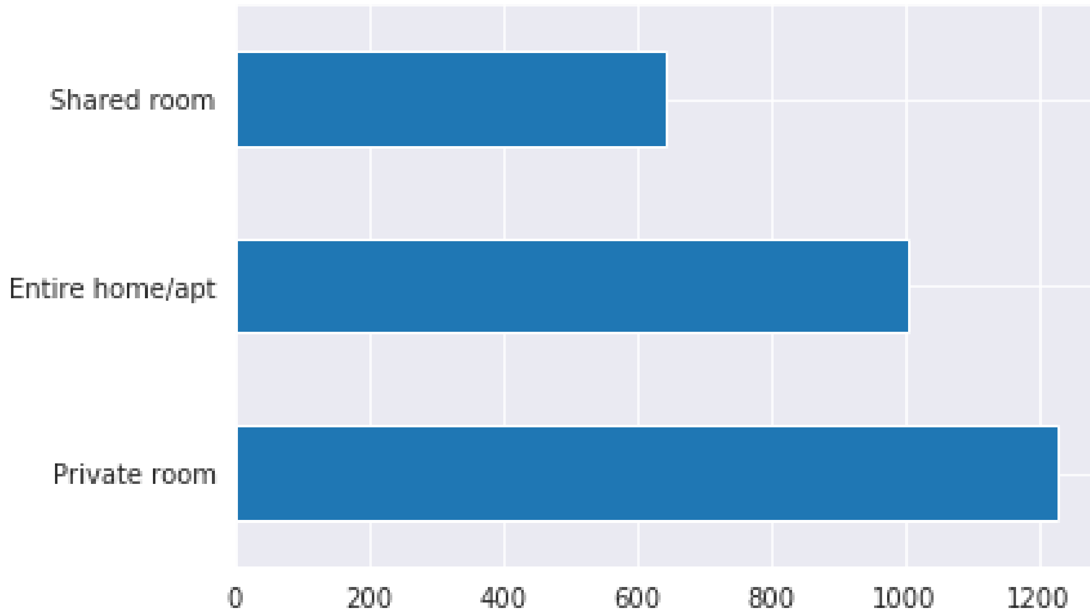
[654]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3d3a17cc50>



Accommodation Type wise count

[655]:
```
data['accommodation_type'].value_counts().plot(kind = 'barh')
```

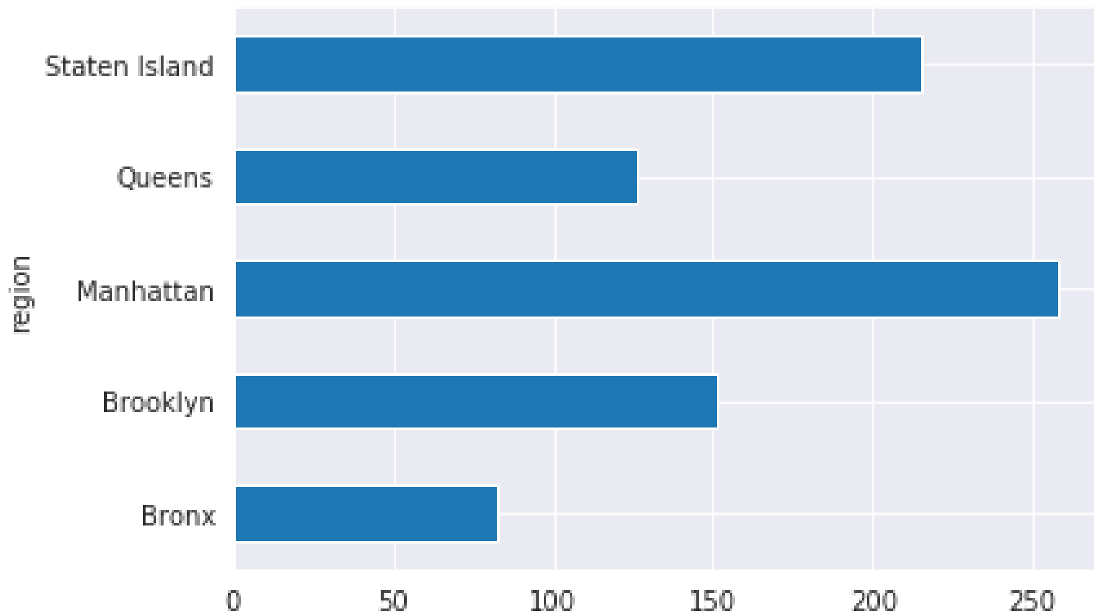
[655]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3d3a0a6f10>



Cost across regions

[656]:
```
data.groupby(['region'])['cost'].mean().plot(kind = 'barh')
```

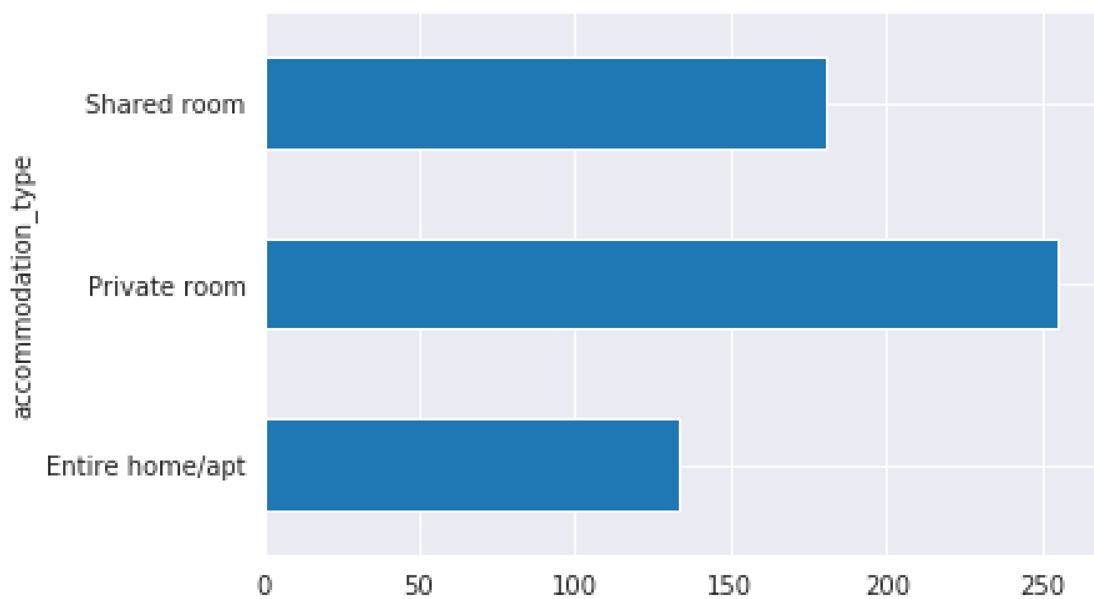
[656]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3d3a027710>



Cost across Accomodation Type

[657]:
```
data.groupby(['accommodation_type'])['cost'].mean().plot(kind = 'barh')
```

[657]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3d39f941d0>



Since percentage of unique values in 'id', and 'owner_id' column in more than 80%, I am dropping the columns, since they wont add much value to the model

[658]:
```
print(data['id'].nunique()/len(data))
print(data['owner_id'].nunique()/len(data))
```

```
1.0
0.8261324041811847
```

[659]:
```
del data['id']
del data['owner_id']
```

[660]:
```
set(data['accommodation_type'])
```

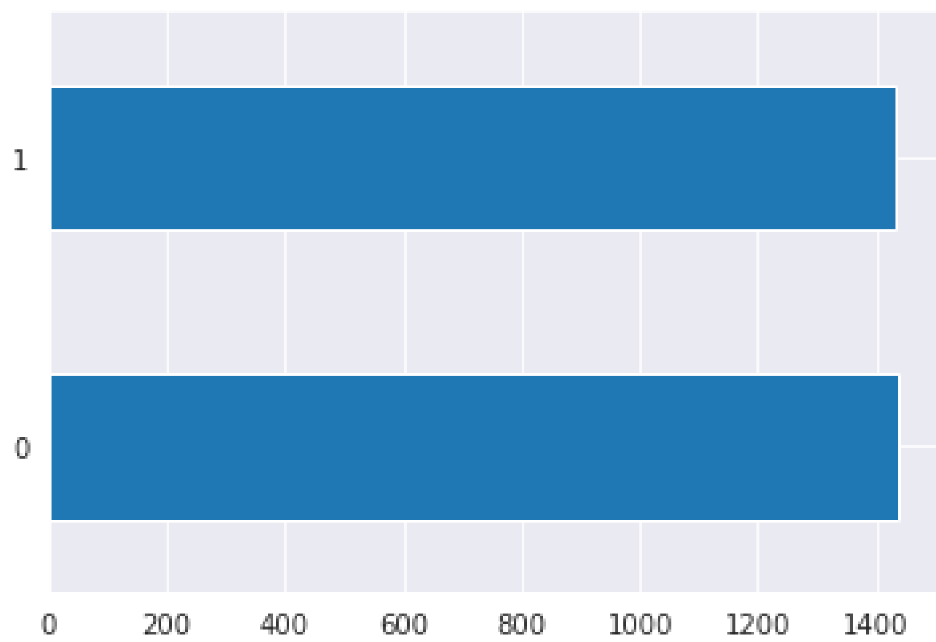
[660]:
```
{'Entire home/apt', 'Private room', 'Shared room'}
```

Plot to check number of classes. The data is well sampled in this case

[661]:
```
print(data['yearly_availability'].value_counts())
data['yearly_availability'].value_counts().plot(kind = 'barh')
```

```
0    1439
1    1431
Name: yearly_availability, dtype: int64
```

[661]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3d39f5ea90>
```



[662]:
```
data['reviews_per_month'].value_counts()
```

[662]:
```
1.00    76
0.16    40
0.11    34
0.05    32
0.12    30
        ..
2.70     1
3.04     1
6.59     1
5.92     1
5.97     1
Name: reviews_per_month, Length: 419, dtype: int64
```

Checking if data contains null values. Dropping the column reviews_per_month, since it has more than 20% missing values and it is correlated with number_of_reviews as shown earlier.

[663]:
```python
data.isnull().sum()/len(data)
```

[663]:
```
region                  0.00000
latitude                0.00000
longitude               0.00000
accommodation_type      0.00000
cost                    0.00000
minimum_nights          0.00000
number_of_reviews       0.00000
reviews_per_month       0.23554
owned_hotels            0.00000
yearly_availability     0.00000
dtype: float64
```

[541]:
```python
del data['reviews_per_month']
```

Getting the numerical columns

[542]:
```python
num_cols = list(data.columns[data.dtypes.apply(lambda c: np.issubdtype(c, np.
 ↪number))])
num_cols.remove('yearly_availability')
num_cols
```

[542]:
```
['latitude',
 'longitude',
 'cost',
 'minimum_nights',
 'number_of_reviews',
 'owned_hotels']
```

Label Encoder vs One hot Encoder : Label Encoder might generate a bias in the data, which is unwarranted. And since one hot encoder did not blew up the data size, going forward with One hot Encoder

[543]:
```
# le = LabelEncoder()
# data['region'] = le.fit_transform(data['region'])
# data['accommodation_type'] = le.fit_transform(data['accommodation_type'])
```

[544]:
```
data.columns
```

[544]:
```
Index(['region', 'latitude', 'longitude', 'accommodation_type', 'cost',
       'minimum_nights', 'number_of_reviews', 'owned_hotels',
       'yearly_availability'],
      dtype='object')
```

[545]:
```
data = pd.get_dummies(data, columns =␣
↪['region','accommodation_type'],drop_first = True)
```

[546]:
```
data.columns
```

[546]:
```
Index(['latitude', 'longitude', 'cost', 'minimum_nights', 'number_of_reviews',
       'owned_hotels', 'yearly_availability', 'region_Brooklyn',
       'region_Manhattan', 'region_Queens', 'region_Staten Island',
       'accommodation_type_Private room', 'accommodation_type_Shared room'],
      dtype='object')
```

[547]:
```
data.shape
```

[547]:
```
(2870, 13)
```

[548]:
```
data.head()
```

[548]:
```
   latitude  longitude  cost  minimum_nights  number_of_reviews  owned_hotels  \
0  40.71854  -74.00439   170               5                  7             1
1  40.64446  -73.95030    65               3                238             1
2  40.78573  -73.81062    85               1                  0             1
3  40.73863  -73.98002   210              30                  0            65
4  40.82426  -73.94630    75               3                 38             3

   yearly_availability  region_Brooklyn  region_Manhattan  region_Queens  \
0                    0                0                 1              0
1                    0                1                 0              0
2                    1                0                 0              1
3                    1                0                 1              0
4                    1                0                 1              0

   region_Staten Island  accommodation_type_Private room  \
0                     0                                0
1                     0                                0
2                     0                                1
```

```
3                      0                                  1
4                      0                                  0

   accommodation_type_Shared room
0                               0
1                               0
2                               0
3                               0
4                               1
```

[549]:
```
#Explore columns
data.columns
```

[549]:
```
Index(['latitude', 'longitude', 'cost', 'minimum_nights', 'number_of_reviews',
       'owned_hotels', 'yearly_availability', 'region_Brooklyn',
       'region_Manhattan', 'region_Queens', 'region_Staten Island',
       'accommodation_type_Private room', 'accommodation_type_Shared room'],
      dtype='object')
```

[550]:
```
#Description
data.describe()
```

[550]:
```
          latitude    longitude         cost  minimum_nights  \
count  2870.000000  2870.000000  2870.000000     2870.000000
mean     40.731224   -73.950158   195.943206       11.530314
std       0.054942     0.049745   406.184714       37.972339
min      40.507080   -74.242850    10.000000        1.000000
25%      40.692462   -73.984003    75.000000        1.000000
50%      40.728250   -73.956720   120.000000        3.000000
75%      40.762658   -73.934202   200.000000        6.000000
max      40.898730   -73.721730  9999.000000      999.000000

       number_of_reviews  owned_hotels  yearly_availability  region_Brooklyn  \
count        2870.000000   2870.000000          2870.000000      2870.000000
mean           16.315331      8.411498             0.498606         0.374564
std            32.481722     27.105522             0.500085         0.484095
min             0.000000      1.000000             0.000000         0.000000
25%             1.000000      1.000000             0.000000         0.000000
50%             4.000000      1.000000             0.000000         0.000000
75%            16.000000      3.000000             1.000000         1.000000
max           395.000000    327.000000             1.000000         1.000000

       region_Manhattan  region_Queens  region_Staten Island  \
count       2870.000000     2870.00000           2870.000000
mean           0.464460        0.12892              0.004878
std            0.498822        0.33517              0.069685
min            0.000000        0.00000              0.000000
```

```
25%            0.000000         0.00000              0.000000
50%            0.000000         0.00000              0.000000
75%            1.000000         0.00000              0.000000
max            1.000000         1.00000              1.000000

       accommodation_type_Private room  accommodation_type_Shared room
count                      2870.000000                     2870.000000
mean                          0.426829                        0.224042
std                           0.494703                        0.417022
min                           0.000000                        0.000000
25%                           0.000000                        0.000000
50%                           0.000000                        0.000000
75%                           1.000000                        0.000000
max                           1.000000                        1.000000
```

Droppping duplicate entries, if any

[551]:
```
data = data.drop_duplicates()
data.shape
```

[551]:
```
(2870, 13)
```

[552]:
```
data.columns
```

[552]:
```
Index(['latitude', 'longitude', 'cost', 'minimum_nights', 'number_of_reviews',
       'owned_hotels', 'yearly_availability', 'region_Brooklyn',
       'region_Manhattan', 'region_Queens', 'region_Staten Island',
       'accommodation_type_Private room', 'accommodation_type_Shared room'],
      dtype='object')
```

[554]:
```
yearly_availability = data['yearly_availability']
del data['yearly_availability']
data['yearly_availability'] = yearly_availability
```

[555]:
```
data
```

[555]:
```
      latitude  longitude  cost  minimum_nights  number_of_reviews  \
0     40.71854  -74.00439   170               5                  7
1     40.64446  -73.95030    65               3                238
2     40.78573  -73.81062    85               1                  0
3     40.73863  -73.98002   210              30                  0
4     40.82426  -73.94630    75               3                 38
…          …         …    …              …                 …
2865  40.74316  -73.98038   400               2                  0
2866  40.73523  -73.99465   180               3                  2
2867  40.76619  -73.98987   179               3                 17
2868  40.74637  -73.97207   200              30                  0
```

```
2869  40.79208  -73.96482  1000             30               24

      owned_hotels  region_Brooklyn  region_Manhattan  region_Queens  \
0                1                0                 1              0
1                1                1                 0              0
2                1                0                 0              1
3               65                0                 1              0
4                3                0                 1              0
…              …                …                 …              …
2865             1                0                 1              0
2866             1                0                 1              0
2867             1                0                 1              0
2868            49                0                 1              0
2869            11                0                 1              0

      region_Staten Island  accommodation_type_Private room  \
0                        0                                0
1                        0                                0
2                        0                                1
3                        0                                1
4                        0                                0
…                      …                                …
2865                     0                                1
2866                     0                                1
2867                     0                                0
2868                     0                                1
2869                     0                                0

      accommodation_type_Shared room  yearly_availability
0                                  0                    0
1                                  0                    0
2                                  0                    1
3                                  0                    1
4                                  1                    1
…                                …                    …
2865                               0                    1
2866                               0                    1
2867                               0                    0
2868                               0                    1
2869                               1                    1

[2870 rows x 13 columns]
```

[556]:
```
X = data.iloc[:,:-1]
y = data.iloc[:,-1]
```

[557]:
```
num_cols
```

[557]:

```
['latitude',
 'longitude',
 'cost',
 'minimum_nights',
 'number_of_reviews',
 'owned_hotels']
```

Scaling numerical columns

[558]:

```
sc = StandardScaler()
X[num_cols] = sc.fit_transform(X[num_cols])
```

[559]:

```
X.columns
```

[559]:

```
Index(['latitude', 'longitude', 'cost', 'minimum_nights', 'number_of_reviews',
       'owned_hotels', 'region_Brooklyn', 'region_Manhattan', 'region_Queens',
       'region_Staten Island', 'accommodation_type_Private room',
       'accommodation_type_Shared room'],
      dtype='object')
```

[560]:

```
X
```

[560]:

```
      latitude  longitude      cost  minimum_nights  number_of_reviews  \
0    -0.230897  -1.090408 -0.063882       -0.172006          -0.286837
1    -1.579468  -0.002863 -0.322430       -0.224685           6.826094
2     0.992246   2.805573 -0.273182       -0.277364          -0.502380
3     0.134826  -0.600420  0.034613        0.486483          -0.502380
4     1.693655   0.077562 -0.297806       -0.224685           0.667712
…          …          …        …              …                  …
2865  0.217291  -0.607658  0.502462       -0.251024          -0.502380
2866  0.072932  -0.894574 -0.039258       -0.224685          -0.440796
2867  0.636535  -0.798466 -0.041720       -0.224685           0.021082
2868  0.275727  -0.440575  0.009989        0.486483          -0.502380
2869  1.107843  -0.294805  1.979880        0.486483           0.236626

      owned_hotels  region_Brooklyn  region_Manhattan  region_Queens  \
0        -0.273479                0                 1              0
1        -0.273479                1                 0              0
2        -0.273479                0                 0              1
3         2.088075                0                 1              0
4        -0.199680                0                 1              0
…             …                …                 …             …
2865     -0.273479                0                 1              0
2866     -0.273479                0                 1              0
2867     -0.273479                0                 1              0
2868      1.497687                0                 1              0
2869      0.095514                0                 1              0
```

```
      region_Staten Island  accommodation_type_Private room  \
0                        0                                0
1                        0                                0
2                        0                                1
3                        0                                1
4                        0                                0
…                      …                              …
2865                     0                                1
2866                     0                                1
2867                     0                                0
2868                     0                                1
2869                     0                                0

      accommodation_type_Shared room
0                                  0
1                                  0
2                                  0
3                                  0
4                                  1
…                                …
2865                               0
2866                               0
2867                               0
2868                               0
2869                               1

[2870 rows x 12 columns]
```

[561]:
```
y
```

[561]:
```
0       0
1       0
2       1
3       1
4       1
       ..
2865    1
2866    1
2867    0
2868    1
2869    1
Name: yearly_availability, Length: 2870, dtype: int64
```

Splitting the train data into train and validation data set.

[562]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=123)
```

Tried Naive Bayes, Decision Tree, Random Forest and XGB classification models.

[563]:
```
gnb = GaussianNB()
gnb.fit(X_train,y_train)
```

[563]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

[699]:
```
y_pred = gnb.predict(X_train)
accuracy = np.sum((y_train==y_pred))/len(y_pred)
print('Accuracy_Train: Naive Bayes', accuracy*100)
```

```
Accuracy_Train: Naive Bayes 80.96689895470384
```

[697]:
```
y_pred = gnb.predict(X_test)
accuracy = np.sum((y_test==y_pred))/len(y_pred)
print('Accuracy_Test: Naive Bayes', accuracy*100)
```

```
Accuracy_Test: Naive Bayes 79.44250871080139
```

[698]:
```
print("The classification report is as follows...\n")
print(classification_report(y_pred,y_test))
```

```
The classification report is as follows…

              precision    recall  f1-score   support

           0       0.91      0.74      0.81       348
           1       0.68      0.88      0.77       226

    accuracy                           0.79       574
   macro avg       0.80      0.81      0.79       574
weighted avg       0.82      0.79      0.80       574
```

[628]:
```
cm = (confusion_matrix(y_test,y_pred))
df_cm = pd.DataFrame(cm,index=['Class 0','Class 1'], columns=['Class 0','Class␣
↪1'])
print("Confusion matrix\n")
df_cm
```

```
Confusion matrix
```

[628]:

	Class 0	Class 1
Class 0	261	21

```
Class 1        19       273
```

Since train and test accuracy is the same, the model is not overfitting

[568]:
```python
dt = DecisionTreeClassifier(criterion='gini',random_state=123)
dt.fit(X_train,y_train)
```

[568]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, 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, presort=False,
                       random_state=123, splitter='best')
```

[695]:
```python
y_pred = dt.predict(X_train)
accuracy = np.sum((y_train==y_pred))/len(y_pred)
print('Accuracy_Train: Decision Tree', accuracy*100)
```

```
Accuracy_Train: Decision Tree 100.0
```

[696]:
```python
y_pred = dt.predict(X_test)
accuracy = np.sum((y_test==y_pred))/len(y_pred)
print('Accuracy_Test: Decision Tree', accuracy*100)
```

```
Accuracy_Test: Decision Tree 88.32752613240417
```

[ ]:
```python

```

[571]:
```python
rf = RandomForestClassifier(n_estimators=500,random_state =123, max_depth =15)
rf.fit(X_train, y_train)
```

[571]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=15, 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=500,
                       n_jobs=None, oob_score=False, random_state=123,
                       verbose=0, warm_start=False)
```

[694]:
```python
y_pred = rf.predict(X_train)
accuracy = np.sum((y_train==y_pred))/len(y_pred)
print('Accuracy_Train - Random Forest', accuracy*100)
```

```
Accuracy_Train - Random Forest 100.0
```

[693]:
```python
y_pred = rf.predict(X_test)
accuracy = np.sum((y_test==y_pred))/len(y_pred)
print('Accuracy_Test - Random Forest',accuracy*100)
```

```
Accuracy_Test - Random Forest 93.90243902439023
```

[ ]:

[574]:
```
xgb = XGBClassifier(seed = 123)
xgb.fit(X_train, y_train)
```

```
[19:05:42] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the
default evaluation metric used with the objective 'binary:logistic' was changed
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore
the old behavior.

/opt/conda/lib/python3.7/site-packages/xgboost/sklearn.py:1224: UserWarning: The
use of label encoder in XGBClassifier is deprecated and will be removed in a
future release. To remove this warning, do the following: 1) Pass option
use_label_encoder=False when constructing XGBClassifier object; and 2) Encode
your labels (y) as integers starting with 0, i.e. 0, 1, 2, …, [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

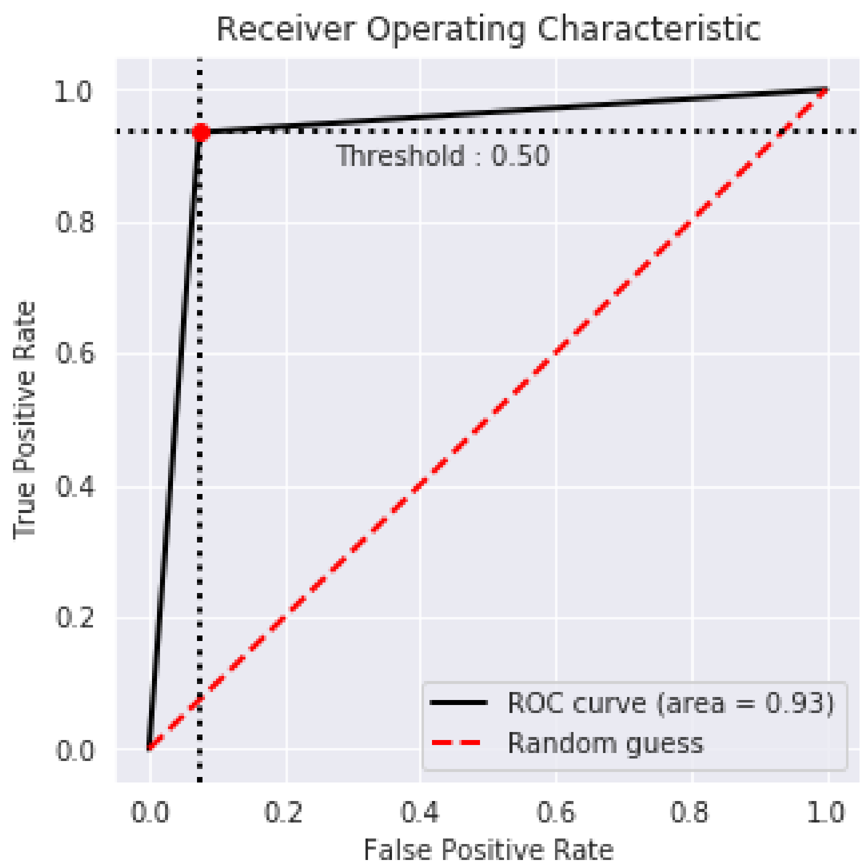
[574]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=8,
              num_parallel_tree=1, objective='binary:logistic',
              predictor='auto', random_state=123, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, seed=123, subsample=1, tree_method='exact',
              use_label_encoder=True, validate_parameters=1, …)
```

[700]:
```
y_pred = xgb.predict(X_test)
accuracy = np.sum((y_test==y_pred))/len(y_pred)
print('Accuracy_Test - XGB:',accuracy*100)
```

```
Accuracy_Test - XGB: 93.03135888501743
```

[627]:
```
bc = BinaryClassification(y_test, y_pred, labels=["Class 0", "Class 1"])

# Figures
plt.figure(figsize=(5,5))
bc.plot_roc_curve()
plt.show()
```



[577]:
```
roc_auc_score(y_test, y_pred)
```

[577]: 0.9302317108714659

The Random Forest model perfoms best with alost 94% accuracy. So will use the same model for predicting values for the test data

2.1 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.**

[664]:
```
#Loading Test data
test=pd.read_csv('test.csv')
test.head()
```

[664]:
```
      id     region  latitude  longitude accommodation_type  cost  \
0  19215   Brooklyn  40.70912  -73.94513        Shared room   135
1  36301   Brooklyn  40.57646  -73.96641     Entire home/apt    69
2  40566  Manhattan  40.76616  -73.98228       Private room   225
3  33694  Manhattan  40.77668  -73.94587        Shared room   125
4  28873  Manhattan  40.80279  -73.94450     Entire home/apt    43

   minimum_nights  number_of_reviews  reviews_per_month   owner_id  \
0               2                 22               0.66    4360212
1               2                  8               0.90  181356989
2              30                  0                NaN   13773574
3              30                  9               0.82    6788748
4               1                 13               0.72  105061915

   owned_hotels
0             1
1             2
2            12
3             1
4             2
```

Doing the same preprocessing steps done for the train data

[665]:
```python
id = test['id']
```

[666]:
```python
del test['id']
del test['owner_id']
del test['reviews_per_month']
```

[667]:
```python
test = pd.get_dummies(test, columns =␣
↪['region','accommodation_type'],drop_first = True)
```

[668]:
```python
test.shape
```

[668]:
```
(718, 12)
```

[669]:
```python
test = test.drop_duplicates()
test.shape
```

[669]:
```
(718, 12)
```

[670]:
```python
test[num_cols] = sc.transform(test[num_cols])
```

[671]:
```python
test
```

[671]:
```
     latitude  longitude      cost  minimum_nights  number_of_reviews  \
0   -0.402381   0.101086 -0.150064       -0.251024           0.175042
```

```
1   -2.817356  -0.326774 -0.312580       -0.251024          -0.256045
2    0.635989  -0.645860  0.071548        0.486483          -0.502380
3    0.827497   0.086207 -0.174688        0.486483          -0.225253
4    1.302810   0.113753 -0.376602       -0.277364          -0.102085
..        …          …        …              …                  …
713  2.087595   0.695225 -0.292881       -0.251024           0.606129
714 -0.381628   0.054641 -0.125441       -0.224685           0.821672
715 -0.315365  -0.825207 -0.260871       -0.251024           1.345134
716  0.232401  -0.439972  0.009989        0.486483          -0.502380
717 -0.432054  -1.286243 -0.066344        0.486483          -0.440796

     owned_hotels  region_Brooklyn  region_Manhattan  region_Queens  \
0       -0.273479                1                 0              0
1       -0.236580                1                 0              0
2        0.132413                0                 1              0
3       -0.273479                0                 1              0
4       -0.236580                0                 1              0
..            …               …                …              …
713     -0.273479                0                 0              0
714     -0.273479                1                 0              0
715     -0.273479                0                 1              0
716      0.538305                0                 1              0
717      4.154435                0                 1              0

     region_Staten Island  accommodation_type_Private room  \
0                       0                                0
1                       0                                0
2                       0                                1
3                       0                                0
4                       0                                0
..                    …                              …
713                     0                                0
714                     0                                0
715                     0                                0
716                     0                                1
717                     0                                1

     accommodation_type_Shared room
0                                 1
1                                 0
2                                 0
3                                 1
4                                 0
..                              …
713                               0
714                               0
715                               0
```

```
716                                 0
717                                 0

[718 rows x 12 columns]
```

[672]:
```python
test_output = rf.predict(test)
```

Highlight the most important features of the model for management.

Task:

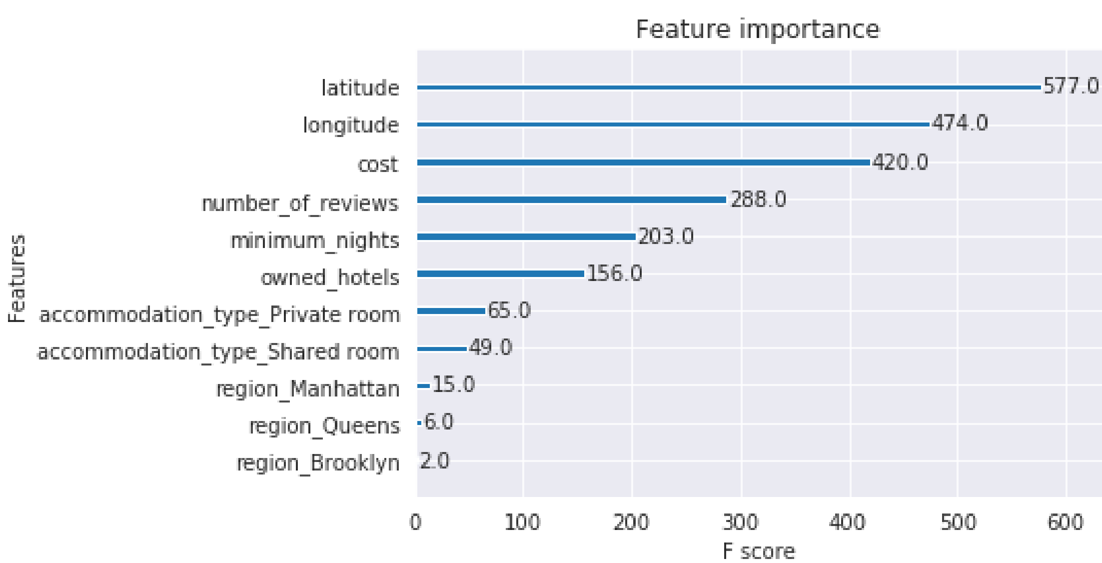
- **Visualize the top 20 features and their feature importance.**

[676]:
```python
xgb.feature_importances_
```

[676]:
```
array([0.0114952 , 0.01148161, 0.01226087, 0.01423117, 0.01824551,
       0.10496806, 0.01417442, 0.00985813, 0.01053431, 0.        ,
       0.6363641 , 0.15638658], dtype=float32)
```

[677]:
```python
plot_importance(xgb)
```

[677]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3d39f58490>
```



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**

[678]:
```
submission_df = pd.DataFrame(columns=['id','yearly_availability'])
```

[679]:
```
test_output.shape
```

[679]:
```
(718,)
```

[682]:
```
submission_df['id'] = id
submission_df['yearly_availability'] = test_output
```

[683]:
```
submission_df
```

[683]:
```
        id  yearly_availability
0    19215                    0
1    36301                    0
2    40566                    1
3    33694                    0
4    28873                    0
..     ...                  ...
713  26801                    0
714  20110                    0
715  31383                    0
716  47135                    1
717  13154                    1

[718 rows x 2 columns]
```

[684]:
```
#Submission
submission_df.to_csv('submissions.csv',index=False)
```
