Chohort 7: Screening Assignment

Name: A. S. M. Toha

Email:asmtohahere@gmail.com

Github Link:https://github.com/asmtoha/Cohort7_asmToha

Linkedin:https://www.linkedin.com/in/asmtoha/

Task 1 - Download the dataset and create a Google Colab notebook, an IPython Notebook

Download the dataset and create a Google Colab notebook, an IPython Notebook, or a Jupyter Notebook for this assignment.

Data source: https://www.kaggle.com/datasets/harlfoxem/housesalesprediction. It is about house sales information for king county.

Task 2 - familiarize yourself with the data structure, load the dataset into a pandas DataFrame and display the first 10 rows

To familiarize yourself with the data structure, load the dataset into a pandas DataFrame and display the first 10 rows.

```
#libraries needed for these problems
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
pd.set_option('display.max_columns', None)
#set columns limits to maiximum so that we can see every column
clearly because by default head() function hide columns if ther is so
much columns in data
```

```
#kc_house_data csv file loaded here
data = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/kc_house_data.csv')

#head function used for quick inspection of the dataset without seeing
large dataset
data.head(10)
```

```
{"type":"dataframe","variable_name":"data"}
```

Task 3 - Display basic information about the dataset (column names, data types, non-null counts).

Display basic information about the dataset (column names, data types, non-null counts).

```
# info() function helps to find information about data type,column,non
null valus all together
print('Data information about columns,non null values and data type:\
n')
data.info()
```

```
Data information about columns,non null values and data type:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
```

```
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

Task 4 - Data Cleaning

- Identify and handle missing values appropriately. Document your approach for each column.
- Check for and remove any duplicate records.
- Convert date columns to a proper datetime format.
- Create the derived feature 'total_area' by adding 'sqft_living' and 'sqft_lot'.
- Create derived features that might be useful for analysis (e.g., property age from year built or date, price per square foot, and so on).
- Identify and handle outliers using appropriate statistical methods.

```
print('Checking null values if any:\n',data.isnull().sum())
```

```
Checking null values if any:
 id               0
date              0
price             0
bedrooms          0
bathrooms         0
sqft_living       0
sqft_lot          0
floors            0
waterfront        0
view              0
condition         0
grade             0
sqft_above        0
sqft_basement     0
yr_built          0
yr_renovated      0
zipcode           0
lat               0
long              0
sqft_living15     0
sqft_lot15        0
dtype: int64
```

```
print('Checking duplicates if any:\n',data.duplicated().sum())
```

```
Checking duplicates if any:
 0
```

Avobe,there is no missing and duplicates values. So, we do not need to handle missing and duplicates values. But the process is for missing handling data["col_name"].fillna(data["col_name"].meadin/mode(),inplace=True). Median for numeriacl and Mode for catagorical. For dropping duplicates we can use data.drop_duplicates(inplace=True)

```
print("Converting 'date' columns to proper dateTime format:\n")
data['date'] = pd.to_datetime(data['date'])
```

```
Converting 'date' columns to proper dateTime format:
```

```
data.head(10)
```

```
{"type":"dataframe","variable_name":"data"}
```

```
#derived features
data['total_area'] = data['sqft_living'] + data['sqft_lot']
data['property_age'] = 2025 - data['yr_built']
data['price_per_sqft'] = data['price'] / data['sqft_living']
data['year'] = data['date'].dt.year
data['month'] = data['date'].dt.month_name()
data['year_month'] = data['date'].dt.strftime('%Y-%m')
```

```
data.head(10)
```

```
{"type":"dataframe","variable_name":"data"}
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 27 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              21613 non-null  int64
 1   date            21613 non-null  datetime64[ns]
 2   price           21613 non-null  float64
 3   bedrooms        21613 non-null  int64
 4   bathrooms       21613 non-null  float64
 5   sqft_living     21613 non-null  int64
 6   sqft_lot        21613 non-null  int64
 7   floors          21613 non-null  float64
 8   waterfront      21613 non-null  int64
 9   view            21613 non-null  int64
 10  condition       21613 non-null  int64
 11  grade           21613 non-null  int64
 12  sqft_above      21613 non-null  int64
 13  sqft_basement   21613 non-null  int64
 14  yr_built        21613 non-null  int64
 15  yr_renovated    21613 non-null  int64
```

```
 16  zipcode          21613 non-null  int64
 17  lat              21613 non-null  float64
 18  long             21613 non-null  float64
 19  sqft_living15    21613 non-null  int64
 20  sqft_lot15       21613 non-null  int64
 21  total_area       21613 non-null  int64
 22  property_age     21613 non-null  int64
 23  price_per_sqft   21613 non-null  float64
 24  year             21613 non-null  int32
 25  month            21613 non-null  object
 26  year_month       21613 non-null  object
dtypes: datetime64[ns](1), float64(6), int32(1), int64(17), object(2)
memory usage: 4.4+ MB
```

Outliers

In simplea word,outlier is different data among a data group. It is important to remove outliers to avoid error in measurement. We use five number summary to remove outliers.

- Minimum Value
- First Quartile (Q1)
- Median
- Third Quartile (Q3)
- Maximum Value

Theorotical equation:

Q1 = (25/100) * (n+1)

Q3 = (75/100) * (n+1) = >Q1,Q3 will provide index position.

We need to find out inter quartile range (IQR). IQR = Q3 - Q1

Minimum Value = Q1 - 1.5(IQR)

Maximum Value = Q3 + 1.5(IQR)

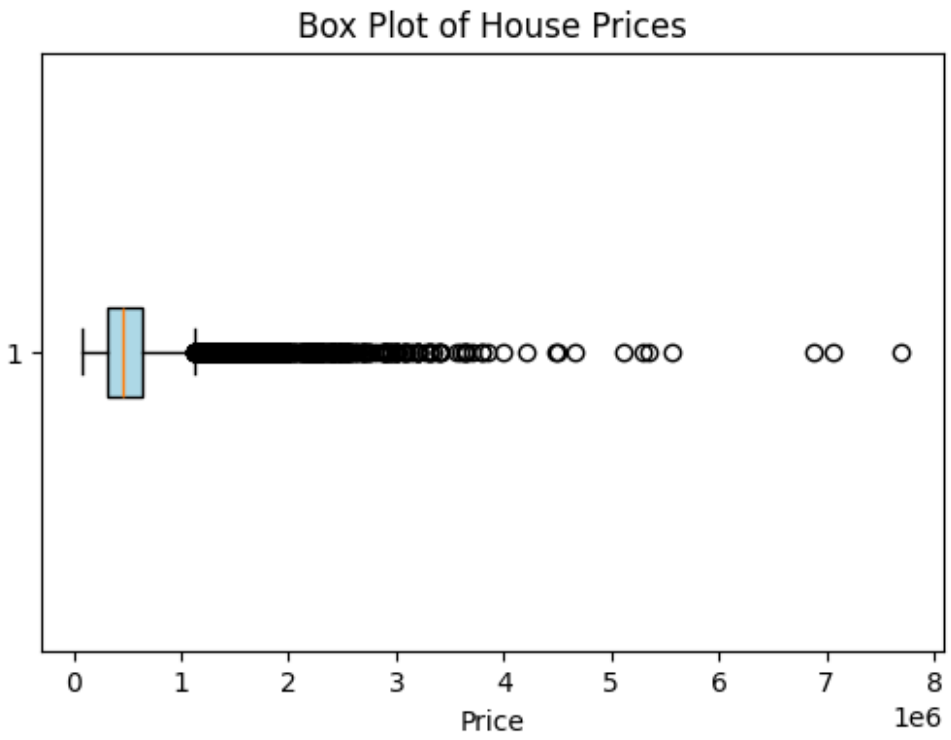
Any values < minimum value and any values > maximum value are outliers. We can use box plot to visualize it

```
#frist quatile,thrid quartile,IQR
Q1 = data['price'].quantile(0.25)
print('First quartile: ',Q1)
Q3 = data['price'].quantile(0.75)
print('Third quartile: ',Q3)
IQR = Q3 - Q1
print('Inter Quartile Range: ',IQR)

# Define outlier range
minimum_val = Q1 - 1.5 * IQR
print('Minimum value: ',minimum_val)
maximum_val = Q3 + 1.5 * IQR
print('Maximum value: ',maximum_val)
```

```
First quartile:  321950.0
Third quartile:  645000.0
Inter Quartile Range:  323050.0
Minimum value:  -162625.0
Maximum value:  1129575.0
```

```
# box plot
plt.figure(figsize=(6, 4))
plt.boxplot(data['price'], vert=False, patch_artist=True,
boxprops=dict(facecolor='lightblue'))
plt.title('Box Plot of House Prices')
plt.xlabel('Price')
plt.show()
```




```
# Filter out outliers
data_after_filtering_ouliers = data[(data['price'] >= minimum_val) &
(data['price'] <= maximum_val)]
data_after_filtering_ouliers.head()
```

{"type":"dataframe","variable_name":"data_after_filtering_ouliers"}

```
data_after_filtering_ouliers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 20467 entries, 0 to 21612
Data columns (total 27 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              20467 non-null  int64
 1   date            20467 non-null  datetime64[ns]
 2   price           20467 non-null  float64
 3   bedrooms        20467 non-null  int64
 4   bathrooms       20467 non-null  float64
 5   sqft_living     20467 non-null  int64
 6   sqft_lot        20467 non-null  int64
 7   floors          20467 non-null  float64
 8   waterfront      20467 non-null  int64
 9   view            20467 non-null  int64
 10  condition       20467 non-null  int64
 11  grade           20467 non-null  int64
 12  sqft_above      20467 non-null  int64
```

```
 13  sqft_basement   20467 non-null  int64
 14  yr_built        20467 non-null  int64
 15  yr_renovated    20467 non-null  int64
 16  zipcode         20467 non-null  int64
 17  lat             20467 non-null  float64
 18  long            20467 non-null  float64
 19  sqft_living15   20467 non-null  int64
 20  sqft_lot15      20467 non-null  int64
 21  total_area      20467 non-null  int64
 22  property_age    20467 non-null  int64
 23  price_per_sqft  20467 non-null  float64
 24  year            20467 non-null  int32
 25  month           20467 non-null  object
 26  year_month      20467 non-null  object
dtypes: datetime64[ns](1), float64(6), int32(1), int64(17), object(2)
memory usage: 4.3+ MB
```

```
print('Checking null values if any:\
n',data_after_filtering_ouliers.isnull().sum())
```

```
Checking null values if any:
 id                0
date               0
price              0
bedrooms           0
bathrooms          0
sqft_living        0
sqft_lot           0
floors             0
waterfront         0
view               0
condition          0
grade              0
sqft_above         0
sqft_basement      0
yr_built           0
yr_renovated       0
zipcode            0
lat                0
long               0
sqft_living15      0
sqft_lot15         0
total_area         0
property_age       0
price_per_sqft     0
year               0
month              0
year_month         0
dtype: int64
```

```
print('Checking duplicates if any:\n',data.duplicated().sum())
```

```
Checking duplicates if any:
 0
```

Task 5 - Data Exploratory Data Analysis

Data Exploratory Data Analysis:

- Provide summary statistics for all numerical features.
- Analyze the distribution of the target variable (price) using visualization techniques.
- Explore relationships between house prices and key features (area, number of bedrooms, location, etc.). Create visualizations showing
- Time trends in house prices over the period covered by the dataset
- Correlation between numerical features using a heatmap, and more

```
# calculates statistics like mean, standard deviation, min, max, and percentiles
# and for categorical data, it provides counts, unique values
data_after_filtering_ouliers.describe()
```

```
{"type":"dataframe"}
```

Histogram plot

In histogram plot,it is clear that it is right-skewed distribution because it has long tail towards right side. Here mean > median > mode. Most of the houses are lower to mid range,few of them are luxurious.

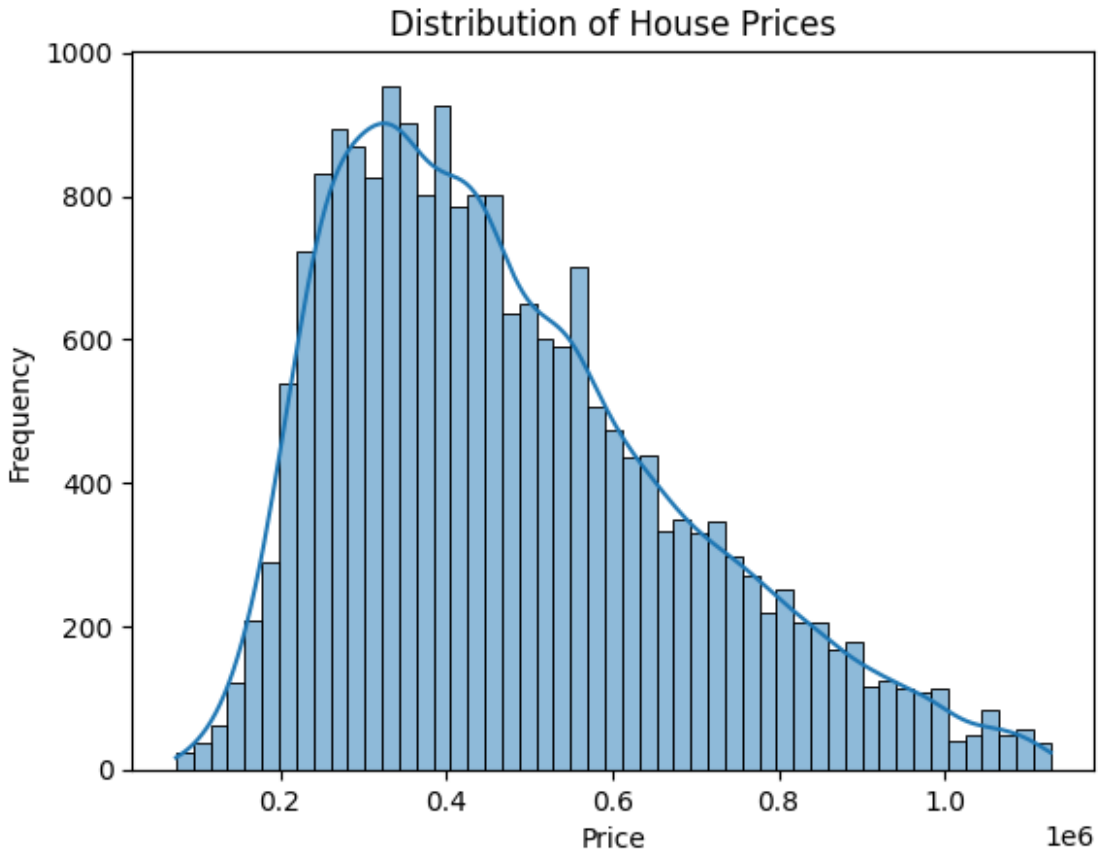
I have used AI here to describe the chart (ChatGPT)

This histogram with a density curve shows the distribution of house prices across the dataset. At first glance, it is clear that the distribution is right-skewed, meaning most homes are priced on the lower end, while fewer homes fall in the higher price range. The peak of the distribution where the frequency is highest lies between 200,000 and 400,000, indicating that the majority of homes in the dataset are within this price range.

As the price increases beyond 500,000, the number of houses decreases significantly. Very few houses are priced near or above $1 million, which is why the tail on the right side is long but thinner. The smooth curve over the bars represents the density distribution, helping to better visualize the overall shape and concentration of prices.

In summary, most houses are moderately priced, and luxury homes (higher-priced ones) are comparatively rare. This type of distribution is common in real estate markets, where the bulk of transactions happen in affordable to mid-range price segments.

```
sns.histplot(data_after_filtering_ouliers['price'], kde=True)
plt.title('Distribution of House Prices')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```



Hexbin Plot

Hexbin plot used for large dataset. It is more usefull then scatterplot.

I have used AI here to describe the chart (ChatGPT)

The hexbin plot titled "House Prices vs Living Area" provides a visual representation of the relationship between house prices and living area (in square feet) using a density-based

approach. At the lower end of the x-axis (Living Area) — starting from 0 to around 1000 sqft, we observe a concentration of house prices ranging from 100,000 to 400,000, but the density is relatively lower. As we move towards the central region (around 1000–2500 sqft), the plot becomes densely populated with dark red hexagons, indicating a high concentration of houses in this living area range. Correspondingly, prices in this range mostly fall between 200,000 and 600,000, suggesting that most properties are mid-sized and mid-priced.

Continuing further to 2500–4000 sqft, we still observe a good number of houses, but the density starts to spread out. Prices in this segment tend to increase, with values commonly in the 400,000 to 800,000 range.

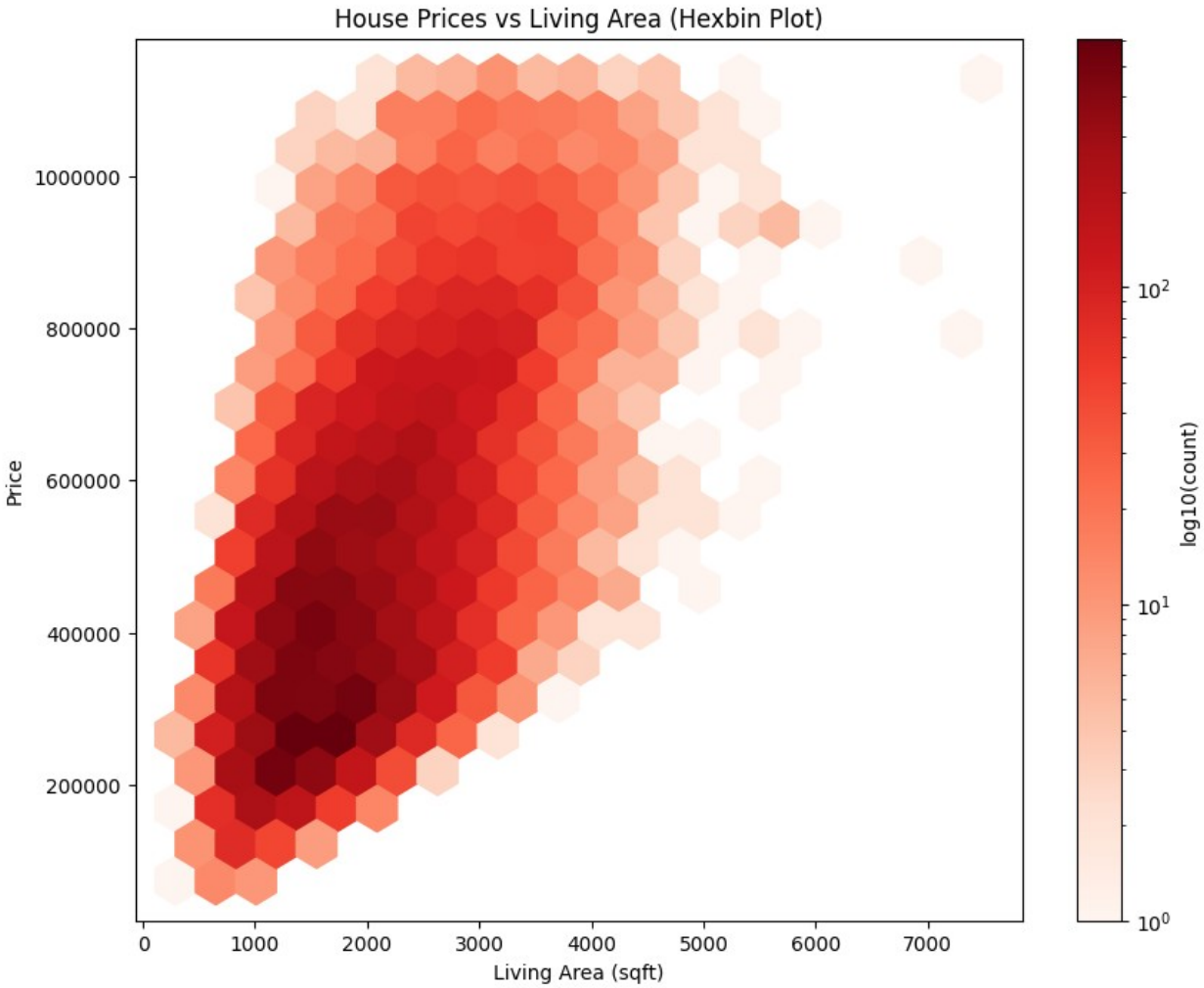
As we approach the higher end of the x-axis (above 4000 sqft), the density of hexagons becomes significantly lighter, indicating fewer properties in this range. However, the prices keep rising, with some properties going above $1,000,000, especially those with large living areas (5000–7000 sqft) — though these are rarer.

The color bar on the right represents the logarithmic count scale, showing that darker red areas have higher counts of data points, while lighter areas represent fewer data points.

Summary:

- Most houses are concentrated in the 1000–2500 sqft range with prices between 200,000 and 600,000.
- There's a positive correlation: as living area increases, prices tend to increase.
- Larger and more expensive houses are less common.
- The plot effectively highlights market trends and property distribution, showing where the majority of properties lie and how price scales with size.

```
plt.figure(figsize=(10, 8))
plt.hexbin(data_after_filtering_ouliers['sqft_living'],
data_after_filtering_ouliers['price'], gridsize=20, cmap='Reds',
bins='log')
cb = plt.colorbar(label='log10(count)')
plt.ticklabel_format(axis='y', style='plain', useOffset=False)
plt.title('House Prices vs Living Area (Hexbin Plot)')
plt.xlabel('Living Area (sqft)')
plt.ylabel('Price')
plt.show()
```



Line Chart

line chart for showing monthly price trends from 2014 to 2015. Not using year becasue i have only 2014 and 2015 years data in data set and it will not make understable diffrence in visualization to understand pattern.

I have used AI here to describe the chart (ChatGPT)

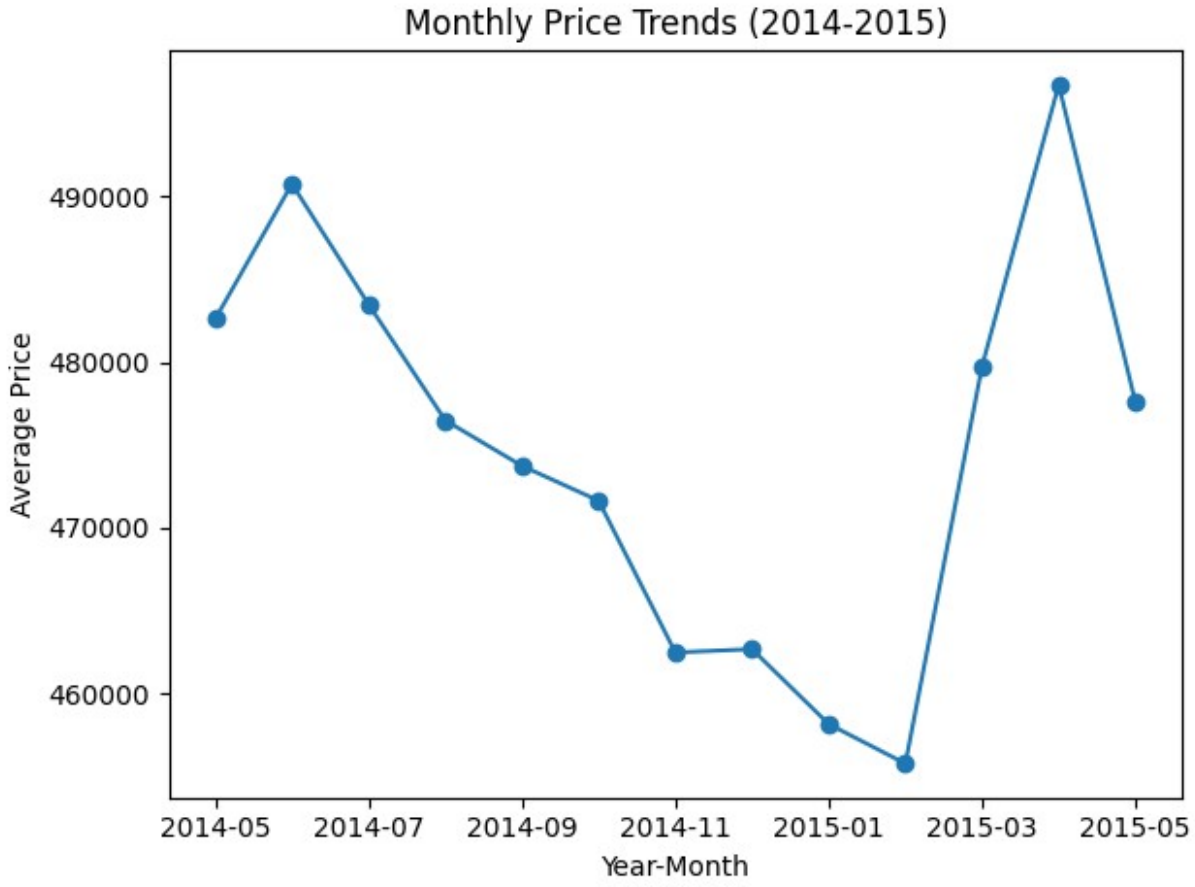
The chart "Monthly Price Trends (2014–2015)" illustrates the average price fluctuations over a 12-month period from May 2014 to April 2015. At the start in May 2014, the average price was around 483,000, which then increased sharply in June 2014, reaching a peak of over 490,000 — the highest point in the early part of the timeline. However, from July 2014, prices began to gradually decline, continuing this downward trend through the next several months.

By October 2014, the price had dropped to around 472,000, and this decline continued more steeply through November and December 2014, hitting approximately 463,000. The lowest point was observed in January 2015, where the price fell further to just above 455,000.

Following this low, there was a significant price recovery in February 2015, and by March 2015, the average price peaked again — this time even higher than the initial peak, surpassing 495,000, which marks the highest point on the entire chart. Finally, in April 2015, the price dropped slightly to around 477,000, indicating some stabilization after the sharp increase.

Overall, the chart shows an initial rise, a prolonged decline, a sharp rebound, and then a slight fall — reflecting a full cycle of market fluctuation within a year.

```
monthly_price_trend =
data_after_filtering_ouliers.groupby('year_month')['price'].mean()
monthly_price_trend.plot(kind='line', marker='o')
plt.title('Monthly Price Trends (2014-2015)')
plt.xlabel('Year-Month')
plt.ylabel('Average Price')
plt.tight_layout()
plt.show()
```



Correlation - > Heatmap

Correlation is used to see the relationship of two variable and indentify the patterns. Heatmap use pearson correlation by default. Range is (+1,-1). I do not used all columns for correaltion to make it more clear.

I have used AI here to describe the chart (ChatGPT)

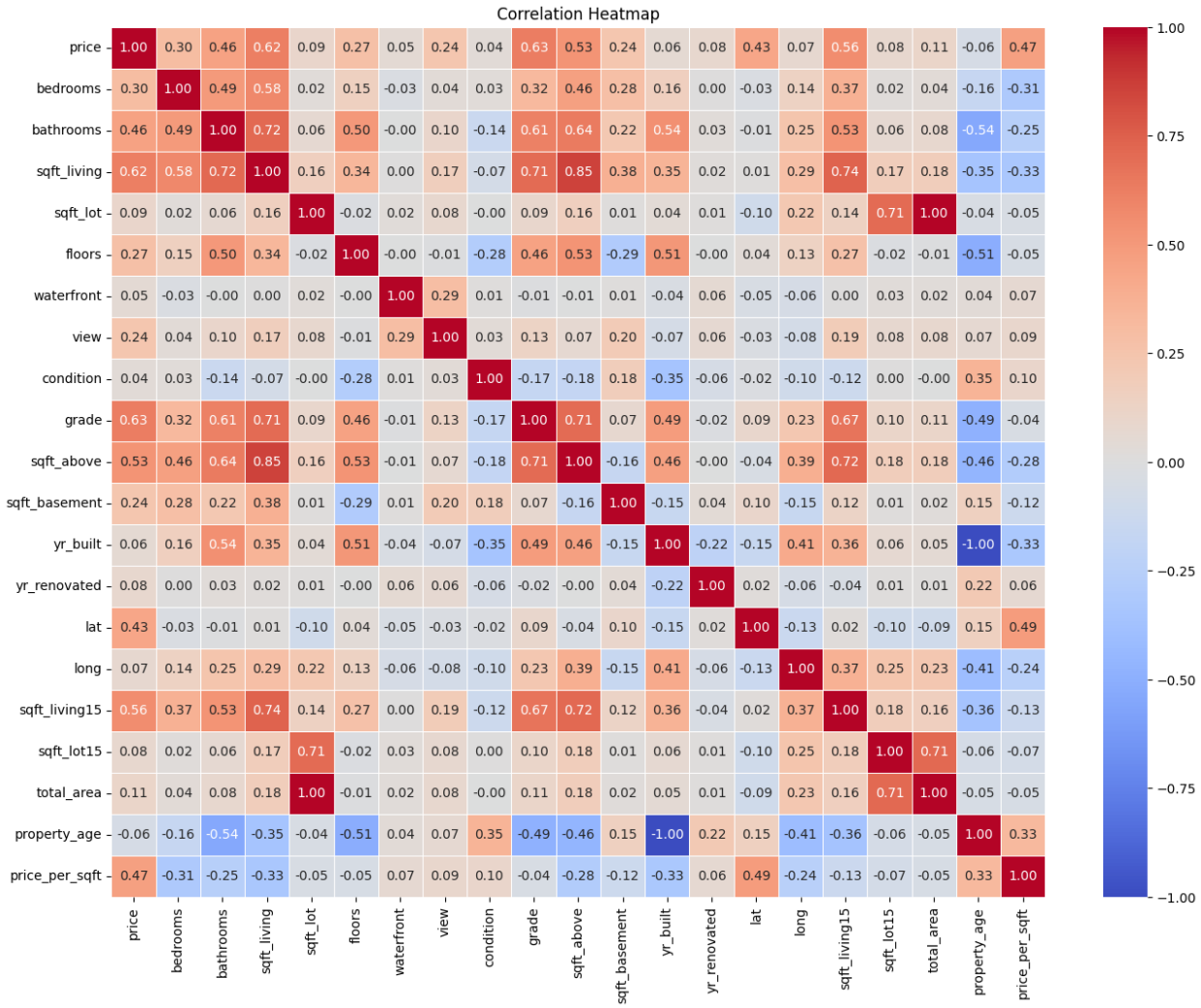
The correlation heatmap provides a visual representation of how different housing features relate to each other, particularly with the house price. The color scale ranges from dark red (strong positive correlation) to dark blue (strong negative correlation), helping to quickly identify which variables are most influential.

From the heatmap, it's evident that 'sqft_living' (living space area) and 'grade' (overall quality of construction and design) have the strongest positive correlations with price, with coefficients of 0.62 and 0.63, respectively. This means that houses with more living space or higher construction quality tend to be priced higher. Other factors like 'bathrooms' (0.46), 'sqft_above' (0.53), and 'sqft_living15' (0.56) also show moderate positive correlations, indicating their significance in influencing price.

Interestingly, 'bedrooms' has a weaker positive correlation (0.30) with price, suggesting that simply having more bedrooms does not guarantee a much higher value. On the other hand, features like 'sqft_lot' and 'condition' show very low or negligible correlations with price, implying they have little direct impact on property value.

There are also a few notable negative correlations. For example, 'property_age' has a weak negative correlation with price (-0.36), indicating that older properties tend to be slightly less expensive. Overall, the heatmap effectively highlights that house size, quality, and interior features have a greater influence on price than external or less significant attributes.

```
selected_columns = [
    'price', 'bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot',
'floors',
    'waterfront', 'view', 'condition', 'grade', 'sqft_above',
'sqft_basement',
    'yr_built', 'yr_renovated', 'lat', 'long', 'sqft_living15',
'sqft_lot15',
    'total_area', 'property_age', 'price_per_sqft'
]
corr_matrix = data_after_filtering_ouliers[selected_columns].corr()
plt.figure(figsize=(16, 12))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```



Task 7 - Basic Statistical Analysis

- Calculate the mean, median, and mode for the 'price', 'total area' (derived) columns.
- Perform a t-test to determine if there's a significant difference in price based on bedrooms and total area.

```
# Mean, median, and mode for 'price' and 'total_area'
print("Mean Price:", data_after_filtering_ouliers['price'].mean())
print("Median Price:", data_after_filtering_ouliers['price'].median())
print("Mode Price:", data_after_filtering_ouliers['price'].mode()[0])

print("Mean Total Area:",
data_after_filtering_ouliers['total_area'].mean())
print("Median Total Area:",
data_after_filtering_ouliers['total_area'].median())
print("Mode Total Area:",
data_after_filtering_ouliers['total_area'].mode()[0])
```

```
Mean Price: 476984.55943714274
Median Price: 437500.0
Mode Price: 350000.0
Mean Total Area: 16585.96633605316
Median Total Area: 9423.0
Mode Total Area: 5940
```

T-Test

Here I am using two sample t-test because I will compare between two independent group based on price like number of bedroom 1 and number of bedroom 2 or number of bedroom 3 and number of bedroom 4. I would pick one sample t-test if it is for only one group like "ages of class room 10". However, if i want to compare among 3 or more it would give us type-I error. we will make dicision base on default significance level value 0.05.

Let's say, we are testing whether the average price of houses with 2 bedrooms is different from the average price of houses with 3 bedrooms.

Null Hypothesis (H0): The mean price of houses with 2 bedrooms is equal to the mean price of houses with 3 bedrooms.

Alternative Hypothesis (H1): The mean price of houses with 2 bedrooms is not equal to the mean price of houses with 3 bedrooms.

And also, for total area.

Null Hypothesis (H0): The mean price of houses with small total area is equal to the mean price of houses with large total area.

Alternative Hypothesis (H1): The mean price of houses with small total area is not equal to the mean price of houses with large total area.

```
from scipy.stats import ttest_ind
# T-test for price based on number of bedrooms
bedroom_3 =
data_after_filtering_ouliers[data_after_filtering_ouliers['bedrooms']
== 3]['price']
bedroom_4 =
data_after_filtering_ouliers[data_after_filtering_ouliers['bedrooms']
== 4]['price']
t_stat, p_value = ttest_ind(bedroom_3, bedroom_4)
if p_value < 0.05:
    print(f"p-value: {p_value} < {0.05}. Reject the null hypothesis.
There is a significant difference in price.")
else:
```

```
    print(f"p-value: {p_value} >= {0.05}. Fail to reject the null
hypothesis. There is no significant difference in price.")
```

```
p-value: 7.919677827730361e-230 < 0.05. Reject the null hypothesis.
There is a significant difference in price.
```

```
# t-test based on price on total_area
# making two group based on median value
median_total_area =
data_after_filtering_ouliers['total_area'].median()
house_small_area =
data_after_filtering_ouliers[data_after_filtering_ouliers['total_area'
] < median_total_area]['price']
house_large_area =
data_after_filtering_ouliers[data_after_filtering_ouliers['total_area'
] >= median_total_area]['price']
t_stat, p_value = ttest_ind(house_small_area, house_large_area)
if p_value < 0.05:
    print(f"p-value: {p_value} < {0.05}. Reject the null hypothesis.
There is a significant difference in price based on total area.")
else:
    print(f"p-value: {p_value} >= {0.05}. Fail to reject the null
hypothesis. There is no significant difference in price based on total
area.")
```

```
p-value: 1.1376447010303406e-47 < 0.05. Reject the null hypothesis.
There is a significant difference in price based on total area.
```

Creativity Section

In the dataset,i have zipcode,langitude,longitude. As far i know it is possible to find out location by these three values. It comes in my mind but i do not have proper coding knowledge to make it real. Here, I have searched and collected some information and knowledge from google,AI chatbot. So, i had the idea and took help from various source to make it real.

Here, I have taken 1000 rows sample data from arround 20,500 rows data. My laptop configuration doesn't meet the requirement for huge amount of data specially when i am working with API's for extracting city and state based on zipcode. That's why i have followed this way.

```
# import requests
# from time import sleep
# import random

# random.seed(42)
```

```
# #Sample 1000 random rows
# sampled_data = data_after_filtering_ouliers.sample(n=1000,
random_state=42).copy()

# def get_city_state(zipcode):
#     zip_str = str(zipcode).zfill(5)  # Ensure 5-digit format
#     print(f"\nProcessing ZIP: {zip_str}...", end=" ")

#     try:
#         # API call with timeout
#         response = requests.get(
#             f"http://api.zippopotam.us/us/{zip_str}",
#             timeout=3
#         )

#         # Check response status
#         if response.status_code == 200:
#             data = response.json()
#             city = data['places'][0]['place name']
#             state = data['places'][0]['state']
#             print(f"Found → {city}, {state}")
#             return f"{city}, {state}"
#         elif response.status_code == 404:
#             print("ZIP code not found in database")
#             return "Not found"
#         else:
#             print(f"API error (HTTP {response.status_code})")
#             return "Error"

#     except requests.exceptions.Timeout:
#         print("Request timed out")
#         return "Timeout"
#     except Exception as e:
#         print(f"Unexpected error: {str(e)[:50]}...")
#         return "Error"

# # Add progress tracker
# total_zips = len(sampled_data)
# print(f"Starting processing of {total_zips} ZIP codes...")

# # Process with rate limiting (1 request/sec)
# results = []
# j=0
# for i, row in sampled_data.iterrows():
#     j=j+1
#     print(f"\nProgress: {j}/{total_zips}", end=" | ")
#     result = get_city_state(row['zipcode'])
#     results.append(result)
#     sleep(1)  # Rate limiting
```

```
# # Add results to DataFrame
# sampled_data['city_state'] = results

# # Show summary
# print("\n\nProcessing complete! Sample results:")
# print(sampled_data[['zipcode',
'city_state']].head().to_string(index=False))

# sampled_data.to_csv("extend_sample_data.csv", index=False)

sampled_data = pd.read_csv('/content/drive/MyDrive/Colab
Notebooks/extend_sample_data.csv')

sampled_data.head()

{"type":"dataframe","variable_name":"sampled_data"}

# getting city and state diffrent column by split function
sampled_data[['city', 'state']] =
sampled_data['city_state'].str.split(', ', expand=True)

sampled_data.head(10)

{"type":"dataframe","variable_name":"sampled_data"}

sampled_data.info() #checking data if need to handling missing or null
values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 30 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   id              1000 non-null   int64
 1   date            1000 non-null   object
 2   price           1000 non-null   float64
 3   bedrooms        1000 non-null   int64
 4   bathrooms       1000 non-null   float64
 5   sqft_living     1000 non-null   int64
 6   sqft_lot        1000 non-null   int64
 7   floors          1000 non-null   float64
 8   waterfront      1000 non-null   int64
 9   view            1000 non-null   int64
 10  condition       1000 non-null   int64
 11  grade           1000 non-null   int64
 12  sqft_above      1000 non-null   int64
 13  sqft_basement   1000 non-null   int64
 14  yr_built        1000 non-null   int64
 15  yr_renovated    1000 non-null   int64
 16  zipcode         1000 non-null   int64
 17  lat             1000 non-null   float64
```

```
 18  long            1000 non-null   float64
 19  sqft_living15   1000 non-null   int64
 20  sqft_lot15      1000 non-null   int64
 21  total_area      1000 non-null   int64
 22  property_age    1000 non-null   int64
 23  price_per_sqft  1000 non-null   float64
 24  year            1000 non-null   int64
 25  month           1000 non-null   object
 26  year_month      1000 non-null   object
 27  city_state      1000 non-null   object
 28  city            1000 non-null   object
 29  state           1000 non-null   object
dtypes: float64(6), int64(18), object(6)
memory usage: 234.5+ KB
```

Bar Chart

I have used AI here to describe the chart (ChatGPT)

The bar chart visualizes the total property prices across different cities in Washington state. The x-axis represents various cities, each labeled with a tilted orientation for better readability, while the y-axis represents the total price values in an increasing order. The chart displays bars of varying heights, indicating the relative property price totals per city. Notably, Seattle, Washington, exhibits the highest total property price, with a significantly taller bar compared to others. Other cities like Bellevue, Kirkland, Sammamish, and Issaquah also show relatively higher total prices but are much lower than Seattle. The bars are colored in light blue, making them easily distinguishable. The overall layout is spread out horizontally, emphasizing the comparison between multiple cities while maintaining clarity in representation.

```
#total prices by city
city_prices = sampled_data.groupby('city_state')['price'].sum()

plt.figure(figsize=(15,5))
city_prices.plot(kind='bar', color='skyblue')

plt.title('Cities by Total Property Prices')
plt.ylabel('Total Price')
plt.ticklabel_format(axis='y', style='plain', useOffset=False)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

Pie Chart

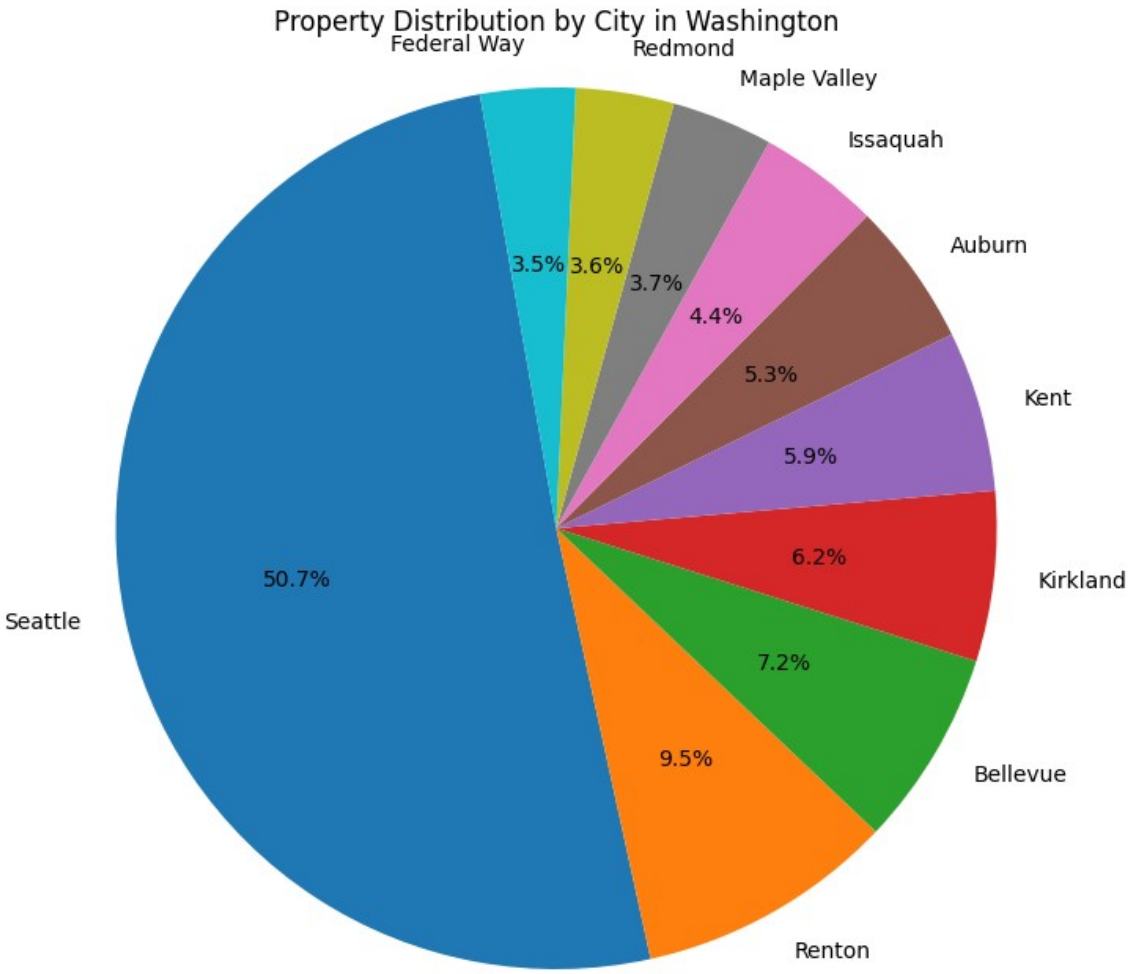
I have used AI here to describe the chart (ChatGPT)

The pie chart titled "Property Distribution by City in Washington" visually represents the percentage of properties distributed across different cities in Washington. The chart is divided into multiple colored sections, each corresponding to a city and labeled with its respective percentage of the total property distribution.

Seattle dominates the distribution with 50.7%, making up more than half of the total properties. The second-largest portion belongs to Renton (9.5%), followed by Bellevue (7.2%) and Kirkland (6.2%). The cities of Kent (5.9%), Auburn (5.3%), and Issaquah (4.4%) have slightly lower shares. Other cities, including Maple Valley (3.7%), Redmond (3.6%), and Federal Way (3.5%), hold the smallest portions of the distribution.

Each slice of the pie is distinctly colored to differentiate between the cities, and the percentages are labeled on the chart for clarity. The layout effectively highlights the significant disparity in property distribution, with Seattle having a much larger share compared to the other cities.

```
# Count properties by city
city_counts = sampled_data['city'].value_counts().nlargest(10)
plt.figure(figsize=(10, 8))
plt.pie(city_counts, labels=city_counts.index, autopct='%1.1f%%',
startangle=100)
plt.title('Property Distribution by City in Washington')
plt.axis('equal')
plt.show()
```


Property Distribution by City in Washington
Federal Way
Redmond
Maple Valley
Issaquah
Auburn
Kent
Kirkland
Bellevue
Renton
Seattle
3.5%
3.6%
3.7%
4.4%
5.3%
5.9%
6.2%
7.2%
9.5%
50.7%