Airbnb Business Analysis Using a Data Science Approach

Introduction.

The aim of this report is to present to the executive team of Airbnb a comprehensive analysis of a dataset which contains valuable information about the listing activity in New York during the year 2019. The scope is to provide with accurate predictions about the Airbnb trends in order the executive team to improve profitability and maximize their business actions in that particular area.

Airbnb is a company founded in 2007 and offer short and long-term homestays. The company main focus is to offer unique experiences and stays which allows its guests to have a better and unique connection with the local communities (Airbnb, Inc., 2025).

The dataset used for the analysis is “AB_NYC_2019” a dataset downloaded from the platform Kaggle which is “a data platform that includes sections titled Competitions, Datasets, Code, Discussions, Learn, and, most recently, Models.” (Preda, 2023). The dataset contains 48,895 rows of data and sixteen columns.

For the analysis of the dataset and for the visualisation, Google Colab has been used. In Appendix A the phyton code for the Machine Learning Project can be found.

Business Context and Business Questions.

Based on the first check of the above-mentioned dataset, two primary business questions have been selected to be developed and analysed through a classic Machine Learning (ML in this report) methodology.
The questions are the following:

Question 1: What factors have the strongest influence on Airbnb listing prices in New York City?

Question 2: How do neighbourhood characteristics and listing attributes interact to influence Airbnb pricing patterns across different New York City boroughs, and what pricing strategies can hosts implement to optimize revenue based on these spatial dynamics?

Question 1. Visualisation and Insights.

To answer the first question, Regression Analysis has been used. Various models have been tested in order to find the best performer: Linear Regression, Random Forest, Ridge, Lasso cross-validation and, based on the overall results, the best performer has been the Random Forest.
Random Forest results in a R squared of 0.490 (49%) which, from a business perspective is a good results and offer the executive Airbnb team useful insights for business decisions. In Appendix A the complete coding of the Random Forest analysis can be consulted.

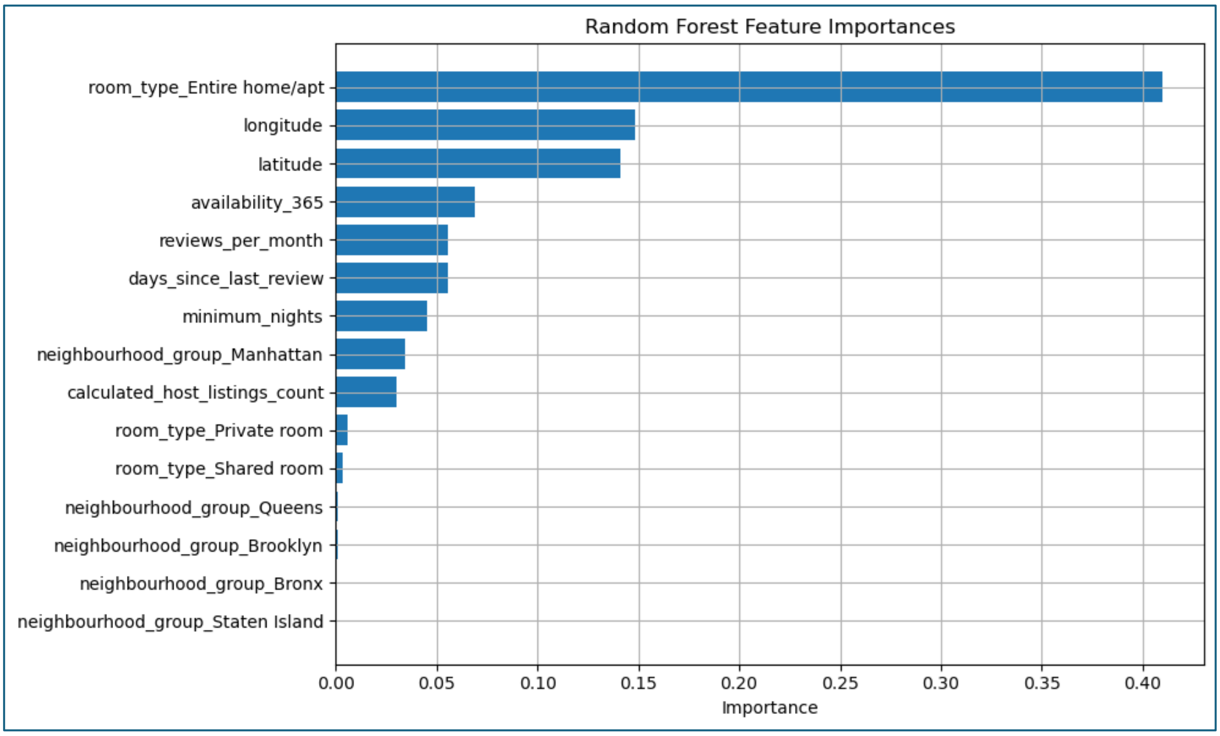


Fig. 1 – Random Forest Feature Importances (Appendix A).

The results visible in Figure 1 highlight the significant importance of the type of room (entire home or apartment) as a main factor influencing the prices in New York Airbnb's listing. Is evident that the entire home or apartment is the biggest value proposition and the host is willing to pay a premium price for.
Location is another crucial factor influencing the price a host is willing to pay. A strategic geographical position within the city is an important factor influencing the rental price.
Last factor influencing the Airbnb listing price is the availability patterns which highlight that a wise seasonal availability affects sensibly the price.
Is important to mention that reviews metrics, which could be consider as an important influencer, is not affecting the prices as the three above-mentioned factors.

Those results could be very important for the Airbnb executive team to plan a focused targeting of entire homes and apartments in strategic locations of the city with a wide availability to increase revenues.

Question 2. Visualisation and Insights.

For analysing the neighbourhood characteristics and listing attributes in order to see the interaction with the pricing pattern of Airbnb the first action has been identify clusters in the city of New York. To obtain a clear map of clusters it has been used a K-means clustering using mainly three scores: Elbow Method, Silhouette Score and Calinski-Harabasz Score (Appendix A).
Five main clusters have been defined and, in figure 2, a Principal Component

Analysis (PCA) has been used to provide a visual representation of the 5 main clusters discovered.

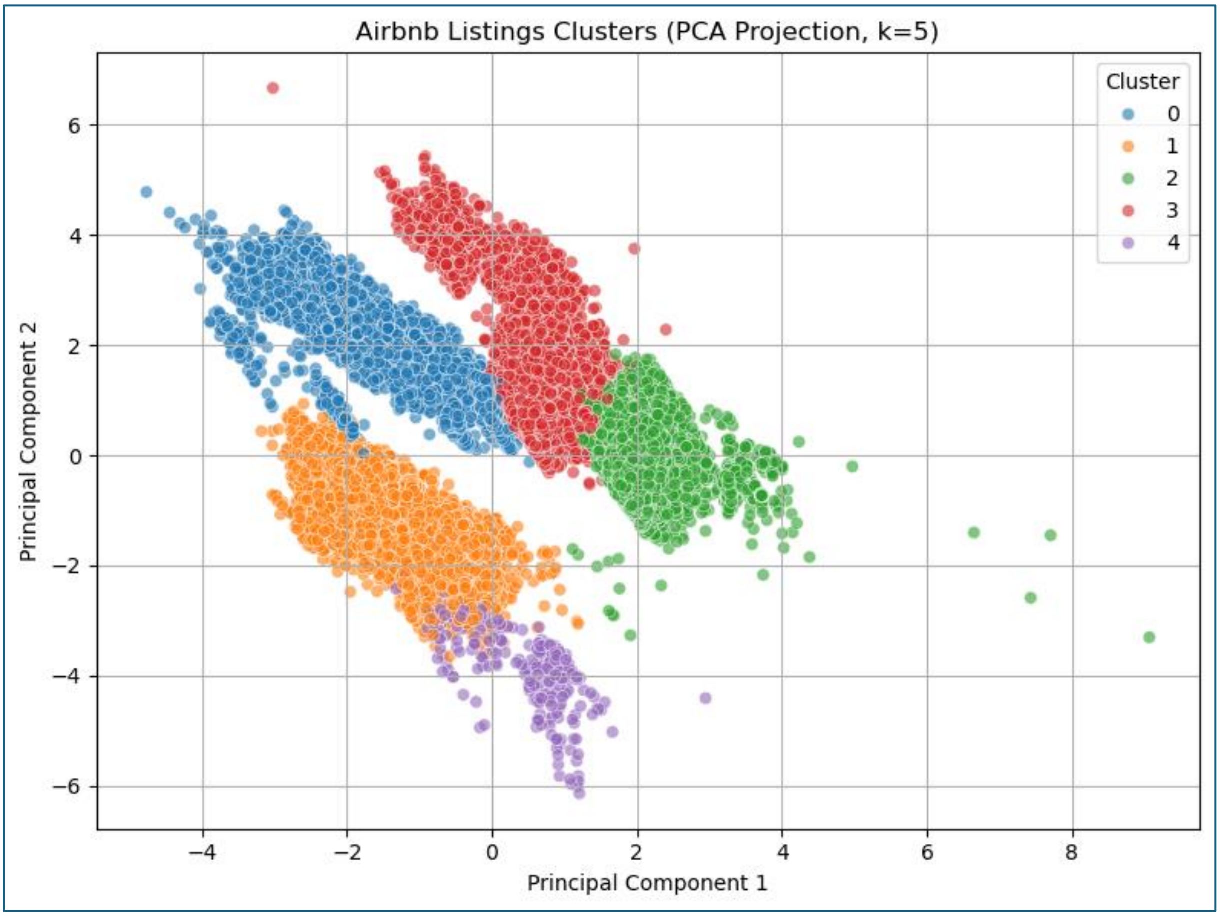


Fig. 2 – Principal Component Analysis of the five K. K-means clustering (Appendix A).

Among these five main clusters we have the number 1 (indicated as cluster zero in Figure 2) which represents the high-price, low-availability. This cluster represents a prime cluster for high-end customers willing to pay prime price for premium location (*Upper Manhattan*) and all the benefits that these locations generate.
For the executive team of Airbnb the cluster number 2 (indicated as cluster one in Figure 2) should be the one to invest time and effort in. It represents the mid-tier pricing option with decent availability in an attractive and alive location (*Central Brooklyn*). This area represents a remarkably interesting potential area for expansion and market growth and the executive team could think about slightly increasing the prices for this cluster and use it as a potentially strategic new area of focus.

Both of these clusters could represent a potential growth, but based on the findings of the analysis, Brooklyn cluster (number 1 in figure 2) could be the short-term period strategy to increase profitability and market share.

Conclusions and recommendations.

This analysis identifies three major factors that drive higher Airbnb prices in New York City: listings located in premium areas such as Manhattan, properties listed as entire homes or apartments, and consistent availability throughout the year.

The clustering analysis also highlights Brooklyn as a key area of opportunity. While it may not be realistic to expect hosts to acquire new properties there, Airbnb can still support growth in Brooklyn by enhancing platform visibility, targeted marketing, and host-focused tools. This borough offers a strong balance of affordability and guest demand, making it well-positioned for strategic investment at the platform level.

Looking ahead, improving pricing models by incorporating more detailed, location-based context like proximity to subway stations, tourist attractions, or cultural events could provide a more accurate reflection of listing value. As Bronnenberg, Dubé, and Gentzkow (2012) explain, “geographic frictions play a significant role in shaping consumer behaviour.” In other words, even minor differences in location can influence booking decisions. By recognising and modelling these subtle spatial dynamics, Airbnb can better align pricing recommendations with what guests are actually willing to pay.

References:

Airbnb Inc. (2025) About Airbnb: What it is and how it works. Available from: https://www.airbnb.com.sg/help/article/2503 [Accessed 2nd June 2025].

Bronnenberg, B. J., Dubé, J.-P., & Gentzkow, M. (2012) The Evolution of Brand Preferences: Evidence from Consumer Migration. *Journal of Marketing Research* 49(1), 61–73. Available from: https://doi.org/10.1509/jmr.11.0203 [Accessed 7th June 2025]

Preda, G. (2023) *Developing Kaggle Notebooks : Pave Your Way to Becoming a Kaggle Notebooks Grandmaster*. First edition. Birmingham, England: Packt Publishing Ltd. Available from: https://learning.oreilly.com/library/view/developing-kaggle-notebooks/9781805128519/Text/Chapter_1.xhtml#_idParaDest-16 [Accessed 2nd June 2025].

Appendix A.

Machine Learning code (Phyton) used by the team to analyse the given dataset "AB_NYC_2019".

```
# @title
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LinearRegression, Ridge, LassoCV,
ElasticNetCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score,
root_mean_squared_error, silhouette_score, calinski_harabasz_score
from sklearn.cluster import KMeans


data = pd.read_csv('AB_NYC_2019.csv')
data['room_type'].unique()
array(['Private room', 'Entire home/apt', 'Shared room'], dtype=object)
```

Data Cleaning

```
# Copy data to avoid chained assignment warnings
data_encoded = data
data_encoded = data_encoded.copy()
# Step 1: Fill missing values explicitly as strings
data_encoded['last_review'] = data_encoded['last_review'].fillna('2019-
01-01')

# Step 2: Convert to datetime safely
data_encoded['last_review'] =
pd.to_datetime(data_encoded['last_review'], errors='coerce')

# Step 3: Calculate days since last review
```

```
reference_date = pd.to_datetime('2019-06-30')
data_encoded['days_since_last_review'] = (reference_date -
data_encoded['last_review']).dt.days


# Remove top 5% price outliers
upper_limit = data_encoded['price'].quantile(0.99)
print(f"99th percentile cutoff: ${upper_limit:.2f}")
data_encoded = data_encoded[data_encoded['price'] <=
upper_limit].copy()

# One-hot encode room_type and neighbourhood_group
data_encoded = pd.get_dummies(
    data_encoded, columns=['room_type', 'neighbourhood_group'],
drop_first=False
)


# Preview result
data_encoded
```

```
99th percentile cutoff: $799.00
```

```
print("Available columns:", data_encoded.columns.tolist())
```

```
Available columns: ['id', 'name', 'host_id', 'host_name',
'neighbourhood', 'latitude', 'longitude', 'price', 'minimum_nights',
'number_of_reviews', 'last_review', 'reviews_per_month',
'calculated_host_listings_count', 'availability_365',
'days_since_last_review', 'room_type_Entire home/apt',
'room_type_Private room', 'room_type_Shared room',
'neighbourhood_group_Bronx', 'neighbourhood_group_Brooklyn',
'neighbourhood_group_Manhattan', 'neighbourhood_group_Queens',
'neighbourhood_group_Staten Island']
```

```
print("Available columns:", data.columns.tolist())
```

```
Available columns: ['id', 'name', 'host_id', 'host_name',
'neighbourhood_group', 'neighbourhood', 'latitude', 'longitude',
'room_type', 'price', 'minimum_nights', 'number_of_reviews',
'last_review', 'reviews_per_month', 'calculated_host_listings_count',
'availability_365']
```

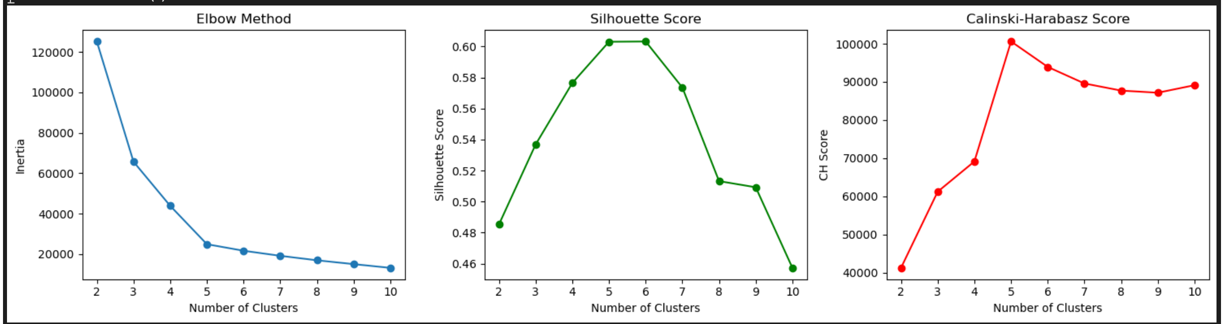
X = input feature = reviews_per_month / availability_365 y = what we want to predict = reviews_per_month

Clustering

```
# Step 1: Define features
cluster_features = [
    'minimum_nights','reviews_per_month','latitude', 'longitude',
    'calculated_host_listings_count', 'availability_365',
'days_since_last_review',
```

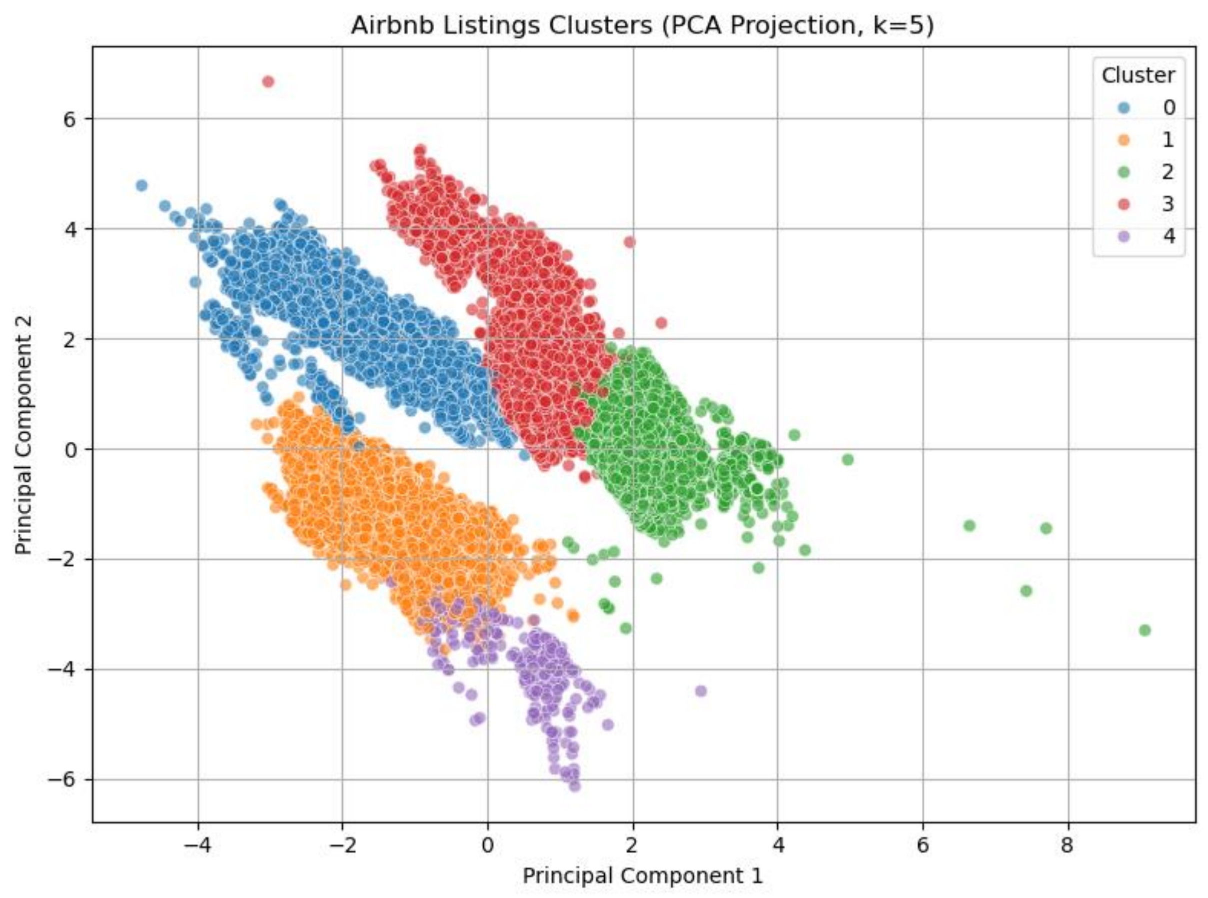
```
    'room_type_Entire home/apt', 'room_type_Private room',
'room_type_Shared room',
    'neighbourhood_group_Bronx', 'neighbourhood_group_Brooklyn',
    'neighbourhood_group_Manhattan', 'neighbourhood_group_Queens',
    'neighbourhood_group_Staten Island'
]

# Step 2: Prepare data
X = data_encoded[cluster_features].copy()
X = X.fillna(0)  # Handle missing values

# Step 3: Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Step 4: Reduce to 2D using PCA
pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X_scaled)

# Step 5: Try clustering with various k
ks = range(2, 11)
inertias, sil_scores, ch_scores = [], [], []

for k in ks:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=25)
    labels = kmeans.fit_predict(X_pca)
    inertias.append(kmeans.inertia_)
    sil_scores.append(silhouette_score(X_pca, labels))
    ch_scores.append(calinski_harabasz_score(X_pca, labels))

# Step 6: Plot evaluation metrics
plt.figure(figsize=(15, 4))

plt.subplot(1, 3, 1)
plt.plot(ks, inertias, marker='o')
plt.title("Elbow Method")
plt.xlabel("Number of Clusters")
plt.ylabel("Inertia")

plt.subplot(1, 3, 2)
plt.plot(ks, sil_scores, marker='o', color='green')
plt.title("Silhouette Score")
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Score")

plt.subplot(1, 3, 3)
plt.plot(ks, ch_scores, marker='o', color='red')
plt.title("Calinski-Harabasz Score")
plt.xlabel("Number of Clusters")
```

```
plt.ylabel("CH Score")

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




```
# 1. Select features for clustering
cluster_features = [
    'minimum_nights','reviews_per_month','latitude', 'longitude',
    'calculated_host_listings_count', 'availability_365',
'days_since_last_review',
    'room_type_Entire home/apt', 'room_type_Private room',
'room_type_Shared room',
    'neighbourhood_group_Bronx', 'neighbourhood_group_Brooklyn',
    'neighbourhood_group_Manhattan', 'neighbourhood_group_Queens',
    'neighbourhood_group_Staten Island'
]

# 2. Prepare X_cluster
X_cluster = data_encoded[cluster_features].copy()
X_cluster = X_cluster.fillna(0)

# 3. Scale
scaler = StandardScaler()
X_cluster = scaler.fit_transform(X_cluster)

# 4. Apply KMeans clustering
kmeans = KMeans(n_clusters=5, random_state=42, n_init=25)
cluster_labels = kmeans.fit_predict(X_cluster)

# 5. Apply PCA for 2D projection
pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X_cluster)

# 6. Create DataFrame for plotting
pca_df = pd.DataFrame(X_pca, columns=['PC1', 'PC2'])
pca_df['cluster'] = cluster_labels

# 7. Visualise clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='cluster',
palette='tab10', alpha=0.6)
```

```
plt.title("Airbnb Listings Clusters (PCA Projection, k=5)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(title='Cluster')
plt.grid(True)
plt.tight_layout()
plt.savefig("airbnb_clusters_pca_2D_k5.png", dpi=300,
bbox_inches='tight')
plt.show()
```



Regression model

```
features = [
    'minimum_nights', 'reviews_per_month', 'latitude', 'longitude',
    'calculated_host_listings_count', 'availability_365',
'days_since_last_review',
    'room_type_Entire home/apt', 'room_type_Private room',
'room_type_Shared room',
    'neighbourhood_group_Bronx', 'neighbourhood_group_Brooklyn',
    'neighbourhood_group_Manhattan', 'neighbourhood_group_Queens',
    'neighbourhood_group_Staten Island'
]

# Step 1: Prepare features and target
X = data_encoded[features].copy()
```

```
y = np.log1p(data_encoded['price'])

# Step 2: Handle missing values
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Step 3: Scale
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

# Step 4: Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

# Step 5: LassoCV
lasso_cv = LassoCV(alphas=np.logspace(-4, 0, 50), cv=5,
random_state=42)
lasso_cv.fit(X_train, y_train)

# Step 6: Define models
models = {
    "Linear": LinearRegression(),
    "Ridge": Ridge(),
    "Lasso (CV)": lasso_cv,
    "Random Forest": RandomForestRegressor(n_estimators=100,
random_state=42)
}

# Step 7: Evaluate each model
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred_log = model.predict(X_test)
    y_pred = np.expm1(y_pred_log)
    y_test_actual = np.expm1(y_test)

    rmse = root_mean_squared_error(y_test_actual, y_pred)

    print(f"\n{name} Results:")
    print(f"  MAE:  ${mean_absolute_error(y_test_actual, y_pred):.2f}")
    print(f"  RMSE: ${rmse:.2f}")
    print(f"  R²:   {r2_score(y_test_actual, y_pred):.3f}")
```

```
Linear Results:
  MAE:  $48.18
  RMSE: $84.49
  R²:   0.336

Ridge Results:
```

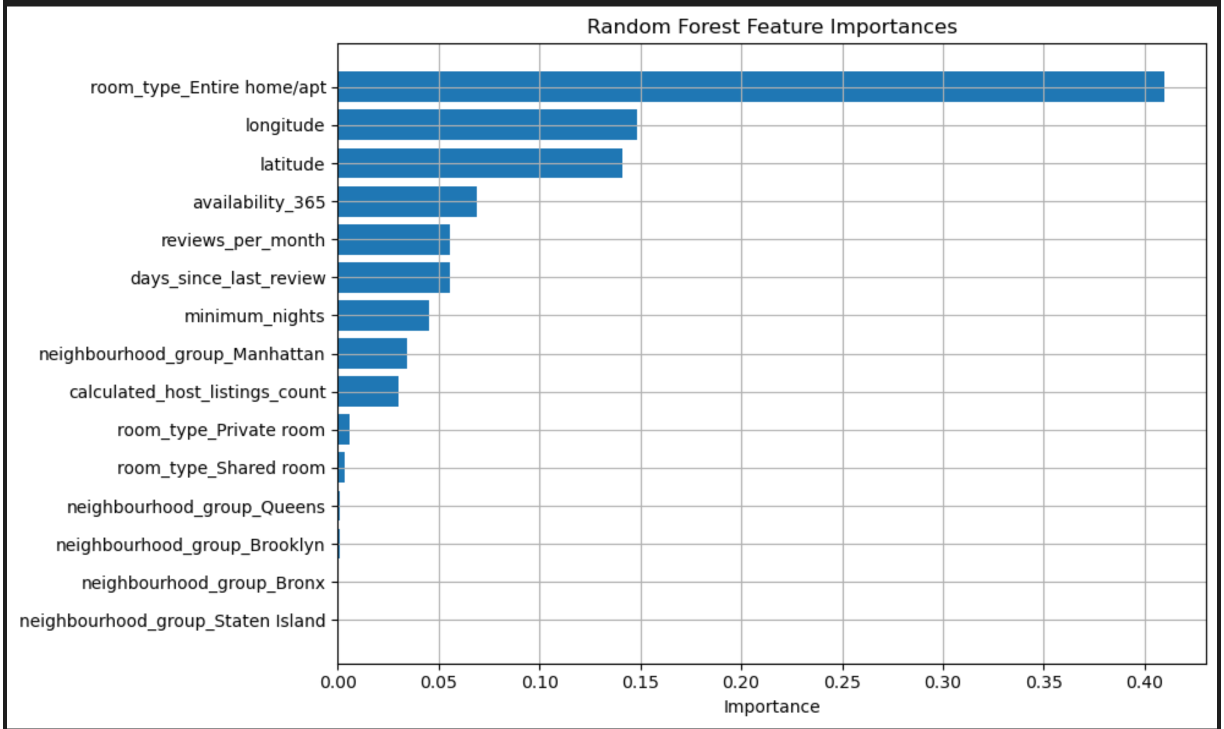
```
  MAE:  $48.20
  RMSE: $84.45
  R²:   0.337

Lasso (CV) Results:
  MAE:  $48.20
  RMSE: $84.46
  R²:   0.337

Random Forest Results:
  MAE:  $42.62
  RMSE: $74.04
  R²:   0.490
```

Random Forest

```python
# Feature list
features = [
    'minimum_nights', 'reviews_per_month', 'latitude', 'longitude',
    'calculated_host_listings_count', 'availability_365',
'days_since_last_review',
    'room_type_Entire home/apt', 'room_type_Private room',
'room_type_Shared room',
    'neighbourhood_group_Bronx', 'neighbourhood_group_Brooklyn',
    'neighbourhood_group_Manhattan', 'neighbourhood_group_Queens',
    'neighbourhood_group_Staten Island'
]

# Step 1: Prepare features and target
X = data_encoded[features].copy()
y = np.log1p(data_encoded['price'])  # log-transform target

# Step 2: Impute missing values
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Step 3: Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_imputed)

# Step 4: Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

# Step 5: Train Random Forest
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# Step 6: Predict and evaluate
y_pred_log = rf_model.predict(X_test)
```

```
y_pred = np.expm1(y_pred_log)
y_test_actual = np.expm1(y_test)
print("\nRandom Forest Results:")
print(f"  MAE:  ${mean_absolute_error(y_test_actual, y_pred):.2f}")
print(f"  RMSE: ${root_mean_squared_error(y_test_actual, y_pred):.2f}")
print(f"  R²:    {r2_score(y_test_actual, y_pred):.3f}")

# Step 7: Plot feature importances
importances = rf_model.feature_importances_
sorted_idx = np.argsort(importances)[::-1]
sorted_features = np.array(features)[sorted_idx]

plt.figure(figsize=(10, 6))
plt.barh(sorted_features, importances[sorted_idx])
plt.gca().invert_yaxis()
plt.title("Random Forest Feature Importances")
plt.xlabel("Importance")
plt.grid(True)
plt.tight_layout()
plt.show()
```

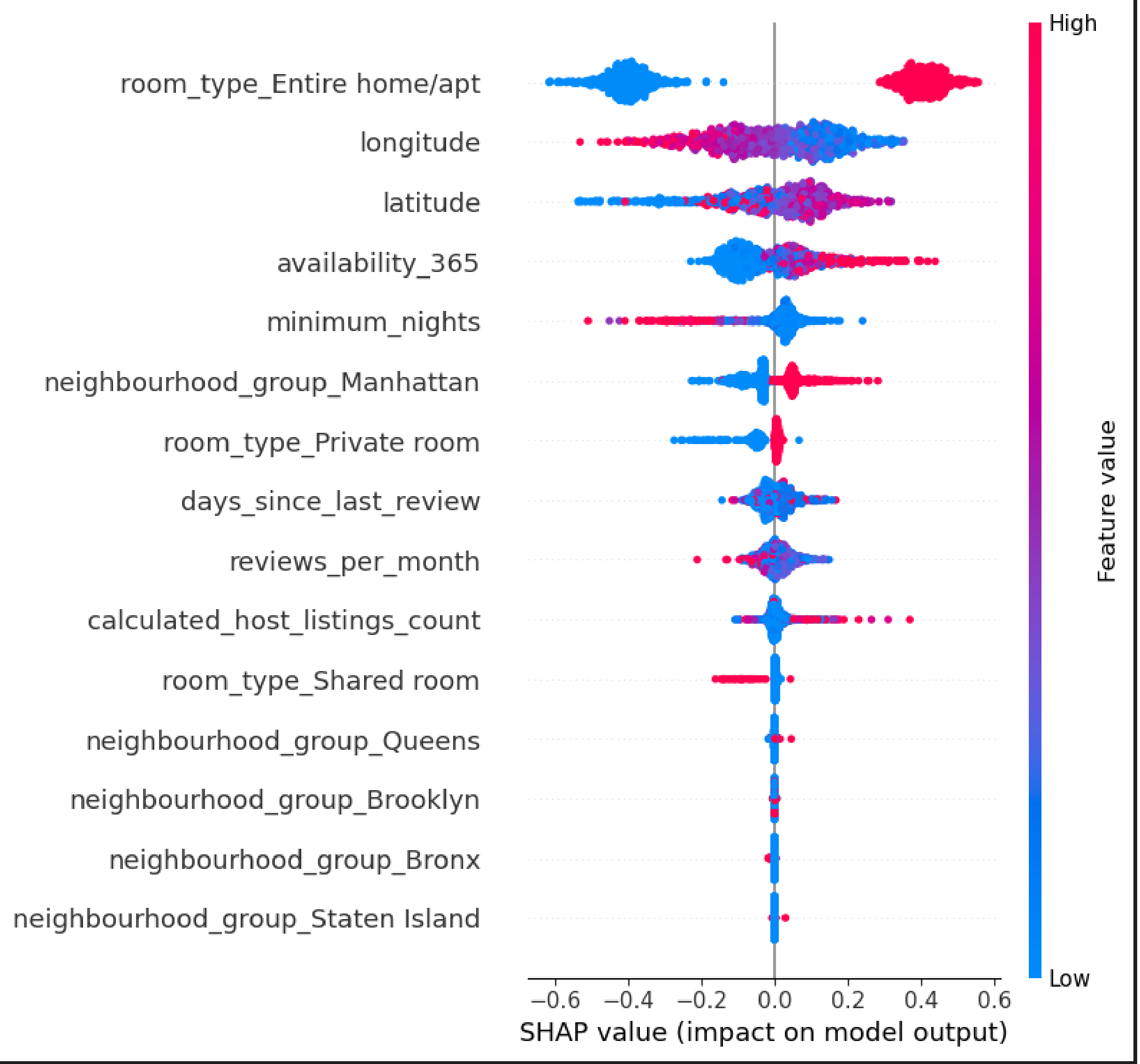
```
Random Forest Results:
  MAE:   $42.62
  RMSE:  $74.04
  R²:    0.490
```



SHAP

```
import shap

# Step 8: SHAP summary (fast with sample)
```

```
X_train_df = pd.DataFrame(X_train, columns=features)
X_sample = X_train_df.sample(1000, random_state=42)  # ✅ sample only

# Create SHAP explainer
explainer = shap.TreeExplainer(rf_model)

# Compute SHAP values
shap_values = explainer.shap_values(X_sample)

# SHAP summary plot (beeswarm)
shap.summary_plot(shap_values, X_sample)  # ✅ no indentation error
here
```



Price prediction model

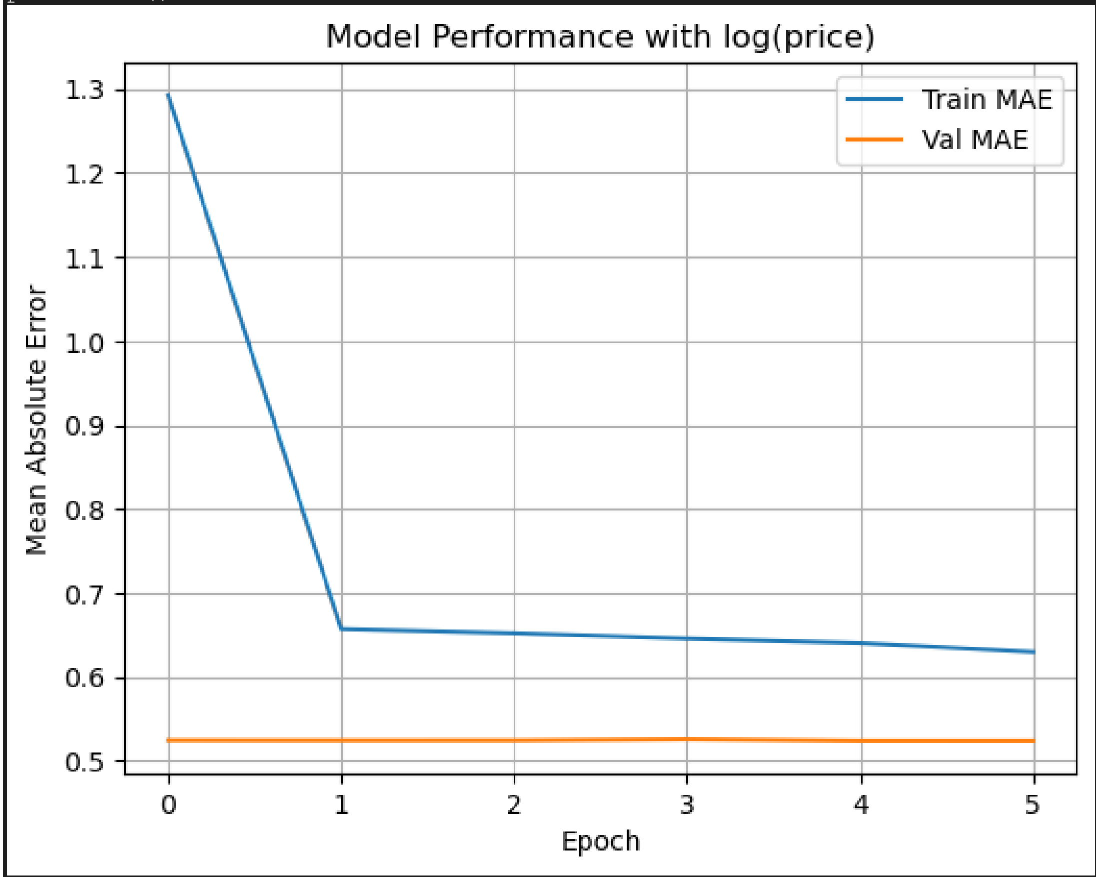
```
from tensorflow.keras.layers import Dropout
# Features and target
data['log_reviews'] = np.log1p(data['reviews_per_month'])
data['log_min_nights'] = np.log1p(data['minimum_nights'])
```

```
features = [  'minimum_nights','reviews_per_month','latitude',
'longitude',
    'calculated_host_listings_count', 'availability_365',
'days_since_last_review',
    'room_type_Entire home/apt', 'room_type_Private room',
'room_type_Shared room', 'neighbourhood_group_Bronx',
                    'neighbourhood_group_Brooklyn',
'neighbourhood_group_Manhattan', 'neighbourhood_group_Queens',
                    'neighbourhood_group_Staten Island']
numeric_features = ['minimum_nights','reviews_per_month','latitude',
'longitude',
    'calculated_host_listings_count', 'availability_365',
'days_since_last_review']
categorical_features = [ 'room_type_Entire home/apt',
'room_type_Private room', 'room_type_Shared room',
'neighbourhood_group_Bronx',
                    'neighbourhood_group_Brooklyn',
'neighbourhood_group_Manhattan', 'neighbourhood_group_Queens',
                    'neighbourhood_group_Staten Island']
target = 'price'

preprocessor = ColumnTransformer(transformers=[
    ('num', StandardScaler(), numeric_features),
    ('cat', OneHotEncoder(drop='first'), categorical_features)
])

X = data_encoded[features]
y = np.log1p(data['price'])

X_processed = preprocessor.fit_transform(X)

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

model = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1)
])

model.compile(optimizer='adam', loss='mse', metrics=['mae'])
history = model.fit(X_train, y_train, validation_split=0.2, epochs=50,
batch_size=32, verbose=1)
```

```
from tensorflow.keras.layers import Dropout
from tensorflow.keras.callbacks import EarlyStopping

data_encoded['log_reviews'] =
np.log1p(data_encoded['reviews_per_month'])
data_encoded['log_min_nights'] =
np.log1p(data_encoded['minimum_nights'])

# Feature list
features = [
    'minimum_nights', 'reviews_per_month', 'latitude', 'longitude',
    'calculated_host_listings_count', 'availability_365',
'days_since_last_review',
    'log_reviews', 'log_min_nights',
    'room_type_Entire home/apt', 'room_type_Private room',
'room_type_Shared room',
    'neighbourhood_group_Bronx', 'neighbourhood_group_Brooklyn',
    'neighbourhood_group_Manhattan', 'neighbourhood_group_Queens',
    'neighbourhood_group_Staten Island'
]

# Feature groups
numeric_features = [
    'minimum_nights', 'reviews_per_month', 'latitude', 'longitude',
    'log_reviews', 'log_min_nights',
    'calculated_host_listings_count', 'availability_365',
'days_since_last_review'
]

categorical_features = [
    'room_type_Entire home/apt', 'room_type_Private room',
'room_type_Shared room',
    'neighbourhood_group_Bronx', 'neighbourhood_group_Brooklyn',
    'neighbourhood_group_Manhattan', 'neighbourhood_group_Queens',
    'neighbourhood_group_Staten Island'
]

# Preprocessing
preprocessor = ColumnTransformer(transformers=[
    ('num', StandardScaler(), numeric_features),
    ('cat', 'passthrough', categorical_features)  # Already one-hot
encoded
])

# Define X and y
X = data_encoded[features]
y = np.log1p(data_encoded['price'])  # Log-transformed target

# Apply preprocessing
```

```python
X_processed = preprocessor.fit_transform(X)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X_processed, y, test_size=0.2, random_state=42
)

# Build model
model = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(1)
])

model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Add early stopping
early_stop = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)

# Train model
history = model.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=50,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)

# Evaluate performance in original scale
y_pred_log = model.predict(X_test).flatten()
y_pred = np.expm1(y_pred_log)
y_test_actual = np.expm1(y_test)

print("\nEvaluation on actual price scale:")
print(f"  MAE:  ${mean_absolute_error(y_test_actual, y_pred):.2f}")
print(f"  RMSE: ${root_mean_squared_error(y_test_actual, y_pred):.2f}")
print(f"  R²:   {r2_score(y_test_actual, y_pred):.3f}")


import matplotlib.pyplot as plt
```

```python
plt.plot(history.history['mae'], label='Train MAE')
plt.plot(history.history['val_mae'], label='Val MAE')
plt.xlabel('Epoch')
plt.ylabel('Mean Absolute Error')
plt.title('Model Performance with log(price)')
plt.legend()
plt.grid(True)

# Save the figure BEFORE calling plt.show()
plt.savefig("model_performance_log_price.png", bbox_inches='tight',
dpi=300)

plt.show()
```

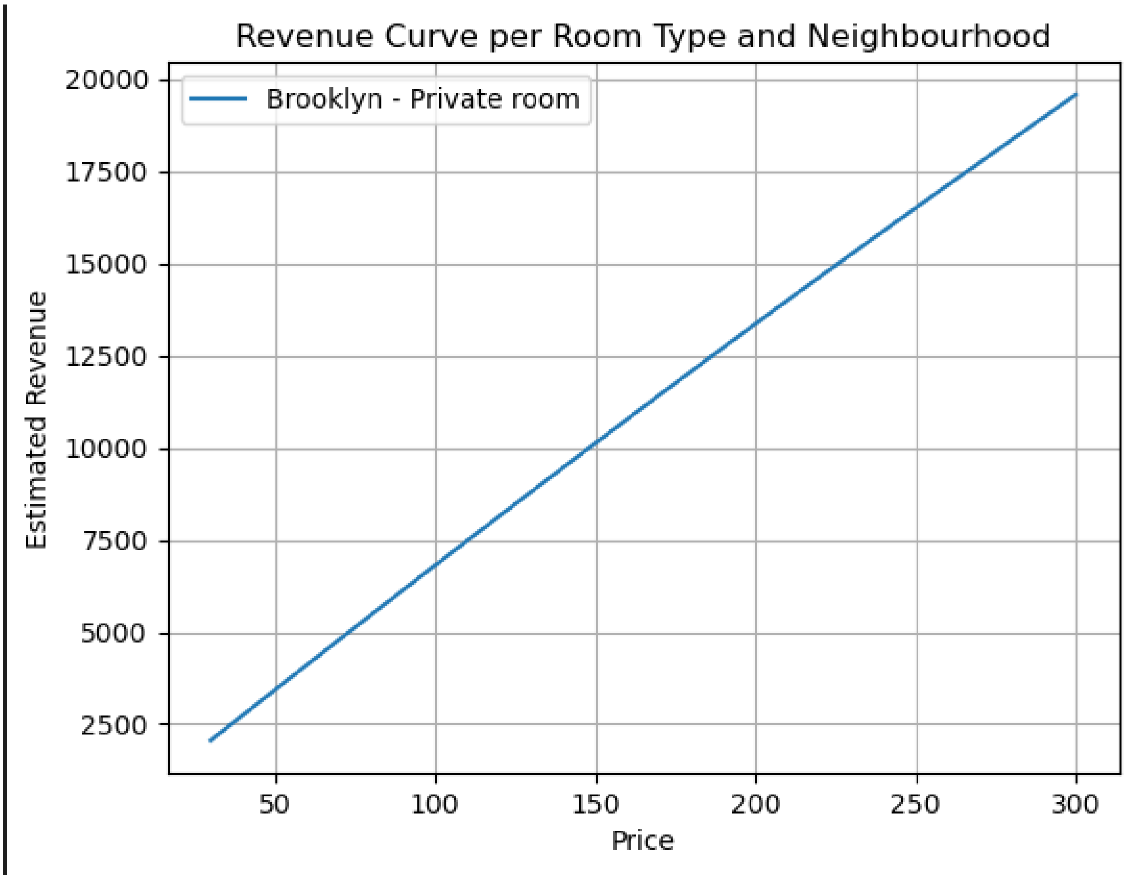



```python
min_night_range = np.arange(1, 31)
sim_data = pd.DataFrame([{
    'occupancy_ratio': 0.3,
    'price': 100,
    'minimum_nights': mn,
    'availability_365': 365
} for mn in min_night_range])

sim_scaled = scaler.transform(sim_data)

# Predict demand
predicted_demand = model.predict(sim_scaled).flatten()
```

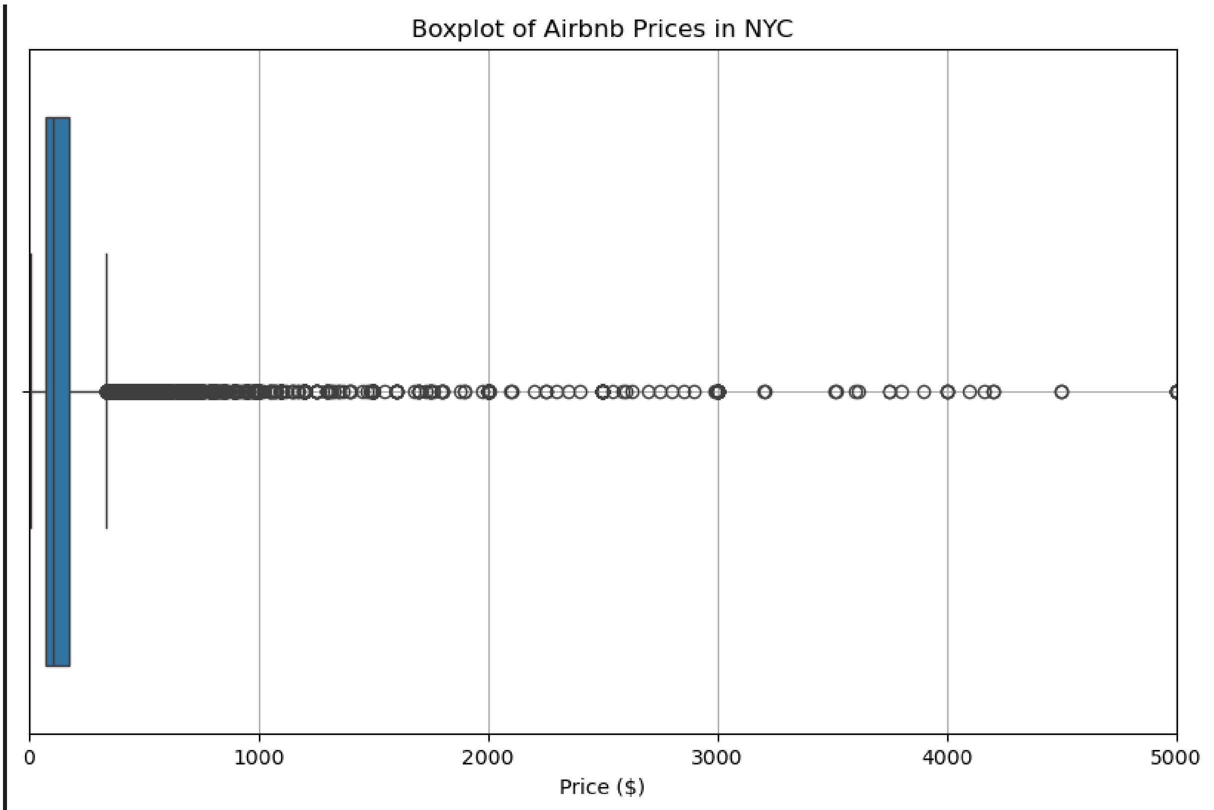
```python
pred = model.predict(sim_scaled).flatten()

pd.DataFrame({
    'minimum_nights': min_night_range,
    'predicted_reviews_per_month': pred
})
price_range = np.linspace(30, 300, 100)

base_listing = {
    'occupancy_ratio': 0.3,
    'minimum_nights': 2,
    'availability_365': 365,
    'room_type': 'Private room',
    'neighbourhood_group': 'Brooklyn'
}

# Repeat base listing and vary price
sim_data = pd.DataFrame([
    {**base_listing, 'price': p} for p in price_range
])


X_sim = preprocessor.transform(sim_data)

pred_demand = model.predict(X_sim).flatten()
sim_data['predicted_reviews_per_month'] = pred_demand
sim_data['estimated_revenue'] = sim_data['price'] * pred_demand
best = sim_data.loc[sim_data['estimated_revenue'].idxmax()]
print(f"✅ Best price: ${best['price']:.2f}")
print(f"📈 Expected reviews/month: {best['predicted_reviews_per_month']:.2f}")
print(f"💰 Estimated revenue: ${best['estimated_revenue']:.2f}")
```

```
Best price: $300.00
📈 Expected reviews/month: 65.24
💰 Estimated revenue: $19572.62
```

```python
for (ng, rt), group in sim_data.groupby(['neighbourhood_group', 'room_type']):
    plt.plot(group['price'], group['estimated_revenue'], label=f"{ng} - {rt}")
plt.xlabel("Price")
plt.ylabel("Estimated Revenue")
plt.title("Revenue Curve per Room Type and Neighbourhood")
plt.legend()
plt.grid(True)
plt.show()
```




```
mport shap

explainer = shap.Explainer(model, X_train)
shap_values = explainer(X_test[:100])
shap.plots.beeswarm(shap_values)
feature_names = preprocessor.get_feature_names_out()
print(feature_names)
data_price = data[data['price'] > 0]

# Draw boxplot
plt.figure(figsize=(10, 6))
sns.boxplot(x=data_price['price'])
plt.title('Boxplot of Airbnb Prices in NYC')
plt.xlabel('Price ($)')
plt.xlim(0, 5000)  # adjust to ignore extreme outliers in view
plt.grid(True)
plt.show()
```




```
upper_limit = data['price'].quantile(0.99)  # top 1% cutoff
print(f"99th percentile cutoff: ${upper_limit:.2f}")

data_price = data[data['price'] <= upper_limit]
Q1 = data['price'].quantile(0.25)
Q3 = data['price'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

print(f"Lower bound: {lower_bound:.2f}, Upper bound:
{upper_bound:.2f}")
# Handle missing last_review
data['last_review'] = data['last_review'].fillna('2019-01-01')  # Fill
missing
data['last_review'] = pd.to_datetime(data['last_review'])        #
Convert to datetime
reference_date = pd.to_datetime('2019-06-30')                    # Fixed
reference
data['days_since_last_review'] = (reference_date -
data['last_review']).dt.days  # Calculate

missing_count = data['room_type'].isna().sum()
print(f"Missing values in 'room_type': {missing_count}")
```

Log Price comparison

```
import seaborn as sns
```

```python
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
sns.histplot(data['price'], bins=100)
plt.title("Original Price Distribution")

plt.subplot(1, 2, 2)
sns.histplot(np.log1p(data['price']), bins=100)
plt.title("Log-Transformed Price Distribution")

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