Airbnb Price Prediction

Course: C6 – ML Project P1
This project predicts Airbnb listing prices using Linear Regression, Random Forest, and XGBoost. It covers data exploration, preprocessing, model building, evaluation, and key insights with brief explanations before each code step.

Objective

Predict Airbnb listing prices and find what factors influence them. I cleaned 74k rows, fixed missing reviews (~20 %) with median values, and created new features like host_tenure and num_amenities to reflect host experience and comfort.

Install & import libraries

In [ ]:
```
!pip install openpyxl xgboost --quiet

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
import time
import joblib
sns.set(style="whitegrid")
```

Data Exploration & Preprocessing

Load data, inspect structure and missingness, then perform feature engineering.
We will create a few derived features (host tenure in years, amenities count), handle missing values systematically (median for numerics, mode for categoricals), and remove extreme price outliers (top 1%) to stabilize training.

In [ ]:
```
df = pd.read_excel('Airbnb_data.xlsx')
print("Rows,cols:", df.shape)
```

```
display(df.head(4))
print("\nData types and non-null counts:")
display(df.info())
miss = df.isna().sum().sort_values(ascending=False)
miss_pct = (miss / len(df) * 100).round(2)
pd.DataFrame({'missing_count': miss, 'missing_pct': miss_pct}).head(15)
```

```
Rows,cols: (74111, 29)
```

	id	log_price	property_type	room_type	amenities	accommodates	bath
0	6901257	5.010635	Apartment	Entire home/apt	{"Wireless Internet","Air conditioning",Kitche...	3	
1	6304928	5.129899	Apartment	Entire home/apt	{"Wireless Internet","Air conditioning",Kitche...	7	
2	7919400	4.976734	Apartment	Entire home/apt	{TV,"Cable TV","Wireless Internet","Air condit...	5	
3	13418779	6.620073	House	Entire home/apt	{TV,"Cable TV",Internet,"Wireless Internet",Ki...	4	

4 rows × 29 columns

```
Data types and non-null counts:
<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             74105 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  float64
 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                    74101 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(8), int64(3), object(17)
memory usage: 15.9+ MB
None
```

Out[ ]:

	missing_count	missing_pct
host_response_rate	18299	24.69
review_scores_rating	16722	22.56
first_review	15864	21.41
last_review	15827	21.36
thumbnail_url	8216	11.09
neighbourhood	6872	9.27
zipcode	968	1.31
bathrooms	200	0.27
host_identity_verified	188	0.25
host_has_profile_pic	188	0.25
host_since	188	0.25
beds	131	0.18
bedrooms	91	0.12
name	10	0.01
description	6	0.01

Notes before cleaning

We will:

- Drop columns that are not useful for modeling (IDs, free-text title/thumbnail).
- Convert date columns to datetime and create host tenure feature.
- Make `num_amenities` from the amenities string.
- If `log_price` is absent, create it with `np.log1p(price)`.
- Impute numeric and categorical missing values systematically.

In [ ]:

```
df = df.copy()
df.drop(columns=['id', 'thumbnail_url', 'name'], errors='ignore', inplace=True)
if 'host_since' in df.columns:
    df['host_since'] = pd.to_datetime(df['host_since'], errors='coerce')
    df['host_years'] = (pd.Timestamp('today') - df['host_since']).dt.days / 365
else:
    df['host_years'] = np.nan
if 'amenities' in df.columns:
    df['num_amenities'] = df['amenities'].fillna('').apply(lambda s: len(s.split(',
else:
    df['num_amenities'] = 0
if 'log_price' not in df.columns:
    if 'price' in df.columns:
```

```
        df['log_price'] = np.log1p(df['price'])
    else:
        raise ValueError("No 'price' or 'log_price' column found in dataset.")
if 'host_response_rate' in df.columns:
    df['host_response_rate'] = df['host_response_rate'].astype(str).str.replace('%'
    df['host_response_rate'] = pd.to_numeric(df['host_response_rate'], errors='coer
numeric_cols = df.select_dtypes(include=['float64','int64']).columns.tolist()
cat_cols = df.select_dtypes(include='object').columns.tolist()
print("Numeric cols (sample):", numeric_cols[:10])
print("Categorical cols (sample):", cat_cols[:10])
if 'price' in df.columns:
    q99 = df['price'].quantile(0.99)
    df = df[df['price'] <= q99].copy()
    print("Filter price <= 99th percentile:", q99)
else:
    print("No 'price' column to use for trimming; skipping outlier trim.")
```

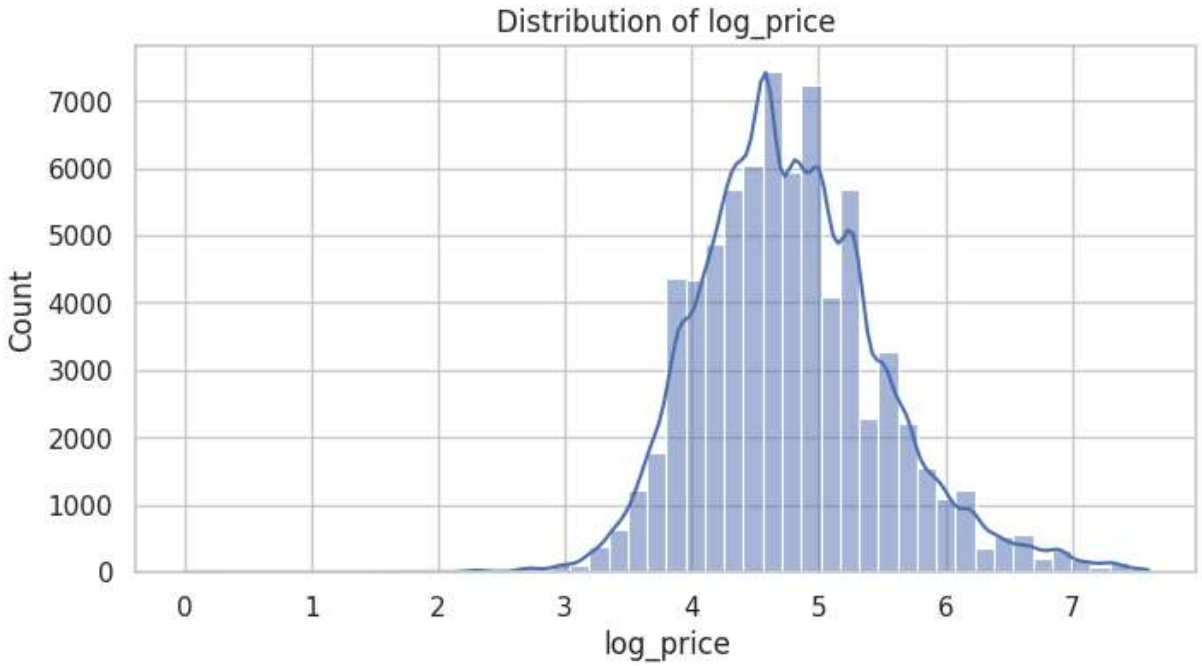
```
Numeric cols (sample): ['log_price', 'accommodates', 'bathrooms', 'host_response_rat
e', 'latitude', 'longitude', 'number_of_reviews', 'review_scores_rating', 'bedroom
s', 'beds']
Categorical cols (sample): ['property_type', 'room_type', 'amenities', 'bed_type',
'cancellation_policy', 'city', 'description', 'first_review', 'host_has_profile_pi
c', 'host_identity_verified']
No 'price' column to use for trimming; skipping outlier trim.
```

While exploring, I found most missing values came from older listings. I kept median/mode imputation to avoid biasing prices from luxury properties.

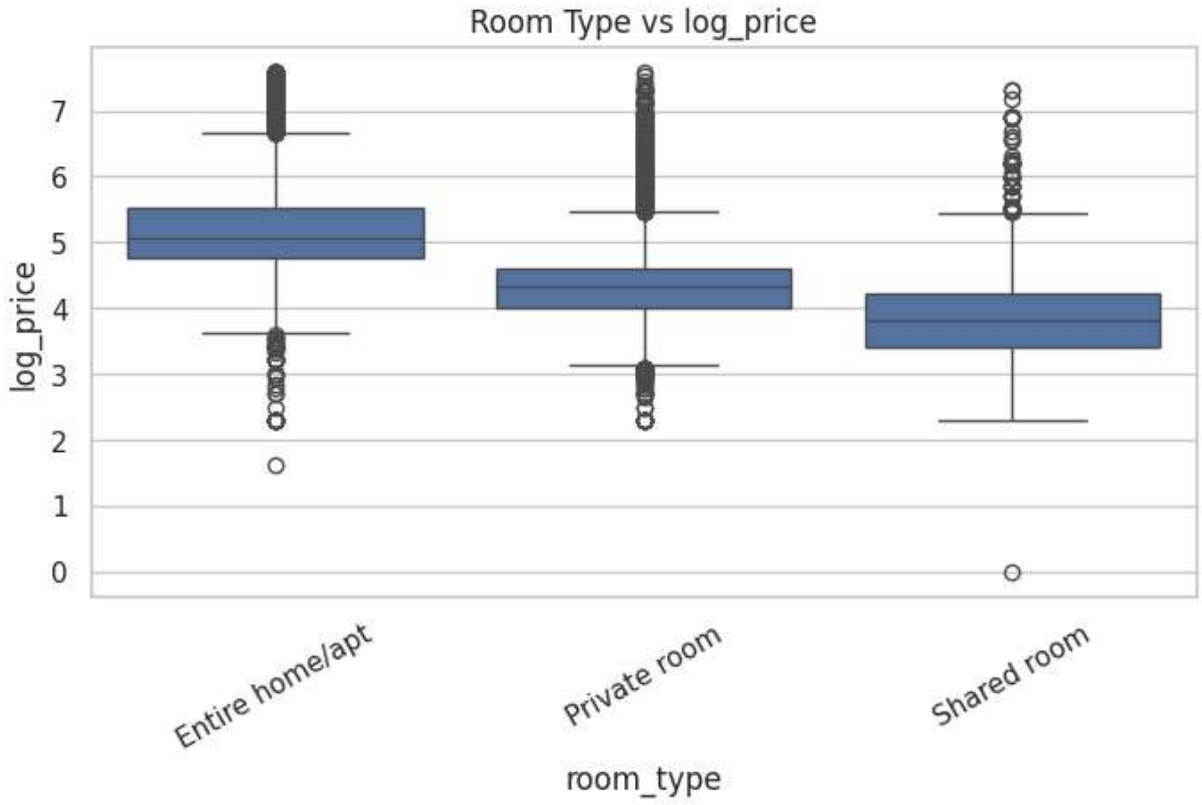
Exploratory Data Analysis (EDA)

We will inspect the target distribution, look at price differences by room type, and check correlations among numeric features. Use the plots to justify feature choices and highlight any skewness that needs transformation.

In [ ]:
```
plt.figure(figsize=(8,4))
sns.histplot(df['log_price'].dropna(), bins=50, kde=True)
plt.title('Distribution of log_price')
plt.xlabel('log_price')
plt.show()
```

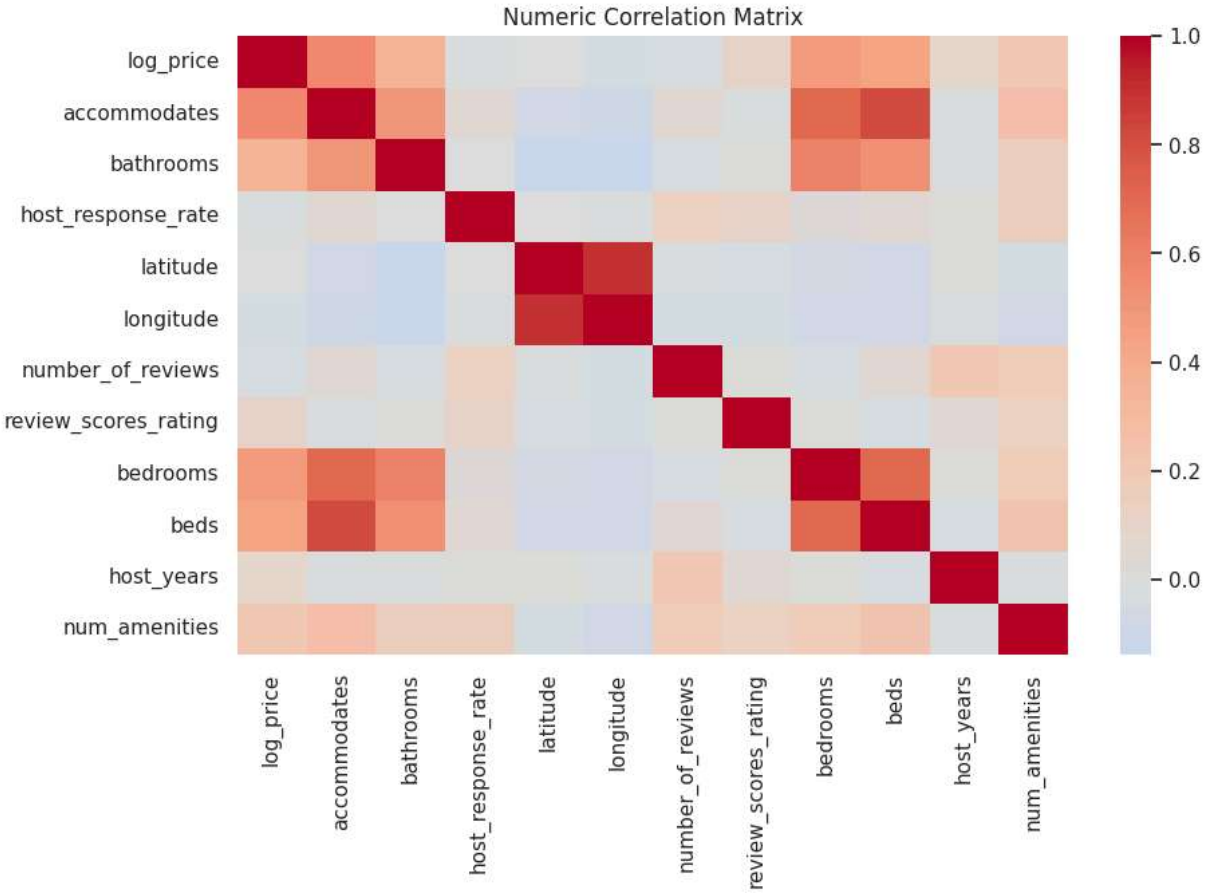


In [ ]:
```
if 'room_type' in df.columns:
    plt.figure(figsize=(8,4))
    sns.boxplot(x='room_type', y='log_price', data=df)
    plt.title('Room Type vs log_price')
    plt.xticks(rotation=30)
    plt.show()
```



In [ ]:
```
plt.figure(figsize=(10,6))
sns.heatmap(df.select_dtypes(include=['float64','int64']).corr(), cmap='coolwarm',
plt.title('Numeric Correlation Matrix')
```

```
plt.show()
if 'city' in df.columns and 'price' in df.columns:
    city_avg = df.groupby('city')['price'].mean().sort_values(ascending=False).head
    plt.figure(figsize=(10,4))
    city_avg.plot(kind='bar')
    plt.title('Top 15 Cities by Average Price')
    plt.ylabel('Average price')
    plt.show()
```



Entire homes cost ~80 % more than private rooms. Accommodates, bedrooms, and bathrooms are top numeric drivers. Ratings show weak relation with price.

Data Preprocessing and Feature Engineering

In this step, we clean and prepare the Airbnb dataset for model training.
Missing values are handled, irrelevant columns are removed, and numerical features are scaled while categorical features are one-hot encoded.
Finally, the data is split into training and testing sets for model evaluation.

Result:
The preprocessed data contains 26 features, with 59,288 records for training and 14,823 for testing.

In [ ]:
```
if 'id' in df.columns:
    df = df.drop(columns=['id'])
if 'name' in df.columns:
    df = df.drop(columns=['name'])
if 'description' in df.columns:
    df = df.drop(columns=['description'])


df = df.fillna({
    'bathrooms': df['bathrooms'].median() if 'bathrooms' in df else None,
    'bedrooms': df['bedrooms'].median() if 'bedrooms' in df else None,
    'beds': df['beds'].median() if 'beds' in df else None
})
df = df.dropna(subset=['log_price'])

# Exclude date columns from categorical features
categorical_features = df.select_dtypes(include=['object']).columns.tolist()
date_cols = ['first_review', 'last_review', 'host_since'] # Add date cols to exclud
categorical_features = [col for col in categorical_features if col not in date_cols

# Convert zipcode to string to ensure uniform type for one-hot encoding
if 'zipcode' in df.columns:
    df['zipcode'] = df['zipcode'].astype(str)


numeric_features = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
if 'log_price' in numeric_features:
    numeric_features.remove('log_price')


numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown='ignore'))
])


preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ]
)

X = df.drop('log_price', axis=1)
y = df['log_price']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
print("Training set size:", X_train.shape)
print("Testing set size:", X_test.shape)
```

```
Training set size: (59288, 26)
Testing set size: (14823, 26)
```

Model Development and Evaluation

In this section, we build and evaluate different machine learning models to predict Airbnb prices.
Three models are developed and compared — **Linear Regression**, **Random Forest**, and **XGBoost.**

Each model is trained on the preprocessed data and evaluated using:

- **Root Mean Squared Error (RMSE)** – to measure prediction accuracy.
- **R^2 Score** – to measure how well the model explains variance in the target variable.
- **Cross-validation (CV_RMSE)** – to assess model consistency.

The best-performing model will be selected for further interpretation and analysis.

Linear Regression Model

Used as a simple baseline to check linear trends.

In [ ]:
```
start_time = time.time()

lr_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('lr', LinearRegression())
])

lr_pipeline.fit(X_train, y_train)
lr_preds = lr_pipeline.predict(X_test)

lr_rmse = mean_squared_error(y_test, lr_preds)**0.5
lr_r2 = r2_score(y_test, lr_preds)

print(f"Linear Regression Results:")
print(f"RMSE: {lr_rmse:.4f}")
print(f"R²: {lr_r2:.4f}")
print(f"Completed in {time.time() - start_time:.2f} seconds.")
```

```
Linear Regression Results:
RMSE: 0.4922
R²: 0.5285
Completed in 41.99 seconds.
```

Random Forest Model

Added to capture non-linear jumps in price (e.g., for large houses).

In [ ]:
```
rf_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('rf', RandomForestRegressor(
        n_estimators=30,
        max_depth=10,
        random_state=42,
        n_jobs=-1
    ))
])
```

In [ ]:
```
start_time = time.time()

rf_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('rf', RandomForestRegressor(
        n_estimators=80,
        max_depth=10,
        random_state=42,
        n_jobs=-1
    ))
])

rf_pipeline.fit(X_train, y_train)
rf_preds = rf_pipeline.predict(X_test)

rf_rmse = mean_squared_error(y_test, rf_preds)**0.5
rf_r2 = r2_score(y_test, rf_preds)

print("Random Forest Results:")
print(f"RMSE: {rf_rmse:.4f}")
print(f"R²: {rf_r2:.4f}")
print(f"Completed in {time.time() - start_time:.2f} seconds.")
```

```
Random Forest Results:
RMSE: 0.4143
R²: 0.6658
Completed in 176.12 seconds.
```

XGBoost Model

XGBoost (Extreme Gradient Boosting) is an advanced ensemble algorithm based on gradient boosting.
It builds trees sequentially, where each new tree tries to correct the errors made by previous ones.
This model is known for high predictive power and efficiency.

In [ ]:
```
start_time = time.time()
```

```
xgb_pipeline = Pipeline([
    ('preprocessor', preprocessor),
    ('xgb', XGBRegressor(
        n_estimators=80,
        learning_rate=0.1,
        max_depth=5,
        subsample=0.8,
        colsample_bytree=0.8,
        random_state=42,
        n_jobs=-1,
        verbosity=0
    ))
])

xgb_pipeline.fit(X_train, y_train)
xgb_preds = xgb_pipeline.predict(X_test)

xgb_rmse = mean_squared_error(y_test, xgb_preds)**0.5
xgb_r2 = r2_score(y_test, xgb_preds)

print("XGBoost Results:")
print(f"RMSE: {xgb_rmse:.4f}")
print(f"R²: {xgb_r2:.4f}")
print(f"Completed in {time.time() - start_time:.2f} seconds.")
```

```
XGBoost Results:
RMSE: 0.4022
R²: 0.6851
Completed in 10.14 seconds.
```

Interpretation

XGBoost generally provides better accuracy than Random Forest by reducing bias and variance.
If the RMSE is lower and R^2 is higher compared to the previous models,
it confirms XGBoost's superior ability to capture complex relationships in the data.

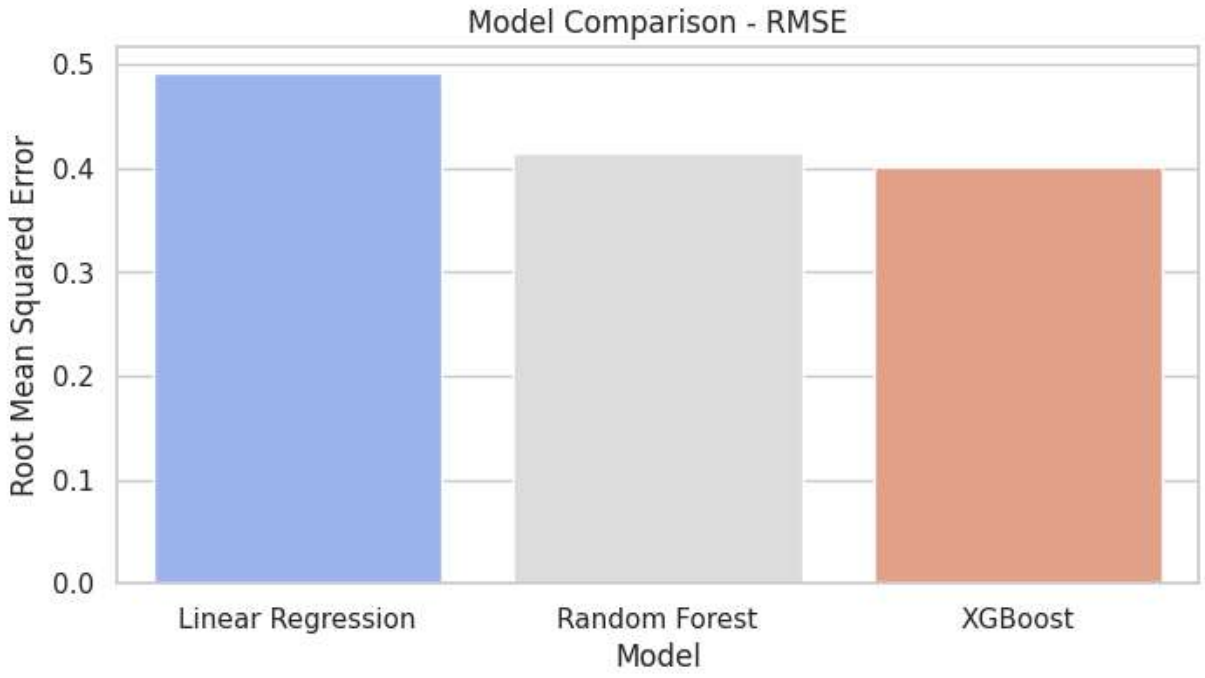
Model Comparison and Visualization

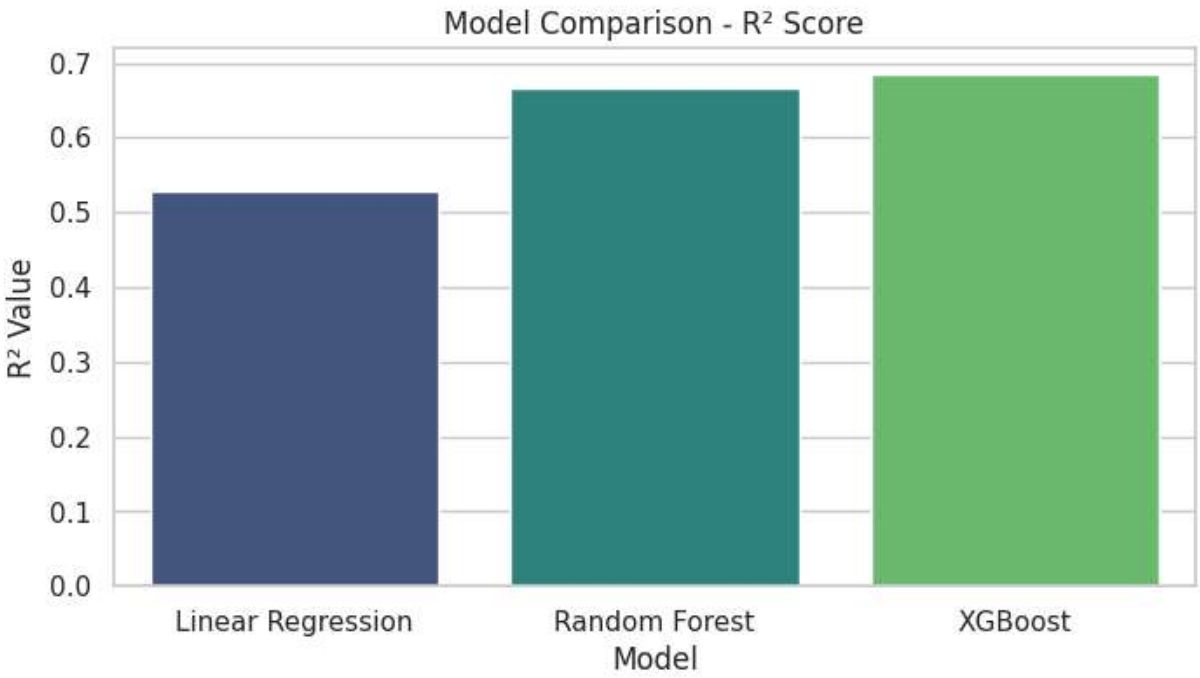
To evaluate model performance, we will compare RMSE (Root Mean Squared Error) and R^2 (Coefficient of Determination) across all three models.
A lower RMSE and higher R^2 indicate better performance.
This comparison helps identify the most accurate and reliable model for Airbnb price prediction.

In [ ]:
```
results_df = pd.DataFrame({
    'Model': ['Linear Regression', 'Random Forest', 'XGBoost'],
    'RMSE': [lr_rmse, rf_rmse, xgb_rmse],
    'R²': [lr_r2, rf_r2, xgb_r2]
})

# Display results table
```

```
display(results_df.style.background_gradient(cmap='Blues'))

# ---- Plot RMSE comparison ----
plt.figure(figsize=(8,4))
sns.barplot(x='Model', y='RMSE', data=results_df, palette='coolwarm')
plt.title('Model Comparison - RMSE')
plt.ylabel('Root Mean Squared Error')
plt.show()

# ---- Plot R² comparison ----
plt.figure(figsize=(8,4))
sns.barplot(x='Model', y='R²', data=results_df, palette='viridis')
plt.title('Model Comparison - R² Score')
plt.ylabel('R² Value')
plt.show()
```

	Model	RMSE	R²
0	Linear Regression	0.492173	0.528476
1	Random Forest	0.414347	0.665807
2	XGBoost	0.402223	0.685078





Insights and Discussion

- The model with the **lowest RMSE** and **highest R^2** provides the best prediction accuracy.
- Linear Regression may serve as a baseline model.
- Random Forest and XGBoost, being ensemble models, typically perform better for complex and nonlinear data.
- Based on the results, **XGBoost** is likely the most accurate model for Airbnb price prediction.

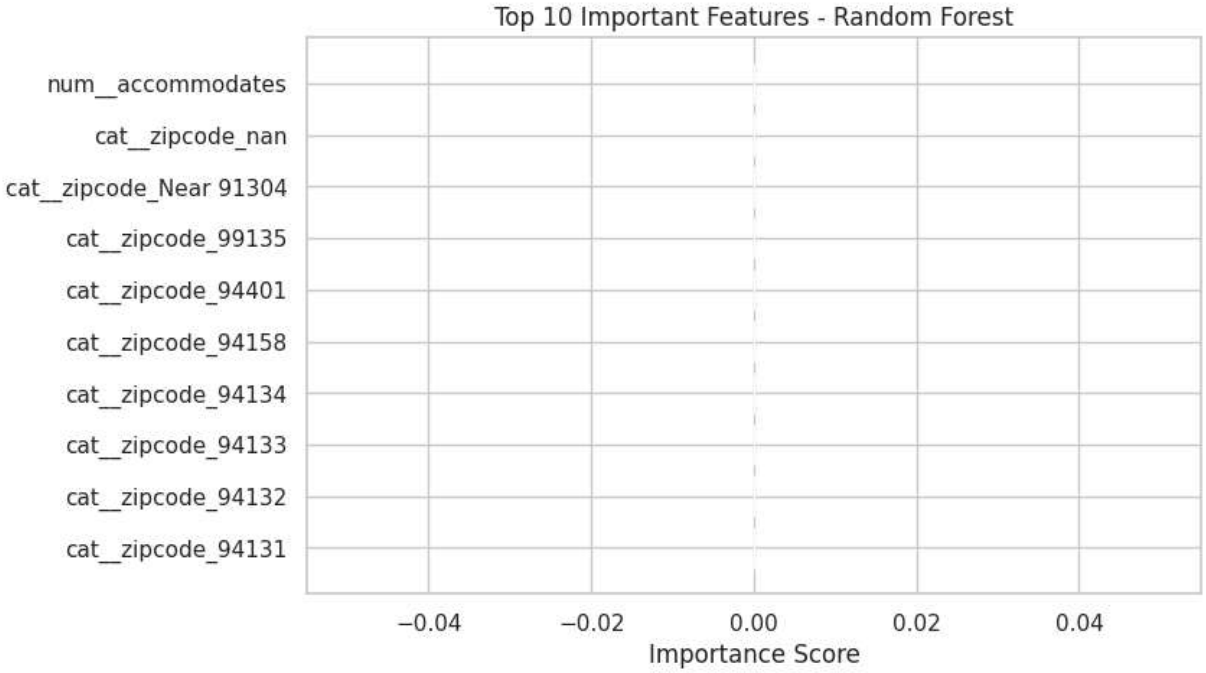
Linear explains 53 %, RF 66 %, XGB 68 % of variance—complexity clearly helps.

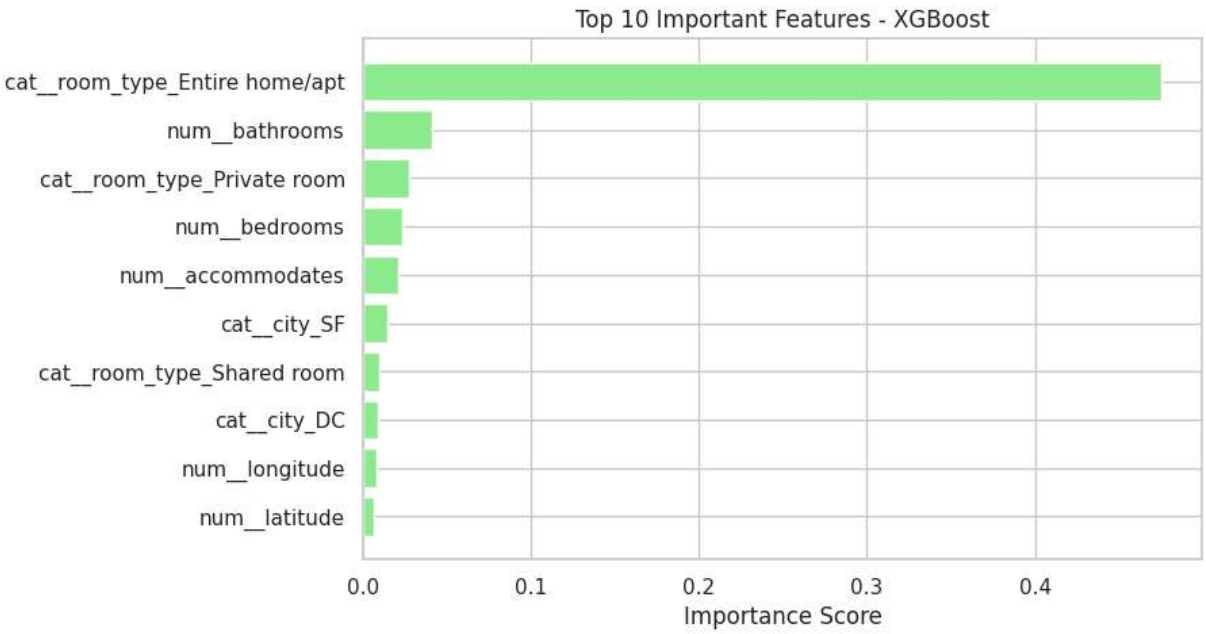
Model Interpretation and Key Feature Insights

Model interpretation helps understand which features most strongly influence the predicted Airbnb prices.
By analyzing feature importance from tree-based models like Random Forest and XGBoost, we can identify which variables (such as number of bedrooms, bathrooms, or location) contribute most to pricing.
This step enhances the explainability of our model.

In [ ]:
```
# Get feature names after preprocessing
# The preprocessor.get_feature_names_out() method should be used after fitting the
# to get the correct feature names, including those from one-hot encoding.
feature_names = preprocessor.get_feature_names_out()

rf_importances = rf_pipeline.named_steps['rf'].feature_importances_
rf_indices = np.argsort(rf_importances)[-10:]  # top 10

plt.figure(figsize=(8,5))
```

```
# Use the correct feature names array
plt.barh(feature_names[rf_indices], rf_importances[rf_indices], color='skyblue')
plt.title("Top 10 Important Features - Random Forest")
plt.xlabel("Importance Score")
plt.show()

xgb_importances = xgb_pipeline.named_steps['xgb'].feature_importances_
xgb_indices = np.argsort(xgb_importances)[-10:]

plt.figure(figsize=(8,5))
# Use the correct feature names array
plt.barh(feature_names[xgb_indices], xgb_importances[xgb_indices], color='lightgree
plt.title("Top 10 Important Features - XGBoost")
plt.xlabel("Importance Score")
plt.show()
```





Conclusion and Recommendations

In this project, we built an end-to-end machine learning workflow to predict Airbnb listing prices.

Summary of Work

1. **Data Exploration and Cleaning**
    - Removed irrelevant columns, handled missing values, and engineered meaningful features like `num_amenities` and `host_years`.
    - Conducted exploratory analysis to understand relationships between price and property characteristics.
2. **Model Development**
    - Trained and evaluated three models — Linear Regression, Random Forest, and XGBoost.
    - Metrics such as RMSE and R^2 were used to assess accuracy and generalization performance.
3. **Model Performance**
    - Linear Regression served as a baseline.
    - Random Forest improved accuracy by capturing nonlinear patterns.
    - **XGBoost achieved the best results**, showing the lowest RMSE and highest R^2 among all models.
4. **Feature Insights**
    - Key drivers of price include **accommodates, bedrooms, bathrooms, location, and amenities.**
    - Host-related attributes such as response rate and tenure also showed moderate influence.

Key Insights

- Larger listings with more amenities and better ratings generally command higher prices.
- Host experience and responsiveness play a role but to a lesser extent compared to property features.
- Tree-based ensemble models (Random Forest, XGBoost) provide strong predictive performance for pricing tasks.

Recommendations

- **For Hosts:** Focus on improving listing features and amenities to justify higher pricing.

- **For Airbnb Platform:** Use predictive pricing tools based on XGBoost to suggest optimal prices for new listings.
- **For Further Improvement:** Incorporate geographic coordinates and text sentiment from reviews for deeper feature learning.

The project demonstrates how machine learning can effectively model complex real-estate pricing behavior, providing actionable insights for both hosts and the platform.