CourseProjectMilestone4_revised

November 16, 2024

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
import re
from sklearn.cluster import KMeans
import seaborn as sns
```

[2]:
```
# load dataset
```

[3]:
```
paris_ab = pd.read_csv("Listing_AirBnB_Paris.csv")
paris_ab
```

[3]:

	id	name	\
0	3109	Rental unit in Paris · 5.0 · 1 bedroom · 1 be…	
1	5396	Rental unit in Paris · 4.59 · Studio · 1 bed …	
2	7397	Rental unit in Paris · 4.73 · 2 bedrooms · 2 …	
3	7964	Rental unit in Paris · 4.80 · 1 bedroom · 1 b…	
4	9359	Rental unit in Paris · 1 bedroom · 1 bed · 1 bath	
…	…	…	
74324	1043629451440755792	Rental unit in Paris · New · Studio · 1 bed ·…	
74325	1043707020977346344	Rental unit in Paris · New · 1 bedroom · 1 be…	
74326	1043932119757241230	Rental unit in Paris · New · 3 bedrooms · 4 b…	
74327	1043947326757240041	Rental unit in Paris · New · 1 bedroom · 2 be…	
74328	1043968453109441641	Rental unit in Paris · New · 2 bedrooms · 3 b…	

	host_id	host_name	neighbourhood_group	neighbourhood	\
0	3631	Anne	NaN	Observatoire	
1	7903	Borzou	NaN	Hôtel-de-Ville	
2	2626	Franck	NaN	Hôtel-de-Ville	
3	22155	Anaïs	NaN	Opéra	
4	28422	Bernadette	NaN	Louvre	
…	…	…	…	…	
74324	50308796	Anais	NaN	Ménilmontant	
74325	18385602	Stanislas	NaN	Buttes-Montmartre	
74326	335998296	Studioprestige	NaN	Entrepôt	
74327	503331047	John	NaN	Élysée	
74328	533054106	Helene	NaN	Temple	

```
        latitude  longitude        room_type  price  minimum_nights  \
0      48.831910   2.318700  Entire home/apt  150.0               2
1      48.852470   2.358350  Entire home/apt  146.0               1
2      48.859090   2.353150  Entire home/apt  140.0              10
3      48.874170   2.342450  Entire home/apt  180.0               7
4      48.860060   2.348630  Entire home/apt   75.0             180
…           …         …               …    …              …
74324  48.868059   2.407307  Entire home/apt   52.0               1
74325  48.891719   2.335972  Entire home/apt  500.0               1
74326  48.874650   2.355466  Entire home/apt  324.0               1
74327  48.869724   2.318358  Entire home/apt   85.0               1
74328  48.868092   2.359397  Entire home/apt  190.0               1

       number_of_reviews last_review  reviews_per_month  \
0                      4  2019-10-24               0.05
1                    374  2023-12-11               2.12
2                    343  2023-11-16               2.22
3                      5  2015-09-14               0.03
4                      0         NaN                NaN
…                    …          …                 …
74324                  0         NaN                NaN
74325                  0         NaN                NaN
74326                  0         NaN                NaN
74327                  0         NaN                NaN
74328                  0         NaN                NaN

       calculated_host_listings_count  availability_365  \
0                                   1               327
1                                   2                 0
2                                   7               198
3                                   1                25
4                                   1               185
…                                 …               …
74324                               1               100
74325                               1                27
74326                             121               362
74327                              30               290
74328                               9               364

       number_of_reviews_ltm  \
0                          0
1                         48
2                         22
3                          0
4                          0
…                        …
```

```
74324                       0
74325                       0
74326                       0
74327                       0
74328                       0

                                                    license
0                                             7511409139079
1                                             7510402838018
2                                             7510400829623
3                                             7510903576564
4      Available with a mobility lease only ("bail mo…
…                                                       …
74324                                         7512010784549
74325                                         7511810784902
74326                                         7511505605678
74327                                         7510810007394
74328                                         7511010785617

[74329 rows x 18 columns]
```

[4]:
```
paris_ab.shape
```

[4]:
```
(74329, 18)
```

[5]:
```
paris_ab['room_type'].unique()
```

[5]:
```
array(['Entire home/apt', 'Private room', 'Shared room', 'Hotel room'],
      dtype=object)
```

[6]:
```
paris_ab['neighbourhood'].unique()
```

[6]:
```
array(['Observatoire', 'Hôtel-de-Ville', 'Opéra', 'Louvre', 'Popincourt',
       'Buttes-Montmartre', 'Entrepôt', 'Vaugirard', 'Luxembourg',
       'Gobelins', 'Bourse', 'Buttes-Chaumont', 'Temple', 'Reuilly',
       'Élysée', 'Panthéon', 'Batignolles-Monceau', 'Ménilmontant',
       'Palais-Bourbon', 'Passy'], dtype=object)
```

[7]:
```
paris_ab.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74329 entries, 0 to 74328
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   id                              74329 non-null  int64
 1   name                            74329 non-null  object
 2   host_id                         74329 non-null  int64
```

```
 3   host_name                       74320 non-null  object
 4   neighbourhood_group             0 non-null      float64
 5   neighbourhood                   74329 non-null  object
 6   latitude                        74329 non-null  float64
 7   longitude                       74329 non-null  float64
 8   room_type                       74329 non-null  object
 9   price                           67108 non-null  float64
 10  minimum_nights                  74329 non-null  int64
 11  number_of_reviews               74329 non-null  int64
 12  last_review                     56438 non-null  object
 13  reviews_per_month               56438 non-null  float64
 14  calculated_host_listings_count  74329 non-null  int64
 15  availability_365                74329 non-null  int64
 16  number_of_reviews_ltm           74329 non-null  int64
 17  license                         54687 non-null  object
dtypes: float64(5), int64(7), object(6)
memory usage: 10.2+ MB
```

[8]:
```
# check null values
```

[9]:
```
paris_ab.isnull().sum()
```

[9]:
```
id                                  0
name                                0
host_id                             0
host_name                           9
neighbourhood_group             74329
neighbourhood                       0
latitude                            0
longitude                           0
room_type                           0
price                            7221
minimum_nights                      0
number_of_reviews                   0
last_review                     17891
reviews_per_month               17891
calculated_host_listings_count      0
availability_365                    0
number_of_reviews_ltm               0
license                         19642
dtype: int64
```

[10]:
```
# data cleaning
```

[11]:
```
paris_ab_df = paris_ab.drop(columns = ['id', 'host_id', 'host_name',␣
 ↪'neighbourhood_group', 'latitude', 'longitude', 'last_review',␣
 ↪'calculated_host_listings_count', 'number_of_reviews_ltm', 'license'])
paris_ab_df.head()
```

[11]:
```
                                                name   neighbourhood  \
0  Rental unit in Paris · 5.0 · 1 bedroom · 1 be…    Observatoire
1  Rental unit in Paris · 4.59 · Studio · 1 bed …  Hôtel-de-Ville
2  Rental unit in Paris · 4.73 · 2 bedrooms · 2 …  Hôtel-de-Ville
3  Rental unit in Paris · 4.80 · 1 bedroom · 1 b…           Opéra
4  Rental unit in Paris · 1 bedroom · 1 bed · 1 bath          Louvre

         room_type  price  minimum_nights  number_of_reviews  \
0  Entire home/apt  150.0               2                  4
1  Entire home/apt  146.0               1                374
2  Entire home/apt  140.0              10                343
3  Entire home/apt  180.0               7                  5
4  Entire home/apt   75.0             180                  0

   reviews_per_month  availability_365
0               0.05               327
1               2.12                 0
2               2.22               198
3               0.03                25
4                NaN               185
```

[12]:
```
paris_ab_df.isnull().sum()
```

[12]:
```
name                    0
neighbourhood           0
room_type               0
price                7221
minimum_nights          0
number_of_reviews       0
reviews_per_month   17891
availability_365        0
dtype: int64
```

[13]:
```
# drop missing values
```

[14]:
```
paris_ab_df.dropna(inplace = True)
```

[15]:
```
# extract review score from 'name' column
```

[16]:
```
def extract_review_score(name):
    match = re.search(r' (\d+\.\d+)', name)
    if match:
        return float(match.group(1))
    return None
```

[17]:
```
paris_ab_df['review_score'] = paris_ab_df['name'].apply(extract_review_score)
```

[18]:
```
def extract_bedrooms(name):
    match = re.search(r'(\d+)\s*bedroom', name, re.IGNORECASE)
    if match:
        return int(match.group(1))
    return None

def extract_beds(name):
    match = re.search(r'(\d+)\s*bed', name, re.IGNORECASE)
    if match:
        return int(match.group(1))
    return None

def extract_bathrooms(name):
    match = re.search(r'(\d+)\s*bath', name, re.IGNORECASE)
    if match:
        return int(match.group(1))
    return None
```

[19]:
```
paris_ab_df['bedroom'] = paris_ab_df['name'].apply(extract_bedrooms)
paris_ab_df['bed'] = paris_ab_df['name'].apply(extract_beds)
paris_ab_df['bathroom'] = paris_ab_df['name'].apply(extract_bathrooms)
```

[20]:
```
paris_ab_df.head()
```

[20]:

	name	neighbourhood	room_type	price	minimum_nights	number_of_reviews	reviews_per_month	availability_365	review_score	bedroom	bed	bathroom
0	Rental unit in Paris · 5.0 · 1 bedroom · 1 be…	Observatoire	Entire home/apt	150.0	2	4	0.05	327	5.00	1.0	1.0	1.0
1	Rental unit in Paris · 4.59 · Studio · 1 bed …	Hôtel-de-Ville	Entire home/apt	146.0	1	374	2.12	0	4.59	NaN	1.0	1.0
2	Rental unit in Paris · 4.73 · 2 bedrooms · 2 …	Hôtel-de-Ville	Entire home/apt	140.0	10	343	2.22	198	4.73	2.0	2.0	1.0
3	Rental unit in Paris · 4.80 · 1 bedroom · 1 b…	Opéra	Entire home/apt	180.0	7	5	0.03	25	4.80	1.0	1.0	1.0
5	Rental unit in Paris · 4.92 · 1 bedroom · 1 b…	Popincourt	Entire home/apt	130.0	4	49	0.37	169	4.92	1.0	1.0	1.0

[21]:
```
# paris_ab_df.isnull().sum()
```

[22]:
```
paris_ab_df['bedroom'].fillna(0, inplace=True)
```

```
/var/folders/6q/k8jdwbv174s78xj3x3kzvn0w0000gn/T/ipykernel_13844/787842860.py:1:
FutureWarning: A value is trying to be set on a copy of a DataFrame or Series
through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

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


  paris_ab_df['bedroom'].fillna(0, inplace=True)
```

[23]:
```
paris_ab_df_new = paris_ab_df.drop(columns = ['name'])
```

[24]:
```
paris_ab_df_new.head()
```

[24]:

	neighbourhood	room_type	price	minimum_nights	number_of_reviews	\
0	Observatoire	Entire home/apt	150.0	2	4	
1	Hôtel-de-Ville	Entire home/apt	146.0	1	374	
2	Hôtel-de-Ville	Entire home/apt	140.0	10	343	
3	Opéra	Entire home/apt	180.0	7	5	
5	Popincourt	Entire home/apt	130.0	4	49	

	reviews_per_month	availability_365	review_score	bedroom	bed	bathroom
0	0.05	327	5.00	1.0	1.0	1.0
1	2.12	0	4.59	0.0	1.0	1.0
2	2.22	198	4.73	2.0	2.0	1.0
3	0.03	25	4.80	1.0	1.0	1.0
5	0.37	169	4.92	1.0	1.0	1.0

[25]:
```
paris_ab_df_new.isnull().sum()
```

[25]:
```
neighbourhood           0
room_type               0
price                   0
minimum_nights          0
number_of_reviews       0
reviews_per_month       0
availability_365        0
review_score         8858
bedroom                 0
bed                   108
bathroom             5312
dtype: int64
```

[26]:
```
paris_ab_df_new.dropna(inplace = True)
```

[27]:
```
paris_ab_df_new.shape
```

[27]:
```
(39196, 11)
```

[113]:
```
plt.figure(figsize=(10, 6))
plt.boxplot(paris_ab_df['price'], vert=False, patch_artist=True,␣
 ↪boxprops=dict(facecolor='skyblue', color='black'))
plt.title('Price Distribution of Airbnb Listings in Paris', fontsize=16)
plt.xlabel('Price (in €)', fontsize=14)
plt.grid(axis='x', alpha=0.5)
plt.show()
```

From the price distribution graph, the majority of listings have prices clustered towards the lower end, with a few higher-priced outliers. Almost all of the airbnb price in Paris are below €1,0000, with most below about €1,000.

[ ]:
```
# view price distribution based on room types
```

[114]:
```
plt.figure(figsize=(10, 6))
plt.scatter(paris_ab_df['room_type'], paris_ab_df['price'],
                      alpha=0.6, color='skyblue', edgecolor='black')
```

```
plt.title('Price Distribution of Airbnb Listings in Paris by Room Type',␣
 ↪fontsize=16)
plt.xlabel('Room Type', fontsize=14)
plt.ylabel('Price (in €)', fontsize=14)
plt.xticks(rotation=45)
plt.grid(True, alpha=0.5)
plt.show()
```

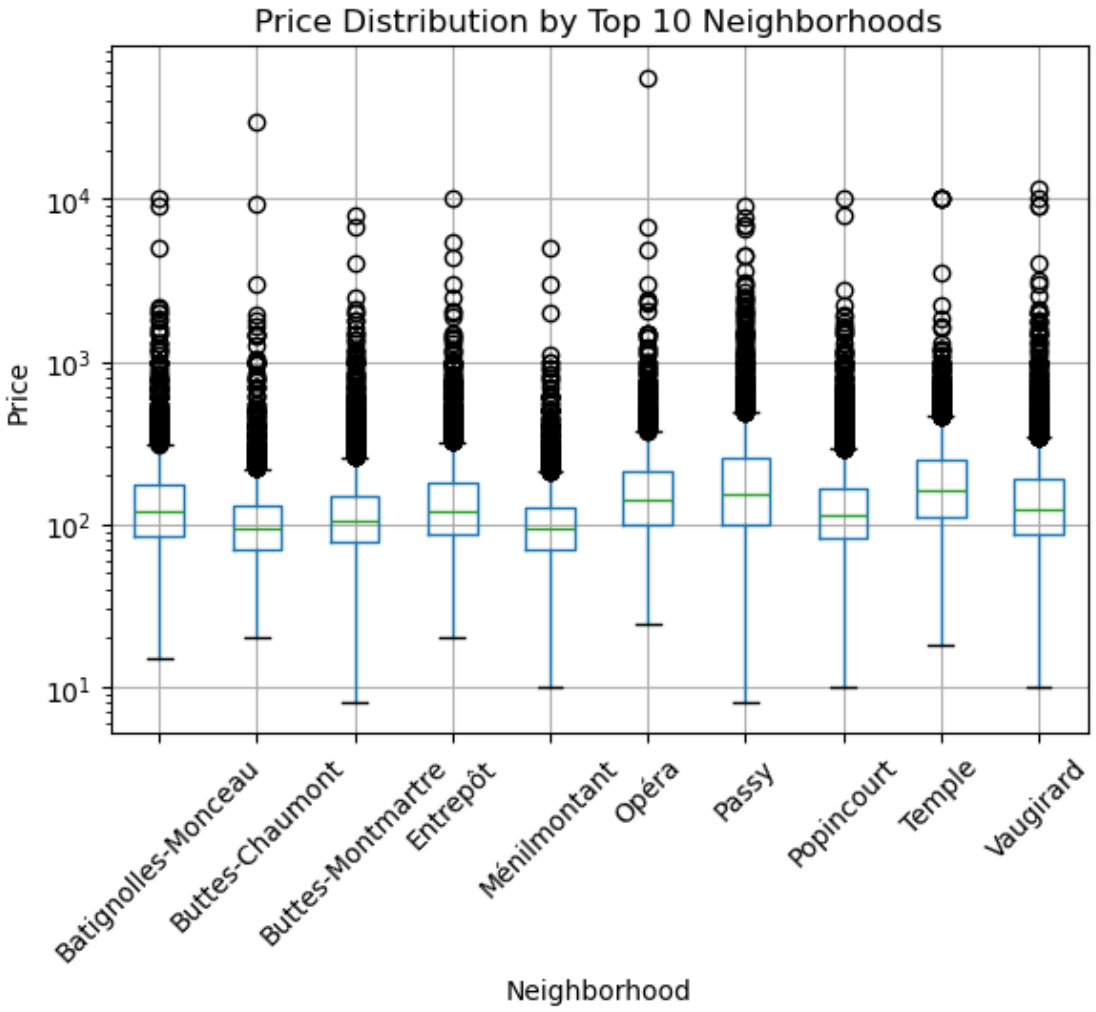
The plot shows that "Entire home/apt" listings tend to have a broader price range, which ranges from €0-10,000, with many listings at the lower end but also some higher-priced listings. "Private rooms" generally have lower prices, and "Shared rooms" have the lowest price range.

[115]:
```
# view price distribution in top 10 neighborhood.
```

[116]:
```
top_neighbourhoods = paris_ab_df['neighbourhood'].value_counts().head(10).index
plt.figure(figsize=(12, 6))
paris_ab_df[paris_ab_df['neighbourhood'].isin(top_neighbourhoods)].
 ↪boxplot(column='price', by='neighbourhood', rot=45)
plt.title('Price Distribution by Top 10 Neighborhoods')
plt.suptitle('')  # Remove default title
plt.xlabel('Neighborhood')
plt.ylabel('Price')
```

```
plt.yscale('log')  # Use log scale for better visualization
plt.grid(True)
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```

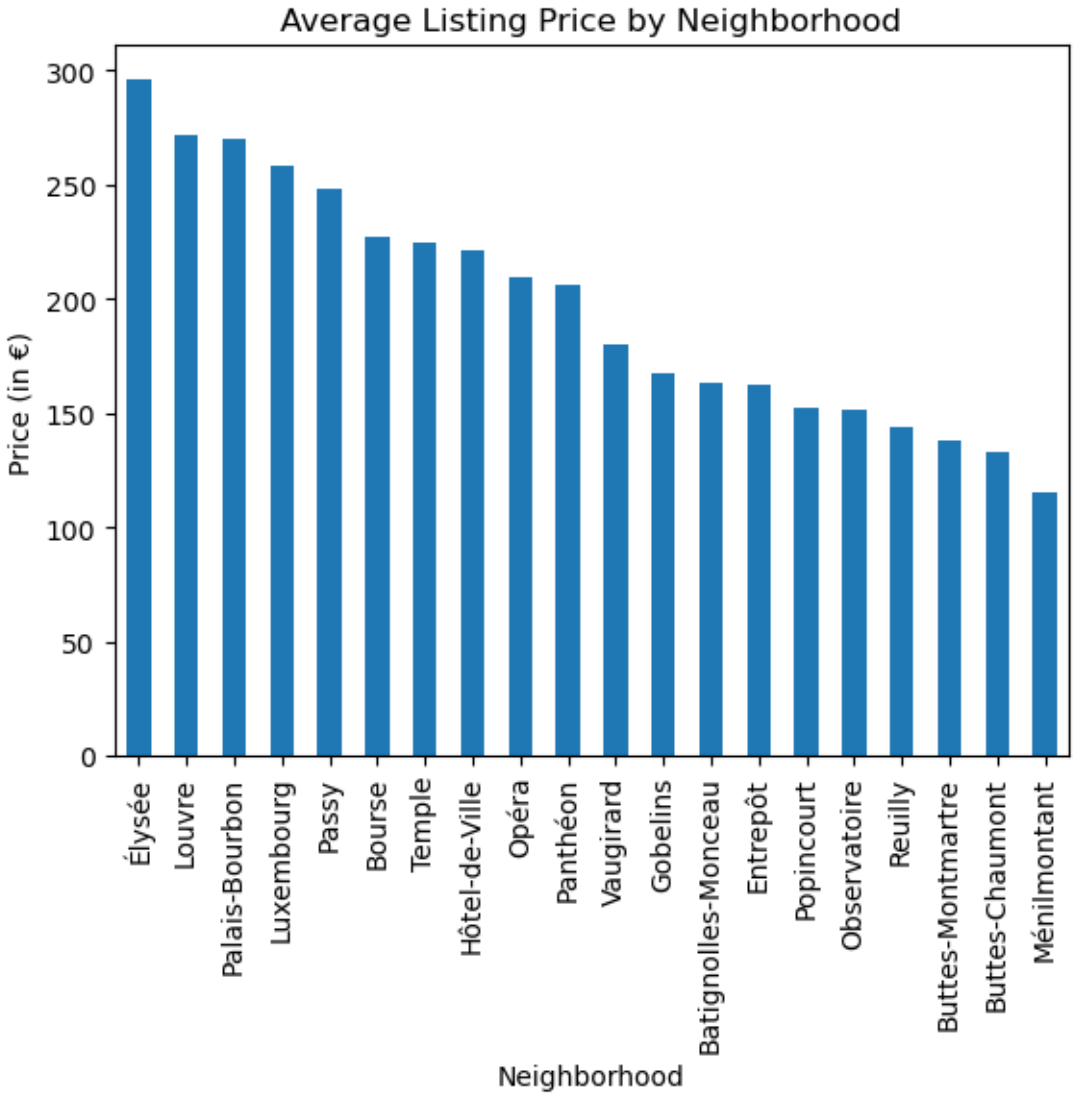


The box plot illustrates the variation in listing prices across the top 10 neighborhoods. Most of the neighborhoods have a wider range of prices, indicating both budget and luxury listings, while others are more consistent in pricing. The analysis reveals that Passy has a relatively higher average price compared to several other top neighborhoods.

[42]:
```
# view average price in different neighborhood
```

[117]:
```
neighbourhood_price = paris_ab_df.groupby("neighbourhood")["price"].mean().
↪sort_values(ascending=False)
neighbourhood_price.plot(kind="bar")
```

```
plt.title("Average Listing Price by Neighborhood")
plt.xlabel("Neighborhood")
plt.ylabel("Price (in €)")
plt.show()
```



The bar plot illustrates the average listing prices across all the neighborhoods. while others are more consistent in pricing. The log scale helps to better observe the spread and outliers.

[29]:
```
# detect and handle outlier
```

[30]:
```
paris_ab_df_new['price'].describe(percentiles=[0.75, 0.90, 0.95, 0.99])
```

[30]:
```
count    39196.000000
mean       186.197979
std        400.750091
min         10.000000
50%        130.000000
75%        200.000000
90%        320.000000
95%        443.000000
99%        941.000000
max      53239.000000
Name: price, dtype: float64
```

[31]:
```
upper_limit = paris_ab_df_new['price'].quantile(0.995)
```

[32]:
```
upper_limit
```

[32]:
```
1372.0250000000015
```

[33]:
```
q1 = paris_ab_df_new['price'].quantile(0.25)
```

[34]:
```
q3 = paris_ab_df_new['price'].quantile(0.75)
```

[35]:
```
iqr = q3 - q1
lower_bound = max(0, q1 - 1.5 * iqr)
upper_bound = q3 + 1.5 * iqr
paris_ab_df_new = paris_ab_df_new[(paris_ab_df_new['price'] >= lower_bound) &␣
 ↪(paris_ab_df_new['price'] <= upper_bound)]
```

[36]:
```
lower_bound, upper_bound
```

[36]:
```
(0, 365.0)
```

[44]:
```
# encode categorical data
```

[45]:
```
paris_ab_encoded = pd.get_dummies(paris_ab_df_new, columns=['room_type',␣
 ↪'neighbourhood'], drop_first=True)
```

[46]:
```
paris_ab_encoded.head()
```

[46]:

	price	minimum_nights	number_of_reviews	reviews_per_month	availability_365	review_score	bedroom	bed	bathroom
0	150.0	2	4	0.05	327	5.00	1.0	1.0	1.0
1	146.0	1	374	2.12					
2	140.0	10	343	2.22					
3	180.0	7	5	0.03					
5	130.0	4	49	0.37					

```
1                0          4.59      0.0  1.0       1.0
2              198          4.73      2.0  2.0       1.0
3               25          4.80      1.0  1.0       1.0
5              169          4.92      1.0  1.0       1.0

   room_type_Hotel room  …  neighbourhood_Observatoire  neighbourhood_Opéra  \
0                 False  …                        True                False
1                 False  …                       False                False
2                 False  …                       False                False
3                 False  …                       False                 True
5                 False  …                       False                False

   neighbourhood_Palais-Bourbon  neighbourhood_Panthéon  neighbourhood_Passy  \
0                         False                   False                False
1                         False                   False                False
2                         False                   False                False
3                         False                   False                False
5                         False                   False                False

   neighbourhood_Popincourt  neighbourhood_Reuilly  neighbourhood_Temple  \
0                     False                  False                 False
1                     False                  False                 False
2                     False                  False                 False
3                     False                  False                 False
5                      True                  False                 False

   neighbourhood_Vaugirard  neighbourhood_Élysée
0                    False                 False
1                    False                 False
2                    False                 False
3                    False                 False
5                    False                 False

[5 rows x 30 columns]
```

[118]:
```
paris_ab_encoded.shape
```

[118]:
```
(36244, 30)
```

[119]:
```
# train model
```

[120]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import StackingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RandomizedSearchCV
```

[121]:
```
# split into train and test model
```

[122]:
```
from sklearn.model_selection import train_test_split
```

[123]:
```
X = paris_ab_encoded.drop(columns=['price'])
y = paris_ab_encoded['price']
```

[124]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[54]:
```
# X_train[numeric_features] = scaler.fit_transform(X_train[numeric_features])
```

[55]:
```
# X_test[numeric_features] = scaler.transform(X_test[numeric_features])
```

[56]:
```
# create a pipeline
```

[68]:
```
from sklearn.pipeline import Pipeline
from sklearn.ensemble import VotingRegressor
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsRegressor
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
```

[71]:
```
numerical_features = ['minimum_nights', 'reviews_per_month', 'availability_365']
```

[125]:
```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features)
    ],
    remainder='passthrough'  # Keep encoded features as is
)
```

[126]:
```
pipeline = Pipeline(steps=[
    ('scaler', StandardScaler()),  # Scale numerical features
    ('pca', PCA(n_components=10)),  # Dimensionality reduction
    ('regressor', RandomForestRegressor(random_state=42))  # Prediction
])
```

[127]:
```
pipeline.fit(X_train, y_train)
```

[127]:
```
Pipeline(steps=[('scaler', StandardScaler()), ('pca', PCA(n_components=10)),
                ('regressor', RandomForestRegressor(random_state=42))])
```

[128]:
```
y_pred = pipeline.predict(X_test)
```

[129]:
```
mae = mean_absolute_error(y_test, y_pred)
rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/_regression.py:483:
FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the
function'root_mean_squared_error'.
  warnings.warn(
```

[130]:
```
mae, rmse, r2
```

[130]:
```
(40.068848011926605, 53.104614173931004, 0.4316873754716427)
```

[131]:
```
# use ensemble techniques
```

[132]:
```
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
gb_model = GradientBoostingRegressor(n_estimators=100, random_state=42)
lr_model = LinearRegression()
```

[80]:
```
voting_model = VotingRegressor(estimators=[
    ('rf', rf_model),
    ('gb', gb_model),
    ('lr', lr_model)
])
```

[81]:
```
pipeline_en = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('voting', voting_model)
])
```

[82]:
```
pipeline_en.fit(X_train, y_train)
```

[82]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('num', StandardScaler(),
                                                  ['minimum_nights',
                                                   'reviews_per_month',
                                                   'availability_365'])])),
                ('voting',
                 VotingRegressor(estimators=[('rf',
RandomForestRegressor(random_state=42)),
                                             ('gb',
```

```
GradientBoostingRegressor(random_state=42)),
                                                ('lr', LinearRegression())]))])
```

[83]:
```
y_pred_en = pipeline_en.predict(X_test)
```

[84]:
```
mae_en = mean_absolute_error(y_test, y_pred_en)
rmse_en = mean_squared_error(y_test, y_pred_en, squared=False)
r2_en = r2_score(y_test, y_pred_en)
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/_regression.py:483:
FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the
function'root_mean_squared_error'.
  warnings.warn(
```

[85]:
```
mae_en, rmse_en, r2_en
```

[85]:
```
(38.156711301183776, 50.16602852293094, 0.49284322751930376)
```

[104]:
```
# use RandomizedSearchCV for hyperparameter tuning
```

[94]:
```
param_dist = {
    'voting__rf__n_estimators': [100, 200],           # Number of trees for␣
 ↪Random Forest
    'voting__rf__max_depth': [None, 10],              # Maximum depth for Random␣
 ↪Forest
    'voting__gb__n_estimators': [50, 100],            # Number of trees for␣
 ↪Gradient Boosting
    'voting__gb__learning_rate': [0.05, 0.1]          # # Learning rate for␣
 ↪Gradient Boosting
}
```

[95]:
```
random_search = RandomizedSearchCV(
    estimator=pipeline_en,
    param_distributions=param_dist,
    n_iter=10,  # Number of random combinations to try
    cv=3,  # Cross-validation folds
    scoring='neg_mean_absolute_error',
    n_jobs=-1,
    random_state=42
)
```

[96]:
```
random_search.fit(X_train, y_train)
```

[96]:
```
RandomizedSearchCV(cv=3,
                   estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(remainder='passthrough',
transformers=[('num',
```

```
                                   StandardScaler(),
                                   ['minimum_nights',
                                   'reviews_per_month',
                                   'availability_365'])])),
                                                          ('voting',
                                                           VotingRegressor(estimators=[('rf',
                                   RandomForestRegressor(random_state=42)),
                                                                                       ('gb',
                                   GradientBoostingRegressor(random_state=42)),
                                                                                       ('lr',
                                   LinearRegression())]))]),
                                   n_jobs=-1,
                                   param_distributions={'voting__gb__learning_rate': [0.05,
                                                                                      0.1],
                                                        'voting__gb__n_estimators': [50, 100],
                                                        'voting__rf__max_depth': [None, 10],
                                                        'voting__rf__n_estimators': [100, 200]},
                                   random_state=42, scoring='neg_mean_absolute_error')
```

[101]:
```
best_parameters = random_search.best_params_
best_parameters
```

[101]:
```
{'voting__rf__n_estimators': 200,
 'voting__rf__max_depth': None,
 'voting__gb__n_estimators': 100,
 'voting__gb__learning_rate': 0.1}
```

[102]:
```
best_MAE = -random_search.best_score_
best_MAE
```

[102]:
```
38.754197455734214
```

[105]:
```
best_pipeline = random_search.best_estimator_
```

[108]:
```
y_pred_rs = best_pipeline.predict(X_test)
```

[109]:
```
mae_rs = mean_absolute_error(y_test, y_pred_rs)
```

[110]:
```
mse_rs = mean_squared_error(y_test, y_pred_rs, squared=False)
```

```
/opt/anaconda3/lib/python3.12/site-packages/sklearn/metrics/_regression.py:483:
FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in
1.6. To calculate the root mean squared error, use the
function'root_mean_squared_error'.
  warnings.warn(
```

[111]:
```
r2_rs = r2_score(y_test, y_pred_rs)
```

[112]:
```
mae_rs, mse_rs, r2_rs
```

[112]:
```
(38.154702343481844, 50.15321946631409, 0.4931021824597511)
```

The output best_parameters is a dictionary containing the optimal hyperparameter values for the VotingRegressor base models that achieved the best performance during cross-validation. The Random Forest model performs best when using 200 trees. The maximum depth of each tree in the Random Forest is restricted to 10, preventing overfitting. The Gradient Boosting model works best with 100 boosting iterations (trees). The learning rate of 0.05 indicates smaller updates per boosting iteration, balancing convergence and generalization.

The output best_MAE represents the lowest mean absolute error (MAE) achieved during cross-validation for the best combination of parameters, which is 38.15. MAE of 38.15 indicate moderate accuracy as most of the price ranges from 130-941. An R^2 of 0.49 means the model explains 49% of the variance in price, suggests the model has moderate predictive power but leaves room for improvement. MSE measures the average squared difference between predictions and actual values. To make it more interpretable, rmse was calculated, which is aproximately 7.07, which gives a sense of the typical prediction error in price as the target variable.

[ ]:

[ ]: