Project- Part A: Airbnb Price Prediction and Insights

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.model_selection import RandomizedSearchCV
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')

# Load the dataset into a pandas DataFrame
df = pd.read_csv('airbnb_data.csv')

# Display the first few rows to understand the structure
print("First few rows of the dataset:")
print(df.head())

# Check data types and missing values
print("\nDataset information:")
print(df.info())

# Summary statistics for numerical columns
print("\nSummary statistics for numerical columns:")
print(df.describe())
```

```
First few rows of the dataset:
         id  log_price property_type        room_type  \
0   6901257   5.010635     Apartment  Entire home/apt
1   6304928   5.129899     Apartment  Entire home/apt
2   7919400   4.976734     Apartment  Entire home/apt
3  13418779   6.620073         House  Entire home/apt
4   3808709   4.744932     Apartment  Entire home/apt

                                           amenities  accommodates  bathrooms
\
0  {"Wireless Internet","Air conditioning",Kitche...             3        1.0
1  {"Wireless Internet","Air conditioning",Kitche...             7        1.0
2  {TV,"Cable TV","Wireless Internet","Air condit...             5        1.0
3  {TV,"Cable TV",Internet,"Wireless Internet",Ki...             4        1.0
```

```
4  {TV,Internet,"Wireless Internet","Air conditio...            2       1.0

   bed_type cancellation_policy  cleaning_fee  ...   latitude   longitude  \
0  Real Bed              strict          True  ...  40.696524  -73.991617
1  Real Bed              strict          True  ...  40.766115  -73.989040
2  Real Bed            moderate          True  ...  40.808110  -73.943756
3  Real Bed            flexible          True  ...  37.772004 -122.431619
4  Real Bed            moderate          True  ...  38.925627  -77.034596

                                    name     neighbourhood  \
0            Beautiful brownstone 1-bedroom  Brooklyn Heights
1  Superb 3BR Apt Located Near Times Square    Hell's Kitchen
2                          The Garden Oasis            Harlem
3       Beautiful Flat in the Heart of SF!      Lower Haight
4               Great studio in midtown DC  Columbia Heights

  number_of_reviews review_scores_rating  \
0                 2                100.0
1                 6                 93.0
2                10                 92.0
3                 0                  NaN
4                 4                 40.0

                                    thumbnail_url zipcode bedrooms  beds
0  https://a0.muscache.com/im/pictures/6d7cbbf7-c...   11201      1.0   1.0
1  https://a0.muscache.com/im/pictures/348a55fe-4...   10019      3.0   3.0
2  https://a0.muscache.com/im/pictures/6fae5362-9...   10027      1.0   3.0
3  https://a0.muscache.com/im/pictures/72208dad-9...   94117      2.0   2.0
4                                               NaN   20009      0.0   1.0

[5 rows x 29 columns]

Dataset information:
<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
None

Summary statistics for numerical columns:
                 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
```

```python
# Check for missing values in each column
print("\n>>> Initially Missing values in each column:")
print(df.isnull().sum())
```

```
>>> Initially Missing values in each column:
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
```

```python
# Handle missing values
# For numerical columns, impute with median
df['bathrooms'] = df['bathrooms'].fillna(df['bedrooms'].median())
df['bedrooms'] = df['bedrooms'].fillna(df['bedrooms'].median())
df['beds'] = df['beds'].fillna(df['beds'].median())
df['review_scores_rating'] =
df['review_scores_rating'].fillna(df['review_scores_rating'].median())

# For text columns, fill with empty strings
```

```
df['host_response_rate'] = df['host_response_rate'].fillna('na')
df['neighbourhood'] = df['neighbourhood'].fillna('na')
df['thumbnail_url'] = df['thumbnail_url'].fillna('na')
df['zipcode'] = df['zipcode'].fillna('na')

# For date columns, use forward fill to handle missing values
df['first_review'] = pd.to_datetime(df['first_review'], format='%d-%m-%Y')  #
First review date (Date)
df['first_review'] = df['first_review'].fillna(method='ffill')
df['host_since'] = pd.to_datetime(df['host_since'], format='%d-%m-%Y')  #
Host join date (Date)
df['host_since'] = df['host_since'].fillna(method='ffill')
df['last_review'] = pd.to_datetime(df['last_review'], format='%d-%m-%Y')  #
Last review date (Date)
df['last_review'] = df['last_review'].fillna(method='ffill')

# Convert categorical boolean columns ('t'/'f' -> 1/0)
bool_cols = ['instant_bookable', 'host_has_profile_pic',
'host_identity_verified']
for col in bool_cols:
    df[col] = df[col].map({'t': 1, 'f': 0})

# For boolean columns, using forward fill to handle missing values
df['host_has_profile_pic'] =
df['host_has_profile_pic'].fillna(method='ffill')
df['host_identity_verified'] =
df['host_identity_verified'].fillna(method='ffill')
df['instant_bookable'] = df['instant_bookable'].fillna(method='ffill')

# Print missing values count after handling them
print("After handling missing values in each column:")
missing_values = df.isnull().sum()
if missing_values.sum() == 0:
    print("(0) No Missing Values")
else:
    print(missing_values)
```

```
After handling missing values in each column:
(0) No Missing Values
```

```
# Verify Data Types and Units
# Ensure data types align with the data dictionary
# Convert columns to their correct data types

df['id'] = df['id'].astype(str)  # Unique identifier (String)
df['log_price'] = df['log_price'].astype(float)  # Log-transformed price
(Float)
df['property_type'] = df['property_type'].astype(str)  # Property type
(String)
df['room_type'] = df['room_type'].astype(str)  # Room type (String)
```

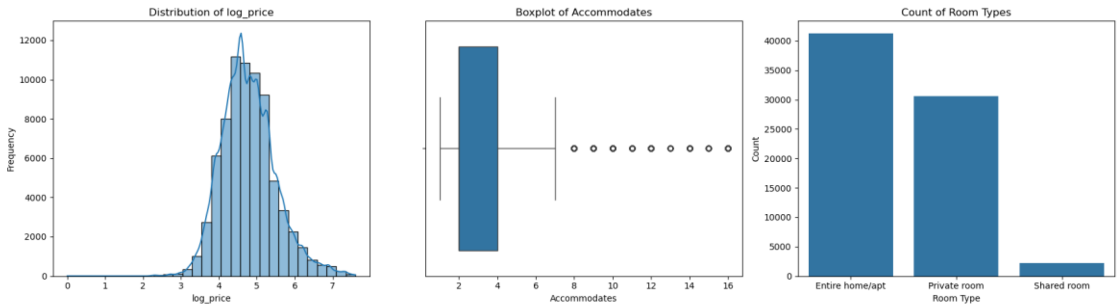
```python
df['amenities'] = df['amenities'].astype(str)  # List of amenities (String)
df['accommodates'] = df['accommodates'].astype(int)  # Number of guests (Integer)
df['bathrooms'] = df['bathrooms'].astype(float)  # Number of bathrooms (Float, as it can be fractional)
df['bed_type'] = df['bed_type'].astype(str)  # Type of bed (String)
df['cancellation_policy'] = df['cancellation_policy'].astype(str)  # Cancellation policy (String)
df['cleaning_fee'] = df['cleaning_fee'].astype(bool)  # Cleaning fee (Boolean)
df['city'] = df['city'].astype(str)  # City (String)
df['description'] = df['description'].astype(str)  # Description (String)

df['first_review'] = pd.to_datetime(df['first_review'], format='%d-%m-%Y', errors='coerce')  # First review date (Date)
df['host_has_profile_pic'] = df['host_has_profile_pic'].astype(bool)  # Host profile picture (Boolean)
df['host_identity_verified'] = df['host_identity_verified'].astype(bool)  # Host identity verified (Boolean)
df['host_response_rate'] = df['host_response_rate'].astype(str)  # Host response rate (String)
df['host_since'] = pd.to_datetime(df['host_since'], format='%d-%m-%Y', errors='coerce')  # Host join date (Date)
df['instant_bookable'] = df['instant_bookable'].astype(bool)  # Instant bookable (Boolean)
df['last_review'] = pd.to_datetime(df['last_review'], format='%d-%m-%Y', errors='coerce')  # Last review date (Date)

df['latitude'] = df['latitude'].astype(float)  # Latitude (Float)
df['longitude'] = df['longitude'].astype(float)  # Longitude (Float)

df['name'] = df['name'].astype(str)  # Listing name (String)
df['neighbourhood'] = df['neighbourhood'].astype(str)  # Neighborhood (String)
df['number_of_reviews'] = df['number_of_reviews'].astype(int)  # Number of reviews (Integer)
df['review_scores_rating'] = df['review_scores_rating'].astype(float)  # Review score (Float)
df['thumbnail_url'] = df['thumbnail_url'].astype(str)  # Thumbnail URL (String)
df['zipcode'] = df['zipcode'].astype(str)  # Zip code (String)
df['bedrooms'] = df['bedrooms'].astype(int)  # Number of bedrooms (Integer)
df['beds'] = df['beds'].astype(int)  # Number of beds (Integer)

# Print the final data types to verify correctness
print("Final Data Types:\n", df.dtypes)
```

```
Final Data Types:
 id                                  object
log_price                           float64
```

```
property_type                         object
room_type                             object
amenities                             object
accommodates                           int32
bathrooms                            float64
bed_type                              object
cancellation_policy                   object
cleaning_fee                            bool
city                                  object
description                           object
first_review                  datetime64[ns]
host_has_profile_pic                    bool
host_identity_verified                  bool
host_response_rate                    object
host_since                    datetime64[ns]
instant_bookable                        bool
last_review                   datetime64[ns]
latitude                             float64
longitude                            float64
name                                  object
neighbourhood                         object
number_of_reviews                      int32
review_scores_rating                 float64
thumbnail_url                         object
zipcode                               object
bedrooms                               int32
beds                                   int32
dtype: object
```

```
# analyze trends and outliers
# Create subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# Visualize the distribution of the target variable (log_price)
# Distribution of log_price with KDE plot
sns.histplot(df['log_price'].dropna(), kde=True, ax=axes[0], bins=30)
axes[0].set_title('Distribution of log_price')
axes[0].set_xlabel('log_price')
axes[0].set_ylabel('Frequency')

# Boxplot for numerical columns to detect outliers
# Boxplot of Accommodates
sns.boxplot(x=df['accommodates'].dropna(), ax=axes[1], orient='h')
axes[1].set_title('Boxplot of Accommodates')
axes[1].set_xlabel('Accommodates')

# Count plot for categorical columns (e.g., room_type)
# Count plot of Room Types
sns.countplot(x=df['room_type'].dropna(), ax=axes[2],
order=df['room_type'].value_counts().index)
```

```
axes[2].set_title('Count of Room Types')
axes[2].set_xlabel('Room Type')
axes[2].set_ylabel('Count')

# Improve layout
plt.tight_layout()
plt.show()
```




```
#feature engineering and data transformation

# Count the number of amenities in the 'amenities' column (Safe Parsing)
df['amenities_count'] = df['amenities'].apply(lambda x:
len(str(x).strip('{}').split(',')) if pd.notna(x) else 0)
print(">>> Number of amenities in the 'amenities' column\n",
df['amenities_count'])

# Convert 'host_since' to datetime and calculate host activity (years since
joining)
df['host_since'] = pd.to_datetime(df['host_since'], errors='coerce')
df['host_activity'] = (pd.Timestamp.now() - df['host_since']).dt.days / 365
df['host_activity'] = df['host_activity'].fillna(0)  # Handle NaN values
safely
print("\n>>> Number of years since the host joined\n", df['host_activity'])

# Compute Neighborhood Popularity (Count of listings per neighborhood)
df['neighbourhood'] = df['neighbourhood'].astype(str)  # Ensure it's a string
to avoid mapping issues
neighbourhood_popularity = df['neighbourhood'].value_counts().to_dict()
df['neighbourhood_popularity'] =
df['neighbourhood'].map(neighbourhood_popularity).fillna(0).astype(int)
print("\n>>> Number of listings in each neighborhood\n",
df['neighbourhood_popularity'])

# Calculate the length of the description
df['description_length'] = df['description'].apply(lambda x: len(str(x)) if
pd.notna(x) else 0)
print("\n>>> Description Length\n", df['description_length'])

# Convert date columns to the number of days since the given date
def convert_date(df, col):
```

```
    df[col] = pd.to_datetime(df[col], errors='coerce')
    df[col] = (pd.Timestamp.now() - df[col]).dt.days
    df[col] = df[col].fillna(-1).astype(int)  # Fill NaN with -1 to indicate
missing values
    return df[col]

date_cols = ['host_since', 'first_review', 'last_review']
for col in date_cols:
    df[col] = convert_date(df, col)
```

```
>>> Number of amenities in the 'amenities' column
 0         9
1        15
2        19
3        15
4        12
         ..
74106     1
74107    16
74108    31
74109    15
74110    18
Name: amenities_count, Length: 74111, dtype: int64

>>> Number of years since the host joined
 0        13.082192
1         7.846575
2         8.495890
3        10.016438
4        10.150685
           ...
74106    12.087671
74107     8.975342
74108    13.304110
74109     7.600000
74110    12.410959
Name: host_activity, Length: 74111, dtype: float64

>>> Number of listings in each neighborhood
 0         111
1        1299
2        1374
3         124
4         298
         ...
74106    2862
74107      80
74108    2862
74109     606
74110     573
```

```
Name: neighbourhood_popularity, Length: 74111, dtype: int32

>>> Description Length
 0         211
1         1000
2         1000
3          468
4          699
          ...
74106       24
74107      302
74108     1000
74109      555
74110     1000
Name: description_length, Length: 74111, dtype: int64

# Model Development

# Define Features and Target Variable
X = df.drop(columns=['log_price', 'id', 'name', 'description', 'amenities',
                     'host_response_rate', 'first_review', 'last_review',
                     'host_since', 'neighbourhood', 'thumbnail_url',
'zipcode'])
y = df['log_price']

# Split 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)

# Convert Boolean ('True'/'False') to Numeric (1/0)
bool_cols = ['instant_bookable', 'host_has_profile_pic',
'host_identity_verified']
for col in bool_cols:
    X_train[col] = X_train[col].astype(int)
    X_test[col] = X_test[col].astype(int)

# One-Hot Encode Categorical Variables
X_train = pd.get_dummies(X_train, drop_first=True)
X_test = pd.get_dummies(X_test, drop_first=True)

# Ensure X_train and X_test Have Same Columns
X_train, X_test = X_train.align(X_test, join='left', axis=1, fill_value=0)

# Standardize Numerical Features
scaler = StandardScaler()
numerical_features = X_train.select_dtypes(include=['int64',
'float64']).columns  # Select only numeric columns

X_train[numerical_features] =
```

```python
scaler.fit_transform(X_train[numerical_features])
X_test[numerical_features] = scaler.transform(X_test[numerical_features])

# Model Development
# Models Evaluation using metrics RMSE, MAE and R²

# Define models
models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(random_state=42),
    "Gradient Boosting": GradientBoostingRegressor(random_state=42),
}

# Function to evaluate models and find the best one
def train_and_evaluate(models, X_train, X_test, y_train, y_test):
    results = {}
    best_model = None
    best_score = float("-inf")

    for name, model in models.items():
        model.fit(X_train, y_train)  # Train the model
        y_pred = model.predict(X_test)  # Make predictions

        # Evaluate performance
        mae = mean_absolute_error(y_test, y_pred)
        mse = mean_squared_error(y_test, y_pred)
        rmse = np.sqrt(mse)  # Calculate RMSE manually
        r2 = r2_score(y_test, y_pred)

        results[name] = {"MAE": mae, "RMSE": rmse, "R² Score": r2}

        # Track the best model based on R² Score
        if r2 > best_score:
            best_score = r2
            best_model = model

    return results, best_model

# Train and evaluate all models
results, best_model = train_and_evaluate(models, X_train, X_test, y_train,
y_test)

# Convert results to a DataFrame for better readability
results_df = pd.DataFrame(results).T
print("Model Evaluation Metrics:\n", results_df)

# Use the best model for final predictions
y_pred_best = best_model.predict(X_test)
```

```python
# Display final evaluation for best model
mse_best = mean_squared_error(y_test, y_pred_best)
print("\nBest Model:", best_model.__class__.__name__)
print(f"Best Model RMSE: {np.sqrt(mse_best)}")
print(f"Best Model MAE: {mean_absolute_error(y_test, y_pred_best)}")
print(f"Best Model R² Score: {r2_score(y_test, y_pred_best)}")
```

```
Model Evaluation Metrics:
                        MAE      RMSE  R² Score
Linear Regression  0.347510  0.462752  0.583164
Decision Tree      0.392921  0.545817  0.420087
Random Forest      0.279446  0.389757  0.704296
Gradient Boosting  0.301487  0.411372  0.670589

Best Model: RandomForestRegressor
Best Model RMSE: 0.38975742040140987
Best Model MAE: 0.2794457364804364
Best Model R² Score: 0.7042959150374211
```

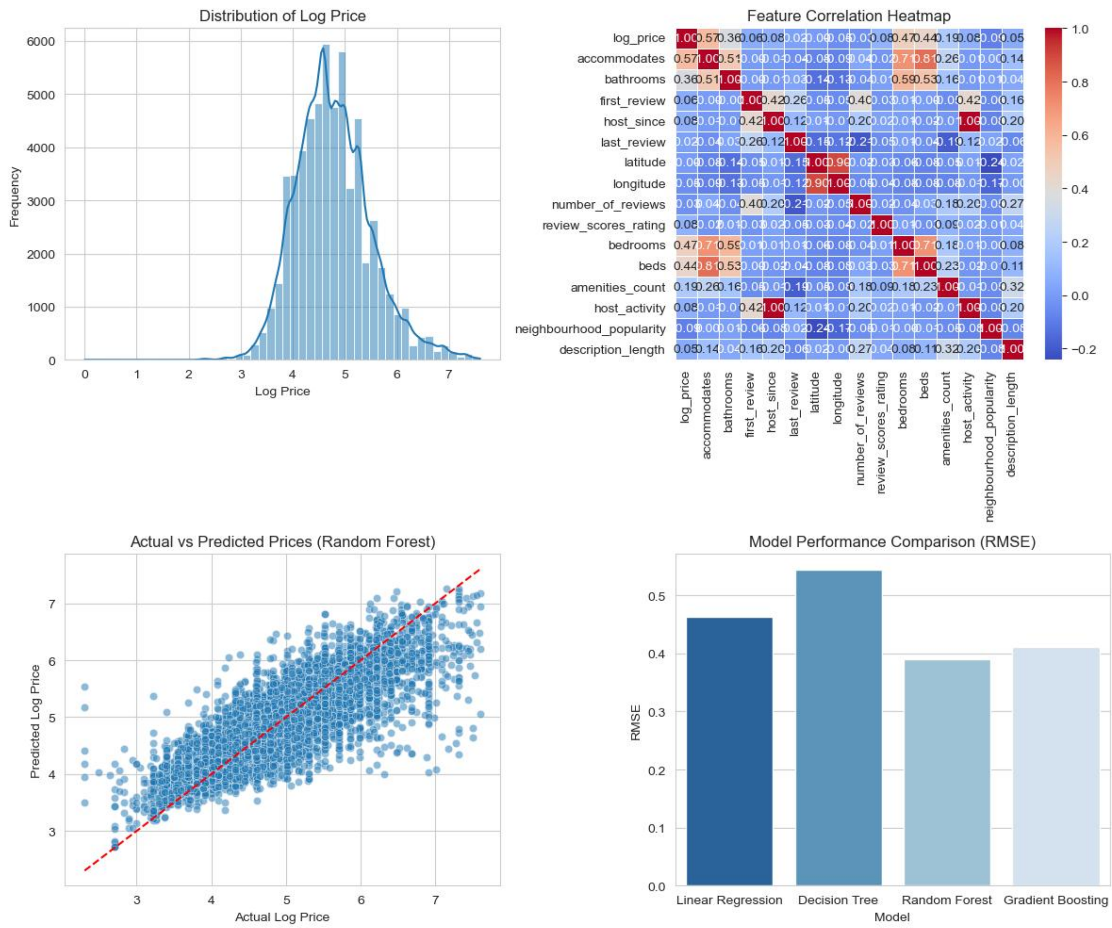
```python
# Model Tuning - Optimized
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Reduced parameter grid for faster tuning
param_dist = {
    'n_estimators': [70, 100],  # Reduced options
    'max_depth': [10, 15, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'max_features': ['sqrt', 0.8]  # Added for better generalization
}

# Initialize with warm_start=False (default) and reduced n_estimators
rf = RandomForestRegressor(random_state=42, n_estimators=100)

# Optimized RandomizedSearchCV
random_search = RandomizedSearchCV(
    rf,
    param_distributions=param_dist,
    n_iter=8,  # Reduced iterations
    cv=3,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    verbose=2,
    random_state=42
)
```

```
print("Starting hyperparameter tuning...")
random_search.fit(X_train, y_train)

# Get best parameters
best_params = random_search.best_params_
print("\nBest Hyperparameters:", best_params)

# Train final model with best params
best_rf = random_search.best_estimator_  # More efficient than re-fitting

# Evaluation
y_pred_tuned = best_rf.predict(X_test)
rmse_tuned = np.sqrt(mean_squared_error(y_test, y_pred_tuned))  # More stable
than squared=False
mae_tuned = mean_absolute_error(y_test, y_pred_tuned)
r2_tuned = r2_score(y_test, y_pred_tuned)

print("\nTuned Random Forest Performance:")
print(f" RMSE: {rmse_tuned:.4f}")
print(f" MAE: {mae_tuned:.4f}")
print(f" R²: {r2_tuned:.4f}")
```

```
Starting hyperparameter tuning...
Fitting 3 folds for each of 8 candidates, totalling 24 fits

Best Hyperparameters: {'n_estimators': 100, 'min_samples_split': 5,
'min_samples_leaf': 1, 'max_features': 0.8, 'max_depth': None}

Tuned Random Forest Performance:
 RMSE: 0.3870
 MAE: 0.2780
 R²: 0.7084
```

```
# Visualizations with Charts and Graphs

# Model Results DataFrame
results_df = pd.DataFrame({
    "Model": ["Linear Regression", "Decision Tree", "Random Forest",
"Gradient Boosting"],
    "RMSE": [0.4627, 0.5439, 0.3896, 0.4109],
    "MAE": [0.3475, 0.3917, 0.2794, 0.3013],
    "R² Score": [0.5831, 0.4241, 0.7044, 0.6713]
})

# Set style
sns.set_style("whitegrid")
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Distribution of Log Price
```

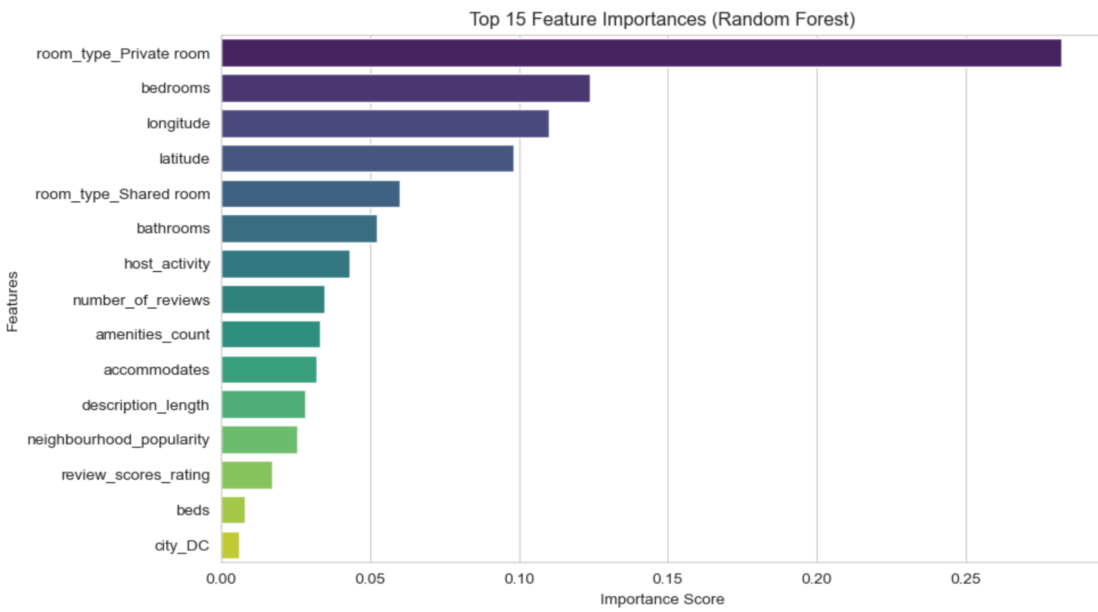
```python
sns.histplot(y_train, bins=50, kde=True, ax=axes[0, 0])
axes[0, 0].set(title="Distribution of Log Price", xlabel="Log Price",
ylabel="Frequency")

# Correlation Heatmap
numeric_df = df.select_dtypes(include=['number'])  # Select only numeric
columns for correlation
sns.heatmap(numeric_df.corr(), cmap="coolwarm", annot=True, fmt=".2f",
linewidths=0.5, ax=axes[0, 1])
axes[0, 1].set_title("Feature Correlation Heatmap")

# Actual vs Predicted Prices (Best Model)
sns.scatterplot(x=y_test, y=y_pred_tuned, alpha=0.5, ax=axes[1, 0])
axes[1, 0].plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color="red", linestyle="--")
axes[1, 0].set(title="Actual vs Predicted Prices (Random Forest)",
xlabel="Actual Log Price", ylabel="Predicted Log Price")

# Model Performance (Bar Chart)
sns.barplot(x="Model", y="RMSE", data=results_df, palette="Blues_r",
ax=axes[1, 1])
axes[1, 1].set_title("Model Performance Comparison (RMSE)")

plt.tight_layout()
plt.show()
```




```python
# Feature Importance

# Compute Feature Importance
feature_importance = pd.Series(best_model.feature_importances_,
index=X_train.columns).sort_values(ascending=False)

# Print Feature Importance
print("\nFeature Importance:")
print(feature_importance)

# Plot Feature Importance
plt.figure(figsize=(10, 6))
sns.barplot(x=feature_importance[:15], y=feature_importance.index[:15],
palette="viridis")
plt.title("Top 15 Feature Importances (Random Forest)")
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.show()
```

```
Feature Importance:
room_type_Private room           2.824651e-01
```

```
bedrooms                          1.240488e-01
longitude                         1.100723e-01
latitude                          9.820304e-02
room_type_Shared room             6.015859e-02
                                      ...
property_type_Earth House         2.489575e-06
property_type_Parking Space       1.269353e-06
property_type_Cave                1.025104e-06
property_type_Chalet              6.584714e-07
property_type_Casa particular     4.149111e-07
Length: 64, dtype: float64
```




```python
# Save the Preprocessed Data and Model
import joblib

# Ensure df contains only processed data
df_processed = df.copy()  # If necessary, create a clean version

# Save the preprocessed dataset
df_processed.to_csv('preprocessed_airbnb_data.csv', index=False)
print("Preprocessed dataset saved as 'preprocessed_airbnb_data.csv'")

# Save the trained model (Ensure 'best_rf' or final model exists)
joblib.dump(best_rf, 'airbnb_price_predictor.pkl')
print("Trained model saved as 'airbnb_price_predictor.pkl'")
```

```
Preprocessed dataset saved as 'preprocessed_airbnb_data.csv'
Trained model saved as 'airbnb_price_predictor.pkl'
```

Video Link

https://drive.google.com/file/d/1riPSYEi_EVLIJepXAK_5vdqg_rz7UZ4D/view?usp=drive_link