In [1]:
```
# Import Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.cluster import KMeans
import datetime as dt
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
# Load and Inspect Data

df = pd.read_csv("Airbnb_data.csv")

# Basic info
df.shape
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
```

Out[3]:

	id	log_price	accommodates	bathrooms	latitude	longitu
count	7.411100e+04	74111.000000	74111.000000	73911.000000	74111.000000	74111.0000
mean	1.126662e+07	4.782069	3.155146	1.235263	38.445958	-92.3975
std	6.081735e+06	0.717394	2.153589	0.582044	3.080167	21.7053
min	3.440000e+02	0.000000	1.000000	0.000000	33.338905	-122.5115
25%	6.261964e+06	4.317488	2.000000	1.000000	34.127908	-118.3423
50%	1.225415e+07	4.709530	2.000000	1.000000	40.662138	-76.9969
75%	1.640226e+07	5.220356	4.000000	1.000000	40.746096	-73.9546
max	2.123090e+07	7.600402	16.000000	8.000000	42.390437	-70.9850

1. Data Exploration and Preprocessing

In [4]:
```
# Handle Missing Values

# Fill numerical nulls
num_cols = ['bathrooms', 'bedrooms', 'beds', 'review_scores_rating']
for col in num_cols:
    df[col].fillna(df[col].median(), inplace=True)

# Fill text and boolean nulls
df['cleaning_fee'].fillna(0, inplace=True)
df['host_response_rate'] = df['host_response_rate'].str.replace('%', '').astype(flo
df['host_response_rate'].fillna(df['host_response_rate'].mean(), inplace=True)

# Convert date fields
for date_col in ['host_since', 'first_review', 'last_review']:
    df[date_col] = pd.to_datetime(df[date_col], errors='coerce')

# Host tenure
df['host_tenure'] = (pd.to_datetime('today') - df['host_since']).dt.days

# Review time gaps
df['days_since_last_review'] = (pd.to_datetime('today') - df['last_review']).dt.day
df['days_active'] = (df['last_review'] - df['first_review']).dt.days

# Fill missing tenure or review gaps
df['host_tenure'].fillna(df['host_tenure'].median(), inplace=True)
df['days_since_last_review'].fillna(df['days_since_last_review'].median(), inplace=
df['days_active'].fillna(0, inplace=True)
```

In [5]:
```
# Feature Engineering

# Amenity count
df['amenities_count'] = df['amenities'].apply(lambda x: len(x.split(',')) if pd.not

# Description & name length
df['description_length'] = df['description'].apply(lambda x: len(str(x)))
df['name_length'] = df['name'].apply(lambda x: len(str(x)))

# Binary encoding
binary_cols = ['host_has_profile_pic', 'host_identity_verified', 'instant_bookable'
for col in binary_cols:
    df[col] = df[col].map({'t': 1, 'f': 0})

# Create location clusters
kmeans = KMeans(n_clusters=5, random_state=42)
df['location_cluster'] = kmeans.fit_predict(df[['latitude', 'longitude']])
```

In [6]:
```
# Encode Categorical Variables

# One-Hot Encoding for low-cardinality columns
onehot_cols = ['room_type', 'cancellation_policy']
df = pd.get_dummies(df, columns=onehot_cols, drop_first=True)

# Frequency Encoding for high-cardinality columns
for col in ['neighbourhood', 'zipcode']:
```

```
    freq_map = df[col].value_counts().to_dict()
    df[col + '_freq'] = df[col].map(freq_map)
```

In [8]:
```
# Outlier Detection (Flag)

from scipy.stats import zscore
df['z_price'] = zscore(df['log_price'])
df['price_outlier'] = df['z_price'].apply(lambda x: 1 if abs(x) > 3 else 0)
```

In [9]:
```
# Final Feature Set

# Drop unneeded columns
drop_cols = ['id', 'name', 'description', 'thumbnail_url', 'first_review', 'last_re
             'host_since', 'zipcode', 'neighbourhood', 'amenities', 'z_price']
df_clean = df.drop(columns=drop_cols)

df_clean.head()
```

Out[9]:

	log_price	property_type	accommodates	bathrooms	bed_type	cleaning_fee	city	host
0	5.010635	Apartment	3	1.0	Real Bed	True	NYC	
1	5.129899	Apartment	7	1.0	Real Bed	True	NYC	
2	4.976734	Apartment	5	1.0	Real Bed	True	NYC	
3	6.620073	House	4	1.0	Real Bed	True	SF	
4	4.744932	Apartment	2	1.0	Real Bed	True	DC	

5 rows × 33 columns

2. Model Development

In [10]:
```
# Define target and features
X = df_clean.drop(columns=['log_price'])
y = df_clean['log_price']
```

In [11]:
```
# Train-Test Split

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)
```

In [14]:
```
# Identify object (categorical) columns
cat_cols = X.select_dtypes(include='object').columns.tolist()
print("Categorical columns:", cat_cols)
```

```
Categorical columns: ['property_type', 'bed_type', 'city']
```

In [15]:
```
# Encode Categorical Columns
#Frequency Encoding

for col in ['property_type', 'bed_type', 'city']:
    freq_map = X[col].value_counts().to_dict()
    X[col] = X[col].map(freq_map)
```

In [19]:
```
# Impute missing values

from sklearn.impute import SimpleImputer

# Impute numerical missing values with median
imputer = SimpleImputer(strategy='median')
X_imputed = imputer.fit_transform(X)

# Reassign back as DataFrame
X = pd.DataFrame(X_imputed, columns=X.columns)
```

In [20]:
```
# Feature Scaling

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [21]:
```
# Linear Regression Model

from sklearn.linear_model import LinearRegression

lr_model = LinearRegression()
lr_model.fit(X_train_scaled, y_train)
```

Out[21]:
```
▾ LinearRegression
LinearRegression()
```

In [24]:
```
# install XGBoost

!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\adithya\anaconda3\lib\site-packag
es (3.0.2)
Requirement already satisfied: numpy in c:\users\adithya\anaconda3\lib\site-packages
(from xgboost) (1.26.4)
Requirement already satisfied: scipy in c:\users\adithya\anaconda3\lib\site-packages
(from xgboost) (1.13.1)
```

In [25]:
```
# XGBoost Regression Model

from xgboost import XGBRegressor
```

```
xgb_model = XGBRegressor(n_estimators=100, learning_rate=0.1, max_depth=6, random_s
xgb_model.fit(X_train, y_train)
```

Out[25]:

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=N
one,
             enable_categorical=False, eval_metric=None, feature_types=N
one,
             feature_weights=None, gamma=None, grow_policy=None,
             importance_type=None, interaction_constraints=None,
             learning_rate=0.1, max_bin=None, max_cat_threshold=None,
```

In [26]:

```
# Save Trained Models

import joblib
joblib.dump(lr_model, "linear_model.pkl")
joblib.dump(xgb_model, "xgboost_model.pkl")
```

Out[26]: ['xgboost_model.pkl']

3. Model Evaluation

In [27]:

```
# Model Evaluation by XGBoost

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Predict using XGBoost model
y_pred_xgb = xgb_model.predict(X_test_scaled)

# Evaluation Metrics
rmse = np.sqrt(mean_squared_error(y_test, y_pred_xgb))
mae = mean_absolute_error(y_test, y_pred_xgb)
r2 = r2_score(y_test, y_pred_xgb)

print(f"XGBoost RMSE: {rmse:.2f}")
print(f"XGBoost MAE: {mae:.2f}")
print(f"XGBoost R²: {r2:.2f}")
```

```
XGBoost RMSE: 0.80
XGBoost MAE: 0.64
XGBoost R²: -0.23
```

In [28]:

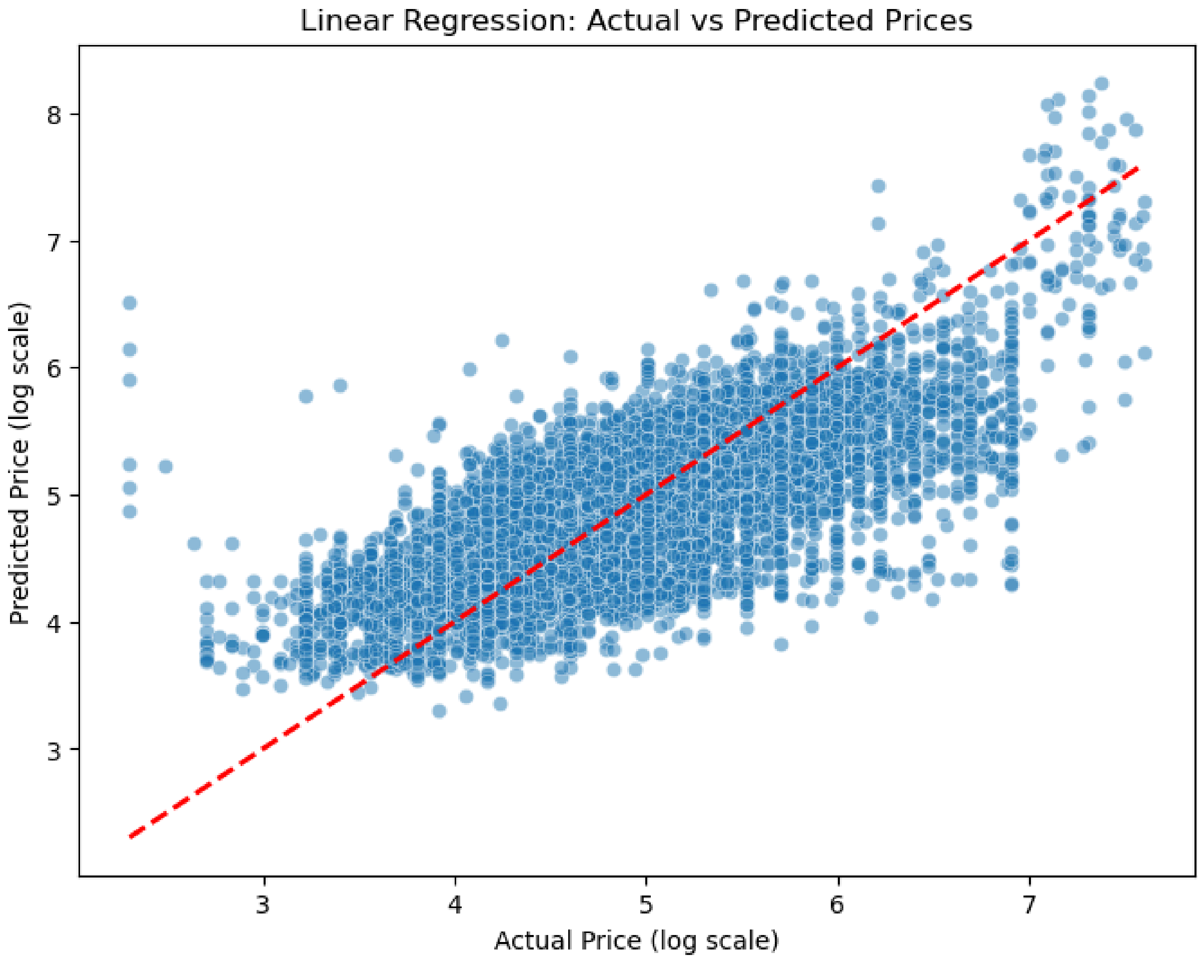
```
# Model Evaluation by Linear Regression

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np
```

```
# Predict on test data
y_pred_lr = lr_model.predict(X_test_scaled)

# Calculate evaluation metrics
rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
mae_lr = mean_absolute_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

print(f"Linear Regression RMSE: {rmse_lr:.2f}")
print(f"Linear Regression MAE: {mae_lr:.2f}")
print(f"Linear Regression R²: {r2_lr:.2f}")
```

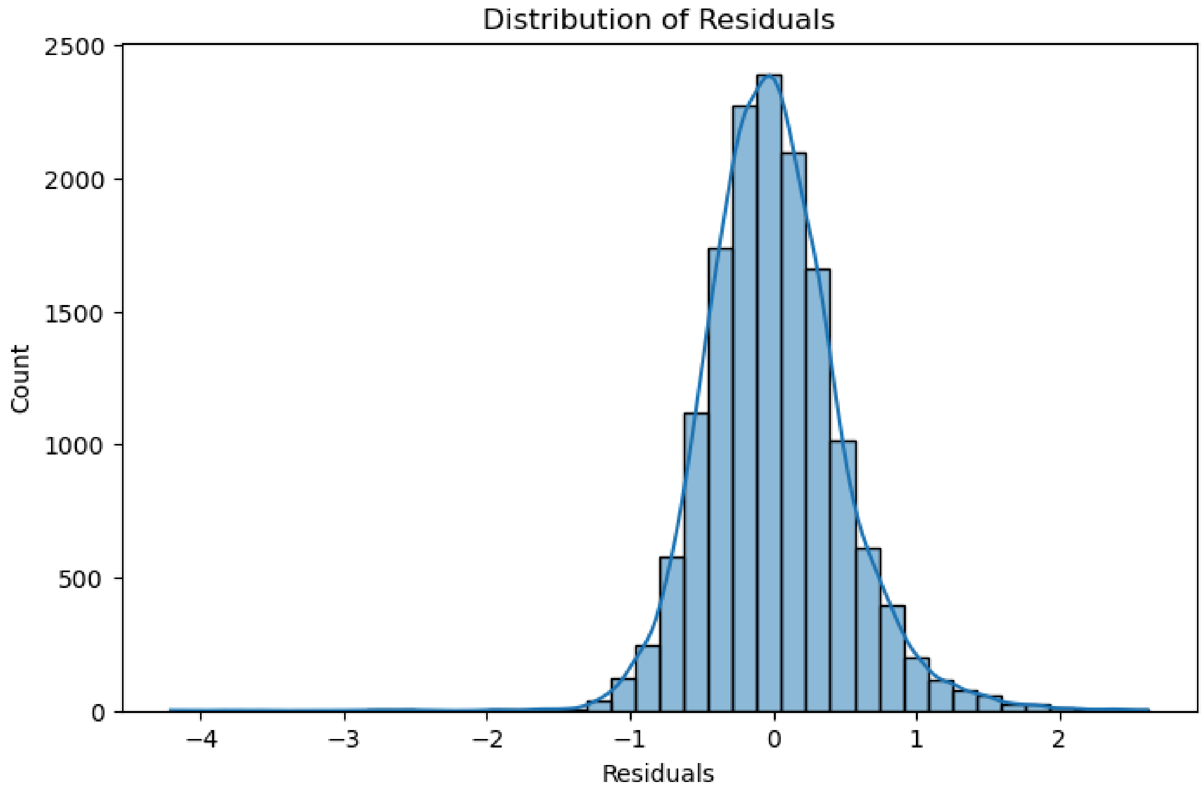
```
Linear Regression RMSE: 0.47
Linear Regression MAE: 0.36
Linear Regression R²: 0.57
```

In [29]:

```
# Actual vs Predicted Prices

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred_lr, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.xlabel('Actual Price (log scale)')
plt.ylabel('Predicted Price (log scale)')
plt.title('Linear Regression: Actual vs Predicted Prices')
plt.show()
```



In [30]:
```
# Residuals Plot

residuals = y_test - y_pred_lr

plt.figure(figsize=(8, 5))
sns.histplot(residuals, bins=40, kde=True)
plt.xlabel('Residuals')
plt.title('Distribution of Residuals')
plt.show()
```



Presentation Video: https://drive.google.com/file/d/1Hz-Yh-zPBnzbhXDhbGSUiGFBaTU0nkdv/view

In [ ]: