MACHINE LEARNING FINAL PROJECT PART-A

NAME: Gazi Aman Khan BATCH: 15th May(DataScience)

EXPLANATION VIDEO LINK:https://www.loom.com/share/7f0330f13e3e43f981a5c9d1d12e0192

PROBLEM STATEMENT The objective of this analysis is to predict the listing prices of Airbnb properties based on various factors such as location, property type, number of bedrooms, amenities, and other listing features. By building and evaluating regression models, the goal is to identify key factors influencing price variations and develop a model capable of accurately estimating the price of new Airbnb listings.

This will help hosts understand pricing patterns and enable data-driven pricing decisions, improving competitiveness and revenue potential.

LIBRARIES USED: pandas, numpy, scikit-learn, matplotlib, seaborn, XGBoost.

In [7]:
```
!pip install xgboost
```
```
Requirement already satisfied: xgboost in c:\users\hp\anaconda3\lib\site-packages (3.1.1)
Requirement already satisfied: numpy in c:\users\hp\anaconda3\lib\site-packages (from xgboost) (2.1.3)
Requirement already satisfied: scipy in c:\users\hp\anaconda3\lib\site-packages (from xgboost) (1.15.3)
```

AIRBNB PRICE PREDICTION AND INSIGHTS

In [41]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from xgboost import XGBRegressor
from xgboost import plot_importance
```

In [9]:
```
#MODEL 1- LINEAR REGRESSION
```

In [10]:
```
#TASK-1 :Data Exploration and Preprocessing
```

In [11]:
```
df=pd.read_csv('E:\Course datasets\ML Datasets\Airbnb_data.csv')
df.head(5)
```

Out[11]:

	id	log_price	property_type	room_type	amenities	accommodates	bathrooms	bed_type	cancellation_policy	cleaning_fee	...	latitude	longitude	name	neighbourhood	number_of_reviews	review_scores_rating	thumbnail_url
0	6901257	5.010635	Apartment	Entire home/apt	{"Wireless Internet","Air conditioning",Kitche...	3	1.0	Real Bed	strict	True	...	40.696524	-73.991617	Beautiful brownstone 1-bedroom	Brooklyn Heights	2	100.0	https://a0.muscache.com/im/pictures/6d7cbbf7-c...
1	6304928	5.129899	Apartment	Entire home/apt	{"Wireless Internet","Air conditioning",Kitche...	7	1.0	Real Bed	strict	True	...	40.766115	-73.989040	Superb 3BR Apt Located Near Times Square	Hell's Kitchen	6	93.0	https://a0.muscache.com/im/pictures/348a55fe-4...
2	7919400	4.976734	Apartment	Entire home/apt	{TV,"Cable TV","Wireless Internet","Air condit...	5	1.0	Real Bed	moderate	True	...	40.808110	-73.943756	The Garden Oasis	Harlem	10	92.0	https://a0.muscache.com/im/pictures/6fae5362-9...
3	13418779	6.620073	House	Entire home/apt	{TV,"Cable TV",Internet,"Wireless Internet",Ki...	4	1.0	Real Bed	flexible	True	...	37.772004	-122.431619	Beautiful Flat in the Heart of SF!	Lower Haight	0	NaN	https://a0.muscache.com/im/pictures/72208dad-9...
4	3808709	4.744932	Apartment	Entire home/apt	{TV,Internet,"Wireless Internet","Air conditio...	2	1.0	Real Bed	moderate	True	...	38.925627	-77.034596	Great studio in midtown DC	Columbia Heights	4	40.0	NaN

5 rows × 29 columns

In [12]:
```
df.info()
```
```
<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
```

In [13]:
```
df.isnull().sum()    #MISSING VALUES
```
Out[13]:
```
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
```

In [14]:
```
df.describe()
```

Out[14]:

	id	log_price	accommodates	bathrooms	latitude	longitude	number_of_reviews	review_scores_rating	bedrooms	beds
count	7.411100e+04	74111.000000	74111.000000	73911.000000	74111.000000	74111.000000	74111.000000	57389.000000	74020.000000	73980.000000
mean	1.126662e+07	4.782069	3.155146	1.235263	38.445958	-92.397525	20.900568	94.067365	1.265793	1.710868
std	6.081735e+06	0.717394	2.153589	0.582044	3.080167	21.705322	37.828641	7.836556	0.852143	1.254142
min	3.440000e+02	0.000000	1.000000	0.000000	33.338905	-122.511500	0.000000	20.000000	0.000000	0.000000
25%	6.261964e+06	4.317488	2.000000	1.000000	34.127908	-118.342374	1.000000	92.000000	1.000000	1.000000
50%	1.225415e+07	4.709530	2.000000	1.000000	40.662138	-76.996965	6.000000	96.000000	1.000000	1.000000
75%	1.640226e+07	5.220356	4.000000	1.000000	40.746096	-73.954660	23.000000	100.000000	1.000000	2.000000
max	2.123090e+07	7.600402	16.000000	8.000000	42.390437	-70.985047	605.000000	100.000000	10.000000	18.000000

In [15]:
```
#outliers
df['price']= np.exp(df['log_price'])

sns.boxplot(x=df['price'])
plt.title("Outliers in price")
plt.show()
```

In [16]:
```
#HANDLING MISSING VALUES
num_cols = df.select_dtypes(include=['int64','float64']).columns   #numerical missing values to median
df[num_cols] = df[num_cols].fillna(df[num_cols].median())

cat_cols = df.select_dtypes(include=['object']).columns     #categorical missing values to "unknown"
df[cat_cols] = df[cat_cols].fillna('Unknown')

df.isnull().sum()
```
Out[16]:
```
id                        0
log_price                 0
property_type             0
room_type                 0
amenities                 0
accommodates              0
bathrooms                 0
bed_type                  0
cancellation_policy       0
cleaning_fee              0
city                      0
description               0
first_review              0
host_has_profile_pic      0
host_identity_verified    0
host_response_rate        0
host_since                0
instant_bookable          0
last_review               0
latitude                  0
longitude                 0
name                      0
neighbourhood             0
number_of_reviews         0
review_scores_rating      0
thumbnail_url             0
zipcode                   0
bedrooms                  0
beds                      0
price                     0
dtype: int64
```

In [17]:
```
df['host_since'] = pd.to_datetime(df['host_since'], errors='coerce')
df['last_review'] = pd.to_datetime(df['last_review'], errors='coerce')
```
```
C:\Users\HP\AppData\Local\Temp\ipykernel_3912\2637232071.py:1: UserWarning: Parsing dates in %d-%m-%Y format when dayfirst=False (the default) was specified. Pass `dayfirst=True` or specify a format to silence this warning.
  df['host_since'] = pd.to_datetime(df['host_since'], errors='coerce')
C:\Users\HP\AppData\Local\Temp\ipykernel_3912\2637232071.py:2: UserWarning: Parsing dates in %d-%m-%Y format when dayfirst=False (the default) was specified. Pass `dayfirst=True` or specify a format to silence this warning.
  df['last_review'] = pd.to_datetime(df['last_review'], errors='coerce')
```

In [18]:
```
df['host_experience_days'] = (pd.Timestamp.today() - df['host_since']).dt.days  #host experience indays
df['n_amenities'] = df['amenities'].apply(lambda x: len(str(x).split(',')))     #count of amenities in each row
```

In [19]:
```
#STANDARDIZING THE DATA
scaler=StandardScaler()
df[['accommodates','bedrooms','beds','n_amenities']] = scaler.fit_transform(
    df[['accommodates','bedrooms','beds','n_amenities']])
```

In [20]:
```
#TASK-2:Model Development.
```

In [21]:
```
cat_cols = ['property_type', 'room_type', 'neighbourhood']    #SELECTGING ONLY MEANINGFUL COLUMNS BECAUSE OF MEMORY ISSUES
num_cols = ['accommodates','bathrooms','bedrooms','beds','number_of_reviews',
            'review_scores_rating','n_amenities','host_experience_days']
```

In [22]:
```
x=df[cat_cols+num_cols]
y=df['log_price']
```

In [23]:
```
#SPLIT THE DATA
x_train,x_test,y_train,y_test= train_test_split(x, y, test_size=0.2, random_state=42)
```

In [24]:
```
#HANDLING CATEGORICAL COLUMNS
x_train = pd.get_dummies(x_train, columns=cat_cols, drop_first=True)
x_test = pd.get_dummies(x_test, columns=cat_cols, drop_first=True)

x_test=x_test.reindex(columns=x_train.columns, fill_value=0)   #ensuring both datasets have same colums in order
```

In [25]:
```
x_train = x_train.fillna(x_train.median())  #handling remaining NaN values in numeric colums
x_test = x_test.fillna(x_train.median())
```

LINEAR REGRESSION

In [26]:
```
#TRAIN MODEL
model= LinearRegression()
model.fit(x_train,y_train)
```
Out[26]:
```
LinearRegression
LinearRegression()
```

In [27]:
```
#PREDICTIONS
y_pred=model.predict(x_test)
```

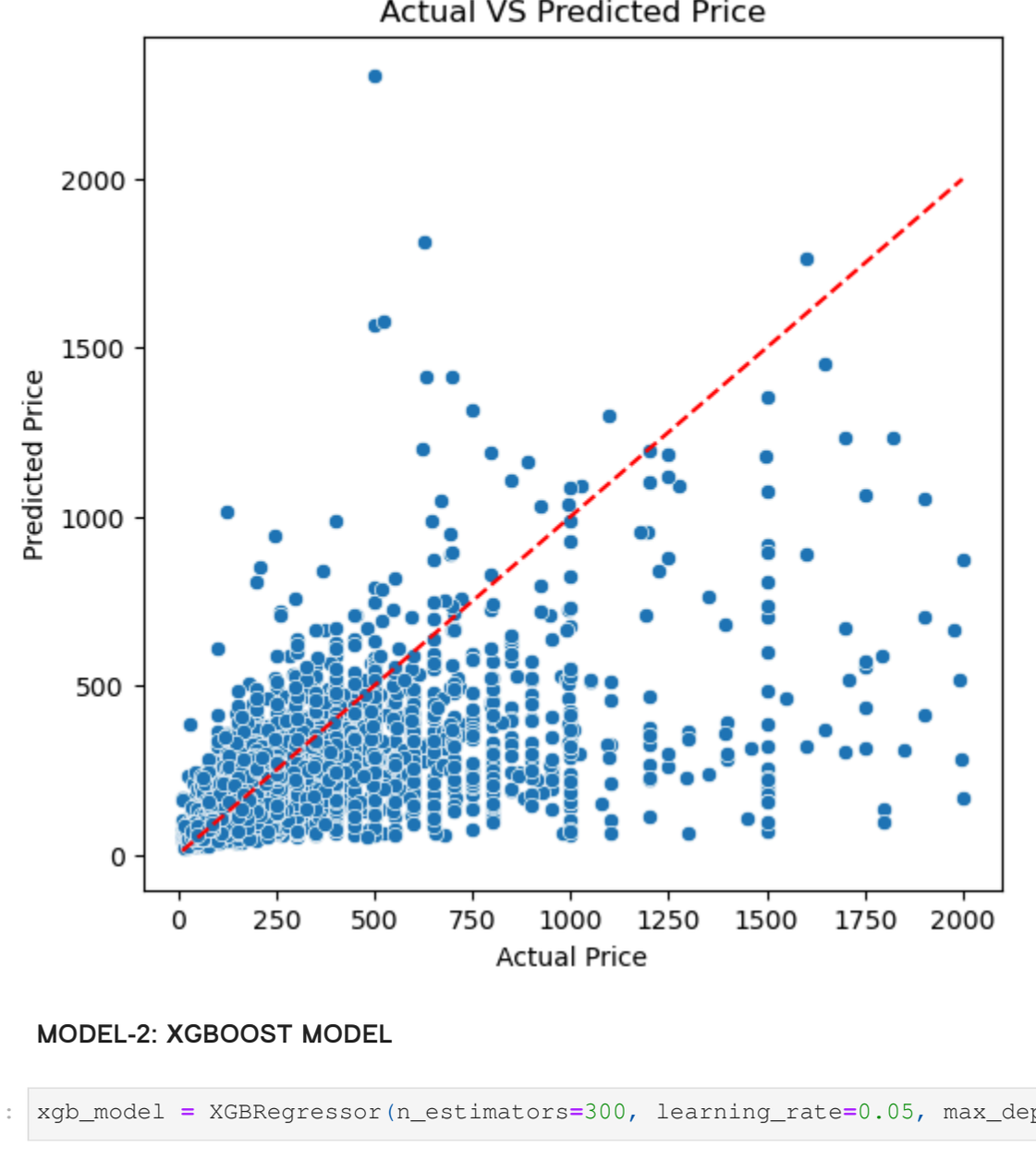
In [28]:
```
#TASK-3:EVALUATING THE MODEL'S PERFORMANCE
mse=mean_squared_error(y_test, y_pred)
rmse=np.sqrt(mse)
mae=mean_absolute_error(y_test, y_pred)
r2= r2_score(y_test, y_pred)

print("RMSE:",rmse)
print("MAE:",mae)
print("R2 SCORE:", r2)
```
```
RMSE: 0.4236632810941468
MAE: 0.3075482177263002
R2 SCORE: 0.6506102257110424
```

In [29]:
```
y_pred_price = np.exp(y_pred)
y_test_price = np.exp(y_test)


pred_df = pd.DataFrame({
    'Actual Price': y_test_price,
    'Predicted Price': y_pred_price})


pred_df.head(10)
```

Out[29]:

	Actual Price	Predicted Price
4079	45.0	52.367957
33735	140.0	121.137535
69475	140.0	127.768853
454	130.0	268.934771
25153	39.0	72.356799
42304	126.0	144.191819
22686	185.0	199.441368
20776	295.0	170.844461
14934	35.0	47.790085
39634	69.0	199.059872

In [30]:
```
#SCATTER PLOT OF ACTUAL VS PREDICTED PRICES
plt.figure(figsize=(6,6))
sns.scatterplot(x=y_test_price, y=y_pred_price)
plt.title("Actual VS Predicted Price")
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.plot([y_test_price.min(), y_test_price.max()],
         [y_test_price.min(), y_test_price.max()], 'r--')
plt.show()
```



MODEL-2: XGBOOST MODEL

In [32]:
```
xgb_model = XGBRegressor(n_estimators=300, learning_rate=0.05, max_depth=6, random_state=42, n_jobs=-1)
```

In [33]:
```
#train the data
xgb_model.fit(x_train, y_train)
```
Out[33]:
```
XGBRegressor
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.05, max_bin=None, max_cat_threshold=None,
             max_cat_to_onehot=None, max_delta_step=None, max_depth=6,
             max_leaves=None, min_child_weight=None, missing=nan,
```

In [34]:
```
#PREDICTIONS
yxg_pred= xgb_model.predict(x_test)
```

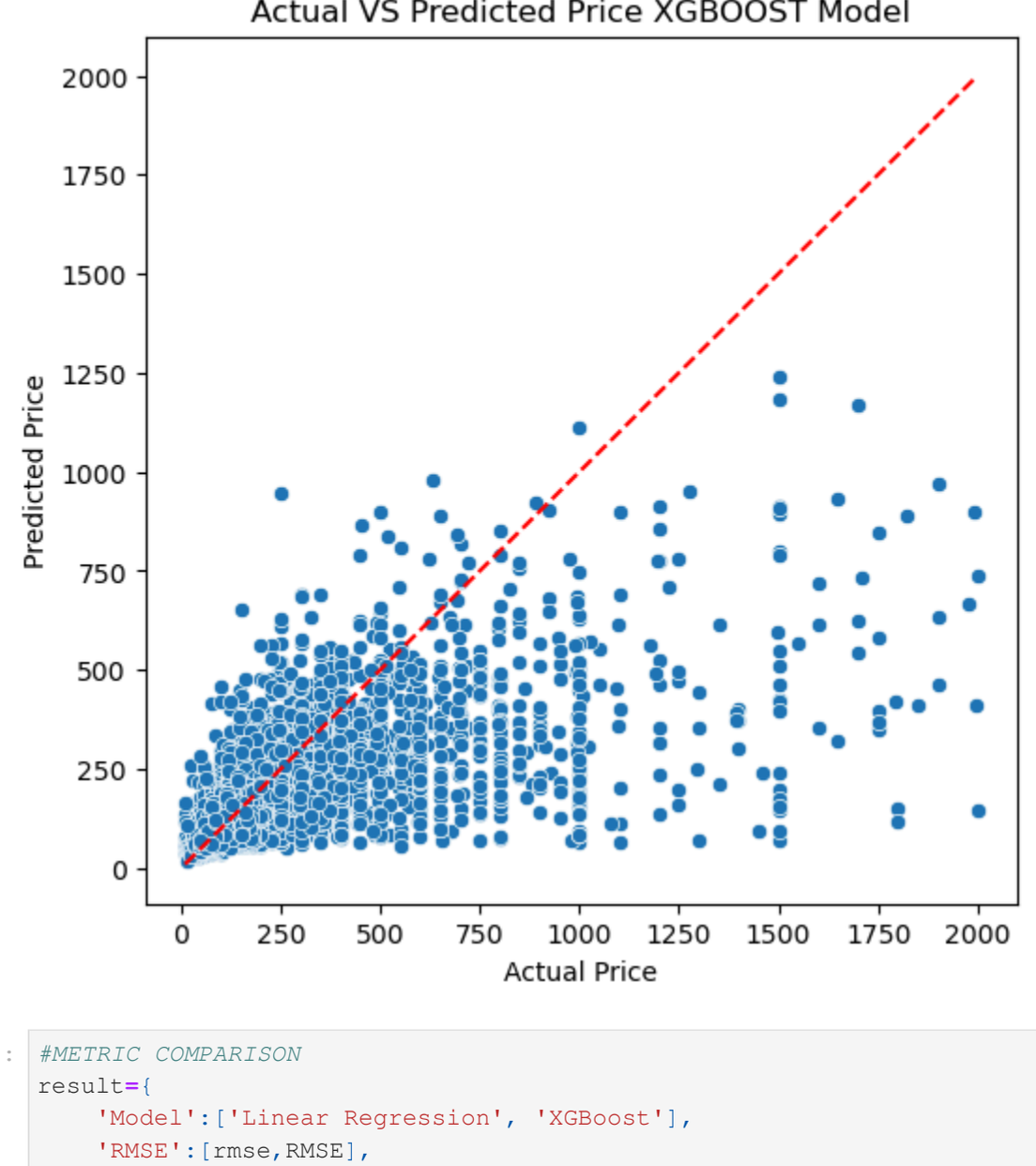
In [35]:
```
#EVALUATE THE DATA
MSE=mean_squared_error(y_test, yxg_pred)
RMSE=np.sqrt(MSE)
MAE=mean_absolute_error(y_test, yxg_pred)
R2X= r2_score(y_test, yxg_pred)


print("RMSE:", RMSE)
print("MAE:", MAE)
print("R2 score:", R2X)
```
```
RMSE: 0.42864549448408484
MAE: 0.3164514480966707
R2 score: 0.6423443699193158
```

In [36]:
```
yx_pred_price= np.exp(yxg_pred)
y_test_price= np.exp(y_test)

predx_df= pd.DataFrame({
        'Actual price' : y_test_price,
        'Predicted price': yx_pred_price})

predx_df.head(10)
```

Out[36]:

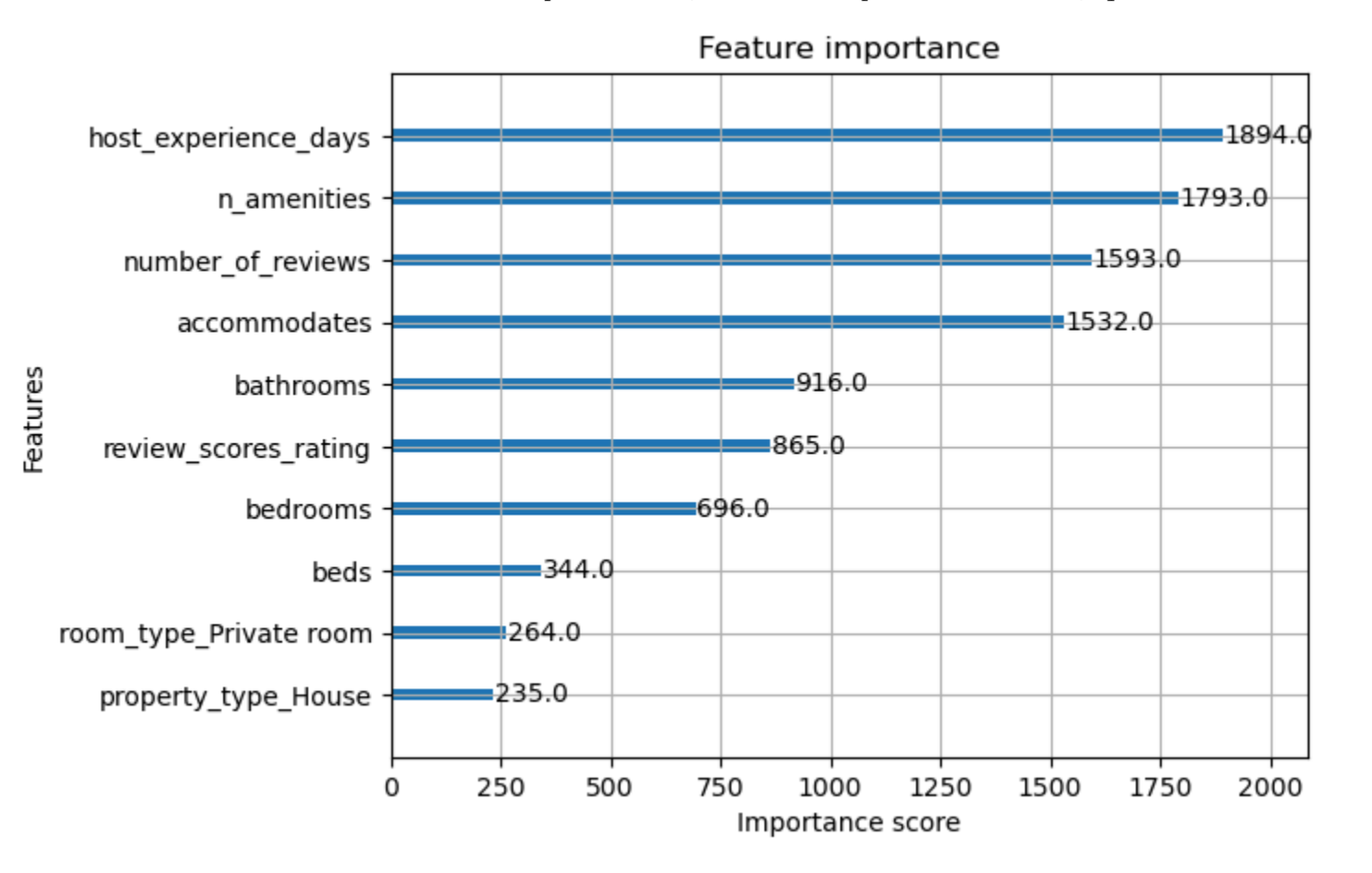
	Actual price	Predicted price
4079	45.0	61.249836
33735	140.0	134.211304
69475	140.0	140.262390
454	130.0	213.955887
25153	39.0	66.719208
42304	126.0	154.585922
22686	185.0	265.870850
20776	295.0	130.928757
14934	35.0	62.748131
39634	69.0	94.732292

In [37]:
```
#SCATTER PLOT OF ACTUAL VS PREDICTED PRICES
plt.figure(figsize=(6,6))
sns.scatterplot(x=y_test_price, y=yx_pred_price)
plt.title("Actual VS Predicted Price XGBOOST Model")
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.plot([y_test_price.min(), y_test_price.max()],
         [y_test_price.min(), y_test_price.max()], 'r--')
plt.show()
```



In [38]:
```
#METRIC COMPARISON
result={
    'Model':['Linear Regression', 'XGBoost'],
    'RMSE':[rmse,RMSE],
    'MAE':[mae,MAE],
    'R2 Score':[r2, R2X]}

result_df=pd.DataFrame(result)
print(result_df)
```
```
               Model      RMSE       MAE  R2 Score
0  Linear Regression  0.423663  0.307548  0.650610
1            XGBoost  0.428645  0.316451  0.642344
```

In [42]:
```
#INSIGHTS
plot_importance(xgb_model, max_num_features=10)
```
Out[42]:
```
<Axes: title={'center': 'Feature importance'}, xlabel='Importance score', ylabel='Features'>
```



Both models achieved R² scores of approximately 0.65, indicating that they can explain around 65% of the variation in Airbnb listing prices. This shows that the model can predict prices for new listings with reasonable accuracy.

MODEL EVALUATION INSIGHTS

Linear Regression

1.Achieved the lowest RMSE (0.4237) and lowest MAE (0.3075), indicating more accurate predictions on average. 2.R² = 0.65 means it explains about 65% of the variation in Airbnb listing prices. 3.Performs consistently and is interpretable, making it a reliable baseline model.

Insight: Linear Regression provides the best balance of simplicity and accuracy for predicting listing prices.

XGBoost 1.Shows RMSE = 0.4286 and MAE = 0.3165, slightly worse than Linear Regression. 2.R² = 0.64, meaning it explains 64% of the price variation. 3.Being a tree-based model, it captures non-linear patterns, but in this dataset, those patterns don’t add much benefit.

Insight: XGBoost performs nearly as well as Linear Regression, but the added complexity doesn’t significantly improve prediction accuracy.