Airbnb Price Prediction and Insights

Part A

[106]:

```
#1. Data Exploration and Preprocessing
## Load the Data
```

[109]:

```
import pandas as pd
```

[112]:

```
# Load the dataset
df = pd.read_csv('Airbnb_data..csv')  # Adjust path if needed
```

[115]:

```
# Initial Exploration
df.head()
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74111 entries, 0 to 74110
Data columns (total 29 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   id                      74111 non-null  int64
 1   log_price               74111 non-null  float64
 2   property_type           74111 non-null  object
 3   room_type               74111 non-null  object
 4   amenities               74111 non-null  object
 5   accommodates            74111 non-null  int64
 6   bathrooms               73911 non-null  float64
 7   bed_type                74111 non-null  object
 8   cancellation_policy     74111 non-null  object
 9   cleaning_fee            74111 non-null  bool
 10  city                    74111 non-null  object
 11  description             74111 non-null  object
 12  first_review            58247 non-null  object
 13  host_has_profile_pic    73923 non-null  object
 14  host_identity_verified  73923 non-null  object
 15  host_response_rate      55812 non-null  object
 16  host_since              73923 non-null  object
 17  instant_bookable        74111 non-null  object
 18  last_review             58284 non-null  object
 19  latitude                74111 non-null  float64
 20  longitude               74111 non-null  float64
 21  name                    74111 non-null  object
 22  neighbourhood           67239 non-null  object
 23  number_of_reviews       74111 non-null  int64
 24  review_scores_rating    57389 non-null  float64
 25  thumbnail_url           65895 non-null  object
 26  zipcode                 73143 non-null  object
 27  bedrooms                74020 non-null  float64
 28  beds                    73980 non-null  float64
dtypes: bool(1), float64(7), int64(3), object(18)
memory usage: 15.9+ MB
```

[115]:

```
                 id     log_price  accommodates     bathrooms      latitude  \
count  7.411100e+04  74111.000000  74111.000000  73911.000000  74111.000000
mean   1.126662e+07      4.782069      3.155146      1.235263     38.445958
std    6.081735e+06      0.717394      2.153589      0.582044      3.080167
min    3.440000e+02      0.000000      1.000000      0.000000     33.338905
25%    6.261964e+06      4.317488      2.000000      1.000000     34.127908
50%    1.225415e+07      4.709530      2.000000      1.000000     40.662138
75%    1.640226e+07      5.220356      4.000000      1.000000     40.746096
max    2.123090e+07      7.600402     16.000000      8.000000     42.390437

          longitude  number_of_reviews  review_scores_rating      bedrooms  \
count  74111.000000       74111.000000          57389.000000  74020.000000
mean     -92.397525          20.900568             94.067365      1.265793
std       21.705322          37.828641              7.836556      0.852143
min     -122.511500           0.000000             20.000000      0.000000
25%     -118.342374           1.000000             92.000000      1.000000
50%      -76.996965           6.000000             96.000000      1.000000
75%      -73.954660          23.000000            100.000000      1.000000
max      -70.985047         605.000000            100.000000     10.000000

               beds
count  73980.000000
mean       1.710868
std        1.254142
min        0.000000
25%        1.000000
50%        1.000000
75%        2.000000
max       18.000000
```

[118]:

```
#  Data Cleaning
## Check for missing values
df.isnull().sum()
```

[118]:
```
id                            0
log_price                     0
property_type                 0
room_type                     0
amenities                     0
accommodates                  0
bathrooms                   200
bed_type                      0
cancellation_policy           0
cleaning_fee                  0
city                          0
description                   0
first_review              15864
host_has_profile_pic        188
host_identity_verified      188
host_response_rate        18299
host_since                  188
instant_bookable              0
last_review               15827
latitude                      0
longitude                     0
name                          0
neighbourhood              6872
number_of_reviews             0
review_scores_rating      16722
thumbnail_url              8216
zipcode                     968
bedrooms                     91
beds                        131
dtype: int64
```

[121]:
```python
# Handle missing values (replace inplace=True with reassignment)

df['bathrooms'] = df['bathrooms'].fillna(df['bathrooms'].median())
print("Missing 'bathrooms' filled with median:", df['bathrooms'].median())

df['bedrooms'] = df['bedrooms'].fillna(df['bedrooms'].median())
print("Missing 'bedrooms' filled with median:", df['bedrooms'].median())

df['beds'] = df['beds'].fillna(df['beds'].median())
print("Missing 'beds' filled with median:", df['beds'].median())

df['review_scores_rating'] = df['review_scores_rating'].
 ↪fillna(df['review_scores_rating'].median())
print("Missing 'review_scores_rating' filled with median:",␣
 ↪df['review_scores_rating'].median())
```

```
df['host_response_rate'] = df['host_response_rate'].fillna("0%")
print("Missing 'host_response_rate' filled with default value '0%'")
```

```
Missing 'bathrooms' filled with median: 1.0
Missing 'bedrooms' filled with median: 1.0
Missing 'beds' filled with median: 1.0
Missing 'review_scores_rating' filled with median: 96.0
Missing 'host_response_rate' filled with default value '0%'
```

[124]:

```
# Fill missing values first (if not already done)
df['host_response_rate'] = df['host_response_rate'].fillna('0%')

# Convert to string, remove '%', convert to float, divide by 100
df['host_response_rate'] = df['host_response_rate'].astype(str).str.rstrip('%').
  ↪astype(float) / 100.0

print("Converted 'host_response_rate' to numeric format (0.0 to 1.0 range).")
```

```
Converted 'host_response_rate' to numeric format (0.0 to 1.0 range).
```

[127]:

```
# Extract number of amenities by counting items in the string
df['num_amenities'] = df['amenities'].apply(lambda x: len(x.split(',')) if pd.
  ↪notnull(x) else 0)
print("Created new feature 'num_amenities' by counting amenities per listing.")
```

```
Created new feature 'num_amenities' by counting amenities per listing.
```

[130]:

```
# Drop only the columns that exist in the DataFrame
columns_to_drop = ['id', 'name', 'description', 'thumbnail_url']
existing_columns_to_drop = [col for col in columns_to_drop if col in df.columns]

df = df.drop(columns=existing_columns_to_drop)
print(f"Dropped columns: {existing_columns_to_drop}")
```

```
Dropped columns: ['id', 'name', 'description', 'thumbnail_url']
```

[133]:

```
# Define the boolean columns
bool_cols = ['cleaning_fee', 'host_has_profile_pic', 'host_identity_verified',␣
  ↪'instant_bookable']

# Map 't' to 1 and 'f' to 0
for col in bool_cols:
    df[col] = df[col].map({'t': 1, 'f': 0, True: 1, False: 0})
    print(f"Converted column '{col}' to integers (1 for True/'t', 0 for False/
  ↪'f').")
```

```
Converted column 'cleaning_fee' to integers (1 for True/'t', 0 for False/'f').
Converted column 'host_has_profile_pic' to integers (1 for True/'t', 0 for
False/'f').
Converted column 'host_identity_verified' to integers (1 for True/'t', 0 for
False/'f').
Converted column 'instant_bookable' to integers (1 for True/'t', 0 for
False/'f').
```

[136]:

```
#  2.Model Development
## Split the Data
```

[139]:

```
from sklearn.model_selection import train_test_split
from xgboost import XGBRegressor
```

[142]:

```
# Step 1: Drop non-numeric or irrelevant object columns
columns_to_drop = ['amenities', 'first_review', 'last_review', 'host_since',␣
 ↪'zipcode']
X = df.drop(columns=['log_price'] + columns_to_drop)

# Step 2: One-hot encode categorical columns
X = pd.get_dummies(X, drop_first=True)
print("Applied one-hot encoding to categorical features.")

# Step 3: Target variable
y = df['log_price']

# Step 4: Split data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
print(f"Split complete: {X_train.shape[0]} training rows, {X_test.shape[0]}␣
 ↪testing rows.")

# Step 5: Train the model
from xgboost import XGBRegressor
model = XGBRegressor()
model.fit(X_train, y_train)
print("Model training complete.")
```

```
Applied one-hot encoding to categorical features.
Split complete: 59288 training rows, 14823 testing rows.
Model training complete.
```

[144]:

```
#3.Model Evaluation (10 Marks)
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
y_pred = model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
```

```
r2 = r2_score(y_test, y_pred)
print(f'RMSE: {rmse:.2f}')
print(f'MAE: {mae:.2f}')
print(f'R²: {r2:.2f}')
```

```
RMSE: 0.39
MAE: 0.28
R²: 0.71
```

[147]:

```
# 4.Insights and Visualization
## Feature Importance
```

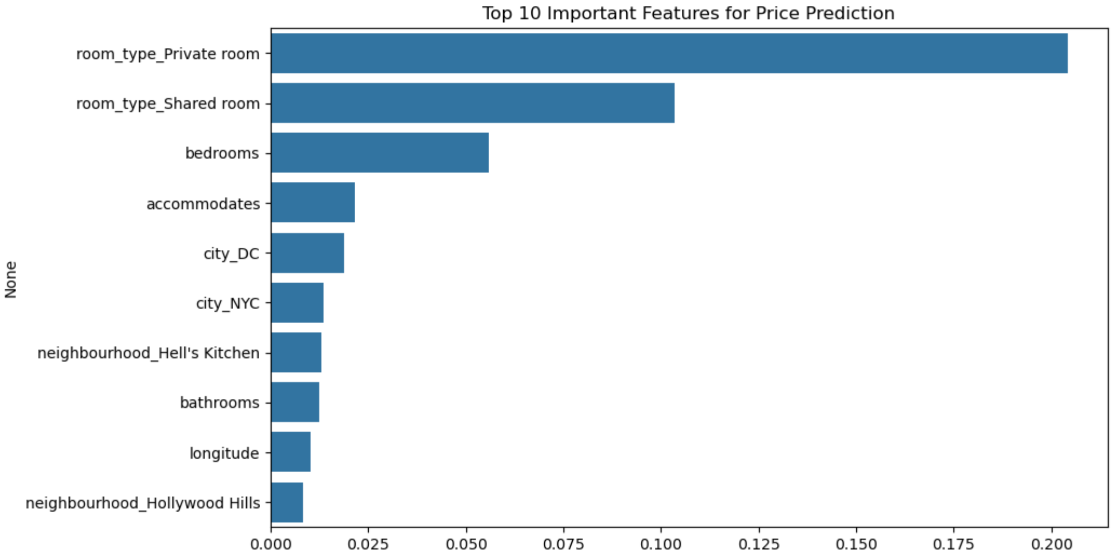
[150]:

```
import matplotlib.pyplot as plt
import seaborn as sns
```

[153]:

```
feature_importances = pd.Series(model.feature_importances_, index=X.columns)
top_features = feature_importances.sort_values(ascending=False)[:10]

plt.figure(figsize=(10,6))
sns.barplot(x=top_features.values, y=top_features.index)
plt.title("Top 10 Important Features for Price Prediction")
plt.show()
```



[ ]: