**Gachon University**

**School of Computing**

**2025-1**

**Data Science (14455\_004)**

**Term project Final Explanation**

Team 2

202135774 박찬우

202334536 정성경

202135716 김강민

202334455 남윤정

202235090 이재혁

**Contents**

**Ⅰ.** Business Objectives

**Ⅱ.** Data Exploration

**Ⅲ.** Data Description

**Ⅳ.** User

A. Data Preprocessing

B. Data Analysis

C. Data Evaluation

D. Conclusion

**Ⅴ.** Host

A. Data Preprocessing

B. Data Analysis

C. Data Evaluation

D. Conclusion

**Ⅰ.** Business Objectives

Our object of this project is to find a dataset that satisfies ‘Reasonable number of records and features & Reasonable amount of dirty data & Combination of numerical data and categorical data’, and set extra goals that can use regression or classification algorithms.(ex: Predict who will cancel the mobile phone carrier by doing regression of ‘phone carrier customers dataset’.)

**Ⅱ.** Data Exploration

We found few datasets:

\*Telco Customer Churn: No missing data.

<https://www.kaggle.com/datasets/blastchar/telco-customer-churn/data>

\*Credit Score Classification: Too many data. (rows)

<https://www.kaggle.com/datasets/parisrohan/credit-score-classification?select=train.csv>

\*New York City Airbnb Open Data: Enough data, Some missing data, Includes categorical data.

<https://www.kaggle.com/datasets/dgomonov/new-york-city-airbnb-open-data/data>

‘New York City Airbnb Open Data’ satisfies our goals, so we decided to choose this dataset.

**- Project objective**

Our goal is get a classification analysis with using regression or clustering. Due to the character of dataset, we decided using clustering.

1. **User-Centric Clustering and Classification**  
   From the end-user perspective, we aim to engineer a new feature that represents the overall *quality* of a listing, based on factors such as price, room type, location, and number of reviews. Using this feature, we will perform clustering to group listings into distinct quality tiers, helping customers easily find accommodations that match their preferences. Ultimately, we will formulate a classification task to predict this newly created quality feature on unseen (test) data.
2. **Host-Oriented Price Optimization via Regression**  
   From the host's perspective, we aim to develop a regression model that predicts an appropriate price for a given listing based on its attributes. This will enable hosts to assess whether their current pricing is aligned with market expectations and optimize their listing strategy accordingly.

By getting through the progress above, we can get meaningful inspection of Air bnb data.

**Ⅲ.** Data Description

Description of each features:

Column: id – Listing ID

* Data Type: int64
* Missing Values: 0
* Min: 2,539
* Max: 36,487,245
* Mean: 19,017,143.24

Column: name – Name of the Listing

* Data Type: object
* Missing Values: 16
* Unique Values: 47,905
* Examples:
  + Clean & quiet apt home by the park
  + Skylit Midtown Castle
  + THE VILLAGE OF HARLEM....NEW YORK !
  + Cozy Entire Floor of Brownstone
  + Entire Apt: Spacious Studio/Loft by Central Park

Column: host\_id – Host ID

* Data Type: int64
* Missing Values: 0
* Min: 2,438
* Max: 274,321,313
* Mean: 67,620,010.65

Column: host\_name – Name of the Host

* Data Type: object
* Missing Values: 21
* Unique Values: 11,452
* Examples: John, Jennifer, Elisabeth, LisaRoxanne, Laura

Column: neighbourhood\_group – Borough

* Data Type: object
* Missing Values: 0
* Unique Values: 5
* Examples: Brooklyn, Manhattan, Queens, Staten Island, Bronx

Column: neighbourhood – Neighborhood

* Data Type: object
* Missing Values: 0
* Unique Values: 221
* Examples: Kensington, Midtown, Harlem, Clinton Hill, East Harlem

Column: latitude – Latitude Coordinates

* Data Type: float64
* Missing Values: 0
* Min: 40.49979
* Max: 40.91306
* Mean: 40.73

Column: longitude – Longitude Coordinates

* Data Type: float64
* Missing Values: 0
* Min: -74.24442
* Max: -73.71299
* Mean: -73.95

Column: room\_type – Type of Space

* Data Type: object
* Missing Values: 0
* Unique Values: 3
* Examples: Private room, Entire home/apt, Shared room

Column: price – Price in USD

* Data Type: int64
* Missing Values: 0
* Min: 0
* Max: 10,000
* **Mean**: 152.72

Column: minimum\_nights – Minimum Nights Required

* Data Type: int64
* Missing Values: 0
* Min: 1
* Max: 1,250
* Mean: 7.03

Column: number\_of\_reviews – Number of Reviews

* Data Type: int64
* Missing Values: 0
* Min: 0
* Max: 629
* Mean: 23.27

Column: last\_review – Date of Most Recent Review

* Data Type: object
* Missing Values: 10,052
* Unique Values: 1,764
* Examples: 2018-10-19, 2019-05-21, 2019-07-05, 2018-11-19, 2019-06-22

Column: reviews\_per\_month – Average Monthly Reviews

* Data Type: float64
* Missing Values: 10,052
* Min: 0.01
* Max: 58.5
* Mean: 1.37

Column: calculated\_host\_listings\_count – Listings per Host

* Data Type: int64
* Missing Values: 0
* Min: 1
* Max: 327
* Mean: 7.14

Column: availability\_365 – Days Available in a Year

* Data Type: int64
* Missing Values: 0
* Min: 0
* Max: 365
* Mean: 112.78

Description, Visualization of data by code:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 48895 entries, 0 to 48894

Data columns (total 16 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 id 48895 non-null int64

1 name 48879 non-null object

2 host\_id 48895 non-null int64

3 host\_name 48874 non-null object

4 neighbourhood\_group 48895 non-null object

5 neighbourhood 48895 non-null object

6 latitude 48895 non-null float64

7 longitude 48895 non-null float64

8 room\_type 48895 non-null object

9 price 48895 non-null int64

10 minimum\_nights 48895 non-null int64

11 number\_of\_reviews 48895 non-null int64

12 last\_review 38843 non-null object

13 reviews\_per\_month 38843 non-null float64

14 calculated\_host\_listings\_count 48895 non-null int64

15 availability\_365 48895 non-null int64

dtypes: float64(3), int64(7), object(6)

memory usage: 6.0+ MB

#Missing data of each features:

텍스트, 스크린샷, 폰트, 디자인이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

**Ⅳ.** User

A. Data Preprocessing

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import folium

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

from datetime import datetime

RANDOM\_STATE = 42

REFERENCE\_DATE = pd.to\_datetime("2019-12-01")

#A function of calculating the distance between the location of each

#accommodation and the location of the city center (central) in the area

def haversine(lat1, lon1, lat2, lon2):

R = 6371

φ1, φ2 = np.radians(lat1), np.radians(lat2)

Δφ = φ2 - φ1

Δλ = np.radians(lon2 - lon1)

a = np.sin(Δφ/2)\*\*2 + np.cos(φ1)\*np.cos(φ2)\*np.sin(Δλ/2)\*\*2

c = 2 \* np.arctan2(np.sqrt(a), np.sqrt(1-a))

return R \* c

centers = {

'Bronx': (40.8448, -73.8648),

'Brooklyn': (40.6782, -73.9442),

'Manhattan': (40.7685, -73.9822),

'Queens': (40.7282, -73.7949),

'Staten Island': (40.5795, -74.1502),

}

#Analyze the properties of each cluster and label the cluster according

#to the characteristics of each of the four clusters

cluster\_names = {

0: "A high-end hotel located in the city center",

1: "A low-cost room outside",

2: "Accommodation suitable for mid-priced sightseeing",

3: "A long-term rental-oriented outer-style accommodation",

}

#Preprocessing function

#one-hot encoding for 'neighbourhood', 'room\_type', drop few columns

def preprocess(df):

df = df.copy()

#Drop all features that do not affect the result

df.drop(columns=['id', 'name', 'host\_id', 'host\_name', 'neighbourhood'], errors='ignore', inplace=True)

if 'neighbourhood\_group' in df.columns:

df['distance\_to\_center'] = df.apply(lambda row: haversine(

row['latitude'], row['longitude'],

\*centers.get(row['neighbourhood\_group'], (0, 0))

), axis=1)

df = pd.get\_dummies(df, columns=['neighbourhood\_group'])

df['last\_review'] = pd.to\_datetime(df['last\_review'], errors='coerce')

df['days\_since\_oldest\_review'] = (REFERENCE\_DATE - df['last\_review']).dt.days

#Missing value processing: no review → considered older than the oldest value max + 30

df['days\_since\_oldest\_review'] = df['days\_since\_oldest\_review'].fillna(df['days\_since\_oldest\_review'].max() + 30)

max\_days = df['days\_since\_oldest\_review'].max()

df['days\_since\_oldest\_review'] = max\_days - df['days\_since\_oldest\_review']

df.drop(columns=['last\_review'], inplace=True)

df['reviews\_per\_month'].fillna(0, inplace=True)

df = pd.get\_dummies(df, columns=['room\_type'])

df.drop(columns=['latitude', 'longitude'], inplace=True)

return df

‘id', 'name', 'host\_id', 'host\_name', 'neighbourhood’ features were dropped, because these features do not affects to result.

Only ‘name’, ‘host\_name’ has missing values, so dealing with missing values are also solved. ‘review\_per\_month’ also has missing values, so filled with 0 if data is missing.

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.‘last\_review‘ column in the Airbnb dataset → the recency of each listing’s last review

* If a listing has no review (NaT → NaN), it is considered to be "30 days older than the oldest review" in the dataset.
* ‘last\_review’ columns has missing values, so we consider it by adding 30 from oldest value, so that it gets lowest score.

One-Hot Encoding : ‘Neighbourhood\_group’, ‘room\_type’

* They are categorical. We used one-hot encoding method to make feature numerical(Reasons are in Open SW(1)).

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.텍스트, 스크린샷, 멀티미디어 소프트웨어, 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

scaler = StandardScaler()

df\_scaled = pd.DataFrame(scaler.fit\_transform(df\_proc), columns=df\_proc.columns)

Data scaled before clustering, classifying.

B. Data Analysis

# ========================

# Training Phase

# ========================

df = pd.read\_csv("AB\_NYC\_2019.csv")

df\_proc = preprocess(df)

X\_for\_clustering = df\_proc.copy()

X\_for\_clustering.drop(columns='cluster', inplace=True, errors='ignore')

sse = [] # Sum of Squared Errors

scaler = StandardScaler()

df\_scaled = pd.DataFrame(scaler.fit\_transform(df\_proc), columns=df\_proc.columns)

#

# Perform KMeans by changing the K value from 1 to 10

for k in range(1, 11):

kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init='auto')

kmeans.fit(X\_for\_clustering)

sse.append(kmeans.inertia\_) # inertia\_ == SSE

# present graph

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

plt.plot(range(1, 11), sse, marker='o')

plt.title('Elbow Method For Optimal k')

plt.xlabel('Number of clusters (k)')

plt.ylabel('SSE (Inertia)')

plt.grid(True)

plt.show()

# Use silhouette analysis to search for optimal K values

silhouette\_scores = []

for k in range(2, 6):

kmeans = KMeans(n\_clusters=k, random\_state=0, n\_init='auto')

kmeans.fit(X\_for\_clustering)

score = silhouette\_score(X\_for\_clustering, kmeans.labels\_)

silhouette\_scores.append(score)

# present graph

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

plt.plot(range(2, 6), silhouette\_scores, marker='o')

plt.title('Silhouette Score For Optimal k')

plt.xlabel('Number of clusters (k)')

plt.ylabel('Silhouette Score')

plt.grid(True)

plt.show()

#

kmeans = KMeans(n\_clusters=4, random\_state=RANDOM\_STATE, n\_init=10)

df\_scaled['cluster'] = kmeans.fit\_predict(df\_scaled)

X = df\_scaled.drop('cluster', axis=1)

y = df\_scaled['cluster']

We tried to find the most optimal K value through elbow by executing kmeans while changing the k value, but we could not confirm a clear elbow above k=3. Therefore, we tried to set the k value using the silhouette method, and it was determined as k=4.

텍스트, 라인, 도표, 그래프이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

\* Elbow of Optimal K

라인, 도표, 그래프, 평행이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

\* Silhouette of Optimal K

kmeans = KMeans(n\_clusters=4, random\_state=RANDOM\_STATE, n\_init=10)

df\_scaled['cluster'] = kmeans.fit\_predict(df\_scaled)

X = df\_scaled.drop('cluster', axis=1)

y = df\_scaled['cluster']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, stratify=y, random\_state=RANDOM\_STATE)

Due to result of Silhouette, set K by 4.

Analyzing clustering results

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* **Cluster 0: Practical**

- Moderate price

- Mostly located in Brooklyn

- Closest to city center

- Higher proportion of Private rooms

=> Practical and accessible mid-range accommodations.

* **Cluster 1: Premium**

- High price

- Mostly located in Manhattan

- Close to city center

- High proportion of Entire home/apartments

=> Premium accommodations with private space in central locations.

* **Cluster 2: Low-cost / Outside**

- Low price

- Located in Bronx and Queens

- Far from city center

- Mostly Private and Shared rooms, very few Entire homes

=> Budget accommodations in outskirts with shared space.

* **Cluster 3: Long-term / Shared**

- Low to medium price

- Located in Queens and Bronx

- Farthest from city center

- Highest proportion of Shared rooms

=> Long-term stay or shared-type accommodations in remote areas.

텍스트, 라인, 도표, 그래프이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

텍스트, 스크린샷, 도표, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

C. Data Evaluation

# Tuned RandomForestClassifier(low depth)

# Lower depth because of OVERFITTING

# Hpyerparameters are set by RandomCV

rf = RandomForestClassifier(

n\_estimators=100,

max\_depth=10,

min\_samples\_split=5,

min\_samples\_leaf=3,

random\_state=RANDOM\_STATE,

oob\_score=True,

n\_jobs=-1

)

rf.fit(X\_train, y\_train)

y\_pred = rf.predict(X\_test)

#cross-validation accuracy(to check that high OOB score means overfitting, or great score of model)

cv\_scores = cross\_val\_score(rf, X, y, cv=5, scoring='accuracy')

print("\nCross-Validation Scores:", cv\_scores)

print("\nMean Cross-Validation Accuracy:", np.mean(cv\_scores))

# ========================

# Visualization

# ========================

def visualize\_results(X, y, y\_pred, model):

fig, axs = plt.subplots(2, 2, figsize=(14, 12))

# PCA visualization

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

axs[0, 0].scatter(X\_pca[:, 0], X\_pca[:, 1], c=y, cmap='viridis', alpha=0.6)

axs[0, 0].set\_title("PCA: Clusters")

axs[0, 0].set\_xlabel("PC 1")

axs[0, 0].set\_ylabel("PC 2")

# Confusion matrix

cm = confusion\_matrix(y, y\_pred)

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axs[0, 1])

axs[0, 1].set\_title("Confusion Matrix")

axs[0, 1].set\_xlabel("Predicted")

axs[0, 1].set\_ylabel("Actual")

# Feature importances

feat\_imp = pd.Series(model.feature\_importances\_, index=X.columns).sort\_values(ascending=False)

sns.barplot(x=feat\_imp[:10], y=feat\_imp[:10].index, ax=axs[1, 0])

axs[1, 0].set\_title("Top 10 Feature Importances")

# OOB vs Test Accuracy

acc = accuracy\_score(y, y\_pred)

axs[1, 1].bar(["OOB Score", "Test Accuracy"], [model.oob\_score\_, acc], color=["orange", "blue"])

axs[1, 1].set\_ylim(0.5, 1.0)

axs[1, 1].set\_title("Overfitting Check: OOB vs Test Accuracy")

plt.tight\_layout()

plt.show()

**\*Clustering with KMeans**

To establish initial labels for classification, KMeans was applied with n\_clusters=4. The number of clusters was chosen based on Elbow Method and Silhouette Score analysis, which indicated that 4 clusters captured meaningful variance in the data.

Importantly, KMeans was performed on the scaled dataset, since raw feature magnitudes could bias the cluster centers otherwise.

These clusters served as target labels (y) for the classification model.

**\*Classification with RandomForest**

We trained a RandomForestClassifier to learn how to predict the cluster a new listing belongs to, based on its features. To avoid overfitting:

We limited the depth of trees (max\_depth=10)

Used conservative splits (min\_samples\_split=5, min\_samples\_leaf=3)

Enabled Out-of-Bag (OOB) scoring to validate generalization on unseen samples.

All hyperparameters are set by results of randomsearchcv.

print("\n--- Best Parameters and Score ---")

print("Best parameters found: ", random\_search.best\_params\_)

print("Best cross-validation accuracy: {:.4f}".format(random\_search.best\_score\_))

best\_rf\_model = random\_search.best\_estimator\_

print("\n--- Final Evaluation on Test Set with Best Model ---")

y\_pred\_final = best\_rf\_model.predict(X\_test)

print("Test Accuracy:", accuracy\_score(y\_test, y\_pred\_final))

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred\_final))

print("\nConfusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_final))

print("\n--- Feature Importances from Best Random Forest Model ---")

feature\_importances = pd.Series(best\_rf\_model.feature\_importances\_, index=X.columns)

sorted\_importances = feature\_importances.sort\_values(ascending=False)

print(sorted\_importances.head(10))

텍스트, 영수증, 스크린샷, 대수학이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

But, actual hyperparameters are changed to solve the overfitting problems.

### **OOB (Out-of-Bag) Score**

**Out-of-Bag (OOB) score** is an internal validation method used by ensemble models like **Random Forests** to estimate model performance without requiring a separate validation set or cross-validation.

#### **How It Works**

Random Forest builds each decision tree using a **bootstrap sample** of the training data — that is, sampling with replacement. On average, about **63%** of the original data is used to train each tree, leaving roughly **37% of the data unused** for that tree. These unused instances are called the **"out-of-bag" samples**.

For each data point, the model:

* Aggregates predictions from all trees **where the data point was out-of-bag**
* Compares this aggregated prediction to the true label
* Computes the accuracy over all such predictions

This provides a robust estimate of the model's generalization ability, **without needing to hold out a validation set**.

#### **Why It’s Useful**

* Efficient: No need to reduce training data size for validation
* Fast: Evaluated during training, without extra loops
* Reliable: Closely approximates cross-validation accuracy when used correctly

#### **When to Be Cautious**

* A **very high OOB score** (e.g., >0.99) may suggest:
  + True clusters or classes are extremely well-separated
  + Or, potential overfitting if the model is memorizing patterns too well

This is why it's important to **compare OOB score with cross-validation accuracy** and **test set performance** to ensure the model is not overfitting.

Training and evaluation were performed using an 80/20 stratified train-test split.

**\*Model Validation and Overfitting Concern**

The OOB score was extremely high (~0.9978), raising initial concerns of overfitting. However, this suspicion was mitigated by the following checks:



5-fold Cross-Validation also yielded a similarly high accuracy (~0.9968), suggesting that performance generalizes well.

- Test accuracy closely matched the OOB score.

- Confusion matrix showed minimal misclassifications.

- PCA visualization demonstrated clear cluster boundaries.

- Feature importances revealed interpretable and sensible predictors.

Hence, we conclude that the high accuracy is not due to overfitting, but rather due to well-separated clusters defined in high-dimensional feature space.

도표, 그래프, 라인, 다채로움이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

**\*Visualization**

We used the following visual diagnostics:

* + PCA-reduced scatter plot of clusters
  + Confusion matrix heatmap
  + Bar chart of top 10 feature importances
  + Comparison of OOB score and test set accuracy

These visualizations helped verify both performance and generalization.

D. Conclusion

The pipeline successfully combines clustering and classification to automatically label new listings based on learned patterns. Though high accuracy initially appeared suspicious, thorough validation supports the model’s generalizability. This setup can be extended to automate Airbnb listing categorization, assisting in dynamic pricing, recommendation, and marketing.

As example, we made random test data set that includes 20 accommodations. Classification was conducted with the test dataset that was created to check whether our model was operating normally, and each data was output to which cluster and visualized on a map. The results are below.

텍스트, 메뉴, 스크린샷, 폰트이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

\* Latitude and Longitude were only used to plot the results.

지도, 텍스트, 아틀라스이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

**Ⅴ.** Host

\* Omitted descriptions overlapping with user parts

A. Data Preprocessing

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

np.random.seed(42)

# -----------------------/// Data Preprocessing ///-----------------------

# Haversine distance function to compute distance between two geo-locations

def haversine(lat1, lon1, lat2, lon2):

    R = 6371  # Earth radius in km

    φ1, φ2 = np.radians(lat1), np.radians(lat2)

    Δφ = φ2 - φ1

    Δλ = np.radians(lon2 - lon1)

    a = np.sin(Δφ/2)\*\*2 + np.cos(φ1)\*np.cos(φ2)\*np.sin(Δλ/2)\*\*2

    c = 2 \* np.arctan2(np.sqrt(a), np.sqrt(1-a))

    return R \* c

# Drop unnecessary columns

df.drop(columns=['id', 'name', 'host\_id', 'host\_name', 'neighbourhood'], inplace=True)

# One-hot encode neighbourhood group

df = pd.get\_dummies(df, columns=['neighbourhood\_group'])

dummy\_cols = [col for col in df.columns if col.startswith('neighbourhood\_group\_')]

# Define geographical centers for each neighbourhood group

centers = {

    'neighbourhood\_group\_Bronx': (40.8448, -73.8648),

    'neighbourhood\_group\_Brooklyn': (40.6782, -73.9442),

    'neighbourhood\_group\_Manhattan': (40.7685, -73.9822),

    'neighbourhood\_group\_Queens': (40.7282, -73.7949),

    'neighbourhood\_group\_Staten Island': (40.5795, -74.1502),

}

# Compute distance from each listing to the center of its neighbourhood group

def compute\_distance(row):

    for col in dummy\_cols:

        if row.get(col, False):

            clat, clon = centers[col]

            return haversine(row['latitude'], row['longitude'], clat, clon)

    return np.nan

df['distance\_to\_center'] = df.apply(compute\_distance, axis=1)

# Process review dates to compute how recent they are

reference\_date = pd.to\_datetime("2019-12-01")

df['last\_review'] = pd.to\_datetime(df['last\_review'], errors='coerce')

df['days\_since\_oldest\_review'] = (reference\_date - df['last\_review']).dt.days

# Handle missing review dates by assigning max + 30 days

temp\_days = df['days\_since\_oldest\_review'].copy()

df['days\_since\_oldest\_review'] = temp\_days.fillna(temp\_days.max() + 30)

# Invert values so that more recent reviews have higher numbers

max\_days = df['days\_since\_oldest\_review'].max()

df['days\_since\_oldest\_review'] = max\_days - df['days\_since\_oldest\_review']

df.drop(columns=['last\_review'], inplace=True)

# Fill missing values in review frequency and one-hot encode room\_type

df['reviews\_per\_month'].fillna(0, inplace=True)

df = pd.get\_dummies(df, columns=['room\_type'])

# Drop latitude and longitude columns

df.drop(columns=['latitude', 'longitude'], inplace=True)

# Remove price outliers

df = df[df['price'] > 0]

df = df[df['price'] < 2000]

# Log-transform price to reduce skewness

df['log\_price'] = np.log1p(df['price'])

df.drop(columns=['price'], inplace=True)

# Remove outliers in minimum\_nights

df = df[df['minimum\_nights'] >= 1]

df = df[df['minimum\_nights'] <= 30]

**- Column Elimination**: Removed irrelevant columns (e.g., ID, host name, neighborhood) to reduce noise.

- **Geographical Feature**: Created distance\_to\_center using one-hot encoded neighbourhood\_group and the Haversine formula.

- **Review Recency**: Transformed last\_review into days\_since\_oldest\_review, representing how recent a review is.

- **Missing Values**: Filled missing reviews\_per\_month with 0, assuming inactivity.

- **Room Type**: One-hot encoded room\_type to represent lodging types.

- **Coordinates**: Dropped latitude and longitude after calculating distance.

- **Price Handling**: Removed extreme outliers (0 or ≥2000 USD) and applied log transformation for normalization.

- **Minimum Nights**: Kept listings with 1–30 nights to focus on typical stays.

# Select all columns for scaling

features\_to\_scale = df.columns.tolist()

# Apply standard scaling

scaler = StandardScaler()

scaled\_array = scaler.fit\_transform(df[features\_to\_scale].values)

# Rebuild DataFrame with scaled values

df\_std = pd.DataFrame(scaled\_array, columns=features\_to\_scale, index=df.index)

print(df\_std.head())

* Standard Scaling before Clustering

# -----------------------/// KMeans Clustering ///-----------------------

from sklearn.cluster import KMeans

df\_standard = df\_std.copy()

X\_for\_clustering = df\_standard

sse = []

# Run KMeans for k from 1 to 10 and record SSE (inertia)

for k in range(1, 11):

    kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init='auto')

    kmeans.fit(X\_for\_clustering)

    sse.append(kmeans.inertia\_)

# Plot Elbow method

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

plt.plot(range(1, 11), sse, marker='o')

plt.title('Elbow Method For Optimal k')

plt.xlabel('Number of clusters (k)')

plt.ylabel('SSE (Inertia)')

plt.grid(True)

plt.show()

K-Means clustering was applied with k ranging from 1 to 10. The Sum of Squared Errors (SSE) for each k was recorded and visualized using the Elbow Method to identify the optimal number of clusters.

라인, 텍스트, 도표, 그래프이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* When you look at the number of clusters up to 6 (not to make the number of clusters too high), it is elbow when there are 4 clusters.

# Final KMeans clustering with k=4

kmeans = KMeans(n\_clusters=4, random\_state=42, n\_init='auto')

df\_standard['cluster'] = kmeans.fit\_predict(X\_for\_clustering)

# View cluster centroids

cluster\_means = df\_standard.groupby('cluster').mean(numeric\_only=True)

print("\n--- Cluster Means ---")

print(cluster\_means)

# Merge cluster info back to original df

df['cluster'] = df\_standard['cluster']

df.head()

K-Means Clustering with k=4

B. Data Analysis

# Standard Scaling

# -----------------------/// Feature/Target Split and Scaling ///-----------------------

X = df.drop(columns=['log\_price'])

y = df['log\_price']

# Scale features for linear regression

scaler = StandardScaler()

X\_scaled\_array = scaler.fit\_transform(X)

X\_scaled = pd.DataFrame(X\_scaled\_array, columns=X.columns, index=X.index)

print("Scaled feature data:")

print(X\_scaled.head())

print("Target (log\_price) data:")

print(y.head())

print("Feature shape:", X\_scaled.shape)

print("Target shape:", y.shape)

# Linear Regression Modeling with scaled data

# -----------------------/// Linear Regression Modeling ///-----------------------

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# Train-test split

X\_train\_lr, X\_test\_lr, y\_train\_lr, y\_test\_lr = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

print(f"\nTraining data shape: X\_train\_lr={X\_train\_lr.shape}, y\_train\_lr={y\_train\_lr.shape}")

print(f"Test data shape: X\_test\_lr={X\_test\_lr.shape}, y\_test\_lr={y\_test\_lr.shape}")

print("\n--- Linear Regression Model Training and Evaluation ---")

# Train linear regression model

lr\_model = LinearRegression()

lr\_model.fit(X\_train\_lr, y\_train\_lr)

# Predict on test set

y\_pred\_log\_lr = lr\_model.predict(X\_test\_lr)

# Evaluate in log scale

mse\_log\_lr = mean\_squared\_error(y\_test\_lr, y\_pred\_log\_lr)

rmse\_log\_lr = np.sqrt(mse\_log\_lr)

mae\_log\_lr = mean\_absolute\_error(y\_test\_lr, y\_pred\_log\_lr)

r2\_log\_lr = r2\_score(y\_test\_lr, y\_pred\_log\_lr)

print(f"Test RMSE (log\_price): {rmse\_log\_lr:.4f}")

print(f"Test MAE (log\_price): {mae\_log\_lr:.4f}")

print(f"Test R-squared (log\_price): {r2\_log\_lr:.4f}")

# Evaluate in original scale

y\_test\_original = np.expm1(y\_test\_lr)

y\_pred\_original\_lr = np.expm1(y\_pred\_log\_lr)

y\_pred\_original\_lr[y\_pred\_original\_lr < 0] = 0

mse\_original\_lr = mean\_squared\_error(y\_test\_original, y\_pred\_original\_lr)

rmse\_original\_lr = np.sqrt(mse\_original\_lr)

mae\_original\_lr = mean\_absolute\_error(y\_test\_original, y\_pred\_original\_lr)

print(f"\nTest RMSE (original price): ${rmse\_original\_lr:.2f}")

print(f"Test MAE (original price): ${mae\_original\_lr:.2f}")

텍스트, 폰트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

* R-squared : an indicator of how much an independent variable describes a dependent variable in a regression model. Also called explanatory power

# Plot actual vs predicted (original scale)

plt.figure(figsize=(10, 6))

plt.scatter(y\_test\_original, y\_pred\_original\_lr, alpha=0.3, label='Predicted vs Actual')

min\_val = min(y\_test\_original.min(), y\_pred\_original\_lr.min())

max\_val = max(y\_test\_original.max(), y\_pred\_original\_lr.max())

plt.plot([min\_val, max\_val], [min\_val, max\_val], 'r--', lw=2, label='Perfect Prediction Line')

plt.xlabel("Actual Price ($)")

plt.ylabel("Predicted Price ($) - Linear Regression")

plt.title("Actual vs. Predicted Prices (Linear Regression) - Original Scale")

plt.legend()

plt.grid(True)

plt.show()

텍스트, 그래프, 라인, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

# Plot residuals

residuals\_lr = y\_test\_original - y\_pred\_original\_lr

plt.figure(figsize=(10, 6))

plt.scatter(y\_pred\_original\_lr, residuals\_lr, alpha=0.3)

plt.axhline(y=0, color='r', linestyle='--')

plt.xlabel("Predicted Price ($) - Linear Regression")

plt.ylabel("Residuals (Actual - Predicted Price) ($)")

plt.title("Residual Plot (Linear Regression) - Original Scale")

plt.grid(True)

plt.show()

텍스트, 도표, 그래프, 라인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

C. Data Evaluation

Using XGBoost

# -----------------------/// XGBoost Modeling and Evaluation ///-----------------------

from sklearn.model\_selection import RandomizedSearchCV

import xgboost as xgb

# Train-test split for XGBoost (no scaling required)

X\_train\_xgb, X\_test\_xgb, y\_train\_xgb, y\_test\_xgb = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Define hyperparameter grid for XGBoost

param\_distributions\_xgb = {

    'n\_estimators': [int(x) for x in np.linspace(start=100, stop=1000, num=10)],

    'learning\_rate': [0.01, 0.05, 0.1, 0.15, 0.2],

    'max\_depth': [3, 4, 5, 6, 7, 8],

    'min\_child\_weight': [1, 3, 5, 7],

    'subsample': [0.6, 0.7, 0.8, 0.9, 1.0],

    'colsample\_bytree': [0.6, 0.7, 0.8, 0.9, 1.0],

    'gamma': [0, 0.1, 0.2, 0.3],

    'reg\_alpha': [0, 0.001, 0.01, 0.1],

    'reg\_lambda': [0.1, 0.5, 1, 1.5, 2]

}

# Initialize XGBoost model

xgb\_base = xgb.XGBRegressor(objective='reg:squarederror', random\_state=42, n\_jobs=-1)

print(f"\n--- Hyperparameter Tuning with RandomizedSearchCV for XGBoost ---")

# Perform randomized hyperparameter search

random\_search\_xgb = RandomizedSearchCV(estimator=xgb\_base,

                                       param\_distributions=param\_distributions\_xgb,

                                       n\_iter=50,

                                       cv=3,

                                       scoring='neg\_mean\_squared\_error',

                                       verbose=2,

                                       random\_state=42,

                                       n\_jobs=-1)

random\_search\_xgb.fit(X\_train\_xgb, y\_train\_xgb)

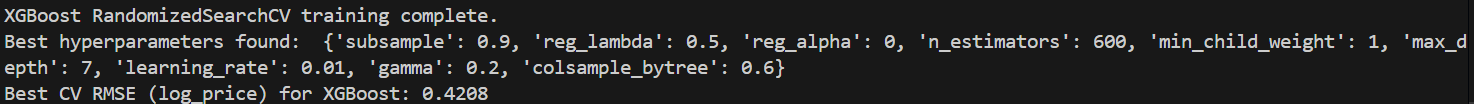
print("\nXGBoost RandomizedSearchCV training complete.")

print("Best hyperparameters found: ", random\_search\_xgb.best\_params\_)

best\_cv\_rmse\_xgb = np.sqrt(-random\_search\_xgb.best\_score\_)

print(f"Best CV RMSE (log\_price) for XGBoost: {best\_cv\_rmse\_xgb:.4f}")

Perform randomized hyperparameter search and find best hyperparameters.



# Evaluate best XGBoost model

best\_xgb\_model = random\_search\_xgb.best\_estimator\_

y\_pred\_log\_xgb\_tuned = best\_xgb\_model.predict(X\_test\_xgb)

mse\_log\_xgb\_tuned = mean\_squared\_error(y\_test\_xgb, y\_pred\_log\_xgb\_tuned)

rmse\_log\_xgb\_tuned = np.sqrt(mse\_log\_xgb\_tuned)

mae\_log\_xgb\_tuned = mean\_absolute\_error(y\_test\_xgb, y\_pred\_log\_xgb\_tuned)

r2\_log\_xgb\_tuned = r2\_score(y\_test\_xgb, y\_pred\_log\_xgb\_tuned)

print(f"Test RMSE (log\_price): {rmse\_log\_xgb\_tuned:.4f}")

print(f"Test MAE (log\_price): {mae\_log\_xgb\_tuned:.4f}")

print(f"Test R-squared (log\_price): {r2\_log\_xgb\_tuned:.4f}")

# Evaluate in original scale

y\_test\_original = np.expm1(y\_test\_xgb)

y\_pred\_original\_xgb\_tuned = np.expm1(y\_pred\_log\_xgb\_tuned)

y\_pred\_original\_xgb\_tuned[y\_pred\_original\_xgb\_tuned < 0] = 0

mse\_original\_xgb\_tuned = mean\_squared\_error(y\_test\_original, y\_pred\_original\_xgb\_tuned)

rmse\_original\_xgb\_tuned = np.sqrt(mse\_original\_xgb\_tuned)

mae\_original\_xgb\_tuned = mean\_absolute\_error(y\_test\_original, y\_pred\_original\_xgb\_tuned)

print(f"\nTest RMSE (original price): ${rmse\_original\_xgb\_tuned:.2f}")

print(f"Test MAE (original price): ${mae\_original\_xgb\_tuned:.2f}")

텍스트, 폰트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

- XGBoost outperformed Linear Regression in all key metrics, including RMSE, MAE, and R².

- Notably, R² improved from 0.51 to 0.60, indicating a significant gain in explanatory power.

# Feature importance plot

# Feature importance plot

print("\n--- Feature Importances (Best XGBoost Regressor) ---")

feature\_names\_for\_importance = X\_train\_xgb.columns

importances\_tuned\_xgb = best\_xgb\_model.feature\_importances\_

feature\_importance\_df\_tuned\_xgb = pd.DataFrame({'feature': feature\_names\_for\_importance, 'importance': importances\_tuned\_xgb})

feature\_importance\_df\_tuned\_xgb = feature\_importance\_df\_tuned\_xgb.sort\_values(by='importance', ascending=False)

print("Top 10 Feature Importances (Tuned XGBoost Model):")

print(feature\_importance\_df\_tuned\_xgb.head(10))

plt.figure(figsize=(10, 8))

top\_n\_features = 15

plt.barh(feature\_importance\_df\_tuned\_xgb['feature'][:top\_n\_features], feature\_importance\_df\_tuned\_xgb['importance'][:top\_n\_features])

plt.xlabel("Feature Importance")

plt.ylabel("Feature")

plt.title(f"Top {top\_n\_features} Feature Importances from Tuned XGBoost Regressor")

plt.gca().invert\_yaxis()

plt.tight\_layout()

plt.show()

텍스트, 스크린샷, 번호, 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

- The most influential feature was room\_type\_Entire home/apt, followed by cluster (a custom feature), room\_type\_Private room, and neighbourhood\_group\_Manhattan.

- This indicates that room type and location are key factors affecting Airbnb prices. The result aligns well with real-world expectations — listings that are entire homes and located in Manhattan tend to be more expensive.

# Prediction vs. Actual

# Prediction vs actual

plt.figure(figsize=(10, 6))

plt.scatter(y\_test\_original, y\_pred\_original\_xgb\_tuned, alpha=0.3, label='Predicted vs Actual')

min\_val = min(y\_test\_original.min(), y\_pred\_original\_xgb\_tuned.min())

max\_val = max(y\_test\_original.max(), y\_pred\_original\_xgb\_tuned.max())

plt.plot([min\_val, max\_val], [min\_val, max\_val], 'r--', lw=2, label='Perfect Prediction Line')

plt.xlabel("Actual Price ($)")

plt.ylabel("Predicted Price ($) - Tuned XGBoost")

plt.title("Actual vs. Predicted Prices (Tuned XGBoost Regressor) - Original Scale")

plt.legend()

plt.grid(True)

plt.show()

텍스트, 그래프, 라인, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**- Linear Regression**: Showed signs of underfitting, with predicted values generally lower than actual ones. High-priced listings were severely underpredicted, revealing its limitation in capturing non-linear relationships.

- **XGBoost**: Produced predictions more spread out and closer to actual values, especially for expensive listings. While not perfect, its predictions aligned better with the actual distribution and offered superior overall fit.

# Residual plot

# Residual plot

residuals\_xgb\_tuned = y\_test\_original - y\_pred\_original\_xgb\_tuned

plt.figure(figsize=(10, 6))

plt.scatter(y\_pred\_original\_xgb\_tuned, residuals\_xgb\_tuned, alpha=0.3)

plt.axhline(y=0, color='r', linestyle='--')

plt.xlabel("Predicted Price ($) - Tuned XGBoost")

plt.ylabel("Residuals (Actual - Predicted Price) ($)")

plt.title("Residual Plot (Tuned XGBoost Regressor) - Original Scale")

plt.grid(True)

plt.show()

텍스트, 스크린샷, 그래프, 도표이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

**- Linear Regression**: Residuals were unevenly distributed, with larger errors for high-priced listings, showing heteroscedasticity and frequent underestimation.

- **XGBoost**: Residuals were more stable and less dispersed, especially at higher price levels. Error asymmetry was less pronounced compared to Linear Regression.

D. Conclusion

In this project, we compared the performance of a Linear Regression model and a tuned XGBoost Regressor for predicting Airbnb prices.

The XGBoost model significantly outperformed Linear Regression in all evaluation metrics. It achieved lower RMSE and MAE on both log-transformed and original price scales, and showed a higher R-squared value (0.60 vs. 0.51), indicating better explanatory power.

From the prediction and residual plots, the Linear Regression model consistently underpredicted high-priced listings, showing signs of underfitting. In contrast, the XGBoost model captured the non-linear relationships more effectively and reduced the residual variance, especially for expensive listings.

Feature importance analysis from XGBoost showed that room type and location were the most influential factors affecting price.

Overall, the XGBoost model is more suitable for this task due to its better accuracy, robustness, and ability to capture complex patterns in the data.

Although the XGBoost model performed well, it still showed some error in predicting extreme prices. Future improvements could ensemble methods combining multiple models.