NYC Airbnb Data Analysis

Presented by

Zixi Tian
Bangzhuo Song
Yutong Gao
Chenyi Dong

Date: May 4th, 2023

Agenda

Introduction

Problem: 1. What factors affect the price of Airbnb?

2. Is the Airbnb worthy of recommendation?

Process data

Visualization

Regression model —Prediction

Classification—Results determine the Airbnb's recommendation status

Conclusion & Improvement

Process Dataset

- **“AB_NYC_2019.csv”**
  Summary information and metrics for listings in New York City

- **“NYPD_Complaint_Data_Historic.csv”**
  All valid felony, misdemeanor, and violation crimes reported to the New York City Police Department (NYPD) from 2006 to the end of year 2019.

- **“Nyc-transit-subway-entrance-and-exit-data.csv”**
  This data file provides a variety of information on subway station entrances and exits which includes but is not limited to: Division, Line, Station Name, Longitude and Latitude coordinates of entrances/exits.

Airbnb Dataset

id listing ID	name name of the listing	neighbourhood_g... location	neighbourhood area	latitude latitude coordinates	longitude longitude coordinates	room_type listing space type	price price in dollars	reviews_per_mon... number of reviews per month
2539 36.5m	47906 unique values	Manhattan 44% Brooklyn 41% Other (7130) 15%	Williamsburg 8% Bedford-Stuyvesant 8% Other (41261) 84%	40.5 40.9	-74.2 -73.7	Entire home/apt 52% Private room 46% Other (1160) 2%	0 10000	0.01 58.5
2539	Clean & quiet apt home by the park	Brooklyn	Kensington	40.64749	-73.97237	Private room	149	0.21
2595	Skylit Midtown Castle	Manhattan	Midtown	40.75362	-73.98377	Entire home/apt	225	0.38
3647	THE VILLAGE OF HARLEM....NEW YORK !	Manhattan	Harlem	40.80902	-73.9419	Private room	150	
3831	Cozy Entire Floor of Brownstone	Brooklyn	Clinton Hill	40.68514	-73.95976	Entire home/apt	89	4.64
5022	Entire Apt: Spacious Studio/Loft by central park	Manhattan	East Harlem	40.79851	-73.94399	Entire home/apt	80	0.10
5099	Large Cozy 1 BR Apartment In Midtown East	Manhattan	Murray Hill	40.74767	-73.975	Entire home/apt	200	0.59
5121	BlissArtsSpace!	Brooklyn	Bedford-Stuyvesant	40.68688	-73.95596	Private room	60	0.40
5178	Large Furnished Room Near B'way	Manhattan	Hell's Kitchen	40.76489	-73.98493	Private room	79	3.47
5203	Cozy Clean Guest Room - Family Apt	Manhattan	Upper West Side	40.80178	-73.96723	Private room	79	0.99
5238	Cute & Cozy Lower East Side 1 bdrm	Manhattan	Chinatown	40.71344	-73.99037	Entire home/apt	150	1.33
5295	Beautiful 1br on Upper West Side	Manhattan	Upper West Side	40.80316	-73.96545	Entire home/apt	135	0.43

```
# check null
df_ny.dropna(inplace=True)
df_ny.isnull().sum()
```

```
id                                  0
name                                0
host_id                             0
host_name                           0
neighbourhood_group                 0
neighbourhood                       0
latitude                            0
longitude                           0
room_type                           0
price                               0
minimum_nights                      0
number_of_reviews                   0
last_review                         0
reviews_per_month                   0
calculated_host_listings_count      0
availability_365                    0
dtype: int64
```

	id	host_id	latitude	longitude	price	minimum_nights	number_of_reviews	reviews_per_month	calculated_host_listings_count	availability_365
count	4.889500e+04	4.889500e+04	48895.000000	48895.000000	48895.000000	48895.000000	48895.000000	38843.000000	48895.000000	48895.000000
mean	1.901714e+07	6.762001e+07	40.728949	-73.952170	152.720687	7.029962	23.274466	1.373221	7.143982	112.781327
std	1.098311e+07	7.861097e+07	0.054530	0.046157	240.154170	20.510550	44.550582	1.680442	32.952519	131.622289
min	2.539000e+03	2.438000e+03	40.499790	-74.244420	0.000000	1.000000	0.000000	0.010000	1.000000	0.000000
25%	9.471945e+06	7.822033e+06	40.690100	-73.983070	69.000000	1.000000	1.000000	0.190000	1.000000	0.000000
50%	1.967728e+07	3.079382e+07	40.723070	-73.955680	106.000000	3.000000	5.000000	0.720000	1.000000	45.000000
75%	2.915218e+07	1.074344e+08	40.763115	-73.936275	175.000000	5.000000	24.000000	2.020000	2.000000	227.000000
max	3.648724e+07	2.743213e+08	40.913060	-73.712990	10000.000000	1250.000000	629.000000	58.500000	327.000000	365.000000

Combine Crime dataset

NYPD_Complaint_Data_Historic.csv (2.22 GB)

Detail Compact Column

# CMPLNT_NUM	CMPLNT_FR_DT	CMPLNT_FR_TM	PD_DESC	BORO_NM	Latitude	Longitude
100m 1000m	Invalid date 30Dec19	15Feb21 16Feb21	ASSAULT 3 9% HARASSMENT,SUB... 8% Other (5808786) 83%	BROOKLYN 30% MANHATTAN 24% Other (3230327) 46%	40.1 59.7	-77.5 -73.7
700381962	05/28/2015	15:00:00	HARASSMENT,SUBD 3,4,5	BRONX	40.84586773	-73.915888033
642234217	10/28/2013	13:50:00	CRIMINAL MISCHIEF,UNCLASSIFIED 4	STATEN ISLAND	40.627060894	-74.077149232
242465164	05/09/2012	20:50:00	WEAPONS, POSSESSION, ETC	MANHATTAN	40.800965968	-73.969047272
927207428	01/03/2014	13:30:00	LARCENY,GRAND BY EXTORTION	QUEENS	40.745241809	-73.894253382
492142357	04/13/2016	00:00:00	CRIMINAL MISCHIEF 4TH, GRAFFIT	BRONX	40.810351863	-73.924942326

```
df_cr['Latitude_block'] = df_cr['Latitude'].round(1)
df_cr['Longitude_block'] = df_cr['Longitude'].round(1)
complaints_per_block = df_cr.groupby(['Latitude_block','Longitude_block']).count()["CMPLNT_NUM"]
complaints_per_block = pd.DataFrame(complaints_per_block)

df_ny['Latitude_block'] = df_ny['latitude'].round(1)
df_ny['Longitude_block'] = df_ny['longitude'].round(1)
df_ny = pd.merge(df_ny,complaints_per_block,on = ['Latitude_block','Longitude_block'])
df_ny.head()
```

Latitude_block	Longitude_block	CMPLNT_NUM
40.5	**-74.3**	46
	-74.2	1870
	-74.1	155
40.6	**-74.2**	3881
	-74.1	12568
	-74.0	30593
	-73.9	13858
	-73.8	6298
	-73.7	740
40.7	**-74.0**	69211
	-73.9	75977
	-73.8	31933
	-73.7	5669
40.8	**-74.0**	42006
	-73.9	97817
	-73.8	13933
	-73.7	539
40.9	**-73.9**	37495
	-73.8	6387

Combine Subway Station Dataset

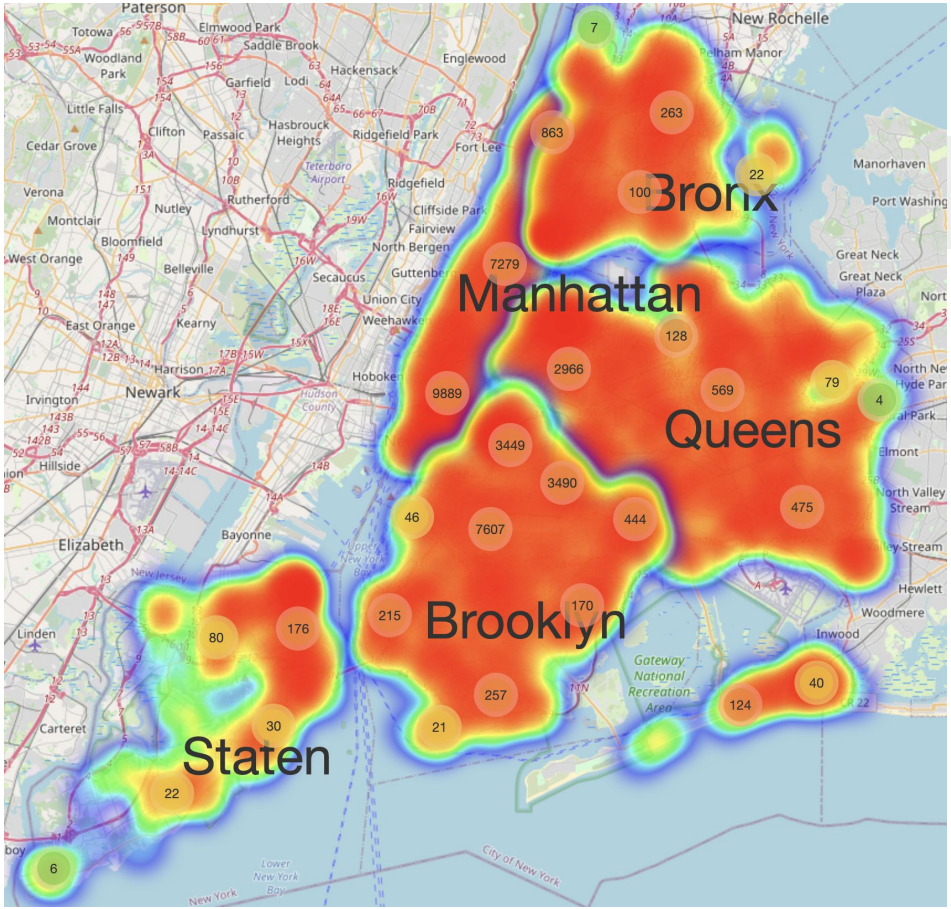
	Division	Line	Station Name	Station Latitude	Station Longitude	Route1	Route2	Route3	Route4	Route5	...	ADA	ADA Notes	Free Crossover	North South Street	East West Street	Corner	Entrance Latitude	Entrance Longitude	Station Location	Entrance Location
0	BMT	4 Avenue	25th St	40.660397	-73.998091	R	NaN	NaN	NaN	NaN	...	False	NaN	False	4th Ave	25th St	SW	40.660489	-73.998220	{'longitude': '-73.998091', 'latitude': '40.66...	{'longitude': '-73.99822', 'latitude': '40.660...
1	BMT	4 Avenue	25th St	40.660397	-73.998091	R	NaN	NaN	NaN	NaN	...	False	NaN	False	4th Ave	25th St	SE	40.660323	-73.997952	{'longitude': '-73.998091', 'latitude': '40.66...	{'longitude': '-73.997952', 'latitude': '40.66...
2	BMT	4 Avenue	36th St	40.655144	-74.003549	N	R	NaN	NaN	NaN	...	False	NaN	True	4th Ave	36th St	NW	40.654676	-74.004306	{'longitude': '-74.003549', 'latitude': '40.65...	{'longitude': '-74.004306', 'latitude': '40.65...
3	BMT	4 Avenue	36th St	40.655144	-74.003549	N	R	NaN	NaN	NaN	...	False	NaN	True	4th Ave	36th St	NE	40.654365	-74.004113	{'longitude': '-74.003549', 'latitude': '40.65...	{'longitude': '-74.004113', 'latitude': '40.65...

```python
def find_nearest(x):
    # approximate radius of earth in km
    R = 6373.0
    # get lat and loc of airbnb
    lats0 = x['latitude']
    long0 = x['longitude']
    distance = list()
    # convert location to radians
    loc0 = np.radians(np.array([long0,lats0]).reshape(-1,2))
    # diff in long and lats
    dist = np.radians(station_locs) - loc0
    # diff in lats & long
    lats_diff = dist[:,1]
    long_diff = dist[:,0]
    # calculate distance
    a = (np.sin(lats_diff/2))**2 + np.cos(lats0) * np.cos(station_locs[:,1]) * (np.sin(long_diff /2))**2
    c = 2 * np.arctan2(np.sqrt(a), np.sqrt(1-a))
    distance = R * c
    # return minimum distancs in km
    return min(distance)
```

```python
df_ny["distance to nearest subway (km)"] = df_ny.apply(lambda x : find_nearest(x),axis=1 )
df_ny["distance to nearest subway (km)"]
```

id	neighbourhood	CMPLNT_NUM	distance subway (km)
504322	Greenwich Village	69211	0.368715
11743513	Clinton Hill	69211	0.596407
19700871	Bedford-Stuyvesant	69211	0.316213
30443961	Lower East Side	69211	0.166326
34903861	Westchester Square	13933	0.081579
6376606	Kips Bay	69211	0.333895
32267833	Parkchester	97817	0.218212
10365516	Bushwick	75977	0.482817
33995462	Belmont	37495	1.412291
8468662	Upper West Side	42006	0.095694
20207132	Bath Beach	30593	1.123573
26573069	Bushwick	75977	0.217065
306702	Fort Greene	69211	0.152295
10016832	Murray Hill	69211	0.620270
33429437	Hell's Kitchen	42006	0.554039
20659824	East New York	75977	0.364960
13770179	Upper West Side	42006	0.379036
20331995	Richmond Hill	31933	0.223691
33532216	Bedford-Stuyvesant	75977	0.139271
29829054	Williamsburg	75977	0.101468
228979	Hell's Kitchen	42006	0.449750
20364240	Crown Heights	75977	0.560470
29869390	Corona	75977	0.755267
7654931	Bedford-Stuyvesant	75977	0.580132
2739978	Greenwich Village	69211	0.184181

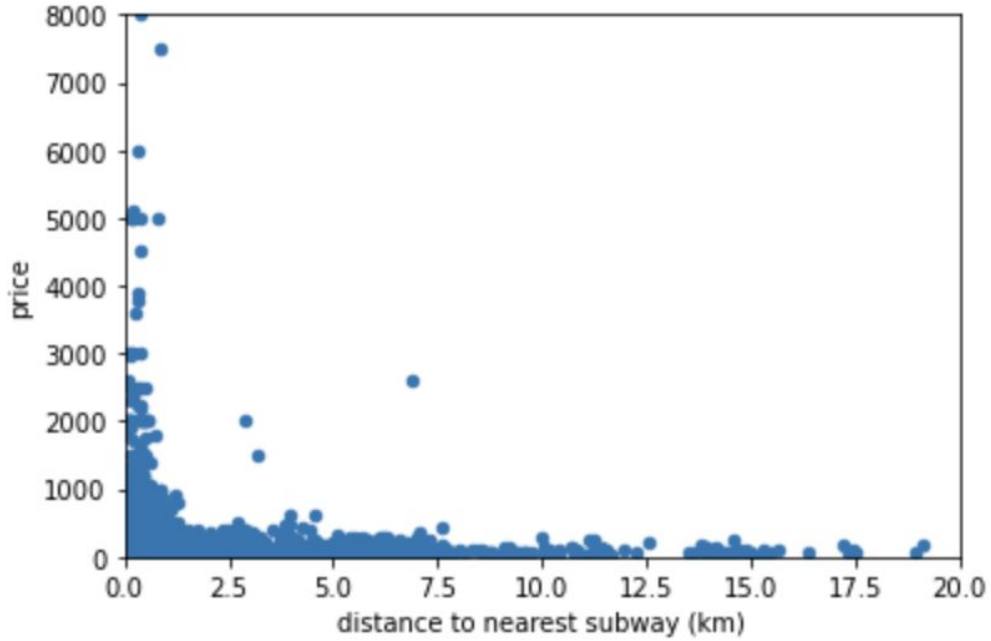
Folium Map of Airbnb Price Data



Price vs Distance to Nearest Subway (km)

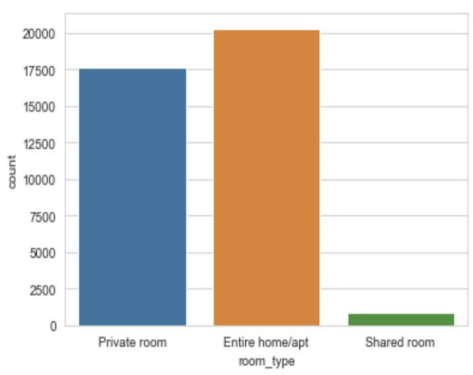
```
# price vs distance to nearest subway (km)
mn = 'distance to nearest subway (km)'
data = pd.concat([df_ny['price'], df_ny[mn]], axis = 1)
data.plot.scatter(x = mn, y = 'price', xlim = (0,20),ylim = (0,8000),grid = False)
```

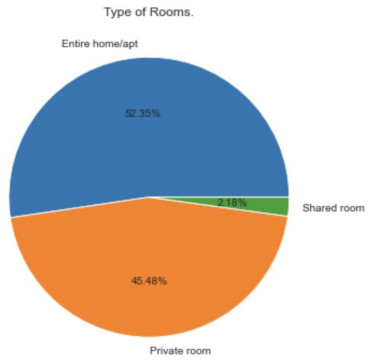
```
<AxesSubplot:xlabel='distance to nearest subway (km)', ylabel='price'>
```

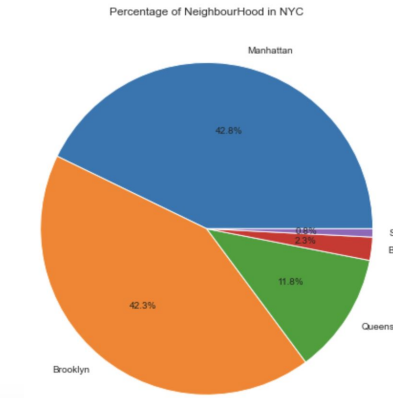


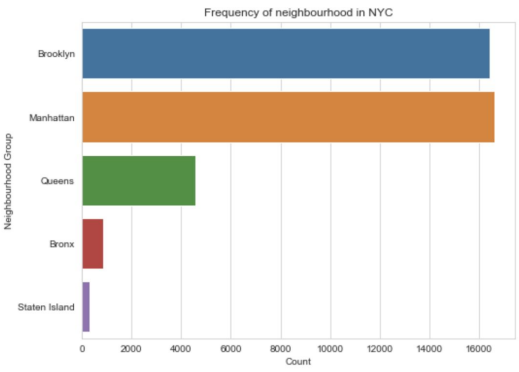
Airbnb prices will increase with closer proximity to a metro station.

Room Type vs Price vs Neighbourhood Group









- The percentage of shared rooms is low. In addition, the price range is from 0 to 2,000, which is the lowest of these three room categories.

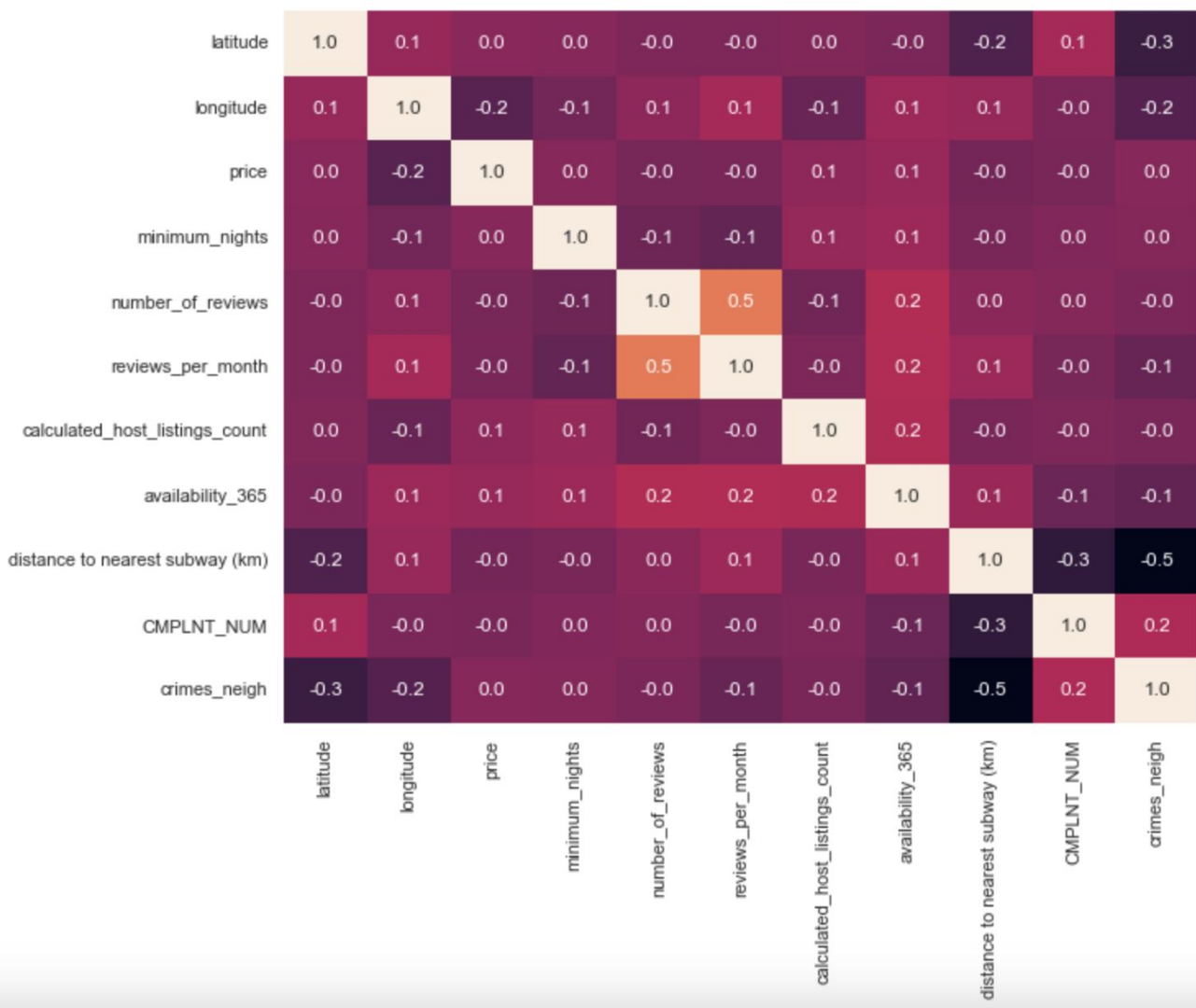
- More than half of the Airbnb is “Entire home/apt”.
- Airbnbs in both Manhattan and Brooklyn are more than 40%.

Visualization

Correlation Coefficient

Except for "crimes_neigh," these variables are independent of one another.

To avoid collinearity, "crimes_neigh" is removed when we do regression and classification.

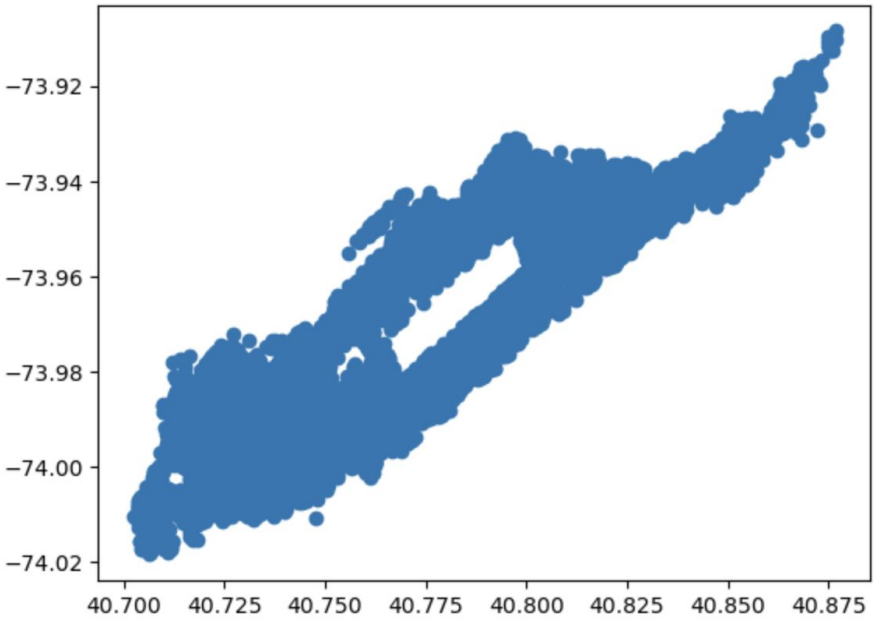


Delete the Outlier

When it is two dimensionals, we use the scatterplot to see the distribution of our data (for example: Latitude and Longitude)

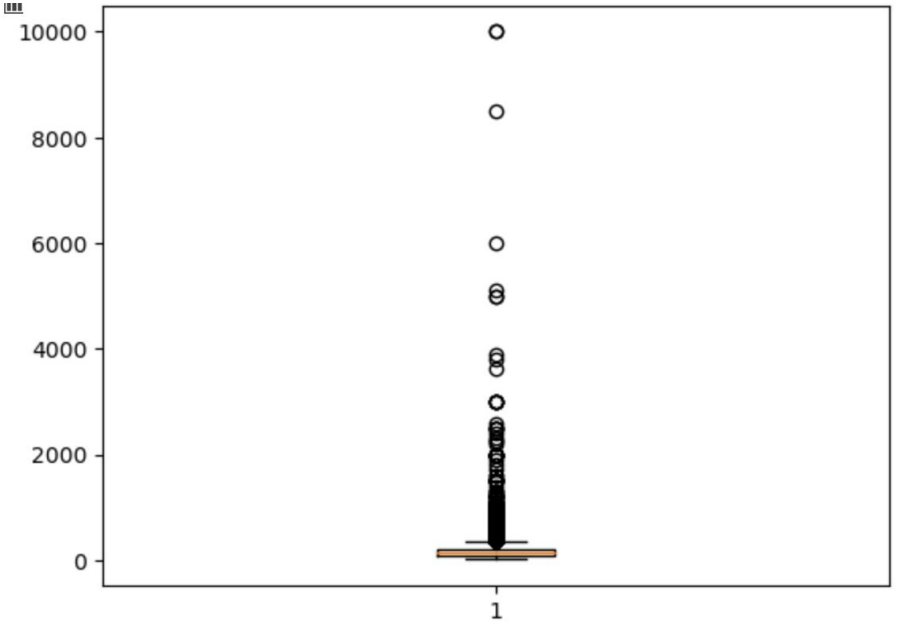
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.scatter(x = df1.latitude, y = df1.longitude)
```

```
<matplotlib.collections.PathCollection at 0x172299e70>
```



Using boxplot when it is one dimensional (for example :price)

```
fig, ax = plt.subplots()
ax.boxplot(x = df1.price)
```

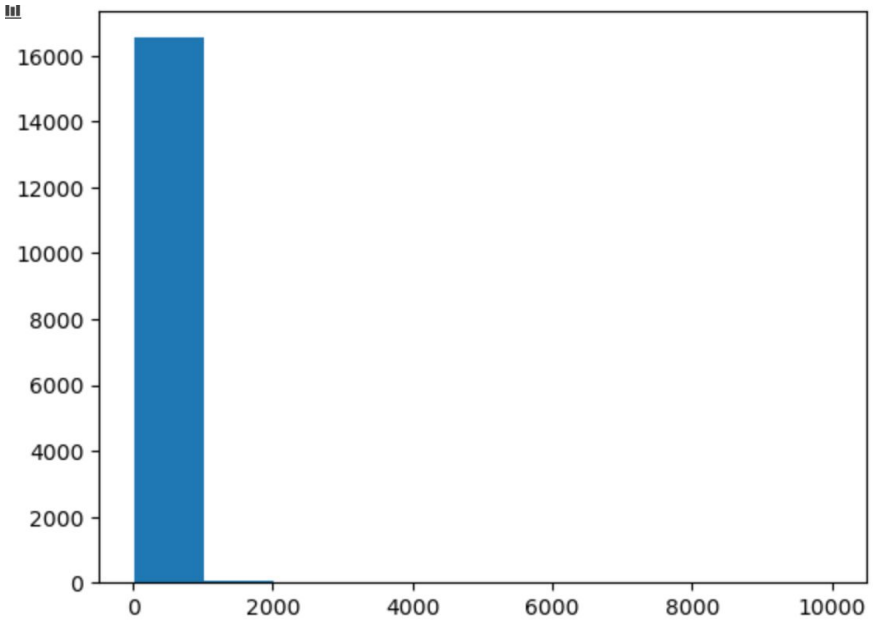


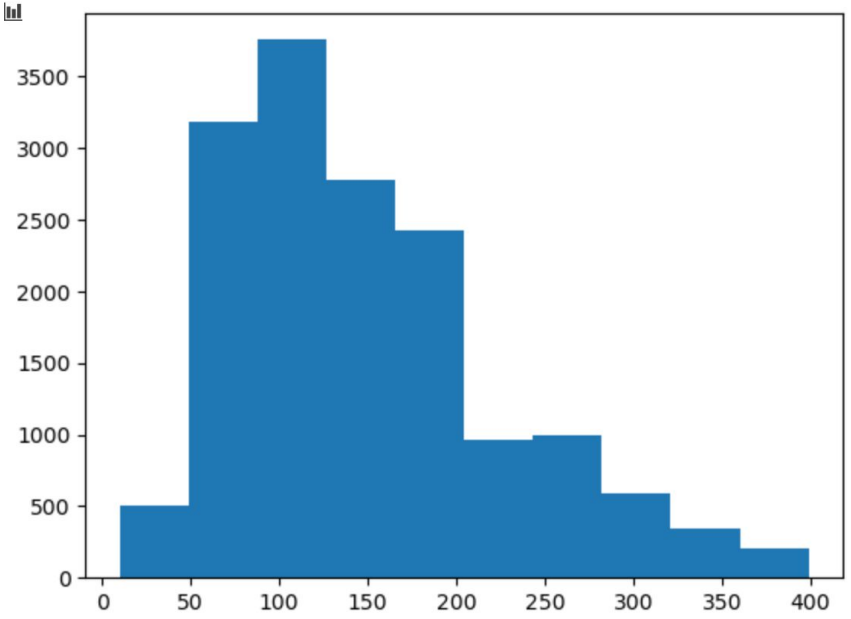
Delete the Outlier

```
filtered_df1 = df1[df1.price < df1.price.quantile(0.95)]
```

```
fig, ax = plt.subplots()
ax.hist(x = filtered_df1.price)
```

Before and after filtering the outlier

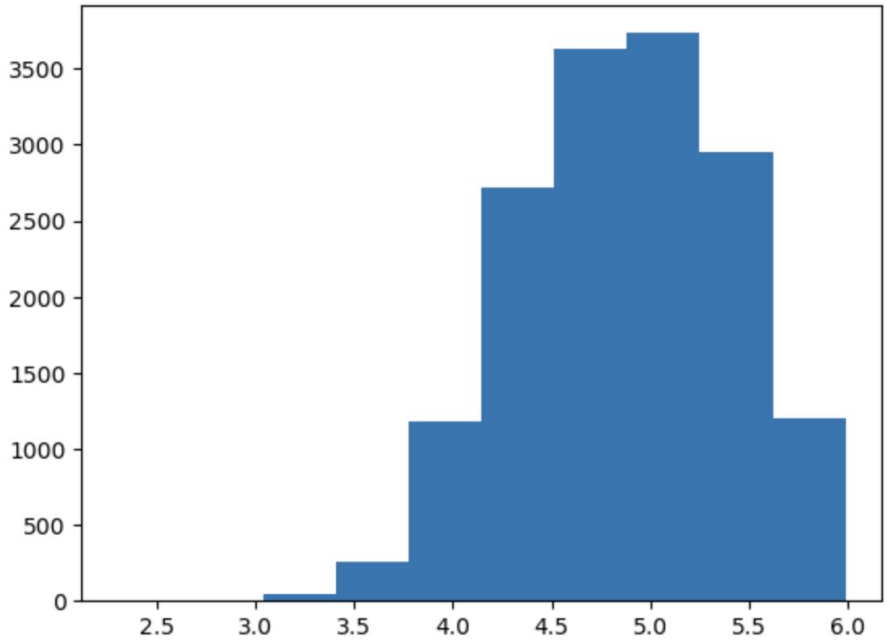




Normalize using Log

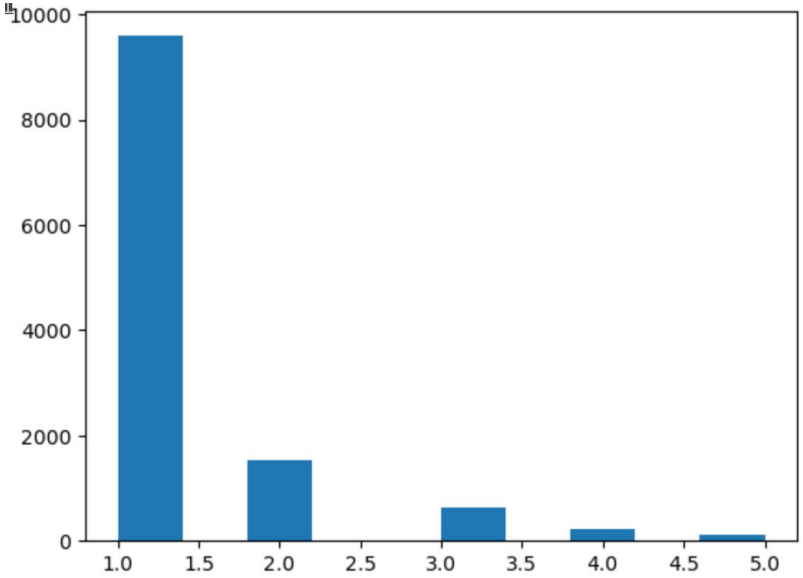
```
fig, ax = plt.subplots()
ax.hist(x = np.log(filtered_df1.price))
```

```
filtered_df1["log_price"] = np.log(filtered_df1["price"])
```

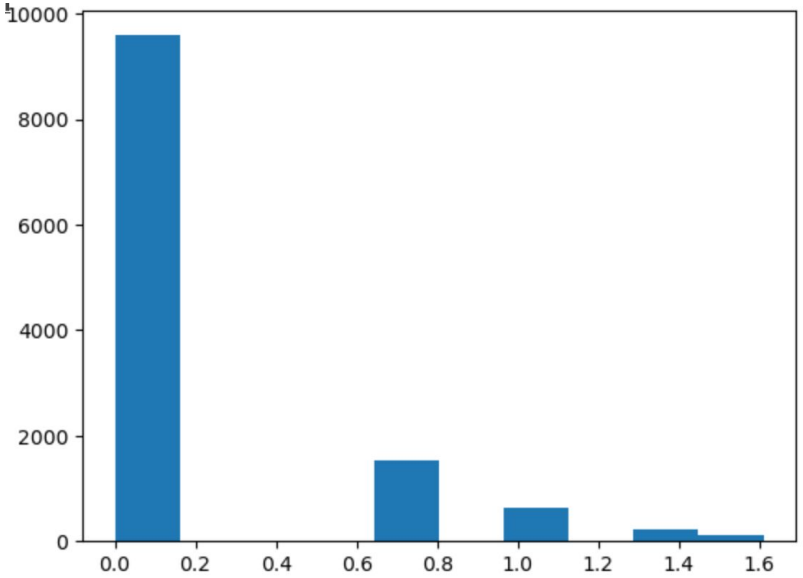


Log is not always good

```
fig, ax = plt.subplots()
ax.hist(x = filtered_df1.calculated_host_listings_count)
```




```
fig, ax = plt.subplots()
ax.hist(x = np.log(filtered_df1.calculated_host_listings_count))
```



Regression

- **Aim:**
  Build a model to predict the price for an Airbnb based on selected features

- **Model:**
  Linear Regression, Lasso Regression, Ridge Regression, Decision Tree, Random Forest

- **Evaluation:**
  MSE, RMSE, R-square

Linear Regression

```
from sklearn.linear_model import LinearRegression
lr_model = LinearRegression().fit(X_train, y_train)
y_pred = lr_model.predict(X_test)
from sklearn import metrics
lr_mse = metrics.mean_squared_error(y_test, y_pred)
lr_rmse = np.sqrt(lr_mse)
lr_r2 = metrics.r2_score(y_test, y_pred)
print("Mean Squared Error:", lr_mse)
print("Root Mean Squared Error:", lr_rmse)
print("R-squared:", lr_r2)
```

```
Mean Squared Error: 0.1210962266334967
Root Mean Squared Error: 0.3479888311907391
R-squared: 0.6126547348790505
```

lasso regressin

```
from sklearn.linear_model import Lasso, Ridge
from sklearn.model_selection import GridSearchCV
lasso = Lasso()

param_grid = {'alpha': [0.1, 1, 10, 100],
              'max_iter': [1000, 5000, 10000]}

grid_search = GridSearchCV(estimator=lasso, param_grid=param_grid, cv=5)
grid_search.fit(X_encoded, y)
print(f'Best hyperparameters: {grid_search.best_params_}')
print(f'Best score: {grid_search.best_score_}')

lasso = Lasso(alpha=0.01,max_iter = 1000)
lasso.fit(X_train, y_train)
y_pred_lasso = lasso.predict(X_test)

lasso_mse = metrics.mean_squared_error(y_test, y_pred_lasso)
lasso_rmse = np.sqrt(lasso_mse)
lasso_r2 = metrics.r2_score(y_test,y_pred_lasso)
print("Mean Squared Error:", lasso_mse)
print("Root Mean Squared Error:", lasso_rmse)
print("R-squared:", lasso_r2)
```

```
Best hyperparameters: {'alpha': 0.1, 'max_iter': 1000}
Best score: 0.3500199800961982
Mean Squared Error: 0.4361776428131018
Root Mean Squared Error: 0.6604374632113943
R-squared: 0.5657930246184062
```

ridge regression

```
from sklearn.pipeline import Pipeline
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler

ridge = Ridge()
param_grid = {'alpha': [0.1, 1, 10, 100],
              'max_iter': [1000, 5000, 10000]}

grid_search = GridSearchCV(ridge, param_grid=param_grid, cv=5)
grid_search.fit(X_encoded, y)
print(f'Best hyperparameters: {grid_search.best_params_}')
print(f'Best score: {grid_search.best_score_}')
```

```
Best hyperparameters: {'alpha': 10, 'max_iter': 1000}
Best score: 0.49358820643654566
```

```
ridge = Ridge(alpha=0.01,max_iter = 1000)
ridge.fit(X_train, y_train)
y_pred_ridge = ridge.predict(X_test)
ridge_mse = metrics.mean_squared_error(y_test, y_pred_ridge)
ridge_rmse = np.sqrt(ridge_mse)
ridge_r2 = metrics.r2_score(y_test,y_pred_ridge)
print("Mean Squared Error:", ridge_mse)
print("Root Mean Squared Error:", ridge_rmse)
print("R-squared:", ridge_r2)
```

```
Mean Squared Error: 0.38905626359814394
Root Mean Squared Error: 0.6237437483439364
R-squared: 0.6127015076226647
```

random forest regressor

```
# from sklearn.ensemble import RandomForestRegressor
# rfr = RandomForestRegressor()
# param_grid = {'n_estimators': [50, 100, 200],
#               'max_depth': [5, 10],
#               'min_samples_leaf': [1, 2, 4]}

# grid_search = GridSearchCV(rfr, param_grid=param_grid, cv=5)
# grid_search.fit(X_train, y_train)

# print(f'Best hyperparameters: {grid_search.best_params_}')
# print(f'Best score: {grid_search.best_score_}')
```

```
from sklearn.ensemble import RandomForestRegressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)
rf_val_predictions = rf_model.predict(X_test)
rf_mse = metrics.mean_squared_error(y_test, rf_val_predictions)
rf_rmse = np.sqrt(rf_mse)
rf_r2 = metrics.r2_score(y_test, rf_val_predictions)
print("Mean Squared Error:", rf_mse)
print("Root Mean Squared Error:", rf_rmse)
print("R-squared:", rf_r2)
```

```
Mean Squared Error: 0.3836769129286417
Root Mean Squared Error: 0.6194165907760638
R-squared: 0.6180565541781398
```

Decision Tree

```
from sklearn.tree import DecisionTreeRegressor

depth_range = [2, 5, 10, 20, 50]
scores = []

for d in depth_range:
    dt = DecisionTreeRegressor(max_depth=d, random_state=42)

    score = cross_val_score(dt, X_train, y_train, cv=5).mean()

    # Append the mean score to the list of scores
    scores.append(score)

# Find the value of max_depth with the highest cross-validation score
best_depth = depth_range[scores.index(max(scores))]

# Print the best value of max_depth
print("Best max_depth:", best_depth)
```

```
Best max_depth: 5
```

```
from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor(max_depth=best_depth)
dtr.fit(X_train, y_train)
dtr_pred = dtr.predict(X_test)
# Evaluate the model's performance using mean squared error
#evaluation
dtr_mse = metrics.mean_squared_error(y_test, dtr_pred)
dtr_rmse = np.sqrt(dtr_mse)
dtr_r2 = metrics.r2_score(y_test,dtr_pred)
print("Mean Squared Error:", dtr_mse)
print("Root Mean Squared Error:", dtr_rmse)
print("R-squared:", dtr_r2)
```

```
Mean Squared Error: 0.12864321794956793
Root Mean Squared Error: 0.3586686743354763
R-squared: 0.5885145000141229
```

evaluation

```
evaluation = {'model':['Linear Regression','Lasso','Ridge','Decision Tree','Random Forest'],
              'MSE':[lr_mse,lasso_mse,ridge_mse,dtr_mse,rf_mse],
              'RMSE':[lr_rmse,lasso_rmse,ridge_rmse,dtr_rmse,rf_rmse],
              'R2':[lr_r2,lasso_r2,ridge_r2,dtr_r2,rf_r2]}
eva = pd.DataFrame(evaluation)
eva
```

	model	MSE	RMSE	R2
0	Linear Regression	0.121096	0.347989	0.612655
1	Lasso	0.436178	0.660437	0.565793
2	Ridge	0.389056	0.623744	0.612702
3	Decision Tree	0.413353	0.642925	0.588515
4	Random Forest	0.383677	0.619417	0.618057

Possible reason that linear regression performs well

1. Underlying relationship is linear

2. Data preprocessing: log transform; outliers; standardize – meet the assumption of linear regression

3. Low multicollinearity (feature selection); also may be the reason that lasso performs bad

4. Few omitted variables (introduce some new variables)

Linear Regression Analyst

```
import statsmodels.api as sm
reg = sm.OLS(y, sm.add_constant(X_encoded)).fit()
print(reg.summary())
```

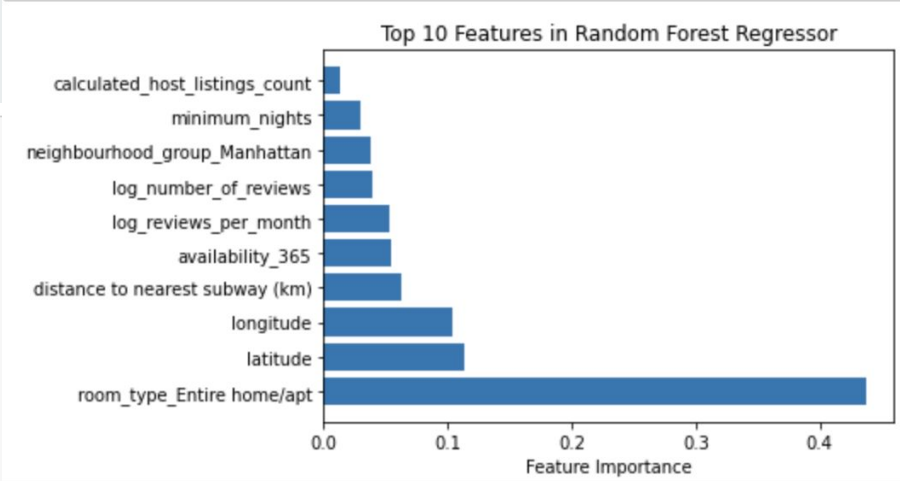
```
Covariance Type:            nonrobust
==============================================================================================
                                 coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
const                         -0.5263      0.027    -19.775      0.000      -0.578      -0.474
latitude                      -0.1100      0.038     -2.859      0.004      -0.185      -0.035
longitude                     -0.3342      0.024    -13.908      0.000      -0.381      -0.287
minimum_nights                -0.0741      0.004    -19.554      0.000      -0.081      -0.067
log_number_of_reviews         -0.0355      0.005     -6.669      0.000      -0.046      -0.025
log_reviews_per_month          0.0427      0.005      7.789      0.000       0.032       0.053
calculated_host_listings_count -0.0038     0.004     -0.924      0.356      -0.012       0.004
availability_365               0.1331      0.004     31.840      0.000       0.125       0.141
CMPLNT_NUM                    -0.0185      0.007     -2.480      0.013      -0.033      -0.004
distance to nearest subway (km) 0.0146     0.014      1.018      0.309      -0.013       0.043
room_type_Entire home/apt      0.8088      0.012     65.337      0.000       0.784       0.833
room_type_Private room        -0.3675      0.012    -30.162      0.000      -0.391      -0.344
room_type_Shared room         -0.9676      0.023    -42.505      0.000      -1.012      -0.923
neighbourhood_group_Bronx      0.3533      0.106      3.332      0.001       0.145       0.561
neighbourhood_group_Brooklyn  -0.1475      0.053     -2.783      0.005      -0.251      -0.044
neighbourhood_group_Manhattan  0.5235      0.045     11.660      0.000       0.435       0.611
neighbourhood_group_Queens     0.7711      0.075     10.216      0.000       0.623       0.919
neighbourhood_group_Staten Island -2.0266  0.176    -11.539      0.000      -2.371      -1.682
```

Subway coefficient

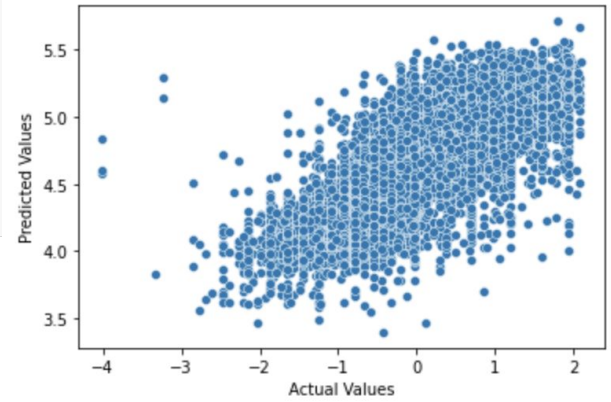
1. Measurement error: the accuracy of measurement is low (straight-line distance)
2. Nonlinear relationship
3. Collinearity: relationship between neighbour or subway

Random Forest Analyst

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.ensemble import RandomForestRegressor

# Initialize and train the random forest regressor
rfr = RandomForestRegressor(n_estimators=100, random_state=42)
rfr.fit(X_train, y_train)

# Get the feature importances and sort them in ascending order
importances = rfr.feature_importances_
sorted_idx = importances.argsort()

# Get the top 10 features
top_n = 10
top_idx = sorted_idx[-top_n:]

# Reverse the order of the arrays to plot the features in descendin
top_idx = top_idx[::-1]
importances = importances[top_idx]
feature_names = X_train.columns[top_idx]

# Plot the feature importances in a horizontal bar chart
plt.barh(np.arange(top_n), importances, align='center')
plt.yticks(np.arange(top_n), feature_names)
plt.xlabel('Feature Importance')
plt.title(f'Top {top_n} Features in Random Forest Regressor')
plt.show()
```




```
import seaborn as sns
df_pred = pd.DataFrame({'actual': y_test, 'predicted': y_pred})
sns.scatterplot(x='actual', y='predicted', data=df_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```



Stacking

```
from sklearn.model_selection import train_test_split, KFold
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
import numpy as np

X_train_part, X_holdout, y_train_part, y_holdout = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

base_models = [
    LinearRegression(),
    RandomForestRegressor(n_estimators=100, random_state=42),
    GradientBoostingRegressor(n_estimators=100, random_state=42)
]
base_models_pred = np.zeros((X_holdout.shape[0], len(base_models)))
for i, model in enumerate(base_models):
    model.fit(X_train_part, y_train_part)
    base_models_pred[:, i] = model.predict(X_holdout)

meta_model = LinearRegression()
meta_model.fit(base_models_pred, y_holdout)

test_pred = np.zeros((X_test.shape[0], len(base_models)))
for i, model in enumerate(base_models):
    test_pred[:, i] = model.predict(X_test)
stacked_pred = meta_model.predict(test_pred)

mse_stacked = mean_squared_error(y_test, stacked_pred)
print(f"Stacked model MSE: {mse_stacked}")
from sklearn.metrics import r2_score

r2_stacked = r2_score(y_test, stacked_pred)
print(f"Stacked model R-squared: {r2_stacked}")
```

```
Stacked model MSE: 0.3695118488502566
Stacked model R-squared: 0.6321576199500907
```

First, the dataset is split into training and holdout sets using the train_test_split function. The training set is further split into training and validation sets using the same function with the test_size parameter set to 0.2, which means that 20% of the training data is used as the validation set.

Three base models are trained on the training set: Linear Regression, Random Forest Regressor, and Gradient Boosting Regressor. The predictions of these models on the validation set are then used as the input for the meta-model, which is also a Linear Regression model. The meta-model is trained on the validation set to predict the target variable (y_holdout).

Once the meta-model is trained, the three base models are used to make predictions on the test set. These predictions are then used as the input for the meta-model to produce the final stacked predictions.

Classification

- **Aim:**
  Classify whether an Airbnb will be popular so we can determine whether it is worth being recommended.
  (Review per month as y label to classify whether an Airbnb is popular or not)

- **Model:**
  Naive Bayes, KNN, Decision Tree, Random Forest

- **Evaluation:**
  AUC, Confusion Metric, ROC Curve

Classification - Whether recommend

```
df_clf = df.copy()
from sklearn.model_selection import train_test_split
features = ['neighbourhood_group','latitude', 'longitude', 'neighbourhood','room_type','minimum_nights',
            'calculated_host_listings_count', 'availability_365','crimes_neigh',
            'CMPLNT_NUM','distance to nearest subway (km)','log_price']
X_clf = df_clf[[col for col in features]]
X_encoded_clf = pd.get_dummies(X_clf,columns = ['room_type','neighbourhood_group','neighbourhood'])

df_clf['whether_recommend'] = (df_clf['reviews_per_month'] > df_clf['reviews_per_month'].mean()).astype(int)
y_clf = df_clf['whether_recommend']
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_encoded_clf, y_clf, test_size=0.30, random_state=42)
```

GaussianNB

```
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()

classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)

classif_results()
```

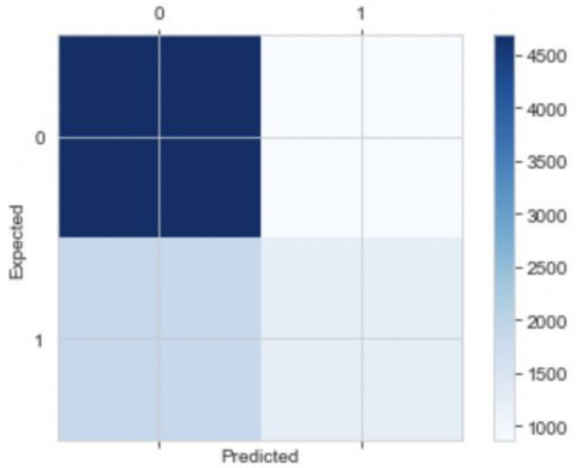
```
Confusion matrix:
 [[4683  869]
 [1867 1231]]
```

KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier()

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_encoded_clf, y_clf,
                                                    test_size=0.30,
                                                    random_state=42)


classifier.fit(X_train,y_train)

y_pred = classifier.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)


classif_results()
```

```
Confusion matrix:
 [[4393 1159]
 [1296 1802]]
```

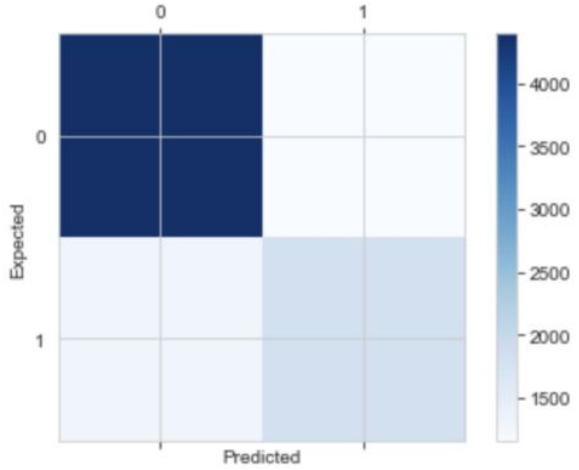
Naive Bayes(left) & KNN(right)



Accuracy 0.683699421965318

	precision	recall	f1-score	support
0	0.71	0.84	0.77	5552
1	0.59	0.40	0.47	3098
accuracy			0.68	8650
macro avg	0.65	0.62	0.62	8650
weighted avg	0.67	0.68	0.67	8650

AUC Score:
0.6204164790708144



Accuracy 0.7161849710982658

	precision	recall	f1-score	support
0	0.77	0.79	0.78	5552
1	0.61	0.58	0.59	3098
accuracy			0.72	8650
macro avg	0.69	0.69	0.69	8650
weighted avg	0.71	0.72	0.71	8650

AUC Score:
0.6864559941991022

DecisionTreeClassifier

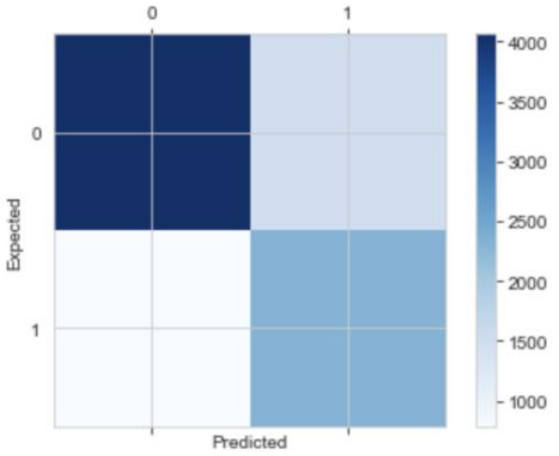
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt

depth_range = [2, 5, 10, 20, 50]
scores = []

for d in depth_range:
    dt = DecisionTreeClassifier(max_depth=d, random_state=42)
    score = cross_val_score(dt, X_train, y_train, cv=5).mean()
    scores.append(score)

y_pred = classifier.predict(X_test)

best_depth = depth_range[scores.index(max(scores))]
print("Best max_depth:", best_depth)

dt = DecisionTreeClassifier(max_depth=best_depth, random_state=42)
dt.fit(X_train,y_train)

y_pred = dt.predict(X_test)
cm = confusion_matrix(y_test,y_pred)
classif_results()
```

```
Best max_depth: 5
Confusion matrix:
 [[4061 1491]
 [ 788 2310]]
```

RandomForestClassifier

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

param_grid = {
    'n_estimators': [10, 50, 100, 500, 1000],
    'max_depth': [3, 5, 7, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
#     'bootstrap': [True, False],
#     'criterion': ['gini', 'entropy']
}

rf = RandomForestClassifier(random_state=42)

grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, n_jobs=-1)
grid_search.fit(X_train, y_train)

print(f'Best hyperparameters: {grid_search.best_params_}')
print(f'Best score: {grid_search.best_score_}')

# Best hyperparameters: {'max_depth': None, 'min_samples_leaf': 2,
#                        'min_samples_split': 2, 'n_estimators': 1000}
# Best score: 0.7465559776232388

rf_model = RandomForestClassifier(**grid_search.best_params_, random_state=42)
rf_model.fit(X_train, y_train)

y_pred = rf_model.predict(X_test)
cm = confusion_matrix(y_test,y_pred)

classif_results()
```

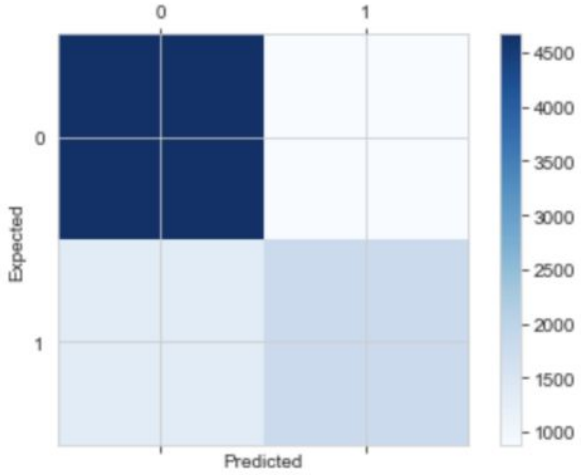
Decision Tree(left) & Random Forest(right)



Accuracy 0.7365317919075145

	precision	recall	f1-score	support
0	0.84	0.73	0.78	5552
1	0.61	0.75	0.67	3098
accuracy			0.74	8650
macro avg	0.72	0.74	0.73	8650
weighted avg	0.76	0.74	0.74	8650

AUC Score:
0.7385452383521581

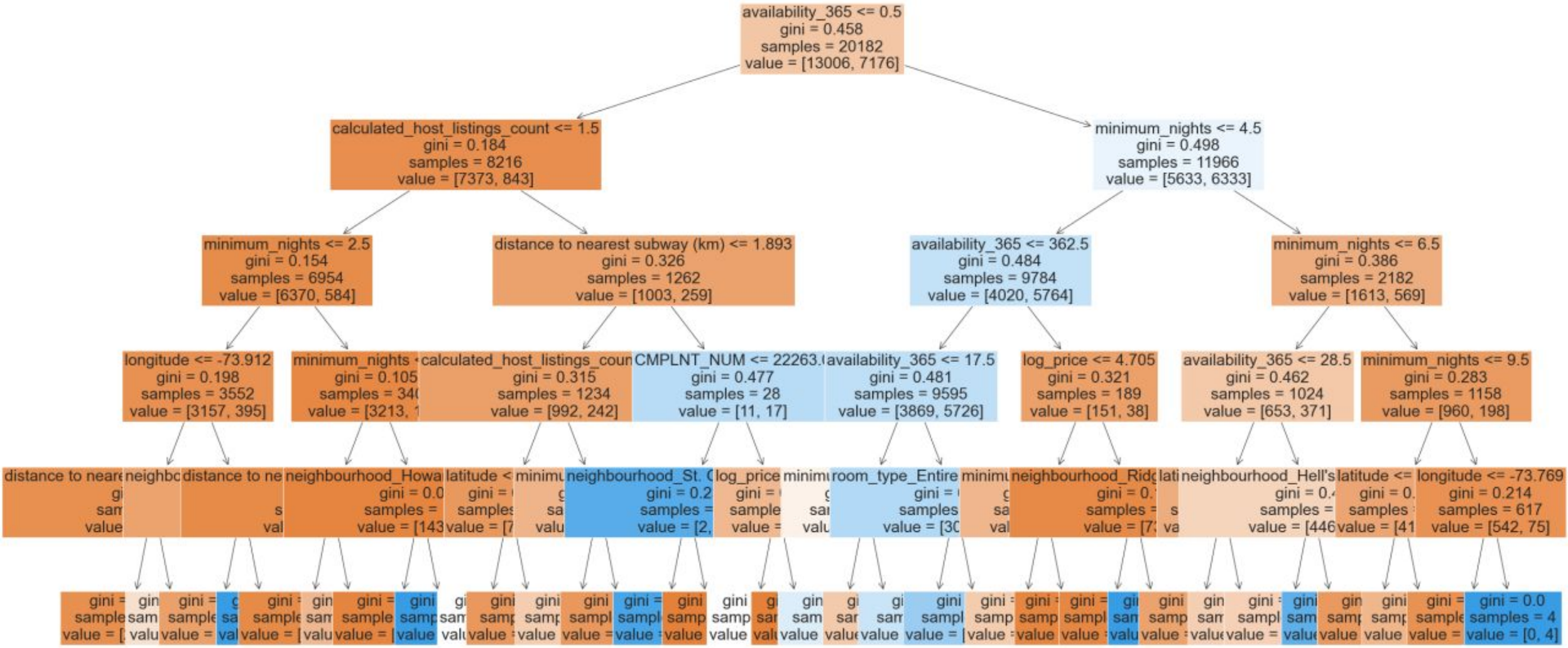


Accuracy 0.7447398843930636

	precision	recall	f1-score	support
0	0.78	0.84	0.81	5552
1	0.67	0.57	0.62	3098
accuracy			0.74	8650
macro avg	0.72	0.71	0.71	8650
weighted avg	0.74	0.74	0.74	8650

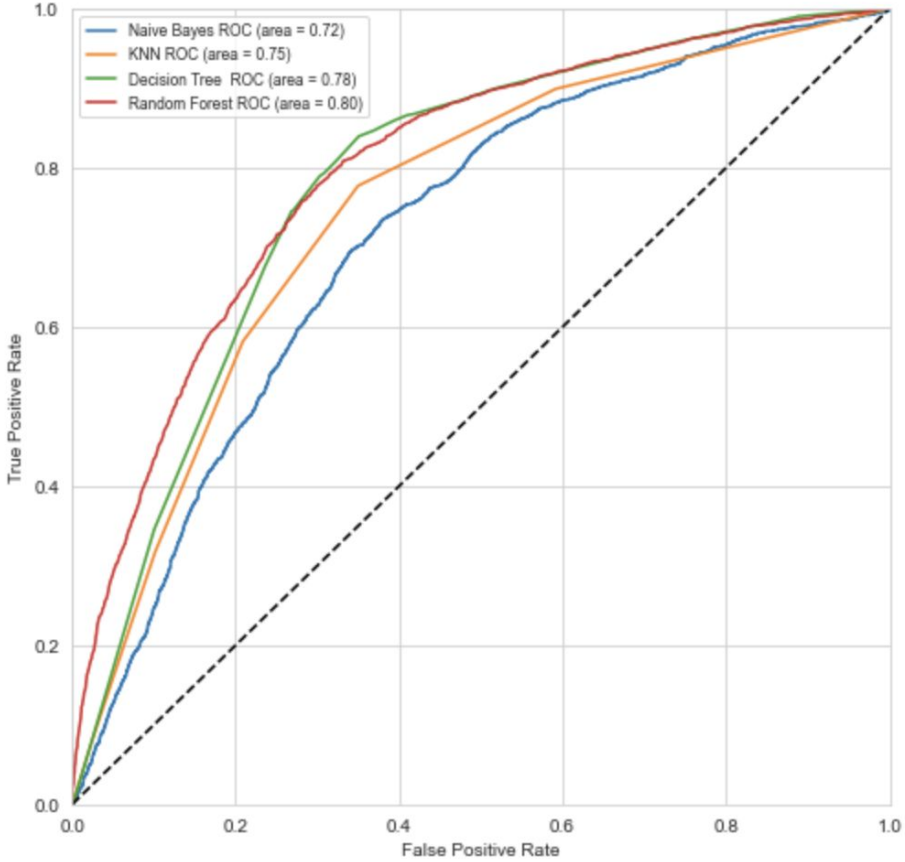
AUC Score:
0.7066314629871833

Decision Tree



ROC Curve

```
knn_model = KNeighborsClassifier()
nb_model = GaussianNB()
des_model = DecisionTreeClassifier(max_depth=best_depth, random_state=42)
clf_model = RandomForestClassifier(n_estimators=best_n, random_state=42)
models = [
    {'label': 'Naive Bayes',
    'model': nb_model},
    {'label': 'KNN',
     'model': knn_model},
    {'label': 'Decision Tree ',
     'model': des_model},
    {'label': 'Random Forest',
    'model': clf_model}
]
from sklearn.metrics import roc_curve, roc_auc_score, auc

plt.clf()
plt.figure(figsize=(8,8))
for m in models:
    m['model'].probability = True
    probas = m['model'].fit(X_train,y_train).predict_proba(X_test)
    fpr, tpr, thresholds = roc_curve(y_test, probas[:, 1])
    roc_auc  = auc(fpr, tpr)
    plt.plot(fpr, tpr, label='%s ROC (area = %0.2f)' % (m['label'], roc_auc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=0, fontsize='small')
plt.show()
```



Our results show that the Random Forest model outperformed the other models in terms of both accuracy and AUC score, with an accuracy of 0.7447 and an AUC score of 0.7066. This suggests that Random Forest was the most effective model for classifying whether an Airbnb is popular and worth recommending.

Although the Decision Tree model had a relatively high accuracy of 0.7365, its AUC score was lower than that of Random Forest, indicating that the model may not perform as well on unseen data.

The K-Nearest Neighbors model had an accuracy of 0.7162 and an AUC score of 0.6864, which is also a reasonable performance but still falls behind the Random Forest model.

Lastly, the Gaussian Naive Bayes model had the lowest accuracy of all the models, at 0.6837, and the lowest AUC score, at 0.6204. This indicates that this model was the least effective for classifying whether an Airbnb is popular and worth recommending.

Overall, our results suggest that Random Forest is the most effective model for our classification task. Its ability to handle non-linear relationships, handle missing values and outliers, and prevent overfitting are likely contributing factors to its superior performance.

Multi Classification - Level of recommendation

```
import pandas as pd
df_rl = df.copy()
df_rl['level'] = pd.cut(df_rl['reviews_per_month'], bins=4, labels=[1, 2, 3, 4])
df_rl.level.value_counts()
```

```
1    19054
2     5058
3     3077
4     1643
Name: level, dtype: int64
```

```
from sklearn.model_selection import train_test_split
features = ['neighbourhood_group','latitude', 'longitude', 'neighbourhood','room_type','minimum_nights',
            'calculated_host_listings_count', 'availability_365','crimes_neigh',
            'CMPLNT_NUM','distance to nearest subway (km)','log_price']
X_clf = df_rl[[col for col in features]]
X_encoded_clf = pd.get_dummies(X_clf,columns = ['room_type','neighbourhood_group','neighbourhood'])
y_clf = df_rl['level']

from imblearn.over_sampling import RandomOverSampler
oversampler = RandomOverSampler(random_state=42)
X_resampled, y_resampled = oversampler.fit_resample(X_encoded_clf, y_clf)
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.30, random_state=42)
# X_train, X_test, y_train, y_test = train_test_split(X_encoded_clf, y_clf, test_size=0.30, random_state=42)
```

```
y_train.value_counts()
```

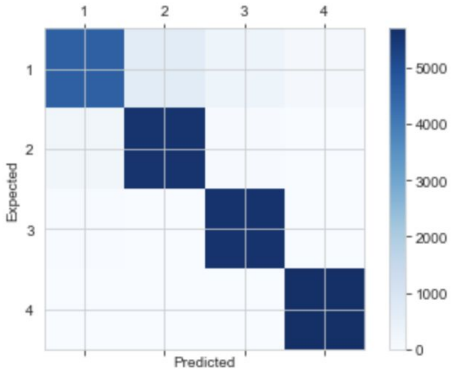
```
3    13420
1    13364
4    13359
2    13208
```

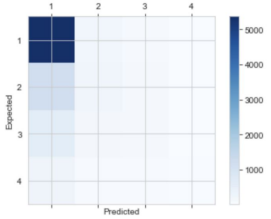
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
estimators_range = [10, 50, 100, 500, 1000]
scores = []
for n in estimators_range:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    score = cross_val_score(rf, X_train, y_train, cv=5).mean()
    scores.append(score)

best_n = estimators_range[scores.index(max(scores))]
print("Best n_estimators:", best_n)
```

```
Best n_estimators: 500
```

```
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
rf = RandomForestClassifier(n_estimators=best_n, random_state=42)
rf.fit(X_train,y_train)
y_pred = rf.predict(X_test)
cm = confusion_matrix(y_test,y_pred)
classif_results()
```

```
Confusion matrix:
 [[4619  658  300  113]
 [ 181 5580   64   21]
 [  25   10 5592    7]
 [   0    0    0 5695]]
```






```
Accuracy 0.9396894817406517
```

	precision	recall	f1-score	support
1	0.96	0.81	0.88	5690
2	0.89	0.95	0.92	5846
3	0.94	0.99	0.96	5634
4	0.98	1.00	0.99	5695
accuracy			0.94	22865
macro avg	0.94	0.94	0.94	22865
weighted avg	0.94	0.94	0.94	22865

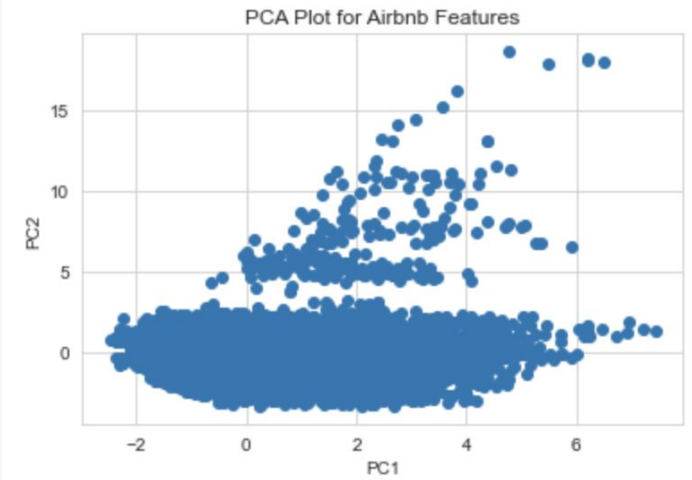
Furthermore, we went beyond just identifying whether an Airbnb is popular and divided the review per month into four different levels, ranging from level one to level four. Using Random Forest, we then aimed to determine the recommend level for each Airbnb.

First, we categorizing the 'reviews_per_month' column into 4 bins. The multi-classification results show that the Random Forest model was able to classify the Airbnb listings into four different recommendation levels with an overall accuracy of 0.6675. The confusion matrix indicates that the model was able to accurately predict the majority of the listings in the top recommendation level (Level 1) with a precision of 0.72 and a recall of 0.94. However, the precision and recall were much lower for the other three levels, indicating that the model struggled to accurately classify listings in those levels.

To deal with the imbalanced data, we used RandomOverSampler approach to generate synthetic samples for the minority classes to balance the number of samples across different classes. Then, the resampled data is split into train and test sets and train the model. The results shows that there is a significantly improvement compared to the original result after oversampling the imbalanced data. The overall accuracy has increased from 0.6675 to 0.9397, indicating that the model is now much better at correctly classifying the different levels of reviews.

PCA

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

features = ['latitude', 'longitude', 'minimum_nights', 'number_of_reviews', 'calculated_host_listings_count',
            'availability_365', 'distance to nearest subway (km)']
df_subset = df[features]

scaler = StandardScaler()
df_std = scaler.fit_transform(df_subset)

pca = PCA(n_components=2)

principal_components = pca.fit_transform(df_std)

import matplotlib.pyplot as plt

plt.scatter(principal_components[:, 0], principal_components[:, 1])
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('PCA Plot for Airbnb Features')
plt.show()

loadings = pd.DataFrame(pca.components_.T, columns=['PC1', 'PC2'], index=features)

print(loadings)
```



	PC1	PC2
latitude	-0.119000	-0.722556
longitude	0.336585	-0.417321
minimum_nights	-0.150336	0.070704
number_of_reviews	0.444044	-0.081346
calculated_host_listings_count	0.489296	-0.069275
availability_365	0.578372	-0.026303
distance to nearest subway (km)	0.280801	0.535395

Conclusion

- Introduce new variables using latitude and longitude
- Using different model fit the dataset.
- Try to analyze the result from model and gain some insight about our data.
- Decision tree visualization.
- Multi-classification.