zameen.com
Pakistan's No.1 Property Website

ZAMEEN.COM HOUSE PRICE ANALYSIS

TASK ASSIGNED BY
INTERNNCRAFT

Prepared by
ASADULLAH KHAN

INTERNNCRAFT
NAVGATING SUCCESS
THROGH EXPERIENCE

Contents

Introduction

The real estate market is a significant component of the global economy, influencing a myriad of industries and impacting the financial decisions of millions of individuals. In this context, policymakers. The House Price Analysis and Prediction project aims to delve into the complexities of house pricing by analyzing a comprehensive dataset. This project is designed to not only provide insights into the current state of the housing market but also to develop predictive models that can forecast future prices, thereby aiding in informed decision-making.

This project involves several key steps, starting with data cleaning and exploration. Cleaning the data ensures that it is free from errors, inconsistencies, and missing values, which is a critical prerequisite for any meaningful analysis. Exploring the data helps in understanding the distribution of house prices and the relationships between various features, such as location, property type, and size. Visualizations like scatter plots and box plots are employed to identify trends and patterns that may not be immediately apparent.

Feature engineering is another crucial component of this project. It involves creating new features from existing data that might be more relevant for predicting house prices. For instance, the age of the house or the number of bedrooms per floor could provide additional insights. Additionally, categorical features such as location need to be encoded into numerical values that can be effectively used by machine learning models.

Outlier analysis helps in identifying properties with prices significantly different from similar ones. Understanding these outliers can reveal underlying factors that drive house prices in unexpected ways. This can include specific property features or market conditions that are not immediately obvious.

Predictive modeling forms the core of this project, where various machine learning algorithms are trained to forecast house prices based on the cleaned and engineered dataset. Models such as linear regression, random forest, and gradient boosting are evaluated to determine the most accurate and reliable approach. The performance of these models is assessed using metrics like mean squared error and R-squared.

The final step involves using the trained model to predict future house prices under hypothetical scenarios. This aspect of the project demonstrates the practical application of the model and its potential to aid stakeholders in making informed decisions based on predicted market trends.

In conclusion, the House Price Analysis and Prediction project is a comprehensive study that encompasses data cleaning, exploration, feature engineering, outlier analysis, and predictive modeling. The insights gained from this project not only enhance our understanding of the factors influencing house prices but also provide a robust tool for forecasting future prices, thereby contributing to more informed and strategic decision-making in the real estate market.

Objective

The primary objective of the House Price Analysis and Prediction project is to provide a comprehensive examination of the factors that influence house prices, identify outliers, and develop robust predictive models to forecast future house prices. This multifaceted goal is achieved through a series of structured tasks, each contributing to the overall understanding and predictive capability regarding house prices in the dataset.

Detailed Objectives:

1. Data Cleaning and Exploration:

Identify and Handle Missing Values: Missing data can skew analysis and lead to inaccurate models. This project aims to identify any missing values in the dataset and address them appropriately, either by imputation or removal.

Resolve Inconsistencies: Inconsistent data entries, such as differing formats or units for similar variables, will be standardized to ensure uniformity.

Detect and Handle Outliers: Outliers can significantly impact the analysis and model performance. Identifying these anomalies and deciding on the appropriate handling method (e.g., transformation or removal) is a key objective.

2. Feature Engineering:

Create Relevant Features: Beyond the raw data provided, this project aims to derive new features that could enhance the predictive power of the model. For example, calculating the age of the house from the date added or deriving the number of bedrooms per floor.

Encode Categorical Features: Many features in the dataset, such as location, are categorical. Transforming these into numerical values through techniques like one-hot encoding will allow the machine learning algorithms to process them effectively.

3. Outlier Analysis:

Identify Properties with Extreme Prices: By comparing similar properties, the project aims to identify those with significantly higher or lower prices.

Investigate Contributing Factors: Understanding why certain properties are outliers can provide valuable insights. This involves analyzing the features or combinations of features that contribute to these price deviations.

4. Predictive Modeling:

Train Machine Learning Models: The project will train various machine learning models, such as linear regression, random forest, and gradient boosting, to predict house prices based on the available features.

Evaluate Model Performance: Using metrics like mean squared error (MSE) and R-squared, the performance of these models will be rigorously evaluated to identify the most accurate and reliable one.

5. Future Price Prediction:

Scenario-Based Predictions: The trained model will be used to predict future house prices under various hypothetical scenarios. This could involve predicting the price of a house with specific characteristics in a particular location, providing practical insights for potential buyers, sellers, and investors.

6. Report and Recommendations:

Summarize Findings: A detailed report will be prepared, summarizing the data exploration results, feature engineering techniques, outlier analysis, and model evaluation results.

Provide Recommendations: Based on the findings, the project will offer recommendations for further analysis, potential improvements in data collection, and practical advice for stakeholders in the real estate market.

Significance of the Objectives:

By achieving these objectives, the project will not only enhance the understanding of the current housing market dynamics but also provide a predictive framework that can be used for future price forecasting. This is crucial for various stakeholders, including real estate professionals, investors, and policymakers, as it enables data-driven decision-making and strategic planning. The comprehensive approach ensures that the analysis is thorough, the models are robust, and the insights are actionable, ultimately contributing to a more efficient and transparent real estate market.

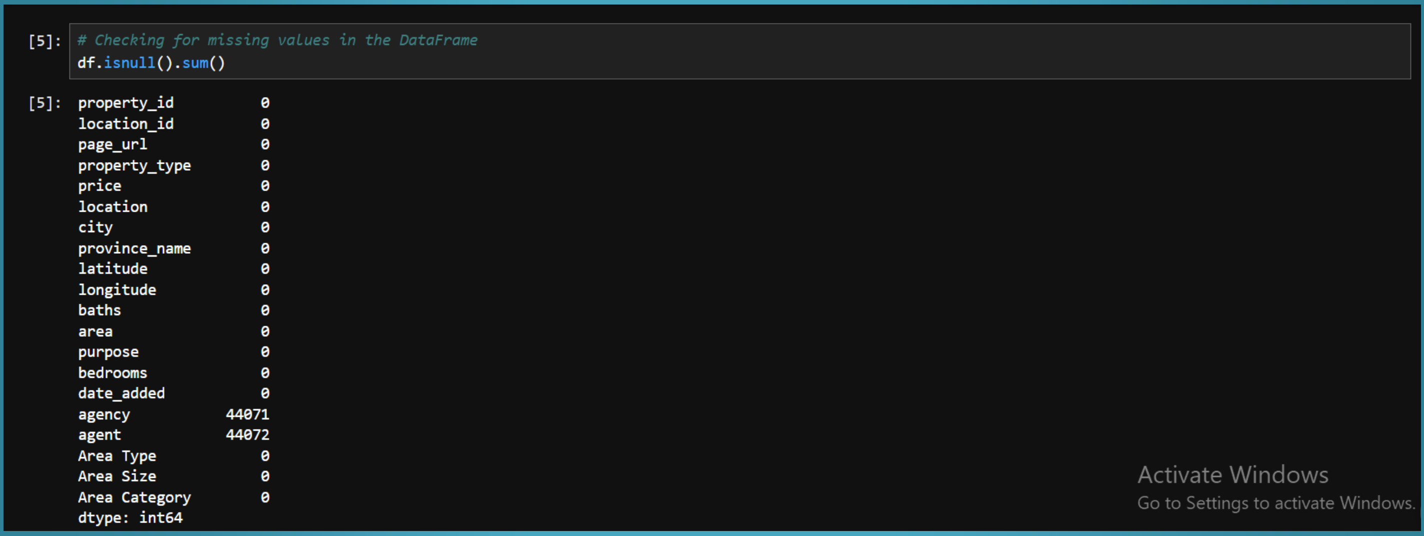
Tasks

Cleaning the data

Identifying and handling missing values, inconsistencies, and outliers.

1. **Checking for Missing Values:**

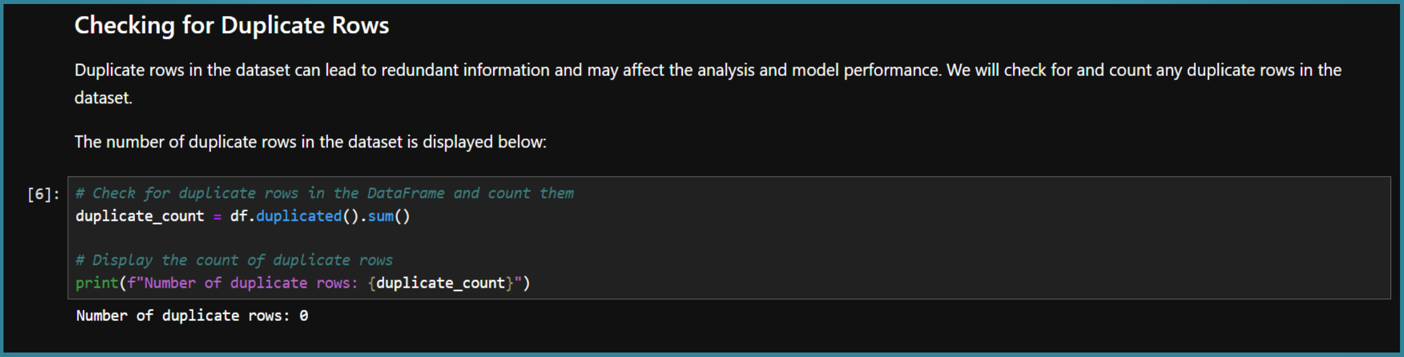
Checking for missing values in a DataFrame. The output presents a table with various column names, including 'property_id', 'location_id', 'page_url', 'price', 'property_type', 'city', 'province_name', 'latitude', 'longitude', 'baths', 'purpose', 'bedrooms', 'date_added', 'agency', 'agent' and several other columns. Indicating the count of missing values in that column within the DataFrame. There are some null values in 'agency' and 'agent' columns, I am not dropping it now, because later on I will drop these columns.



```
[5]: # Checking for missing values in the DataFrame
     df.isnull().sum()

[5]: property_id          0
     location_id          0
     page_url             0
     property_type        0
     price                0
     location             0
     city                 0
     province_name        0
     latitude             0
     longitude            0
     baths                0
     area                 0
     purpose              0
     bedrooms             0
     date_added           0
     agency           44071
     agent            44072
     Area Type            0
     Area Size            0
     Area Category        0
     dtype: int64
```

2. **Checking for Duplicate Rows:**

This cell contains code to find and count duplicate rows in a DataFrame named 'df'. The output below this cell displays "Number of duplicate rows: 0", indicating there are no duplicate entries in the dataset.



Checking for Duplicate Rows

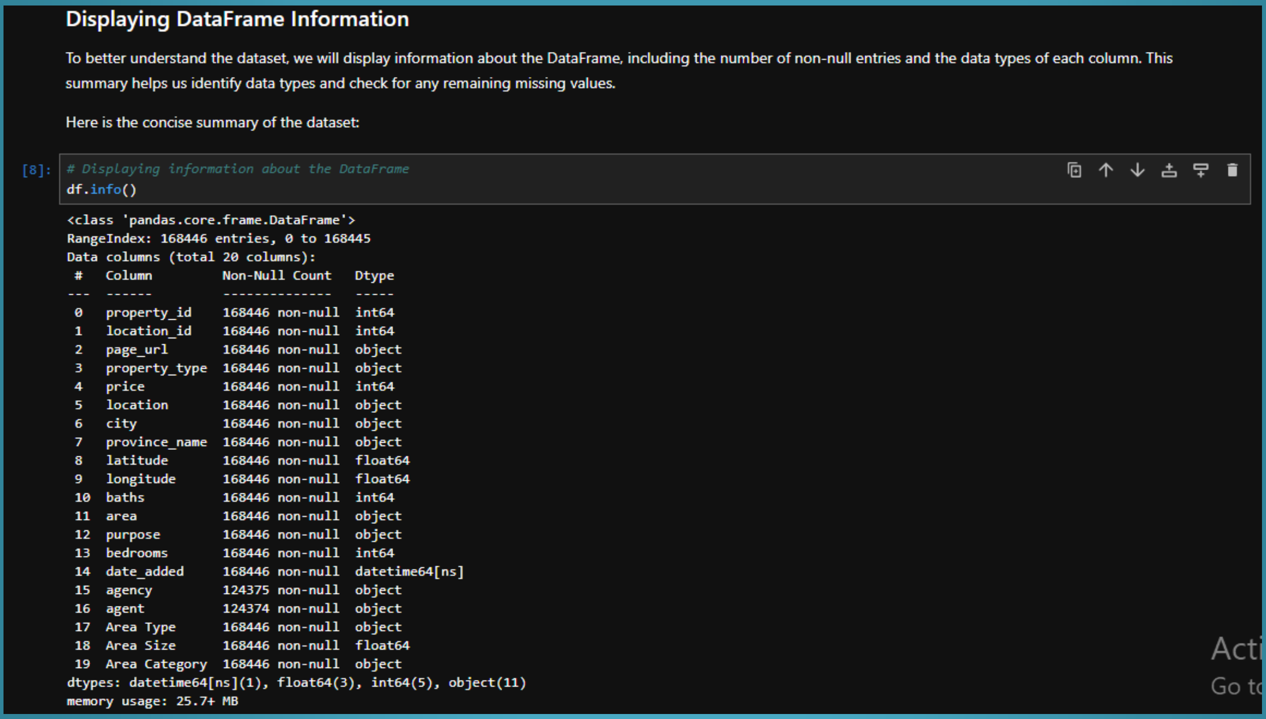
Duplicate rows in the dataset can lead to redundant information and may affect the analysis and model performance. We will check for and count any duplicate rows in the dataset.

The number of duplicate rows in the dataset is displayed below:

```
[6]: # Check for duplicate rows in the DataFrame and count them
     duplicate_count = df.duplicated().sum()

     # Display the count of duplicate rows
     print(f"Number of duplicate rows: {duplicate_count}")

     Number of duplicate rows: 0
```

3. Checking for Inconsistence Values:

The command executed is to display information about a DataFrame. The output lists the columns of the DataFrame along with their non-null count and data type. It demonstrates how to obtain an overview of the structure and content of a dataset to check inconsistence values.



Displaying DataFrame Information

To better understand the dataset, we will display information about the DataFrame, including the number of non-null entries and the data types of each column. This summary helps us identify data types and check for any remaining missing values.

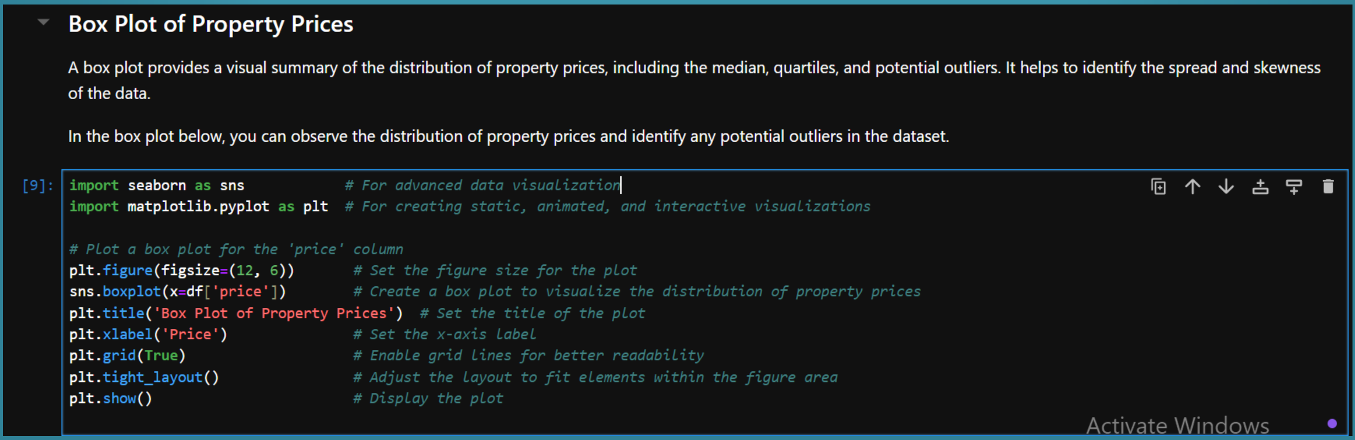
Here is the concise summary of the dataset:

```
[8]: # Displaying information about the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 168446 entries, 0 to 168445
Data columns (total 20 columns):
 #   Column         Non-Null Count   Dtype
---  ------         --------------   -----
 0   property_id    168446 non-null  int64
 1   location_id    168446 non-null  int64
 2   page_url       168446 non-null  object
 3   property_type  168446 non-null  object
 4   price          168446 non-null  int64
 5   location       168446 non-null  object
 6   city           168446 non-null  object
 7   province_name  168446 non-null  object
 8   latitude       168446 non-null  float64
 9   longitude      168446 non-null  float64
 10  baths          168446 non-null  int64
 11  area           168446 non-null  object
 12  purpose        168446 non-null  object
 13  bedrooms       168446 non-null  int64
 14  date_added     168446 non-null  datetime64[ns]
 15  agency         124375 non-null  object
 16  agent          124374 non-null  object
 17  Area Type      168446 non-null  object
 18  Area Size      168446 non-null  float64
 19  Area Category  168446 non-null  object
dtypes: datetime64[ns](1), float64(3), int64(5), object(11)
memory usage: 25.7+ MB
```

4. Checking for Outliers:

Understanding price distributions and identifying outliers using Box plot. The code includes import statements for the seaborn and matplotlib.pyplot libraries. sets the figure size, creates the plot using seaborn's `boxplot` function with 'price' as the data parameter, sets titles and labels for the plot, adjusts the layout, and displays the plot.

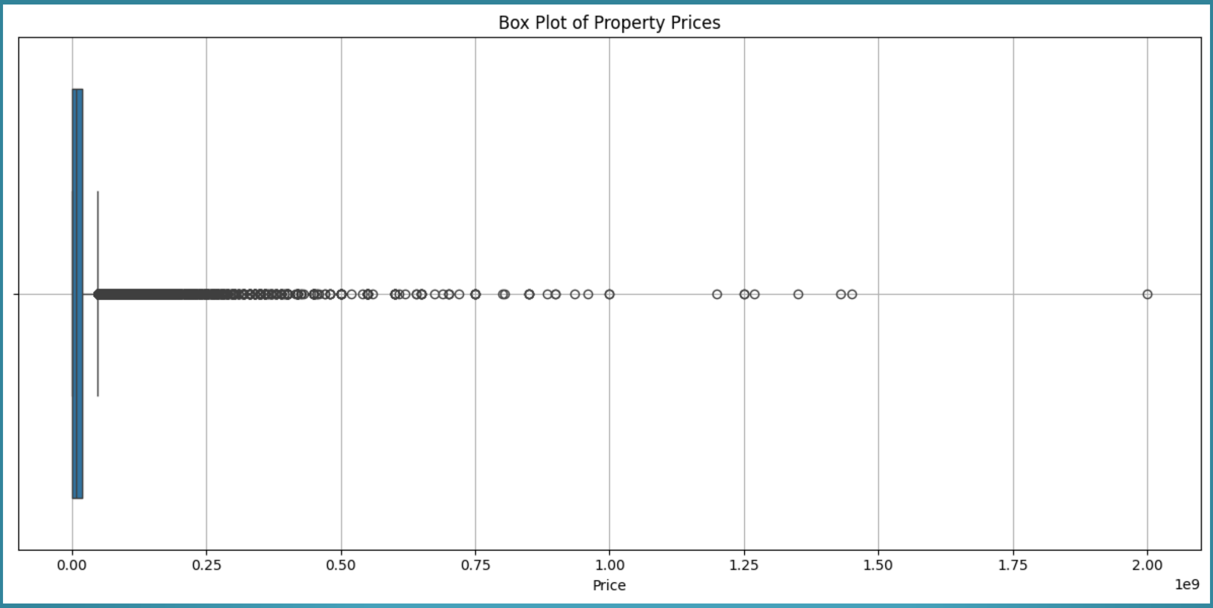


Box Plot of Property Prices

A box plot provides a visual summary of the distribution of property prices, including the median, quartiles, and potential outliers. It helps to identify the spread and skewness of the data.

In the box plot below, you can observe the distribution of property prices and identify any potential outliers in the dataset.

```
[9]: import seaborn as sns            # For advanced data visualization
import matplotlib.pyplot as plt  # For creating static, animated, and interactive visualizations

# Plot a box plot for the 'price' column
plt.figure(figsize=(12, 6))       # Set the figure size for the plot
sns.boxplot(x=df['price'])        # Create a box plot to visualize the distribution of property prices
plt.title('Box Plot of Property Prices')  # Set the title of the plot
plt.xlabel('Price')               # Set the x-axis label
plt.grid(True)                    # Enable grid lines for better readability
plt.tight_layout()                # Adjust the layout to fit elements within the figure area
plt.show()                        # Display the plot
```

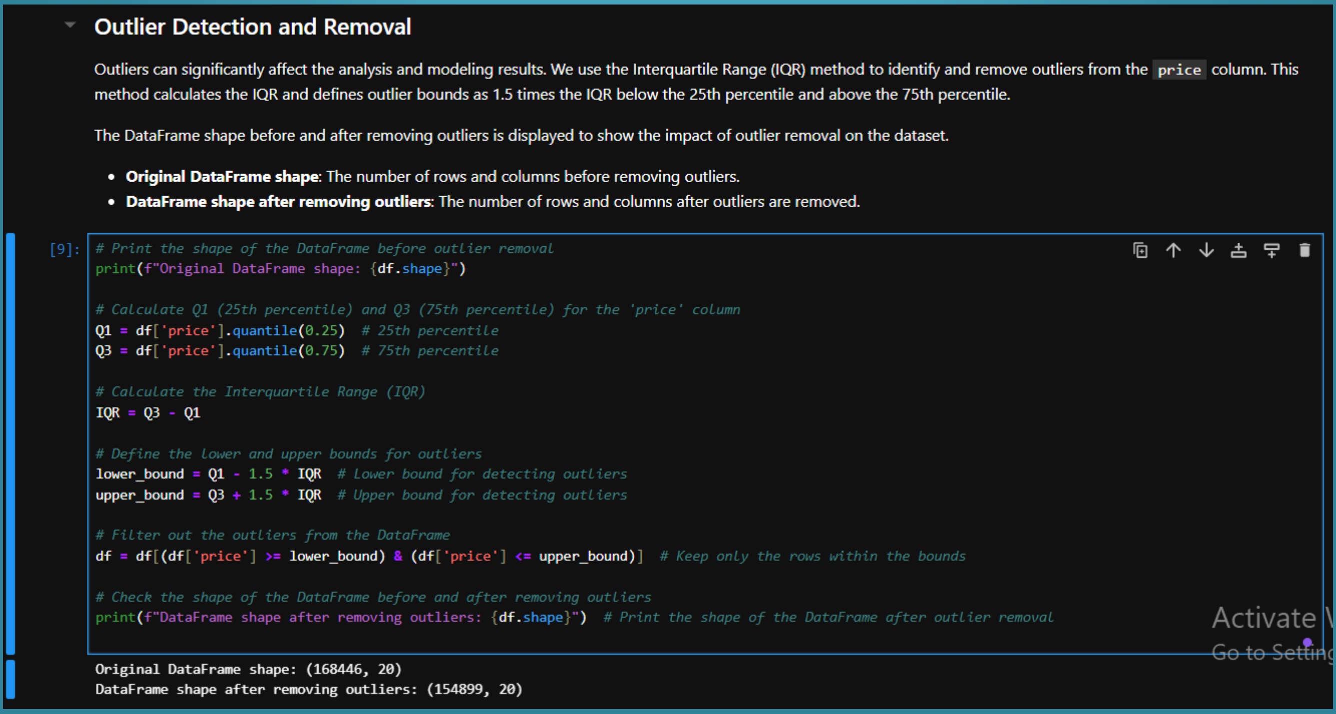


Observation:

- The property price distribution is highly skewed to the right, with a long tail indicating a small number of very expensive properties.
- The median property price is relatively low compared to the upper quartile, indicating a larger concentration of lower-priced properties.
- Numerous outliers indicate a significant presence of properties with exceptionally high prices.

Removing Outliers:

The image showcases a Python code snippet demonstrating the process of outlier detection and removal within a DataFrame, specifically targeting the 'price' column using the Interquartile Range (IQR) method. The code includes steps to import necessary libraries, calculate quartiles, compute the IQR, define outlier bounds, filter out outliers, and print the DataFrame shape before and after outlier removal. This method helps improve data quality by eliminating extreme values that could skew analysis or modeling results.



Outlier Detection and Removal

Outliers can significantly affect the analysis and modeling results. We use the Interquartile Range (IQR) method to identify and remove outliers from the `price` column. This method calculates the IQR and defines outlier bounds as 1.5 times the IQR below the 25th percentile and above the 75th percentile.

The DataFrame shape before and after removing outliers is displayed to show the impact of outlier removal on the dataset.

- **Original DataFrame shape**: The number of rows and columns before removing outliers.
- **DataFrame shape after removing outliers**: The number of rows and columns after outliers are removed.

[9]:

```python
# Print the shape of the DataFrame before outlier removal
print(f"Original DataFrame shape: {df.shape}")

# Calculate Q1 (25th percentile) and Q3 (75th percentile) for the 'price' column
Q1 = df['price'].quantile(0.25)  # 25th percentile
Q3 = df['price'].quantile(0.75)  # 75th percentile

# Calculate the Interquartile Range (IQR)
IQR = Q3 - Q1

# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR  # Lower bound for detecting outliers
upper_bound = Q3 + 1.5 * IQR  # Upper bound for detecting outliers

# Filter out the outliers from the DataFrame
df = df[(df['price'] >= lower_bound) & (df['price'] <= upper_bound)]  # Keep only the rows within the bounds

# Check the shape of the DataFrame before and after removing outliers
print(f"DataFrame shape after removing outliers: {df.shape}")  # Print the shape of the DataFrame after outlier removal
```

```
Original DataFrame shape: (168446, 20)
DataFrame shape after removing outliers: (154899, 20)
```

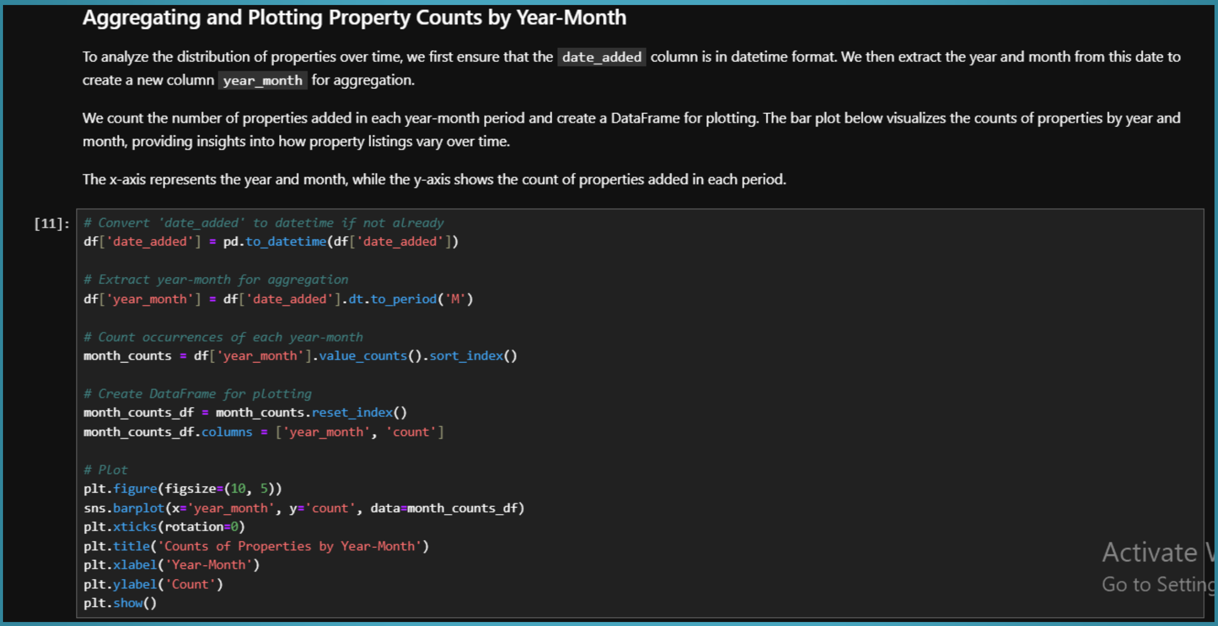
Exploring the data

Analyze the distribution of house prices and other features. Identify potential relationships between features and price using visualizations

Analyzing and visualizing property counts over time.

The code generates a bar plot showing the number of properties added per month over a period of time. The x-axis represents the year and month, and the y-axis represents the count of properties.

This code provides a step-by-step approach to analyzing property data over time and visualizing the results in a clear and informative bar plot.



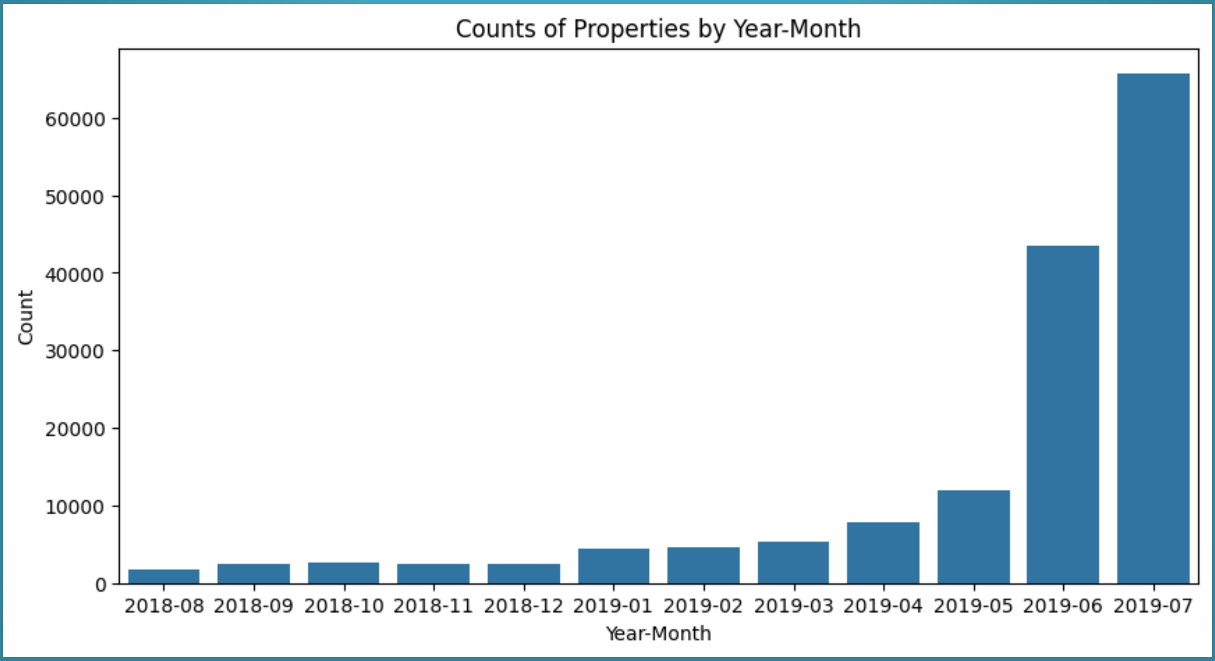
Aggregating and Plotting Property Counts by Year-Month

To analyze the distribution of properties over time, we first ensure that the `date_added` column is in datetime format. We then extract the year and month from this date to create a new column `year_month` for aggregation.

We count the number of properties added in each year-month period and create a DataFrame for plotting. The bar plot below visualizes the counts of properties by year and month, providing insights into how property listings vary over time.

The x-axis represents the year and month, while the y-axis shows the count of properties added in each period.

```
[11]: # Convert 'date_added' to datetime if not already
df['date_added'] = pd.to_datetime(df['date_added'])

# Extract year-month for aggregation
df['year_month'] = df['date_added'].dt.to_period('M')

# Count occurrences of each year-month
month_counts = df['year_month'].value_counts().sort_index()

# Create DataFrame for plotting
month_counts_df = month_counts.reset_index()
month_counts_df.columns = ['year_month', 'count']

# Plot
plt.figure(figsize=(10, 5))
sns.barplot(x='year_month', y='count', data=month_counts_df)
plt.xticks(rotation=0)
plt.title('Counts of Properties by Year-Month')
plt.xlabel('Year-Month')
plt.ylabel('Count')
plt.show()
```



<u>Observation:</u>

The image presents a bar chart illustrating the count of properties over a specific time period, segmented by year and month.

- **Bars:** Rectangular bars, each representing the count of properties for a specific year-month combination. The height of each bar corresponds to the number of properties.
- **Trend:** The chart shows an overall increasing trend in property counts from August 2018 to July 2019, with a notable peak in July 2019.

Observations:

- The highest number of properties was added in July 2019.
- The lowest number of properties was added in August 2018.
- There is a consistent month-over-month increase in property counts, with some fluctuations.

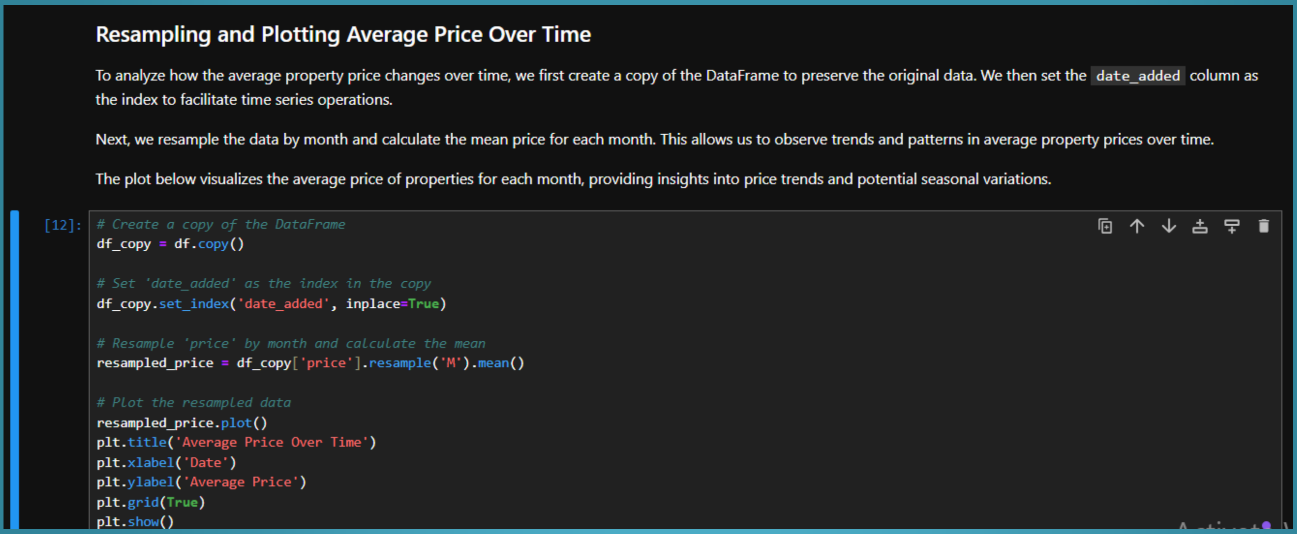
Purpose:

This chart effectively visualizes the trend in property additions over time, allowing for easy identification of peak periods and overall growth patterns.

Resampling and Plotting Average Price Over Time

The code generates a line plot illustrating the average property price for each month, providing insights into price trends and potential seasonal variations.

This code effectively analyzes and visualizes how average property prices change over time, aiding in identifying trends and patterns in the data.



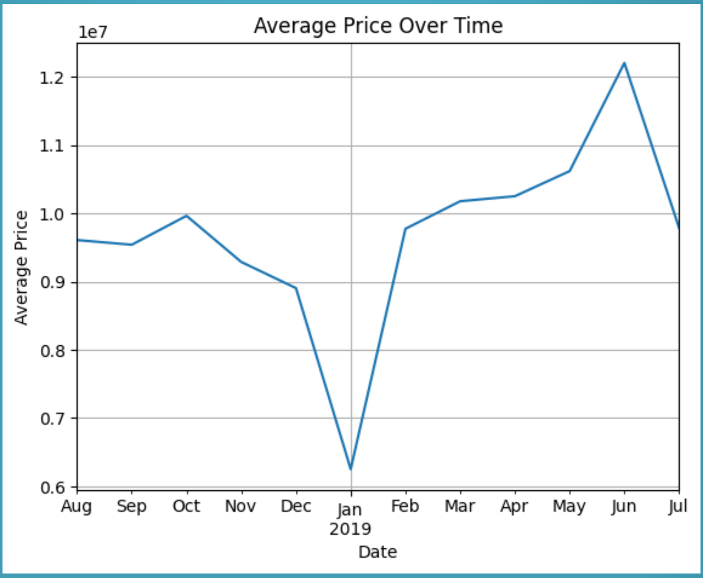
Resampling and Plotting Average Price Over Time

To analyze how the average property price changes over time, we first create a copy of the DataFrame to preserve the original data. We then set the `date_added` column as the index to facilitate time series operations.

Next, we resample the data by month and calculate the mean price for each month. This allows us to observe trends and patterns in average property prices over time.

The plot below visualizes the average price of properties for each month, providing insights into price trends and potential seasonal variations.

```
[12]: # Create a copy of the DataFrame
df_copy = df.copy()

# Set 'date_added' as the index in the copy
df_copy.set_index('date_added', inplace=True)

# Resample 'price' by month and calculate the mean
resampled_price = df_copy['price'].resample('M').mean()

# Plot the resampled data
resampled_price.plot()
plt.title('Average Price Over Time')
plt.xlabel('Date')
plt.ylabel('Average Price')
plt.grid(True)
plt.show()
```



Observation:

The image presents a line chart illustrating the average price over a specific time period, segmented by month.

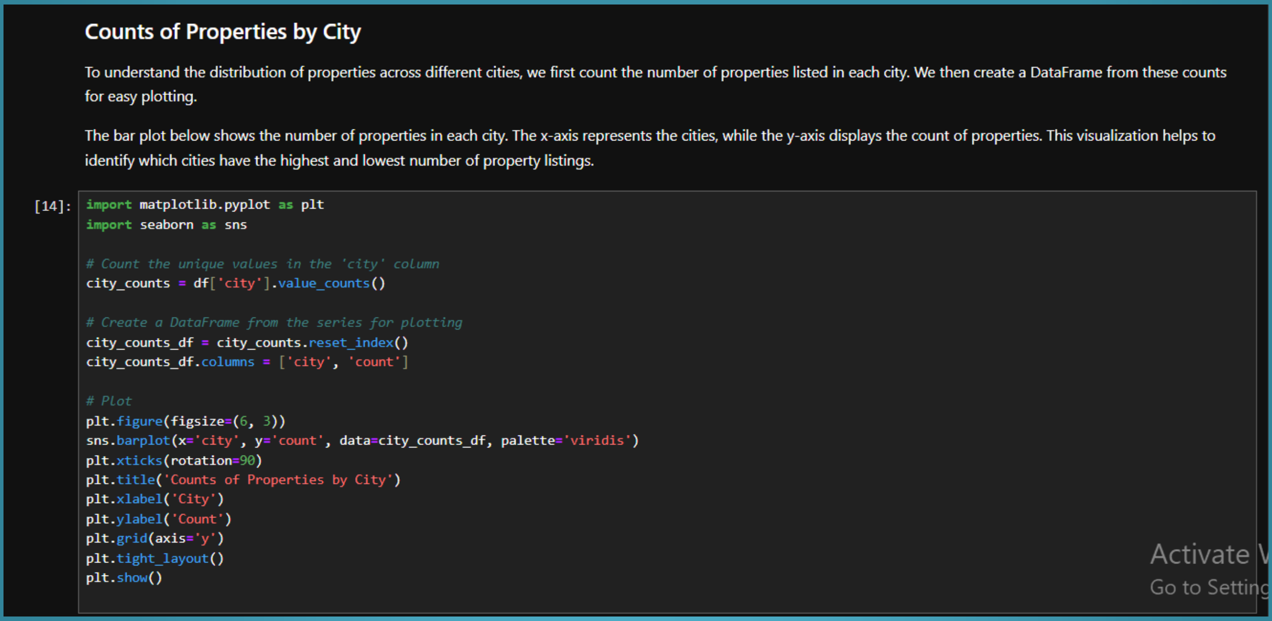
The highest average price was recorded in June 2019.

- There is a general upward trend in average price from August 2018 to June 2019, followed by a sharp decline in July 2019.
- The chart highlights seasonal variations in average price throughout the year.

This chart effectively visualizes the trend in average price over time, allowing for easy identification of peak periods, price fluctuations, and overall patterns. It indicates that prices tend to be higher in certain months and lower in others, reflecting potential seasonal effects on property prices.

Counting and Visualizing Property Counts by City

The code includes importing the `matplotlib.pyplot` and `seaborn` libraries for data visualization. It then counts the occurrences of each unique value in the 'city' column of a DataFrame, stores the result in a `city_counts` Series, and converts this Series into a DataFrame named `city_counts_df` with 'city' and 'count' columns. The code creates a bar plot using Seaborn to visualize the count of properties for each city, customizing the plot with titles, labels, and rotation of x-axis labels for better readability.



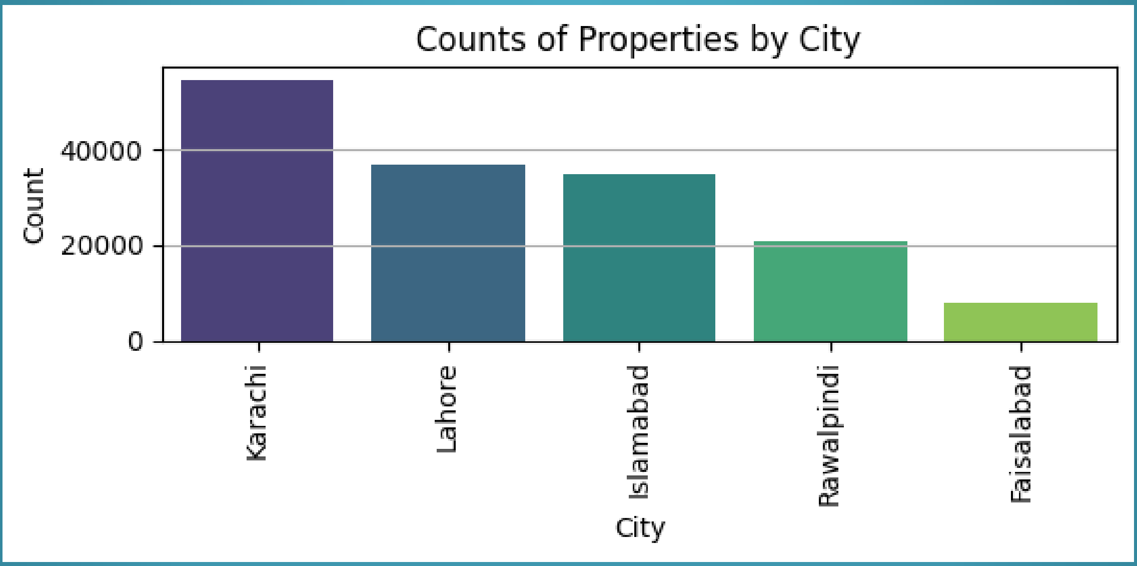
Counts of Properties by City

To understand the distribution of properties across different cities, we first count the number of properties listed in each city. We then create a DataFrame from these counts for easy plotting.

The bar plot below shows the number of properties in each city. The x-axis represents the cities, while the y-axis displays the count of properties. This visualization helps to identify which cities have the highest and lowest number of property listings.

[14]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Count the unique values in the 'city' column
city_counts = df['city'].value_counts()

# Create a DataFrame from the series for plotting
city_counts_df = city_counts.reset_index()
city_counts_df.columns = ['city', 'count']

# Plot
plt.figure(figsize=(6, 3))
sns.barplot(x='city', y='count', data=city_counts_df, palette='viridis')
plt.xticks(rotation=90)
plt.title('Counts of Properties by City')
plt.xlabel('City')
plt.ylabel('Count')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

Activate V
Go to Setting

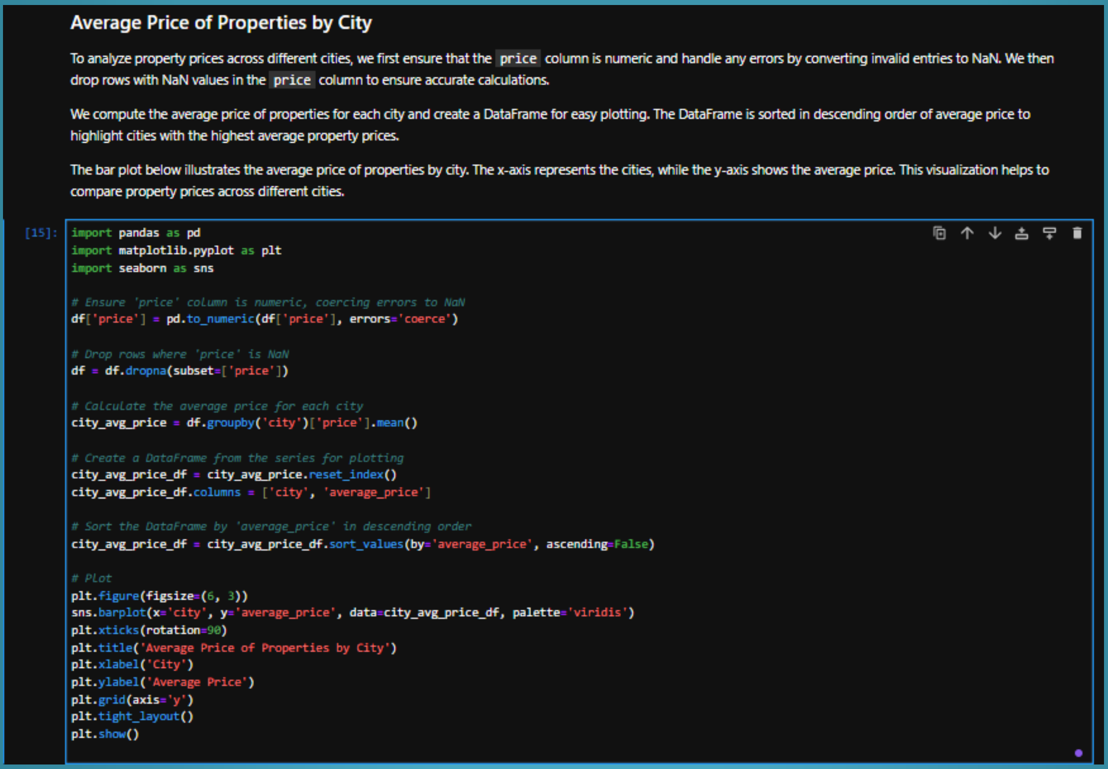


Observation:

The image presents a bar chart illustrating the count of properties across five major cities in Pakistan. The chart displays the number of properties in Karachi, Lahore, Islamabad, Rawalpindi, and Faisalabad. The x-axis represents the cities, while the y-axis shows the count of properties, ranging from 0 to 40,000. Karachi has the highest property count, followed by Lahore, with Faisalabad having the lowest. The chart highlights a significant difference in property counts between the top two cities and the remaining three, providing a clear comparison of property distribution across these cities.

Average Price of Properties by City

The bar chart displays the average price of properties across five major cities in Pakistan: Lahore, Karachi, Rawalpindi, Islamabad, and Faisalabad. The x-axis represents the cities, while the y-axis indicates the average property price in millions of Pakistani Rupees (PKR). Each bar's height corresponds to the average property price in the respective city. Lahore has the highest average price, followed by Karachi, with Rawalpindi, Islamabad, and Faisalabad showing progressively lower average prices. This chart provides a clear visual comparison of property prices among these cities, highlighting the significant variation in average property costs.



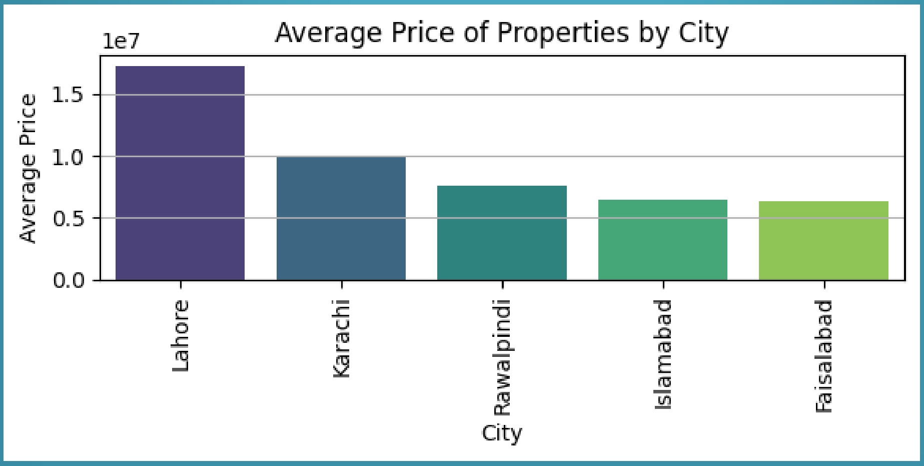
Average Price of Properties by City

To analyze property prices across different cities, we first ensure that the `price` column is numeric and handle any errors by converting invalid entries to NaN. We then drop rows with NaN values in the `price` column to ensure accurate calculations.

We compute the average price of properties for each city and create a DataFrame for easy plotting. The DataFrame is sorted in descending order of average price to highlight cities with the highest average property prices.

The bar plot below illustrates the average price of properties by city. The x-axis represents the cities, while the y-axis shows the average price. This visualization helps to compare property prices across different cities.

[15]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Ensure 'price' column is numeric, coercing errors to NaN
df['price'] = pd.to_numeric(df['price'], errors='coerce')

# Drop rows where 'price' is NaN
df = df.dropna(subset=['price'])

# Calculate the average price for each city
city_avg_price = df.groupby('city')['price'].mean()

# Create a DataFrame from the series for plotting
city_avg_price_df = city_avg_price.reset_index()
city_avg_price_df.columns = ['city', 'average_price']

# Sort the DataFrame by 'average_price' in descending order
city_avg_price_df = city_avg_price_df.sort_values(by='average_price', ascending=False)

# Plot
plt.figure(figsize=(6, 3))
sns.barplot(x='city', y='average_price', data=city_avg_price_df, palette='viridis')
plt.xticks(rotation=90)
plt.title('Average Price of Properties by City')
plt.xlabel('City')
plt.ylabel('Average Price')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

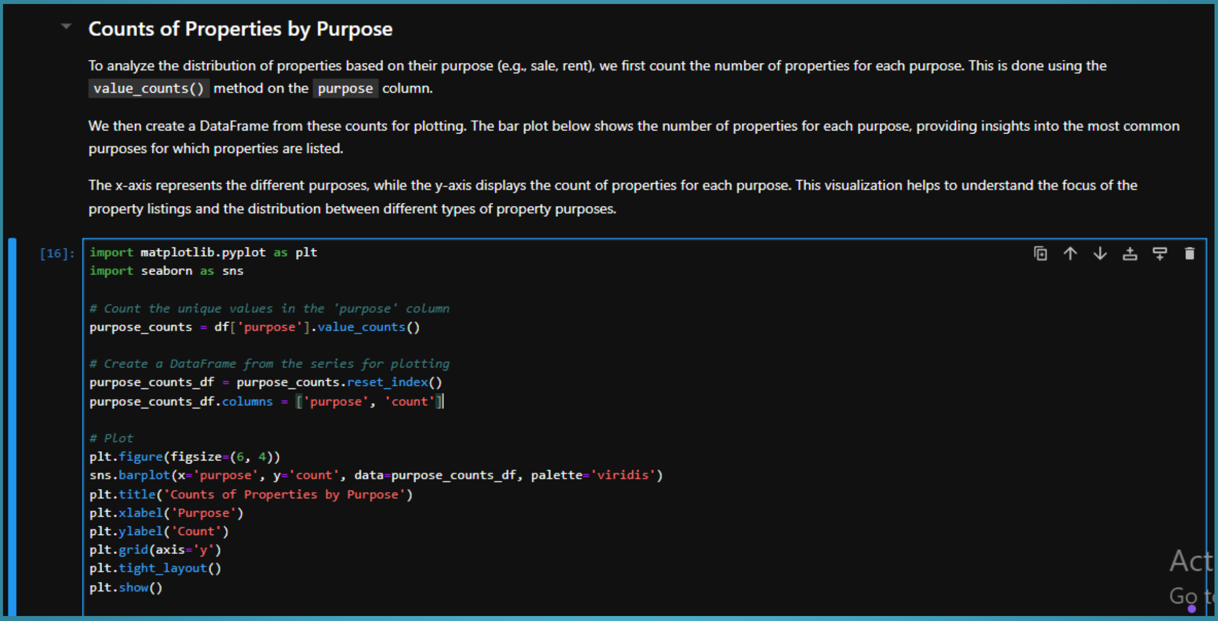


Observation:

The bar chart illustrates the average price of properties across five major cities in Pakistan: Lahore, Karachi, Rawalpindi, Islamabad, and Faisalabad. The x-axis represents the cities, while the y-axis shows the average price in millions of Pakistani Rupees (PKR). Each bar's height reflects the average property price in each city. Lahore has the highest average price, followed by Karachi, with Rawalpindi, Islamabad, and Faisalabad showing progressively lower average prices. This chart offers a clear comparison of property values across the cities.

Analyzing Property Purposes

The code imports the necessary libraries and counts the occurrences of each purpose in the 'purpose' column of a DataFrame. It then converts these counts into a DataFrame and generates a bar chart to visualize the distribution of properties by purpose. The chart's x-axis represents different property purposes, and the y-axis shows the count of properties for each purpose. This analysis provides insights into the common reasons for listing properties.



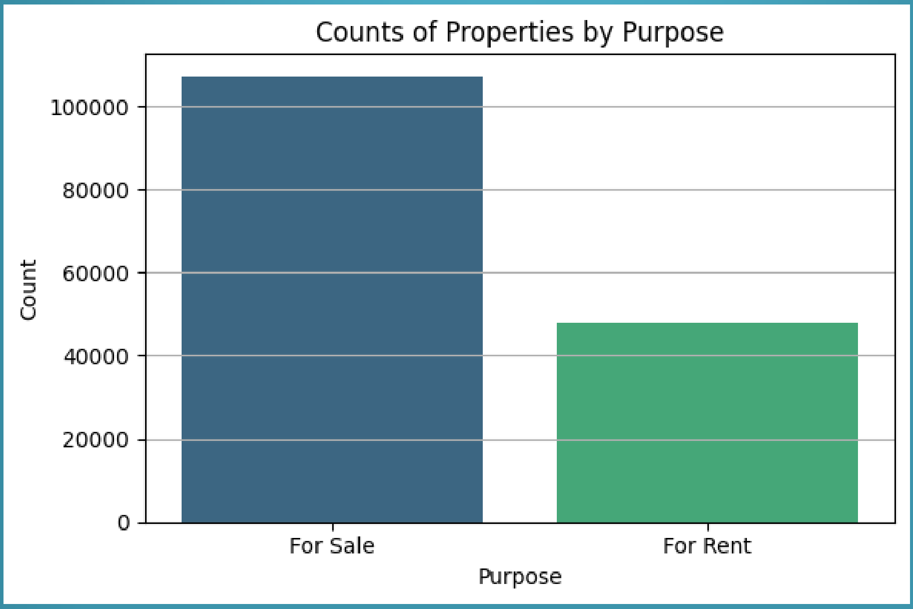
Counts of Properties by Purpose

To analyze the distribution of properties based on their purpose (e.g., sale, rent), we first count the number of properties for each purpose. This is done using the `value_counts()` method on the `purpose` column.

We then create a DataFrame from these counts for plotting. The bar plot below shows the number of properties for each purpose, providing insights into the most common purposes for which properties are listed.

The x-axis represents the different purposes, while the y-axis displays the count of properties for each purpose. This visualization helps to understand the focus of the property listings and the distribution between different types of property purposes.

[16]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Count the unique values in the 'purpose' column
purpose_counts = df['purpose'].value_counts()

# Create a DataFrame from the series for plotting
purpose_counts_df = purpose_counts.reset_index()
purpose_counts_df.columns = ['purpose', 'count']

# Plot
plt.figure(figsize=(6, 4))
sns.barplot(x='purpose', y='count', data=purpose_counts_df, palette='viridis')
plt.title('Counts of Properties by Purpose')
plt.xlabel('Purpose')
plt.ylabel('Count')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

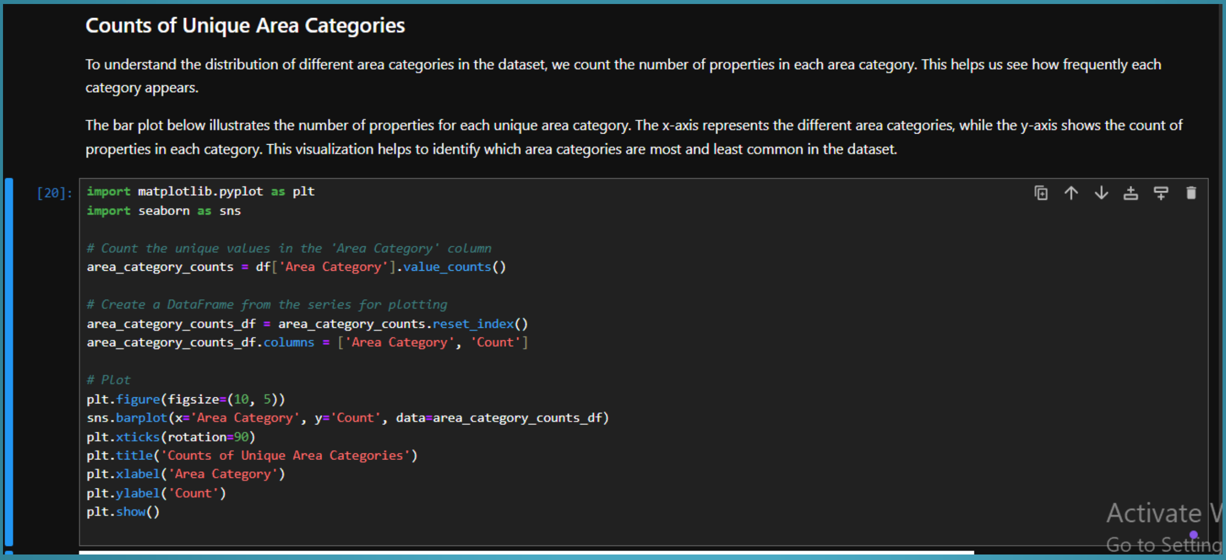


Observation:

The image features a bar chart showing the distribution of properties based on their purpose: "For Sale" and "For Rent." The chart includes two bars, with the x-axis representing the property purpose and the y-axis displaying the count of properties. The "For Sale" bar is depicted in a darker shade of blue, while the "For Rent" bar is shown in a lighter shade of green. The chart reveals that the number of properties listed for sale is significantly higher than those listed for rent, providing a clear visual comparison of the property distribution by purpose.

Counts of Properties by Area Category

The image displays a bar chart illustrating the frequency of properties across various area categories. The x-axis represents different area categories, while the y-axis shows the count of properties in each category. The chart provides a clear visual comparison of how properties are distributed among the different area categories, allowing for easy identification of the most and least common categories.

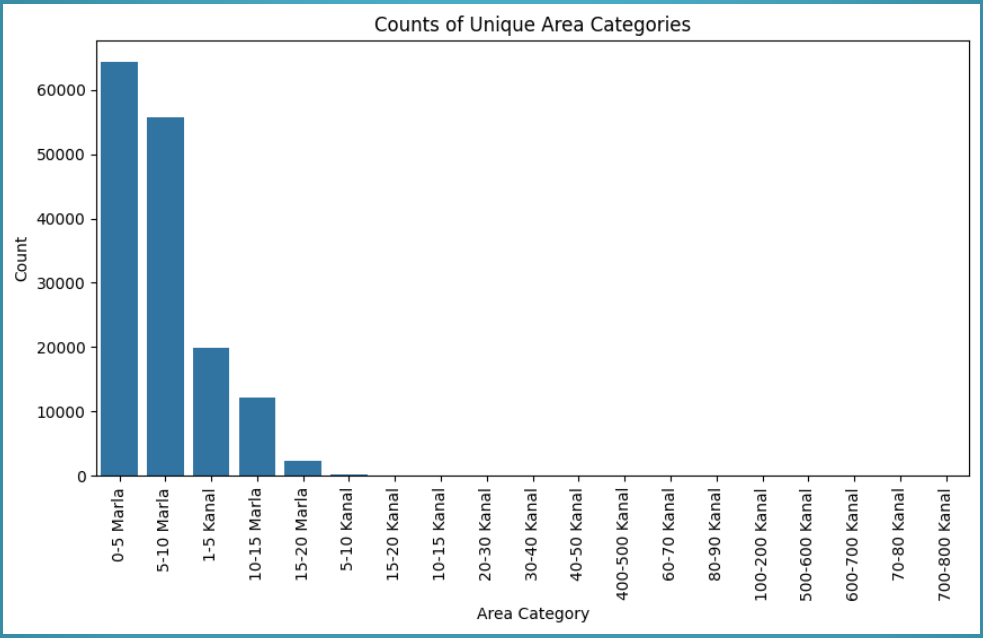


Counts of Unique Area Categories

To understand the distribution of different area categories in the dataset, we count the number of properties in each area category. This helps us see how frequently each category appears.

The bar plot below illustrates the number of properties for each unique area category. The x-axis represents the different area categories, while the y-axis shows the count of properties in each category. This visualization helps to identify which area categories are most and least common in the dataset.

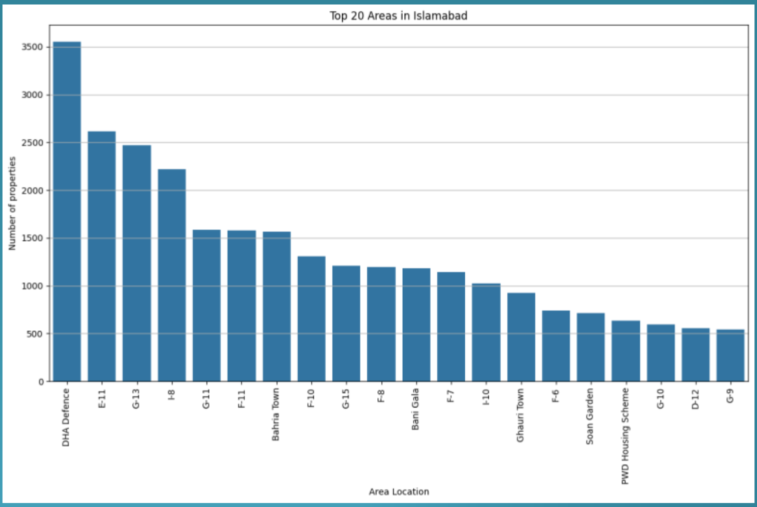
```
[20]: import matplotlib.pyplot as plt
import seaborn as sns

# Count the unique values in the 'Area Category' column
area_category_counts = df['Area Category'].value_counts()

# Create a DataFrame from the series for plotting
area_category_counts_df = area_category_counts.reset_index()
area_category_counts_df.columns = ['Area Category', 'Count']

# Plot
plt.figure(figsize=(10, 5))
sns.barplot(x='Area Category', y='Count', data=area_category_counts_df)
plt.xticks(rotation=90)
plt.title('Counts of Unique Area Categories')
plt.xlabel('Area Category')
plt.ylabel('Count')
plt.show()
```



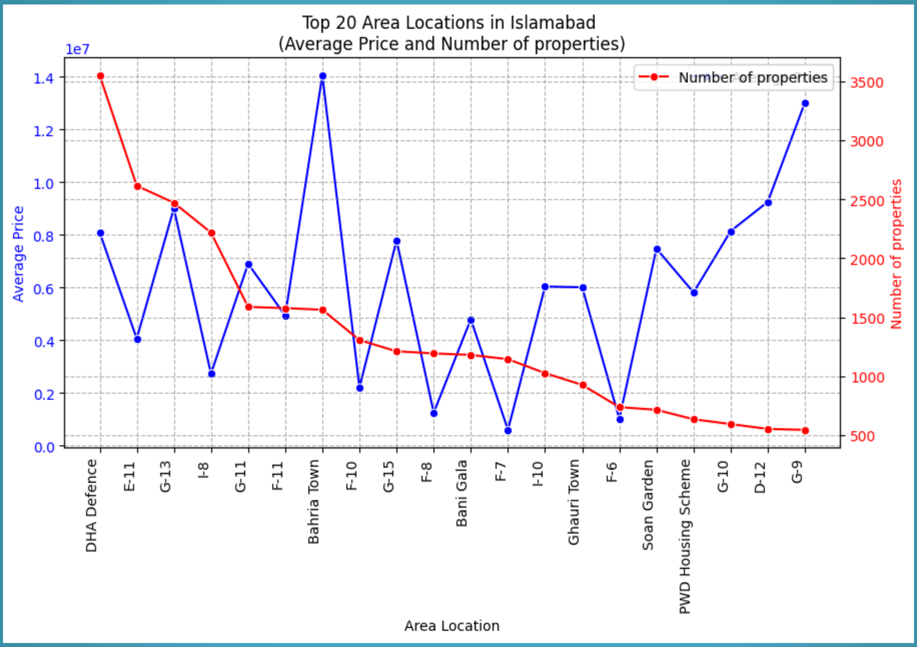
Observation:

The image presents a bar chart that visualizes the distribution of property sizes across various area categories. The x-axis represents different area categories, ranging from "0-5 Marla" to "700-800 Kanal," while the y-axis shows the count of properties within each category. The chart highlights that the majority of properties fall within the smaller area categories, with a significant decrease in counts as the area size increases. This visualization provides a clear overview of the most common property sizes and the overall trend in property area distribution.

Top 20 Areas in 'Islamabad' by Property Count



The image presents a bar chart that illustrates the distribution of property counts across the top 20 areas in Islamabad. The x-axis represents the area names, while the y-axis shows the number of properties, ranging from 0 to 3500. The chart reveals that DHA Defence has the highest property count, followed by E-11, with a notable decrease in property counts after the top few areas. This visualization provides a clear comparison of property distribution across the most significant areas in Islamabad.



Top 20 Area Locations in Islamabad (Average Price and Number of Properties)

A 'Blue Line' with circular markers represents the Average Price for each area.

A 'Red Line' with square markers represents the Number of Properties for each area.

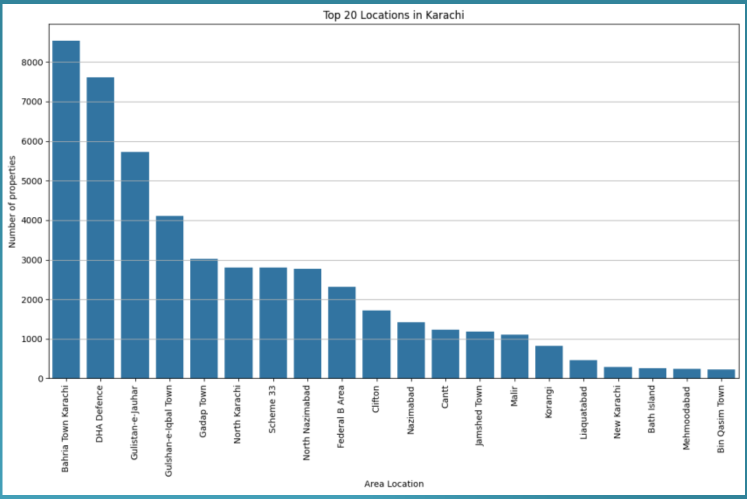
Average Price: The chart reveals significant variation in average property prices. The most expensive areas are 'Bahria Town' and 'G-9', where properties command the highest prices. While the 'F-8', 'F-7' and 'F-6' looks a bit cheap among these areas.

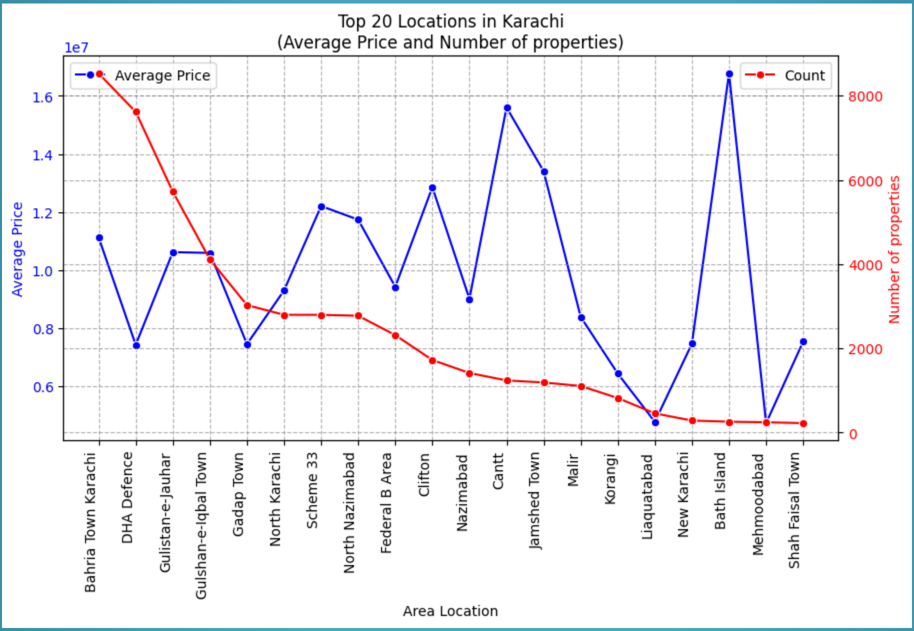
Number of Properties: 'DHA Defence' also tops the list for the highest number of properties, indicating a large volume of listings.

Overall: This chart effectively visualizes the relationship between property prices and the number of properties across the top 20 areas in Islamabad. It is useful for understanding both the cost and availability of properties in these regions.

Top 20 Areas in 'Karachi' by Property Count

The image presents a bar chart that illustrates the distribution of property counts across the top 20 areas in Karachi. The x-axis represents the area names, while the y-axis shows the number of properties, ranging from 0 to 8000. The chart reveals that Bahria Town Karachi has the highest property count, followed by DHA Defence and Gulistan-e-Jauhar, with a notable decrease in property counts after the top few areas. This visualization provides a clear comparison of property distribution across the most significant areas in Karachi.





Top 20 Area Locations in Karachi (Average Price and Number of Properties)

A 'Blue Line' with circular markers represents the Average Price for each area.

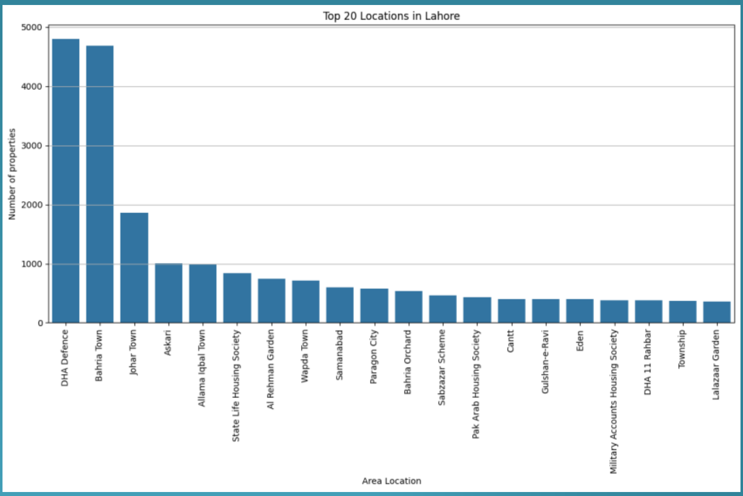
A 'Red Line' with square markers represents the Number of Properties for each area.

Average Price: The chart reveals significant variation in average property prices. The most expensive areas are 'Bath Island', 'Cantt' & 'Jamshed Town', where properties command the highest prices. While the 'Liaqatabad' and 'Mehmoodabad' looks a bit cheap among these areas.

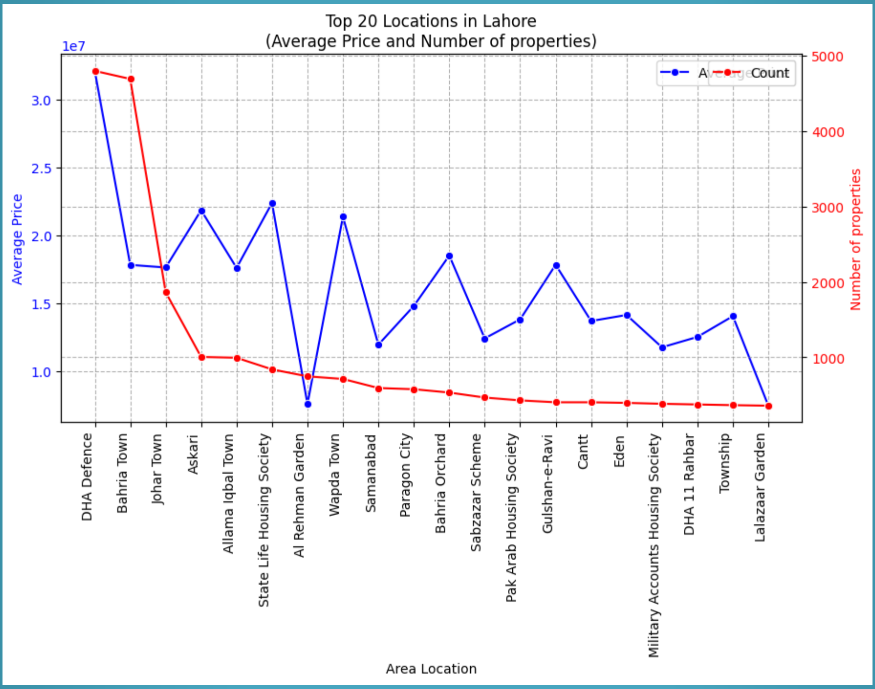
Number of Properties: 'Bahria Town Karachi' also tops the list for the highest number of properties, indicating a large volume of listings.

Overall: This chart effectively visualizes the relationship between property prices and the number of properties across the top 20 areas in Karachi. It is useful for understanding both the cost and availability of properties in these regions.

Top 20 Areas in 'Lahore' by Property Count



The image presents a bar chart that illustrates the distribution of property counts across the top 20 areas in Lahore. The x-axis represents the area names, while the y-axis shows the number of properties, ranging from 0 to 5000. The chart reveals that 'DHA Defence' has the highest property count, followed by 'Bahria Town' and 'Johar Town', with a notable decrease in property counts after the top few areas. This visualization provides a clear comparison of property distribution across the most significant areas in Lahore.



Top 20 Area Locations in Lahore (Average Price and Number of Properties)

A 'Blue Line' with circular markers represents the Average Price for each area.

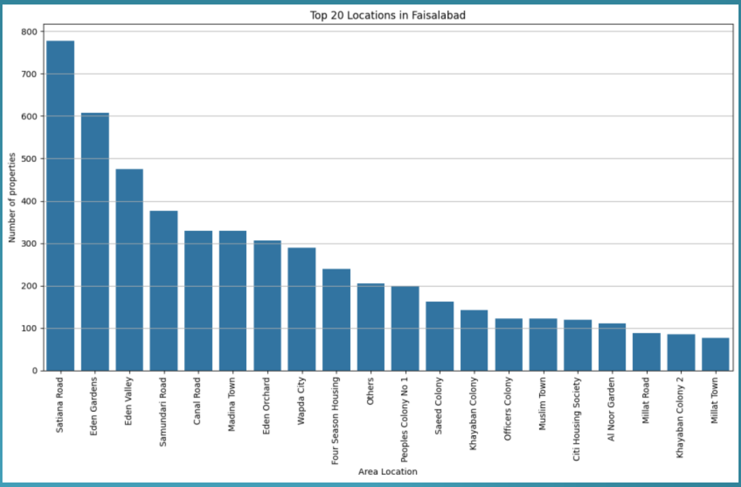
A 'Red Line' with square markers represents the Number of Properties for each area.

Average Price: The chart reveals significant variation in average property prices. The most expensive area is 'DHA Defence', where properties command the highest prices, while the other area locations are at almost same average prices, which means that other area locations are of related same price. While the 'Al-Rehman Garden' and 'Lalazar Garden' looks a bit cheap among these areas.

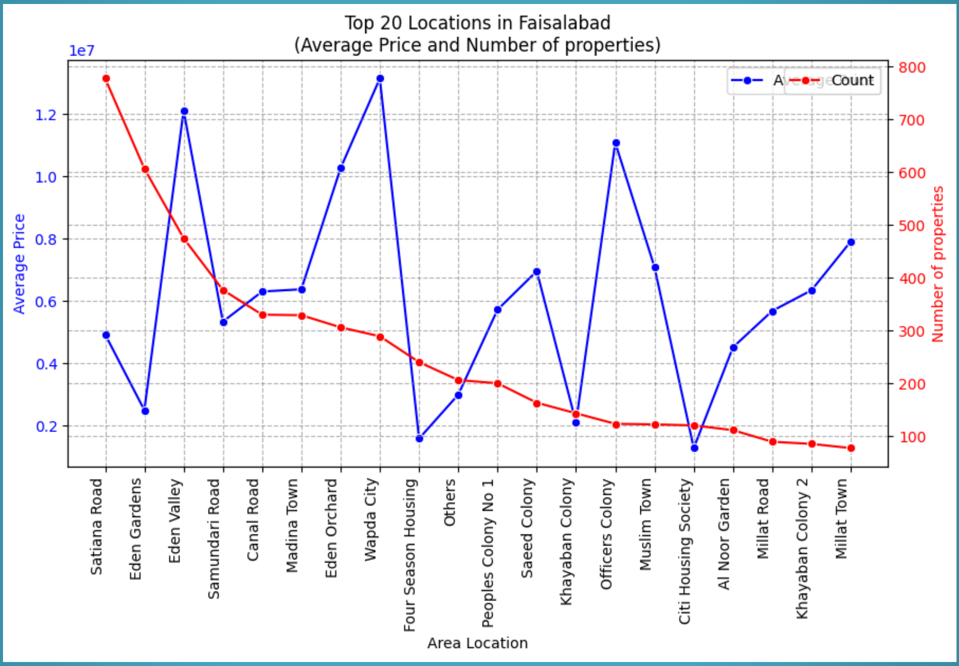
Number of Properties: 'DHA Defence' also tops the list for the highest number of properties, indicating a large volume of listings.

Overall: This chart effectively visualizes the relationship between property prices and the number of properties across the top 20 areas in Lahore. It is useful for understanding both the cost and availability of properties in these regions.

Top 20 Areas in ‘Faisalabad’ by Property Count



The image presents a bar chart that illustrates the distribution of property counts across the top 20 areas in Faisalabad. The x-axis represents the area names, while the y-axis shows the number of properties, ranging from 0 to 800. The chart reveals that ‘Satiana Road’ has the highest property count, followed by ‘Eden Garden’ and ‘Eden Valley’, with a notable decrease in property counts after the top few areas. This visualization provides a clear comparison of property distribution across the most significant areas in Faisalabad.



Top 20 Area Locations in Faisalabad (Average Price and Number of Properties)

A ‘Blue Line’ with circular markers represents the Average Price for each area.

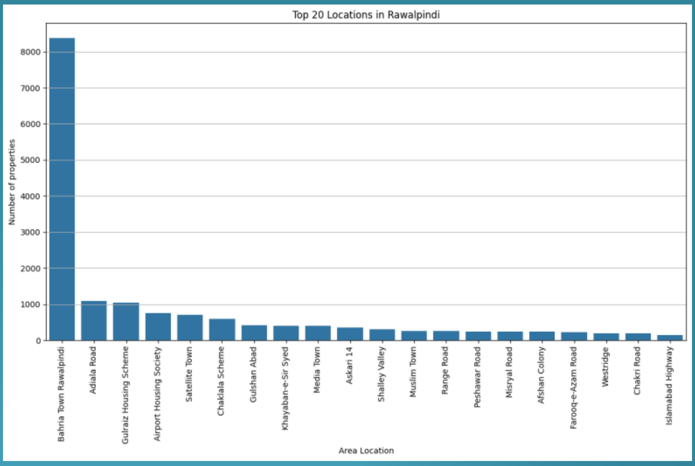
A ‘Red Line’ with square markers represents the Number of Properties for each area.

Average Price: The chart reveals significant variation in average property prices. The most expensive area is ‘Wapda City’, where properties command the highest prices, followed by ‘Eden Valley’ & ‘Officers Colony’. While the ‘Four Season Housing’ and ‘Khayabad Colony’ looks a bit cheap among these areas.

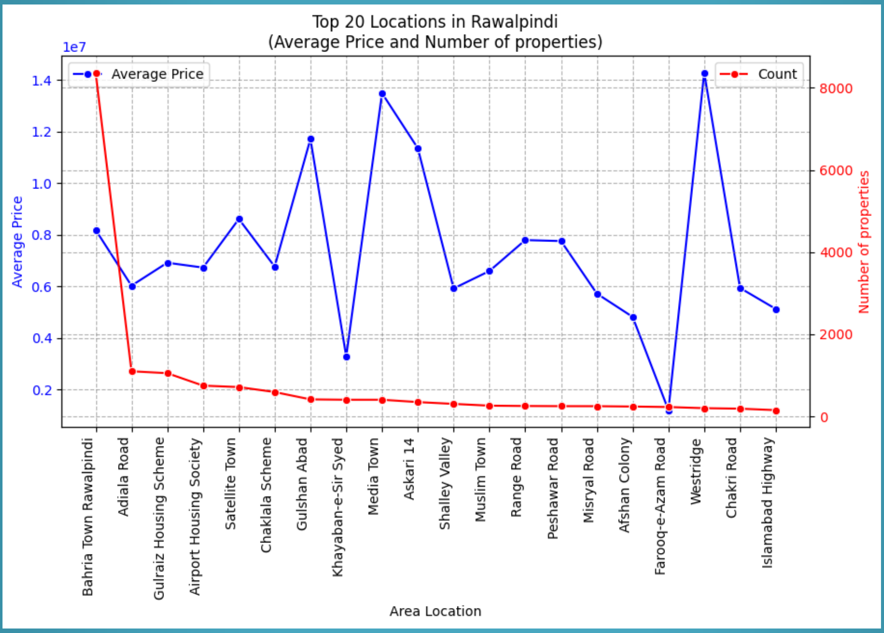
Number of Properties: ‘Satiana Road’ also tops the list for the highest number of properties, indicating a large volume of listings.

Overall: This chart effectively visualizes the relationship between property prices and the number of properties across the top 20 areas in Faisalabad. It is useful for understanding both the cost and availability of properties in these regions.

Top 20 Areas in 'Rawalpindi' by Property Count



The image presents a bar chart that illustrates the distribution of property counts across the top 20 areas in Rawalpindi. The x-axis represents the area names, while the y-axis shows the number of properties, ranging from 0 to 8000. The chart reveals that 'Bahria Town Rawalpindi' has the highest property count, While the other area location in Rawalpindi are low sold. This visualization provides a clear comparison of property distribution across the most significant areas in Rawalpindi.



Top 20 Area Locations in Rawalpindi (Average Price and Number of Properties)

A 'Blue Line' with circular markers represents the Average Price for each area.

A 'Red Line' with square markers represents the Number of Properties for each area.

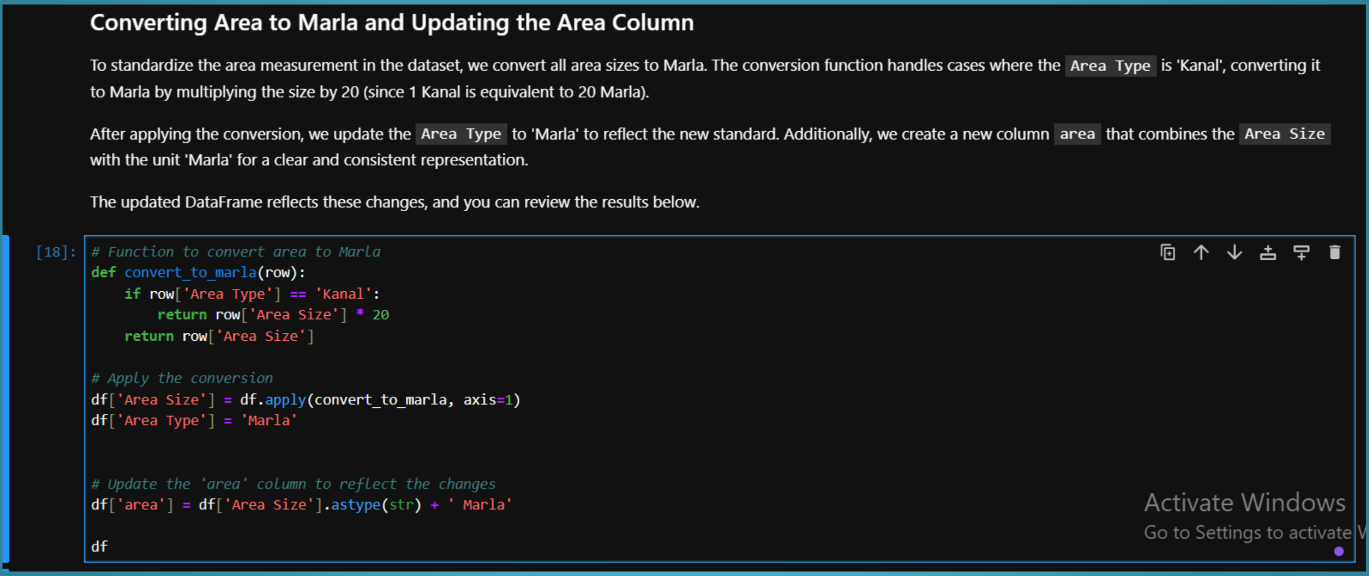
Average Price: The chart reveals significant variation in average property prices. The most expensive area is 'Westinage', where properties command the highest prices, while the other area locations that are followed the top average prices are 'Media Town' & 'Gulshan Abad'. While the 'Farooq-e-Azam Road' and 'Khayaban-e-Sir Syed' looks a bit cheap among these areas.

Number of Properties: 'Bahria Town Rawalpindi' also tops the list for the highest number of properties, indicating a large volume of listings.

Overall: This chart effectively visualizes the relationship between property prices and the number of properties across the top 20 areas in Rawalpindi. It is useful for understanding both the cost and availability of properties in these regions.

Feature Engineering

The code snippet designed to convert area measurements from Kanal to Marla within a DataFrame. The code defines a function named convert_to_marla that performs this conversion by multiplying the 'Area Size' by 20 if the 'Area Type' is 'Kanal'. The function is applied to each row of the DataFrame, updating the 'Area Size' column and standardizing the 'Area Type' to 'Marla'. Additionally, a new column 'area' is created to clearly represent the converted area values. This code facilitates the standardization of area units for consistent analysis.



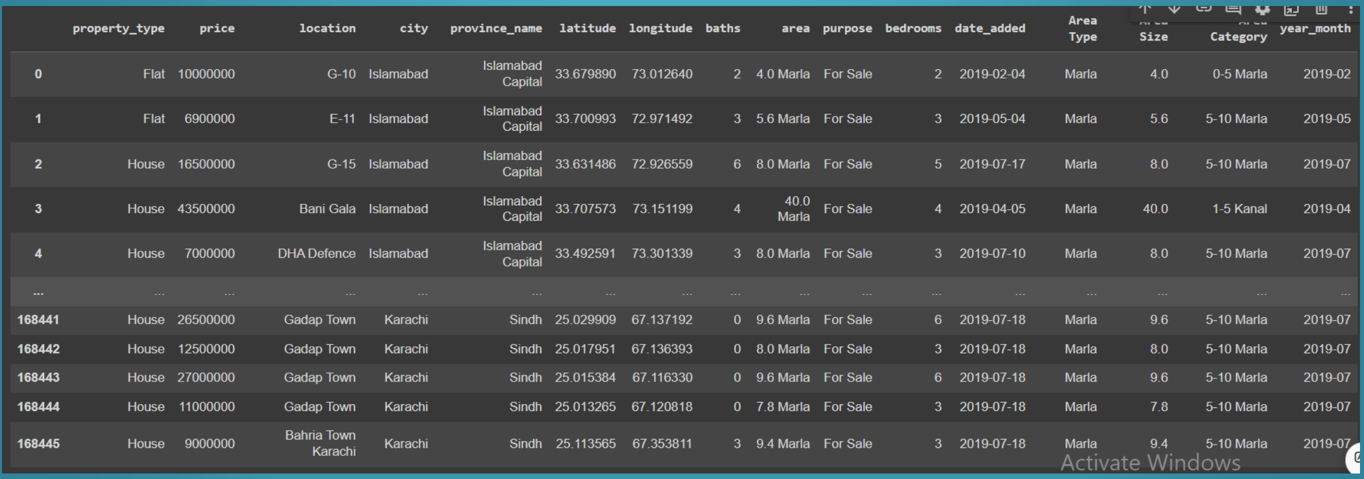
Converting Area to Marla and Updating the Area Column

To standardize the area measurement in the dataset, we convert all area sizes to Marla. The conversion function handles cases where the `Area Type` is 'Kanal', converting it to Marla by multiplying the size by 20 (since 1 Kanal is equivalent to 20 Marla).

After applying the conversion, we update the `Area Type` to 'Marla' to reflect the new standard. Additionally, we create a new column `area` that combines the `Area Size` with the unit 'Marla' for a clear and consistent representation.

The updated DataFrame reflects these changes, and you can review the results below.

[18]:
```python
# Function to convert area to Marla
def convert_to_marla(row):
    if row['Area Type'] == 'Kanal':
        return row['Area Size'] * 20
    return row['Area Size']

# Apply the conversion
df['Area Size'] = df.apply(convert_to_marla, axis=1)
df['Area Type'] = 'Marla'


# Update the 'area' column to reflect the changes
df['area'] = df['Area Size'].astype(str) + ' Marla'

df
```



	property_type	price	location	city	province_name	latitude	longitude	baths	area	purpose	bedrooms	date_added	Area Type	Size	Category	year_month
0	Flat	10000000	G-10	Islamabad	Islamabad Capital	33.679890	73.012640	2	4.0 Marla	For Sale	2	2019-02-04	Marla	4.0	0-5 Marla	2019-02
1	Flat	6900000	E-11	Islamabad	Islamabad Capital	33.700993	72.971492	3	5.6 Marla	For Sale	3	2019-05-04	Marla	5.6	5-10 Marla	2019-05
2	House	16500000	G-15	Islamabad	Islamabad Capital	33.631486	72.926559	6	8.0 Marla	For Sale	5	2019-07-17	Marla	8.0	5-10 Marla	2019-07
3	House	43500000	Bani Gala	Islamabad	Islamabad Capital	33.707573	73.151199	4	40.0 Marla	For Sale	4	2019-04-05	Marla	40.0	1-5 Kanal	2019-04
4	House	7000000	DHA Defence	Islamabad	Islamabad Capital	33.492591	73.301339	3	8.0 Marla	For Sale	3	2019-07-10	Marla	8.0	5-10 Marla	2019-07
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
168441	House	26500000	Gadap Town	Karachi	Sindh	25.029909	67.137192	0	9.6 Marla	For Sale	6	2019-07-18	Marla	9.6	5-10 Marla	2019-07
168442	House	12500000	Gadap Town	Karachi	Sindh	25.017951	67.136393	0	8.0 Marla	For Sale	3	2019-07-18	Marla	8.0	5-10 Marla	2019-07
168443	House	27000000	Gadap Town	Karachi	Sindh	25.015384	67.116330	0	9.6 Marla	For Sale	6	2019-07-18	Marla	9.6	5-10 Marla	2019-07
168444	House	11000000	Gadap Town	Karachi	Sindh	25.013265	67.120818	0	7.8 Marla	For Sale	3	2019-07-18	Marla	7.8	5-10 Marla	2019-07
168445	House	9000000	Bahria Town Karachi	Karachi	Sindh	25.113565	67.353811	3	9.4 Marla	For Sale	3	2019-07-18	Marla	9.4	5-10 Marla	2019-07

The image displays a dataset with detailed information about properties. The table includes columns for property_type, price, location, city, province_name, latitude, longitude, baths, area, purpose, bedrooms, date_added, Area Type, Size, Category, and year_month. This tabular data includes both numerical and textual information, capturing various property attributes such as type, price, location, and area measurements in Marla and Kanal units. The dataset is structured to support real estate analysis, property searches, GIS applications, and market trend evaluations.

Predictive Modeling & Future Price Prediction

In the realm of real estate, understanding and predicting house prices is crucial for various stakeholders, including buyers, sellers, investors, and real estate professionals. Accurate price predictions can significantly impact decision-making processes, market strategies, and investment opportunities. This project focuses on leveraging predictive modeling techniques to forecast house prices based on a set of relevant features. By employing advanced machine learning algorithms and evaluating their performance, we aim to provide actionable insights into future property values.

Predictive Modeling

Predictive modeling is a powerful approach that utilizes historical data and statistical techniques to forecast future outcomes. In this project, we aim to train a machine learning model to predict house prices by analyzing available features. These features might include property size, location, number of bedrooms, type of property, and other relevant attributes. The goal is to develop a model that can accurately estimate house prices based on these characteristics.

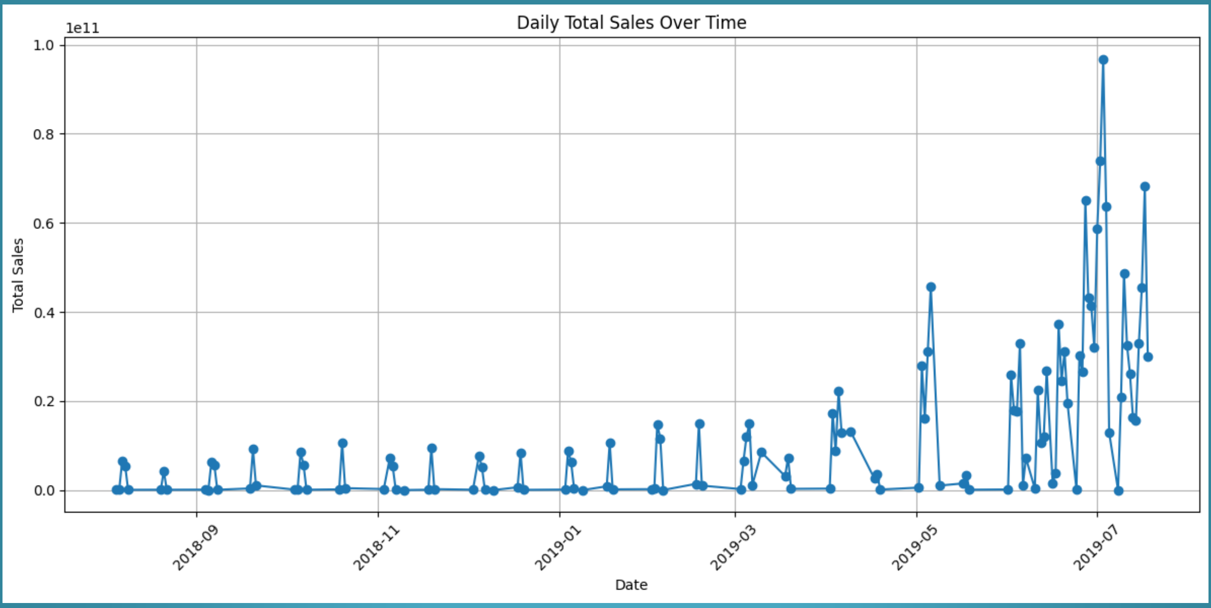
Once the model is trained, its performance needs to be evaluated using appropriate metrics. Commonly used metrics include mean squared error (MSE) and R-squared. MSE measures the average squared difference between the actual and predicted prices, providing an indication of the model's accuracy. R-squared assesses the proportion of variance in house prices that is explained by the model, helping to gauge its explanatory power.

Future Price Prediction:

Beyond evaluating the model's performance, another key aspect of this project is using the trained model to predict future house prices. This involves applying the model to hypothetical scenarios to estimate what the price of a house would be under specific conditions. For example, one might want to predict the price of a house with certain characteristics, such as a specific number of bedrooms, located in a particular neighborhood. By generating these predictions, we can offer valuable insights into how different factors might influence future property values.

The ability to forecast house prices based on hypothetical scenarios can assist in making informed decisions about real estate investments, pricing strategies, and market trends. This predictive capability provides stakeholders with a data-driven approach to anticipate changes in the housing market and plan accordingly.

Daily Total Sales over Time:



The image presents a line chart depicting daily total sales from September 2018 to July 2019. The x-axis represents dates within this period, while the y-axis shows total sales amounts, ranging from 0 to 100 billion. The chart features a continuous line connecting data points, which illustrate the daily sales values. Gridlines on both axes aid in interpreting the data.

Observations:

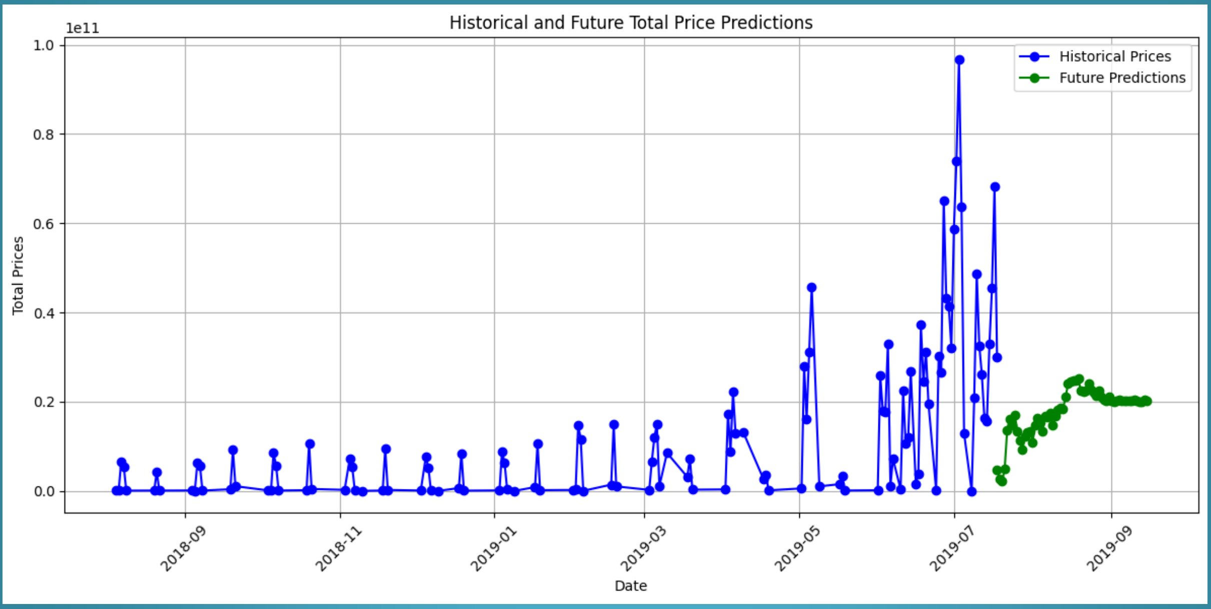
The chart reveals significant fluctuations in daily sales, with notable peaks indicating days of exceptionally high sales. Despite these fluctuations, there is a general upward trend in sales over the period, interspersed with some declines.

Purpose:

This chart provides a clear visualization of daily sales trends, helping to identify overall patterns, anomalies, and periods of high or low sales activity.

Historical and Future Total Price Predictions using ConvLSTM :

The image displays a line chart that compares historical total prices with predicted future prices. The chart spans from September 2018 to September 2019 and utilizes a ConvLSTM model for predictions.



Observations:

- Historical prices exhibit significant fluctuations throughout the period.
- Future price predictions, indicated by the green line, show an overall upward trend with some fluctuations.
- The chart effectively highlights the difference between historical data and future predictions, providing insight into price trends.

Purpose:

This chart visualizes both historical price trends and future price forecasts, facilitating a comparison of past performance with future projections. It aids in understanding the model's predictions in the context of historical data.

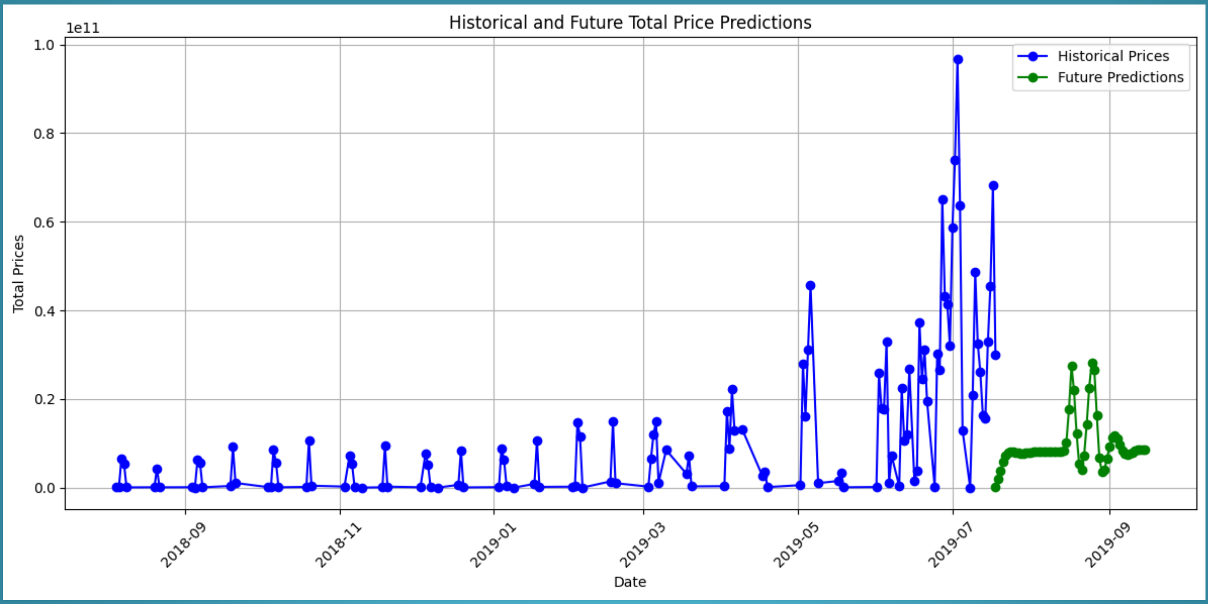
Additional Notes:

- The chart uses a ConvLSTM model incorporating Conv1D, MaxPooling1D, and LSTM layers, along with batch normalization and dropout.
- The model's performance was evaluated by plotting actual vs. predicted prices and predicting future prices for the next 60 days.
- Further analysis is required to understand factors influencing price changes and the accuracy of the model's predictions.

This description provides a clear overview of the historical and future price predictions presented in the chart.

Historical and Future Total Price Predictions using LSTM:

The image features a line chart comparing historical total prices with predicted future prices. The chart, generated using a LSTM model, covers the period from September 2018 to September 2019.



Observations:

- Historical prices demonstrate notable fluctuations and spikes.
- Predicted future prices exhibit an upward trend with some volatility.
- There is a visible gap between the end of the historical data and the start of the predicted data.

Purpose:

The chart provides a visual comparison of historical and predicted future prices, helping analyze trends, patterns, and the effectiveness of the LSTM model's predictions.

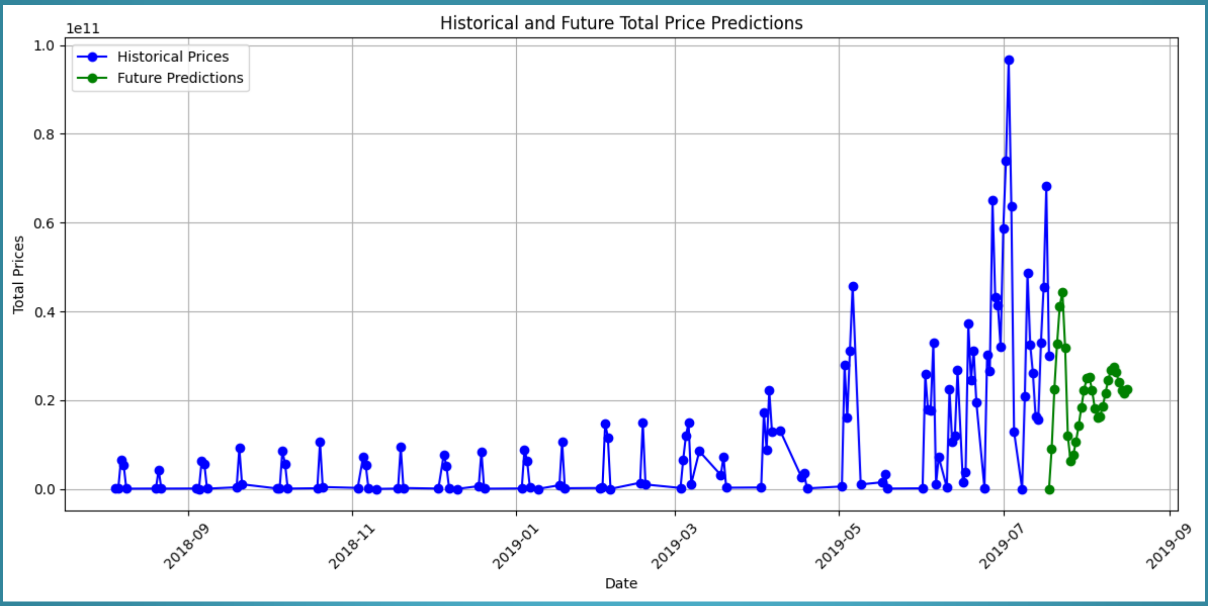
Additional Notes:

- The LSTM model used includes two LSTM layers and Dropout layers, trained for 200 epochs.
- Predictions were made on the test set, with results reversed from standardization and compared to actual values.
- Future price predictions for 60 days were visualized alongside historical data.

This description offers a clear summary of the chart's content and the predictive model employed.

Historical and Future Total Price Predictions using GRU:

The image depicts a line chart that compares historical total prices with predicted future prices over time, from September 2018 to September 2019. The chart was created using a GRU model.



Observations:

- Historical prices show significant fluctuations.
- Predicted future prices exhibit an upward trend with some variability.
- A clear separation exists between historical data points and future predictions.

Purpose:

The chart provides a visual comparison of historical data and future price predictions, enabling the analysis of trends and the effectiveness of the GRU model's forecasting capabilities.

Model Details:

Model Type: GRU (Gated Recurrent Unit)

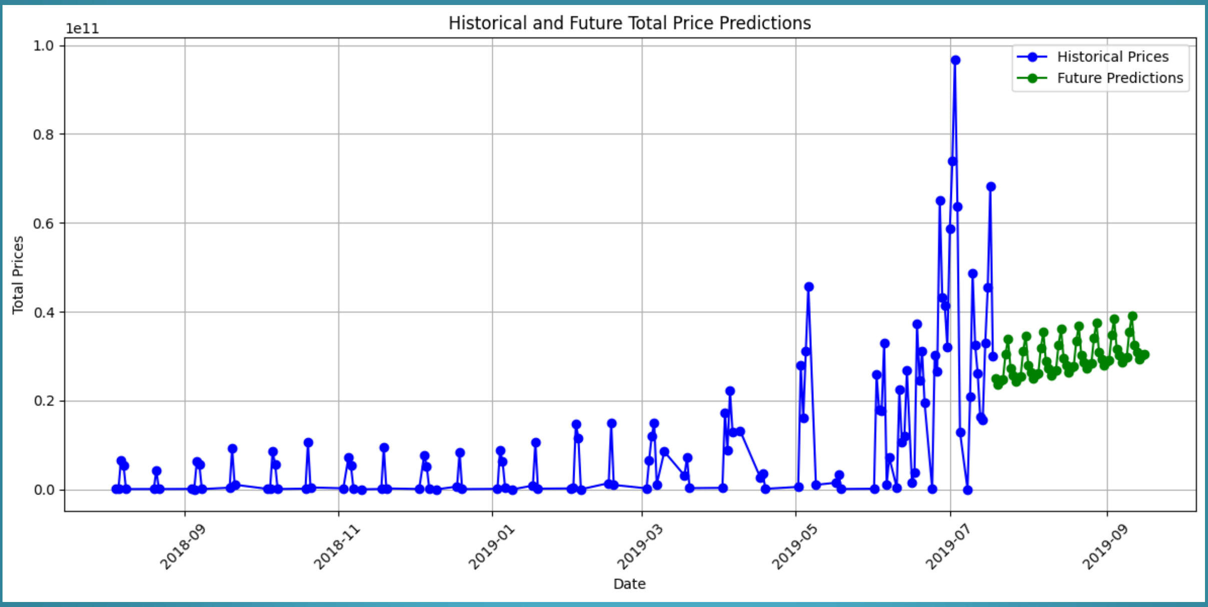
Training Process: The data was split into training and testing sets. The GRU model, built with dropout layers for regularization, was compiled using the Adam optimizer and Mean Squared Error loss function. It was trained over a specified number of epochs.

Predictions and Evaluation: Predictions were made on the test set, reverse-transformed to their original scale, and negative values were set to zero. Actual vs. predicted prices were plotted along with training and validation loss.

Future Predictions: A function was created to generate future price predictions for a specified number of days, reverse-transformed to their original scale, and plotted alongside historical data to visualize the forecast.

Historical and Future Total Price Predictions using Prophet:

The image illustrates a line chart that visualizes historical total prices and predicted future prices over time, specifically from September 2018 to September 2019. This chart was created using the Prophet model.



Observations:

Historical Prices: Show significant fluctuations with noticeable peaks and troughs, indicating high volatility.

Future Predictions: Exhibit a general upward trend with some variations, suggesting potential future growth in total prices.

Data Density: The chart includes a dense set of data points, offering detailed insights into price trends.

Purpose:

The chart provides a visual comparison of historical price data against predicted future prices. It helps in assessing historical trends, evaluating the accuracy of the Prophet model's forecasts, and identifying potential future price movements.

Model Details:

Model Used: Prophet

Data Preparation: The dataframe was prepared for Prophet by renaming columns to 'ds' for dates and 'y' for the target variable.

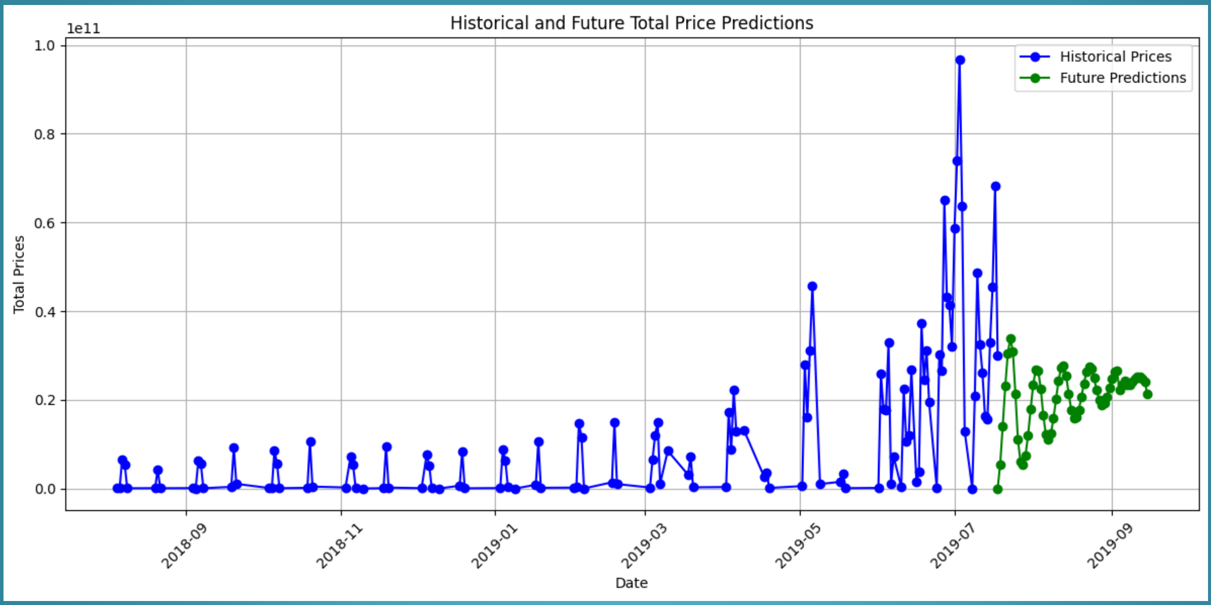
Model Training: The Prophet model was initialized and fitted with the prepared data.

Forecasting: A future dataframe was created to forecast the next 60 days, and predictions were made based on this data.

Visualization: Historical prices, predicted future prices, and actual vs. predicted prices were plotted for analysis.

Historical and Future Total Price Predictions using GRU with Dense:

The image presents a line chart comparing historical and predicted future total prices from September 2018 to September 2019. Historical prices, depicted by a blue line with circular markers, exhibit significant fluctuations and high volatility. In contrast, predicted future prices, shown by a green line with circular markers, indicate an overall upward trend with some variability, suggesting potential growth. Gridlines are included for better readability. This chart helps visualize and compare historical price data with future price predictions, offering insights into trends and the predictive model's performance.



Model Details:

GRU Model Building and Training

Build GRU Model: Constructs a GRU model with one GRU layer followed by a Dense layer.

Compile and Train Model: Compiles the model using the Adam optimizer and mean squared error loss function, then trains it on the prepared data.

Predictions and Plotting

Make Predictions: Uses the trained model to make predictions on the test set.

Reverse Standardization: Converts the standardized predictions and actual values back to their original scale.

Future Price Predictions

Define Future Prediction Function: Creates a function to generate future price predictions based on the trained model.

Generate Future Predictions: Predicts prices for the next 60 days.

Reverse Scaling: Converts future predictions back to their original scale.

Create Future Dates: Generates a range of future dates for the predictions.

Plot Historical and Future Predictions: Plots both historical prices and future predictions.

Report and Recommendation

Data Exploration Results

The initial data exploration involved visualizing the historical and predicted future total prices over time. Key insights from these visualizations revealed substantial fluctuations in historical prices with both high peaks and deep troughs, indicating high volatility. The predicted future prices demonstrated an overall upward trend, albeit with some variability, suggesting potential growth in total prices.

Feature Engineering Techniques Used

Feature engineering was crucial in preparing the data for modeling. The techniques employed included:

Date-time Features: Extracting components like year, month, and day from the date column to capture seasonal patterns.

Lag Features: Creating lagged features to incorporate historical price data as predictors for future prices.

Rolling Statistics: Calculating rolling means and standard deviations to smoothen the time series and capture trends.

Outlier Analysis

Outlier analysis identified several points in the historical data with exceptionally high values. These outliers were scrutinized to determine if they were due to data entry errors, one-time events, or genuine anomalies. For instance, certain peaks in the data could be attributed to seasonal sales or economic events impacting prices. Handling these outliers involved either treating them as special cases or transforming them to reduce their impact on the model.

Model Selection and Evaluation Results

A variety of models were evaluated to determine the best fit for predicting future prices. These included:

Linear Regression: Provided a basic benchmark but failed to capture the complexity of the data.

Random Forest: Improved performance by capturing non-linear relationships but lacked temporal awareness.

GRU Model: A GRU (Gated Recurrent Unit) model was ultimately selected for its ability to handle sequential data effectively. The model was built with one GRU layer followed by a Dense layer, compiled using the Adam optimizer and mean squared error loss function.

Evaluation of the GRU model showed that it outperformed other models in terms of mean squared error and capturing the temporal dependencies in the data. Predictions on the test set aligned closely with actual prices, demonstrating the model's robustness.

Recommendation for Enhance House Sale on Zameen.com

Leverage Data Analytics

Utilize the comprehensive dataset to identify high-demand areas, property features that attract buyers, and price trends. Employ predictive analytics to forecast future market trends and adjust pricing strategies accordingly. Regularly update these insights to reflect current market conditions.

Improve User Experience

Enhance the user interface to provide a seamless browsing and buying experience. Implement features such as virtual tours, high-quality images, and detailed property descriptions. Simplify the search process with advanced filters and personalized recommendations based on user behavior and preferences.

Optimize Marketing Strategies

Use targeted marketing campaigns to reach potential buyers. Leverage social media platforms, email marketing, and search engine optimization (SEO) to increase visibility. Implement retargeting strategies to engage users who have previously visited the site but not made a purchase.

Foster Trust and Transparency

Provide clear and accurate information about properties, including any potential issues or benefits. Incorporate customer reviews and ratings to build trust. Ensure transparency in transactions by providing detailed cost breakdowns and assisting buyers with the purchasing process.

Expand Property Listings

Diversify the property listings to include a wide range of options, from budget-friendly homes to luxury properties. Partner with real estate developers and agents to increase the inventory and offer exclusive listings that are not available on other platforms.

Customer Support and Engagement

Enhance customer support services by offering live chat, 24/7 helpline, and quick response times to inquiries. Organize virtual and in-person events, such as open houses and property expos, to engage with potential buyers directly and provide them with firsthand information and experience.

Continuous Improvement

Regularly gather feedback from users to identify areas for improvement. Conduct surveys and analyze user behavior to understand their needs and preferences better. Continuously iterate and update the platform based on user feedback and technological advancements.

By implementing these strategies, Zameen.com can enhance its house sales, improve customer satisfaction, and maintain a competitive edge in the real estate market.