CS5764 Data Visualization

RentalGPT Dash Final Project Report

*****

Student:

Minh T. Nguyen, mnguyen0226@vt.edu

Instructor:

Dr. Reza Jafari

Institution:

Virginia Polytechnic Institute and State University

Location: Falls Church, VA

Date:

December 6th, 2023

Figures

ABSTRACT
This final project report presents the data analysis, data visualization, and Dash application development called "RentalGPT", an interactive, user friendly dashboard that provides services to multiple stakeholders. Using the "Two Sigma Connect: Rental Listing Inquiries" dataset collected from Kaggle, we can do in-depth data analysis, interactive data visualization, and an app that has predictive analytics and virtual assistance (Two Sigma Connect).

The RentalGPT app can be accessed here. The app can be deployed both on Google Cloud Platform or Virginia Tech's Kubernetes Rancher server. The source code with detailed instructions can be found here.

INTRODUCTION

To accomplish this project, I adopted the Data Science Life Cycle scheme learned from experience including six steps: Business Understanding and Data Acquisition, Data Processing, Exploratory Data Analysis, Feature Engineering, Machine Learning Modeling, and Production. The tools and techniques for each step are summarized in Figure 1.

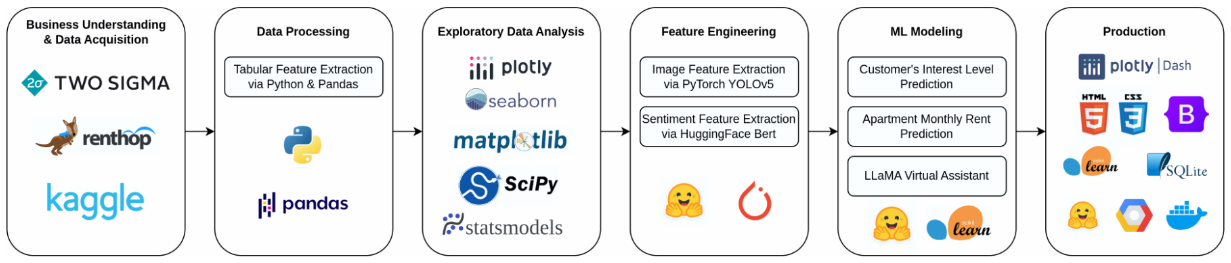

Business Understanding & Data Acquisition
TWO SIGMA
renthop
kaggle
Data Processing
Tabular Feature Extraction via Python & Pandas
pandas
Exploratory Data Analysis
plotly
seaborn
matplotlib
SciPy
statsmodels
Feature Engineering
Image Feature Extraction via PyTorch YOLOv5
Sentiment Feature Extraction via HuggingFace Bert
ML Modeling
Customer's Interest Level Prediction
Apartment Monthly Rent Prediction
LLaMA Virtual Assistant
learn
Production
plotly | Dash
HTML
CSS
learn
SQLite

Figure 1. Data Science Life Cycle - RentalGPT Project.

The report is divided into seven sections: "Introduction" will mention the data science life cycle I adopted for this project; "Data Description" will provide an overview of the dataset; "Data Processing" will discuss outlier processing; "Data Analysis" will go in-depth in the lessons learned from the provided data; "Data Transformation" will go about the application of computer vision and sentiment analysis; "Dashboard" will illustrate the sections in the developed RentalGPT app, data-driven solutions, enabling renters, landlords, investors, and real estate agents to optimize their decision; "Conclusion" will provide lessons learned from this project.

DATA DESCRIPTION

The "Two Sigma Connect: Rental Listing Inquiries" dataset is collected from Two Sigma's Renthop platform, a collection of rental properties listed in New York City (Two Sigma Connect: Rental Listing Inquiries | Kaggle. (2023)). The data repository provides a tabular dataset of the listing's features with some property images collected in 3 months (April, May, June). The research question the dataset was trying to answer is: Given a feature of a property, can we predict the customer's interest level? Such rich datasets can be useful for predictive modeling, data analysis, and data visualization tasks.

The dataset met the project requirements for several reasons:

- The size is 123001 (49352 in train.json and 74659 in test.json; although I only use train.json for this project).
- There are 6 numerical features with 3 categorical features provided. However, through feature engineering, we can extract more.
- The dataset contains a time-series aspect (dates).
- The dataset can be downloaded from a non-classified database (Kaggle).

By default, the independent variables are:

- Numerical: price, bathrooms, bedrooms. Features such as chair_count and oven_count can be extracted from images.
- Categorical variables: building_id, manager_id, street_address. Features such as has_laundry and has_dishwasher can be extracted from the apartment's description.
- Geospatial features: longitude and latitude.

By default, the dependent variable is interest_level (low, medium, high).

This dataset is significant in the real estate industry as investors can provide services such as property listing, interest-level prediction, rental cost prediction, or urban planning.

DATA PROCESSING

This section will discuss the methods for outlier detection and removal.
The code can be found in notebook/eda.ipynb and eda/eda.py.

Outlier Detection & Removal

Using the scatter plot, we can see the data points concentrate mostly in the bottom indicating the prices are mostly low. There are also extreme prices. The outliers can make the dataset skew which can make the predictive models developed later on inaccurate.

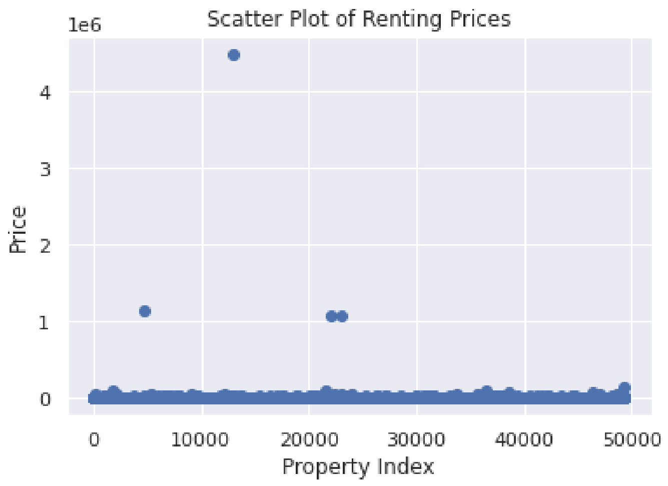


Figure 2. Scatter Plot of Renting Price

Here, we can see there are around 3-4 data points that are outliers. It is intuitive that the price of the apartment is a numerically important factor in determining the interest level, thus, we definitely want to remove them. The boxen plot gives a detailed view of the distribution's shape and the presence of the outliers. The majority of the data is concentrated at the lower end of the price spectrum

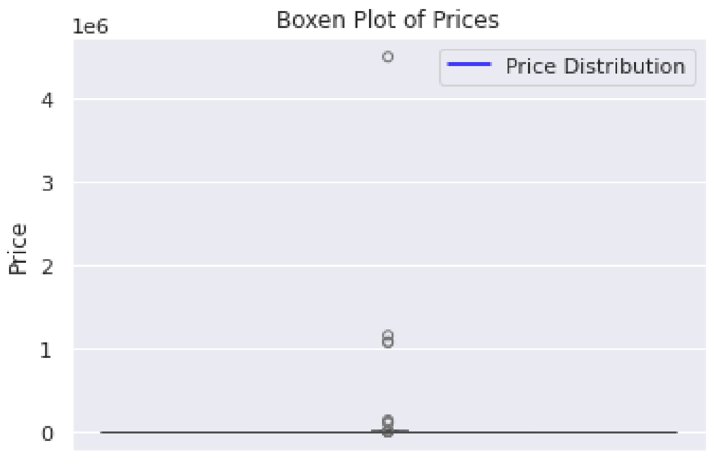


Figure 3. Boxen Plot of Renting Price

The scatter plot now shows rental prices without extreme outliers, providing a clearer view of the main data distribution. It seems most rental prices are clustered within a lower price range, with a gradual decrease in frequency as the price increases. Such a dataset is much better for predictive modeling.

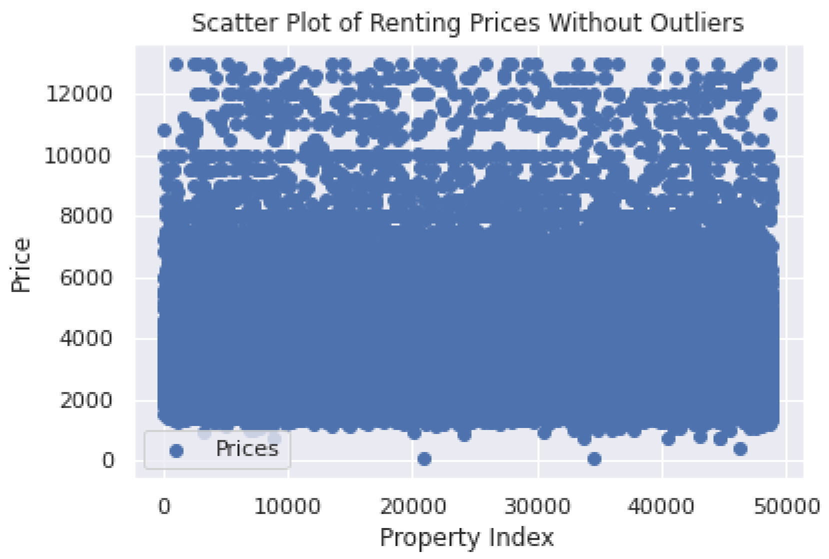


Figure 4. Scatter Plot of Renting Price

The boxen plot of rental prices without outliers displays a more refined distribution of the data. There's a gradual increase in rental prices with a few higher-priced rentals extending the upper whiskers of the plot. The median price is likely within the lower range, indicated by the line within the central box.

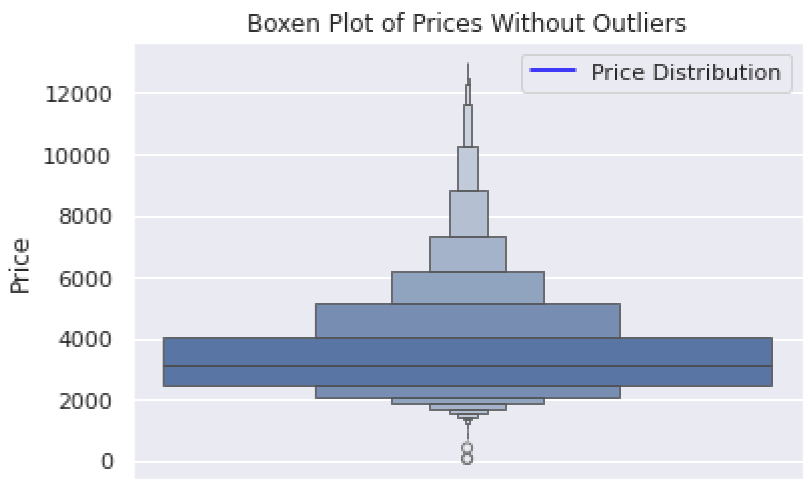


Figure 5. Boxen Plot of Renting Price

DATA ANALYSIS

This section will discuss PCA, Normality Test, Heatmap, Pearson Correlation, Statistical Test, KDE, and in-depth data analysis/visualization.

Principle Component Analysis

From the results, we get the % as the variance of the data. This means that with the first 11 features (principal components), we will capture 95.75% of the data variance. Thus, we don't sacrifice much when remove the last 3 features.

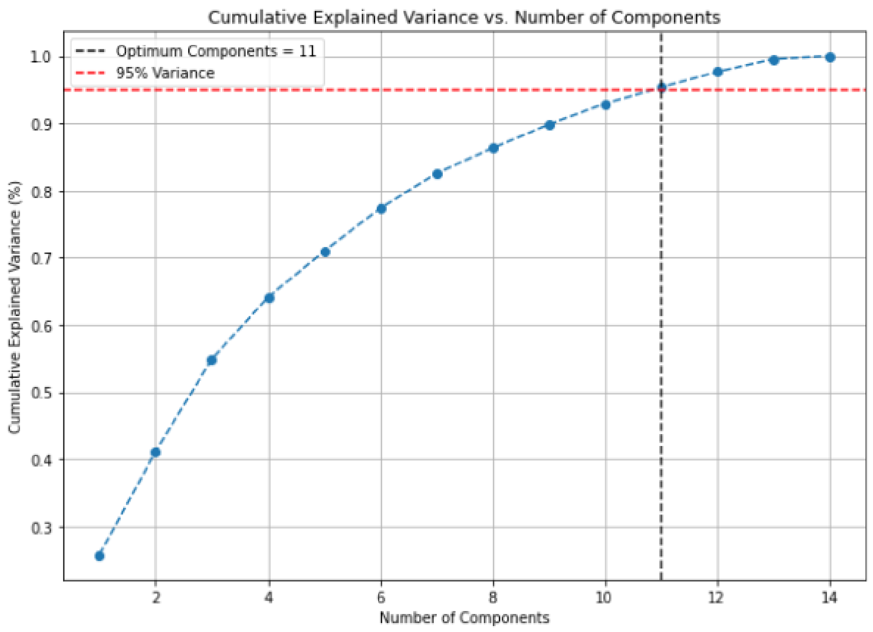
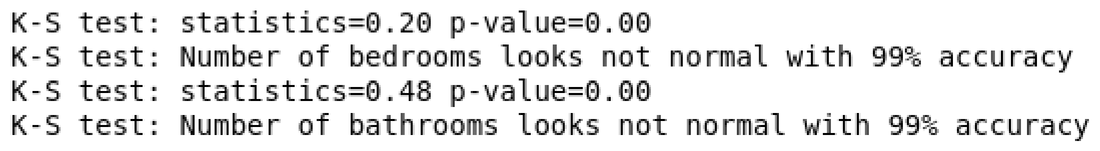


Figure 6. PCA

Normality Test

By applying all three normality tests (K-S, Shapiro-Wilk, and D'Agostino's K^2), we can provide the analysis:



```
K-S test: statistics=0.20 p-value=0.00
K-S test: Number of bedrooms looks not normal with 99% accuracy
K-S test: statistics=0.48 p-value=0.00
K-S test: Number of bathrooms looks not normal with 99% accuracy
```

The K-S test results indicate that both features do not follow the Gaussian distribution as the statistics are 0.2 and 0.48 respectively with p-values are 0, which is less than the expected 0.05 found in the Gaussian-distributed dataset.

```
Shapiro-Wilk test: statistics=0.90 p-value=0.00
Shapiro-Wilk test: Number of bedrooms looks not normal with 99% accura
cy
Shapiro-Wilk test: statistics=0.52 p-value=0.00
Shapiro-Wilk test: Number of bathrooms looks not normal with 99% accur
acy
```

The Shapiro-Wilk test results indicate that both features do not follow the Gaussian distribution as the statistics are 0.9 and 0.52 respectively with p-values are 0, which is less than the expected 0.05 found in the Gaussian-distributed dataset.

```
D'Agostino's K^2 test: statistics=1497.85 p-value=0.00
D'Agostino's K^2 test: Number of bedrooms looks not normal with 99% ac
curacy
D'Agostino's K^2 test: statistics=23495.67 p-value=0.00
D'Agostino's K^2 test: Number of bathrooms looks not normal with 99% a
ccuracy
```

The D'Agostino's K^2 test results indicate that both features do not follow the Gaussian distribution as the statistics are 1497.5 and 23495.67 respectively with p-values are 0, which is less than the expected 0.05 found in the Gaussian-distributed dataset.

Heatmap & Pearson Correlation

The heatmap displays the correlation matrix for bathrooms, bedrooms, and prices. The values, ranging from 0.52 to 0.67, indicate a moderate positive correlation between these features. This means as the number of bedrooms or bathrooms increases, the price tends to increase as well. There is a strongest correlation between price and the number of bathrooms, suggesting that the number of bathrooms is a more significant factor in determining the price than the number of bedrooms.

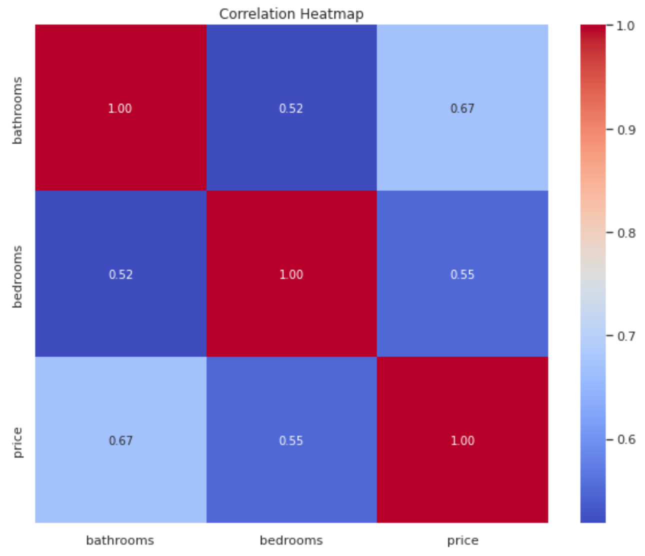
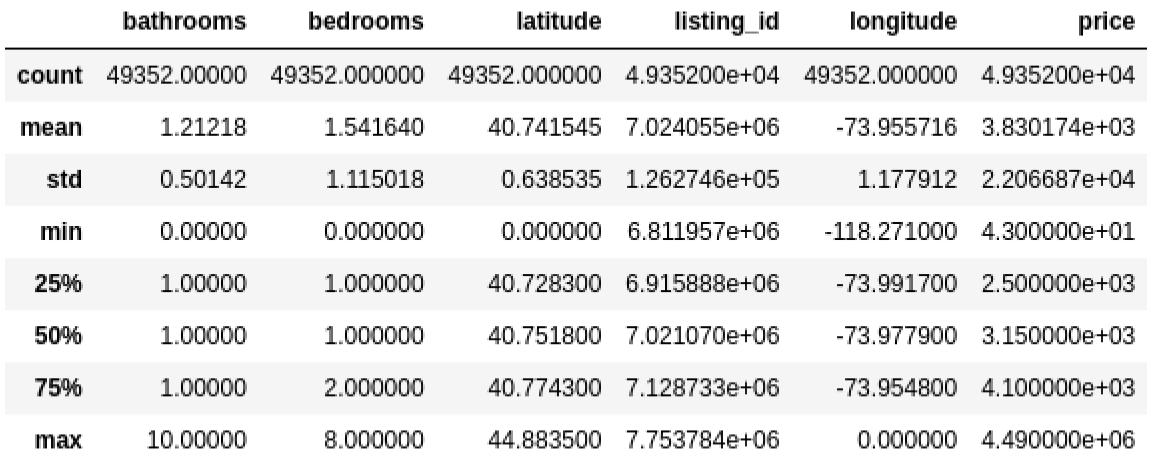


Figure 7. Heatmap and Pearson correlation.

Statistical Analysis and Kernel Density Estimation

<u>Statistical Analysis</u>

On average, properties have around 1.2 bathrooms and 1.5 bedrooms, suggesting a mix of single and small-family accommodations. The mean price of listings is approximately $3,830, which is indicative of a mid-to-high rental market. The standard deviations for bathrooms and bedrooms are low, while the price's higher standard deviation suggests a wider range of rental costs, possibly due to varied locations.



	bathrooms	bedrooms	latitude	listing_id	longitude	price
count	49352.00000	49352.000000	49352.000000	4.935200e+04	49352.000000	4.935200e+04
mean	1.21218	1.541640	40.741545	7.024055e+06	-73.955716	3.830174e+03
std	0.50142	1.115018	0.638535	1.262746e+05	1.177912	2.206687e+04
min	0.00000	0.000000	0.000000	6.811957e+06	-118.271000	4.300000e+01
25%	1.00000	1.000000	40.728300	6.915888e+06	-73.991700	2.500000e+03
50%	1.00000	1.000000	40.751800	7.021070e+06	-73.977900	3.150000e+03
75%	1.00000	2.000000	40.774300	7.128733e+06	-73.954800	4.100000e+03
max	10.00000	8.000000	44.883500	7.753784e+06	0.000000	4.490000e+06

Figure 8. Statistical Analysis

<u>Kernel Density Plot</u>

The KDE plot with the histogram shows the price distribution of rental listings. The distribution is right-skewed, with a peak indicating that most of the rentals are in the lower price range. The mean (red) is higher than the median (blue), which is typical for a right-skewed distribution, indicating that higher-priced rentals pull the average up.

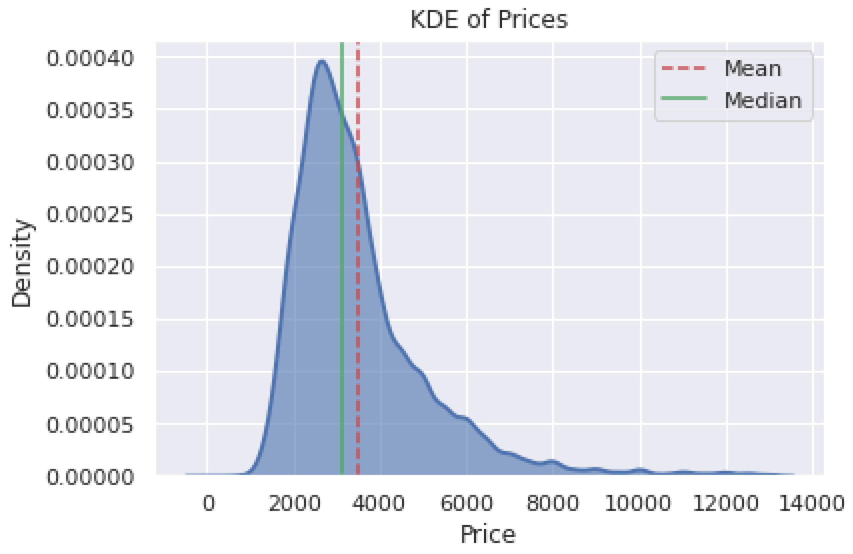


Figure 9. KDE of Prices

Static Data Visualization & Analysis

<u>Line Plot</u>

The line plot shows the price trend over time suggesting that rental prices fluctuate but do not exhibit a clear long-term trend in the given timeframe, which is sensible as our dataset is only 3-month long. There are spikes in rental prices that could correspond to specific events or seasonal demand changes. However, since we don't have an entire year dataset, it's hard to determine the seasonality of the price trends for New York city apartments.

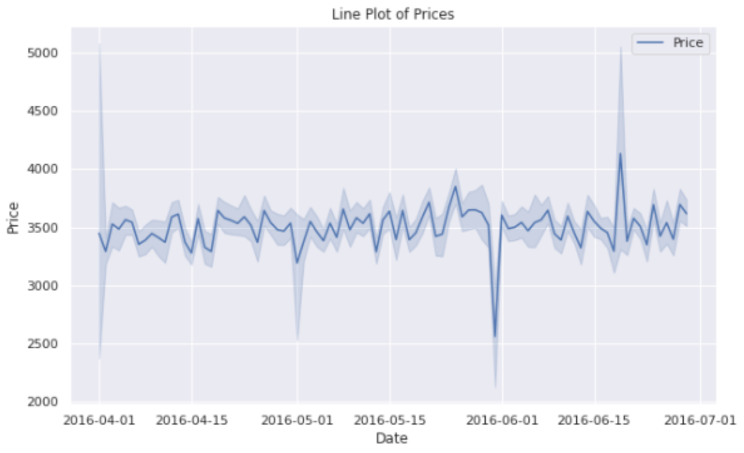


Figure 10. KDE of Prices

<u>Bar Plot</u>

The bar chart shows the frequency of listings by the number of bedrooms. One-bedroom and two-bedroom apartments are the most common, with a significant drop-off in the number of listings as the number of bedrooms increases. Ranked 3rd is the studio. The data suggests that single individuals or small families might be the primary target demographic for the rental market in New York city. Properties with more than three bedrooms are considerably less common, which may indicate a lower demand or a smaller supply of larger apartments as well.

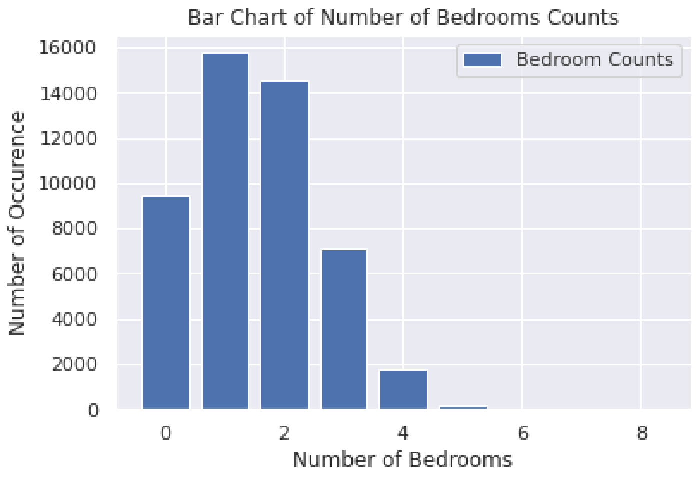


Figure 11. Bar Chart of Number of Bedrooms Counts

The bar chart indicates the number of bathrooms in listings shows that one-bathroom units are the most common, followed by two-bathroom units. Three or more bathrooms are relatively rare. This pattern suggests that the housing market in New York city primarily adjusts to singles and small families. Properties with a higher number of bathrooms are less common and could potentially offer to a luxury market segment.

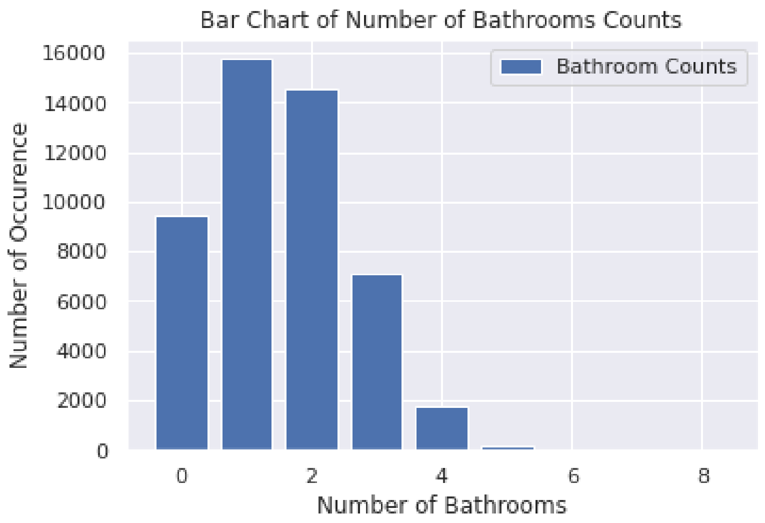


Figure 12. Bar Chart of Number of Bathrooms Counts

<u>Count Plot</u>

The bar chart indicates the counts of interest levels for rental listings throughout different hours of the day. There is a notable peak in activity during the early hours (likely around midnight), with a high volume of low-interest levels. Interest levels are low during the early morning and start to rise again during work hours. Meanwhile, interest is high during the night indicating serious renters are active.

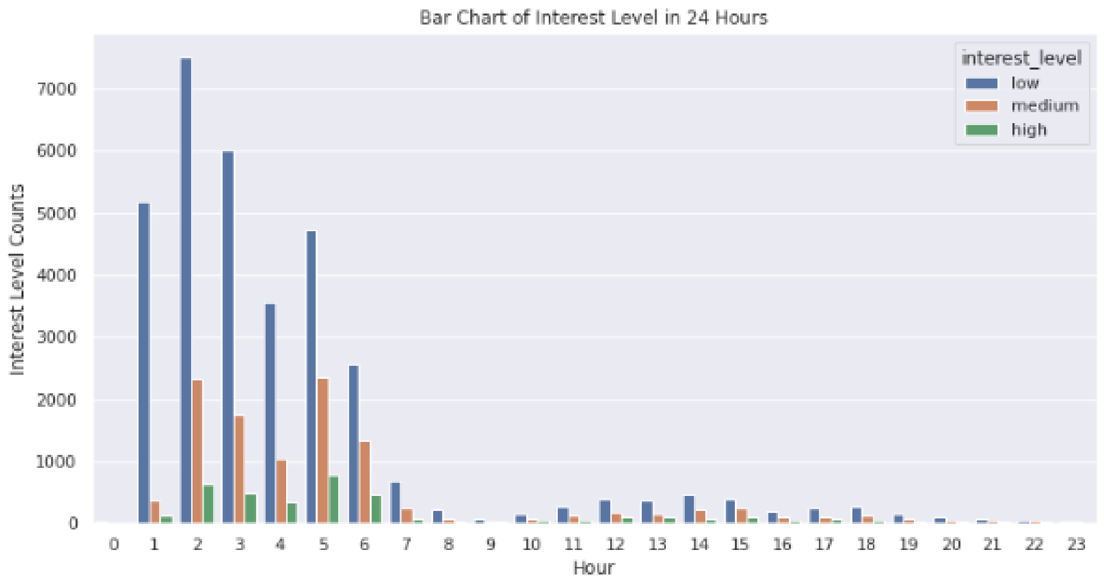


Figure 13. Count Plot of Interest Level in 24 Hours

The bar chart shows the distribution of interest levels for rental listings over a year, with only one year displayed (2016). The majority of listings have a low-interest level, followed by medium and then high.

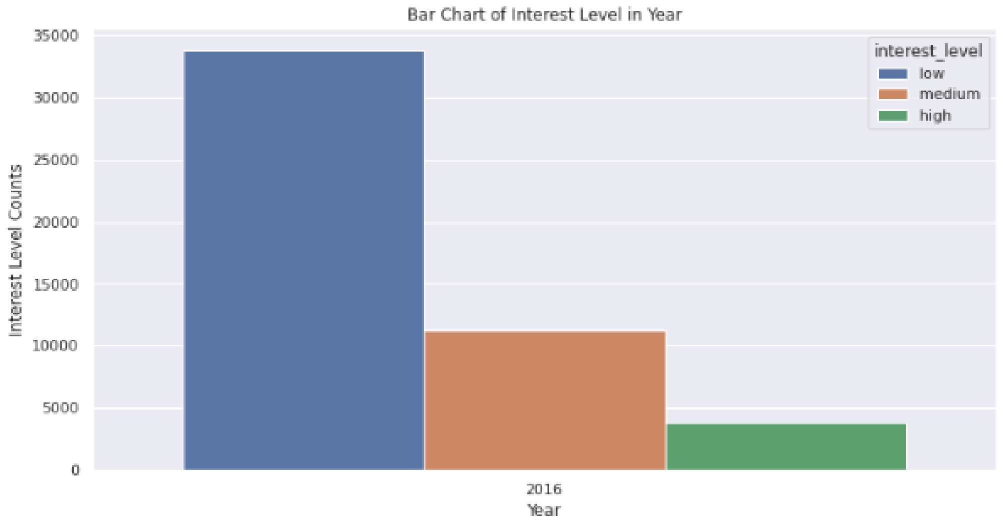


Figure 14. Count Plot of Interest Level in Year

The bar chart compares the interest level counts across three months, showing a clear summer seasonal trend. Interest levels, both low and medium, are highest in June and lower in the preceding months. The high-interest levels do not show a significant monthly variation, suggesting that high-interest properties are consistently sought after, regardless of the season. This trend is common in real estate markets due to various factors such as the end of the school year and the peak moving season (internship, co-op,...).

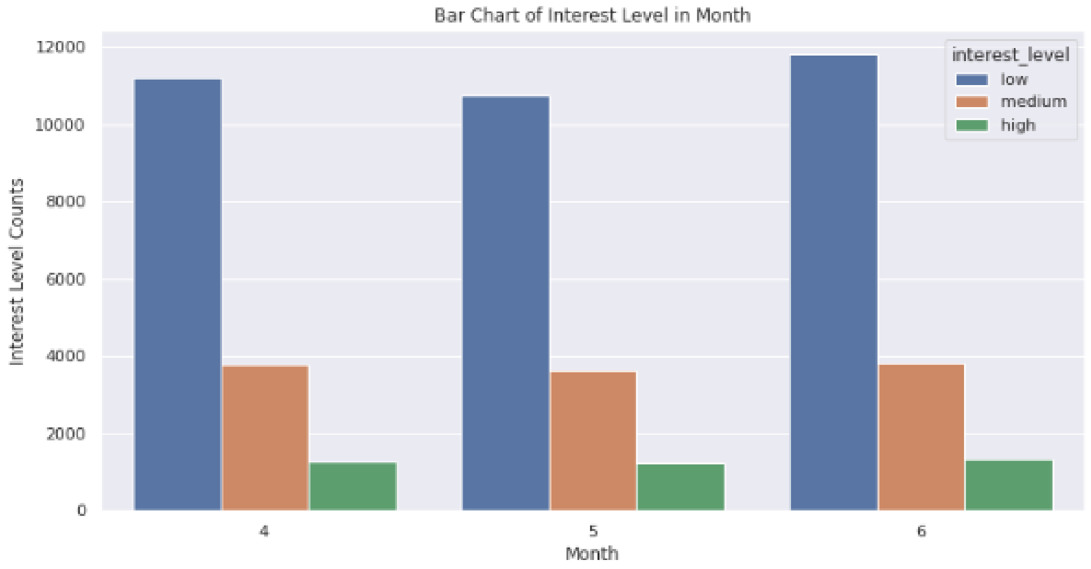


Figure 15. Count Plot of Interest Level in Month

The bar chart indicates daily interest level counts within a month. Interest levels fluctuate throughout the month, with some days showing notable peaks in medium and low interest. However, it seems like within the summer, the interest levels are even throughout the days in a month.

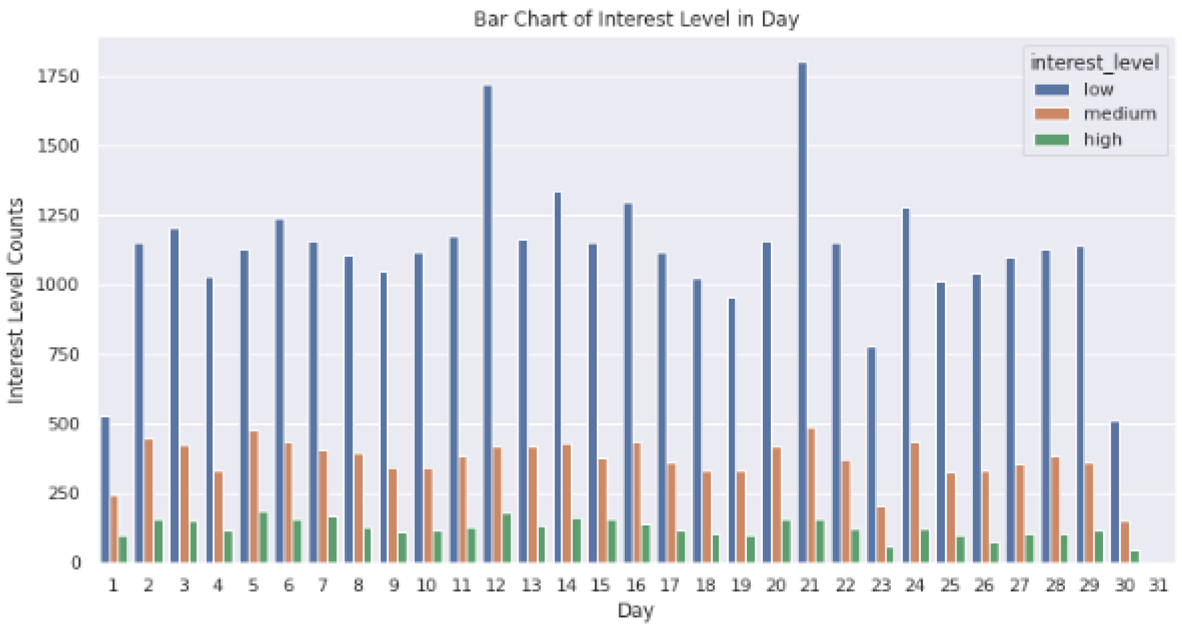


Figure 16. Count Plot of Interest Level in Day

The bar chart shows the interest level counts across weekdays. It shows the lowest levels of interest typically on the first day of the week and higher interest levels in the middle of the week. The trend suggests that renters might be more actively looking for listings during mid-week rather than at the beginning or end of the week.

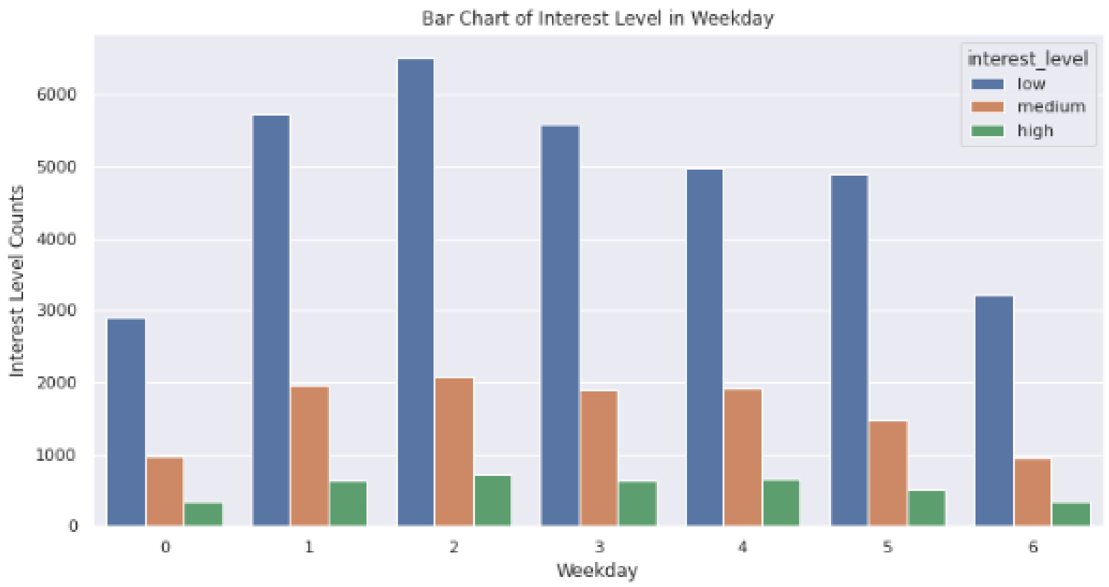


Figure 17. Count Plot of Interest Level in Weekday

The bar chart indicates the interest level counts segmented by quarters. With only the second quarter (summer) shown, there's a significant number of listings with low interest, a moderate amount with medium interest, and relatively few with high interest.

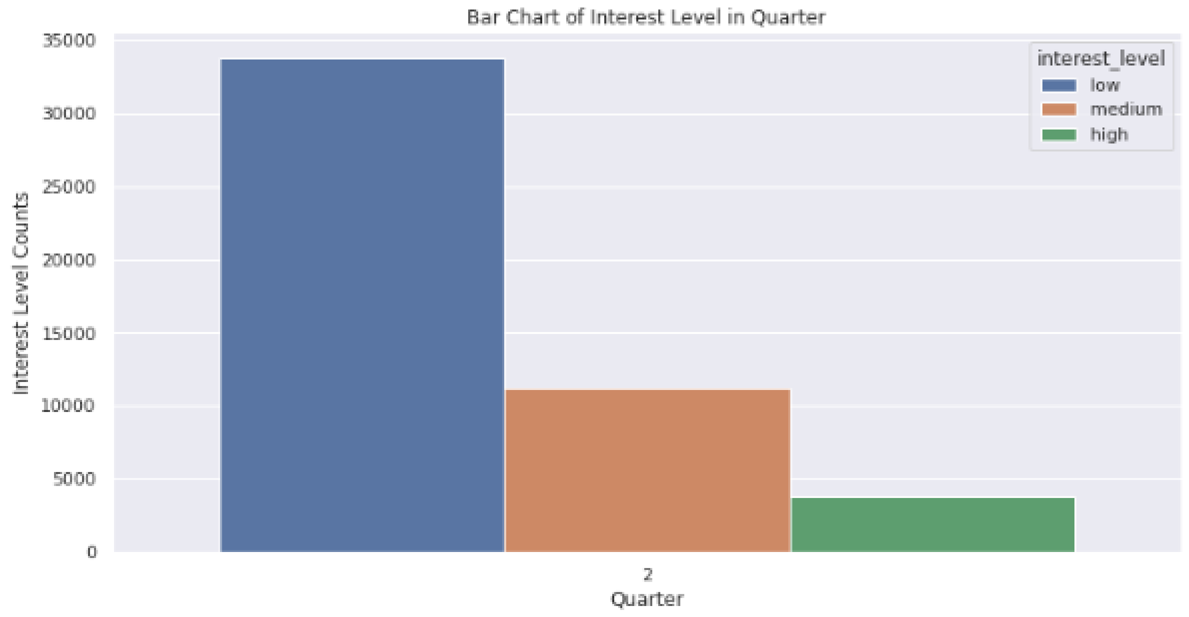


Figure 18. Count Plot of Interest Level in Quarter

<u>Pie Chart</u>

The pie chart presents the distribution of interest levels across rental listings. A large majority of the listings have a low-interest level, while a smaller fraction have medium interest, and only a very small percentage show high interest. This distribution could be due to various factors like price, location, features, or listing quality. The low percentage of high-interest listings may indicate a competitive market where only a few listings stand out enough to attract a high volume of potential renters in New York city.

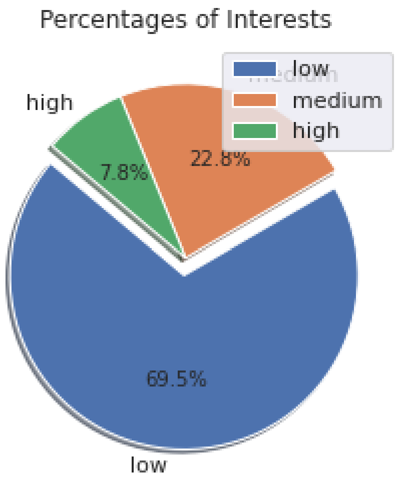


Figure 19. Pie Chart of Percentage of Interests

Distribution Plot

The distribution plot of rental prices, with a KDE overlay, indicates that the distribution of prices is right-skewed, meaning there are a number of higher-priced listings pulling the mean above the median. The median price is a better indicator of the central tendency for this dataset due to the skew. The skewness in the price distribution suggests that while most of the rentals are at a lower price point, there are enough premium-priced rentals to skew the average price higher.

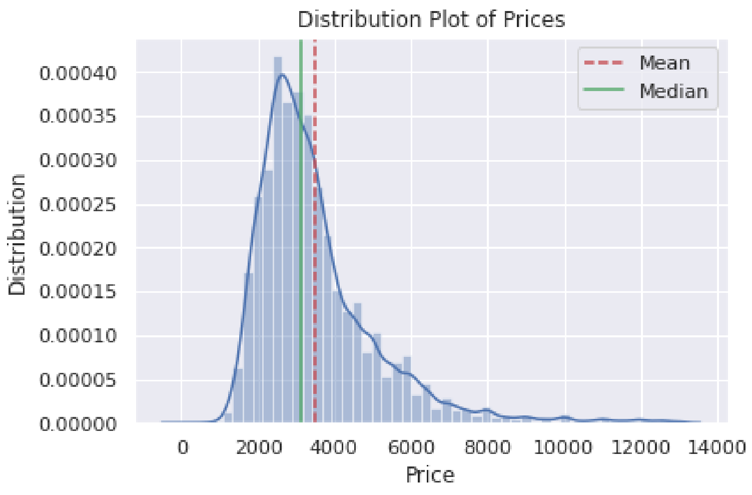


Figure 20. Distribution Plot of Prices

Pair Plot

The pair plot indicates relationships between the number of bathrooms, bedrooms, and price. There is a positive correlation between the number of bathrooms and bedrooms, which is intuitive as larger homes tend to have more of both. The scatter plots also suggest a positive correlation between price and both the number of bedrooms and bathrooms, indicating that as the count of these features increases, so does the price.

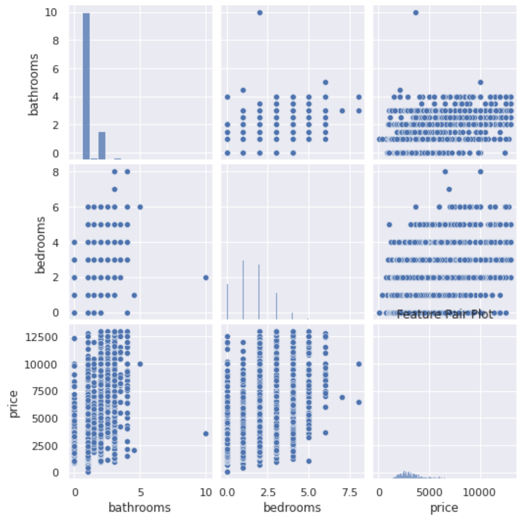


Figure 21. Pair Plot of Bathrooms, Bedrooms, and Prices

<u>Histogram Plot with Kernel Density Estimation</u>

The KDE plot with the histogram shows the price distribution of rental listings. The distribution is right-skewed, with a peak indicating that most of the rentals are in the lower price range. The mean (red) is higher than the median (blue), which is typical for a right-skewed distribution, indicating that higher-priced rentals pull the average up.

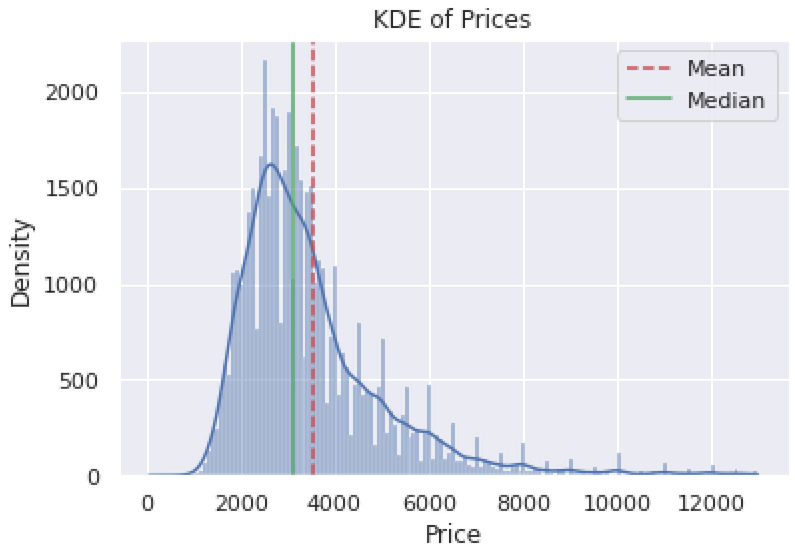


Figure 22. KDE of Prices

<u>QQ Plot</u>

The QQ plot of prices shows that the distribution of prices does not perfectly follow a normal distribution, as the points does not directly fit with the red line, especially at the lower and higher quantiles. This suggests that even after a log transformation (a non-linear transformation), there are extreme values in the data. Indeed that 'prices' does not follow the Gaussian (normal) distribution.

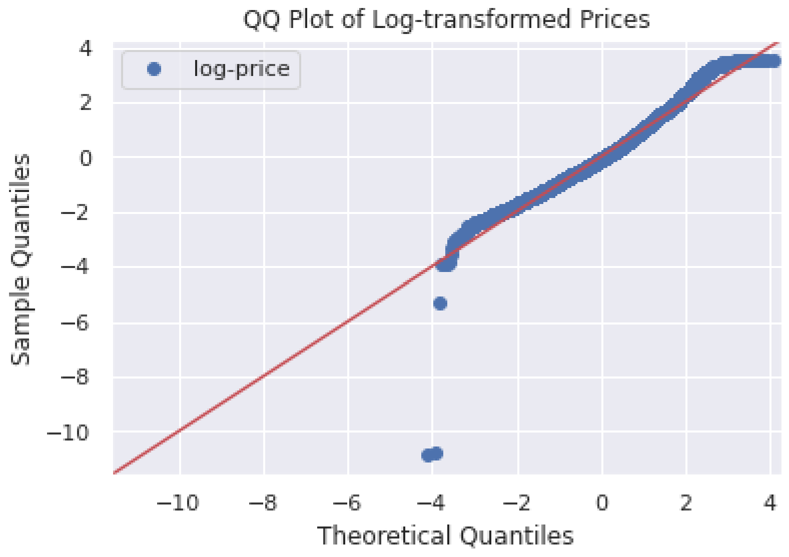


Figure 23. QQ Plot of Log-transformed Prices

Regression Plot with Scatter Representation and Regression Line

The regression plot with scatter representation shows a positive correlation between the number of bathrooms and bedrooms in rental listings. The trend line suggests that as the number of bathrooms increases, the number of bedrooms typically increases as well. Most data points cluster at the lower end of both axes, indicating that listings with fewer bedrooms and bathrooms are more common.

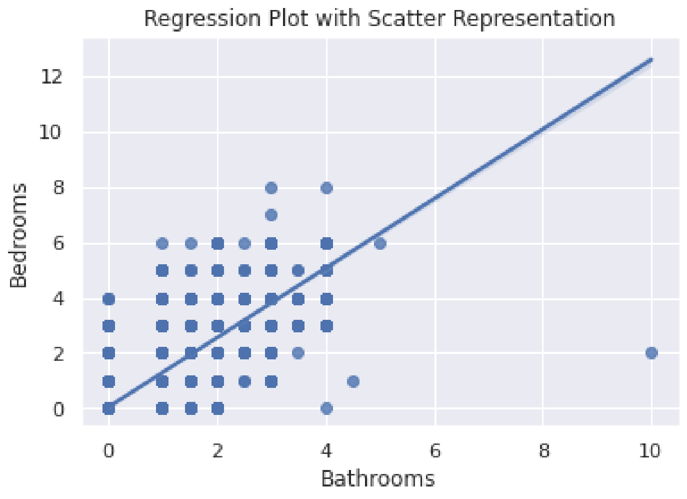


Figure 24. Regression Plot with Scatter Representation

The regression plot shows a positive correlation between the number of bathrooms in a property and its price. As the number of bathrooms increases, the price also tends to rise. However, the scatter plot reveals that most data points are concentrated at the lower end of the bathroom count, with prices showing a wide range even for properties with the same number of bathrooms. Most data points cluster at the lower end of both axes, indicating that listings with fewer price and bathrooms are more common.

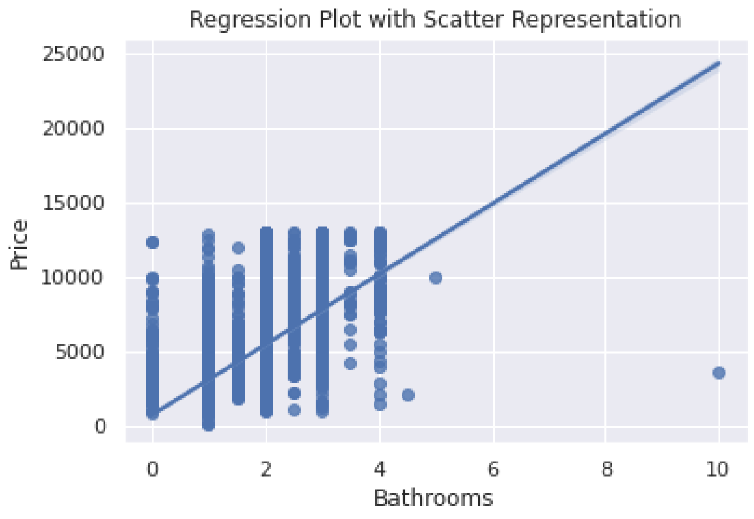


Figure 25. Regression Plot with Scatter Representation

The regression plot suggests a positive correlation between the number of bedrooms and the price of rental properties. The general trend shows that as the number of bedrooms increases, so does the price. However, there is significant variability in price within properties with the same number of bedrooms, indicating other factors also play a crucial role in determining rental prices, unlike between price and bathrooms.

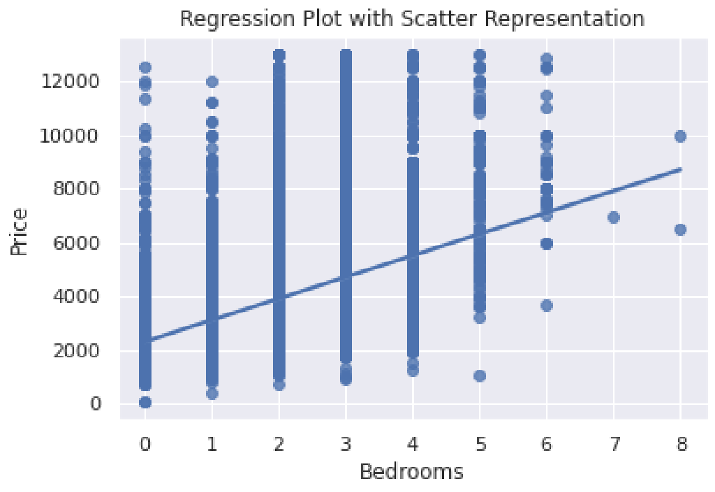


Figure 26. Regression Plot with Scatter Representation

Area Plot

The area plot reveals fluctuations in average listing prices over time across different interest levels. High-interest properties generally command higher prices, while low-interest properties are clustered at lower price points. The data indicates variability in the rental market, with occasional peaks suggesting the entry of premium listings or seasonal demand surges. The overlap between medium and high interest levels points to the price being one of several factors influencing interest. Without having to analyze the annual dataset, it's hard to determine the seasonality of low, medium, and high interest levels.

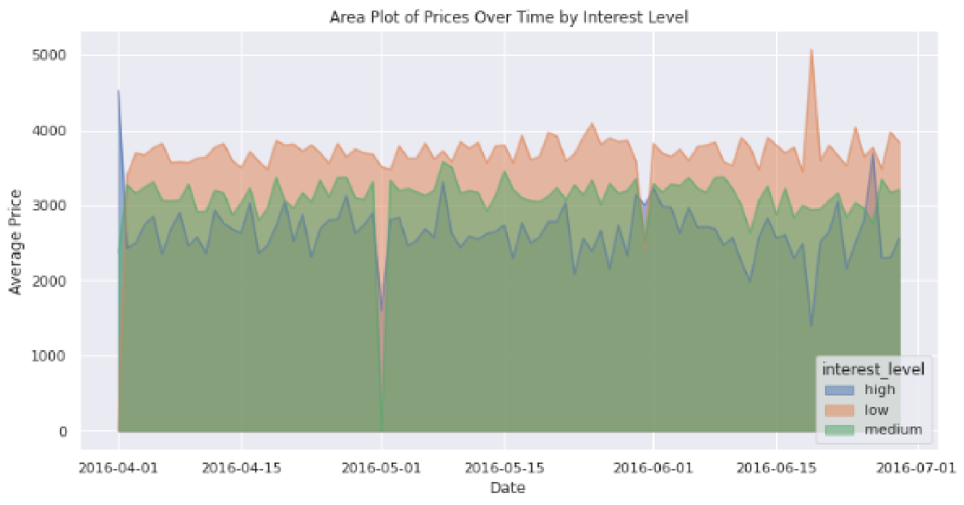


Figure 27. Area Plot of Prices Over Time by Interest Level

Violin Plot

The violin plot shows the distribution of prices at different interest levels. It demonstratd that lower-priced listings have a wider range of prices and are more commonly found at the 'low' interest level, while the 'high' interest level has a narrower price distribution, indicating that high-interest listings are more consistently priced. The 'medium' interest level shows a price distribution that is somewhat between the 'low' and 'high' interest levels.

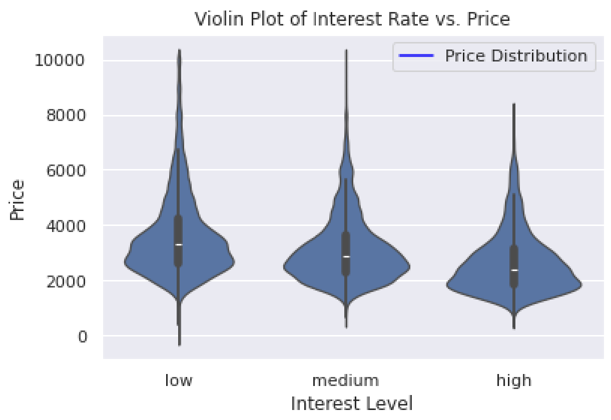


Figure 28. Violin Plot of Interest Rate vs. Price

Joint Plot

The joint plot showcases the relationship between the price, the number of bedrooms, and the interest level of listings. The scatter points are color-coded by interest level, indicating that listings with high interest are not necessarily those with the highest price or most bedrooms. It suggests that while there might be a trend for higher prices with more bedrooms, the interest level is influenced by (known/unknown) factors beyond just these two.

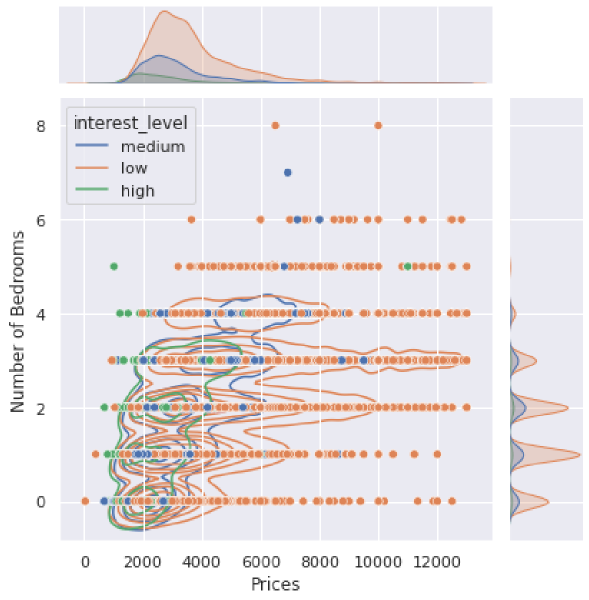


Figure 29. Joint Plot between Number of Bedrooms and Prices

The joint plot indicates a positive relationship between the number of bathrooms in a property and its price, color-coded by interest level. While higher-priced listings tend to have more bathrooms, the interest level doesn't seem exclusively higher for these properties, which suggests that (known/unknown) factors other than the number of bathrooms and price contribute to the interest level.

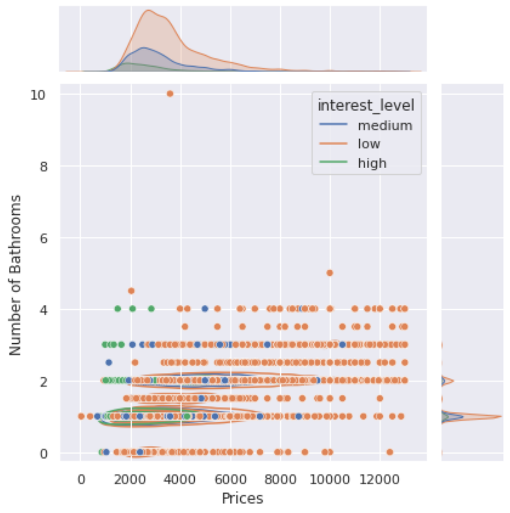


Figure 30. Joint Plot between Number of Bathrooms and Prices

<u>Rug Plot</u>

The rug plot visualizes the distribution of rental prices along the price axis. Each tick represents an individual listing's price. The concentration of ticks at the lower end of the price scale indicates a higher density of more affordable listings. The mean and median are marked, showing the mean to the right of the median due to the right-skewed distribution of prices. This suggests that while the majority of listings are unexpensive, there are enough high-priced listings to skew the average price higher.

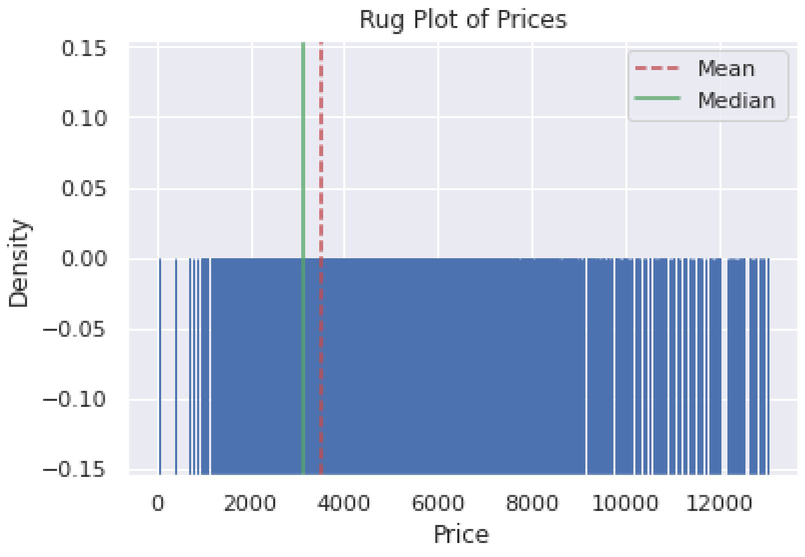


Figure 31. Rug Plot of Prices

<u>3D Plot</u>

The 3D scatter plot provides a visual representation of the relationship between the number of bedrooms and bathrooms against the price. It shows that generally, as the number of bedrooms and bathrooms increases, so does the price. The spread in the 'price' dimension suggests variability in pricing at similar bedroom and bathroom counts, which could be due to other (known/unknown) factors like location and apartment size.

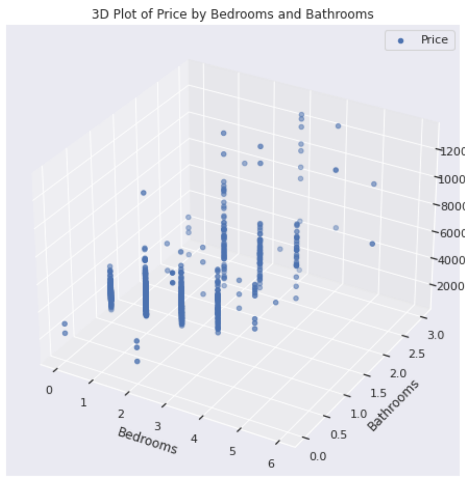


Figure 32. 3D Plot of Price by Bedrooms and Bathrooms

<u>Contour Plot</u>

The contour plot visualizes the price distribution across different numbers of bedrooms and bathrooms. It suggests a general trend where price increases with the number of bathrooms and bedrooms. The most significant price increases are observed in properties with higher numbers of bathrooms. The concentration of darker colors (higher prices) in areas with more bathrooms indicates that bathrooms may have a stronger influence on price in this dataset compared to bedrooms.

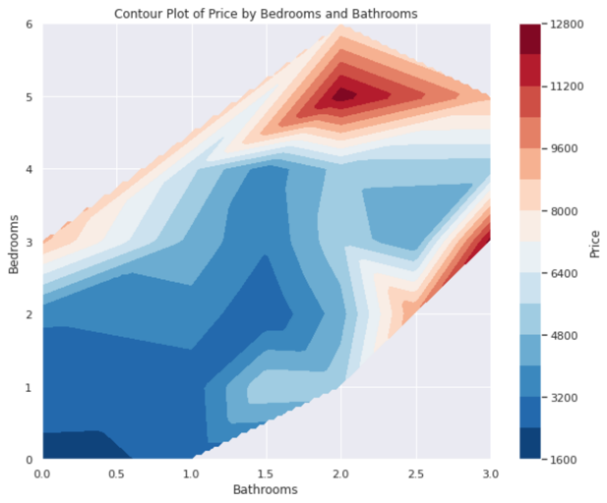


Figure 33. Contour Plot of Price by Bedrooms and Bathrooms

<u>3D Contour Plot</u>

The 3D contour plot presents a visual representation of the relationship between the number of bedrooms and bathrooms and the price of rental properties. Peaks in the contour suggest that listings with a higher number of bedrooms and bathrooms tend to have higher prices. This 3D view allows for an understanding of how these variables interact. The steepness (higher prices) in areas with more bathrooms indicates that bathrooms may have a stronger influence on the price in this dataset compared to bedrooms.

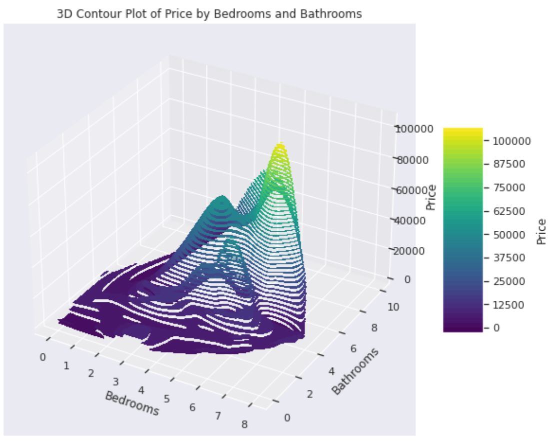


Figure 34. 3D Contour Plot of Price by Bedrooms and Bathrooms

<u>Cluster Map</u>

The cluster map shows hierarchical clustering applied to a subset of the dataset's numerical features: bathrooms, bedrooms, latitude, longitude, and price. Properties that are similar in terms of the selected features are clustered together.

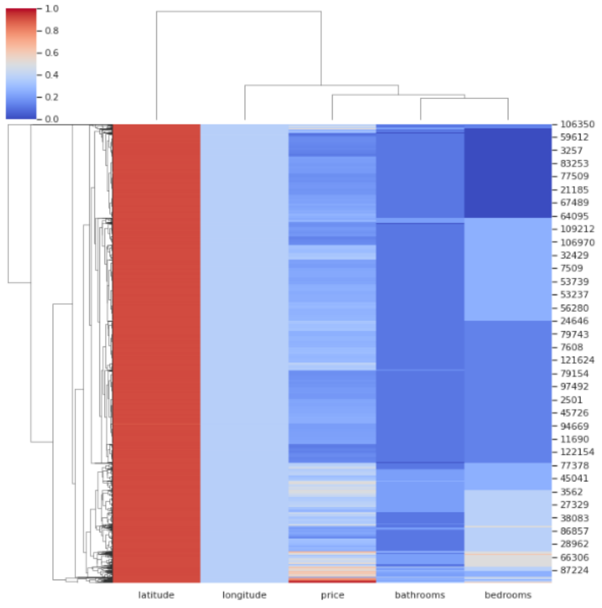


Figure 35. Cluster Map between selected features

<u>Hexbin</u>

The hexbin plot provides a visual density representation of listings based on their geographic location within a specific area of New York city. Darker hexagons represent a higher density of listings, indicating that these areas are more popular or have a higher concentration of available properties. The plot reveals that there are specific hotspots where listings are particularly dense. This plot by itself will not be helpful without an understanding of New York city's map.

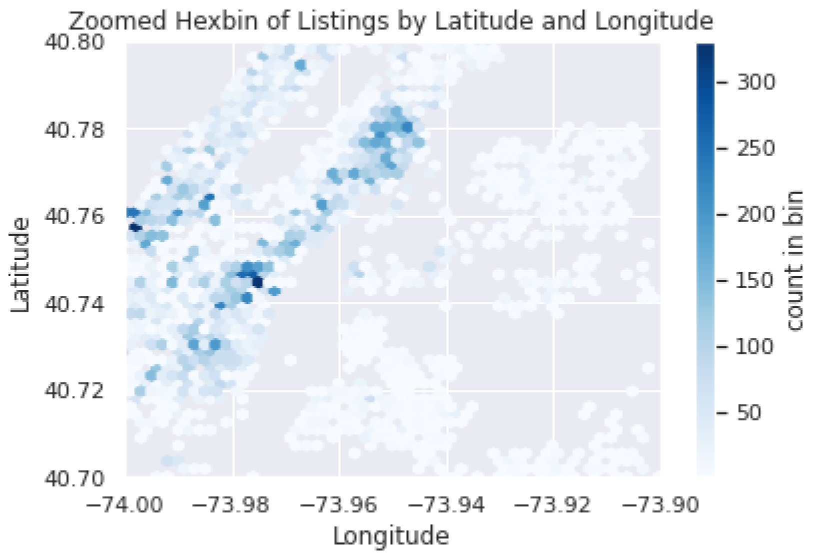


Figure 36. Hexbin between Latitude and Longitude

<u>Strip Plot</u>

The strip plot shows the distribution of rental prices across different interest levels. Listings with a 'low' interest level span a wide range of prices, while 'medium' and 'high' interest listings are more concentrated in lower price ranges. This could suggest that more competitively priced listings generate higher interest, regardless of the actual price point.

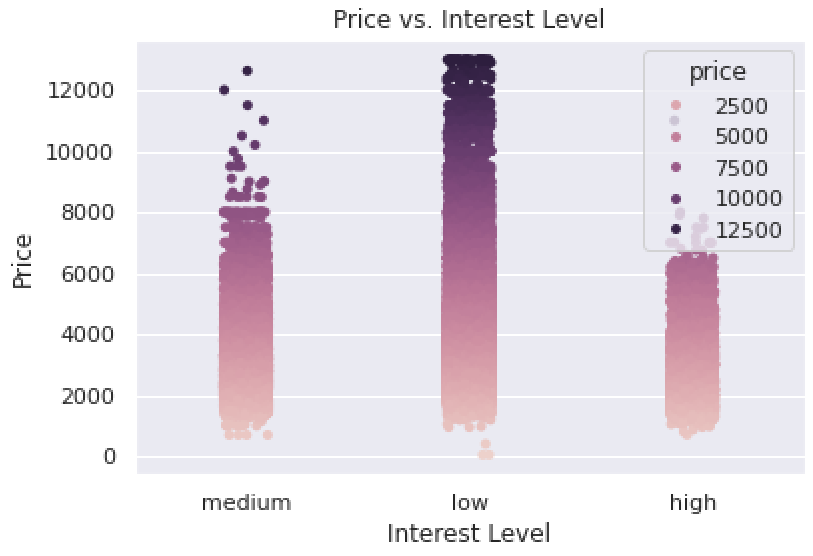


Figure 37. Strip Plot between Price and Interest Level

The strip plot visualizes the relationship between the number of bedrooms in a property and its price, with color intensity indicating the price level. The plot shows a wide distribution of prices across apartments with different bedroom counts. Higher bedroom counts do not necessarily correlate with significantly higher prices. Also, properties with more bedrooms do not always command higher prices, indicating a diverse market with varied preferences and budgets.

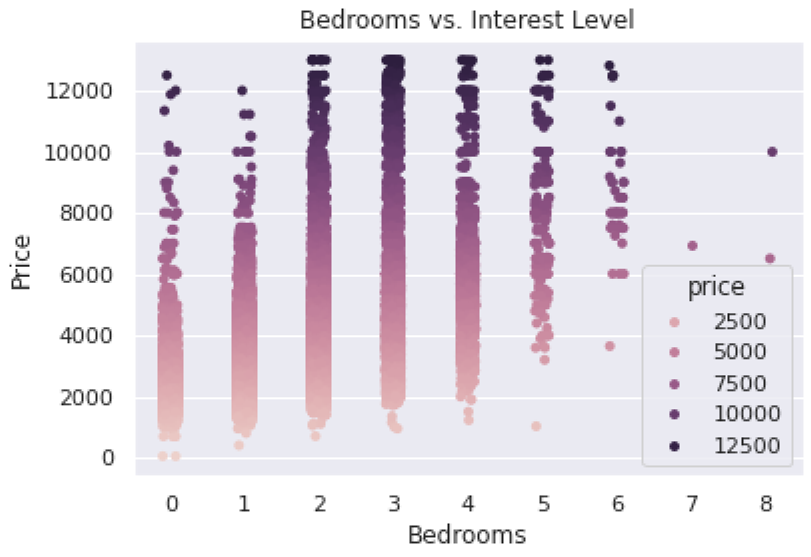


Figure 38. Strip Plot between Price and Bedrooms

The strip plot compares the number of bathrooms to the price of listings, with the color intensity reflecting the price level. It indicates a trend where apartments with more bathrooms tend to have higher prices. The plot suggests that while the number of bathrooms can be an indicator of higher prices, the relationship is not strictly linear, and other factors likely play a role in the final property price.

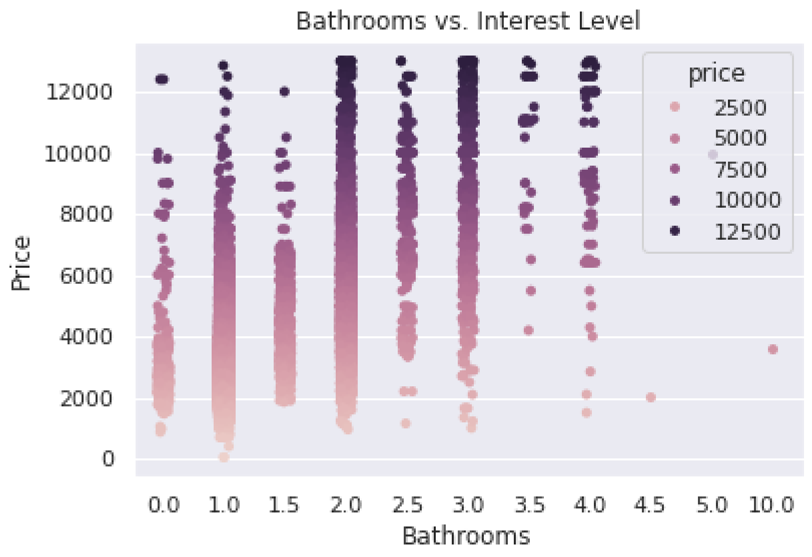


Figure 39. Strip Plot between Price and Bathrooms

Swarm Plot

The swarm plot displays individual rental listings categorized by interest level ('high', 'low', 'medium') and price, filtered to show only those under $2000. It shows that the 'low' interest category has the widest range of prices, while 'high' interest listings are generally lower-priced. This suggests that affordability might drive interest levels. Apartments are densely packed at lower price points across all interest levels, indicating a higher volume of lower-priced listings on the market. Due to the dataset size and computational power, I can only run a subset of the dataset.

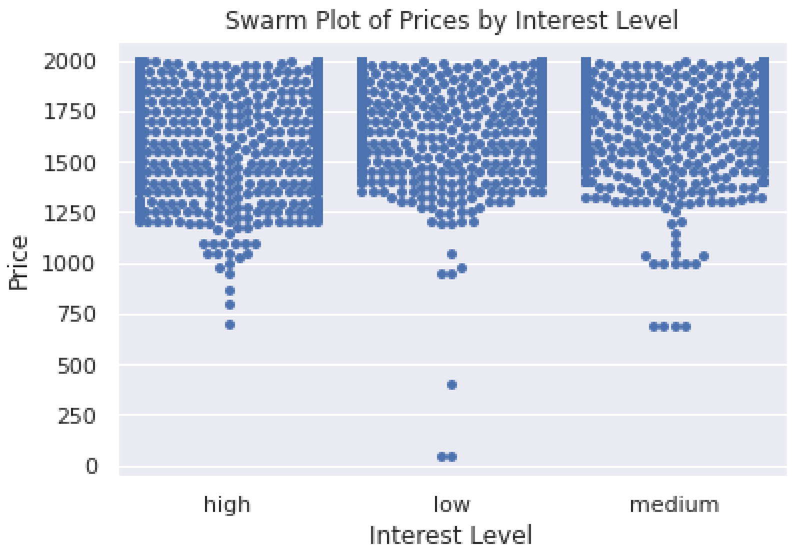


Figure 40. Swarm Plot between Price and Interest Level

Subplot Analysis

<u>Subplot 1</u>

The combined observation of the four plots indicates how different factors relate to property prices in New York city. The first plot shows prices vary significantly across different interest levels, with the 'low' interest level showing a wider range of prices, suggesting a diverse array of property values are considered less desirable. The second plot indicates most listings have 1 to 2 bedrooms, with the 'low' interest level having the highest counts, which could suggest that smaller properties are less desirable or not available often. The third plot shows that 'low' interest level dominates the count across all numbers of bathrooms, which might indicate that properties with any number of bathrooms can be found undesirable based on other factors. Lastly, the violin plot shows the distribution of prices at different interest levels. It demonstrated that lower-priced listings have a wider range of prices and are more commonly found at the 'low' interest level, while the 'high' interest level has a narrower price distribution, indicating that high-interest listings are more consistently priced. The 'medium' interest level shows a price distribution that is somewhat between the 'low' and 'high' interest levels.

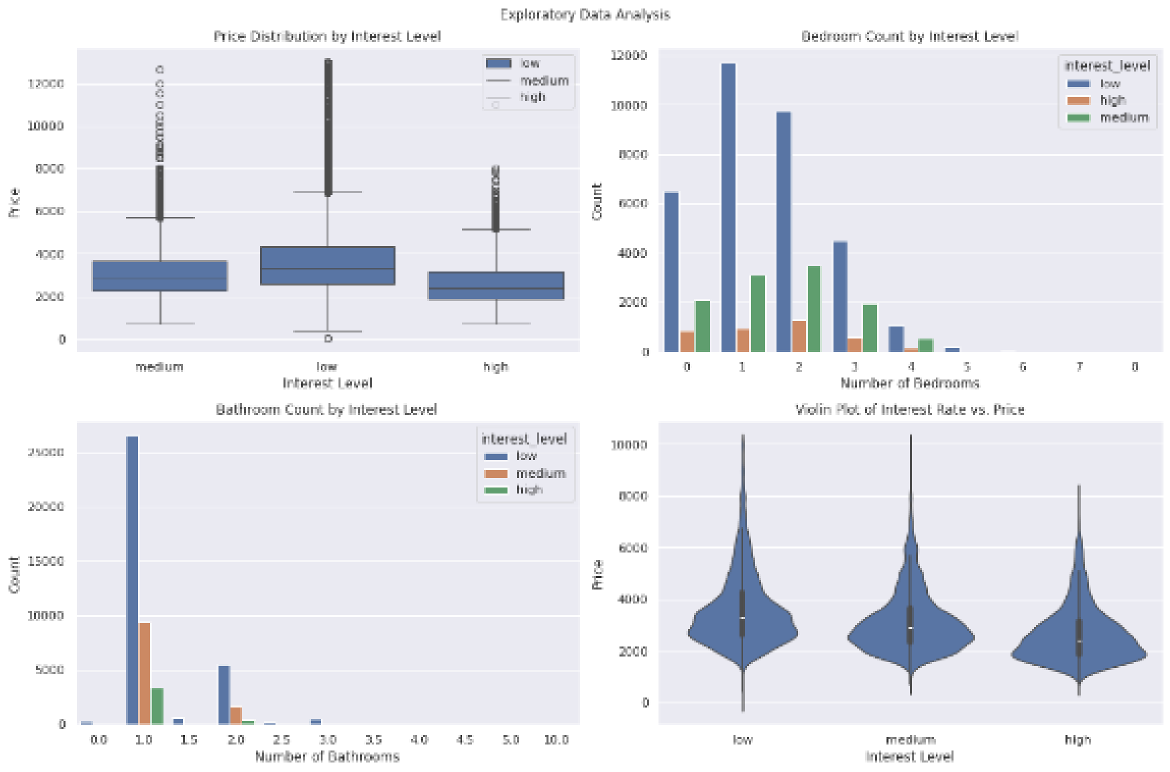


Figure 41. Subplot 1

<u>Subplot 2</u>

The combined observation from the four plots tells a multidimensional story of the dataset. The pie chart in the first subplot indicates that a significant majority of listings have a low-interest level, suggesting that most apartments fail to capture high customer attention. The second and third subplots show the average price distribution across the number of bedrooms and bathrooms, respectively, revealing a trend where more bedrooms or bathrooms typically correspond to higher prices. The fourth subplot, a line plot of price trends over time, suggests price fluctuations with no clear upward or downward trend (considering we only have 3 months of data).

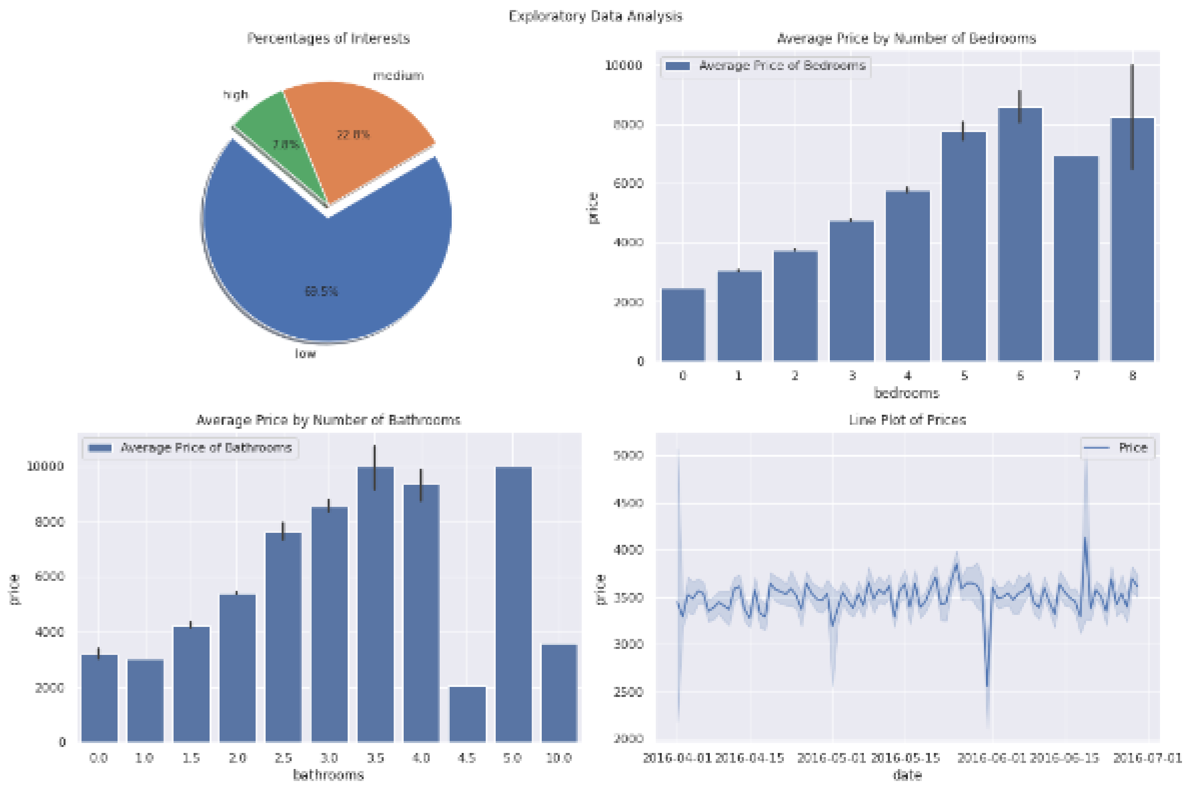


Figure 42. Subplot 2

Subplot 3

The combined observation from the four plots tells a multidimensional story of the dataset. The first plot provides a foundational understanding of the general interest distribution in the dataset, showing most listings have low interest, followed by medium and very few high-interest listings. The second shows the positive correlation between price/bathrooms, price/bedrooms, and bedrooms/bathrooms. The third and fourth plots dive deeper into the features, illustrating how the number of bedrooms and bathrooms impacts the average price. As expected, more bedrooms and bathrooms generally correlate with higher prices, indicating a trend where larger apartments tend to be more expensive.

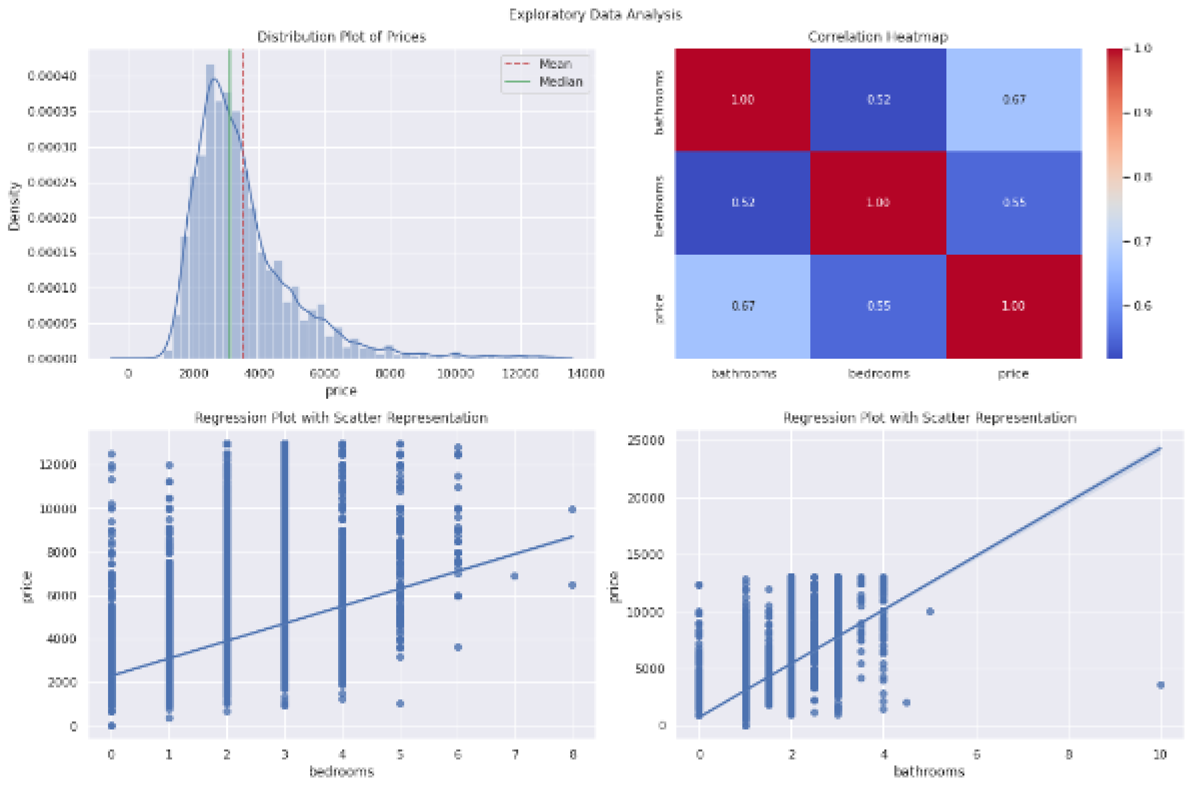


Figure 43. Subplot 3

<u>Subplot 4</u>

The combined observation from the four plots tells a multidimensional story of the dataset. The Hexbin plot indicates a concentration of listings in a specific area, suggesting a potentially higher demand or density in the NYC region. The second plot shows a spread of prices across different levels of interest, with no clear trend indicating that interest level may not be directly proportional to price. The third plot indicates a general trend where price increases with the number of bedrooms, but with a wide variance in price at each bedroom level, suggesting other (known/unknown) factors also significantly influence price. The last plot displays a trend where more bathrooms could correlate with higher prices, but similar to the bedroom analysis, there's considerable variance.

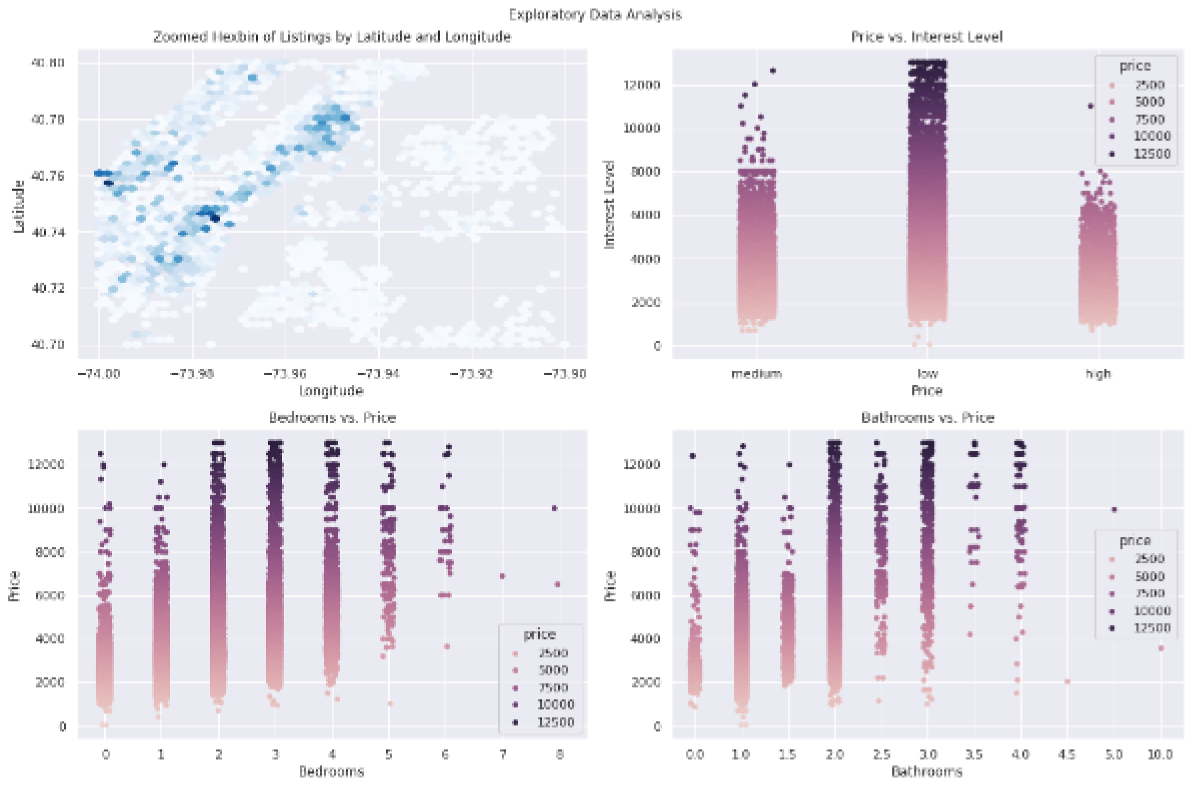
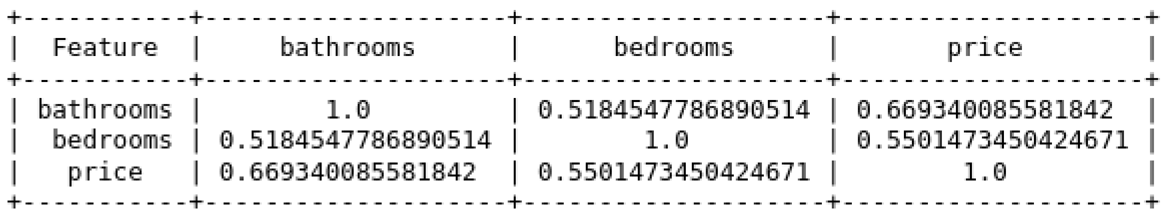


Figure 44. Subplot 4

Tables Analysis
The heatmap (table) displays the correlation matrix for bathrooms, bedrooms, and price. The values, ranging from 0.52 to 0.67, indicate a moderate positive correlation between these features. This means as the number of bedrooms or bathrooms increases, the price tends to increase as well. There is a strongest correlation between price and the number of bathrooms, suggesting that the number of bathrooms is a more significant factor in determining the price than the number of bedrooms.



```
+-----------+--------------------+--------------------+--------------------+
|  Feature  |     bathrooms      |      bedrooms      |       price        |
+-----------+--------------------+--------------------+--------------------+
| bathrooms |        1.0         | 0.5184547786890514 | 0.669340085581842  |
|  bedrooms | 0.5184547786890514 |        1.0         | 0.5501473450424671 |
|   price   | 0.669340085581842  | 0.5501473450424671 |        1.0         |
+-----------+--------------------+--------------------+--------------------+
```

Figure 45. Table of Correlation between Bathrooms, Bedrooms, and Price

DATA TRANSFORMATION / ENGINEERING

This section will explain the process of data transformation and feature engineering, thus providing us with a more robust dataset for predictive analysis.

Feature Extraction

The code can be found in eda/eda.py, notebook/eda.ipynb, and notebooks/feature_extraction.ipynb

The dataset provides a column called "features" with information about the services of the apartment. With a features count of more than 10000, I select the feature as binary with 0 as "not provided" and 1 as "provided".

Here's the list of features I extracted: 'Laundry in Building', 'Dishwasher', 'Hardwood Floors', 'Dogs Allowed', 'Cats Allowed', 'Doorman', 'Elevator', 'No Fee', 'Fitness Center'.

Sentiment Analysis with HuggingFace's BERT

The code can be found in notebooks/sentimental_extraction_run_kaggle_gpu.ipynb and notebooks/sentimental_extraction_run_local.ipynb.

As the dataset also has the description from the owner, I extract sentiment via HuggingFace's pre-trained BERT model which is trained on the Stanford Sentiment Treebank (SST-2) dataset (Getting Started with Sentiment Analysis using Python, Socher, R.). Since their is no available pretrained-BERT for apartment-vocab, this is a good general-purpose sentimental analysis model.

Image Feature Extraction with PyTorch's YOLO.v5

The code can be found in notebooks/feature_extraction_with_image.ipynb.

The dataset also provides ~100 images of selected apartments bound by their unique IDs. Using pre-trained YOLO.v5 from PyTorch for multi-object classification, I counted the numbers of unique objects and stored them in a dictionary, the top 10 items were selected to be the 10 new features of the dataset.

Here's the list of features I extracted: 'chair', 'sink', 'oven', 'refrigerator', 'toilet', 'person', 'potted plant', 'microwave', 'bottle', 'tv'.

DASHBOARD

The code base and running instructions can be found in dash/.

Here's the basic architecture design of the entire app with selected tools that I used.

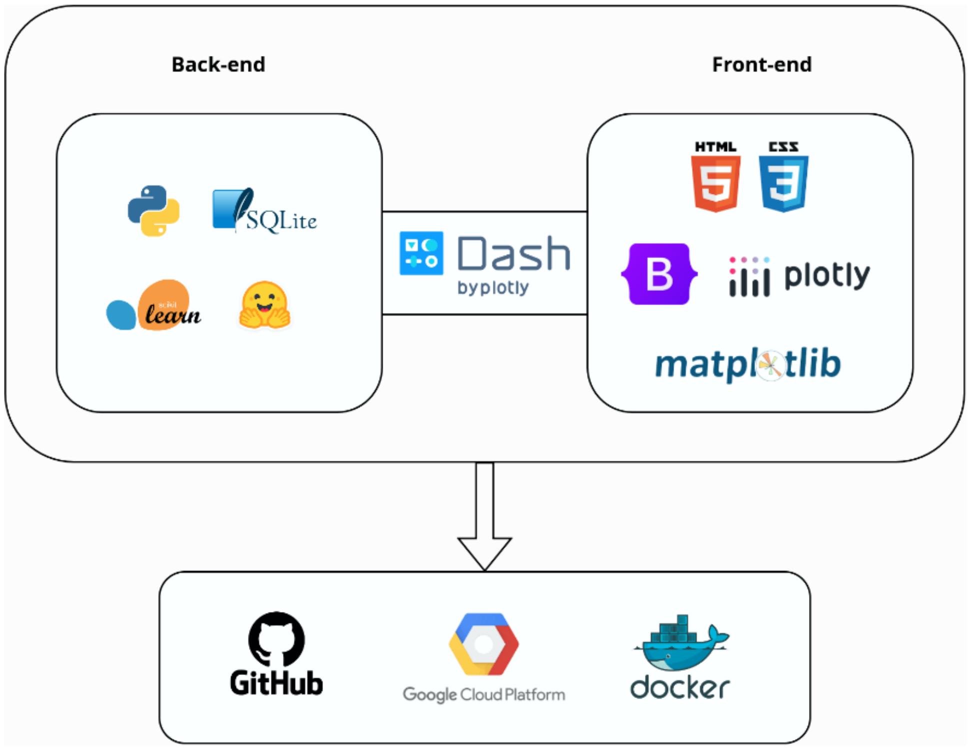
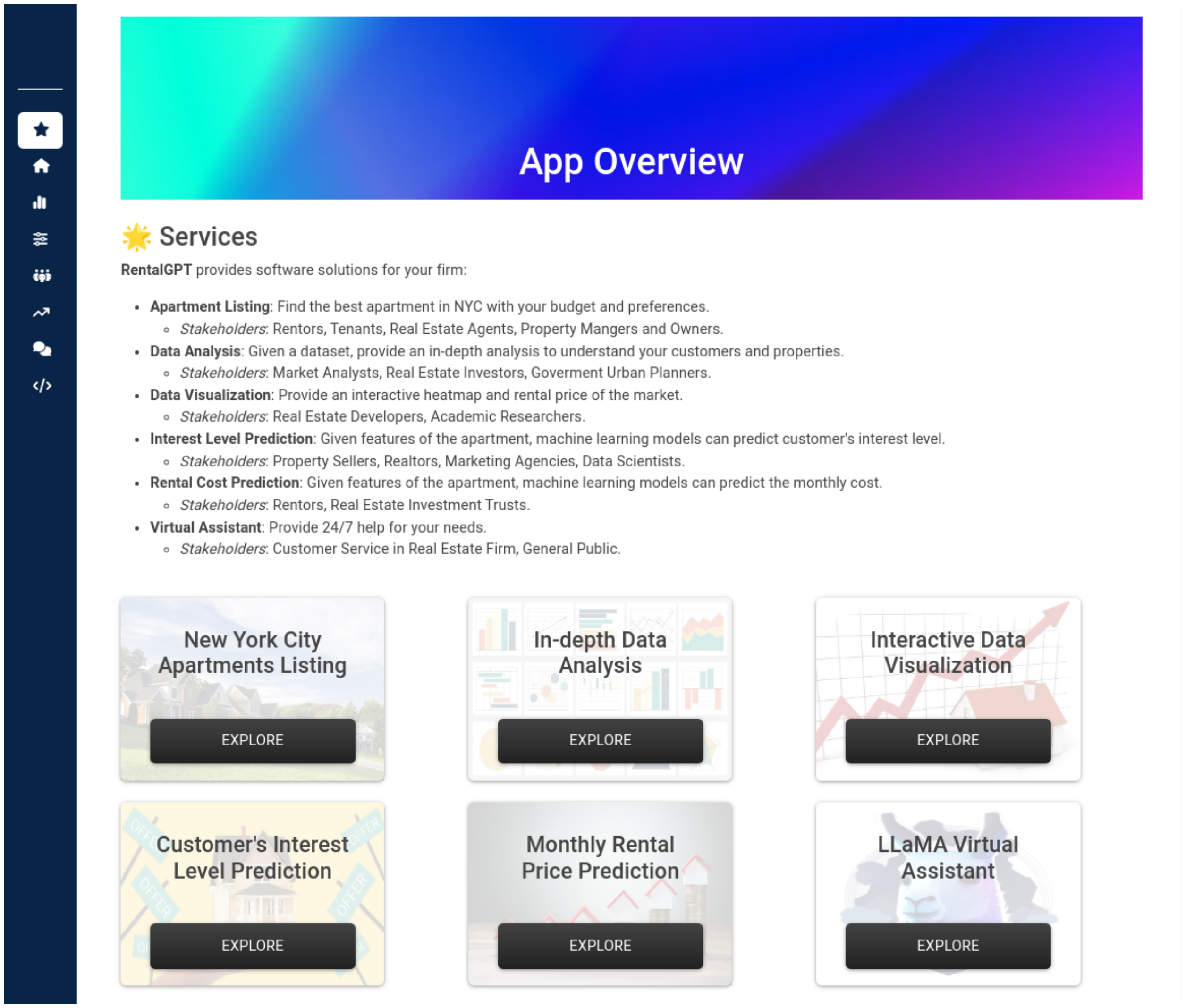


Figure 46. Architecture Design of RentalGPT app.

The app contains eight pages: Overview, Apartments Listing Page, Data Analysis, Data Visualization, Interest Level Prediction, Rental Cost Prediction, Virtual Assistant, and About.

Overview Page

The overview page explained briefly the services offered by RentalGPT and potential stakeholders.



App Overview

Services

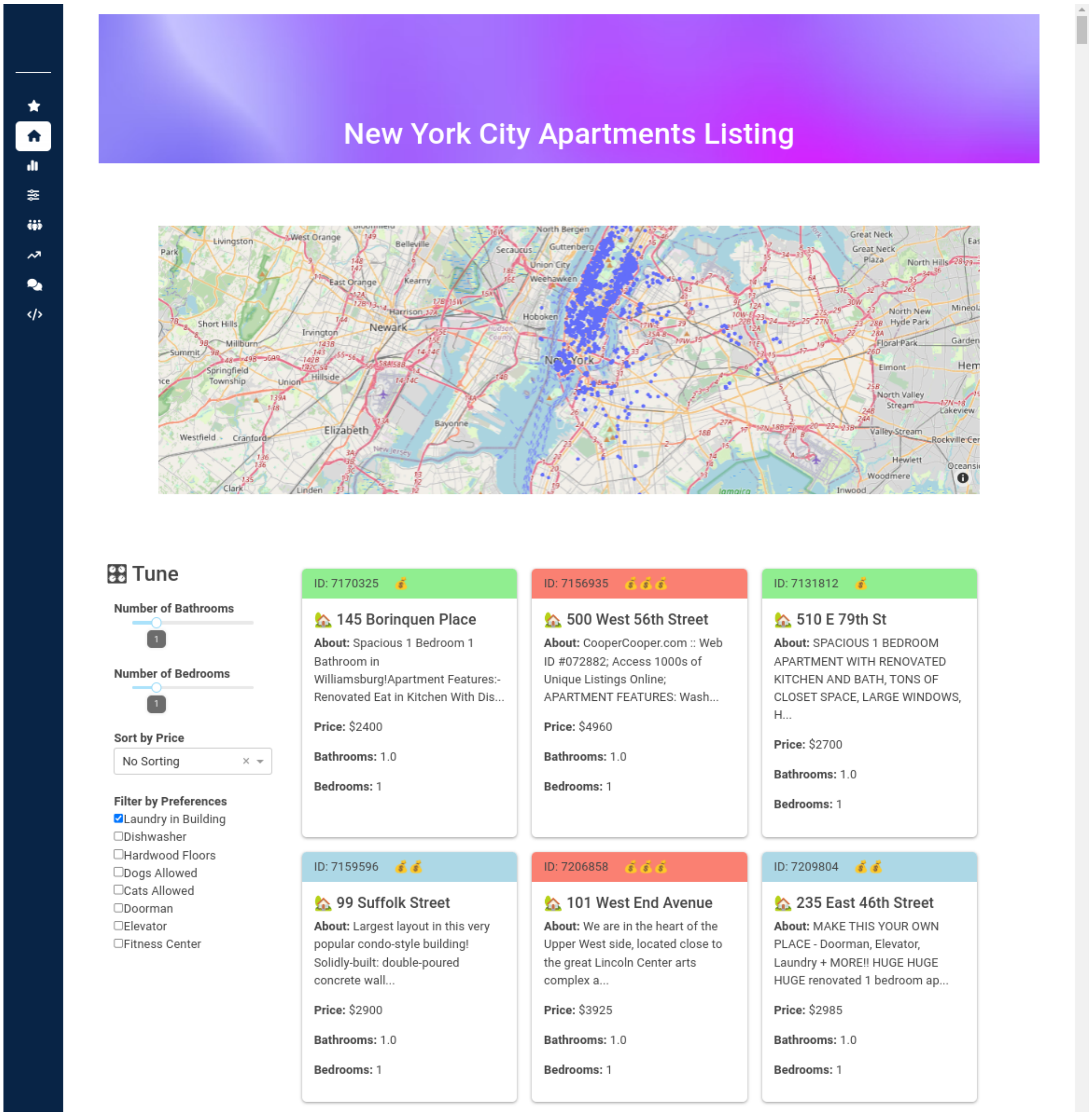
RentalGPT provides software solutions for your firm:

- **Apartment Listing**: Find the best apartment in NYC with your budget and preferences.
  - *Stakeholders*: Rentors, Tenants, Real Estate Agents, Property Mangers and Owners.
- **Data Analysis**: Given a dataset, provide an in-depth analysis to understand your customers and properties.
  - *Stakeholders*: Market Analysts, Real Estate Investors, Goverment Urban Planners.
- **Data Visualization**: Provide an interactive heatmap and rental price of the market.
  - *Stakeholders*: Real Estate Developers, Academic Researchers.
- **Interest Level Prediction**: Given features of the apartment, machine learning models can predict customer's interest level.
  - *Stakeholders*: Property Sellers, Realtors, Marketing Agencies, Data Scientists.
- **Rental Cost Prediction**: Given features of the apartment, machine learning models can predict the monthly cost.
  - *Stakeholders*: Rentors, Real Estate Investment Trusts.
- **Virtual Assistant**: Provide 24/7 help for your needs.
  - *Stakeholders*: Customer Service in Real Estate Firm, General Public.

New York City Apartments Listing
EXPLORE
In-depth Data Analysis
EXPLORE
Interactive Data Visualization
EXPLORE
Customer's Interest Level Prediction
EXPLORE
Monthly Rental Price Prediction
EXPLORE
LLaMA Virtual Assistant
EXPLORE

Figure 47. Overview Page

Apartments Listing Page
The apartments listing page allows the user to filter the apartments based on their preference while seen in the location. The info is queried directly from the SQLite database.


New York City Apartments Listing

Tune

Number of Bathrooms: 1

Number of Bedrooms: 1

Sort by Price: No Sorting

Filter by Preferences
☑Laundry in Building
☐Dishwasher
☐Hardwood Floors
☐Dogs Allowed
☐Cats Allowed
☐Doorman
☐Elevator
☐Fitness Center

ID: 7170325
145 Borinquen Place
About: Spacious 1 Bedroom 1 Bathroom in Williamsburg!Apartment Features:- Renovated Eat in Kitchen With Dis...
Price: $2400
Bathrooms: 1.0
Bedrooms: 1

ID: 7156935
500 West 56th Street
About: CooperCooper.com :: Web ID #072882; Access 1000s of Unique Listings Online; APARTMENT FEATURES: Wash...
Price: $4960
Bathrooms: 1.0
Bedrooms: 1

ID: 7131812
510 E 79th St
About: SPACIOUS 1 BEDROOM APARTMENT WITH RENOVATED KITCHEN AND BATH, TONS OF CLOSET SPACE, LARGE WINDOWS, H...
Price: $2700
Bathrooms: 1.0
Bedrooms: 1

ID: 7159596
99 Suffolk Street
About: Largest layout in this very popular condo-style building! Solidly-built: double-poured concrete wall...
Price: $2900
Bathrooms: 1.0
Bedrooms: 1

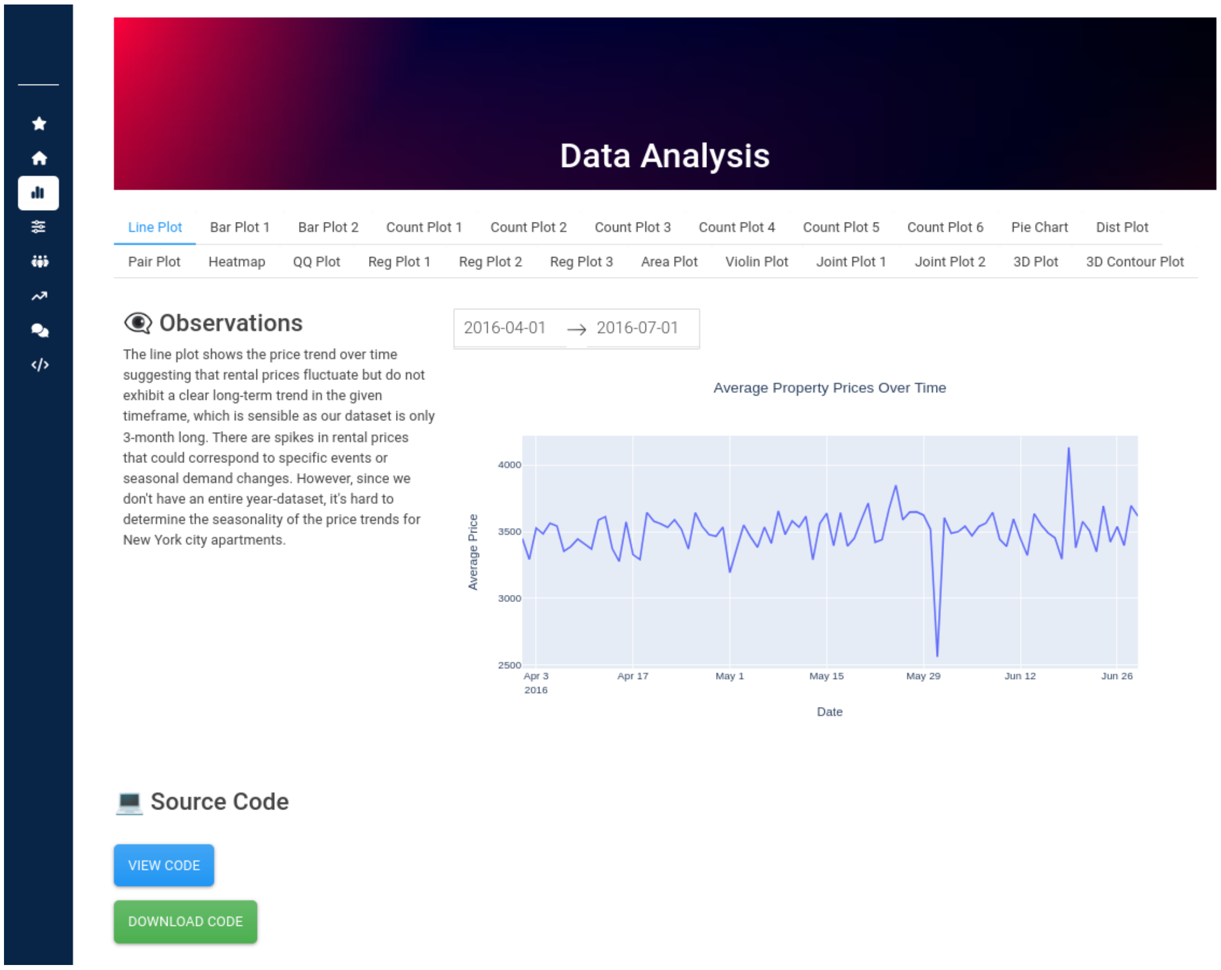
ID: 7206858
101 West End Avenue
About: We are in the heart of the Upper West side, located close to the great Lincoln Center arts complex a...
Price: $3925
Bathrooms: 1.0
Bedrooms: 1

ID: 7209804
235 East 46th Street
About: MAKE THIS YOUR OWN PLACE - Doorman, Elevator, Laundry + MORE!! HUGE HUGE HUGE renovated 1 bedroom ap...
Price: $2985
Bathrooms: 1.0
Bedrooms: 1

Figure 48. Apartments Listing Page

Data Analysis Page

This page is a conversion of some plots from eda/eda.py or notebooks/eda.ipynb into Plotly static and interactive plots. The observation and codebase for each (static) plot can be viewed and downloaded via the interactive button.



The line plot shows the price trend over time suggesting that rental prices fluctuate but do not exhibit a clear long-term trend in the given timeframe, which is sensible as our dataset is only 3-month long. There are spikes in rental prices that could correspond to specific events or seasonal demand changes. However, since we don't have an entire year-dataset, it's hard to determine the seasonality of the price trends for New York city apartments.

Figure 49. In-depth Data Analysis Page

Data Visualization Page

This page is a conversion of some plots from eda/eda.py or notebooks/eda.ipynb into Plotly interactive plots. The observation and codebase for each (static) plot can be viewed and downloaded via the interactive button.

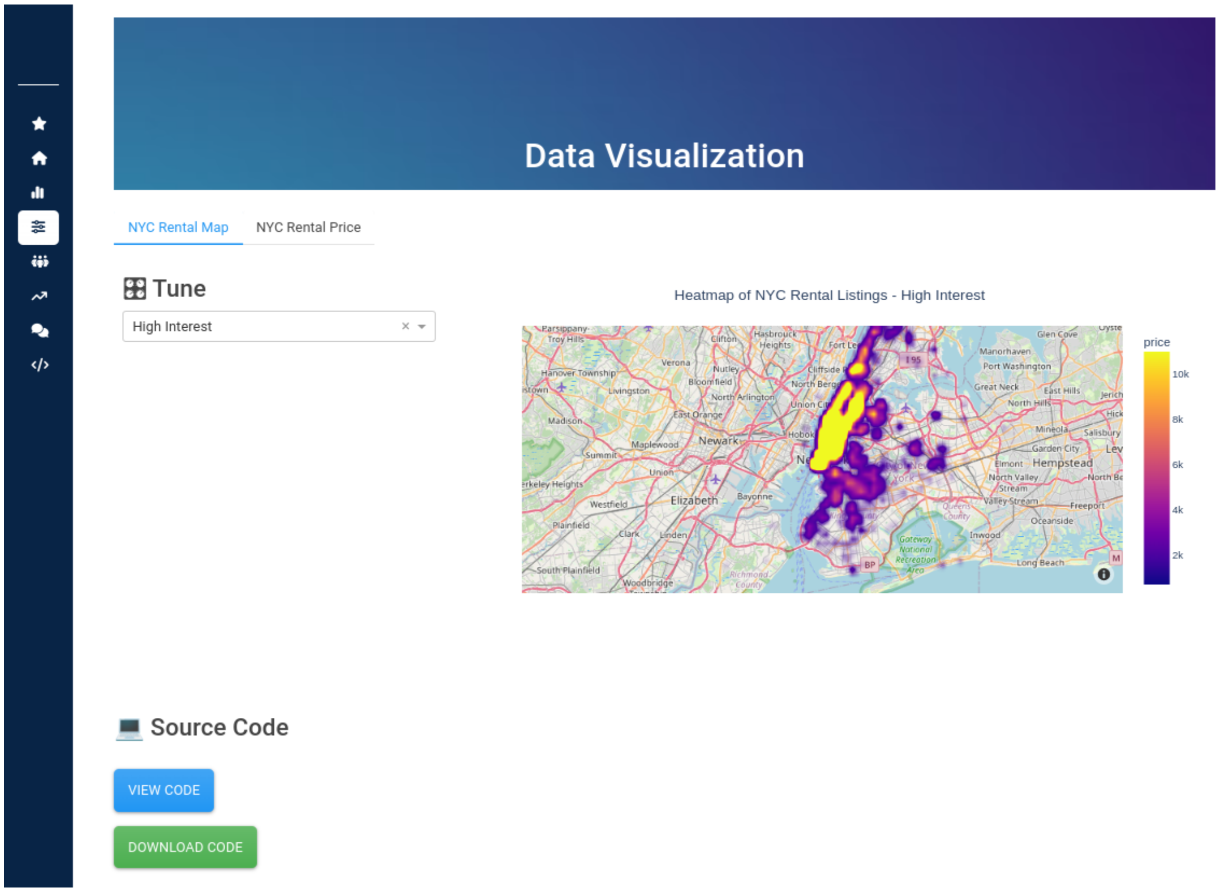
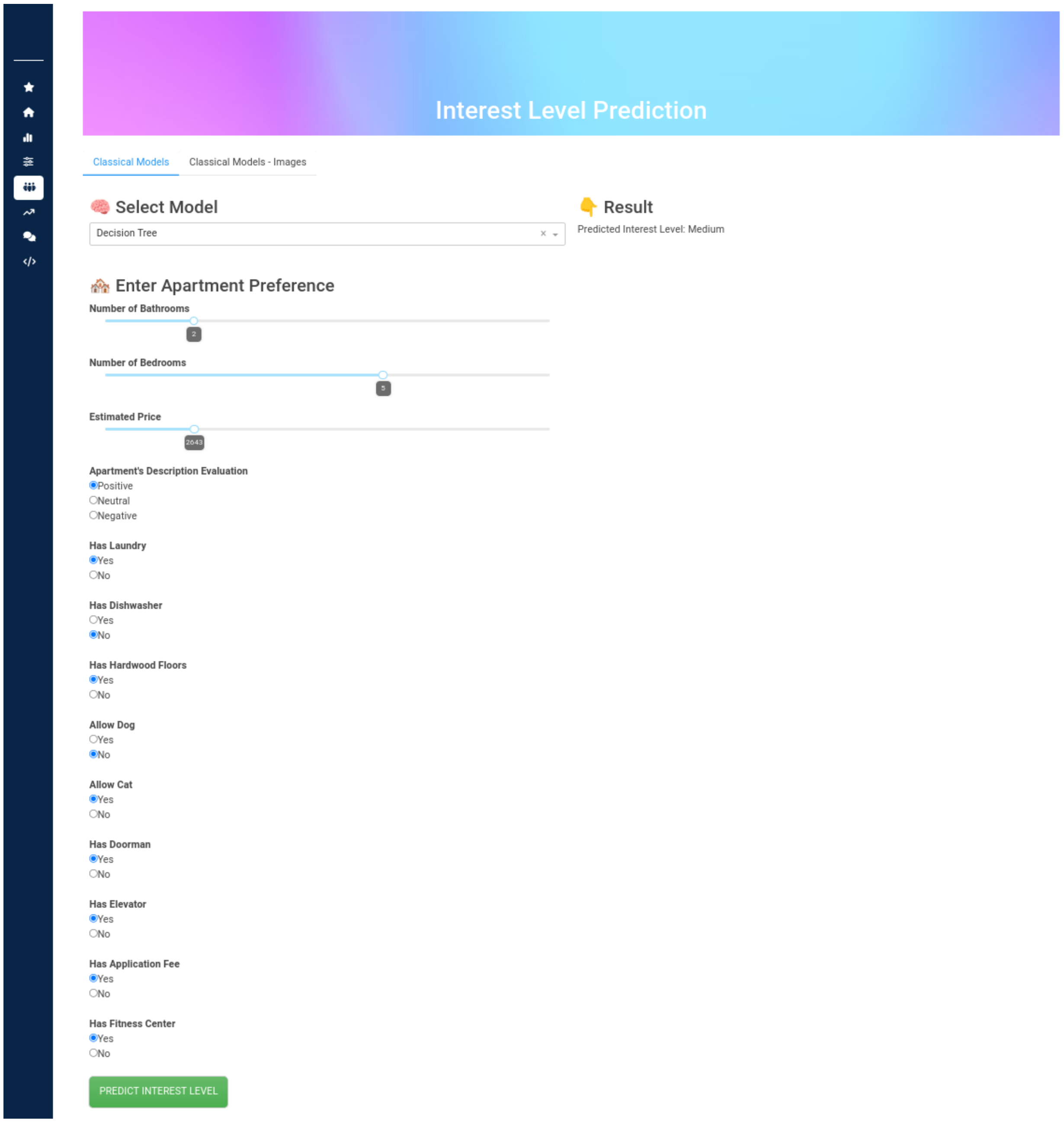


Figure 50. Interactive Data Visualization Page

Interest Level Prediction Page

Based on the selected (classical) ML models (out of 5) such as Decision Tree or SVM and the apartment's features the customer's interest level will be classified as Low, Medium, or High.



Interest Level Prediction

Classical Models | Classical Models - Images

🧠 Select Model

Decision Tree

👇 Result

Predicted Interest Level: Medium

🏘️ Enter Apartment Preference

Number of Bathrooms: 2

Number of Bedrooms: 5

Estimated Price: 2643

Apartment's Description Evaluation
[✓ Positive] Neutral Negative

Has Laundry
[✓ Yes] No

Has Dishwasher
Yes [✓ No]

Has Hardwood Floors
[✓ Yes] No

Allow Dog
Yes [✓ No]

Allow Cat
[✓ Yes] No

Has Doorman
[✓ Yes] No

Has Elevator
[✓ Yes] No

Has Application Fee
[✓ Yes] No

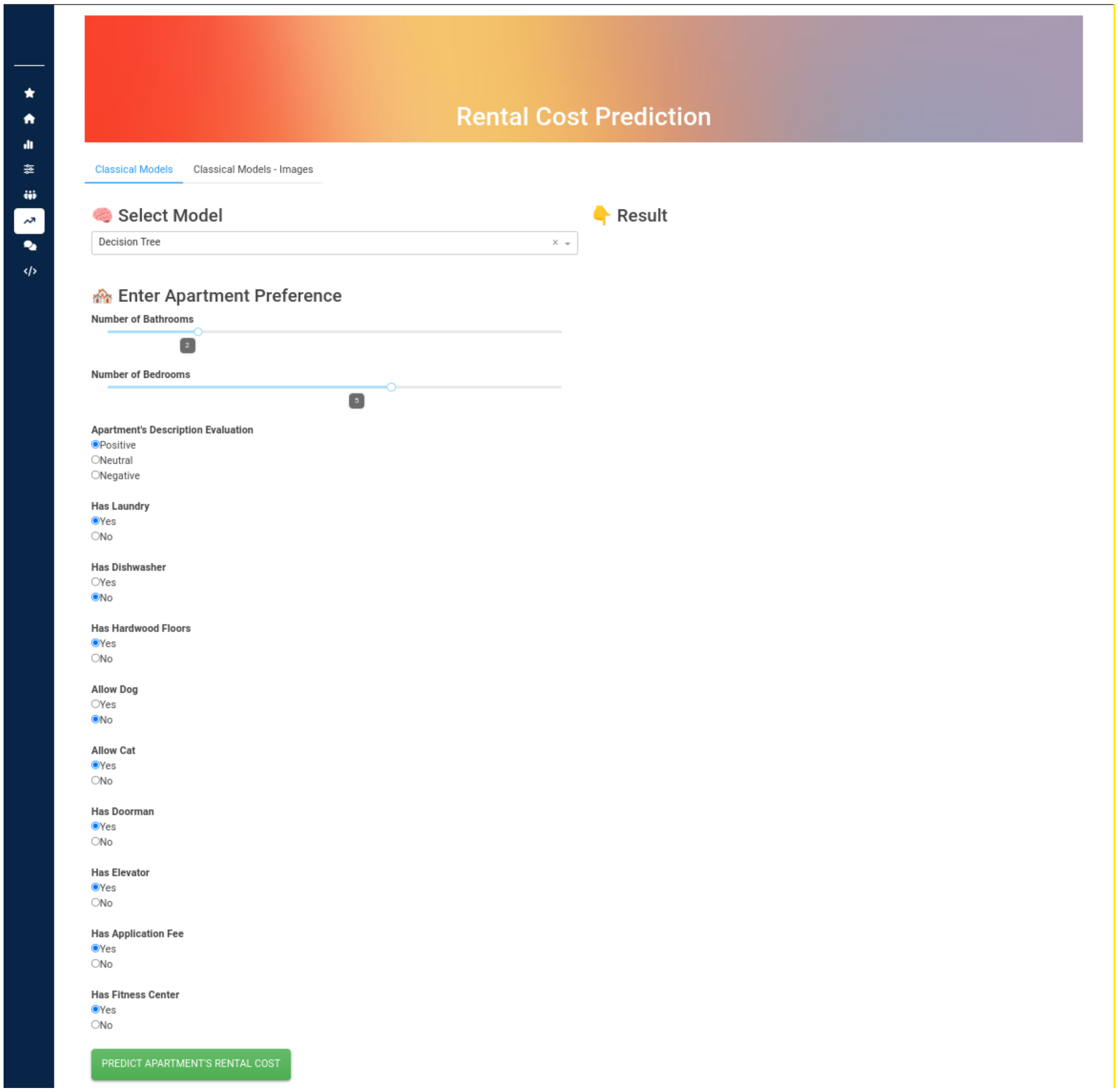
Has Fitness Center
[✓ Yes] No

PREDICT INTEREST LEVEL

Figure 51. Customer's Interest Level Prediction

Rental Cost Prediction Page

Based on the selected (classical) ML models (out of 5) such as Decision Tree or SVM and the apartment's features the apartment's monthly rental cost will be predicted.


Rental Cost Prediction
Classical Models
Classical Models - Images
Select Model
Result
Enter Apartment Preference
PREDICT APARTMENT'S RENTAL COST

Select Model: Decision Tree

Number of Bathrooms: 2

Number of Bedrooms: 5

Apartment's Description Evaluation: [✓ Positive] Neutral Negative

Has Laundry: [✓ Yes] No

Has Dishwasher: Yes [✓ No]

Has Hardwood Floors: [✓ Yes] No

Allow Dog: Yes [✓ No]

Allow Cat: [✓ Yes] No

Has Doorman: [✓ Yes] No

Has Elevator: [✓ Yes] No

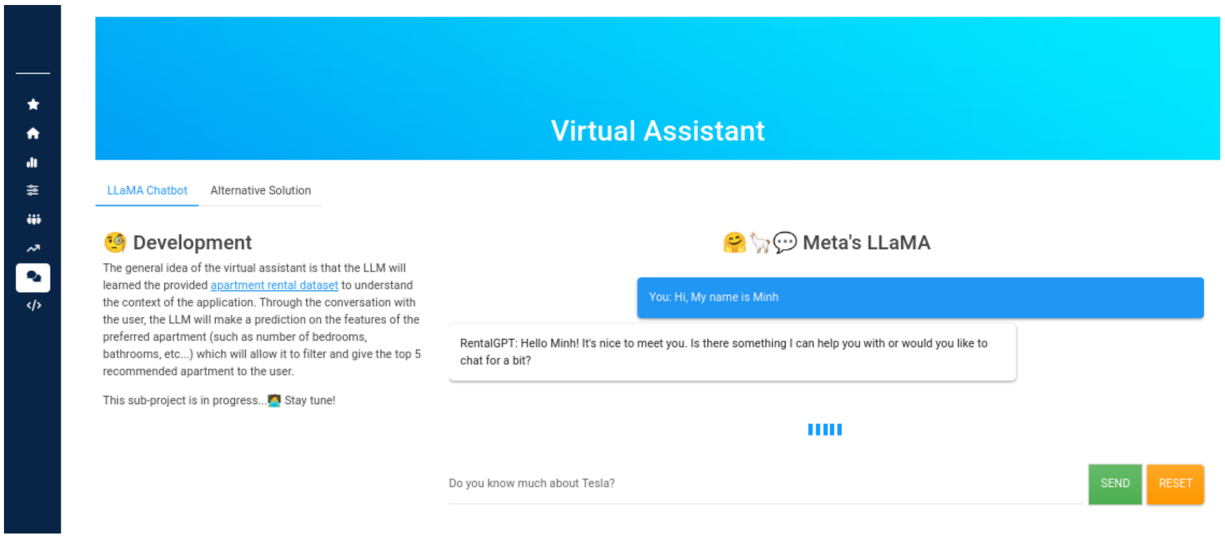
Has Application Fee: [✓ Yes] No

Has Fitness Center: [✓ Yes] No

Figure 52. Rental Cost Prediction

Virtual Assistant Page

The general idea of the virtual assistant is that the LLM will learn the provided apartment rental dataset to understand the context of the application. Through the conversation with the user, the LLM will predict the features of the preferred apartment (such as number of bedrooms, bathrooms, etc...) which will allow it to filter and give the top 5 recommended apartments to the user. Here I used HuggingFace Chat (HugChat) for Open Assistant's API call (a model built using Meta's LLaMA). However, this feature is still in progress.



The general idea of the virtual assistant is that the LLM will learned the provided apartment rental dataset to understand the context of the application. Through the conversation with the user, the LLM will make a prediction on the features of the preferred apartment (such as number of bedrooms, bathrooms, etc...) which will allow it to filter and give the top 5 recommended apartment to the user.

This sub-project is in progress...🧑‍💻 Stay tune!

You: Hi, My name is Minh

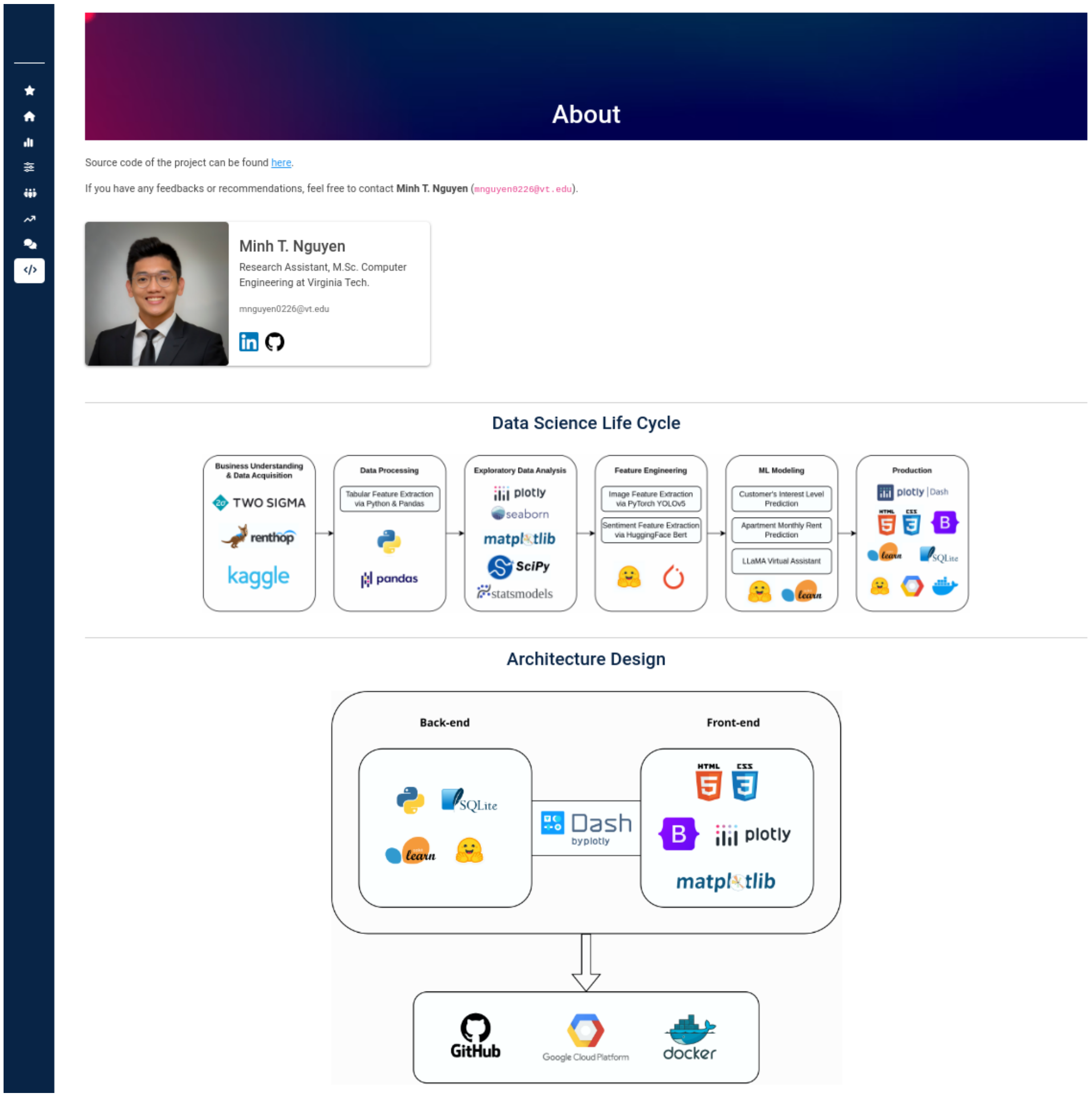
RentalGPT: Hello Minh! It's nice to meet you. Is there something I can help you with or would you like to chat for a bit?

Do you know much about Tesla?

Figure 53. LLaMA Virtual Assistant

About Page

The About page provides details about the developer and diagrams for developing this project.



About

Source code of the project can be found here.

If you have any feedbacks or recommendations, feel free to contact **Minh T. Nguyen** (`mnguyen0226@vt.edu`).

Minh T. Nguyen

Research Assistant, M.Sc. Computer Engineering at Virginia Tech.

mnguyen0226@vt.edu

Data Science Life Cycle

Architecture Design

Figure 54. About Page

CONCLUSION

From this project, I have the opportunity to practice the full data science life cycle and look at the provided dataset through multidimensional angles via various plots. The feature extraction steps using pre-trained models are extremely valuable for building more robust ML models. Each static plot can be deployed into the Dash app.

By creating a Dash app, the information is presented interactively without the concerns of information overload. Various stakeholders can benefit from this application.

The app is indeed user-friendly. Although I did not post my app on LinkedIn, I did have the opportunity to present it during technical interviews with professionals. The UI is well defined with separated sections.

As mentioned, the app is functional with several services such as:

- Apartment Listing: Find the best apartment in NYC with your budget and preferences.
    + Stakeholders: Renters, Tenants, Real Estate Agents, Property Managers, and Owners.
- Data Analysis: Given a dataset, provide an in-depth analysis to understand your customers and properties.
    + Stakeholders: Market Analysts, Real Estate Investors, Government Urban Planners.
- Data Visualization: Provide an interactive heatmap and rental price of the market.
    + Stakeholders: Real Estate Developers, Academic Researchers.
- Interest Level Prediction: Given features of the apartment, machine learning models can predict the customer's interest level.
    + Stakeholders: Property Sellers, Realtors, Marketing Agencies, Data Scientists.
- Rental Cost Prediction: Given the features of the apartment, machine learning models can predict the monthly cost.
    + Stakeholders: Renters, Real Estate Investment Trusts.
- Virtual Assistant: Provide 24/7 help for your needs.
    + Stakeholders: Customer Service in Real Estate Firm, General Public.

REFERENCES

Two Sigma Connect: Rental Listing Inquiries | Kaggle. (2023). Kaggle.com. https://www.kaggle.com/competitions/two-sigma-connect-rental-listing-inquiries/overview

Getting Started with Sentiment Analysis using Python. (2022). Huggingface.co. https://huggingface.co/blog/sentiment-analysis-python

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013). Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. ACL Anthology, 1631–1642. https://aclanthology.org/D13-1170/

APPENDICES

Deployment on Google Cloud Platform

1. Make sure you have a GCP account with some credit amount.
2. Go to Google Cloud Console > Create a New Project > Activate Cloud Shell
3. Create a new project
4. On the terminal, type `python3 -m venv .venv` to create a new Python environment.
5. On the terminal, type `. .venv/bin/activate` to activate your newly created environment.
6. Open Editor, add code for app.py, requirements.txt, Dockerfile.
7. On the terminal, install package via `pip install -r requirements.txt`
8. On the terminal, enable services through GCP terminal: `gcloud services enable containerregistry.googleapis.com`
9. On the terminal, enable permission via `gcloud auth configure-docker`
10. On the terminal, build your docker image via `docker build -f Dockerfile -t gcr.io/your-project/test:test .`
11. On the terminal, push your docker image via `docker push gcr.io/your-project/test:test`
12. On the terminal, deploy your application via `gcloud run deploy dashapp --image gcr.io/your-project/test:test`
13. Your application has been successfully deployed!

Deployment on Virginia Tech's Kubernetes Rancher Server

1. Consider reading through the Documentation - Cloud Quickstart.
2. Make sure you have DockerHub account and access to Virginia Tech's Kubernetes Rancher (cloud.cs.vt.edu).
3. Make sure you have Docker Desktop installed locally. Open Docker Desktop and sign in with your DockerHub to enable Docker daemon to run on the background.
4. On the terminal, build your docker image via `docker build -t your-dockerhub-username/project-name .`
5. On the terminal, push your docker image via `docker push your-dockerhub-username/project-name`
6. After granted access to cloud.cs.vt.edu via asking admin, create your Workload with the port that you set on Dockerfile. The name of the image for the server to pull should also be filled in here (such as your-dockerhub-username/project-name). Here, you can also define your domain name.
7. Now, create the ingress, and select your workload. Activate 1 pod to initialize and run your application.
8. Your application has been successfully deployed!

Selected Codebase

Since the codebase for the project is relatively large, I will post only selected sections. The instructions on the reproducibility of the entire app can be found here. It is recommended that grader should check the code on github or submission files. Detailed instructions are provided.

EDA In-depth

```python
# import libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import statsmodels.api as sm
from mpl_toolkits.mplot3d import Axes3D
from scipy.interpolate import griddata
from prettytable import PrettyTable

sns.set(style="whitegrid", color_codes=True)
sns.set(font_scale=1)

import warnings

warnings.filterwarnings("ignore")

# import the dataset
df = pd.read_json("../data/train.json")
df.head(5)

# price scatter plot
df_price = df["price"].values
plt.scatter(range(df.shape[0]), df_price)
plt.title("Scatter Plot of Renting Prices")
plt.xlabel("Property Index")
plt.ylabel("Price")
plt.show()

# similarly, we can do a boxen plot for outlier detection
sns.boxenplot(y="price", data=df)
```

```
plt.plot([], label="Price Distribution", color="blue")
plt.title("Boxen Plot of Prices")
plt.ylabel("Price")
plt.legend()
plt.show()

# outlier removal
upper_bound = np.percentile(df["price"].values, 99)
df_filtered = df[df["price"] <= upper_bound]

plt.scatter(range(df_filtered.shape[0]), df_filtered["price"],
label="Prices")
plt.title("Scatter Plot of Renting Prices Without Outliers")
plt.xlabel("Property Index")
plt.ylabel("Price")
plt.legend(loc="best")
plt.show()

# similarly, we can do a boxen plot for outlier detection
sns.boxenplot(y="price", data=df_filtered)
plt.plot([], label="Price Distribution", color="blue")
plt.title("Boxen Plot of Prices Without Outliers")
plt.ylabel("Price")
plt.legend()
plt.show()

# create new date columns
df_filtered["date"] = pd.to_datetime(
   df_filtered["created"]
).dt.date  # Returns numpy array of python datetime.date objects.
df_filtered["year"] = pd.to_datetime(
   df_filtered["created"]
).dt.year  # The year of the datetime.
df_filtered["month"] = pd.to_datetime(
   df_filtered["created"]
).dt.month  # The month as January=1, December=12.
df_filtered["day"] = pd.to_datetime(
   df_filtered["created"]
).dt.day  # The day of the datetime.
```

```
df_filtered["hour"] = pd.to_datetime(
   df_filtered["created"]
).dt.hour  # The hours of the datetime.
df_filtered["weekday"] = pd.to_datetime(
   df_filtered["created"]
).dt.weekday  # The day of the week with Monday=0, Sunday=6.
df_filtered["quarter"] = pd.to_datetime(
   df_filtered["created"]
).dt.quarter  # The quarter of the date.

plt.figure(figsize=(10, 6))
sns.lineplot(data=df_filtered, x="date", y="price", label="Price")
plt.title("Line Plot of Prices")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.show()

bedroom_counts = df_filtered["bedrooms"].value_counts()
plt.bar(bedroom_counts.index, bedroom_counts.values, label="Bedroom
Counts")
plt.title("Bar Chart of Number of Bedrooms Counts")
plt.xlabel("Number of Bedrooms")
plt.ylabel("Number of Occurence")
plt.legend()
plt.show()

bathroom_counts = df_filtered["bathrooms"].value_counts()
plt.bar(bedroom_counts.index, bedroom_counts.values, label="Bathroom
Counts")
plt.title("Bar Chart of Number of Bathrooms Counts")
plt.xlabel("Number of Bathrooms")
plt.ylabel("Number of Occurence")
plt.legend()
plt.show()

fig = plt.figure(figsize=(12, 6))
sns.countplot(
   x="hour",
    hue="interest_level",
    hue_order=["low", "medium", "high"],
    data=df_filtered,
)
plt.title("Bar Chart of Interest Level in 24 Hours")
plt.xlabel("Hour")
plt.ylabel("Interest Level Counts")
plt.show()

fig = plt.figure(figsize=(12, 6))
sns.countplot(
    x="year",
    hue="interest_level",
    hue_order=["low", "medium", "high"],
    data=df_filtered,
)
plt.title("Bar Chart of Interest Level in Year")
plt.xlabel("Year")
plt.ylabel("Interest Level Counts")
plt.show()

fig = plt.figure(figsize=(12, 6))
sns.countplot(
    x="month",
    hue="interest_level",
    hue_order=["low", "medium", "high"],
    data=df_filtered,
)
plt.title("Bar Chart of Interest Level in Month")
plt.xlabel("Month")
plt.ylabel("Interest Level Counts")
plt.show()

fig = plt.figure(figsize=(12, 6))
sns.countplot(
    x="day", hue="interest_level", hue_order=["low", "medium", "high"],
data=df_filtered
)
plt.title("Bar Chart of Interest Level in Day")
plt.xlabel("Day")
plt.ylabel("Interest Level Counts")
plt.show()

fig = plt.figure(figsize=(12, 6))
sns.countplot(
    x="weekday",
    hue="interest_level",
    hue_order=["low", "medium", "high"],
    data=df_filtered,
)
plt.title("Bar Chart of Interest Level in Weekday")
plt.xlabel("Weekday")
plt.ylabel("Interest Level Counts")
plt.show()

fig = plt.figure(figsize=(12, 6))
sns.countplot(
    x="quarter",
    hue="interest_level",
    hue_order=["low", "medium", "high"],
    data=df_filtered,
)
plt.title("Bar Chart of Interest Level in Quarter")
plt.xlabel("Quarter")
plt.ylabel("Interest Level Counts")
plt.show()

# plot pie chart
df_interest_split = df.interest_level.value_counts().values
df_interest_label = ["low", "medium", "high"]
explode = (0.1, 0, 0)
plt.pie(
    df_interest_split,
    explode=explode,
    labels=df_interest_label,
    autopct="%1.1f%%",
    shadow=True,
    startangle=140,
)
plt.title("Percentages of Interests")
plt.legend(df_interest_label, loc="best")
plt.show()

sns.distplot(df_filtered["price"])

# add means and variance to plot
plt.axvline(x=df_filtered["price"].mean(), color="r", linestyle="--",
label="Mean")
plt.axvline(x=df_filtered["price"].median(), color="g", linestyle="-",
label="Median")

plt.title("Distribution Plot of Prices")
plt.xlabel("Price")
plt.ylabel("Distribution")
plt.legend()
plt.show()

sns.pairplot(df_filtered[["bathrooms", "bedrooms", "price"]])
plt.title("Feature Pair Plot")
plt.show()

# get the correlation
corr = df_filtered[["bathrooms", "bedrooms", "price"]].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", cbar=True)
cbar = plt.gcf().axes[-1]
plt.title("Correlation Heatmap")
plt.show()

# similarly, histogram plot with KDE
sns.histplot(data=df_filtered, x="price", kde=True)

# add means and variance to plot
plt.axvline(x=df_filtered["price"].mean(), color="r", linestyle="--",
label="Mean")
plt.axvline(x=df_filtered["price"].median(), color="g", linestyle="-",
label="Median")

plt.title("KDE of Prices")
plt.xlabel("Price")
plt.ylabel("Density")
plt.legend()
plt.show()

prices = np.log1p(df_filtered["price"])
sm.qqplot(prices, line="45", fit=True, label="log-price")
plt.title("QQ Plot of Log-transformed Prices")
plt.legend()
plt.show()

sns.kdeplot(df_filtered["price"], fill=True, alpha=0.6, linewidth=2)

# add means and variance to plot
plt.axvline(x=df_filtered["price"].mean(), color="r", linestyle="--",
label="Mean")
plt.axvline(x=df_filtered["price"].median(), color="g", linestyle="-",
label="Median")

plt.title("KDE of Prices")
plt.xlabel("Price")
plt.ylabel("Density")
plt.legend()
plt.show()

sns.regplot(x="bathrooms", y="bedrooms", data=df_filtered, scatter=True,
fit_reg=True)
plt.title("Regression Plot with Scatter Representation")
plt.xlabel("Bathrooms")
plt.ylabel("Bedrooms")
plt.show()

sns.regplot(x="bathrooms", y="price", data=df_filtered, scatter=True,
fit_reg=True)
plt.title("Regression Plot with Scatter Representation")
plt.xlabel("Bathrooms")
plt.ylabel("Price")
plt.show()

sns.regplot(x="bedrooms", y="price", data=df_filtered, scatter=True,
fit_reg=True)
plt.title("Regression Plot with Scatter Representation")
plt.xlabel("Bedrooms")
plt.ylabel("Price")
plt.show()

df_filtered["date"] = pd.to_datetime(df_filtered["created"]).dt.date
grouped = df_filtered.groupby(["date",
"interest_level"])["price"].mean().reset_index()
pivot_df = grouped.pivot(index="date", columns="interest_level",
values="price")
pivot_df.plot(kind="area", stacked=False, alpha=0.5, figsize=(12, 6))
plt.title("Area Plot of Prices Over Time by Interest Level")
plt.xlabel("Date")
plt.ylabel("Average Price")
plt.show()

sns.violinplot(
   x="interest_level",
   y="price",
   order=["low", "medium", "high"],
   data=df_filtered[df_filtered.price <=
df_filtered.price.quantile(0.99)],
)
plt.plot([], label="Price Distribution", color="blue")
plt.title("Violin Plot of Interest Rate vs. Price")
plt.xlabel("Interest Level")
plt.ylabel("Price")
plt.legend()
plt.show()

sns.jointplot(
   data=df_filtered,
   x="price",
   y="bedrooms",
   hue="interest_level",
   kind="scatter",
   color="b",
).plot_joint(sns.kdeplot, zorder=0, levels=6)
plt.xlabel("Prices")
plt.ylabel("Number of Bedrooms")
plt.show()

sns.jointplot(
   data=df_filtered,
   x="price",
   y="bathrooms",
   hue="interest_level",
   kind="scatter",
   color="b",
).plot_joint(sns.kdeplot, zorder=0, levels=6)
plt.xlabel("Prices")
plt.ylabel("Number of Bathrooms")
plt.show()

sns.rugplot(data=df_filtered, x="price", height=0.5)

# add means and variance to plot
plt.axvline(x=df_filtered["price"].mean(), color="r", linestyle="--",
label="Mean")
plt.axvline(x=df_filtered["price"].median(), color="g", linestyle="-",
label="Median")

plt.title("Rug Plot of Prices")
plt.xlabel("Price")
plt.ylabel("Density")
plt.legend()
plt.show()

sampled_df = df_filtered[["bedrooms", "bathrooms",
"price"]].sample(n=1000)

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection="3d")
```

```
scatter = ax.scatter(
   sampled_df["bedrooms"], sampled_df["bathrooms"], sampled_df["price"],
label="Price"
)
ax.set_xlabel("Bedrooms")
ax.set_ylabel("Bathrooms")
ax.set_zlabel("Price")
plt.title("3D Plot of Price by Bedrooms and Bathrooms")
ax.legend([scatter], ["Price"])
plt.show()

xi = np.linspace(sampled_df["bathrooms"].min(),
sampled_df["bathrooms"].max(), 100)
yi = np.linspace(sampled_df["bedrooms"].min(),
sampled_df["bedrooms"].max(), 100)
zi = griddata(
   (sampled_df["bathrooms"], sampled_df["bedrooms"]),
   sampled_df["price"],
   (xi[None, :], yi[:, None]),
   method="linear",
)

fig = plt.figure(figsize=(10, 8))
contour = plt.contourf(xi, yi, zi, 14, cmap=plt.cm.RdBu_r)
plt.colorbar(contour, label="Price")
plt.xlabel("Bathrooms")
plt.ylabel("Bedrooms")
plt.title("Contour Plot of Price by Bedrooms and Bathrooms")
plt.show()

x = df["bedrooms"]
y = df["bathrooms"]
z = df["price"]

# create grid values first
xi = np.linspace(x.min(), x.max(), 100)
yi = np.linspace(y.min(), y.max(), 100)
xi, yi = np.meshgrid(xi, yi)
```

```
# interpolate
zi = griddata((x, y), z, (xi, yi), method="cubic")

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection="3d")

# plot the surface.
surf = ax.contourf(xi, yi, zi, 50, cmap="viridis", extend3d=True)

ax.set_xlabel("Bedrooms")
ax.set_ylabel("Bathrooms")
ax.set_zlabel("Price")
cbar = fig.colorbar(surf, shrink=0.5, aspect=5)
cbar.set_label("Price")

plt.title("3D Contour Plot of Price by Bedrooms and Bathrooms")
plt.show()

numerical_cols = ["bathrooms", "bedrooms", "latitude", "longitude",
"price"]
df_numerical = df_filtered[numerical_cols]
df_numerical = df_numerical.dropna()
sns.clustermap(
   df_numerical, method="average", cmap="coolwarm", standard_scale=1,
figsize=(10, 10)
)
plt.show()

df_zoomed = df_filtered[
    (df_filtered["latitude"] > 40.7)
    & (df_filtered["latitude"] < 40.8)
    & (df_filtered["longitude"] > -74)
    & (df_filtered["longitude"] < -73.9)
]

plt.hexbin(
   df_zoomed["longitude"], df_zoomed["latitude"], gridsize=50,
cmap="Blues", mincnt=1
)
```

```
cb = plt.colorbar(label="count in bin")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.title("Zoomed Hexbin of Listings by Latitude and Longitude")
plt.xlim(-74, -73.9)
plt.ylim(40.7, 40.8)
plt.show()

df_interest_label = ["low", "medium", "high"]
sns.stripplot(x="interest_level", y="price", data=df_filtered,
hue="price")
plt.ylabel("Price")
plt.xlabel("Interest Level")
plt.title("Price vs. Interest Level")
plt.show()

df_interest_label = ["low", "medium", "high"]
sns.stripplot(x="bedrooms", y="price", data=df_filtered, hue="price")
plt.ylabel("Price")
plt.xlabel("Bedrooms")
plt.title("Bedrooms vs. Interest Level")
plt.show()

df_interest_label = ["low", "medium", "high"]
sns.stripplot(x="bathrooms", y="price", data=df_filtered, hue="price")
plt.ylabel("Price")
plt.xlabel("Bathrooms")
plt.title("Bathrooms vs. Interest Level")
plt.show()

df_sampled = df_filtered[df_filtered["price"] < 2000]  # Adjust the
range as needed

sns.swarmplot(x="interest_level", y="price", data=df_sampled)

plt.title("Swarm Plot of Prices by Interest Level")
plt.xlabel("Interest Level")
plt.ylabel("Price")
plt.show()

fig, axs = plt.subplots(2, 2, figsize=(15, 10))

# Price vs Interest Level
df_interest_label = ["low", "medium", "high"]
sns.boxplot(ax=axs[0, 0], x="interest_level", y="price",
data=df_filtered)
axs[0, 0].set_title("Price Distribution by Interest Level")
axs[0, 0].set_xlabel("Interest Level")
axs[0, 0].legend(df_interest_label, loc="best")
axs[0, 0].set_ylabel("Price")

# Bedrooms vs Interest Level
sns.countplot(ax=axs[0, 1], x="bedrooms", hue="interest_level",
data=df_filtered)
axs[0, 1].set_title("Bedroom Count by Interest Level")
axs[0, 1].set_xlabel("Number of Bedrooms")
axs[0, 1].set_ylabel("Count")

# Bathrooms vs Interest Level
sns.countplot(ax=axs[1, 0], x="bathrooms", hue="interest_level",
data=df_filtered)
axs[1, 0].set_title("Bathroom Count by Interest Level")
axs[1, 0].set_xlabel("Number of Bathrooms")
axs[1, 0].set_ylabel("Count")

# Violin plot of interest rate vs price
sns.violinplot(
   ax=axs[1, 1],
   x="interest_level",
   y="price",
   order=["low", "medium", "high"],
   data=df_filtered[df_filtered.price <=
df_filtered.price.quantile(0.99)],
)
axs[1, 1].set_title("Violin Plot of Interest Rate vs. Price")
axs[1, 1].set_xlabel("Interest Level")
axs[1, 1].set_ylabel("Price")
```

```
fig.suptitle("Exploratory Data Analysis")
plt.tight_layout()
plt.show()

fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Percentage of interest level
df_interest_split = df.interest_level.value_counts().values
df_interest_label = ["low", "medium", "high"]
explode = (0.1, 0, 0)
axes[0, 0].pie(
   df_interest_split,
   explode=explode,
   labels=df_interest_label,
   autopct="%1.1f%%",
   shadow=True,
   startangle=140,
)
axes[0, 0].set_title("Percentages of Interests")
axes[0, 0].grid(True)

# Count of listings by number of bedrooms and interest level
sns.barplot(
   x="bedrooms",
   y="price",
   data=df_filtered,
   label="Average Price of Bedrooms",
   ax=axes[0, 1],
)
axes[0, 1].set_title("Average Price by Number of Bedrooms")
axes[0, 1].legend()
axes[0, 1].grid(True)

# Count of listings by number of bathrooms and interest level
sns.barplot(
   x="bathrooms",
   y="price",
   data=df_filtered,
    label="Average Price of Bathrooms",
    ax=axes[1, 0],
)
axes[1, 0].set_title("Average Price by Number of Bathrooms")
axes[1, 0].legend()
axes[1, 0].grid(True)

# Scatter plot of price vs latitude
sns.lineplot(data=df_filtered, x="date", y="price", label="Price")
axes[1, 1].set_title("Line Plot of Prices")
axes[1, 1].grid(True)

fig.suptitle("Exploratory Data Analysis")
plt.tight_layout()
plt.show()

fig, axes = plt.subplots(2, 2, figsize=(15, 10))

# Distplot of prices
sns.distplot(df_filtered["price"], ax=axes[0, 0])
axes[0, 0].axvline(
    x=df_filtered["price"].mean(), color="r", linestyle="--",
label="Mean"
)
axes[0, 0].axvline(
    x=df_filtered["price"].median(), color="g", linestyle="-",
label="Median"
)
axes[0, 0].set_title("Distribution Plot of Prices")
axes[0, 0].legend()
axes[0, 0].grid(True)

# Heatmap of features
corr = df_filtered[["bathrooms", "bedrooms", "price"]].corr()
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", cbar=True,
ax=axes[0, 1])
cbar = plt.gcf().axes[-1]
axes[0, 1].set_title("Correlation Heatmap")
axes[0, 1].grid(True)
```

```
# Regression plot between bedrooms and price
sns.regplot(
   x="bedrooms", y="price", data=df_filtered, scatter=True,
fit_reg=True, ax=axes[1, 0]
)
axes[1, 0].set_title("Regression Plot with Scatter Representation")
axes[1, 0].grid(True)

# Regression plot between bathrooms and price
sns.regplot(
   x="bathrooms",
   y="price",
   data=df_filtered,
   scatter=True,
   fit_reg=True,
   ax=axes[1, 1],
)
axes[1, 1].set_title("Regression Plot with Scatter Representation")
axes[1, 1].grid(True)

fig.suptitle("Exploratory Data Analysis")
plt.tight_layout()
plt.show()

fig, axes = plt.subplots(2, 2, figsize=(15, 10))
df_interest_label = ["low", "medium", "high"]

# Hexbin of NYC latitude and longitude
df_zoomed = df_filtered[
   (df_filtered["latitude"] > 40.7)
   & (df_filtered["latitude"] < 40.8)
   & (df_filtered["longitude"] > -74)
   & (df_filtered["longitude"] < -73.9)
]
axes[0, 0].hexbin(
   df_zoomed["longitude"], df_zoomed["latitude"], gridsize=50,
cmap="Blues", mincnt=1
)
```

```
axes[0, 0].set_title("Zoomed Hexbin of Listings by Latitude and
Longitude")
axes[0, 0].set_xlabel("Longitude")
axes[0, 0].set_ylabel("Latitude")
axes[0, 0].grid(True)

# Price vs Interests
sns.stripplot(
   x="interest_level", y="price", data=df_filtered, hue="price",
ax=axes[0, 1]
)
axes[0, 1].set_title("Price vs. Interest Level")
axes[0, 1].set_xlabel("Price")
axes[0, 1].set_ylabel("Interest Level")
axes[0, 1].grid(True)

# Price vs Number of Bedrooms
sns.stripplot(x="bedrooms", y="price", data=df_filtered, hue="price",
ax=axes[1, 0])
axes[1, 0].set_title("Bedrooms vs. Price")
axes[1, 0].set_xlabel("Bedrooms")
axes[1, 0].set_ylabel("Price")
axes[1, 0].grid(True)

# Price vs Number of Bathrooms
sns.stripplot(x="bathrooms", y="price", data=df_filtered, hue="price",
ax=axes[1, 1])
axes[1, 1].set_title("Bathrooms vs. Price")
axes[1, 1].set_xlabel("Bathrooms")
axes[1, 1].set_ylabel("Price")
axes[1, 1].grid(True)

fig.suptitle("Exploratory Data Analysis")
plt.tight_layout()
plt.show()

corr = df_filtered[["bathrooms", "bedrooms", "price"]].corr()
table = PrettyTable()
table.field_names = ["Feature"] + list(corr.columns)
```

```
for index, row in corr.iterrows():
   table.add_row([index] + list(row))
print(table)
```

EDA with PCA & Normality Test

```
# import libraries
import numpy as np
import pandas as pd
from collections import Counter
import re
import os

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from numpy.linalg import svd
import scipy.stats as st

import warnings

warnings.filterwarnings("ignore")

# import the dataset (this dataset has already remove outlier)
df = pd.read_json("../data/sentimental_extraction_kaggle.json")
df.head(5)

print(f"There are {len(df)} samples.")

# check the first 3 row's features list
print(df.features.iloc[0])
print(df.features.iloc[1])
print(df.features.iloc[2])

# flatten the list of features from all rows
all_features = [feature for sublist in df["features"] for feature in
sublist]
```

```
# count the frequency of each feature
feature_counts = Counter(all_features)

# filter features that have a frequency above the threshold
frequency_threshold = 10000
high_freq_features = [
   feature for feature, count in feature_counts.items() if count >=
frequency_threshold
]

# create binary columns for each high-frequency feature
for feature in high_freq_features:
   df["feature_" + feature.lower()] = df["features"].apply(
       lambda x: 1 if feature in x else 0
   )

print(f"The features with most (>= 10k) counts are
{high_freq_features}")

# convert target interest_level from categorical to numerical
interest_level_mapping = {"high": 1, "medium": 0, "low": -1}
df["interest_level"] = df["interest_level"].map(interest_level_mapping)

# extract numerical features
final_df = df[
   [
       "bathrooms",
       "bedrooms",
       "price",
       "sentiment_label",
       "feature_laundry in building",
       "feature_dishwasher",
       "feature_hardwood floors",
       "feature_dogs allowed",
       "feature_cats allowed",
       "feature_doorman",
       "feature_elevator",
       "feature_no fee",
       "feature_fitness center",
       "interest_level",
   ]
]

# standardize the dataset
scaler = StandardScaler()
df_standardized = scaler.fit_transform(final_df)

# perform PCA without limiting the number of components to preserve all
the variance
pca = PCA()
pca.fit(df_standardized)

# find the number of components required to explain at least 95%
variance
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
num_components = (
   np.argmax(cumulative_variance >= 0.95) + 1
)  # adding 1 because of 0-based indexing

# number of features to be removed
num_features_removed = df_standardized.shape[1] - num_components

# perform PCA again, this time with the desired number of components
pca_reduced = PCA(n_components=num_components)
pca_reduced.fit(df_standardized)

print(f"Number of features to be removed: {num_features_removed}")
print(
   "Explained Variance Ratio of Original Feature Space:",
pca.explained_variance_ratio_
)
print(
   "Explained Variance Ratio of Reduced Feature Space:",
   pca_reduced.explained_variance_ratio_,
)

# calculate the cumulative explained variance
cumulative_variance = np.cumsum(pca.explained_variance_ratio_)
```

```python
# plotting the cumulative explained variance
plt.figure(figsize=(10, 7))
plt.plot(
   range(1, len(cumulative_variance) + 1),
   cumulative_variance,
   marker="o",
   linestyle="--",
)
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance (%)")
plt.title("Cumulative Explained Variance vs. Number of Components")

# drawing the vertical and horizontal dashed lines for 95% variance
optimum_components = np.argmax(cumulative_variance >= 0.95) + 1
plt.axvline(
   x=optimum_components,
   color="black",
   linestyle="--",
   label=f"Optimum Components = {optimum_components}",
)
plt.axhline(y=0.95, color="red", linestyle="--", label="95% Variance")

plt.legend()
plt.grid(True)
plt.show()

# perform K-S test for normality on bedrooms
ks_statistic_bedrooms, ks_pvalue_bedrooms = st.kstest(
   final_df["bedrooms"],
   "norm",
   args=(final_df["bedrooms"].mean(), final_df["bedrooms"].std()),
)

# perform K-S test for normality on bathrooms
ks_statistic_bathrooms, ks_pvalue_bathrooms = st.kstest(
   final_df["bathrooms"],
   "norm",
   args=(final_df["bathrooms"].mean(), final_df["bathrooms"].std()),
)

print(
   f"K-S test: statistics={ks_statistic_bedrooms:.2f}
p-value={ks_pvalue_bedrooms:.2f}"
)
print(
   f"K-S test: Number of bedrooms looks {'normal' if ks_pvalue_bedrooms
> 0.01 else 'not normal'} with 99% accuracy"
)
print(
   f"K-S test: statistics={ks_statistic_bathrooms:.2f}
p-value={ks_pvalue_bathrooms:.2f}"
)
print(
   f"K-S test: Number of bathrooms looks {'normal' if
ks_pvalue_bathrooms > 0.01 else 'not normal'} with 99% accuracy"
)

# perform Shapiro-Wilk test for normality on bedrooms
shapiro_statistic_bedrooms, shapiro_pvalue_bedrooms =
st.shapiro(final_df["bedrooms"])

# perform Shapiro-Wilk test for normality on bathrooms
shapiro_statistic_bathrooms, shapiro_pvalue_bathrooms = st.shapiro(
   final_df["bathrooms"]
)

print(
   f"Shapiro-Wilk test: statistics={shapiro_statistic_bedrooms:.2f}
p-value={shapiro_pvalue_bedrooms:.2f}"
)
print(
   f"Shapiro-Wilk test: Number of bedrooms looks {'normal' if
shapiro_pvalue_bedrooms > 0.01 else 'not normal'} with 99% accuracy"
)
print(
   f"Shapiro-Wilk test: statistics={shapiro_statistic_bathrooms:.2f}
p-value={shapiro_pvalue_bathrooms:.2f}"
```

```
)
print(
   f"Shapiro-Wilk test: Number of bathrooms looks {'normal' if
shapiro_pvalue_bathrooms > 0.01 else 'not normal'} with 99% accuracy"
)

# perform DA test for normality on bedrooms
da_statistic_bedrooms, da_pvalue_bedrooms =
st.normaltest(final_df["bedrooms"])

# perform DA test for normality on bathrooms
da_statistic_bathrooms, da_pvalue_bathrooms =
st.normaltest(final_df["bathrooms"])

print(
   f"D'Agostino's K^2 test: statistics={da_statistic_bedrooms:.2f}
p-value={da_pvalue_bedrooms:.2f}"
)
print(
   f"D'Agostino's K^2 test: Number of bedrooms looks {'normal' if
da_pvalue_bedrooms > 0.01 else 'not normal'} with 99% accuracy"
)
print(
   f"D'Agostino's K^2 test: statistics={da_statistic_bathrooms:.2f}
p-value={da_pvalue_bathrooms:.2f}"
)
print(
   f"D'Agostino's K^2 test: Number of bathrooms looks {'normal' if
da_pvalue_bathrooms > 0.01 else 'not normal'} with 99% accuracy"
)
```

Dash main

```
# dash imports
import dash
from dash import html
from dash import Input
from dash import Output
from dash import dcc
import dash_bootstrap_components as dbc

# file imports
from maindash import my_app
from components.overview import overview
from components.analysis import analysis
from components.visualization import visualization
from components.interest_level_prediction import
interest_level_prediction
from components.price_prediction import price_prediction
from components.virtual_assistant import virtual_assistant
from components.listing import listing
from components.about import about
from components.overview import overview

#######################################
# Initial Settings
#######################################
server = my_app.server

CONTENT_STYLE = {
    "transition": "margin-left .1s",
    "padding": "1rem 1rem",
}

#######################################
# Layout
########################################
sidebar = html.Div(
    [
        html.Div(
            [
                html.H2("RentalGPT", style={"color": "white"}),
            ],
            className="sidebar-header",
        ),
        html.Br(),
        html.Div(style={"border-top": "2px solid white"}),
        html.Br(),
        # nav component
        dbc.Nav(
            [
                dbc.NavLink(
                    [
                        html.I(className="fas fa-solid fa-star me-2"),
                        html.Span("Overview"),
                    ],
                    href="/",
                    active="exact",
                ),
                dbc.NavLink(
                    [
                        html.I(className="fas fa-home me-2"),
                        html.Span("Apartments Listing"),
                    ],
                    href="/listing",
                    active="exact",
                ),
                dbc.NavLink(
                    [
                        html.I(className="fas fa-solid fa-chart-simple
me-2"),
                        html.Span("Data Analysis"),
                    ],
                    href="/analysis",
                    active="exact",
                ),
                dbc.NavLink(
                    [
                        html.I(className="fas fa-solid fa-sliders me-2"),
                        html.Span("Data Visualization"),
                    ],
                    href="/visualization",
                    active="exact",
                ),
                dbc.NavLink(
                    [
                        html.I(className="fas fa-solid fa-people-group
me-2"),
                        html.Span("Interest Level Prediction"),
                    ],
                    href="/interest_level_prediction",
                    active="exact",
                ),
                dbc.NavLink(
                    [
                        html.I(className="fas fa-solid fa-arrow-trend-up
me-2"),
                        html.Span("Rental Cost Prediction"),
                    ],
                    href="/price_prediction",
                    active="exact",
                ),
                # dbc.NavLink(
                #     [
                #         html.I(className="fas fa-solid fa-eye me-2"),
                #         html.Span("Generative Apartment"),
                #     ],
                #     href="/gen_apartment",
                #     active="exact",
                # ),
                dbc.NavLink(
                    [
                        html.I(className="fas fa-solid fa-comments
me-2"),
                        html.Span("Virtual Assistant"),
                    ],
                    href="/virtual_assistant",
                    active="exact",
                ),
                dbc.NavLink(
                    [
                        html.I(className="fas fa-solid fa-code me-2"),
                        html.Span("About"),
                    ],
                    href="/about",
                    active="exact",
                ),
            ],
            vertical=True,
            pills=True,
        ),
    ],
    className="sidebar",
)


my_app.layout = html.Div(
    [
        dcc.Location(id="url"),
        sidebar,
        html.Div(
            [
                dash.page_container,
            ],
            className="content",
            style=CONTENT_STYLE,
            id="page-content",
        ),
    ]
)


@my_app.callback(Output("page-content", "children"), [Input("url",
"pathname")])
def render_page_content(pathname):
    if pathname == "/":
        return overview.overview_layout()
    elif pathname == "/listing":
        return listing.listing_layout()
    elif pathname == "/analysis":
        return analysis.analysis_layout()
    elif pathname == "/visualization":
        return visualization.visualization_layout()
    elif pathname == "/interest_level_prediction":
        return
interest_level_prediction.interest_level_prediction_layout()
    elif pathname == "/price_prediction":
        return price_prediction.price_prediction_layout()
    # elif pathname == "/gen_apartment":
    #     pass
    elif pathname == "/virtual_assistant":
        return virtual_assistant.virtual_assistant_layout()
    elif pathname == "/about":
        return about.about_layout()
    return dbc.Container(
        children=[
            html.H1(
                "404 Error: Page Not found",
                style={"textAlign": "center", "color": "#082446"},
            ),
            html.Br(),
            html.P(
                f"Oh no! The pathname '{pathname}' was not
recognised...",
                style={"textAlign": "center"},
            ),
            # image
            html.Div(
                style={"display": "flex", "justifyContent": "center"},
                children=[
                    html.Img(

src="https://elephant.art/wp-content/uploads/2020/02/gu_announcement_01-
1.jpg",
                        alt="hokie",
                        style={"width": "400px"},
                    ),
                ],
            ),
        ]
    )
```

```
if __name__ == "__main__":
   my_app.run_server(debug=True, host="0.0.0.0", port=80)
```

Overview Page

```
# dash imports
import dash
from dash import html
from dash import Input
from dash import Output
from dash import dcc
import dash_bootstrap_components as dbc

# file imports
from maindash import my_app
from utils.file_operation import read_file_as_str


#######################################
# Layout
#######################################
def overview_layout():
   layout = html.Div(
       [
           html.Div(
               [
                   html.Div(
                       [
                           html.Img(

src="https://images.unsplash.com/photo-1614850523425-eec693b15af5?q=80&w
=1470&auto=format&fit=crop&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wY
WdlfHx8fGVufDB8fHx8fA%3D%3D",
                               style={
                                   "width": "100%",
                                   "height": "auto",
                                   "position": "relative",
                               },
                           ),
                       ],
                       style={
                           "height": "200px",
                           "overflow": "hidden",
                           "position": "relative",
                       },
                   ),
                   html.H1(
                       "App Overview",
                       style={
                           "position": "absolute",
                           "top": "80%",
                           "left": "50%",
                           "transform": "translate(-50%, -50%)",
                           "color": "white",
                           "text-align": "center",
                           "width": "100%",
                       },
                   ),
               ],
               style={
                   "position": "relative",
                   "text-align": "center",
                   "color": "white",
               },
           ),
           html.Br(),
           html.H3("🌟 Services"),
           dcc.Markdown(

children=read_file_as_str("./utils/markdown/overview/overview.md"),
               mathjax=True,
           ),
           html.Br(),
           card(),
       ]
   )
   return layout


def card():
   layout = html.Div(
       # style={"width":"100px"},
       children=[
           dbc.Row(
               style={"display": "flex", "justifyContent": "flex-end"},
               children=[
                   dbc.Col(
                       [
                           dbc.Card(
                               [
                                   dbc.CardImg(

src="https://www.investopedia.com/thmb/bfHtdFUQrl7jJ_z-utfh8w1TMNA=/1500
x0/filters:no_upscale():max_bytes(150000):strip_icc()/houses_and_land-5b
fc3326c9e77c0051812eb3.jpg",
                                        top=True,
                                        style={
                                            "opacity": 0.18,
                                            "height": "200px",
                                        },
                                   ),
                                   dbc.CardImgOverlay(
                                        dbc.CardBody(
                                            [
                                                html.Div(
                                                    style={"height":
"100px"},

                                                    children=[
                                                        html.H4(
                                                            "New York
City Apartments Listing",

className="card-title text-center",  # apply text-center class
                                                        ),
                                                    ],
                                                ),
                                                dbc.Button(
                                                    "Explore",
                                                    color="dark",

href="https://rentalgpt.discovery.cs.vt.edu/listing",
                                                    className="mx-auto
d-block",  # center the button horizontally
                                                ),
                                            ],
                                            className="text-center",  #
center the card body content vertically
                                        ),
                                    ),
                                ],
                                style={"width": "18rem"},
                            ),
                            html.Br(),
                        ]
                    ),
                    html.Br(),
                    dbc.Col(
                        [
                            dbc.Card(
                                [
                                    dbc.CardImg(

src="https://cdn.thenewstack.io/media/2023/01/285d68dd-charts-1024x581.j
pg",
                                        top=True,
                                        style={
                                            "opacity": 0.18,
                                            "height": "200px",
                                        },
                                    ),
                                    dbc.CardImgOverlay(
                                        dbc.CardBody(
                                            [
                                                html.Div(
                                                    style={"height":
 "100px"},
                                                    children=[
                                                        html.H4(
                                                            "In-depth
 Data Analysis",

 className="card-title text-center",  # apply text-center class
                                                        ),
                                                    ],
                                                ),
                                                # html.P(
                                                #     "An example of
 using an image in the background of "
                                                #     "a card.",
                                                #
 className="card-text",
                                                # ),
                                                dbc.Button(
                                                    "Explore",
                                                    color="dark",

 href="https://rentalgpt.discovery.cs.vt.edu/analysis",
                                                    className="mx-auto
 d-block",  # center the button horizontally
                                                ),
                                            ],
                                        ),
                                    ),
                                ],
                                style={"width": "18rem"},
                            ),
                            html.Br(),
                        ]
                    ),
                    html.Br(),
                    dbc.Col(
                        [
                            dbc.Card(
                                [
                                    dbc.CardImg(

 src="https://media.licdn.com/dms/image/C4D12AQENxlEKXtt4Tw/article-cover
 _image-shrink_600_2000/0/1588295143025?e=2147483647&v=beta&t=3A_tBd2veQH
 zontm2NcuQ4lSuxL4lRNMeZJ8vw1MkOc",
                                        top=True,
                                        style={
                                            "opacity": 0.18,
                                            "height": "200px",
                                        },
                                    ),
                                    dbc.CardImgOverlay(
                                        dbc.CardBody(
                                            [
                                                html.Div(
                                                    style={"height":
 "100px"},

                                                    children=[
                                                        html.H4(
                                                            "Interactive
 Data Visualization",

 className="card-title text-center",  # apply text-center class
                                                        ),
                                                    ],
                                                ),
                                                # html.P(
                                                #     "An example of
 using an image in the background of "
                                                #     "a card.",
                                                #
 className="card-text",
                                                # ),
                                                dbc.Button(
                                                    "Explore",
                                                    color="dark",

 href="https://rentalgpt.discovery.cs.vt.edu/visualization",
                                                    className="mx-auto
 d-block",  # center the button horizontally
                                                ),
                                            ],
                                        ),
                                    ),
                                ],
                                style={"width": "18rem"},
                            ),
                            html.Br(),
                        ]
                    ),
                    dbc.Col(
                        [
                            dbc.Card(
                                [
                                    dbc.CardImg(

 src="https://thehustle.co/wp-content/uploads/2023/02/HS-News-Brief_2023-
 02-03T015826.025Z.png",
                                        top=True,
                                        style={
                                            "opacity": 0.18,
                                            "height": "200px",
                                        },
                                    ),
                                    dbc.CardImgOverlay(
                                        dbc.CardBody(
                                            [
                                                html.Div(
                                                    style={"height":
 "100px"},
                                                    children=[
                                                        html.H4(
                                                            "Customer's
 Interest Level Prediction",

 className="card-title text-center",  # apply text-center class
                                                        ),
                                                    ],
                                                ),
                                                # html.P(
                                                #     "An example of
 using an image in the background of "
                                                #     "a card.",
                                                #
 className="card-text",
                                                # ),
                                                dbc.Button(
                                                    "Explore",
                                                    color="dark",

 href="https://rentalgpt.discovery.cs.vt.edu/interest_level_prediction",
                                                    className="mx-auto
 d-block",  # center the button horizontally
                                                ),
                                            ],
                                        ),
                                    ),
                                ],
                                style={"width": "18rem"},
                            ),
                        ]
                    ),
                    #     ],
                    #     className="mb-4",
                    # ),
                    # dbc.Row(
                    #     [
                    dbc.Col(
                        [
                            dbc.Card(
                                [
                                    dbc.CardImg(

 src="https://www.vantage-ai.com/hubfs/House%20prices.jpg",
                                        top=True,
                                        style={
                                            "opacity": 0.18,
                                            "height": "200px",
                                        },
                                    ),
                                    dbc.CardImgOverlay(
                                        dbc.CardBody(
                                            [
                                                html.Div(
                                                    style={"height":
"100px"},
                                                    children=[
                                                        html.H4(
                                                            "Monthly
Rental Price Prediction",

className="card-title text-center",  # apply text-center class
                                                        ),
                                                    ],
                                                ),
                                                # html.P(
                                                #     "An example of
using an image in the background of "
                                                #     "a card.",
                                                #
className="card-text",
                                                # ),
                                                dbc.Button(
                                                    "Explore",
                                                    color="dark",

href="https://rentalgpt.discovery.cs.vt.edu/price_prediction",
                                                    className="mx-auto
d-block",  # center the button horizontally
                                                ),
                                            ],
                                        ),
                                    ),
                                ],
                                style={"width": "18rem"},
                            ),
                            html.Br(),
                        ]
                    ),
                    dbc.Col(
                        [
                            dbc.Card(
                                [
                                    dbc.CardImg(

src="https://huggingface.co/wordcab/llama-natural-instructions-7b/resolv
e/main/llama-natural-instructions-removebg-preview.png",
                                        top=True,
                                        style={
                                            "opacity": 0.18,
                                            "height": "200px",
                                        },
                                    ),
                                    dbc.CardImgOverlay(
                                        dbc.CardBody(
                                            [
                                                html.Div(
                                                    style={"height":
"100px"},
                                                    children=[
                                                        html.H4(
                                                            "LLaMA
Virtual Assistant",

className="card-title text-center",  # apply text-center class
                                                        ),
                                                    ],
                                                ),
                                                # html.P(
                                                #     "An example of
using an image in the background of "
                                                #     "a card.",
                                                #
className="card-text",
                                                            # ),
                                                            dbc.Button(
                                                                "Explore",
                                                                color="dark",

href="https://rentalgpt.discovery.cs.vt.edu/virtual_assistant",
                                                                className="mx-auto
d-block",  # center the button horizontally
                                                            ),
                                                        ],
                                                    ),
                                                ),
                                            ],
                                            style={"width": "18rem"},
                                        ),
                                    ]
                                ),
                            ],
                            className="mb-4",
                        ),
                        # html.Br(),
                        # html.Hr(),
                        # html.H3("📱Mobile App"),
                    ],
                )
    return layout
```

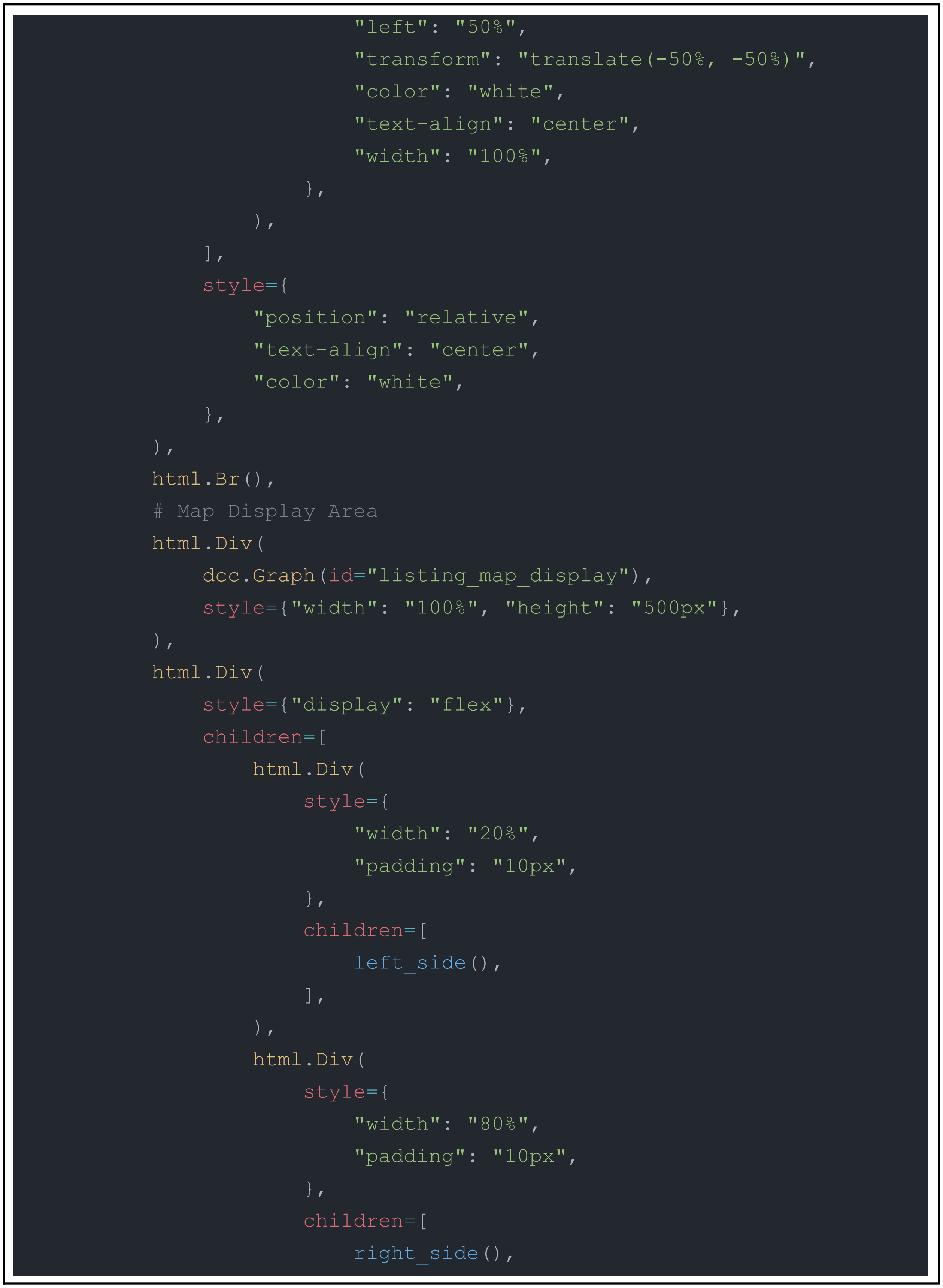
Apartments Listing Page

```
# dash imports
import dash
from dash import html
from dash import Input
from dash import Output
from dash import dcc
import dash_bootstrap_components as dbc
import sqlite3
import plotly.express as px
import pandas as pd

# file imports
from maindash import my_app


#######################################
# Layout
#######################################
def listing_layout():
    layout = html.Div(
        [
            html.Div(
                [
                    html.Div(
                        [
                            html.Img(

src="https://images.unsplash.com/photo-1641160858304-6aded85fa2c4?q=80&w
=1332&auto=format&fit=crop&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wY
WdlfHx8fGVufDB8fHx8fA%3D%3D",
                                style={
                                    "width": "100%",
                                    "height": "auto",
                                    "position": "relative",
                                },
                            ),
                        ],
                        style={
                            "height": "200px",
                            "overflow": "hidden",
                            "position": "relative",
                        },
                    ),
                    html.H1(
                        "New York City Apartments Listing",
                        style={
                            "position": "absolute",
                            "top": "80%",
```



```
                            "left": "50%",
                            "transform": "translate(-50%, -50%)",
                            "color": "white",
                            "text-align": "center",
                            "width": "100%",
                        },
                    ),
                ],
                style={
                    "position": "relative",
                    "text-align": "center",
                    "color": "white",
                },
            ),
            html.Br(),
            # Map Display Area
            html.Div(
                dcc.Graph(id="listing_map_display"),
                style={"width": "100%", "height": "500px"},
            ),
            html.Div(
                style={"display": "flex"},
                children=[
                    html.Div(
                        style={
                            "width": "20%",
                            "padding": "10px",
                        },
                        children=[
                            left_side(),
                        ],
                    ),
                    html.Div(
                        style={
                            "width": "80%",
                            "padding": "10px",
                        },
                        children=[
                            right_side(),
                        ],
                    ),
                ],
            ),
        ]
    )

    return layout


def left_side():
    return html.Div(
        [
            html.Div([html.H3("🎛️ Tune")]),
            html.Div(
                [
                    html.Label("Number of Bathrooms",
style={"fontWeight": "bold"}),
                    dcc.Slider(
                        id="listing_bathrooms_filter",
                        min=0,
                        max=5,
                        step=0.5,
                        value=1,
                        marks=None,
                        tooltip={"placement": "bottom", "always_visible":
True},
                    ),
                    html.Br(),
                    html.Label("Number of Bedrooms", style={"fontWeight":
"bold"}),
                    dcc.Slider(
                        id="listing_bedrooms_filter",
                        min=0,
                        max=5,
                        step=1,
                        value=1,
                        marks=None,
                        tooltip={"placement": "bottom", "always_visible":
```

```
True},
                   ),
                   html.Br(),
                   html.Label("Sort by Price", style={"fontWeight":
"bold"}),
                   dcc.Dropdown(
                       id="listing_price_sorting",
                       options=[
                           {"label": "No Sorting", "value": "none"},
                           {"label": "Low to High", "value": "asc"},
                           {"label": "High to Low", "value": "desc"},
                       ],
                       value="none",
                   ),
                   html.Br(),
                   html.Label("Filter by Preferences",
style={"fontWeight": "bold"}),
                   dcc.Checklist(
                       id="listing_features_filter",
                       options=[
                           {
                               "label": "Laundry in Building",
                               "value": "feature_laundry_in_building",
                           },
                           {"label": "Dishwasher", "value":
"feature_dishwasher"},
                           {
                               "label": "Hardwood Floors",
                               "value": "feature_hardwood_floors",
                           },
                           {"label": "Dogs Allowed", "value":
"feature_dogs_allowed"},
                           {"label": "Cats Allowed", "value":
"feature_cats_allowed"},
                           {"label": "Doorman", "value":
"feature_doorman"},
                           {"label": "Elevator", "value":
"feature_elevator"},
                           {
                               "label": "Fitness Center",
                               "value": "feature_fitness_center",
                           },
                       ],
                       value=["feature_laundry_in_building"],
                   ),
               ],
               style={"display": "flex", "flexDirection": "column",
"padding": "10px"},
           ),
       ]
   )


def right_side():
   return html.Div(
       [
           dcc.Loading(
               id="loading-cards",
               children=[html.Div(id="listing_cards_container")],
               type="circle",
           )
       ]
   )


def query_database(filters):
   con = sqlite3.connect("database.db")

   # SQL query based on preference
   query = """
   SELECT * FROM properties
   WHERE bathrooms BETWEEN ? AND ?
     AND bedrooms BETWEEN ? AND ?
     AND {}  -- this will be formatted with the correct string for
features
   """

   # handling binary feature filters
   feature_conditions = []
   for feature in filters["features"]:
       feature_conditions.append(f'"{feature}" = 1')
   feature_query = " AND ".join(feature_conditions) if
feature_conditions else "1=1"

   # formatting the query with the feature conditions
   query = query.format(feature_query)

   df = pd.read_sql_query(
       query,
       con,
       params=[
           filters["bathrooms"][0],
           filters["bathrooms"][1],
           filters["bedrooms"][0],
           filters["bedrooms"][1],
       ],
   )

   # categorize the listings into terciles
   terciles = df["price"].quantile([1 / 3, 2 / 3]).tolist()

   def categorize_price(price):
       if price <= terciles[0]:
           return "Poor"
       elif price <= terciles[1]:
           return "Median"
       else:
           return "Rich"

   df["price_category"] = df["price"].apply(categorize_price)

   con.close()
   return df


@my_app.callback(
   [
       Output("listing_cards_container", "children"),
       Output("listing_map_display", "figure"),
   ],
   [
       Input("listing_bathrooms_filter", "value"),
       Input("listing_bedrooms_filter", "value"),
       Input("listing_features_filter", "value"),
       Input("listing_price_sorting", "value"),  # Add input for sorting
   ],
)
def update_content(bathrooms, bedrooms, features, sort_order):
   filters = {
       "bathrooms": [bathrooms, bathrooms + 0.5],
       "bedrooms": [bedrooms, bedrooms],
       "features": features,
   }
   filtered_df = query_database(filters)

   # apply sorting based on user selection
   if sort_order == "asc":
       filtered_df = filtered_df.sort_values(by="price", ascending=True)
   elif sort_order == "desc":
       filtered_df = filtered_df.sort_values(by="price",
ascending=False)

   # create cards and update map as before
   cards_content = html.Div(
       create_cards(filtered_df), style={"display": "flex", "flexWrap":
"wrap"}
   )
   fig = px.scatter_mapbox(
       filtered_df,
       lat="latitude",
       lon="longitude",
       zoom=10,
       center=dict(lat=40.7128, lon=-74.0060),
       mapbox_style="open-street-map",
       height=500,
   )
   return cards_content, fig


def create_cards(df):
   cards = []
   for _, row in df.iterrows():
       # define header color based on price category
       header_color, money_bags = {
           "Poor": ("lightgreen", "💰"),  # 1 money bag for Poor
           "Median": ("lightblue", "💰💰"),  # 2 money bags for Median
           "Rich": ("salmon", "💰💰💰"),  # 3 money bags for Rich
       }.get(row["price_category"], ("lightgrey", ""))

       header_style = {"backgroundColor": header_color}

       # truncate the description to 100 characters to standardize card
size
       description = (
           row["description"]
           if len(row["description"]) <= 100
           else row["description"][:100] + "..."
       )

       card_content = [
           dbc.CardHeader(
               [
                   html.Span(f"ID: {row['listing_id']}",
style={"flexGrow": 1}),
                   html.Span(money_bags, style={"marginLeft": "20px"}),
               ],
               style=header_style,
           ),
           dbc.CardBody(
               [
                   html.H5(f"🏡 {row['street_address']}",
className="card-title"),
                   html.P([html.Strong("About: "), f"{description}"]),
                   html.P([html.Strong("Price: "), f"${row['price']}"]),
                   html.P([html.Strong("Bathrooms: "),
f"{row['bathrooms']}"]),
                   html.P([html.Strong("Bedrooms: "),
f"{row['bedrooms']}"]),
               ]
           ),
       ]
       cards.append(dbc.Card(card_content, style={"width": "18rem",
"margin": "10px"}))
   return cards
```

Data Analysis Page

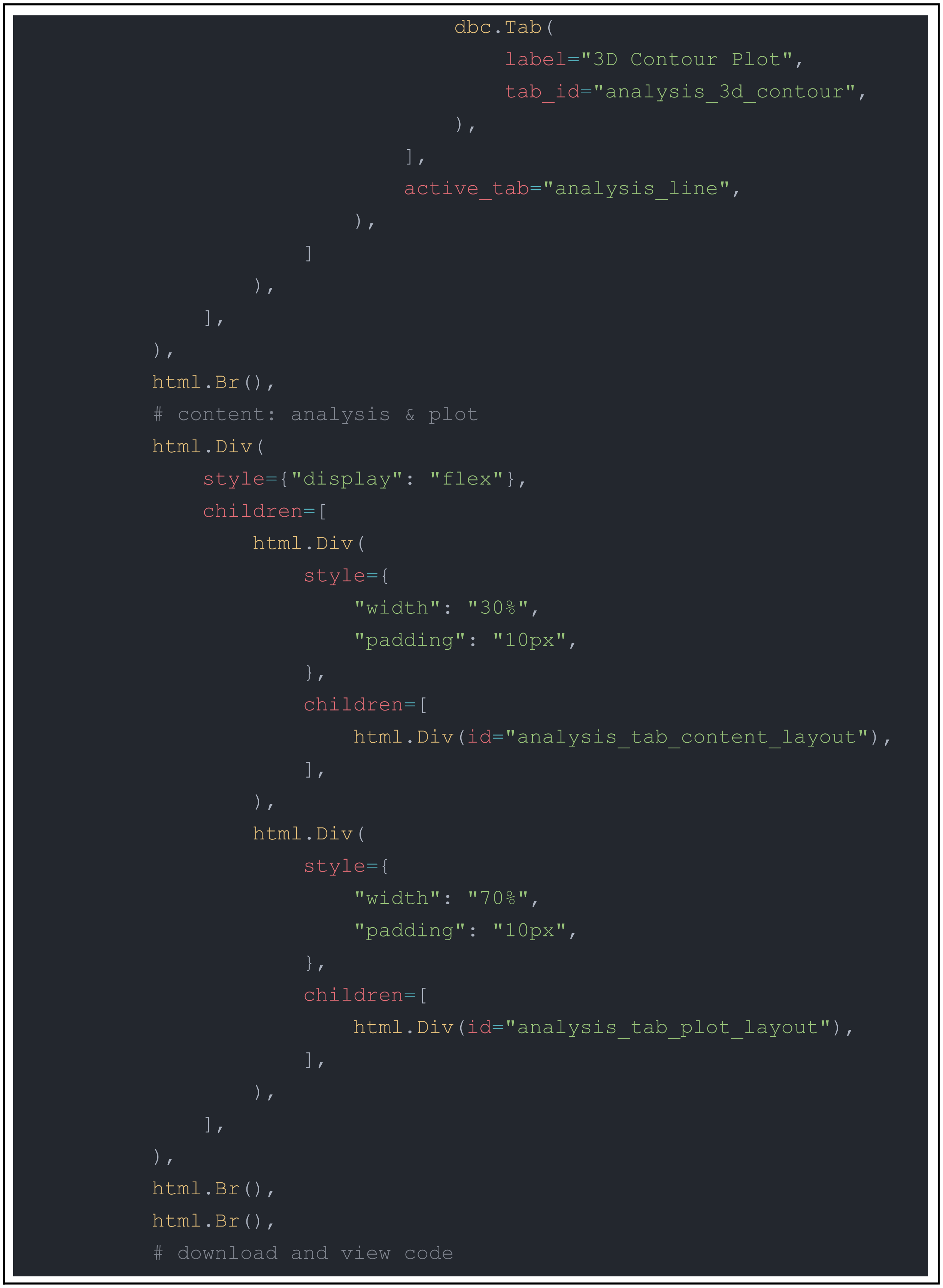
```
# dash imports
import dash
from dash import html
from dash import Input
from dash import Output
from dash import dcc
import dash_bootstrap_components as dbc

# file imports
from maindash import my_app
from components.analysis.line_plot import line_plot_info
from components.analysis.bar_plot_1 import bar_plot_1_info
from components.analysis.bar_plot_2 import bar_plot_2_info
from components.analysis.count_plot_1 import count_plot_1_info
from components.analysis.count_plot_2 import count_plot_2_info
from components.analysis.count_plot_3 import count_plot_3_info
from components.analysis.count_plot_4 import count_plot_4_info
from components.analysis.count_plot_5 import count_plot_5_info
from components.analysis.count_plot_6 import count_plot_6_info
from components.analysis.pie_chart import pie_chart_info
from components.analysis.dist_plot import dist_plot_info
from components.analysis.pair_plot import pair_plot_info
from components.analysis.heatmap import heatmap_info
from components.analysis.qq_plot import qq_plot_info
from components.analysis.reg_plot_1 import reg_plot_1_info
from components.analysis.reg_plot_2 import reg_plot_2_info
from components.analysis.reg_plot_3 import reg_plot_3_info
from components.analysis.area_plot import area_plot_info
from components.analysis.violin_plot import violin_plot_info
from components.analysis.joint_plot_1 import joint_plot_1_info
from components.analysis.joint_plot_2 import joint_plot_2_info
from components.analysis.plot_3d import plot_3d_info
from components.analysis.plot_3d_contour import plot_3d_contour_info


########################################
# Layout
########################################
def analysis_layout():
    layout = html.Div(
        [
            # image
            html.Div(
                [
                    html.Div(
                        [
                            html.Img(

src="https://images.unsplash.com/photo-1614851099511-773084f6911d?q=80&w
=1170&auto=format&fit=crop&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wY
WdlfHx8fGVufDB8fHx8fA%3D%3D",
                                style={
                                    "width": "100%",
                                    "height": "auto",
                                    "position": "relative",
                                },
                            ),
                        ],
                        style={
                            "height": "200px",
                            "overflow": "hidden",
                            "position": "relative",
                        },
                    ),
                    html.H1(
                        "Data Analysis",
                        style={
                            "position": "absolute",
                            "top": "80%",
                            "left": "50%",
                            "transform": "translate(-50%, -50%)",
                            "color": "white",
                            "text-align": "center",
                            "width": "100%",
                        },
                    ),
                ],
                style={
                    "position": "relative",
                    "text-align": "center",
                    "color": "white",
                },
            ),
            html.Br(),
            # tab
            html.Div(
                style={"display": "flex"},
                children=[
                    html.Div(
                        [
                            dbc.Tabs(
                                id="analysis_selected_tab",
                                children=[
                                    dbc.Tab(
                                        label="Line Plot",
                                        tab_id="analysis_line",
                                    ),
                                    dbc.Tab(
                                        label="Bar Plot 1",
                                        tab_id="analysis_bar_1",
                                    ),
                                    dbc.Tab(
                                        label="Bar Plot 2",
```

```
                                        tab_id="analysis_bar_2",
                                    ),
                                    dbc.Tab(
                                        label="Count Plot 1",
                                        tab_id="analysis_count_1",
                                    ),
                                    dbc.Tab(
                                        label="Count Plot 2",
                                        tab_id="analysis_count_2",
                                    ),
                                    dbc.Tab(
                                        label="Count Plot 3",
                                        tab_id="analysis_count_3",
                                    ),
                                    dbc.Tab(
                                        label="Count Plot 4",
                                        tab_id="analysis_count_4",
                                    ),
                                    dbc.Tab(
                                        label="Count Plot 5",
                                        tab_id="analysis_count_5",
                                    ),
                                    dbc.Tab(
                                        label="Count Plot 6",
                                        tab_id="analysis_count_6",
                                    ),
                                    dbc.Tab(
                                        label="Pie Chart",
                                        tab_id="analysis_pie",
                                    ),
                                    dbc.Tab(
                                        label="Dist Plot",
                                        tab_id="analysis_dist",
                                    ),
                                    dbc.Tab(
                                        label="Pair Plot",
                                        tab_id="analysis_pair",
                                    ),
                                    dbc.Tab(
                                        label="Heatmap",
                                        tab_id="analysis_heatmap",
                                    ),
                                    dbc.Tab(
                                        label="QQ Plot",
                                        tab_id="analysis_qq",
                                    ),
                                    dbc.Tab(
                                        label="Reg Plot 1",
                                        tab_id="analysis_reg_1",
                                    ),
                                    dbc.Tab(
                                        label="Reg Plot 2",
                                        tab_id="analysis_reg_2",
                                    ),
                                    dbc.Tab(
                                        label="Reg Plot 3",
                                        tab_id="analysis_reg_3",
                                    ),
                                    dbc.Tab(
                                        label="Area Plot",
                                        tab_id="analysis_area",
                                    ),
                                    dbc.Tab(
                                        label="Violin Plot",
                                        tab_id="analysis_violin",
                                    ),
                                    dbc.Tab(
                                        label="Joint Plot 1",
                                        tab_id="analysis_joint_1",
                                    ),
                                    dbc.Tab(
                                        label="Joint Plot 2",
                                        tab_id="analysis_joint_2",
                                    ),
                                    dbc.Tab(
                                        label="3D Plot",
                                        tab_id="analysis_3d",
                                    ),
```



```
                                    dbc.Tab(
                                        label="3D Contour Plot",
                                        tab_id="analysis_3d_contour",
                                    ),
                                ],
                                active_tab="analysis_line",
                            ),
                        ]
                    ),
                ],
            ),
            html.Br(),
            # content: analysis & plot
            html.Div(
                style={"display": "flex"},
                children=[
                    html.Div(
                        style={
                            "width": "30%",
                            "padding": "10px",
                        },
                        children=[
                            html.Div(id="analysis_tab_content_layout"),
                        ],
                    ),
                    html.Div(
                        style={
                            "width": "70%",
                            "padding": "10px",
                        },
                        children=[
                            html.Div(id="analysis_tab_plot_layout"),
                        ],
                    ),
                ],
            ),
            html.Br(),
            html.Br(),
            # download and view code
            html.Div(id="analysis_code"),
        ]
    )

    return layout


########################################
# Callbacks
########################################
@my_app.callback(
    [
        Output(
            component_id="analysis_tab_content_layout",
component_property="children"
        ),
        Output(component_id="analysis_tab_plot_layout",
component_property="children"),
        Output(component_id="analysis_code",
component_property="children"),
    ],
    [Input(component_id="analysis_selected_tab",
component_property="active_tab")],
)
def render_tab(tab_choice):
    """Renders the selected subtab's layout

    Args:
        tab_choice (str): selected subtab

    Returns:
        selected subtab's layout
    """
    if tab_choice == "analysis_line":
        return line_plot_info()
    if tab_choice == "analysis_bar_1":
        return bar_plot_1_info()
    if tab_choice == "analysis_bar_2":
        return bar_plot_2_info()
    if tab_choice == "analysis_count_1":
        return count_plot_1_info()
    if tab_choice == "analysis_count_2":
        return count_plot_2_info()
    if tab_choice == "analysis_count_3":
        return count_plot_3_info()
    if tab_choice == "analysis_count_4":
        return count_plot_4_info()
    if tab_choice == "analysis_count_5":
        return count_plot_5_info()
    if tab_choice == "analysis_count_6":
        return count_plot_6_info()
    if tab_choice == "analysis_pie":
        return pie_chart_info()
    if tab_choice == "analysis_dist":
        return dist_plot_info()
    if tab_choice == "analysis_pair":
        return pair_plot_info()
    if tab_choice == "analysis_heatmap":
        return heatmap_info()
    if tab_choice == "analysis_qq":
        return qq_plot_info()
    if tab_choice == "analysis_reg_1":
        return reg_plot_1_info()
    if tab_choice == "analysis_reg_2":
        return reg_plot_2_info()
    if tab_choice == "analysis_reg_3":
        return reg_plot_3_info()
    if tab_choice == "analysis_area":
        return area_plot_info()
    if tab_choice == "analysis_violin":
        return violin_plot_info()
    if tab_choice == "analysis_joint_1":
        return joint_plot_1_info()
    if tab_choice == "analysis_joint_2":
        return joint_plot_2_info()
    if tab_choice == "analysis_3d":
        return plot_3d_info()
    if tab_choice == "analysis_3d_contour":
        return plot_3d_contour_info()
```

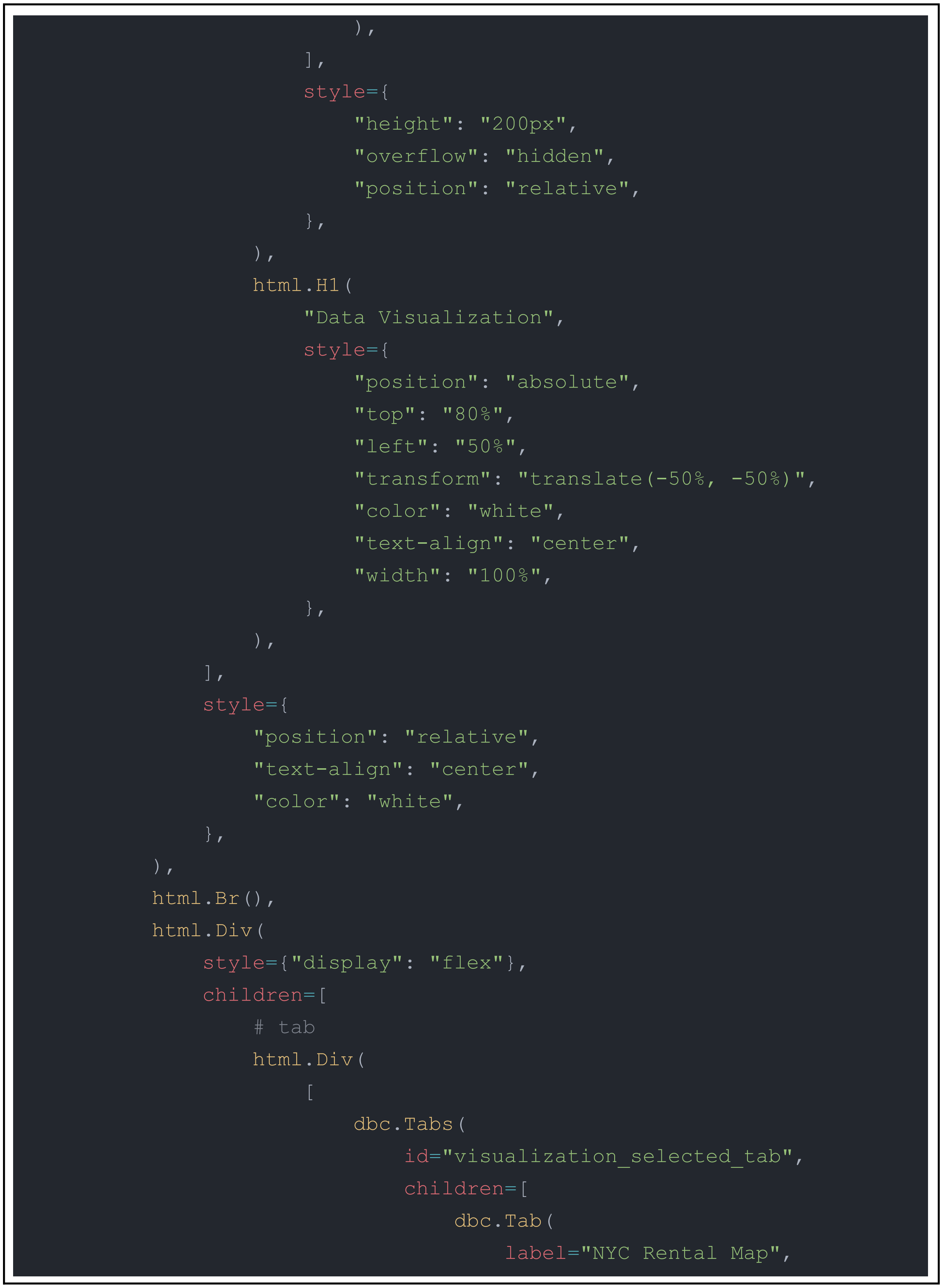
Data Visualization Page

```
# dash imports
import dash
from dash import html
from dash import Input
from dash import Output
from dash import dcc
import dash_bootstrap_components as dbc

# file imports
from maindash import my_app
from components.visualization.map import map_info
from components.visualization.price import price_info


#######################################
# Layout
#######################################
def visualization_layout():
    layout = html.Div(
        [
            html.Div(
                [
                    html.Div(
                        [
                            html.Img(

src="https://images.unsplash.com/photo-1557683316-973673baf926?q=80&w=11
29&auto=format&fit=crop&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdl
fHx8fGVufDB8fHx8fA%3D%3D",
                                style={
                                    "width": "100%",
                                    "height": "auto",
                                    "position": "relative",
                                },
```



```
                        ),
                    ],
                    style={
                        "height": "200px",
                        "overflow": "hidden",
                        "position": "relative",
                    },
                ),
                html.H1(
                    "Data Visualization",
                    style={
                        "position": "absolute",
                        "top": "80%",
                        "left": "50%",
                        "transform": "translate(-50%, -50%)",
                        "color": "white",
                        "text-align": "center",
                        "width": "100%",
                    },
                ),
            ],
            style={
                "position": "relative",
                "text-align": "center",
                "color": "white",
            },
        ),
        html.Br(),
        html.Div(
            style={"display": "flex"},
            children=[
                # tab
                html.Div(
                    [
                        dbc.Tabs(
                            id="visualization_selected_tab",
                            children=[
                                dbc.Tab(
                                    label="NYC Rental Map",
                                        tab_id="visualization_map",
                                    ),
                                    dbc.Tab(
                                        label="NYC Rental Price",
                                        tab_id="visualization_price",
                                    ),
                                ],
                                active_tab="visualization_map",
                            ),
                        ]
                    ),
                ],
            ),
            html.Br(),
            # content
            html.Div(
                style={"display": "flex"},
                children=[
                    html.Div(
                        style={
                            "width": "30%",
                            "padding": "10px",
                        },
                        children=[

html.Div(id="visualization_tab_content_layout"),
                        ],
                    ),
                    html.Div(
                        style={
                            "width": "70%",
                            "padding": "10px",
                        },
                        children=[
                            html.Div(id="visualization_tab_plot_layout"),
                        ],
                    ),
                ],
            ),
            html.Br(),
            html.Br(),
            # download and view code
            html.Div(id="visualization_code"),
        ]
    )

    return layout


#######################################
# Callbacks
#######################################
@my_app.callback(
    [
        Output(
            component_id="visualization_tab_content_layout",
            component_property="children",
        ),
        Output(
            component_id="visualization_tab_plot_layout",
component_property="children"
        ),
        Output(component_id="visualization_code",
component_property="children"),
    ],
    [Input(component_id="visualization_selected_tab",
component_property="active_tab")],
)
def render_tab(tab_choice):
    """Renders the selected subtab's layout

    Args:
        tab_choice (str): selected subtab

    Returns:
        selected subtab's layout
    """
    if tab_choice == "visualization_map":
        return map_info()
    if tab_choice == "visualization_price":
        return price_info()
```

Interest Level Prediction Page

```
# dash imports
import dash
from dash import html
from dash import Input
from dash import Output
from dash import dcc
import dash_bootstrap_components as dbc

# file imports
from maindash import my_app
from components.interest_level_prediction.ml import ml_info


#######################################
# Layout
#######################################
def interest_level_prediction_layout():
    layout = html.Div(
        [
            html.Div(
                [
                    html.Div(
                        [
                            html.Img(

src="https://images.unsplash.com/photo-1641326038434-01b0217c18f1?q=80&w
=1032&auto=format&fit=crop&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wY
WdlfHx8fGVufDB8fHx8fA%3D%3D",
                                style={
                                    "width": "100%",
                                    "height": "auto",
                                    "position": "relative",
                                },
                            ),
                        ],
                        style={
                            "height": "200px",
                            "overflow": "hidden",
                            "position": "relative",
                        },
                    ),
                    html.H1(
                        "Interest Level Prediction",
                        style={
                            "position": "absolute",
                            "top": "80%",
                            "left": "50%",
                            "transform": "translate(-50%, -50%)",
                            "color": "white",
                            "text-align": "center",
                            "width": "100%",
                        },
                    ),
                ],
                style={
                    "position": "relative",
                    "text-align": "center",
                    "color": "white",
                },
            ),
            html.Br(),
            html.Div(
                style={"display": "flex"},
                children=[
                    # tab
                    html.Div(
                        [
                            dbc.Tabs(

id="interest_level_prediction_selected_tab",
                                children=[
                                    dbc.Tab(
                                        label="Classical Models",

tab_id="interest_level_prediction_ml",
                                    ),
                                    dbc.Tab(
                                        label="Classical Models -
Images",

tab_id="interest_level_prediction_ml_img",
                                    ),
                                ],

active_tab="interest_level_prediction_ml",
                            ),
                        ]
                    ),
                ],
            ),
            html.Br(),
            # content
            html.Div(
                style={"display": "flex"},
                children=[
                    html.Div(
                        style={
                            "width": "50%",
                            "padding": "10px",
                        },
                        children=[

html.Div(id="interest_level_prediction_tab_content_layout"),
                        ],
                    ),
                    html.Div(
                        style={
                            "width": "50%",
                            "padding": "10px",
                        },
                            children=[

html.Div(id="interest_level_prediction_tab_plot_layout"),
                            ],
                       ),
                  ],
             ),
        ]
    )

    return layout


########################################
# Callbacks
########################################
@my_app.callback(
    [
        Output(
            component_id="interest_level_prediction_tab_content_layout",
            component_property="children",
        ),
        Output(
            component_id="interest_level_prediction_tab_plot_layout",
            component_property="children",
        ),
    ],
    [
        Input(
            component_id="interest_level_prediction_selected_tab",
            component_property="active_tab",
        )
    ],
)
def render_tab(tab_choice):
    """Renders the selected subtab's layout

    Args:
        tab_choice (str): selected subtab
```

```
   Returns:
       selected subtab's layout
   """
   if tab_choice == "interest_level_prediction_ml":
       return ml_info()
   if tab_choice == "interest_level_prediction_ml_img":
       note = (
           html.Div(
               [
                   html.Div([html.H3("🧪 Experimentation")]),
                   html.P(
                       [
                           "Due to the lack of data and poor performance
of the trained model, I have decided not to deploy this tab. The demo
can be seen on the right. General idea: The user will be able to attach
image(s), YOLOv5 will extract additional features of interests, which
will (hopefully) help the ML make a more accurate prediction. The code
of this prototype can be accessed ",
                           html.A(
                               "here",

href="https://github.com/mnguyen0226/rental_gpt_dash/tree/main/experimen
tation/price_prediction_with_images",
                               target="_blank",
                           ),
                           ".",
                       ]
                   ),
               ]
           ),
       )
       img = html.Div(
           html.Img(

src="https://raw.githubusercontent.com/mnguyen0226/rental_gpt_dash/main/
dash/assets/photos/experience_ilp_img.png",
               style={
                   "width": "100%",
                   "height": "auto",
               },
           ),
       )
       return note, img
```

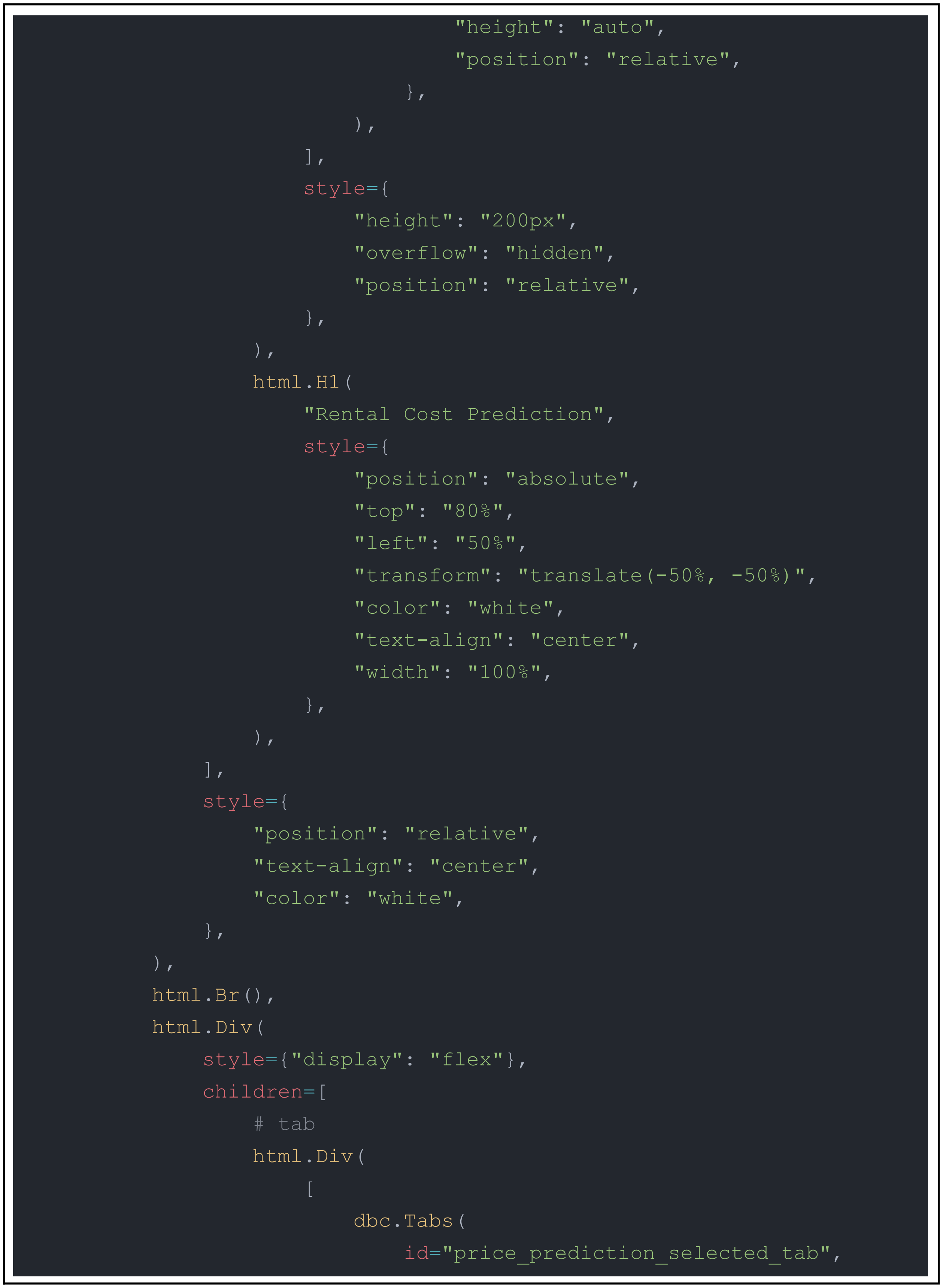
Rental Cost Prediction Page

```
# dash imports
import dash
from dash import html
from dash import Input
from dash import Output
from dash import dcc
import dash_bootstrap_components as dbc

# file imports
from maindash import my_app
from components.price_prediction.ml import ml_info


#######################################
# Layout
#######################################
def price_prediction_layout():
    layout = html.Div(
        [
            html.Div(
                [
                    html.Div(
                        [
                            html.Img(

src="https://images.unsplash.com/photo-1649393832219-0ad856a1e119?q=80&w
=1170&auto=format&fit=crop&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wY
WdlfHx8fGVufDB8fHx8fA%3D%3D",
                                style={
                                    "width": "100%",
```



```
                                "height": "auto",
                                "position": "relative",
                            },
                        ),
                    ],
                    style={
                        "height": "200px",
                        "overflow": "hidden",
                        "position": "relative",
                    },
                ),
                html.H1(
                    "Rental Cost Prediction",
                    style={
                        "position": "absolute",
                        "top": "80%",
                        "left": "50%",
                        "transform": "translate(-50%, -50%)",
                        "color": "white",
                        "text-align": "center",
                        "width": "100%",
                    },
                ),
            ],
            style={
                "position": "relative",
                "text-align": "center",
                "color": "white",
            },
        ),
        html.Br(),
        html.Div(
            style={"display": "flex"},
            children=[
                # tab
                html.Div(
                    [
                        dbc.Tabs(
                            id="price_prediction_selected_tab",
                                children=[
                                    dbc.Tab(
                                        label="Classical Models",
                                        tab_id="price_prediction_ml",
                                    ),
                                    dbc.Tab(
                                        label="Classical Models - 
Images",

                                        tab_id="price_prediction_ml_img",
                                    ),
                                ],
                                active_tab="price_prediction_ml",
                            ),
                        ]
                    ),
                ],
            ),
            html.Br(),
            # content
            html.Div(
                style={"display": "flex"},
                children=[
                    html.Div(
                        style={
                            "width": "50%",
                            "padding": "10px",
                        },
                        children=[

html.Div(id="price_prediction_tab_content_layout"),
                        ],
                    ),
                    html.Div(
                        style={
                            "width": "50%",
                            "padding": "10px",
                        },
                        children=[
html.Div(id="price_prediction_tab_plot_layout"),
                              ],
                         ),
                    ],
               ),
          ]
     )

     return layout


#######################################
# Callbacks
#######################################
@my_app.callback(
     [
          Output(
               component_id="price_prediction_tab_content_layout",
               component_property="children",
          ),
          Output(
               component_id="price_prediction_tab_plot_layout",
               component_property="children",
          ),
     ],
     [
          Input(
               component_id="price_prediction_selected_tab",
               component_property="active_tab",
          )
     ],
)
def render_tab(tab_choice):
     """Renders the selected subtab's layout

     Args:
          tab_choice (str): selected subtab

     Returns:
        selected subtab's layout
    """
    if tab_choice == "price_prediction_ml":
        return ml_info()
    if tab_choice == "price_prediction_ml_img":
        note = (
            html.Div(
                [
                    html.Div([html.H3("🧪 Experimentation")]),
                    html.P(
                        [
                            "Due to the lack of data and poor performance
of the trained model, I have decided not to deploy this tab. The demo
can be seen on the right. General idea: The user will be able to attach
image(s), YOLOv5 will extract additional features of interests, which
will (hopefully) help the ML make a more accurate prediction. The code
of the prototype can be accessed ",
                            html.A(
                                "here",

href="https://github.com/mnguyen0226/rental_gpt_dash/tree/main/experimen
tation/price_prediction_with_images",
                                target="_blank",
                            ),
                            ".",
                        ]
                    ),
                ]
            ),
        )
        img = html.Div(
            html.Img(

src="https://raw.githubusercontent.com/mnguyen0226/rental_gpt_dash/main/
dash/assets/photos/experience_ilp_img.png",
                style={
                    "width": "100%",
                    "height": "auto",
                },
            ),
        )
        return note, img
```

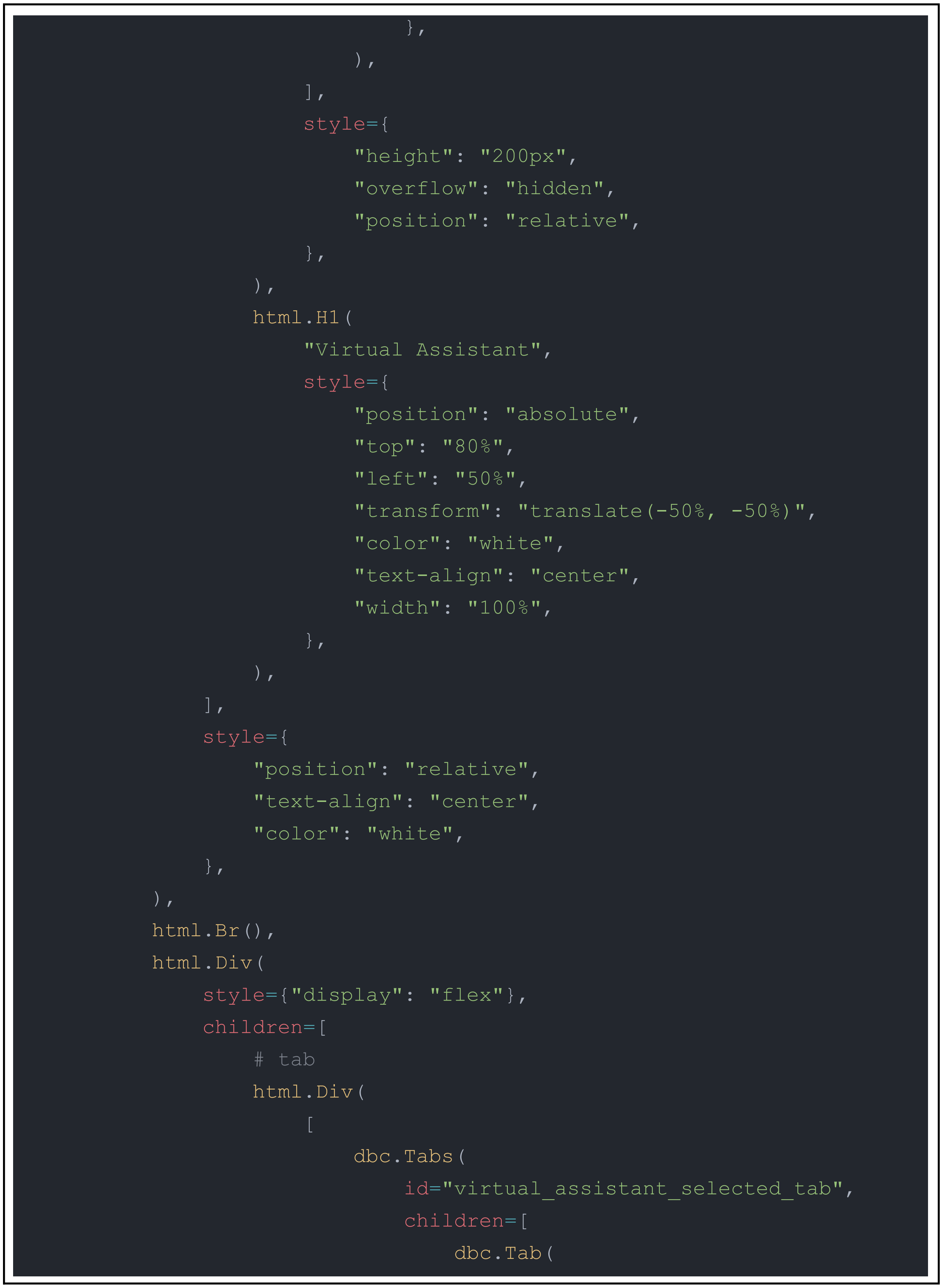
Virtual Assistant Page

```
# dash imports
import dash
from dash import html
from dash import Input
from dash import Output
from dash import dcc
import dash_bootstrap_components as dbc

# file imports
from maindash import my_app
from components.virtual_assistant.chatbot import chatbot_info


#######################################
# Layout
#######################################
def virtual_assistant_layout():
    layout = html.Div(
        [
            html.Div(
                [
                    html.Div(
                        [
                            html.Img(

src="https://images.unsplash.com/photo-1641160923894-b1a80920187d?q=80&w
=1632&auto=format&fit=crop&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wY
WdlfHx8fGVufDB8fHx8fA%3D%3D",
                                style={
                                    "width": "100%",
                                    "height": "auto",
                                    "position": "relative",
```



```
                    },
                ),
            ],
            style={
                "height": "200px",
                "overflow": "hidden",
                "position": "relative",
            },
        ),
        html.H1(
            "Virtual Assistant",
            style={
                "position": "absolute",
                "top": "80%",
                "left": "50%",
                "transform": "translate(-50%, -50%)",
                "color": "white",
                "text-align": "center",
                "width": "100%",
            },
        ),
    ],
    style={
        "position": "relative",
        "text-align": "center",
        "color": "white",
    },
),
html.Br(),
html.Div(
    style={"display": "flex"},
    children=[
        # tab
        html.Div(
            [
                dbc.Tabs(
                    id="virtual_assistant_selected_tab",
                    children=[
                        dbc.Tab(
                                        label="LLaMA Chatbot",
                                        tab_id="hugging_face",
                                    ),
                                    dbc.Tab(
                                        label="Alternative Solution",
                                        tab_id="streamlit",
                                    ),
                                ],
                                active_tab="hugging_face",
                            ),
                        ]
                    ),
                ],
            ),
            html.Br(),
            # content
            html.Div(
                style={"display": "flex"},
                children=[
                    html.Div(
                        style={
                            "width": "30%",
                            "padding": "10px",
                        },
                        children=[

html.Div(id="virtual_assistant_tab_content_layout"),
                        ],
                    ),
                    html.Div(
                        style={
                            "width": "70%",
                            # "height": "calc(70vh - 250px)",
                            # "overflow-y": "auto",
                            "padding": "10px",
                        },
                        children=[

html.Div(id="virtual_assistant_tab_plot_layout"),
                        ],
                    ),
                ],
            ),
        ]
    )

    return layout


########################################
# Callbacks
########################################
@my_app.callback(
    [
        Output(
            component_id="virtual_assistant_tab_content_layout",
            component_property="children",
        ),
        Output(
            component_id="virtual_assistant_tab_plot_layout",
            component_property="children",
        ),
    ],
    [
        Input(
            component_id="virtual_assistant_selected_tab",
            component_property="active_tab",
        )
    ],
)
def render_tab_1(tab_choice):
    """Renders the selected subtab's layout

    Args:
        tab_choice (str): selected subtab

    Returns:
        selected subtab's layout
    """
    if tab_choice == "hugging_face":
        return chatbot_info()
    if tab_choice == "streamlit":
        note = html.Div(
            [
                html.H3("🧪 Experimentation"),
                html.P(
                    [
                        "The template for HuggingFace chatbot developed
via Streamlit & HugChat can be found ",
                        html.A(
                            "here",

href="https://github.com/mnguyen0226/rental_gpt_dash/tree/main/experimen
tation/streamlit_llama_chatbot",
                            target="_blank",
                        ),
                        ".",
                    ]
                ),
                html.P(
                    [
                        "The template for HuggingFace chatbot developed
via Dash & HugChat can be found ",
                        html.A(
                            "here",

href="https://github.com/mnguyen0226/rental_gpt_dash/tree/main/experimen
tation/dash_llama_chatbot",
                            target="_blank",
                        ),
                        ".",
                    ]
                ),
            ]
        )

        img = html.Div(
            [
                html.Img(

src="https://raw.githubusercontent.com/mnguyen0226/rental_gpt_dash/main/
dash/assets/photos/experience_streamlit_chatbot.png",
                    style={
                        "width": "63%",
                        "height": "auto",
                    },
                ),
                html.Img(

src="https://raw.githubusercontent.com/mnguyen0226/rental_gpt_dash/main/
dash/assets/photos/experience_dash_chatbot.png",
                    style={
                        "width": "37%",
                        "height": "auto",
                    },
                ),
            ]
        )

        return note, img
```

About Page

```
# dash imports
import dash
from dash import html
from dash import Input
from dash import Output
from dash import dcc
import dash_bootstrap_components as dbc

# file imports
from maindash import my_app
from utils.file_operation import read_file_as_str
```

```
########################################
# Layout
########################################
def about_layout():
    layout = html.Div(
        [
            html.Div(
                [
                    html.Div(
                        [
                            html.Img(

src="https://images.unsplash.com/photo-1614854262340-ab1ca7d079c7?q=80&w
=1470&auto=format&fit=crop&ixlib=rb-4.0.3&ixid=M3wxMjA3fDB8MHxwaG90by1wY
WdlfHx8fGVufDB8fHx8fA%3D%3D",
                                style={
                                    "width": "100%",
                                    "height": "auto",
                                    "position": "relative",
                                },
                            ),
                        ],
                        style={
                            "height": "200px",
                            "overflow": "hidden",
                            "position": "relative",
                        },
                    ),
                    html.H1(
                        "About",
                        style={
                            "position": "absolute",
                            "top": "80%",
                            "left": "50%",
                            "transform": "translate(-50%, -50%)",
                            "color": "white",
                            "text-align": "center",
                            "width": "100%",
                        },
                    ),
                ],
                style={
                    "position": "relative",
                    "text-align": "center",
                    "color": "white",
                },
            ),
            html.Br(),
            html.Div(
                style={"display": "flex"},
                children=[
                    dcc.Markdown(

 children=read_file_as_str("./utils/markdown/about/about.md"),
                        mathjax=True,
                    ),
                ],
            ),
            card(),
            html.Br(),
            html.Hr(),
            html.H3(
                "Data Science Life Cycle",
                style={"textAlign": "center", "color": "#082446"},
            ),
            html.Br(),
            html.Img(

 src="https://raw.githubusercontent.com/mnguyen0226/rental_gpt_dash/main/
 dash/assets/photos/data_science_life_cycle.png",
                style={
                    "width": "1200px",
                    "display": "block",
                    "margin-left": "auto",
                    "margin-right": "auto",
                },
            ),
            html.Br(),
            html.Hr(),
            html.H3(
                "Architecture Design",
                style={"textAlign": "center", "color": "#082446"},
            ),
            html.Br(),
            html.Img(

src="https://raw.githubusercontent.com/mnguyen0226/rental_gpt_dash/main/
dash/assets/photos/rental_gpt_dash_architecture.png",
                style={
                    "width": "800px",
                    "display": "block",
                    "margin-left": "auto",
                    "margin-right": "auto",
                },
            ),
        ]
    )

    return layout


def card():
    layout = html.Div(
        [
            html.Br(),
            dbc.Card(
                [
                    dbc.Row(
                        [
                            dbc.Col(
                                dbc.CardImg(

src="https://raw.githubusercontent.com/mnguyen0226/mnguyen0226.github.io
/main/static/profile_images/personal/professional.png",
                                    className="img-fluid rounded-start",
                                ),
                                className="col-md-5",
                            ),
                            dbc.Col(
                                dbc.CardBody(
                                    children=[
                                        html.H4(
                                            "Minh T. Nguyen",
                                            className="card-title",
                                        ),
                                        html.P(
                                            "Research Assistant, "
                                            "M.Sc. Computer Engineering "
                                            "at Virginia Tech.",
                                            className="card-text",
                                        ),
                                        html.Small(
                                            "mnguyen0226@vt.edu",
                                            className="card-text
text-muted",
                                        ),
                                        html.Br(),
                                        html.Br(),
                                        html.A(
                                            html.Img(

src="https://cdn-icons-png.flaticon.com/512/174/174857.png",
                                                alt="LinkedIn",
                                                style={
                                                    "width": "30px",
                                                    "height": "30px",
                                                    "marginRight":
"10px",
                                                },
                                            ),

href="https://www.linkedin.com/in/minhbtnguyen/",
                                            target="_blank",
                                            className="text-dark",
                                        ),
                                        html.A(
                                            html.Img(

src="https://cdn-icons-png.flaticon.com/512/25/25231.png",
                                                alt="LinkedIn",
                                                style={
                                                    "width": "30px",
                                                    "height": "30px",
                                                },
                                            ),

href="https://github.com/mnguyen0226",
                                            target="_blank",
                                            className="text-dark",
                                        ),
                                    ]
                                ),
                                className="col-md-7",
                            ),
                        ],
                        className="g-0 d-flex align-items-center",
                    )
                ],
                className="mb-3",
                style={"maxWidth": "540px"},
            ),
        ]
    )
    return layout
```