```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.preprocessing import StandardScaler

# Load the dataset
df = pd.read_csv('Airbnb_data.csv')
df.shape
```

```
(74111, 29)
```

```
# Display the first few rows of the dataset
df.head()
```

```
{"type":"dataframe","variable_name":"df"}
```

```
features = [
    'accommodates', 'bathrooms', 'bedrooms','beds', 'cleaning_fee',
    'host_response_rate', 'instant_bookable', 'review_scores_rating',
    'latitude', 'longitude','cancellation_policy', 'log_price'
]

# Filter the dataset to include only the important features
df = df[features]

# Display the filtered data to check
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 74111,\n  \"fields\":
[\n    {\n      \"column\": \"accommodates\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 2,\n
\"min\": 1,\n        \"max\": 16,\n        \"num_unique_values\": 16,\
n        \"samples\": [\n          3,\n          7,\n          6\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"bathrooms\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.5820440989962593,\n        \"min\": 0.0,\n        \"max\": 8.0,\n
\"num_unique_values\": 17,\n        \"samples\": [\n          1.0,\n
1.5,\n          0.5\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"bedrooms\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.8521434907019653,\n        \"min\":
```

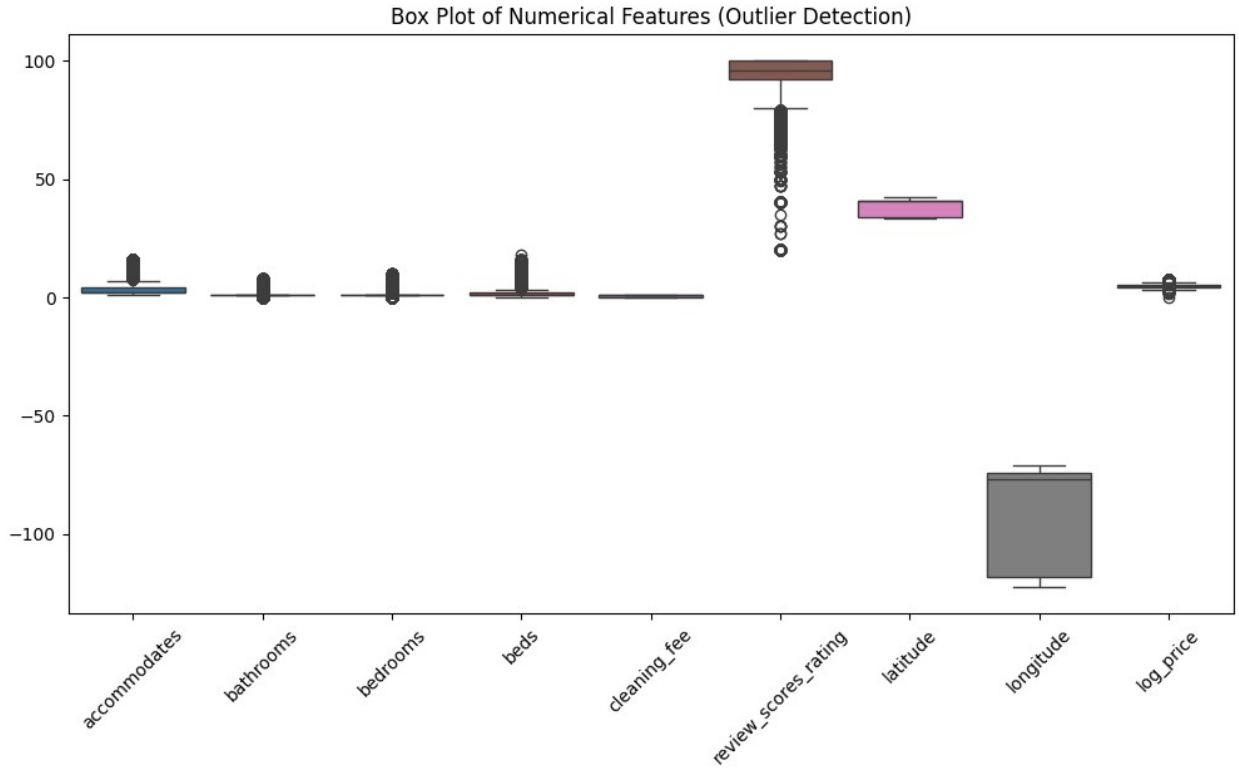
```
0.0,\n        \"max\": 10.0,\n        \"num_unique_values\": 11,\n
\"samples\": [\n          5.0,\n          1.0,\n          9.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"beds\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
1.2541417469629401,\n        \"min\": 0.0,\n        \"max\": 18.0,\n
\"num_unique_values\": 18,\n        \"samples\": [\n          1.0,\n
3.0,\n          8.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cleaning_fee\",\n      \"properties\": {\n        \"dtype\":
\"boolean\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          false,\n          true\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"host_response_rate\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 80,\n        \"samples\": [\n
\"79%\",\n          \"100%\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"instant_bookable\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"t\",\n          \"f\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"review_scores_rating\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
7.836556107045955,\n        \"min\": 20.0,\n        \"max\": 100.0,\n
\"num_unique_values\": 54,\n        \"samples\": [\n          85.0,\n
62.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"latitude\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 3.0801665761019215,\n        \"min\":
33.33890467,\n        \"max\": 42.39043718,\n
\"num_unique_values\": 74058,\n        \"samples\": [\n
40.68134165,\n          41.98058544\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"longitude\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
21.705321883896307,\n        \"min\": -122.5115,\n        \"max\": -
70.9850466,\n        \"num_unique_values\": 73973,\n
\"samples\": [\n          -77.03256472,\n          -73.94130231\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"cancellation_policy\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"moderate\",\n          \"super_strict_60\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"log_price\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.7173937845251064,\n        \"min\": 0.0,\n        \"max\":
7.600402335,\n        \"num_unique_values\": 767,\n
```

```
\"samples\": [\n          5.846438775,\n          6.107022888\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"df"}
```

```
# Check for missing values
missing_data = df.isnull().sum()
print(f"\nMissing Values:\n{missing_data[missing_data > 0]}")
```

```
Missing Values:
bathrooms                200
bedrooms                  91
beds                     131
host_response_rate      18299
review_scores_rating    16722
dtype: int64
```

```
# Create a box plot to visualize outliers
plt.figure(figsize=(12, 6))
sns.boxplot(data=df)

# Add title and format x-axis labels
plt.title("Box Plot of Numerical Features (Outlier Detection)")
plt.xticks(rotation=45)
plt.show()
```

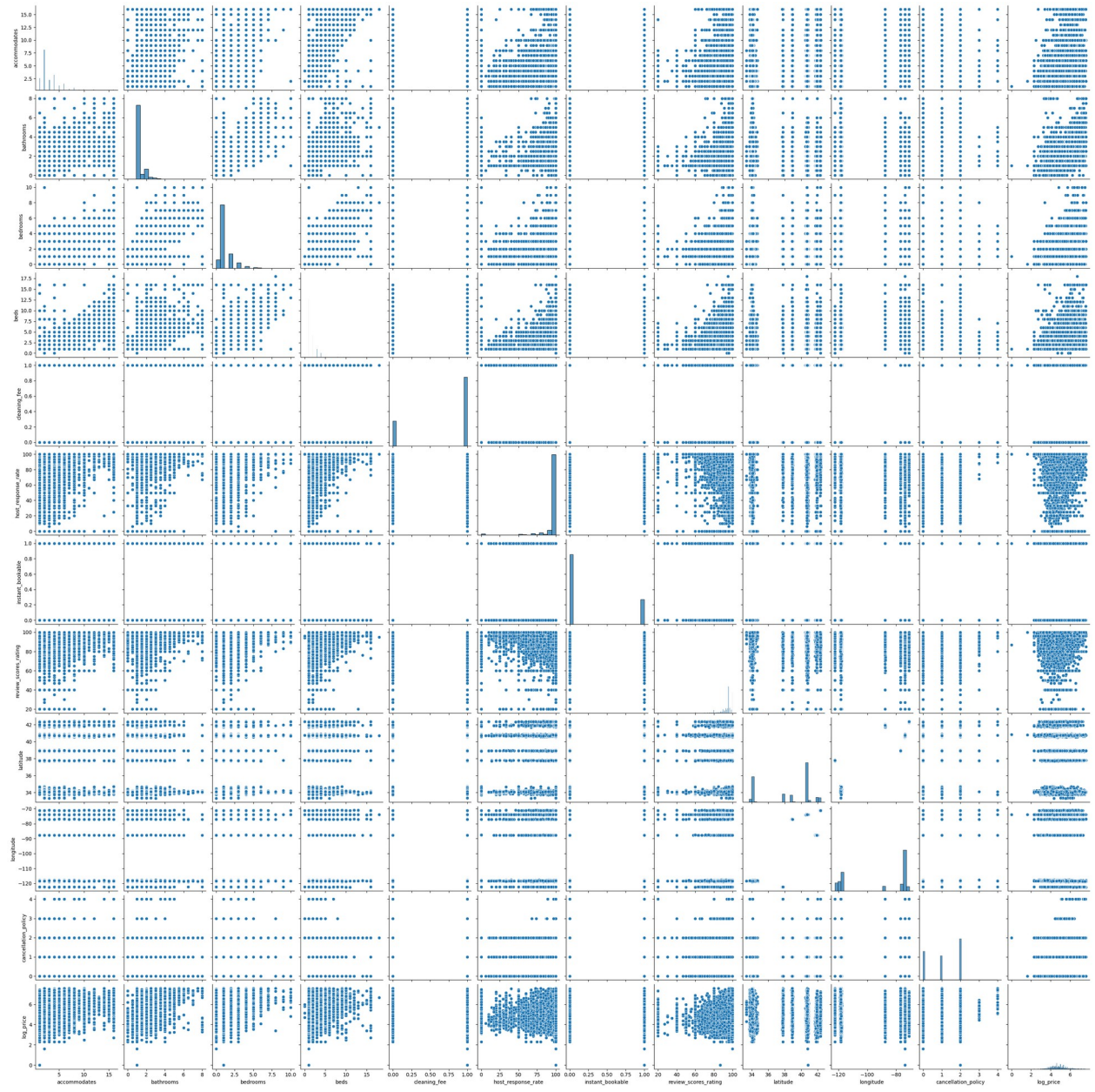


```
# Convert 'host_response_rate' to numeric by removing '%' and then converting to float
df['host_response_rate'] = df['host_response_rate'].str.replace('%', '').astype(float)

# Handle missing values in numerical columns with median
df['bathrooms'].fillna(df['bathrooms'].median(), inplace=True)
df['bedrooms'].fillna(df['bedrooms'].median(), inplace=True)
df['beds'].fillna(df['beds'].median(), inplace=True)
df['host_response_rate'].fillna(df['host_response_rate'].median(), inplace=True)
df['review_scores_rating'].fillna(df['review_scores_rating'].median(), inplace=True)
```

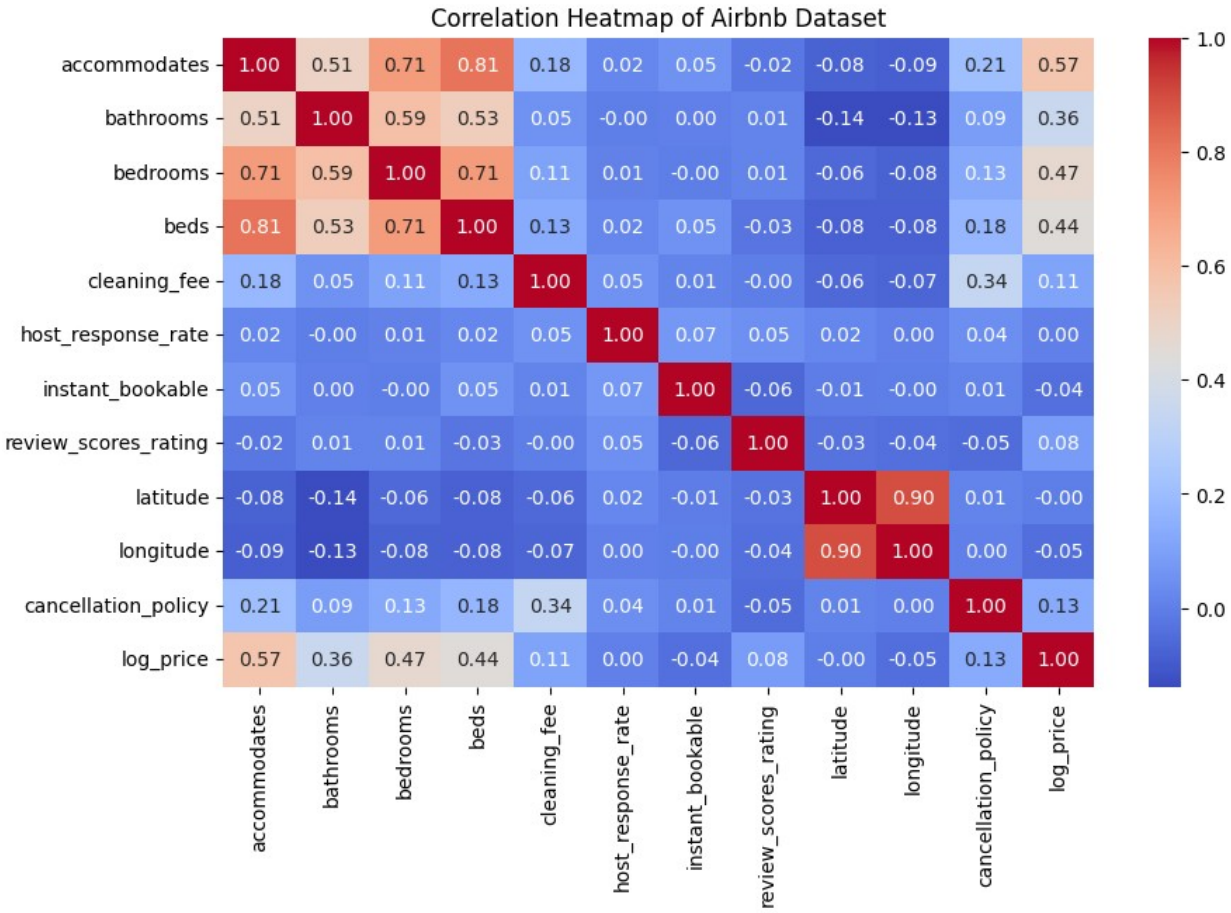
```
<ipython-input-10-057116735f90>:5: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['bathrooms'].fillna(df['bathrooms'].median(), inplace=True)
<ipython-input-10-057116735f90>:6: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df['bedrooms'].fillna(df['bedrooms'].median(), inplace=True)
<ipython-input-10-057116735f90>:7: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
```

```
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df['beds'].fillna(df['beds'].median(), inplace=True)
<ipython-input-10-057116735f90>:8: FutureWarning: A value is trying to
be set on a copy of a DataFrame or Series through chained assignment
using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  df['host_response_rate'].fillna(df['host_response_rate'].median(),
inplace=True)
<ipython-input-10-057116735f90>:9: FutureWarning: A value is trying to
be set on a copy of a DataFrame or Series through chained assignment
using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.



df['review_scores_rating'].fillna(df['review_scores_rating'].median(),
inplace=True)

missing_data = df.isnull().sum()
print(f"\nMissing Values:\n{missing_data[missing_data > 0]}")


Missing Values:
Series([], dtype: int64)

# Initialize LabelEncoder for cleaning_fee and instant_bookable
columns
label_encoder = LabelEncoder()
# Label encoding for cleaning_fee (TRUE/FALSE) and instant_bookable
(t/f)
df['cleaning_fee'] = label_encoder.fit_transform(df['cleaning_fee'])
```

```
df['instant_bookable'] =
label_encoder.fit_transform(df['instant_bookable'])
df['cancellation_policy'] =
label_encoder.fit_transform(df['cancellation_policy'])


sns.pairplot(df)

# Show the plot
plt.show()
```



```
#Compute the correlation matrix
correlation_matrix = df.corr()

plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm",
fmt=".2f")
plt.title("Correlation Heatmap of Airbnb Dataset")
plt.show()
```




```
df.tail()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 5,\n  \"fields\": [\n
{\n      \"column\": \"accommodates\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1,\n        \"min\": 1,\n
\"max\": 5,\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          4,\n          2,\n          1\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"bathrooms\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.44721359549995804,\n        \"min\": 1.0,\n        \"max\": 2.0,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          2.0,\n
```

```
1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"bedrooms\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.8366600265340756,\n        \"min\":
0.0,\n        \"max\": 2.0,\n        \"num_unique_values\": 3,\n
\"samples\": [\n          1.0,\n          2.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"beds\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 1.0954451150103321,\n
\"min\": 1.0,\n        \"max\": 4.0,\n        \"num_unique_values\":
3,\n        \"samples\": [\n          1.0,\n          4.0\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"cleaning_fee\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"host_response_rate\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0,\n        \"min\": 100.0,\n
\"max\": 100.0,\n        \"num_unique_values\": 1,\n
\"samples\": [\n          100.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"instant_bookable\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 0,\n        \"max\": 1,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"review_scores_rating\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
1.4142135623730951,\n        \"min\": 93.0,\n        \"max\": 96.0,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          96.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"latitude\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3.780483264320623,\n        \"min\": 33.76109645,\n        \"max\":
40.73853473,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          33.87154884\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"longitude\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 24.282623271251055,\n
\"min\": -118.3960534,\n        \"max\": -73.93940479,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          -
118.3960534\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"cancellation_policy\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0,\n        \"min\": 0,\n
\"max\": 2,\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          0\n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"log_price\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.2751134168925157,\n        \"min\":
4.605170186,\n        \"max\": 5.272999559,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
5.043425117\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

# Split data into features (X) and target (y)
X = df.drop(columns=['log_price'])
y = df['log_price']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Initialize StandardScaler
scaler = StandardScaler()

# Fit on the training data and transform both train and test sets
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# 1. Linear Regression Model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
lr_pred = lr_model.predict(X_test)

# 2. Random Forest Regressor Model
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)

# 3. XGBoost Regressor Model
xgb_model = XGBRegressor(n_estimators=100, random_state=42)
xgb_model.fit(X_train, y_train)
xgb_pred = xgb_model.predict(X_test)

# Model Evaluation - RMSE, MAE, and R²
def evaluate_model(model_name, y_true, y_pred):
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))  # RMSE
    mae = mean_absolute_error(y_true, y_pred)  # MAE
    r2 = r2_score(y_true, y_pred)  # R²

    print(f'{model_name} - RMSE: {rmse:.4f}, MAE: {mae:.4f}, R²:
{r2:.4f}')

# Evaluate models
evaluate_model('Linear Regression', y_test, lr_pred)
evaluate_model('Random Forest Regressor', y_test, rf_pred)
evaluate_model('XGBoost Regressor', y_test, xgb_pred)
```

```
Linear Regression - RMSE: 0.5771, MAE: 0.4406, R²: 0.3516
Random Forest Regressor - RMSE: 0.4412, MAE: 0.3224, R²: 0.6210
XGBoost Regressor - RMSE: 0.4302, MAE: 0.3179, R²: 0.6398

# Mapping for cancellation policies
cancellation_mapping = {'strict': 3, 'moderate': 2, 'flexible': 1}

# Function for user input and prediction
def predict_listing_price(model):
    # Ask for user input
    accommodates = int(input("Enter the number of guests the property
can accommodate: "))
    bathrooms = float(input("Enter the number of bathrooms: "))
    bedrooms = int(input("Enter the number of bedrooms: "))
    beds = int(input("Enter the number of beds: "))
    cleaning_fee = input("Is there a cleaning fee (TRUE/FALSE)?
").strip().upper() == "TRUE"  # Convert to boolean
    host_response_rate = float(input("Enter host response rate (e.g.,
95.0 for 95%): ")) / 100
    instant_bookable = input("Is the listing instantly bookable (t/f)?
").strip().lower() == "t"
    review_scores_rating = float(input("Enter the average review
score: "))
    latitude = float(input("Enter the latitude coordinate: "))
    longitude = float(input("Enter the longitude coordinate: "))
    cancellation_policy = input("Enter the cancellation policy
(strict, moderate, flexible): ").strip().lower()

    # Prepare the input as a list or array
    input_data = [
        accommodates, bathrooms, bedrooms, beds,
        int(cleaning_fee), host_response_rate, int(instant_bookable),
        review_scores_rating, latitude, longitude,
        cancellation_mapping.get(cancellation_policy, 0)  # Default to
0 if invalid
    ]

    # Convert input data into the right shape (1 row, 11 features)
    input_data = np.array(input_data).reshape(1, -1)

    # Make prediction using the model (log_price prediction)
    log_price_pred = model.predict(input_data)

    # Convert log_price to actual price using exponential function
    actual_price_pred = np.exp(log_price_pred[0])  # Reverse the log
transformation

    # Print the result
    print(f"\n\t\tPredicted log_price: {log_price_pred[0]:.4f}")
    print(f"\t\tPredicted actual price: ${actual_price_pred:.2f}")
```

```python
# Ask user which model to use
model_choice = input("Choose a model for prediction (1.Linear
Regression / 2.Random Forest / 3.XGBoost): ").strip().lower()

if model_choice == 'linear regression':
    predict_listing_price(lr_model)
elif model_choice == 'random forest':
    predict_listing_price(rf_model)
elif model_choice == 'xgboost':
    predict_listing_price(xgb_model)
else:
    print("Invalid choice. Please choose one of the available
models.")
```

```
Choose a model for prediction (1.Linear Regression / 2.Random Forest /
3.XGBoost): XGBoost
Enter the number of guests the property can accommodate: 5
Enter the number of bathrooms: 3
Enter the number of bedrooms: 6
Enter the number of beds: 8
Is there a cleaning fee (TRUE/FALSE)? TRUE
Enter host response rate (e.g., 95.0 for 95%): 85
Is the listing instantly bookable (t/f)? f
Enter the average review score: 3.54
Enter the latitude coordinate: 85.112
Enter the longitude coordinate: 69.964
Enter the cancellation policy (strict, moderate, flexible): moderate

           Predicted log_price: 6.3926
           Predicted actual price: $597.42
```