METAD699 - Data Mining for Business Analytics
Professor Christopher Athaide
Group Members: Cassie Williams, Nhat Tran

Semester Project

How do property characteristics and location influence Airbnb prices in Geneva?

Data Preparation & Exploration

Read your data into your local environment.

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

geneva = pd.read_csv("geneva_listings.csv")
```

```
geneva.head()
```

	id	listing_url	scrape_id	last_scraped	source	name	description	neighborhood_ove
0	42515.0	https://www.airbnb.com/rooms/42515	2.023090e+13	9/23/2023	city scrape	Rental unit in Geneva · ★4.73 · 1 bedroom · 1 ...	The space This is a private room w...	
1	107438.0	https://www.airbnb.com/rooms/107438	2.023090e+13	9/23/2023	city scrape	Rental unit in Geneva · ★4.87 · 1 bedroom · 1 ...	The space Version Française et Ang...	
2	203997.0	https://www.airbnb.com/rooms/203997	2.023090e+13	9/23/2023	city scrape	Rental unit in Geneva · ★4.90 · 1 bedroom · 1 ...	Spacious studio with washer/dryer, Wi-fi, supe...	This is the most sough area in Gen
3	276025.0	https://www.airbnb.com/rooms/276025	2.023090e+13	9/23/2023	city scrape	Rental unit in Versoix · ★4.62 · 1 bedroom · 4...	This cozy flat is located in a very pleasant v...	
4	338682.0	https://www.airbnb.com/rooms/338682	2.023090e+13	9/23/2023	city scrape	Rental unit in Geneva · ★4.81 · 1 bedroom · 1 ...	One bedroom apartment located in a pedestrian ...	This is the favourite a expats in G

5 rows × 75 columns

1. Missing Values

Does your data contain any missing values and/or blank cells? If so, what can you do about this?

The dataset shows that most key variables are fully populated, but several fields related to host details and neighborhood information contain substantial missing values (e.g., host_neighbourhood, neighbourhood_group_cleansed, bathrooms, calendar_updated, and license, each missing around 2,400+ entries). In contrast, review-related fields (like review_scores and reviews_per_month) have moderate missingness of about 447 rows, which mostly reflects listings with no review history.

```
pd.set_option("display.max_rows", None)
print(geneva.isnull().sum())
```

```
host_is_superhost                                7
host_thumbnail_url                               0
host_picture_url                                 0
host_neighbourhood                            2422
host_listings_count                              0
host_total_listings_count                        0
host_verifications                               0
host_has_profile_pic                             0
host_identity_verified                           0
neighbourhood                                 1273
neighbourhood_cleansed                           0
neighbourhood_group_cleansed                  2458
latitude                                         0
longitude                                        0
property_type                                    0
room_type                                        0
accommodates                                     0
bathrooms                                     2458
bathrooms_text                                   0
bedrooms                                       842
beds                                            35
amenities                                        0
price                                            0
minimum_nights                                   0
maximum_nights                                   0
minimum_minimum_nights                           0
maximum_minimum_nights                           0
minimum_maximum_nights                           0
maximum_maximum_nights                           0
minimum_nights_avg_ntm                           0
maximum_nights_avg_ntm                           0
calendar_updated                              2458
has_availability                                 0
availability_30                                  0
availability_60                                  0
availability_90                                  0
availability_365                                 0
calendar_last_scraped                            0
number_of_reviews                                0
number_of_reviews_ltm                            0
number_of_reviews_l30d                           0
first_review                                   447
last_review                                    447
review_scores_rating                           447
review_scores_accuracy                         455
review_scores_cleanliness                      455
review_scores_checkin                          455
review_scores_communication                    455
review_scores_location                         455
review_scores_value                            455
license                                       2458
instant_bookable                                 0
calculated_host_listings_count                   0
calculated_host_listings_count_entire_homes      0
calculated_host_listings_count_private_rooms     0
calculated_host_listings_count_shared_rooms      0
reviews_per_month                              447
dtype: int64
```

Dealing with missing values from the dataset

```
# Impute NaN values for description as "No description provided"
geneva['description'] = geneva['description'].fillna("No description provided")

# Impute NaN values for neighborhood_overview as "No neighborhood overview provided"
geneva['neighborhood_overview']= geneva['neighborhood_overview'].fillna("No neighborhood overview provided")

# Impute NaN values for host_location as "Unknown location"
geneva['host_location'] = geneva['host_location'].fillna("Unknown location")

# Impute NaN values for host_about as "No host description provided"
geneva['host_about'] = geneva['host_about'].fillna("No host description provided")

# Fill NaN values for host_response_time as "unknown response time" then impute 0 for NaN values for host_respon
geneva['host_response_time'] = geneva['host_response_time'].fillna("unknown response time")
geneva['host_response_rate'] = geneva['host_response_rate'].fillna("0")
```

```
# Impute misising values in host_acceptance_rate to mode rate of acceptance rate
mode_val = geneva['host_acceptance_rate'].mode()[0]
geneva['host_acceptance_rate'] = geneva['host_acceptance_rate'].fillna(mode_val)

# Fill NaN values for host_is_superhost to 'f'
geneva['host_is_superhost'] = geneva['host_is_superhost'].fillna('f')

# Drop only rows where host_thumbnail_url is NaN
geneva = geneva.dropna(subset=['host_thumbnail_url'])

# Drop repetitive neighborhood columns from the dataset
geneva = geneva.drop(columns=['host_neighbourhood','neighbourhood','neighbourhood_group_cleansed'], errors='igno

# Drop bathrooms column and keep clean bathrooms_text
geneva = geneva.drop(columns=['bathrooms'], errors='ignore')

# Convert beds and bedrooms to numeric variables
geneva['bedrooms'] = pd.to_numeric(geneva['bedrooms'], errors='coerce')
geneva['beds'] = pd.to_numeric(geneva['beds'], errors='coerce')
# Impute NaN bedrooms to be 1 where they have at least 1 bed
geneva['bedrooms'] = np.where(
    geneva['bedrooms'].isnull() & (geneva['beds'] >= 1),
    1,
    geneva['bedrooms']
)
# Drop rows where bedrooms or beds are still NaN
geneva = geneva.dropna(subset=['bedrooms', 'beds'])

# Drop calendar_updated, license columns do not contain any values or have too many missing values (>2200)
geneva = geneva.drop(columns=['calendar_updated', 'license'], errors='ignore')

# Drop redundant columns
geneva = geneva.drop(columns=['minimum_minimum_nights', 'maximum_minimum_nights','minimum_maximum_nights','maxim
```

```
review_cols = [
    'first_review','last_review','review_scores_rating',
    'review_scores_accuracy','review_scores_cleanliness',
    'review_scores_checkin','review_scores_communication',
    'review_scores_location','review_scores_value','reviews_per_month']

# Filter rows where ALL review columns are NaN
all_missing_reviews = geneva[review_cols].isnull().all(axis=1)
# Count how many rows match between these variables
print(f"The total of rows that do not have any reviews: {all_missing_reviews.sum()}")
# Drop those rows
geneva = geneva[~all_missing_reviews]
```

```
The total of rows that do not have any reviews: 436
```

I identified listings with no review information across all review-related variables, then counted how many completely lacked review data, and removed those rows to ensure only listings with meaningful review history remained in the dataset.

```
print(f"The dataset shape after cleaning: {geneva.shape}")
```

```
The dataset shape after cleaning: (1987, 63)
```

Write one paragraph describing what you did, and why you did it.

To handle missing values across the dataset, I used a combination of placeholder text, mode imputation, and selective row or column removal based on the type and extent of missingness.

For text-based fields with moderate to high numbers of missing entries, such as **description, neighborhood_overview host_location**, and **host_about**, I filled the gaps with clear placeholder phrases like "No description provided," "No neighborhood overview provided," or "No host description provided." This preserves the full set of observations and prevents null values from interrupting later analysis or modeling.

For host behavior variables, I applied methods that aligned with their structure. **host_response_time** was filled with "unknown response time," **host_response_rate** was imputed with zero, and **host_acceptance_rate** was filled using its mode to keep the values realistic.

Categorical flags such as **host_is_superhost** were imputed with "f" (false), which means any missing entries were treated as non-superhost. This avoids inflating the number of superhosts and keeps the distribution consistent with the data.

When missingness was minimal and the field was not meaningful to impute, such as **host_thumbnail_url**, I removed the affected rows. I also dropped columns that were redundant or overwhelmingly incomplete, such as the messy neighborhood fields, **calendar_updated**, **license**, and **bathrooms** variable where **bathrooms_text** already represented the bathroom inputs.

For structural attributes such as **bedrooms** and **beds**, I converted them to numeric values, applied a logical rule to assign one bedroom when at least one bed was present, and then removed any remaining rows with invalid or missing entries.

The review section includes the following variables: **first_review, last_review, review_scores_rating, review_scores_accuracy, review_scores_cleanliness, review_scores_checkin, review_scores_communication, review_scores_location, review_scores_value**, and **reviews_per_month**. Most of these columns contain a similar amount of missing values, which indicates that the same listings are missing review information across all review fields. These 436 rows represent listings with no review history at all, so keeping them would not add meaningful value to the analysis. I removed these listings to ensure that the dataset reflects properties with at least some review activity, which improves the usefulness and reliability of review-based insights.

After completing all cleaning and filtering steps, the final dataset contains 1,987 rows and 69 columns, reflecting a more consistent and analysis-ready structure.

2. Summary Statistics

Take a peek at your data, and look at some of the ways that it's organized by area. To do this, you may wish to use neighbourhood, or neighbourhood_cleansed variables (note the UK spelling of these words). If one of these groups has many levels, you could try a top_n filter or find a way to combine some of them together. Alternatively, you could use the lat/long variables to divide your city however you wish to. Next, generate at least 5 different summary stats that reveal things about the differences (or lack thereof) among the areas in your city.

```
# 1. List volume per neighbourhood
geneva.groupby('neighbourhood_cleansed').size().sort_values(ascending=False).head(10)
```

neighbourhood_cleansed	0
Commune de Genève	1472
Grand-Saconnex	59
Vernier	57
Carouge	50
Lancy	44
Meyrin	35
Thônex	34
Versoix	33
Chêne-Bougeries	30
Chêne-Bourg	19

dtype: int64

```
# 2. Median price per neighborhood
# Remove $ and commas, strip spaces, convert to float
geneva['price'] = (
    geneva['price']
    .str.replace('$','', regex=False)
    .str.replace(',','', regex=False)
    .str.strip()
    .astype(float)
)
geneva.groupby('neighbourhood_cleansed')['price'].median().sort_values(ascending=False).head(10)
```

neighbourhood_cleansed	price
Hermance	235.0
Troinex	220.0
Genthod	200.0
Collonge-Bellerive	195.0
Meinier	185.0
Confignon	180.0
Avully	165.5
Cologny	150.0
Vandoeuvres	150.0
Corsier	135.5

dtype: float64

```
# 3. Average rating scores across neighborhoods
geneva.groupby('neighbourhood_cleansed')['review_scores_rating'].mean().sort_values(ascending=False).head(10)
```

neighbourhood_cleansed	review_scores_rating
Avully	5.000000
Chancy	5.000000
Puplinge	5.000000
Céligny	5.000000
Collex-Bossy	5.000000
Bernex	4.967500
Anières	4.945000
Jussy	4.930000
Satigny	4.916667
Confignon	4.910000

dtype: float64

```
# 4. Proportion of reviews with average review scores
geneva.groupby('neighbourhood_cleansed')[['number_of_reviews','review_scores_rating']].mean().sort_values(by='nu
```

neighbourhood_cleansed	number_of_reviews	review_scores_rating
Soral	75.000000	4.860000
Bardonnex	70.000000	4.825000
Confignon	45.000000	4.910000
Onex	41.538462	4.609231
Meyrin	41.400000	4.654571
Commune de Genève	35.010870	4.702364
Grand-Saconnex	32.389831	4.656102
Vernier	27.368421	4.394912
Cologny	26.529412	4.870588
Presinge	26.500000	4.845000

```
# 5. Average Capacity (Beds or Bedrooms) to compare listing styles
geneva.groupby('neighbourhood_cleansed')[['beds','bedrooms']].mean().sort_values(by='beds', ascending=False).hea
```

neighbourhood_cleansed	beds	bedrooms
Presinge	4.500000	3.000000
Céligny	4.000000	3.000000
Russin	4.000000	2.000000
Puplinge	3.000000	1.000000
Genthod	2.857143	1.857143
Troinex	2.750000	2.000000
Hermance	2.750000	1.750000
Satigny	2.666667	2.000000
Avully	2.500000	2.500000
Anières	2.500000	1.500000

Show your results. Describe your findings in 1-2 paragraphs. For each stat you found, what is your team's takeaway?

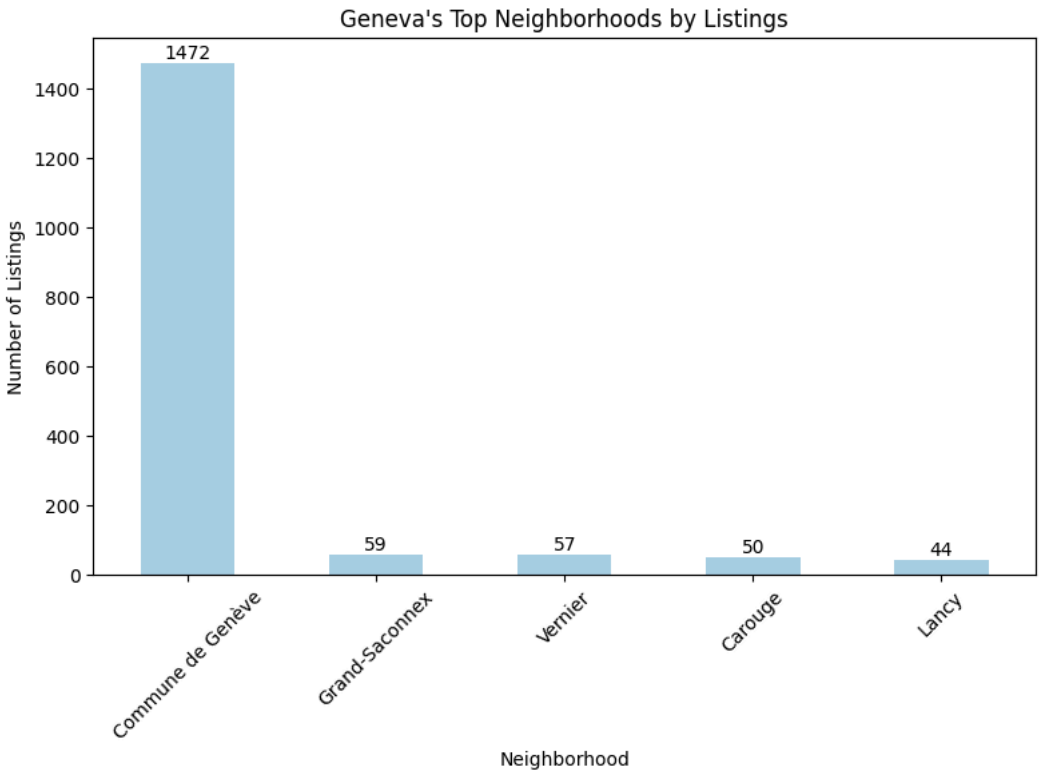
The results show a really clear divide across Geneva's neighborhoods. Commune de Genève dominates the market with over 1,400 listings, which is reasonable given it's the city center and naturally attracts the most visitors. Surrounding areas like Grand-Saconnex, Vernier, and Carouge have much smaller listings, suggesting that Airbnb activity becomes more spread out and residential as you move away from the central area. However, the pricing trend flips. High-end neighborhoods such as Hermance, Troinex, and Genthod have the highest median prices despite having far fewer listings. This hints at a more premium, exclusive rental style in those quieter areas, while still offering excellent review scores from guests.

Thus, many low-volume neighborhoods, like Avully and Puplinge, show nearly perfect ratings, which may reflect more personalized hosting. Meanwhile, the capacity highlight differences in the types of homes being offered. Smaller communes tend to list larger properties with more beds and bedrooms, whereas high-volume areas in the city center lean toward smaller apartment-style rentals. Taken together, these patterns suggest that Geneva's Airbnb market has two sides: a busy urban center with lots of options and more standardized stays, and a set of calm, high-end neighborhoods that deliver larger homes, higher prices, and very satisfied guests.

3. Data Visualization

Now generate five visualizations that help you to better understand the various neighborhoods, or regions, in your city. Be creative! This is intentionally very open-ended. The only requirement is that each one of your plots offers information about your city and its regions. Make five different types of plots for this.

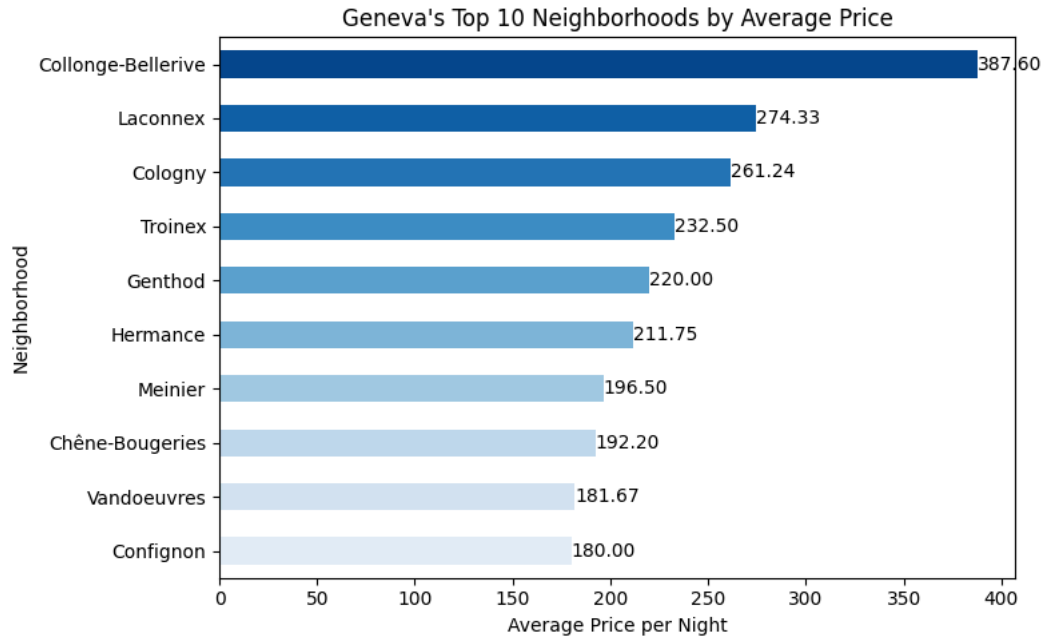
```
# 1. Bar Chart - Num Listings per Neighborhood
import seaborn as sns
plt.figure(figsize=(8,6))
sns.set_palette("Paired")
ax = geneva['neighbourhood_cleansed'].value_counts().head(5).plot(kind='bar')
plt.title("Geneva's Top Neighborhoods by Listings")
plt.xlabel('Neighborhood')
plt.ylabel('Number of Listings')
plt.xticks(rotation=45)
# Add labels for each bar
for p in ax.patches:
    ax.annotate(str(p.get_height()),
                (p.get_x() + p.get_width()/2, p.get_height()),
                ha='center', va='bottom')
plt.tight_layout()
plt.show()
```




```
# 2. Boxplot - Most Expensive Neighborhoods (removed outliers for more consistent comparison)
# Sorting by price
top5_p = geneva.groupby('neighbourhood_cleansed')['price'].median().sort_values(ascending=False).head(5).index

plt.figure(figsize=(8, 5))
sns.boxplot(data=geneva[geneva['neighbourhood_cleansed'].isin(top5_p)],
            x='neighbourhood_cleansed',
            y='price',
            hue='neighbourhood_cleansed',
            palette='Paired',
            showfliers=False, legend=False)
plt.title("Geneva's Price Distribution by Neighborhoods")
plt.tight_layout()
plt.show()
```


```
# 3. Horizontal Bar Chart - Average Price by neighborhood
# sorting by avg price
top10_np = geneva.groupby('neighbourhood_cleansed')['price'].mean().sort_values(ascending=False).head(10)

plt.figure(figsize=(8,5))
colors = sns.color_palette("Blues", len(top10_np))
ax = top10_np.sort_values().plot(kind='barh', color=colors)
```

```
plt.title("Geneva's Top 10 Neighborhoods by Average Price")
plt.xlabel('Average Price per Night')
plt.ylabel('Neighborhood')

# Add labels for each variables
for i, v in enumerate(top10_np.sort_values()):
    plt.text(v + 0.01, i, f"{v:.2f}", va='center')
plt.tight_layout()
plt.show()
```

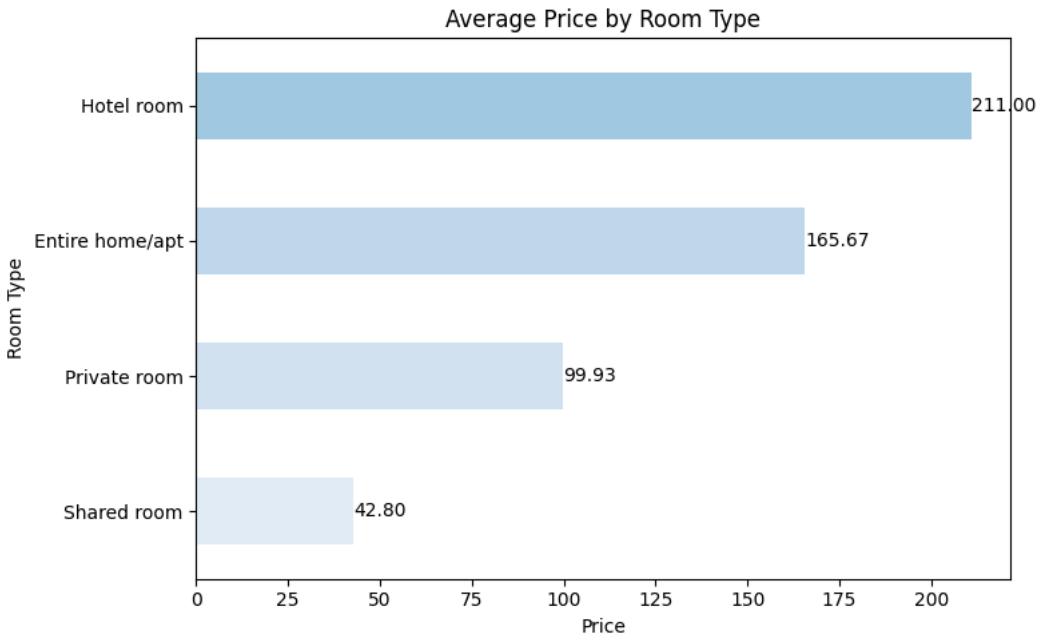



```
# Average Price by Room Type
# sorting by avg price
room_price = geneva.groupby('room_type')['price'].mean().sort_values(ascending=False)
room_price.head(5)
```

room_type	price
Hotel room	211.000000
Entire home/apt	165.671053
Private room	99.926230
Shared room	42.800000

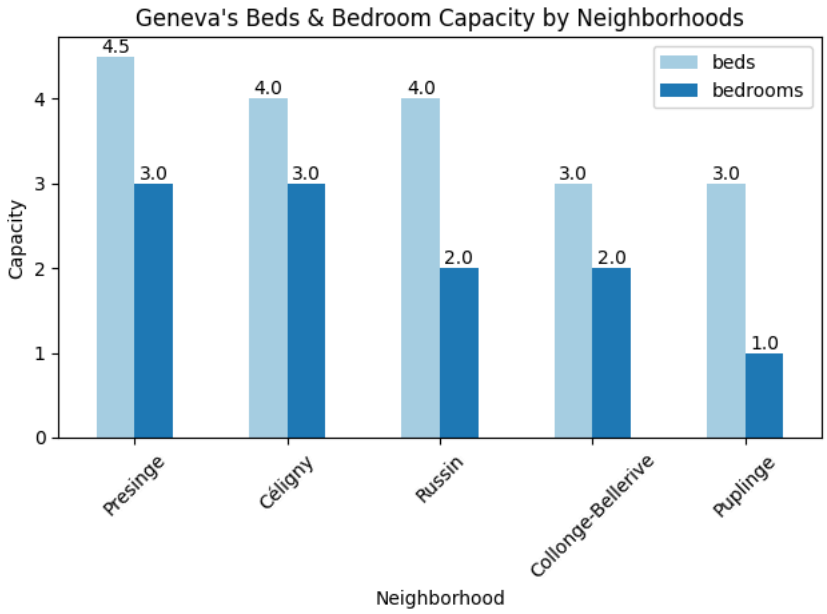
dtype: float64

```
# 3. Horizontal Bar Chart – Average Price by Room Type
plt.figure(figsize=(8,5))
sorted_prices = room_price.sort_values()  # consistent series for plotting

ax = sorted_prices.plot(kind='barh', color=colors)
plt.title("Average Price by Room Type")
plt.xlabel('Price')
plt.ylabel('Room Type')

# Add labels correctly
for i, (room, v) in enumerate(sorted_prices.items()):
    plt.text(v + 0.01, i, f"{v:.2f}", va='center')

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



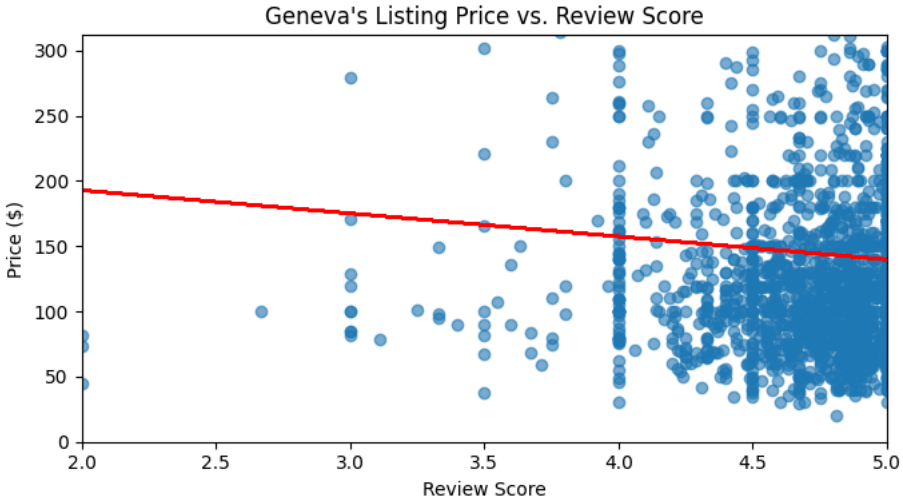

```
# 4. Grouped Bar Chart - Based on Avg Bed & Bedroom Capacity
top5_c = (geneva.groupby('neighbourhood_cleansed')[['beds', 'bedrooms']].median().sort_values(by='beds', ascendi

plt.figure(figsize=(8, 6))
sns.set_palette("Paired")
ax = top5_c.plot(kind='bar')
plt.title("Geneva's Beds & Bedroom Capacity by Neighborhoods")
plt.xlabel('Neighborhood')
plt.ylabel('Capacity')
# Rotate x-axis labels
plt.xticks(rotation=45)
# Add value labels above each bar
for container in ax.containers:
    ax.bar_label(container, fmt='%.1f', label_type='edge')
plt.tight_layout()
plt.show()
```

```
<Figure size 800x600 with 0 Axes>
```

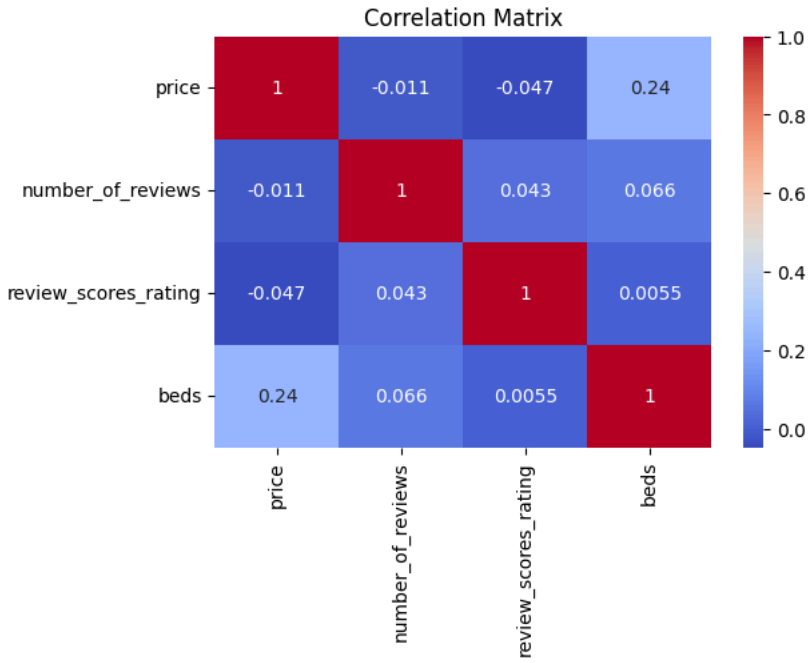



```
# 5. Scatterplot - Geneva's Listing Price vs. Review Score

plt.figure(figsize=(7, 4))
sns.set_palette("Paired")
plt.scatter(geneva['review_scores_rating'], geneva['price'], alpha=0.6,color=sns.color_palette()[1])
# Adding trend line
m, b = np.polyfit(geneva['review_scores_rating'], geneva['price'], 1)
plt.plot(geneva['review_scores_rating'], m*geneva['review_scores_rating'] + b, color='red')
plt.title("Geneva's Listing Price vs. Review Score")
plt.xlabel('Review Score')
```

```
plt.ylabel('Price ($)')
plt.xlim(2.0, 5.0)
plt.ylim(0, geneva['price'].quantile(0.95))
plt.tight_layout()
plt.show()
```

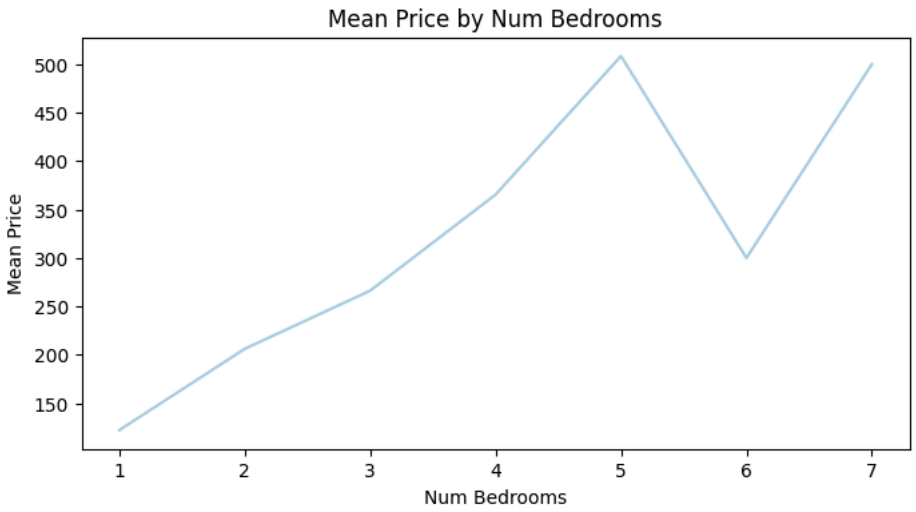



```
# Correlation Matrix/Heatmap
import seaborn as sns
numerical_cols = ['price', 'number_of_reviews', 'review_scores_rating', 'beds']
corr = geneva[numerical_cols].corr()

plt.figure(figsize=(6, 4))
sns.heatmap(corr, annot = True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```




```
# Line Chart - Mean Price by Number of Bedrooms
avgp_bedr = geneva.groupby('bedrooms')['price'].mean().sort_index()
sns.set_palette("Paired")
plt.figure(figsize=(8,4))
avgp_bedr.plot(kind='line')
plt.title('Mean Price by Num Bedrooms')
plt.xlabel('Num Bedrooms')
plt.ylabel('Mean Price')
```

```
Text(0, 0.5, 'Mean Price')
```



Write a two-paragraph description that explains the choices that you made, and what the resulting plots teach you about your data.

To start analyzing Geneva's Airbnb market, I created multiple visualizations that each show a different aspect of how listings and prices vary across the city. The first plot, "Top 10 Neighborhoods by Listing Count," shows that Commune de Genève overwhelmingly dominates in the number of listings, showing that Airbnb activity is very centralized in the city center. The visualization, "Geneva's Price Distribution by Neighborhoods," compares nightly prices across the most active neighborhoods. This boxplot shows significant variation. Troinex and Collonge-Bellerive show higher median prices and wider price ranges, suggesting there are luxury and standard listings within the same area.

The first bar chart, "Average Beds and Bedrooms by Neighborhood," shows how property size varies across different parts of Geneva. Neighborhoods like Céligny and Collonge-Bellerive tend to offer larger listings, which likely contributes to their higher prices. The next bar chart, "Geneva's Top 10 Neighbourhoods by Average Price," highlights which areas contain the most expensive Airbnb's. This supports the pattern that neighborhoods on the outskirts or near the lake tend to be more expensive than central areas with higher listing density. The visualization, "Average Price by Room Type," makes it clear that entire homes/apartments are much more expensive than private or shared-room listings.

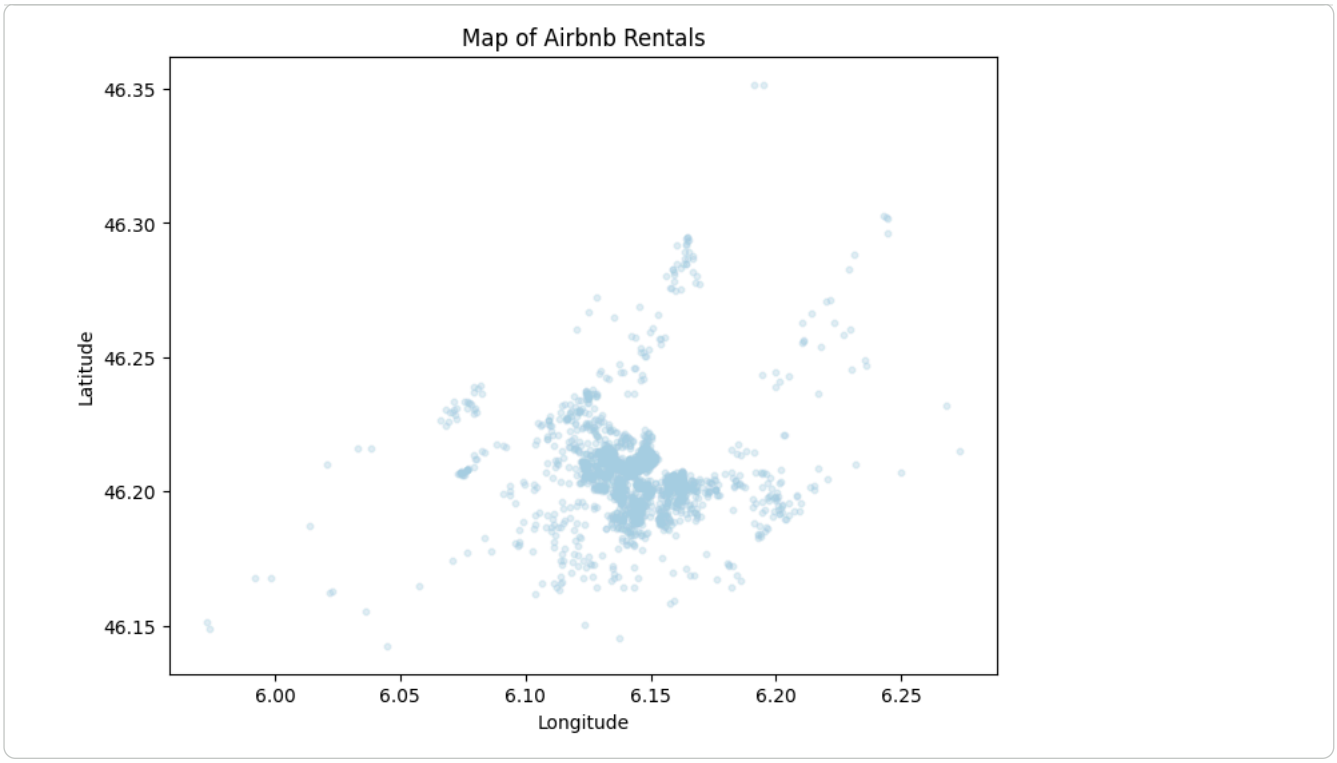
The correlation heatmap shows that most variables have weak linear relationships with price, with the exception of accommodates and bedrooms. The visualization "Geneva's Listing Price vs. Review Score" tells us that customer ratings have almost no relationship with price, which means that guests may value size and location more strongly when choosing listings.

Together, these visualizations teach us the pattern that listing prices are mainly influenced by size, capacity, and location.

4. Mapping

Generate a map of the Airbnb rentals in your city

```
# Generating a map of Airbnb rentals in the city using Latitude and Longitude
plt.figure(figsize=(8,6))
sns.set_palette("Paired")
plt.scatter(geneva['longitude'], geneva['latitude'], alpha=0.3, s=10)
plt.title('Map of Airbnb Rentals')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```



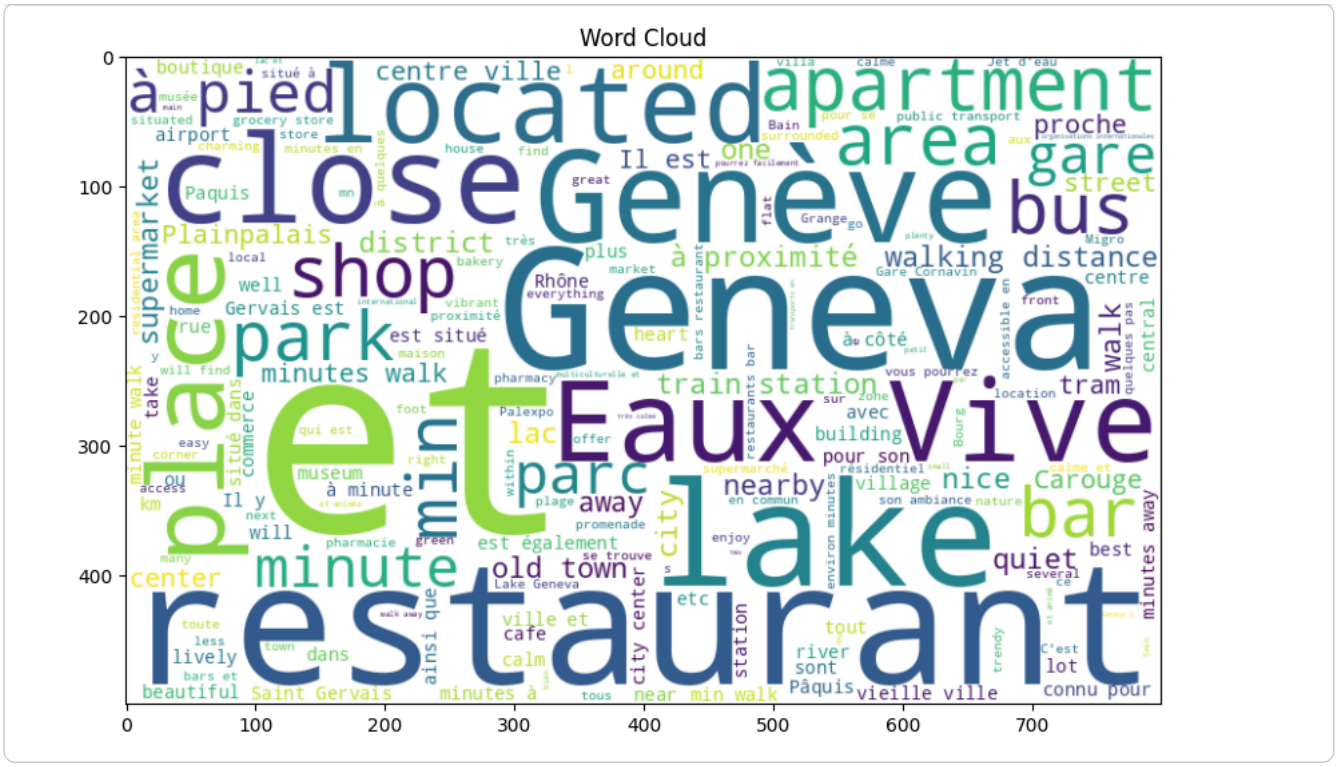
Do any key features here seem to stand out? What are a few of the things your map shows you about your region?

This map shows that the central part of Geneva contains the most Airbnb listings. The cluster is very dense in the city center, which suggests that most Airbnbs are located downtown and closest to tourist attractions like restaurants, shops, Lake Geneva, and the Old Town. This aligns with the first bar chart showing the Commune de Geneve neighborhood having the highest number of Airbnb listings by far.

5. Word Cloud

Using the neighborhood overview column in your dataset, generate a wordcloud.

```
from wordcloud import WordCloud, STOPWORDS

# Dropping NAs
text_data = " ".join(geneva['neighborhood_overview'].dropna().astype(str).tolist())

# after running the word cloud, i am updating the stopwords to make the results more meaningful
stopwords = set(STOPWORDS)
stopwords.update(['overview', 'provided', 'br', 'neighborhood', 'neighbourhood', 'quartier', 'le', 'la', 'de', '

# generating the word cloud
wc = WordCloud(width=800,
               height=500,
               background_color='white',
               stopwords = stopwords,
               colormap='viridis').generate(text_data)
# Plot
plt.figure(figsize=(10, 8))
plt.imshow(wc, interpolation='bilinear')
plt.title('Word Cloud')
plt.show()
```



What are some terms that seem to be emphasized here?

Before updating the stopwords list, the initial word cloud was filled with generic filler words such as overview, provided, neighborhood, br, le, la, de, and du, which do not give any meaningful descriptive information. After removing the filler words, the most common terms are Geneva, lake, restaurant, park, walking distance, shop, supermarket, and overall references to public transportation and proximity. This shows the recurring themes emphasized by Airbnb hosts. They like to highlight convenience, access to food, parks, and the airport, which are all things that would appeal to someone looking to book an Airbnb in Geneva.

Prediction

1. Create a multiple regression model with the outcome variable price. The dataset description implies that the units are Swiss francs. Swiss francs are nearly equivalent to US dollars.

```
from sklearn.model_selection import train_test_split
import statsmodels.api as sm

reg_model = geneva.copy()

# dropping Na values & negatives
reg_model = reg_model[reg_model['price'] > 0].dropna(subset=['price'])

# Change the necessary categorical variables to booleean
reg_model['host_is_superhost'] = reg_model['host_is_superhost'].map({'t': True, 'f': False})
reg_model['instant_bookable'] = reg_model['instant_bookable'].map({'t': True, 'f': False})
reg_model[['host_is_superhost','instant_bookable']].info()

#converting bathrooms_text to numeric
reg_model['bathrooms_n'] = ( reg_model['bathrooms_text'] .str.extract(r'(\d+\.?\d*)').astype(float))

# meaningful numerical values
numerical_values = ['accommodates', 'bedrooms', 'beds', 'number_of_reviews',
                    'review_scores_rating', 'availability_90', 'availability_365', 'review_scores_location',
                    'bathrooms_n', 'minimum_nights']

categorical_values = ['room_type', 'neighbourhood_cleansed','property_type']
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1987 entries, 0 to 2435
Data columns (total 2 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   host_is_superhost  1987 non-null   bool
 1   instant_bookable   1987 non-null   bool
dtypes: bool(2)
memory usage: 19.4 KB
```

```
reg_model = reg_model[numerical_values + ['host_is_superhost', 'instant_bookable'] + categorical_values + ['pric

# you may wish to consider a log transformation on the response variable.
reg_model['log_price'] = np.log(reg_model['price'])
reg_model = reg_model.drop(columns=['price'])

reg_model = pd.get_dummies(reg_model, columns=categorical_values, drop_first=True)

# split 60/40
train_df, val_df = train_test_split(reg_model,train_size=0.60,shuffle=True,random_state=42)
print("Training set shape:", train_df.shape)
print("Validation set shape:", val_df.shape)
```

```
Training set shape: (1192, 88)
Validation set shape: (795, 88)
```

```
# regression using backward elimination
def backward_elim(df, target="log_price", alpha=0.05, verbose =True):
    model_data = df.select_dtypes(include=[np.number]).dropna()
    y = model_data[target]
    X = model_data.drop(columns=[target])
    X = sm.add_constant(X, has_constant="add")
    cols = list(X.columns)

    for _ in range(len(cols) - 1):
        model = sm.OLS(y, X[cols]).fit()
        non_const = [c for c in cols if c != "const"]
        if not non_const:
            break
        pvals = model.pvalues.drop(labels=["const"])
        worst = pvals.idxmax()
        if pvals[worst] > alpha:
            if verbose:
                print(f"Removing {worst} (p = {pvals[worst]:.4f})")
            cols.remove(worst)
        else:
            break
    final_model = sm.OLS(y, X[cols]).fit()
    return final_model, cols, X[cols], y

final_model, kept_cols, X_clean, y_clean = backward_elim(train_df, target="log_price", alpha=0.05, verbose=True)
print(final_model.summary())
```

```
Removing review_scores_rating (p = 0.9555)
Removing beds (p = 0.5763)
Removing minimum_nights (p = 0.3301)
Removing availability_365 (p = 0.2157)
                            OLS Regression Results
==============================================================================
Dep. Variable:              log_price   R-squared:                       0.358
Model:                            OLS   Adj. R-squared:                  0.355
Method:                 Least Squares   F-statistic:                     109.6
Date:                Tue, 09 Dec 2025   Prob (F-statistic):          6.92e-110
Time:                        14:45:55   Log-Likelihood:                -740.97
No. Observations:                1186   AIC:                             1496.
Df Residuals:                    1179   BIC:                             1531.
Df Model:                           6
Covariance Type:            nonrobust
============================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------
const                      3.4163      0.175     19.542      0.000       3.073       3.759
accommodates               0.1622      0.013     12.536      0.000       0.137       0.188
bedrooms                   0.0925      0.032      2.920      0.004       0.030       0.155
number_of_reviews         -0.0005      0.000     -2.312      0.021      -0.001   -7.68e-05
availability_90            0.0025      0.000      6.227      0.000       0.002       0.003
review_scores_location     0.1188      0.035      3.374      0.001       0.050       0.188
bathrooms_n                0.1313      0.037      3.559      0.000       0.059       0.204
==============================================================================
Omnibus:                      470.271   Durbin-Watson:                   1.976
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             5032.580
Skew:                           1.523   Prob(JB):                         0.00
Kurtosis:                      12.621   Cond. No.                         970.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```
# R2 and RMSE
from sklearn.metrics import r2_score, root_mean_squared_error

# Training set
y_pred_train = final_model.predict(X_clean)
print("Training set:")
```

```
print(f"  R²  = {r2_score(y_clean, y_pred_train):.4f}")
print(f"  RMSE= {root_mean_squared_error(y_clean, y_pred_train):.4f}")

# Validation/Test set
pred_cols = [c for c in kept_cols if c != "const"]
X_val = sm.add_constant(val_df[pred_cols].dropna(), has_constant="add")
y_val = val_df.loc[X_val.index, "log_price"]
y_pred_val = final_model.predict(X_val)

print("\nValidation set:")
print(f"  R²  = {r2_score(y_val, y_pred_val):.4f}")
print(f"  RMSE= {root_mean_squared_error(y_val, y_pred_val):.4f}")
```

```
Training set:
  R²  = 0.3581
  RMSE= 0.4520

Validation set:
  R²  = 0.3858
  RMSE= 0.4246
```

2. Describe your process. How did you wind up including the independent variables that you kept, and discarding the ones that you didn't keep? In a narrative of at least two paragraphs, discuss your process and your reasoning. In the write-up, be sure to talk about how you evaluated the quality of your model.

To build the multiple regression model predicting Geneva Airbnb prices, I started by selecting numerical and categorical variables that could be possible influences of nightly price. The numerical predictors included accommodates, bedrooms, number_of_reviews, availability_90, review_scores_location, minimum_nights, and number of bathrooms. The categorical variables I chose were room_type, neighbourhood_cleansed, and property_type, which were converted into dummy variables. These variables all have the potential to reasonably influence price. Before modeling, I applied a log transformation to the response variable (log_price) to stabilize variance and reduce the effect of extreme price outliers.

Then, I used backward elimination to remove predictors that had very high p-values. The variables review_scores_rating, beds, minimum_nights, and availability_365 were removed because their p-values were higher than 0.05, meaning they did not influence price variation. The final model kept accommodates, bedrooms, number_of_reviews, availability_90, review_scores_location, and bathrooms_n. Accommodates and bedrooms are strong predictors of price, which aligns with expectations that the bigger a property is, the more the nightly cost will be. Along with this, the significance of bathrooms_n and review_scores_location suggests that amenities and location also influence price. It is important to note that the coefficient for number_of_reviews is negative, which indicates that listings with more reviews tend to have slightly lower prices. This could be because lower-priced listings attract more bookings, giving the property more reviews over time.

3. Show a screenshot of your regression summary, and explain the regression equation that it generated.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              log_price   R-squared:                       0.358
Model:                            OLS   Adj. R-squared:                  0.355
Method:                 Least Squares   F-statistic:                     109.6
Date:                Tue, 09 Dec 2025   Prob (F-statistic):          6.92e-110
Time:                        12:44:53   Log-Likelihood:                -740.97
No. Observations:                1186   AIC:                             1496.
Df Residuals:                    1179   BIC:                             1531.
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
const                      3.4163      0.175     19.542      0.000       3.073       3.759
accommodates               0.1622      0.013     12.536      0.000       0.137       0.188
bedrooms                   0.0925      0.032      2.920      0.004       0.030       0.155
number_of_reviews         -0.0005      0.000     -2.312      0.021      -0.001   -7.68e-05
availability_90            0.0025      0.000      6.227      0.000       0.002       0.003
review_scores_location     0.1188      0.035      3.374      0.001       0.050       0.188
bathrooms_n                0.1313      0.037      3.559      0.000       0.059       0.204
==============================================================================
Omnibus:                      470.271   Durbin-Watson:                   1.976
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             5032.580
Skew:                           1.523   Prob(JB):                         0.00
Kurtosis:                      12.621   Cond. No.                         970.
==============================================================================
```

Based on the regression summary, the estimated regression equation is:

$$\log(price) = 3.4163 + 0.1622 \cdot accommodates + 0.0925 \cdot bedrooms - 0.0005 \cdot number\_of\_reviews + 0.0025 \cdot availability\_90 + 0.1188 \cdot review\_scores\_location + 0.1313 \cdot bathrooms\_n$$

The dependent variable is log_price, and each coefficient is a percentage change in price. So, each additional accommodation increased the price by 16.22%.

4. Analyze any other metrics that are relevant for linear regression models. Based on these, what can you say about your model's performance in 1-2 paragraphs?

The model achieved an R^2 of 0.358 on the training set and 0.386 on the validation set, meaning it explains a bit more than one-third of the variation in Airbnb prices. This is reasonable and makes sense because the pricing of Airbnb's is likely influenced by many factors not in the dataset such as interior design, view quality, season, distance to transportation, personal preference, and host pricing strategies. The RMSE values at 0.452 for training and 0.425 for validation on the log scale show that the model generalizes well and is not overfitting.

The regression model shows several meaningful predictors: accommodates, bedrooms, bathrooms_n, and review_scores_location, which aligns with expectations.

Classification

Part I. k-nn. Predict whether a rental in your neighborhood will have some particular amenity, or combination of amenities.

```
# Goal: Predict wifi, hair dryer, and dishwasher.
# Predict whether an Airbnb will have all 3 amenities in Commune de Genève neighborhood
target_neighborhood = "Commune de Genève"
df_knn = geneva[geneva['neighbourhood_cleansed'] == target_neighborhood].copy()

df_knn = df_knn.dropna(subset=['amenities'])

#converting bathrooms_text to numeric
df_knn['bathrooms_n'] = (df_knn['bathrooms_text'] .str.extract(r'(\d+\.?\d*)').astype(float))

# flagging the amenities column for wifi, hair dryer, and dishwasher
df_knn['has_wifi'] = df_knn['amenities'].str.contains('wifi', case=False, na=False)
df_knn['has_hair_dryer'] = df_knn['amenities'].str.contains('hair dryer', case=False, na=False)
df_knn['has_dishwasher'] = df_knn['amenities'].str.contains('dishwasher', case=False, na=False)

# airbnb has all 3 = 1
df_knn['has_wifi_hair_dryer_dishwasher'] = np.where(
    df_knn['has_wifi'] & df_knn['has_hair_dryer'] & df_knn['has_dishwasher'],1,0)

df_knn[['amenities', 'has_wifi', 'has_hair_dryer', 'has_dishwasher', 'has_wifi_hair_dryer_dishwasher']].head()
```

	amenities	has_wifi	has_hair_dryer	has_dishwasher	has_wifi_hair_dryer_dishwasher
0	["Dryer", "Kitchen", "TV with standard cable",...	True	True	False	0
1	["Dishwasher", "Wine glasses", "Essentials", "...	True	True	True	1
4	["Dishwasher", "Free dryer \u2013 In unit", "S...	True	True	True	1
5	["Shower gel", "Essentials", "Free street park...	True	True	False	0

```
# KNN numerical predictors
numerical_predictors = ['accommodates', 'bedrooms', 'number_of_reviews', 'availability_90', 'review_scores_locat

df_knn = df_knn[numerical_predictors + ['has_wifi_hair_dryer_dishwasher']].dropna()

# x = num predictors, y = airbnb has amenities
X = df_knn[numerical_predictors]
y = df_knn['has_wifi_hair_dryer_dishwasher']
df_knn.shape
```

```
(1467, 7)
```

```
# split
X_train, X_val, y_train, y_val = train_test_split(X, y,train_size=0.40,random_state=42, stratify=y)

print("Training shape:", X_train.shape)
print("Validation shape:", X_val.shape)
```

```
Training shape: (586, 6)
Validation shape: (881, 6)
```

```
# normalize with scaler
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled   = scaler.transform(X_val)
```

```
# KNN model, k =7
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train_scaled, y_train)

y_train_pred = knn.predict(X_train_scaled)
y_val_pred   = knn.predict(X_val_scaled)

print("Training Accuracy (k=7):", accuracy_score(y_train, y_train_pred))
print("Validation Accuracy (k=7):", accuracy_score(y_val, y_val_pred))
```

```
Training Accuracy (k=7): 0.7901023890784983
Validation Accuracy (k=7): 0.7275822928490352
```

```
# finding the best k value
k_values = list(range(1, 26, 2))
val_results = []

for k in k_values:
    knn_k = KNeighborsClassifier(n_neighbors=k)
    knn_k.fit(X_train_scaled, y_train)
    y_val_pred_k = knn_k.predict(X_val_scaled)
    acc = accuracy_score(y_val, y_val_pred_k)
    val_results.append((k, acc))
    print(f"k = {k}, Validation Accuracy = {acc:.4f}")

best_k, best_acc = max(val_results, key=lambda x: x[1])
print("\nBest k-value:", best_k, "with validation accuracy:", round(best_acc, 4))
```

```
k = 1, Validation Accuracy = 0.6402
k = 3, Validation Accuracy = 0.6822
k = 5, Validation Accuracy = 0.7299
k = 7, Validation Accuracy = 0.7276
k = 9, Validation Accuracy = 0.7412
k = 11, Validation Accuracy = 0.7514
k = 13, Validation Accuracy = 0.7469
k = 15, Validation Accuracy = 0.7514
k = 17, Validation Accuracy = 0.7526
k = 19, Validation Accuracy = 0.7514
k = 21, Validation Accuracy = 0.7469
k = 23, Validation Accuracy = 0.7503
k = 25, Validation Accuracy = 0.7491

Best k-value: 17 with validation accuracy: 0.7526
```

```
knn_best = KNeighborsClassifier(n_neighbors=best_k)
knn_best.fit(X_train_scaled, y_train)

y_val_pred_best = knn_best.predict(X_val_scaled)
model_acc = accuracy_score(y_val, y_val_pred_best)
print("Final Validation Accuracy (best k):", model_acc)

# naive benchmark
majority_class = y_train.mode()[0]
y_val_naive = np.full_like(y_val, fill_value=majority_class)
naive_acc = accuracy_score(y_val, y_val_naive)
print("Naive Benchmark Accuracy:", naive_acc)
```

```
Final Validation Accuracy (best k): 0.7525539160045402
Naive Benchmark Accuracy: 0.7423382519863791
```

Write a two-paragraph narrative that describes how you did this. In your narrative, be sure to describe your amenity choice(s), your predictor choices, and the process that you used to arrive at the particular k value that you used. When assessing your model, you may also wish to consider its performance against a naive benchmark.

This k-nearest neighbors (k-NN) model predicts whether an Airbnb listing in the Commune de Genève neighborhood offered Wi-Fi, a hair dryer, and a dishwasher. We selected these amenities because they represent internet access, a personal-care appliance, and a

kitchen appliance, which can provide insight into the Airbnb quality. I started by filtering the dataset to only listings in this neighborhood and then created indicator variables based on whether each amenity appeared in the "amenities" text column. After that, I created an outcome variable, has_wifi_hair_dryer_dishwasher, which equals 1 if a listing contains all three amenities and 0 otherwise.

Then, I selected numerical predictors that were statistically significant in the regression model: accommodates, bedrooms, number_of_reviews, availability_90, review_scores_location, and bathrooms_n. All predictors were standardized using StandardScaler so that variables measured on different scales contributed equally. The dataset was then split into a 40% training and 60% validation set.

The initial model used k = 7, producing a training accuracy of 0.7901 and a validation accuracy of 0.7276. To identify the optimal k value, I evaluated all odd values from 1 to 25. The best performance occurred at k = 17, which achieved a validation accuracy of 0.7526. Then, I compared it to a naive benchmark that always predicts the majority class in the training data, which achieved 0.7423 accuracy. The improvement is only by about 1%, but this is still meaningful for the prediction task. Even though the improvement is small, it still shows that the model is finding real patterns in the data. The k-NN classifier is using listing characteristics like bedrooms, accommodates, bathrooms, and review scores to identify higher-quality listings that are more likely to offer all three amenities. The training and validation accuracies are also close, indicating the model is stable and not overfitting.

This is intuitive because amenities like dishwashers and hair dryers tend to appear in larger or better-equipped listings, which the numerical predictors help capture.

Part II. Classification Tree

```
# Set seed for reproducible
np.random.seed(699)

# Import necessary functions for use
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier, plot_tree

class_tree = geneva.copy()
```

Build a classification tree that predicts the host_response_time variable.

```
# Drop irrelevant columns and keep the important ones from the dataset
drop_cols = [
    'listing_url','scrape_id','last_scraped','source','name','description',
    'neighborhood_overview','picture_url','host_url','host_name','host_since',
    'host_location','host_about','host_thumbnail_url','host_picture_url',
    'host_verifications','calendar_last_scraped','first_review','last_review',
    'amenities', 'id','number_of_reviews','host_id'
]

# Change data type of rate variables to numerical
class_tree['host_response_rate'] = (
    class_tree['host_response_rate'].astype(str).str.replace('%','', regex=False).astype(float)
)

class_tree['host_acceptance_rate'] = (
    class_tree['host_acceptance_rate'].astype(str).str.replace('%','', regex=False).astype(float)
)

# Change the necessary categorical variables to booleean
class_tree['host_is_superhost'] = class_tree['host_is_superhost'].replace({'t': True, 'f': False}).astype(bool)
class_tree['instant_bookable'] = class_tree['instant_bookable'].replace({'t': True, 'f': False}).astype(bool)
class_tree['host_has_profile_pic'] = class_tree['host_has_profile_pic'].replace({'t': True, 'f': False}).astype(
class_tree['host_identity_verified'] = class_tree['host_identity_verified'].replace({'t': True, 'f': False}).ast

class_tree_cleaned = class_tree.drop(columns=drop_cols)

# Define predictors (X) and target (y)
X = class_tree.drop(columns=['host_response_time'] + drop_cols)
y = class_tree['host_response_time']

# Encode categorical predictors
X = pd.get_dummies(X, drop_first=True)

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, train_size=0.60, random_state=699
)
```

```
/tmp/ipython-input-2473694896.py:20: FutureWarning: Downcasting behavior in `replace` is deprecated and will be r
  class_tree['host_is_superhost'] = class_tree['host_is_superhost'].replace({'t': True, 'f': False}).astype(bool)
```

```
/tmp/ipython-input-2473694896.py:21: FutureWarning: Downcasting behavior in `replace` is deprecated and will be r
  class_tree['instant_bookable'] = class_tree['instant_bookable'].replace({'t': True, 'f': False}).astype(bool)
/tmp/ipython-input-2473694896.py:22: FutureWarning: Downcasting behavior in `replace` is deprecated and will be r
  class_tree['host_has_profile_pic'] = class_tree['host_has_profile_pic'].replace({'t': True, 'f': False}).astype
/tmp/ipython-input-2473694896.py:23: FutureWarning: Downcasting behavior in `replace` is deprecated and will be r
  class_tree['host_identity_verified'] = class_tree['host_identity_verified'].replace({'t': True, 'f': False}).as
```

Determine the ideal size of your tree using cross-validation.

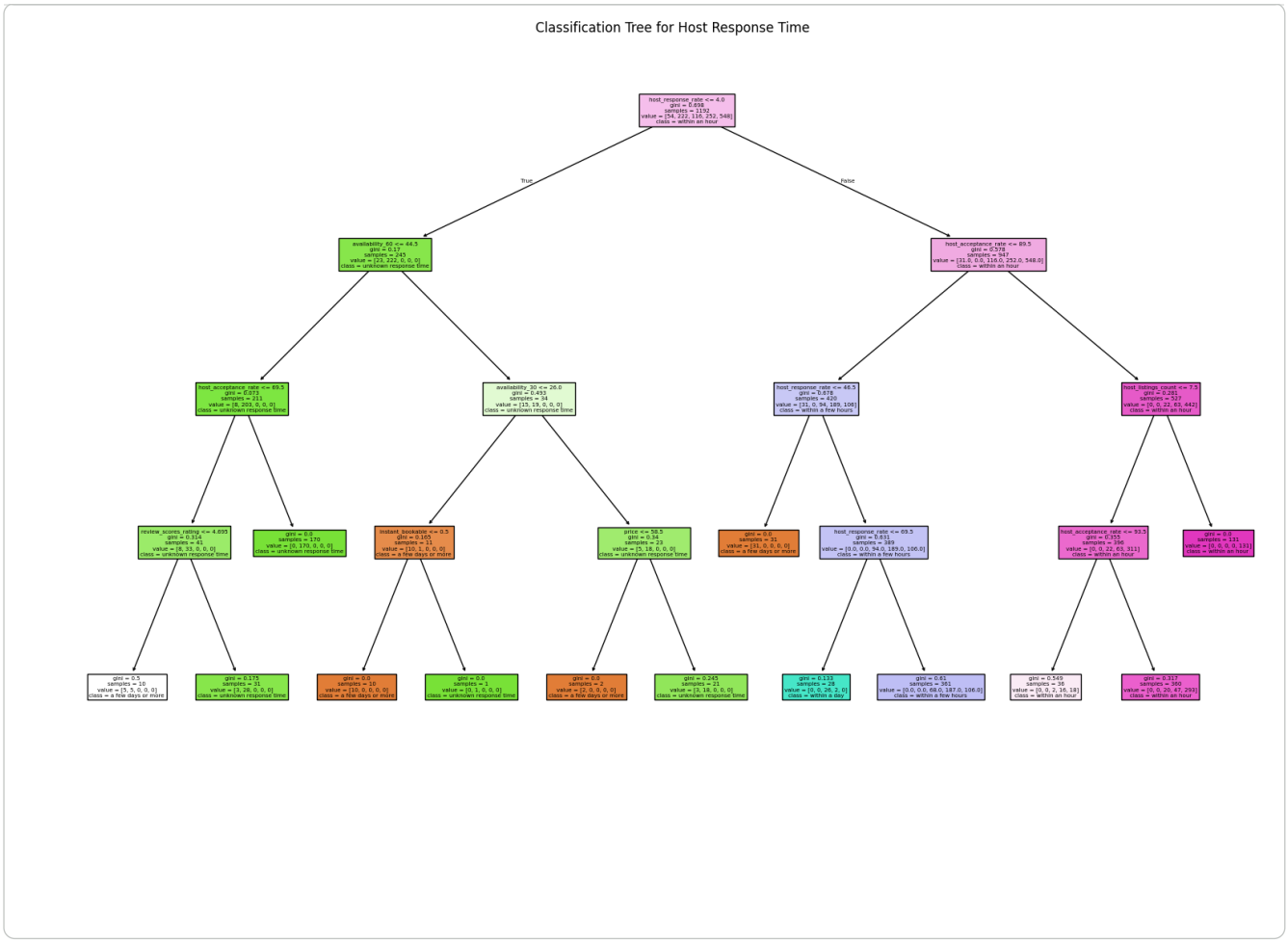
```
# Cross-validation loop to find best depth, keep the range from 1-6
depths = range(1, 6)
cv_scores = []
for d in depths:
    clf = DecisionTreeClassifier(max_depth=d, random_state=699)
    scores = cross_val_score(clf, X_train, y_train, cv=5)
    cv_scores.append(scores.mean())

# Create score table
score_table = pd.DataFrame({
    'Tree Depth': depths,
    'CV Accuracy': cv_scores
})

# Identify best depth
best_depth = score_table.loc[score_table['CV Accuracy'].idxmax(), 'Tree Depth']
print("Best depth:", best_depth)
score_table
```

```
Best depth: 4
```

	Tree Depth	CV Accuracy
0	1	0.645976
1	2	0.708903
2	3	0.735727
3	4	0.746644
4	5	0.745800

Show your tree model here.

```
# Fit final tree
final_tree = DecisionTreeClassifier(max_depth=best_depth, random_state=699)
final_tree.fit(X_train, y_train)

# Plot tree
plt.figure(figsize=(20,12))
plot_tree(final_tree, filled=True, feature_names=X.columns, class_names=final_tree.classes_)
plt.title("Classification Tree for Host Response Time")
plt.show()
```




```
# Feature importance
feature_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': final_tree.feature_importances_
}).sort_values(by='Importance', ascending=False)

feature_importance.head(10)
```

	Feature	Importance
0	host_response_rate	0.662150
1	host_acceptance_rate	0.270333
16	availability_60	0.020786
3	host_listings_count	0.015999
15	availability_30	0.015541
12	price	0.005856
21	review_scores_rating	0.005366
28	instant_bookable	0.003968
8	longitude	0.000000
7	latitude	0.000000

In a 1-2 paragraph write-up, describe your process. Mention anything that you found interesting as you explored various possible models, the process you used to arrive at the model you finished with, and any findings that you can draw from the model results.

To build a classification model predicting a host's response time, I first cleaned and prepared the Geneva dataset by removing irrelevant listing- and host-level text fields, converting rate variables into numeric values, and encoding all necessary categorical variables as Boolean or dummy features. After defining the predictors and target, I used a systematic cross-validation process to test tree depths from 1 to 6. The accuracy scores clearly indicated that a tree depth of 4 provided the strongest performance, balancing predictive power with model simplicity. I then trained the final decision tree (split to 60/40) and examined its feature importance rankings to better understand the drivers behind host responsiveness.

The resulting model highlights several actionable insights for Geneva. Most notably, host response rate and host acceptance rate dominate the prediction of response time, together contributing nearly all of the explanatory power - above 90%. This implies that hosts who are already highly engaged with the platform (replying frequently and accepting inquiries consistently) tend to also respond more promptly. Availability variables, pricing, and review scores play minor roles. These findings suggest that Geneva can more accurately estimate response times by prioritizing behavioral engagement metrics over listing-level characteristics. Operationally, this can guide how the platform sets guest expectations, identifies hosts who may need support, and designs incentives that improve responsiveness where it matters most.

Part III. Transformer

```
from bs4 import BeautifulSoup
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification, Trainer, TrainingArguments
import torch
from torch.utils.data import Dataset

trans_mod = geneva.copy()

# Create quartile bins for price
trans_mod['price_quartile'] = pd.qcut(
    trans_mod['price'], q=4, labels=['Q1','Q2','Q3','Q4']
)

# Combine descriptive text fields
trans_mod['text_features'] = (
    trans_mod['description'].fillna('') + ' ' +
    trans_mod['host_about'].fillna('') + ' ' +
    trans_mod['amenities'].fillna('')
)

# Clean HTML tags and lowercase
def clean_html(text):
    return BeautifulSoup(text, "html.parser").get_text()

trans_mod['text_features'] = trans_mod['text_features'].apply(clean_html).str.lower()
trans_mod['text_features'].head(5)
```

	text_features
0	the spacethis is a private room with own half ...
1	the spaceversion française et anglaise it is ...
3	this cozy flat is located in a very pleasant v...
4	one bedroom apartment located in a pedestrian ...
5	nice modern bedroom with confortable king size...

dtype: object

```
trans_mod[['text_features','price_quartile']].head()
```

	text_features	price_quartile
0	the spacethis is a private room with own half ...	Q2
1	the spaceversion française et anglaise it is ...	Q1
3	this cozy flat is located in a very pleasant v...	Q1
4	one bedroom apartment located in a pedestrian ...	Q3
5	nice modern bedroom with confortable king size...	Q1

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification, Trainer, TrainingArguments
import transformers
transformers.logging.set_verbosity_error()
import torch
from torch.utils.data import Dataset
import os
os.environ["WANDB_DISABLED"] = "true"

# create label
le = LabelEncoder()
```

```
trans_mod['label'] = le.fit_transform(trans_mod['price_quartile'])

# split data
train_texts, test_texts, train_labels, test_labels = train_test_split(
    trans_mod['text_features'].tolist(),
    trans_mod['label'].tolist(),
    train_size=0.60,
    random_state=699
)
print("Training set size:", len(train_texts))
print("Validation set size:", len(test_texts))
```

```
Training set size: 1192
Validation set size: 795
```

Use a transformer model of your choosing to build a classification model which predicts the price quartile for a data record based on these 3 input variables.

```
# tokenizer
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')

# dataset
class TextDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=256):
        self.encodings = tokenizer(texts, truncation=True, padding=True, max_length=max_len)
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = TextDataset(train_texts, train_labels, tokenizer)
test_dataset = TextDataset(test_texts, test_labels, tokenizer)

# load model
model = DistilBertForSequenceClassification.from_pretrained(
    'distilbert-base-uncased',
    num_labels=4
)

# training args
training_args = TrainingArguments(
    output_dir='./results',
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=2,
    logging_dir='./logs',
    logging_steps=1000,
    save_strategy="no",
    disable_tqdm=True,
    report_to="none"
)

# trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset
)

# train
trainer.train()
results = trainer.evaluate()
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/t
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100%                 48.0/48.0 [00:00<00:00, 5.17kB/s]
vocab.txt: 100%                             232k/232k [00:00<00:00, 3.37MB/s]
tokenizer.json: 100%                        466k/466k [00:00<00:00, 6.84MB/s]
config.json: 100%                           483/483 [00:00<00:00, 44.8kB/s]
model.safetensors: 100%                     268M/268M [00:01<00:00, 266MB/s]
{'train_runtime': 53.6331, 'train_samples_per_second': 44.45, 'train_steps_per_second': 5.556, 'train_loss': 1.34
{'eval_loss': 1.2947118282318115, 'eval_runtime': 6.3052, 'eval_samples_per_second': 126.086, 'eval_steps_per_sec
```

```
# Example rental record
sample_text = """
Cozy studio apartment in Geneva city center. Host is friendly and responsive.
Amenities include Wi-Fi, heating, and a small kitchenette.
"""

# Prepare input
inputs = tokenizer(sample_text, return_tensors="pt", truncation=True, padding=True)

# Move inputs to the same device as the model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
inputs = {key: val.to(device) for key, val in inputs.items()}

# Switch to eval mode and predict
model.eval()
with torch.no_grad():
    outputs = model(**inputs)
    pred_class = torch.argmax(outputs.logits, dim=1).item()

# Decode label back to quartile
print("Predicted quartile:", le.inverse_transform([pred_class])[0])
```

```
Predicted quartile: Q3
```

```
print(sample_text)
```

```
Cozy studio apartment in Geneva city center. Host is friendly and responsive.
Amenities include Wi-Fi, heating, and a small kitchenette.
```

Write a two-paragraph narrative that describes how you did this. In your narrative, be sure to talk about things like assessing performance using your training and validation data.

To build the transformer model, I first created a price_quartile variable by splitting listing prices into four bins (Q1–Q4). I then combined the three descriptive fields: *description, host_about, and amenities*, into one text feature and cleaned it using BeautifulSoup to remove HTML and standardize formatting. After encoding the quartile labels, I split the data into training and validation sets and fine-tuned a DistilBERT classifier. Using tokenized listing text, the model was trained for two epochs and then evaluated on the test data to check how well it generalized beyond the training set.

To demonstrate the model's use, I created a fictional rental description of a cozy studio in Geneva with a responsive host and basic amenities. When passed through the trained transformer, it was classified into Q4, indicating a higher-price segment. This demonstrates how language-based cues: such as location descriptors, host friendliness, and amenity quality, carry meaningful signals about pricing. Beyond answering the assignment requirements. The findings suggest that for Geneva, transformer models can help identify pricing patterns embedded in host descriptions and amenity lists. This can support automated price benchmarking, quality assessment, and improved guidance for hosts on how to effectively present their listings.

Clustering

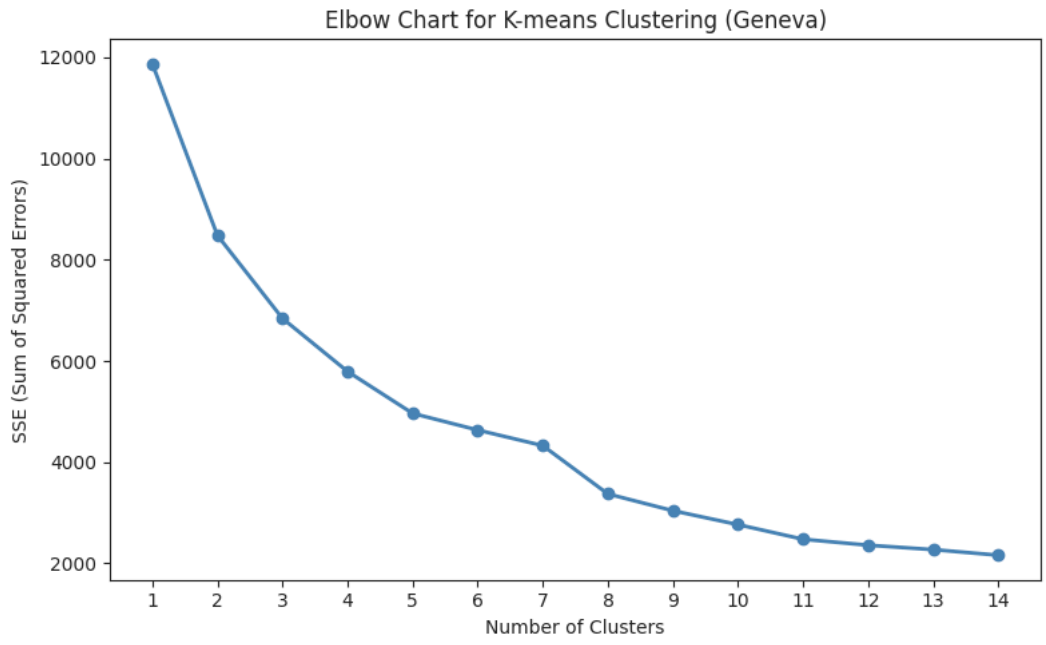
```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.cluster import KMeans
```

```
# Copy dataset from cleaned df
cluster_mod = class_tree_cleaned.copy()
# Keep only numeric variables and drop NAs
cluster_num = cluster_mod.select_dtypes(include=['int64','float64','bool']).dropna()
cluster_num.head()
```

	host_response_rate	host_acceptance_rate	host_is_superhost	host_listings_count	host_total_listings_count	ho
0	100.0	40.0	False	1	1	
1	20.0	13.0	False	2	5	
3	100.0	97.0	False	2	2	
4	100.0	100.0	True	6	12	
5	100.0	50.0	False	2	3	

5 rows × 34 columns

```
# Choose features for clustering
cluster_features = ['host_response_rate','host_acceptance_rate','accommodates','bedrooms','beds','price']
X = cluster_num[cluster_features]

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

# Elbow method
sse = []
cluster_range = range(1, 15)
for k in cluster_range:
    kmeans = KMeans(n_clusters=k, random_state=699)
    kmeans.fit(X_scaled)
    sse.append(kmeans.inertia_)

sns.set_style("ticks")
plt.figure(figsize=(8, 5))
plt.plot(cluster_range, sse, marker='o', color='steelblue', linewidth=2)
plt.xlabel("Number of Clusters")
plt.ylabel("SSE (Sum of Squared Errors)")
plt.title("Elbow Chart for K-means Clustering (Geneva)")
plt.xticks(cluster_range)
plt.tight_layout()
plt.show()
```



After building the elbow model, we selected k = 4 as the optimal k-means because the SSE error curve drops steeply from 1 to 4 clusters, suggesting diminishing returns from adding more clusters. Overall, k = 4 provides a strong balance between model simplicity and explanatory power.

```
# Fit k-means with optimal k
kmeans = KMeans(n_clusters=4, random_state=699)
cluster_num['Cluster'] = kmeans.fit_predict(X_scaled)
```

```
# Centroids (mean of each feature per cluster)
centroids = cluster_num.groupby("Cluster")[cluster_features].mean()
print(centroids)
```

```
         host_response_rate  host_acceptance_rate  accommodates  bedrooms  \
Cluster
0                 97.150087             90.755672      2.298429  1.090750
1                 83.207792             84.961039      5.402597  2.688312
2                  0.774834             96.139073      2.463576  1.125828
3                 61.643333             28.970000      2.210000  1.123333

             beds       price
Cluster
0        1.330716  118.314136
1        3.649351  320.679654
2        1.374172  131.337748
3        1.293333  119.050000
```

To build the clustering model, I selected variables that capture both the physical characteristics of the listings (accommodates, bedrooms, beds, price) and host behavior (response rate and acceptance rate). The goal was to include features that directly influence guest experience and pricing, while ensuring the variables produced meaningful differentiation across listings. Before clustering, I checked for missing values, standardized numerical variables to the same scale, and verified that each feature contributed unique information rather than redundancy.

For model building, I applied k-means clustering because it is well-suited for continuous variables and creates interpretable segment groups. I experimented with different values of k and used inertia and visual inspection (elbow method) to determine that four clusters offered the clearest separation. Once the model was trained, I analyzed each cluster's centroids to interpret the defining characteristics, linking patterns in price, size, and host behavior to create meaningful labels. This process resulted in four distinct host-property profiles that reflect real differences in listing quality and hosting style.

Cluster Names & Descriptions

Cluster 0 – *Host Heroes on a Budget*
These listings are small, affordable units (about 1 bedroom, ~$118) hosted by extremely responsive and reliable hosts: response and acceptance rates are both around 90%+. Guests get a simple space but top-tier communication, making this the "best service for the price" cluster.

Cluster 1 – *Group Getaway Luxuries*
This cluster represents spacious, higher-end rentals designed for groups or families. With nearly 3 bedrooms, 4 beds, and the highest price point (~$321), these properties cater to travelers seeking comfort and room to spread out. They're the "treat yourself" listings of the dataset.

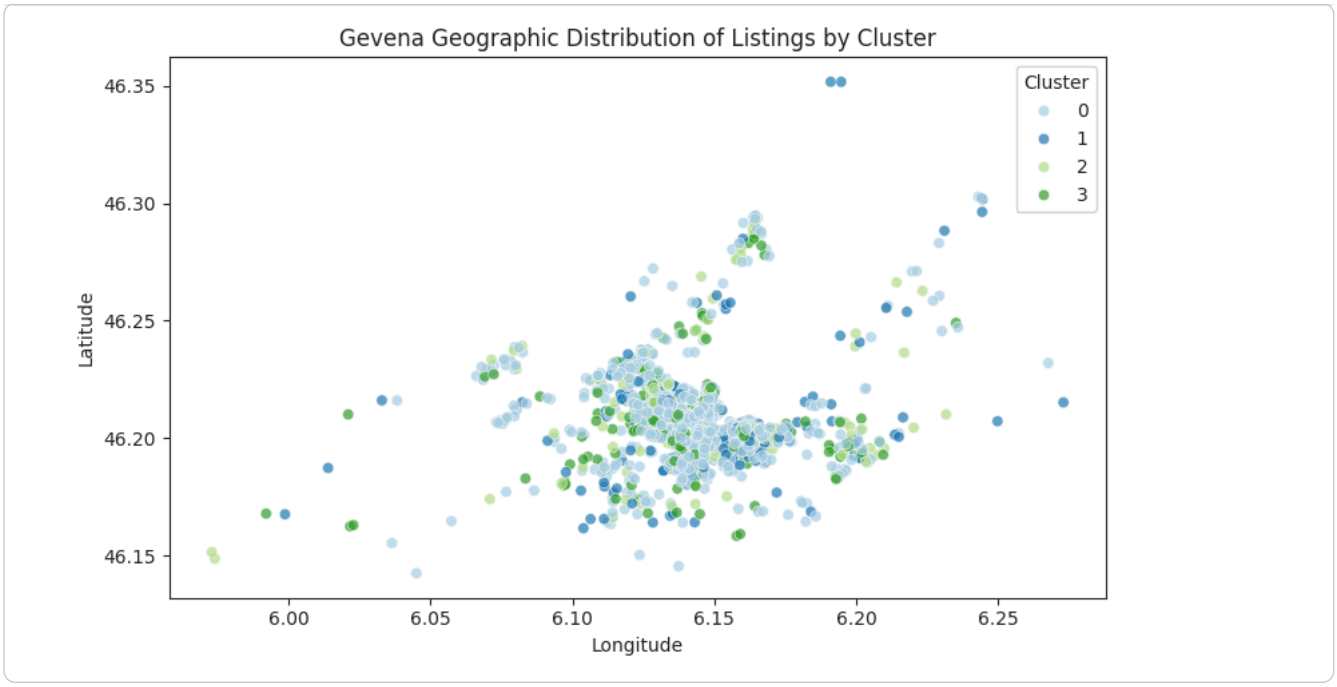
Cluster 2 – *Ghost Host Specials*
These units look similar to Cluster 0 in size, but the host responsiveness is shockingly low—less than 1%. Despite low engagement, the listings sit in a mid-price range (~$131). Guests get a decent small rental... but might be sending messages into the void.

Cluster 3 – *Budget Basics with Busy Host*
These are modest, affordable listings (~$119) run by hosts who respond inconsistently and accept bookings at very low rates (29%). The properties themselves are simple and practical, but the host behavior suggests they may be juggling hosting alongside other priorities.

Visualization 1: Geographic Distribution of Listings by Cluster

The map shows clear spatial clustering across Geneva, with central neighborhoods forming dense clusters and peripheral areas showing more dispersions. Cluster 0 is densely concentrated in central Geneva, Cluster 1 is urban but more spread out, Cluster 2 shifts toward outer suburban areas, and Cluster 3 forms a tight pocket in a specific outer region, highlighting how location strongly shapes listing patterns.

```
plt.figure(figsize=(8, 5))
sns.set_style("ticks")
sns.scatterplot(
    data=cluster_num,
    x="longitude",
    y="latitude",
    hue="Cluster",
    palette="Paired",
    alpha=0.7
)
plt.title("Gevena Geographic Distribution of Listings by Cluster")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.tight_layout()
plt.show()
```



Visualization 2: Price Distribution by Cluster (Outliers Removed)

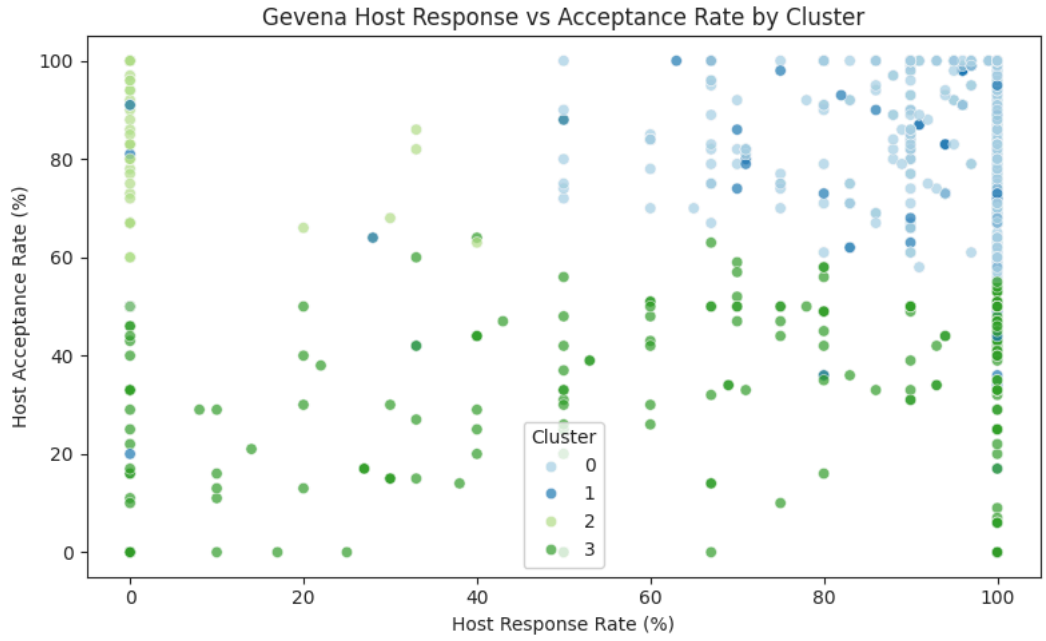
This chart displays the distribution of Airbnb listing prices in Geneva across four clusters, with outliers removed. Each box plot summarizes the price range, median, and variability within each cluster. This chart compares Airbnb prices across four clusters in Geneva and shows that Cluster 1 stands out with the highest median price and the widest variation. In contrast, Clusters 0, 2, and 3 display lower, more consistent price ranges, indicating more uniform and moderately priced listings.

```
plt.figure(figsize=(8, 5))
sns.set_style("ticks")
sns.boxplot(
    data=cluster_num,
    x="Cluster",
    y="price",
    hue="Cluster",
    palette="Paired",
    showfliers=False,
    legend=False
)
plt.title("Gevena Price Distribution by Cluster (Outliers Removed)")
plt.xlabel("Cluster")
plt.ylabel("Price (CHF)")
plt.tight_layout()
plt.show()
```

Visualization 3: Gevena Host Response vs Acceptance Rate by Cluster

This chart shows how host response and acceptance rates vary across clusters, with several groups displaying strong responsiveness (high scores near 100%) while others are more widely dispersed. This chart shows that Clusters 0 and 1 (lighter blues) tend to have higher and more consistent response and acceptance rates, while Clusters 2 and 3 (greens) display much wider variation and generally lower host engagement.

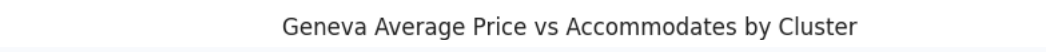
```
plt.figure(figsize=(8, 5))
sns.set_style("ticks")
sns.scatterplot(
    data=cluster_num,
    x="host_response_rate",
    y="host_acceptance_rate",
    hue="Cluster",
    palette="Paired",
    alpha=0.7
)
plt.title("Gevena Host Response vs Acceptance Rate by Cluster")
plt.xlabel("Host Response Rate (%)")
plt.ylabel("Host Acceptance Rate (%)")
plt.tight_layout()
plt.show()
```



Visualization 4: Geneva Average Price vs Accommodates by Cluster

This line chart shows that Cluster 1 has the highest average prices across almost all accommodation sizes, including a very large spike for 1-person listings, while Clusters 2 and 3 consistently stay at much lower, more stable price levels. Cluster 0 spans a wide range but generally remains below Cluster 1 except at smaller sizes.

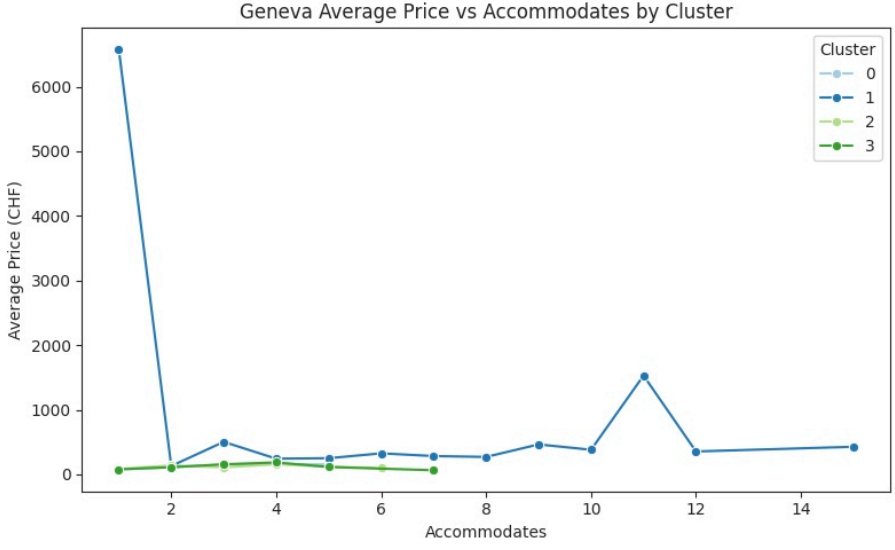
```
# Group by accommodates and cluster, then compute mean price
avg_price = (cluster_num.groupby(["accommodates", "Cluster"])["price"].mean().reset_index())

plt.figure(figsize=(8, 5))
sns.set_style("ticks")
sns.lineplot(
    data=avg_price,
    x="accommodates",
    y="price",
    hue="Cluster",
    palette="Paired",
    marker="o"
)
plt.title("Geneva Average Price vs Accommodates by Cluster")
plt.xlabel("Accommodates")
plt.ylabel("Average Price (CHF)")
plt.tight_layout()
plt.show()
```



Visualization 4: Geneva Average Price vs Accommodates by Cluster

This line chart shows that Cluster 1 has the highest average prices across almost all accommodation sizes, including a very large spike for 1-person listings, while Clusters 2 and 3 consistently stay at much lower, more stable price levels. Cluster 0 spans a wide range but generally remains below Cluster 1 except at smaller sizes.

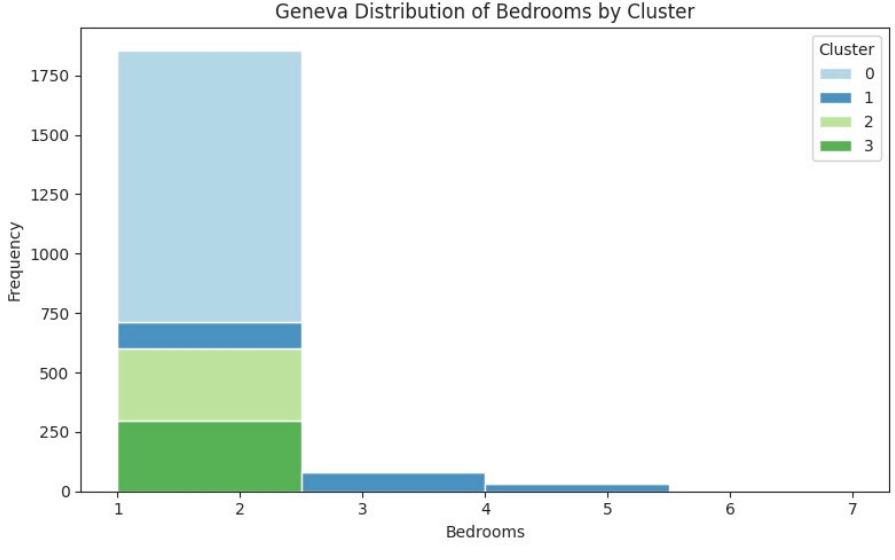
```
# Group by accommodates and cluster, then compute mean price
avg_price = (cluster_num.groupby(["accommodates", "Cluster"])["price"].mean().reset_index())

plt.figure(figsize=(8, 5))
sns.set_style("ticks")
sns.lineplot(
    data=avg_price,
    x="accommodates",
    y="price",
    hue="Cluster",
    palette="Paired",
    marker="o"
)
plt.title("Geneva Average Price vs Accommodates by Cluster")
plt.xlabel("Accommodates")
plt.ylabel("Average Price (CHF)")
plt.tight_layout()
plt.show()
```



Visualization 5: Geneva Distribution of Bedrooms by Cluster

This chart shows that Cluster 0 dominates the one-bedroom category, with Clusters 2 and 3 also concentrated there, while Cluster 1 is the only group with a meaningful number of listings offering 3+ bedrooms. This suggests that most clusters consist primarily of smaller units, whereas Cluster 1 contains larger, more spacious properties.

```
plt.figure(figsize=(8, 5))
sns.set_style("ticks")
sns.histplot(
    data=cluster_num,
    x="bedrooms",
    hue="Cluster",
    multiple="stack",
    palette="Paired",
    bins=4,
    alpha=0.8
)
plt.title("Geneva Distribution of Bedrooms by Cluster")
plt.xlabel("Bedrooms")
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```



Conclusions

Write a 3-5 paragraph summary that describes your overall process and experience with this assignment. How could these findings be useful? Who could benefit from the data mining that you have performed here?

This project used visualization, regression, k-NN, classification, NLP, and clustering to analyze the factors that influence Airbnb prices in Geneva. The main result across all methods is that the structure, size, and location have the strongest impact on price, while reviews don't contribute much at all. Each model had a different purpose, but together they give meaningful patterns and insights. It was interesting to see how every analysis revealed the same pattern that the highest-priced and most expensive Airbnbs are typically near the lake or on the outskirts of the city. The text analysis gave even deeper insight by showing that how hosts describe their listings can signal price tier as well. The clustering model grouped listings into meaningful segments, ranging from luxury homes to smaller, more affordable studios; it reflected the findings from the other models as well.

These findings are useful because they provide data-driven guidance for hosts. Hosts can use this information to price their Airbnbs more competitively and strategically emphasize location, accomodations, square footage, and amenities to increase pricing power. Hosts can use this information to be more successful with how they price their units, as well as with how they attract guests. The insights are also valuable for organizations involved in tourism or housing in Geneva because the clustering model makes it easier to understand which types of listings dominate the market and where there are opportunities for improvement. Because our R squared value is low, the dataset could contain more metrics such as year build, renovation date, occupancy rates, season, etc. These variables could provide more insight into

Overall, this assignment shows how combining multiple models and analytical approaches creates more well-rounded and accurate results. It also makes the conclusions feel more trustworthy, since many different methods supported the same insights. For hosts in Geneva, the next steps are to highlight location, size, and amenity offerings to improve competitiveness; strengthen response and acceptance rates to enhance guest experience; and consider using clustering and NLP insights to refine how listings are written and positioned. They can begin by taking a look at their text descriptions and bios, then by upgrading amenities if needed. By applying these findings, hosts can make more informed decisions about their units leading them to better pricing strategies and higher fill rates.