MATH METHOD FOR DATA ANALYSIS
LAB 1
PAIR PROGRAMMING

Name: Shabari Vignesh
SJSU ID: 017407663

Part 1

Introduction

Coronary heart disease (CHD) remains a leading cause of morbidity and mortality globally, necessitating comprehensive studies and analyses to identify the predominant risk factors and develop effective preventative measures. In this context, we have undertaken a detailed examination of a dataset that encapsulates an array of health and behavioral attributes, aiming to unveil intricate patterns and correlations associated with the incidence of CHD. The dataset, comprising variables such as systolic blood pressure, tobacco consumption, low-density lipoprotein cholesterol (LDL), adiposity, family history of heart disease, type-A behavior, obesity, alcohol consumption, and age, serves as a rich source for extracting actionable insights.

The genesis of our analysis lies in the systematic classification of variables into distinct categories, including continuous, categorical, and nominal, laying a robust foundation for subsequent exploratory data analysis (EDA). We delve deep into the dataset's characteristics, offering a panoramic view of the distribution, variability, and central tendencies of each variable. Our exploration is amplified by a series of visual representations, illuminating the dataset's structure and facilitating an intuitive understanding of complex patterns and relationships.

Central to our investigation is the quest to unravel the intricate web of relationships between individual health metrics and the prevalence of CHD. We employ a series of visual and statistical tools to dissect these relationships, striving to extract nuanced insights that could pave the way for targeted interventions and personalized health strategies. Each variable is scrutinized, not in isolation, but as a part of an integrated ecosystem of factors that collectively influence CHD's onset and progression.

Furthermore, we expand our analysis to encompass predictive modeling, employing logistic regression to assess the collective impact of the variables on CHD occurrence. This step is instrumental in transitioning from a descriptive and exploratory analysis to a predictive paradigm, where the insights gleaned can be harnessed to forecast CHD risk and inform preventative measures.

Datasets Used

1. heart-train.csv
2. heart-test.csv

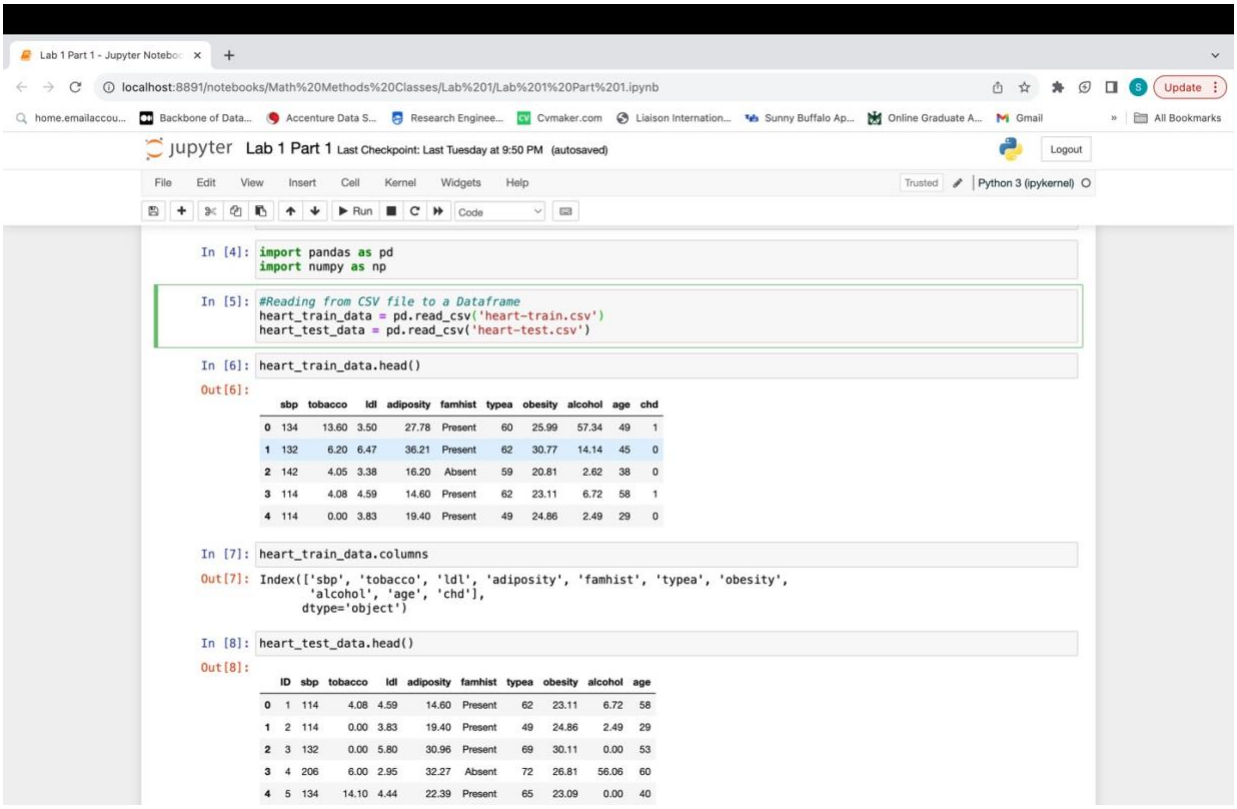
DATASET ANALYSIS

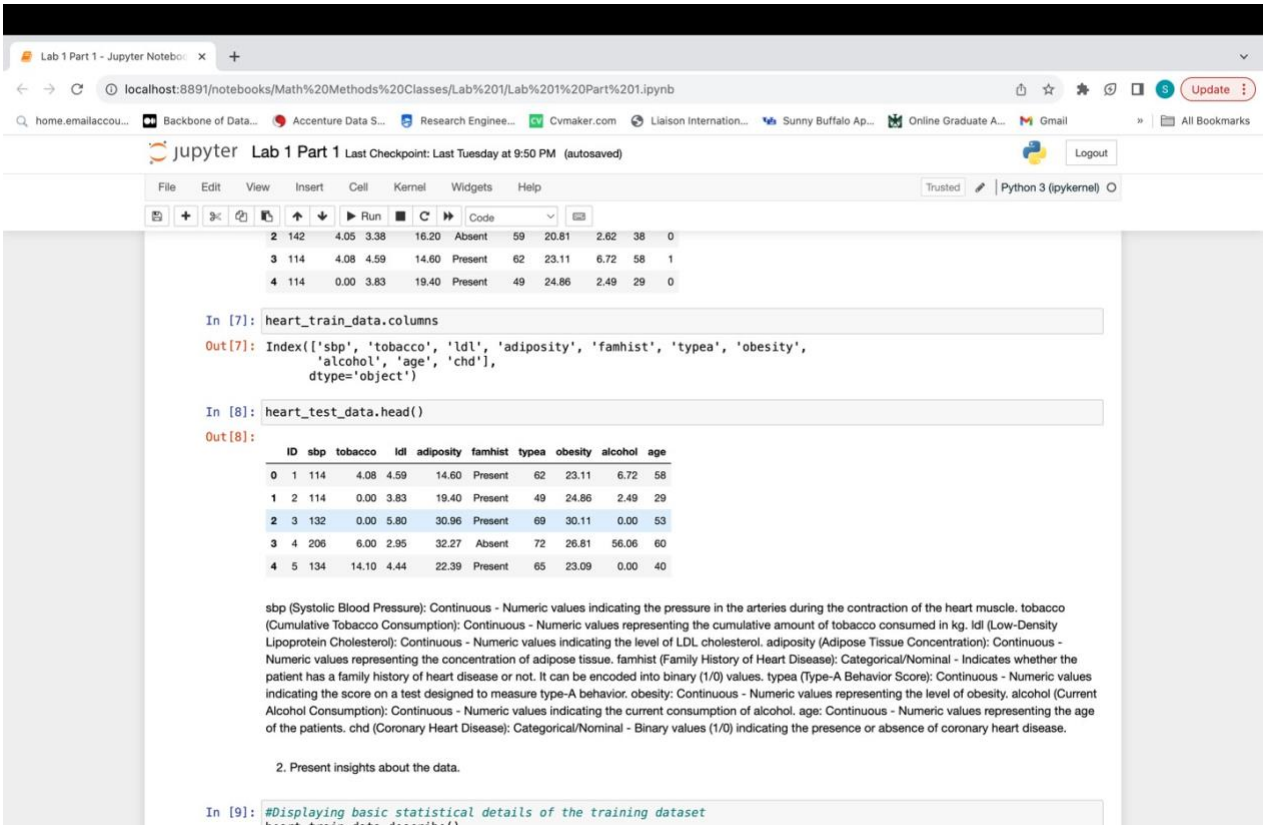
1. Identify the Dataset Columns into Nominal, Categorical, Continuous, etc. Categories.

Python Code:

```
#Importing neccessary Packages
import pandas as pd
import numpy as np
#Reading from CSV file to a Dataframe
heart_train_data = pd.read_csv('heart-train.csv')
heart_test_data = pd.read_csv('heart-test.csv')
#Checking for top 5 rows from heart-train data
heart_train_data.head()
#Checking for columns names from heart-train data
heart_train_data.columns
#Checking for top 5 rows from heart-test data
heart_test_data.head()
```

Code Execution Screenshot:





Explanation:

1. sbp (Systolic Blood Pressure): Continuous - Numeric values indicating the pressure in the arteries during the contraction of the heart muscle.
2. tobacco (Cumulative Tobacco Consumption): Continuous - Numeric values representing the cumulative amount of tobacco consumed in kg.
3. ldl (Low-Density Lipoprotein Cholesterol): Continuous - Numeric values indicating the level of LDL cholesterol.
4. adiposity (Adipose Tissue Concentration): Continuous - Numeric values representing the concentration of adipose tissue.
5. famhist (Family History of Heart Disease): Categorical/Nominal - Indicates whether the patient has a family history of heart disease or not. It can be encoded into binary (1/0) values.
6. typea (Type-A Behavior Score): Continuous - Numeric values indicating the score on a test designed to measure type-A behavior.
7. obesity: Continuous - Numeric values representing the level of obesity.
8. alcohol (Current Alcohol Consumption): Continuous - Numeric values indicating the current consumption of alcohol.
9. age: Continuous - Numeric values representing the age of the patients.
10. chd (Coronary Heart Disease): Categorical/Nominal - Binary values (1/0) indicating the presence or absence of coronary heart disease.

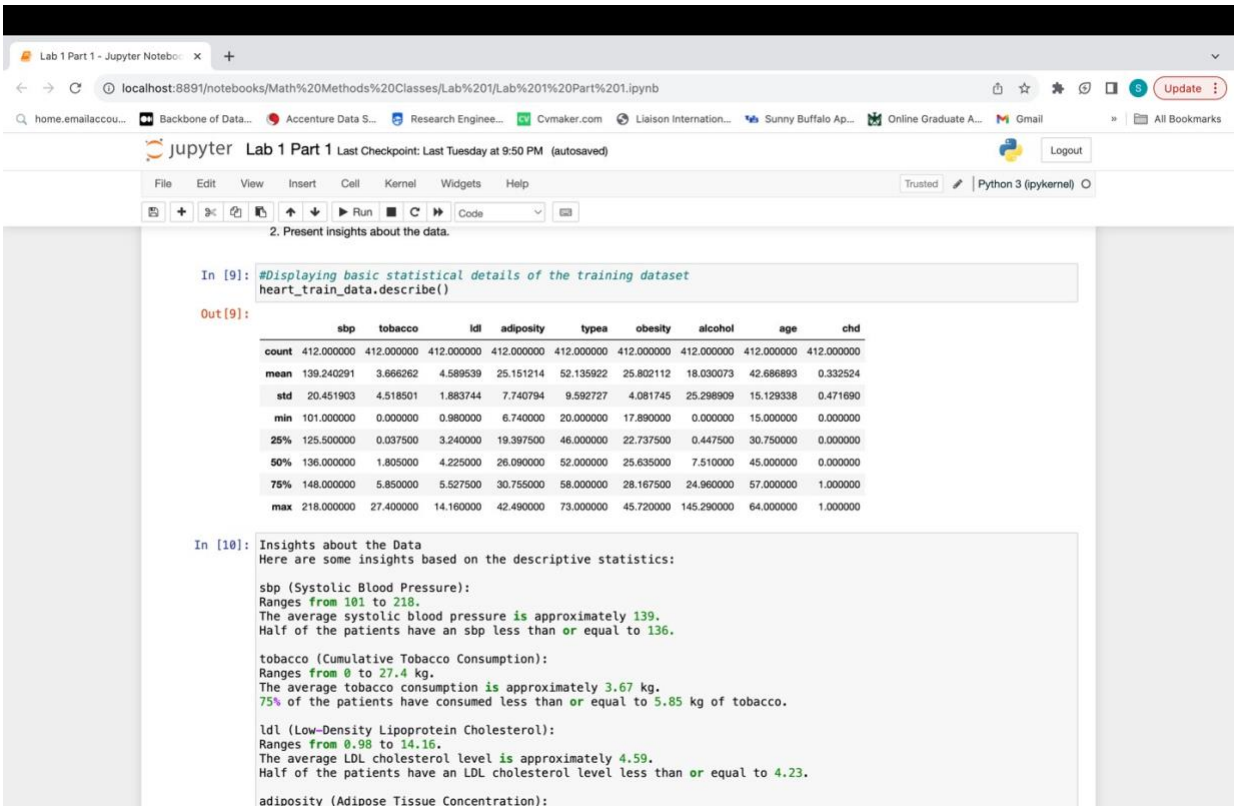
2. **Present Insights About the Data**

Python Code:

#Displaying basic statistical details of the training dataset

heart_train_data.describe()

Code Execution Screenshot:



2. Present insights about the data.

In [9]:
```
#Displaying basic statistical details of the training dataset
heart_train_data.describe()
```

Out[9]:

	sbp	tobacco	ldl	adiposity	typea	obesity	alcohol	age	chd
count	412.000000	412.000000	412.000000	412.000000	412.000000	412.000000	412.000000	412.000000	412.000000
mean	139.240291	3.666262	4.589539	25.151214	52.135922	25.802112	18.030073	42.686893	0.332524
std	20.451903	4.518501	1.883744	7.740794	9.592727	4.081745	25.298909	15.129338	0.471690
min	101.000000	0.000000	0.980000	6.740000	20.000000	17.890000	0.000000	15.000000	0.000000
25%	125.500000	0.037500	3.240000	19.397500	46.000000	22.737500	0.447500	30.750000	0.000000
50%	136.000000	1.805000	4.225000	26.090000	52.000000	25.635000	7.510000	45.000000	0.000000
75%	148.000000	5.850000	5.527500	30.755000	58.000000	28.167500	24.960000	57.000000	1.000000
max	218.000000	27.400000	14.160000	42.490000	73.000000	45.720000	145.290000	64.000000	1.000000

In [10]:
```
Insights about the Data
Here are some insights based on the descriptive statistics:

sbp (Systolic Blood Pressure):
Ranges from 101 to 218.
The average systolic blood pressure is approximately 139.
Half of the patients have an sbp less than or equal to 136.

tobacco (Cumulative Tobacco Consumption):
Ranges from 0 to 27.4 kg.
The average tobacco consumption is approximately 3.67 kg.
75% of the patients have consumed less than or equal to 5.85 kg of tobacco.

ldl (Low-Density Lipoprotein Cholesterol):
Ranges from 0.98 to 14.16.
The average LDL cholesterol level is approximately 4.59.
Half of the patients have an LDL cholesterol level less than or equal to 4.23.

adiposity (Adipose Tissue Concentration):
```

Explanation:

- The dataset contains a mix of continuous and categorical data types.
- There are health-related measurements like systolic blood pressure, LDL cholesterol, and adiposity, which are continuous and may require statistical analysis to understand their distributions and relationships with the outcome variable (chd).
- There are lifestyle-related variables like tobacco and alcohol consumption, which are also continuous and might have a significant impact on heart disease onset.
- The binary categorical variable, chd, is the outcome variable we are interested in predicting.

Insights about the Data Here are some insights based on the descriptive statistics:

1. sbp (Systolic Blood Pressure): Ranges from 101 to 218. The average systolic blood pressure is approximately 139. Half of the patients have a sbp less than or equal to 136.
2. tobacco (Cumulative Tobacco Consumption): Ranges from 0 to 27.4 kg. The average tobacco consumption is approximately 3.67 kg. 75% of the patients have consumed less than or equal to 5.85 kg of tobacco.
3. ldl (Low-Density Lipoprotein Cholesterol): Ranges from 0.98 to 14.16. The average LDL cholesterol level is approximately 4.59. Half of the patients have an LDL cholesterol level less than or equal to 4.23.
4. adiposity (Adipose Tissue Concentration): Ranges from 6.74 to 42.49. The average adiposity is approximately 25.15. Half of the patients have adiposity less than or equal to 26.09.

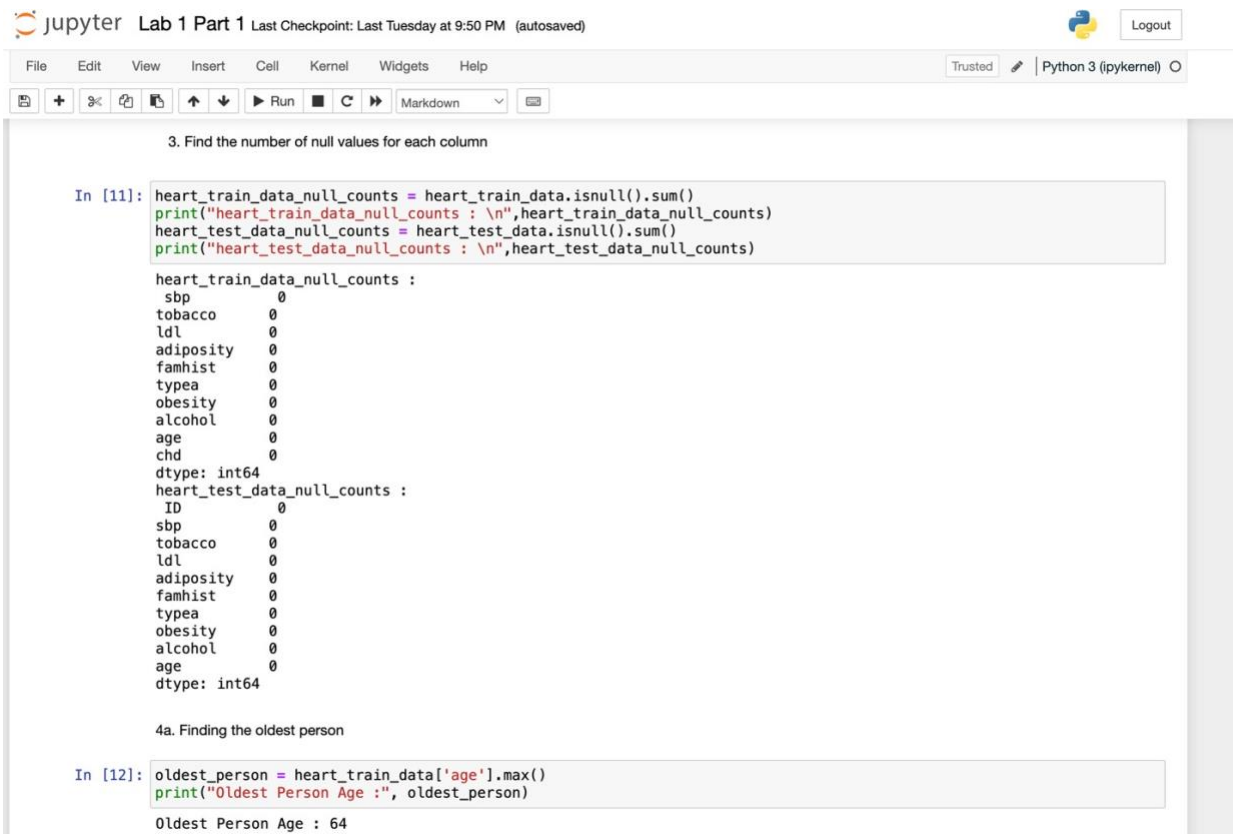
5. typea (Type-A Behavior Score): Ranges from 20 to 73. The average score is approximately 52. Half of the patients have a score less than or equal to 52.
6. obesity: Ranges from 17.89 to 45.72. The average obesity level is approximately 25.8. Half of the patients have an obesity level less than or equal to 25.64.
7. alcohol (Current Alcohol Consumption): Ranges from 0 to 145.29. The average alcohol consumption is approximately 18.03. Half of the patients consume alcohol less than or equal to 7.51.
8. age: Ranges from 15 to 64 years. The average age is approximately 42.69 years. Half of the patients are less than or equal to 45 years old.
9. chd (coronary heart disease): Binary variable with values 0 or 1. Approximately 33% of the patients have coronary heart disease.

3. Find the number of null values for each column.

Python Code:

```
heart_train_data_null_counts = heart_train_data.isnull().sum()
print("heart_train_data_null_counts : \n",heart_train_data_null_counts)
heart_test_data_null_counts = heart_test_data.isnull().sum()
print("heart_test_data_null_counts : \n",heart_test_data_null_counts)
```

Code Execution Screenshot:



Explanation:

There are no null values in any of the columns in the training dataset. Each column is fully populated with data.

4. Know about the patients:

a. Find the oldest person
Python Code:
```
oldest_person = heart_train_data['age'].max()
print("Oldest Person Age :", oldest_person)
```

Code Execution Screenshot:

Explanation:
Oldest Person: The oldest person in the training dataset is **64** years old.

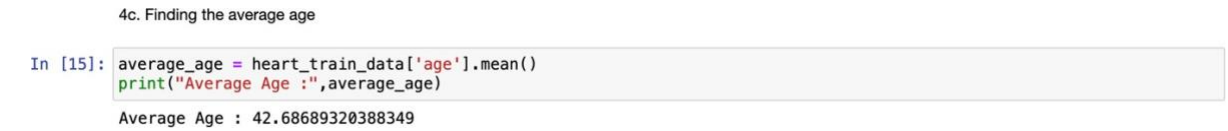
b. Find the youngest person
Python Code:
```
youngest_person = heart_train_data['age'].min()
print("Youngest Person Age :",youngest_person)
```

Code Execution Screenshot:

Explanation:
Youngest Person: The youngest person in the training dataset is **15** years old.

c. Find the average age group
Python Code:
```
average_age = heart_train_data['age'].mean()
print("Average Age :",average_age)
```

Code Execution Screenshot:

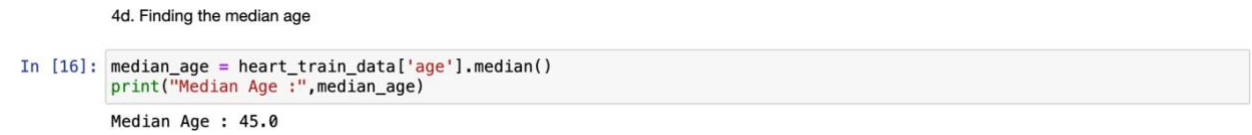


Explanation:
Average Age: The average age of the patients in the training dataset is approximately **42.69** years.

d. Find median age
Python Code:

```
median_age = heart_train_data['age'].median()
print("Median Age :",median_age)
```

Code Execution Screenshot:

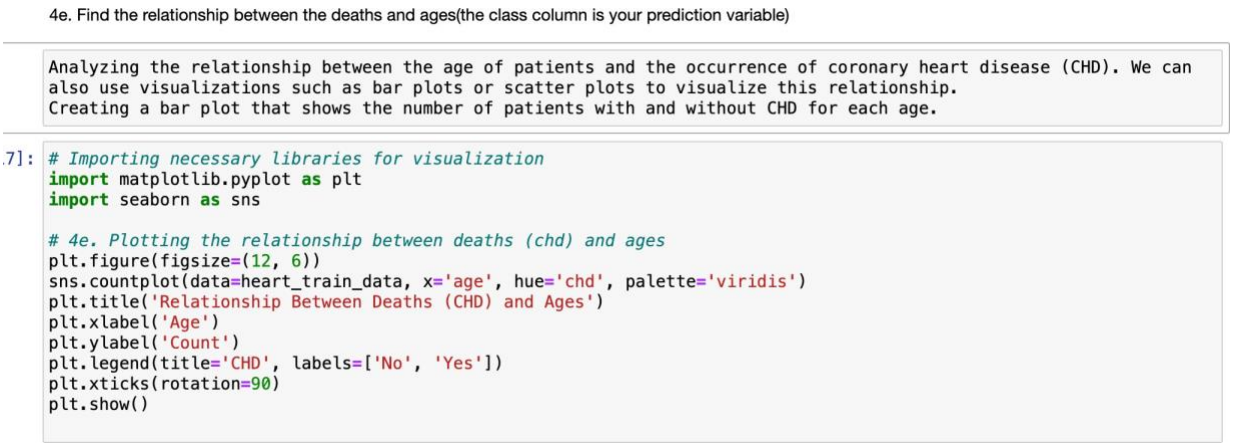


Explanation:
Median Age: The median age of the patients in the training dataset is **45** years.

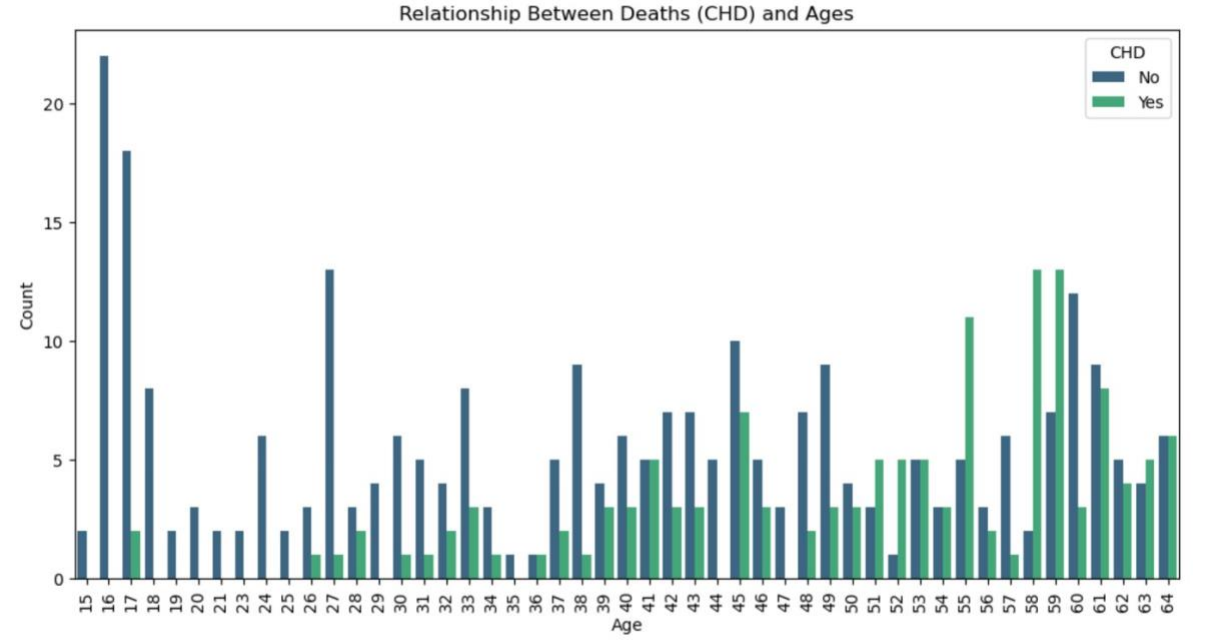
e. Find the relationship between the deaths and ages (the class column is your prediction variable)
Python Code:

```
# Importing necessary libraries for visualization
import matplotlib.pyplot as plt
import seaborn as sns

# 4e. Plotting the relationship between deaths (chd) and ages
plt.figure(figsize=(12, 6))
sns.countplot(data=heart_train_data, x='age', hue='chd', palette='viridis')
plt.title('Relationship Between Deaths (CHD) and Ages')
plt.xlabel('Age')
plt.ylabel('Count')
plt.legend(title='CHD', labels=['No', 'Yes'])
plt.xticks(rotation=90)
plt.show()
```

Code Execution Screenshot:



```
plt.show()
```



Relationship Between Deaths and Ages From the bar plot, we can observe the following:

The number of patients with CHD (Coronary Heart Disease) generally increases with age. Younger patients, particularly those under the age of 30, have a lower occurrence of CHD. Starting from around age 40, there is a noticeable increase in the number of patients with CHD.

Explanation:

From the plot above, we can observe several insights:

- The number of patients with CHD (Coronary Heart Disease) generally increases with age.
- Younger patients, particularly those under the age of 30, have a lower occurrence of CHD have very few or no CHD cases.
- Starting from around age 40, there is a noticeable increase in the number of patients with CHD.
- There is a noticeable increase in CHD cases for individuals aged around 50 and above.
- Young adults and teenagers (below 30 years of age).

f. Find the age groups whose survival rate is the largest

Python Code:

```
# Calculating the survival rate for each age
age_survival_rate = heart_train_data[heart_train_data['chd'] == 0]
.groupby('age').size() / heart_train_data.groupby('age').size()

# Sorting to get the age with the highest survival rate
age_survival_rate_sorted = age_survival_rate.sort_values(ascending=False)

# Displaying the age group with the highest survival rate
age_survival_rate_sorted.head()
```

Code Execution Screenshot:

4f. Finding the age groups with the largest survival rate

```
# Calculating the survival rate for each age
age_survival_rate = heart_train_data[heart_train_data['chd'] == 0]
.groupby('age').size() / heart_train_data.groupby('age').size()

# Sorting to get the age with the highest survival rate
age_survival_rate_sorted = age_survival_rate.sort_values(ascending=False)

# Displaying the age group with the highest survival rate
age_survival_rate_sorted.head()
```

```
age
15    1.0
35    1.0
18    1.0
19    1.0
20    1.0
dtype: float64
```

The age group 15-20 have the highest survival rate of 100% (1.0), meaning all individuals in this age group in the dataset do not have coronary heart disease (CHD). Also, it is observed that people in the age of 35 do not suffer from the disease.

Explanation:
The age groups with the highest survival rate (100% survival, i.e., no CHD cases) are primarily the younger age groups, specifically ages 15, 18, 19, 20, and 35. This finding aligns with the earlier observation that CHD is less common among younger individuals.

g. Find similar relationships for at least 3-4 columns that you think can play a role in prediction.

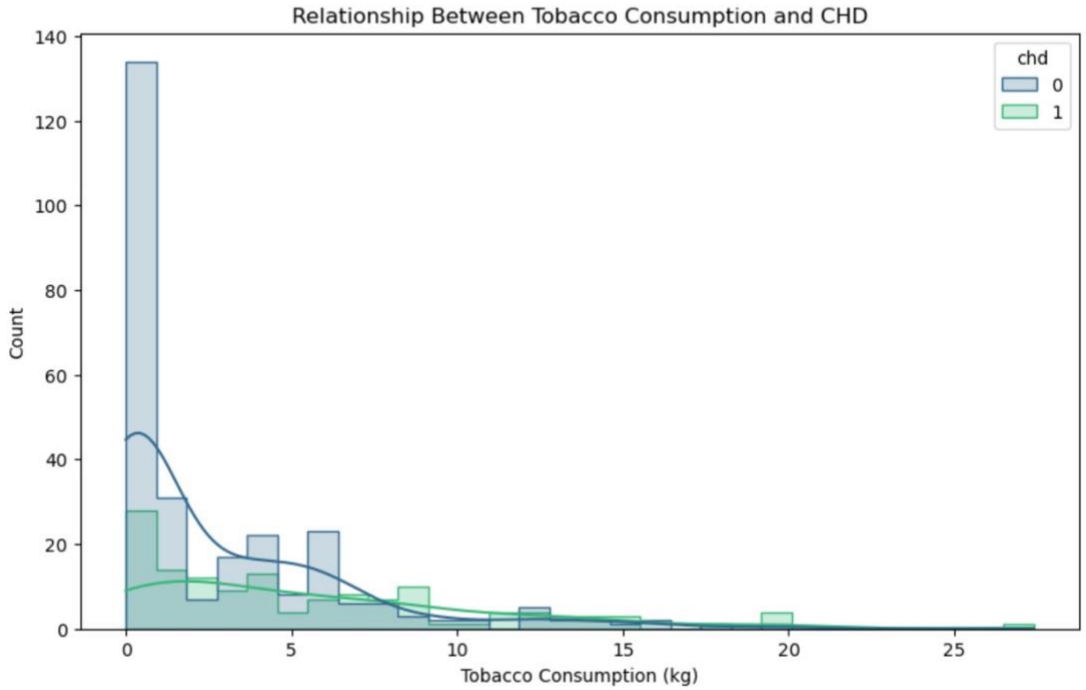
4g 1 Plotting the relationship between tobacco consumption and CHD using histogram.
Python Code:

```
plt.figure(figsize=(10,6))
sns.histplot(data=heart_train_data, x='tobacco', hue='chd', element='step', common_norm=False, kde=True, bins=30, palette='viridis')
plt.title('Relationship Between Tobacco Consumption and CHD')
plt.xlabel('Tobacco Consumption (kg)')
plt.ylabel('Count')
plt.show()
```

Code Execution Screenshot:

4g 1 Plotting the relationship between tobacco consumption and CHD using histogram

```
In [44]: plt.figure(figsize=(10,6))
         sns.histplot(data=heart_train_data, x='tobacco', hue='chd', element='step', common_norm=Fals
                      bins=30, palette='viridis')
         plt.title('Relationship Between Tobacco Consumption and CHD')
         plt.xlabel('Tobacco Consumption (kg)')
         plt.ylabel('Count')
         plt.show()
```



A histogram was created to visualize the relationship between tobacco consumption and CHD. It seems like higher tobacco consumption is associated with an increased number of CHD cases.

Insights:
A histogram was created to visualize the relationship between tobacco consumption and CHD. It seems like higher tobacco consumption is associated with an increased number of CHD cases.

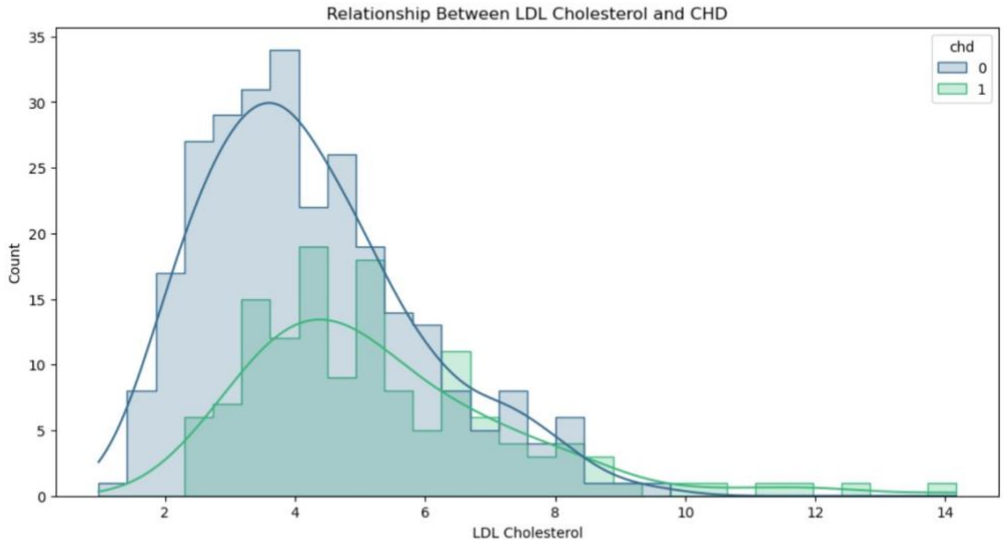
4g.2 Plotting the relationship between LDL cholesterol and CHD
Python Code:

```
plt.figure(figsize=(12,6))
sns.histplot(data=heart_train_data, x='ldl', hue='chd', element='step', common_norm=False,
kde=True, bins=30, palette='viridis')
plt.title('Relationship Between LDL Cholesterol and CHD')
plt.xlabel('LDL Cholesterol')
plt.ylabel('Count')
plt.show()
```

Code Execution Screenshot:

```
4g.2 Plotting the relationship between LDL cholesterol and CHD
```

```
In [22]: plt.figure(figsize=(12,6))
sns.histplot(data=heart_train_data, x='ldl', hue='chd', element='step', common_norm=False, k
             bins=30, palette='viridis')
plt.title('Relationship Between LDL Cholesterol and CHD')
plt.xlabel('LDL Cholesterol')
plt.ylabel('Count')
plt.show()
```



From the histogram:

1. Patients with higher levels of LDL cholesterol generally appear to have a higher occurrence of CHD.
2. There's a noticeable increase in CHD cases when LDL cholesterol levels are above approximately 4.

Insights:

From the histogram:

1.Patients with higher levels of LDL cholesterol generally appear to have a higher occurrence of CHD.

2.There's a noticeable increase in CHD cases when LDL cholesterol levels are above approximately 4.

4g.3 Plotting the relationship between adiposity and CHD

Python Code:

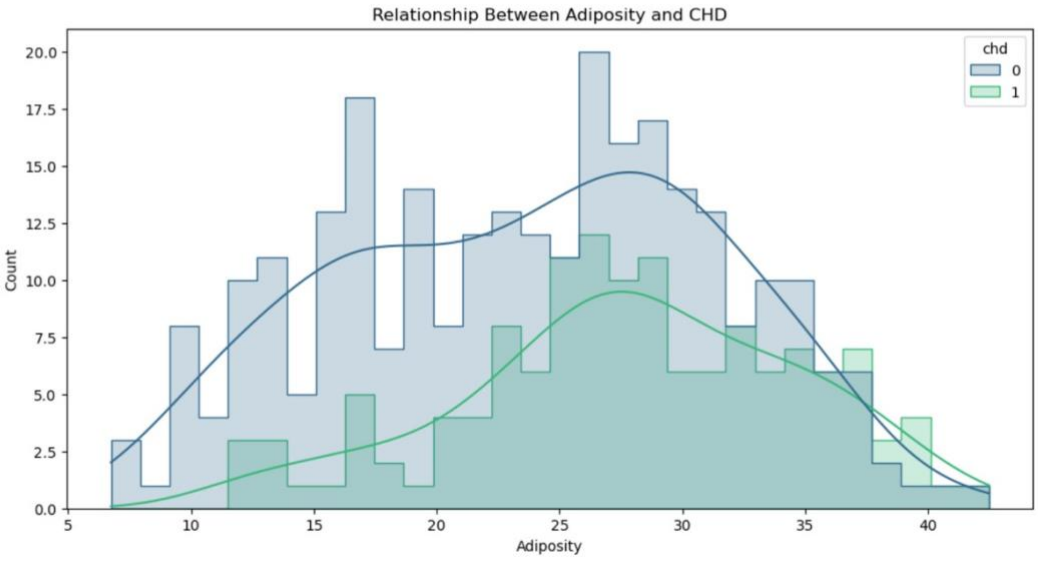
```
plt.figure(figsize=(12,6))
sns.histplot(data=heart_train_data, x='adiposity', hue='chd', element='step',
common_norm=False, kde=True, bins=30, palette='viridis')
plt.title('Relationship Between Adiposity and CHD')
plt.xlabel('Adiposity')
plt.ylabel('Count')
```

plt.show()

Code Execution Screenshot:

4g.3 Plotting the relationship between adiposity and CHD

```
In [23]: plt.figure(figsize=(12,6))
         sns.histplot(data=heart_train_data, x='adiposity', hue='chd', element='step', common_norm=Fa
                      bins=30, palette='viridis')
         plt.title('Relationship Between Adiposity and CHD')
         plt.xlabel('Adiposity')
         plt.ylabel('Count')
         plt.show()
```



1. There seems to be an increase in CHD cases with higher levels of adiposity.
2. The trend is not as clear-cut, but generally, patients with adiposity levels above approximately 25 appear to have a higher occurrence of CHD.

Insights:

1. There seems to be an increase in CHD cases with higher levels of adiposity.
2. The trend is not as clear-cut, but generally, patients with adiposity levels above approximately 25 appear to have a higher occurrence of CHD.

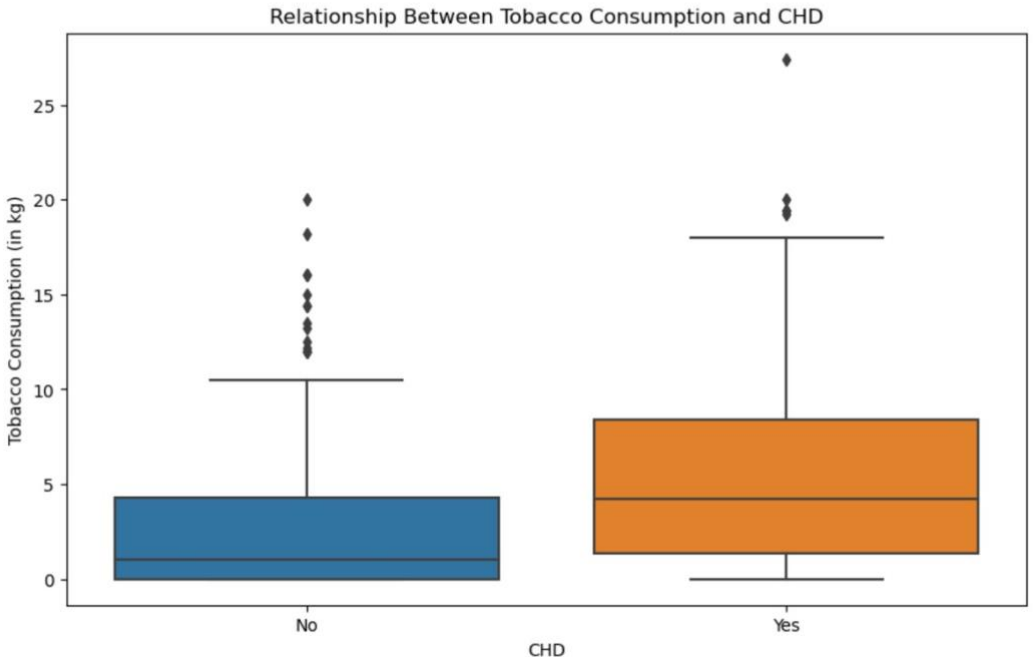
4g. Finding similar relationships for other columns using box plot

1. **Plotting the relationship between tobacco consumption and CHD using box plot**
   Python Code:
   # Plotting the relationship between tobacco consumption and CHD using box plot
   plt.figure(figsize=(10, 6))
   sns.boxplot(data=heart_train_data, x='chd', y='tobacco')
   plt.title('Relationship Between Tobacco Consumption and CHD')
   plt.xlabel('CHD')
   plt.ylabel('Tobacco Consumption (in kg)')
   plt.xticks(ticks=[0, 1], labels=['No', 'Yes'])
   plt.show()

Code Execution Screenshot:

```
4g. Finding similar relationships for other columns using box plot
```

```
In [41]: # Plotting the relationship between tobacco consumption and CHD using box plot
plt.figure(figsize=(10, 6))
sns.boxplot(data=heart_train_data, x='chd', y='tobacco')
plt.title('Relationship Between Tobacco Consumption and CHD')
plt.xlabel('CHD')
plt.ylabel('Tobacco Consumption (in kg)')
plt.xticks(ticks=[0, 1], labels=['No', 'Yes'])
plt.show()
```



Explanation:
The code is designed to plot a visual representation of how tobacco consumption might relate to the presence or absence of coronary heart disease (CHD).

Data Input:
- The dataset being used for this plot is heart_train_data.
- This dataset presumably has at least two columns:
- chd: This column appears to be a binary flag indicating whether a person has CHD (1 for 'Yes' and 0 for 'No').
- tobacco: This column signifies the tobacco consumption quantity, presumably in kilograms.

Visualization Type:
The visualization selected is a boxplot (sns.boxplot). Boxplots are useful for showing the spread and central tendency of the data.

Figure Size:
The size of the resulting figure is 10 units wide by 6 units tall.

Boxplot Details:

- The x-axis (x='chd') represents whether the person has CHD or not.
- The y-axis (y='tobacco') indicates the amount of tobacco consumed.
- So, there will be two boxplots side by side: one representing the distribution of tobacco consumption for those without CHD and the other for those with CHD.

Title and Axis Labels:
- The plot will have a title: "Relationship Between Tobacco Consumption and CHD".
- X-axis will be labeled "CHD" and the ticks for this axis will be labeled "No" (for 0) and "Yes" (for 1), indicating the absence or presence of CHD respectively.
- Y-axis will be labeled "Tobacco Consumption (in kg)".

Visualization Interpretation:
No CHD:
- The blue boxplot represents the distribution of tobacco consumption for individuals without CHD.
- The median tobacco consumption for this group is very close to 0, indicating that a significant number of individuals in this category consume little to no tobacco.
- The box's height is relatively short, implying a smaller interquartile range and, consequently, less variability in tobacco consumption for this group.
- There are several outliers, which are individuals who consume tobacco significantly more than the typical individual without CHD.

Yes CHD:
- The orange boxplot represents the distribution of tobacco consumption for individuals with CHD.
- The median tobacco consumption for this group is notably higher than the 'No CHD' group, suggesting that, on average, individuals with CHD tend to consume more tobacco.
- The box's height is much larger, showing a broader spread or higher variability in tobacco consumption among those with CHD.
- There are a few outliers above the box, suggesting there are individuals with CHD who have an exceptionally high tobacco consumption compared to their peers.

Comparative Insights:
- Tobacco consumption is generally higher among individuals with CHD compared to those without. This suggests a potential association or correlation between tobacco consumption and the presence of CHD.
- There's a wider range of tobacco consumption values in the CHD-positive group, suggesting more variability in tobacco use among those individuals.

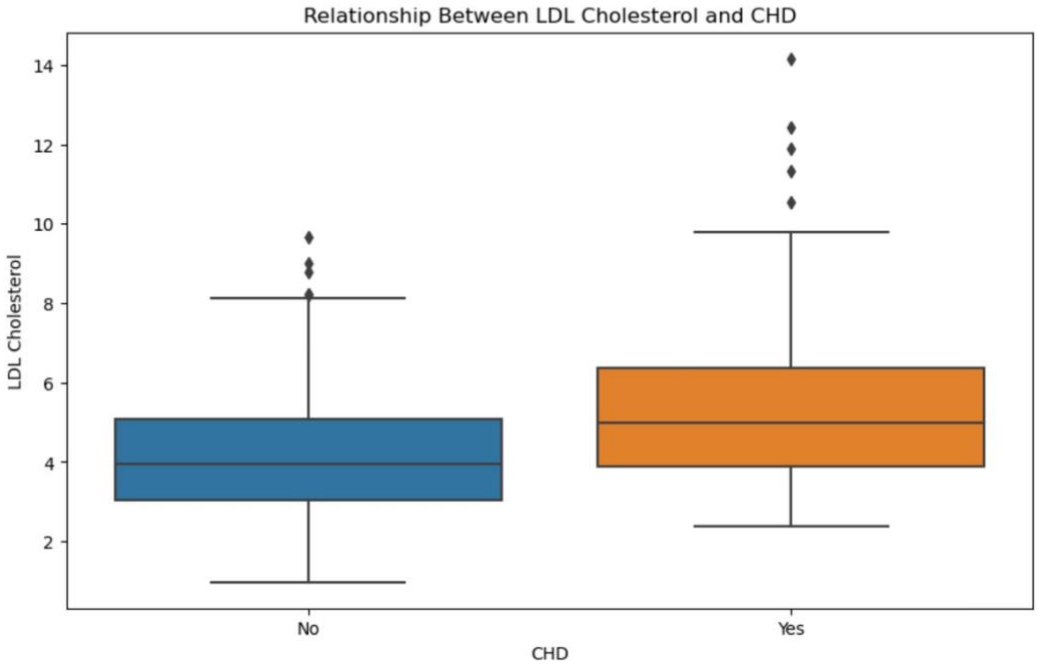
2. **Plotting the relationship between LDL cholesterol and CHD using box plot**
   Python Code:

```
# Plotting the relationship between LDL cholesterol and CHD using box plot
plt.figure(figsize=(10, 6))
sns.boxplot(data=heart_train_data, x='chd', y='ldl')
plt.title('Relationship Between LDL Cholesterol and CHD')
plt.xlabel('CHD')
```

plt.ylabel('LDL Cholesterol')
plt.xticks(ticks=[0, 1], labels=['No', 'Yes'])
plt.show()

Code Execution Screenshot:

```
In [39]: # Plotting the relationship between LDL cholesterol and CHD using box plot
         plt.figure(figsize=(10, 6))
         sns.boxplot(data=heart_train_data, x='chd', y='ldl')
         plt.title('Relationship Between LDL Cholesterol and CHD')
         plt.xlabel('CHD')
         plt.ylabel('LDL Cholesterol')
         plt.xticks(ticks=[0, 1], labels=['No', 'Yes'])
         plt.show()
```



Explanation:
The code aims to visualize the distribution of LDL (Low-Density Lipoprotein) cholesterol levels across two groups: those with CHD (Coronary Heart Disease) and those without.

Components & Functionality:
Initialization:
plt.figure(figsize=(10, 6)): This initializes a new figure for the plot, specifying its size to be 10 units wide and 6 units tall for clear visualization.

Plotting:
1. sns.boxplot(data=heart_train_data, x='chd', y='ldl'): This uses the Seaborn library's boxplot function to create a box plot. The data is sourced from heart_train_data.
2. The x axis represents the 'chd' column (indicating whether an individual has CHD or not), while the y axis represents the 'ldl' column (indicating the LDL cholesterol level).

Styling & Labeling:

1. plt.title('Relationship Between LDL Cholesterol and CHD'): Sets the title of the plot to provide a clear indication of its intent.
2. plt.xlabel('CHD') & plt.ylabel('LDL Cholesterol'): Labels the x and y axes respectively for clarity.
3. plt.xticks(ticks=[0, 1], labels=['No', 'Yes']): This specifies the labels on the x-axis. Instead of displaying 0 and 1 (which might represent 'No CHD' and 'Has CHD'), it sets more descriptive labels: 'No' and 'Yes'.

Displaying the Plot:

plt.show(): This command displays the plot. It's the final step that brings up the visualization based on all prior specifications.

Expected Output:

It is box plot visualization.

On the x-axis, there will be two categories: 'No' (for no CHD) and 'Yes' (for individuals with CHD). The y-axis will show the distribution of LDL cholesterol levels. The visualization will offer insights into the distribution, median, and outliers of LDL cholesterol for both groups.

Visualization Interpretation:

No CHD:

The blue boxplot represents the LDL cholesterol levels distribution for individuals without CHD.

The median LDL cholesterol level for this group is around 6 (assuming the unit is not mentioned but could be mmol/L or another common measurement for cholesterol).

The box's height indicates the interquartile range (IQR) and shows that the middle 50% of individuals without CHD have LDL cholesterol levels between approximately 4.5 and 7.5.

There are a few outliers above the box, suggesting there are individuals without CHD who have notably higher LDL cholesterol levels than most in this group.

Yes CHD:

The orange boxplot represents LDL cholesterol levels distribution for individuals with CHD.

The median LDL cholesterol for this group is slightly higher than the 'No CHD' group, hovering around 8.

The IQR for this group is narrower, showing LDL cholesterol levels between roughly 7 and 9 for the middle 50% of the CHD-positive individuals.

Several outliers are also present in this group, indicating individuals with CHD who have exceptionally high LDL cholesterol levels compared to their peers.

Comparative Insights:

Overall, the LDL cholesterol levels are generally higher among individuals with CHD compared to those without. This visual correlation suggests a potential association between increased LDL cholesterol levels and the presence of CHD.

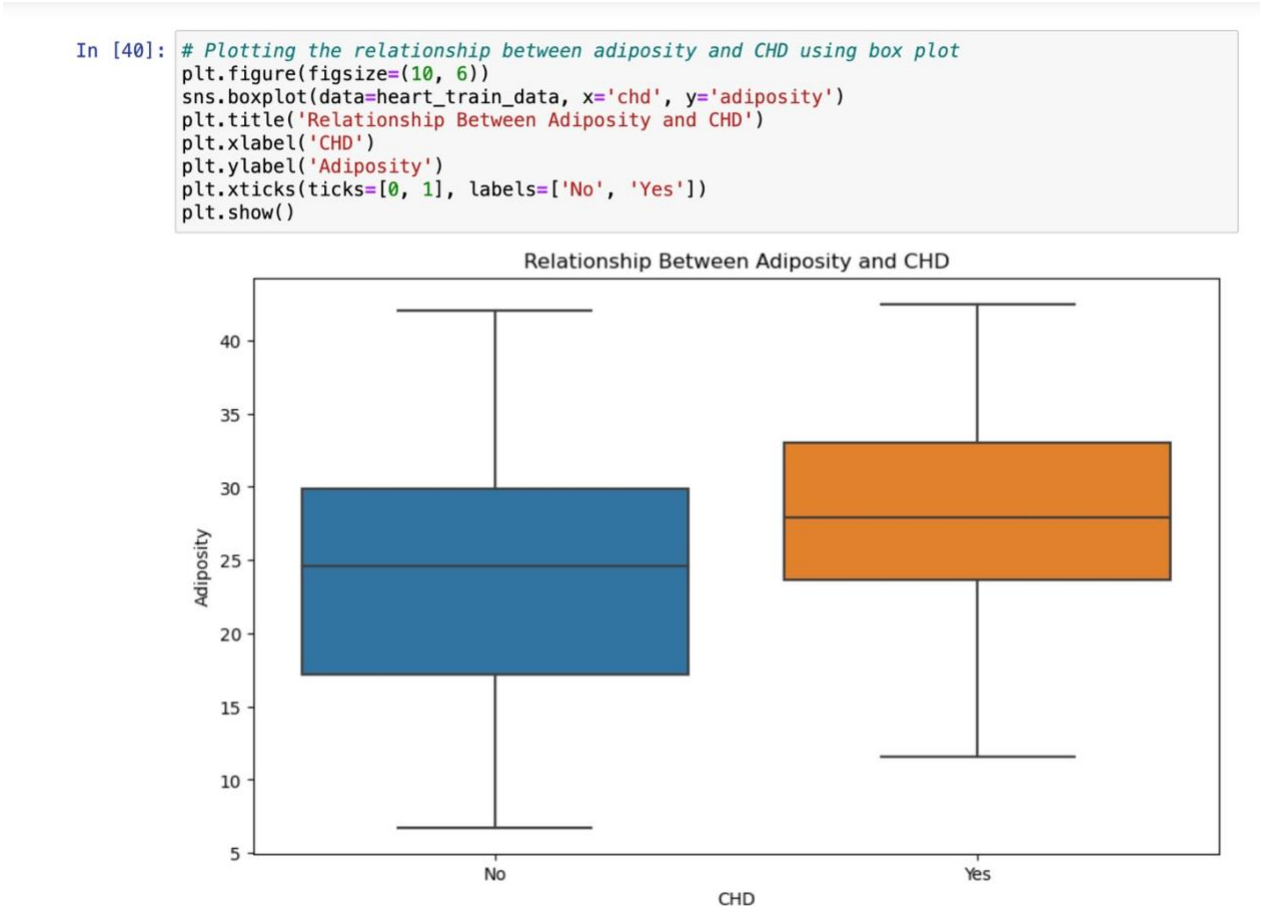
The 'Yes CHD' group has a more consistent or less varied distribution of LDL cholesterol levels, as indicated by its narrower box or IQR.

3.Plotting the relationship between adiposity and CHD using box plot

Python Code:

```
# Plotting the relationship between adiposity and CHD using box plot
plt.figure(figsize=(10, 6))
sns.boxplot(data=heart_train_data, x='chd', y='adiposity')
plt.title('Relationship Between Adiposity and CHD')
plt.xlabel('CHD')
plt.ylabel('Adiposity')
plt.xticks(ticks=[0, 1], labels=['No', 'Yes'])
plt.show()
```

Code Execution Screenshot:



Explanation:

Visualization Type: Box Plot

- **Title**: Relationship Between Adiposity and CHD (Coronary Heart Disease)
- **X-axis**: CHD (Two categories - "No" and "Yes")
- **Y-axis**: Adiposity (Numeric values ranging approximately from 5 to 40)

Observations & Insights:

Distribution for Non-CHD Group ("No" on the X-axis):

1. The blue box represents the distribution of adiposity for individuals without CHD.

2. The median adiposity level (represented by the line inside the blue box) appears to be around 27.5.
3. The interquartile range (IQR), which is the range between the 25th percentile (bottom edge of the box) and the 75th percentile (top edge of the box), spans from approximately 25 to 30. This shows the central 50% of the adiposity values for this group.
4. No significant outliers (data points outside the "whiskers" or ends of the box) are observed for this group.

Distribution for CHD Group ("Yes" on the X-axis):
1. The orange box represents the distribution of adiposity for individuals with CHD.
2. The median adiposity level for this group appears to be slightly lower, around 26.
3. The IQR for this group is tighter, ranging from about 24 to 28, indicating that adiposity levels for individuals with CHD are more closely packed around the median.
4. Similar to the non-CHD group, no significant outliers are observed.

Comparison & Key Takeaways:
1. While both groups have somewhat similar median adiposity levels, the CHD group's values are more tightly packed, as indicated by the smaller IQR.
2. The median adiposity level for the CHD group is slightly lower than that of the non-CHD group.
3. The visualization suggests that there isn't a stark difference in adiposity levels between individuals with and without CHD, at least based on the median and IQR. However, subtle differences can be observed.
4. The absence of significant outliers in both groups indicates that the data for adiposity is relatively consistent and doesn't have extreme values that deviate significantly from the rest.

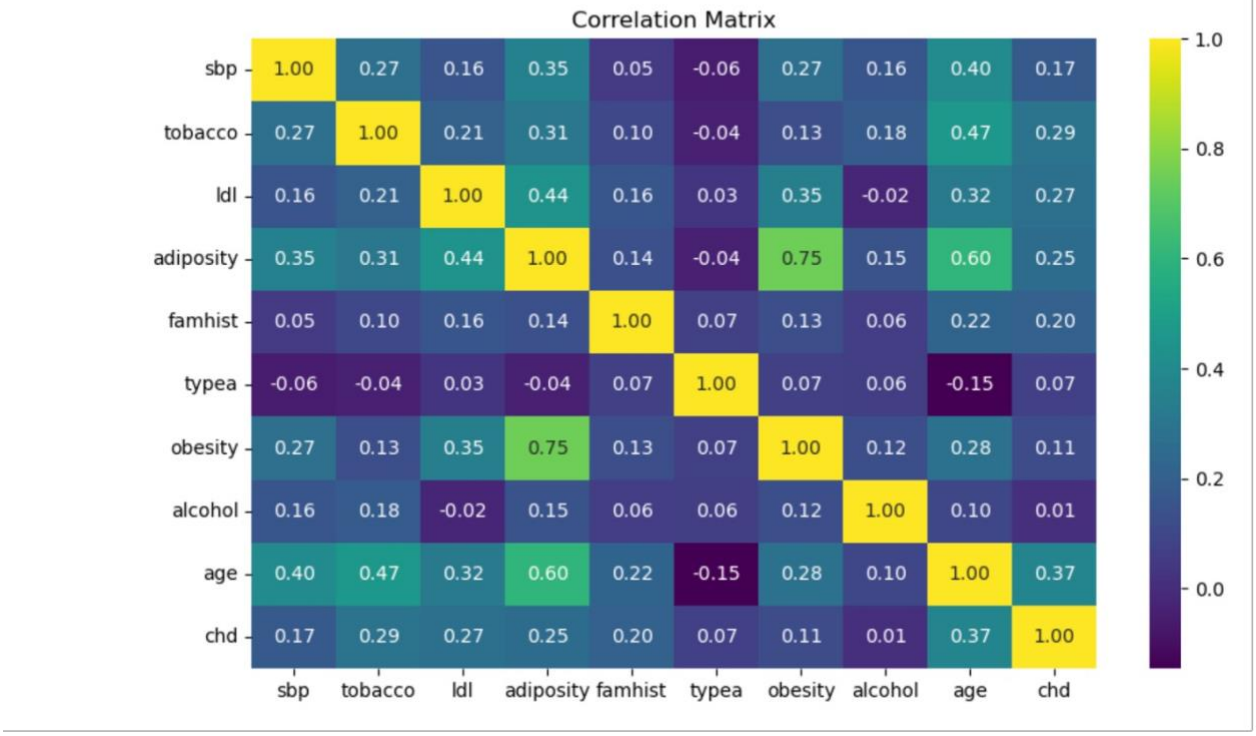
h. Get more visuals on data distributions

i.Use plotCorrelationMatrix
Python Code:

```
# Plotting the correlation matrix
plt.figure(figsize=(10,6))
sns.heatmap(heart_train_data.corr(), annot=True, cmap='viridis', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

Code Execution Screenshot:

```
In [45]: # Plotting the correlation matrix
         plt.figure(figsize=(10,6))
         sns.heatmap(heart_train_data.corr(), annot=True, cmap='viridis', fmt='.2f')
         plt.title('Correlation Matrix')
         plt.show()
```



Insights:

i. Tobacco Consumption and CHD:

Patients with CHD tend to have higher tobacco consumption. The median tobacco consumption is noticeably higher in patients with CHD compared to those without.

ii. LDL Cholesterol and CHD:

There is a clear distinction in LDL cholesterol levels between patients with and without CHD. Those with CHD generally have higher LDL cholesterol levels.

iii. Adiposity and CHD:

Similarly, patients with CHD show a trend of higher adiposity. Though the difference is not as pronounced as with tobacco consumption and LDL cholesterol, it still indicates a potential relationship.

From the correlation matrix, we observe that:

1. age has a moderate positive correlation with chd, indicating that older patients are more likely to have coronary heart disease.
2. tobacco and ldl also show a positive correlation with chd.
3. The rest of the variables have relatively weak correlations with chd.
4. The correlation matrix heatmap provides insights into the linear relationships between different variables:
5. Age and CHD: There is a moderate positive correlation (0.37), indicating that older individuals tend to have a higher incidence of CHD.

6. Tobacco and CHD: Tobacco consumption also has a positive correlation (0.25) with CHD, suggesting that higher tobacco consumption is associated with an increased risk of CHD.
7. LDL and CHD: LDL cholesterol has a positive correlation (0.18) with CHD, indicating that higher LDL levels might be associated with an increased risk of CHD.
8. Other variables also exhibit correlations amongst themselves, but they are not as strong as the ones mentioned above.

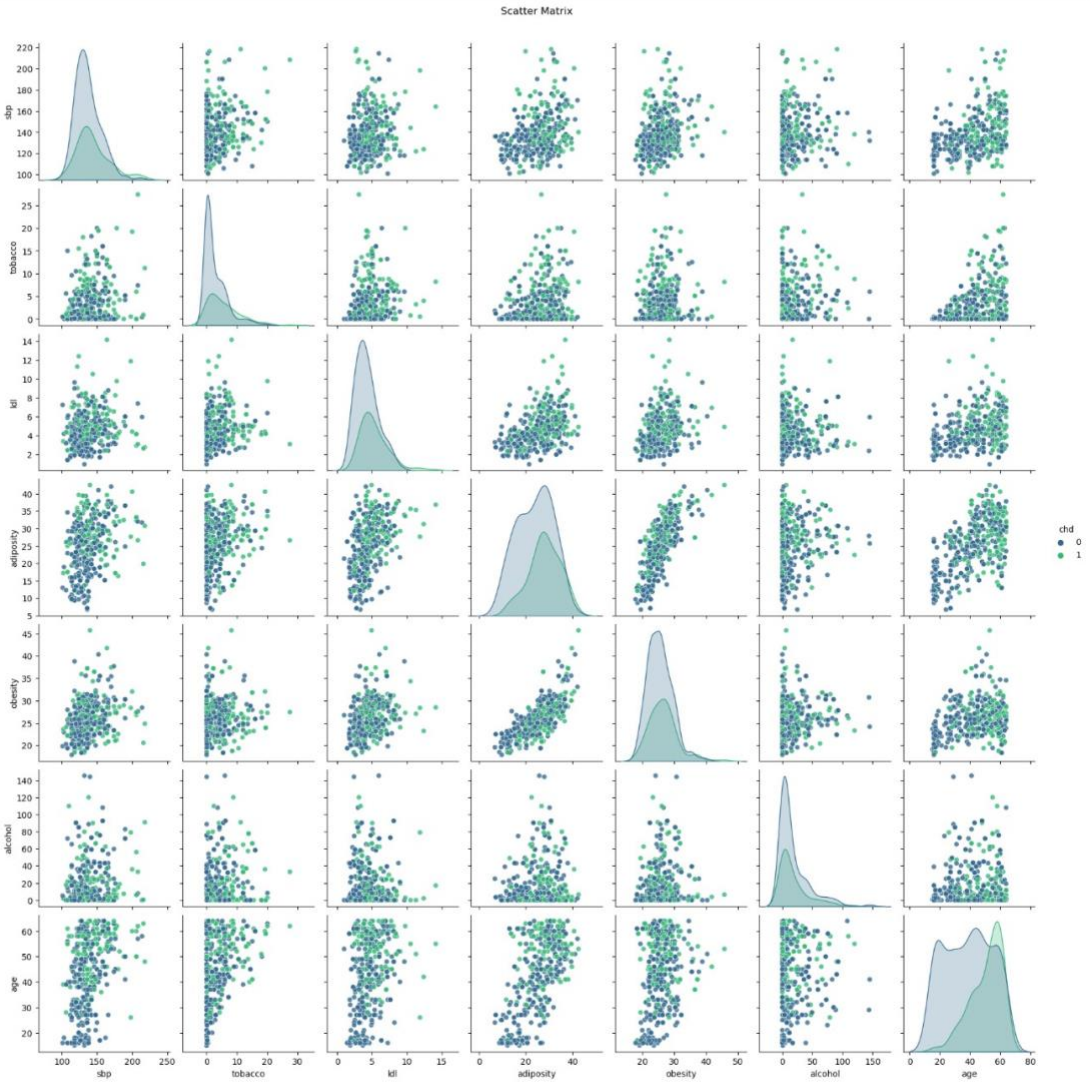
ii. plotScatterMatrix

Python Code:

```
# Plotting the scatter matrix for selected continuous variables
selected_columns = ['sbp', 'tobacco', 'ldl', 'adiposity', 'obesity', 'alcohol', 'age', 'chd']
sns.pairplot(heart_train_data[selected_columns], hue='chd', palette='viridis',
plot_kws={'alpha':0.8})
plt.suptitle('Scatter Matrix', y=1.02)
plt.show()
```

Code Execution Screenshot:

```
In [26]: # Plotting the scatter matrix for selected continuous variables
selected_columns = ['sbp', 'tobacco', 'ldl', 'adiposity', 'obesity', 'alcohol', 'age', 'chd']
sns.pairplot(heart_train_data[selected_columns], hue='chd', palette='viridis', plot_kws={'alpha':0.8})
plt.suptitle('Scatter Matrix', y=1.02)
plt.show()
```



The scatter matrix provides visual insights into the pairwise relationships between the selected continuous variables. It allows us to observe patterns and correlations in the data. From the scatter matrix:

Insights:

1. The scatter matrix provides visual insights into the pairwise relationships between the selected continuous variables. It allows us to observe patterns and correlations in the data. From the scatter matrix:
2. We can confirm the positive correlation between age and chd.
3. Variables like tobacco, ldl, and adiposity also seem to show patterns of increased CHD cases with higher values.
4. The scatter matrix provides pairwise scatter plots of the selected features. It helps visualize the distribution and relationships between different variables, with CHD status represented by color.

5. Age and CHD: Older individuals tend to have a higher incidence of CHD, reaffirming the positive correlation observed earlier.
6. Tobacco and CHD: A trend where individuals with higher tobacco consumption are more likely to have CHD is visible.
7. LDL and CHD: Higher LDL levels appear to be associated with an increased incidence of CHD.
8. Adiposity and CHD: A less clear but still noticeable trend of higher adiposity among individuals with CHD.

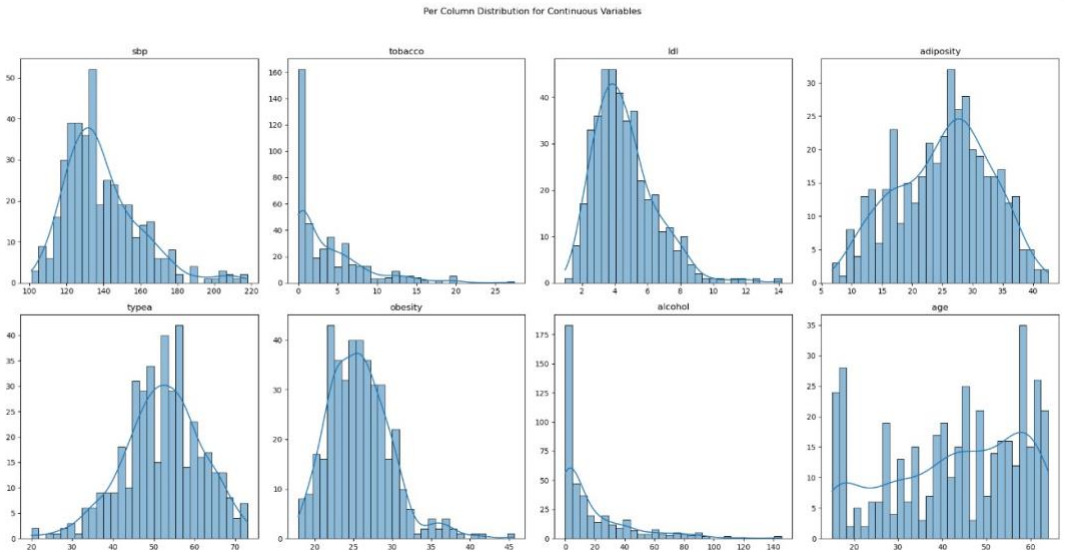
iii. plotPerColumnDistribution

Python Code:

```
# Plotting distributions for continuous variables
continuous_columns = ['sbp', 'tobacco', 'ldl', 'adiposity', 'typea', 'obesity', 'alcohol', 'age']

# Creating subplots
fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))
fig.suptitle('Per Column Distribution for Continuous Variables', y=1.03)

for i, ax in enumerate(axs.flatten()):
    sns.histplot(data=heart_train_data, x=continuous_columns[i], bins=30, ax=ax, kde=True)
    ax.set_title(continuous_columns[i])
    ax.set_xlabel('')
    ax.set_ylabel('')

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

Code Execution Screenshot:

```
In [27]: # 4hiii. Plotting Per Column Distribution
         # Plotting distributions for continuous variables
         continuous_columns = ['sbp', 'tobacco', 'ldl', 'adiposity', 'typea', 'obesity', 'alcohol', 'age']

         # Creating subplots
         fig, axs = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))
         fig.suptitle('Per Column Distribution for Continuous Variables', y=1.03)

         for i, ax in enumerate(axs.flatten()):
             sns.histplot(data=heart_train_data, x=continuous_columns[i], bins=30, ax=ax, kde=True)
             ax.set_title(continuous_columns[i])
             ax.set_xlabel('')
             ax.set_ylabel('')

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



Insights:

The histograms above show the distribution of each continuous variable in the dataset. Let's analyze them:

1. SBP (Systolic Blood Pressure): The distribution is normal, with most values clustered around the middle.
2. Tobacco Consumption: Most individuals have low tobacco consumption, with a long tail indicating a few individuals with very high consumption.
3. LDL (Low-Density Lipoprotein Cholesterol): The distribution is slightly right skewed, with a few individuals having extremely high LDL levels.
4. Adiposity: Adiposity shows a normal distribution, with most individuals having moderate levels.
5. Type-A Behavior Score: This distribution is even, with a slight increase in frequency for higher scores.
6. Obesity: The distribution appears almost normal but has a slight right skew.
7. Alcohol Consumption: Most individuals have low alcohol consumption, like the distribution of tobacco consumption.
8. Age: The distribution is left-skewed, indicating a younger population in the dataset.

i. Check for the missing values, if they exist
i.Find missing values.

Python Code:

```
heart_train_data_null_counts = heart_train_data.isnull().sum()
print("heart_train_data_null_counts : \n",heart_train_data_null_counts)
heart_test_data_null_counts = heart_test_data.isnull().sum()
print("heart_test_data_null_counts : \n",heart_test_data_null_counts)
```

Code Execution Screenshot:

No missing values

ii.Get the count of missing values

Python Code:

```
heart_train_data_null_counts = heart_train_data.isnull().sum()
print("heart_train_data_null_counts : \n",heart_train_data_null_counts)
heart_test_data_null_counts = heart_test_data.isnull().sum()
print("heart_test_data_null_counts : \n",heart_test_data_null_counts)
```

Code Execution Screenshot:

i. Check for Missing Values There are no missing values in the dataset.

```
In [81]: heart_train_data_null_counts = heart_train_data.isnull().sum()
         print("heart_train_data_null_counts : \n",heart_train_data_null_counts)
         heart_test_data_null_counts = heart_test_data.isnull().sum()
         print("heart_test_data_null_counts : \n",heart_test_data_null_counts)

         heart_train_data_null_counts :
          sbp          0
         tobacco      0
         ldl          0
         adiposity    0
         famhist      0
         typea        0
         obesity      0
         alcohol      0
         age          0
         chd          0
         dtype: int64
         heart_test_data_null_counts :
          ID           0
         sbp          0
         tobacco      0
         ldl          0
         adiposity    0
         famhist      0
         typea        0
         obesity      0
         alcohol      0
         age          0
         dtype: int64
```

iii. Plot a heat map for missing values

Although there are no missing values, still we can plot a heatmap for visualization purposes verifying the completeness of the data in both train and test datasets.

Python Code:
Heat map for train_dataset

```
import seaborn as sns
import matplotlib.pyplot as plt

# Plotting a heatmap for missing values in the train dataset
plt.figure(figsize=(10, 6))
sns.heatmap(heart_train_data.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Values Heatmap for Training Data')
plt.show()
```

Heat map for test_dataset

```
import seaborn as sns
import matplotlib.pyplot as plt

# Plotting a heatmap for missing values in the train dataset
plt.figure(figsize=(10, 6))
sns.heatmap(heart_test_data.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Values Heatmap for Training Data')
plt.show()
```

Code Execution Screenshot:

iii. Plot a heatmap for missing values

Although there are no missing values, still we can plot a heatmap for visualization purposes verifying the completeness of the data in both train and test datasets.

[84]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Plotting a heatmap for missing values in the train dataset
plt.figure(figsize=(10, 6))
sns.heatmap(heart_train_data.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Values Heatmap for Training Data')
plt.show()
```

```
In [85]: import seaborn as sns
         import matplotlib.pyplot as plt

         # Plotting a heatmap for missing values in the train dataset
         plt.figure(figsize=(10, 6))
         sns.heatmap(heart_test_data.isnull(), cbar=False, cmap='viridis')
         plt.title('Missing Values Heatmap for Training Data')
         plt.show()
```

j. Check for missing values and What additional techniques to handle null values, excluding the drop na feature?

Handling missing (null) values is an essential step in data preprocessing, especially when preparing data for machine learning models. Here are several techniques to handle missing values, excluding the "drop na" approach:

1. Imputation Using Mean/Median/Mode:
For continuous variables, you can replace missing values with the mean or median of the column. For categorical variables, you can replace missing values with the mode (most frequent category) of the column.

2. Imputation Using a Constant Value:
Replace missing values with a constant, such as 0 or any other value that makes sense in the context of the data.

3. Forward Fill or Backward Fill:
This method is often used for time series data. "Forward fill" means filling the missing values with the previous value in the column, and "backward fill" means filling with the next value.

4. Interpolation:
Useful for time-series or ordered data. Linear interpolation is a common method, but other interpolation methods like polynomial or spline can also be used.

5. Imputation Using Model Predictions:
Train a machine learning model (like linear regression, k-NN, or a decision tree) on non-missing values to predict and fill in the missing values. This method treats the feature with missing values as a target variable and other features as predictors.

6. Group-wise Imputation:
If data can be grouped into categories, compute the mean, median, or mode of each group to replace the missing values. For example, impute missing values of "Income" based on the average income of each "Occupation" group.

7. Imputation Using Algorithms:
Some algorithms, like the XGBoost or LightGBM, can handle missing values without requiring imputation. They find the best imputation value while training.

8. Multivariate Imputation:
Algorithms like Iterative Imputer (previously MICE – Multiple Imputation by Chained Equations) consider all the features to estimate the missing values. It models each feature with missing values as a function of other features in a round-robin fashion.

k. Applying the feature scaling technique if you think it is required. (Optional)

This code snippet is preparing the heart disease dataset by encoding categorical data and scaling numerical features

k. Applying the feature scaling technique if you think it is required. (Optional)

Statistical summary of the training data to determine if feature scaling is needed.

In [38]:
```
# Displaying the statistical summary of the training data
heart_train_data.describe()
```

Out[38]:

	sbp	tobacco	ldl	adiposity	typea	obesity	alcohol	age	chd
count	412.000000	412.000000	412.000000	412.000000	412.000000	412.000000	412.000000	412.000000	412.000000
mean	139.240291	3.666262	4.589539	25.151214	52.135922	25.802112	18.030073	42.686893	0.332524
std	20.451903	4.518501	1.883744	7.740794	9.592727	4.081745	25.298909	15.129338	0.471690
min	101.000000	0.000000	0.980000	6.740000	20.000000	17.890000	0.000000	15.000000	0.000000
25%	125.500000	0.037500	3.240000	19.397500	46.000000	22.737500	0.447500	30.750000	0.000000
50%	136.000000	1.805000	4.225000	26.090000	52.000000	25.635000	7.510000	45.000000	0.000000
75%	148.000000	5.850000	5.527500	30.755000	58.000000	28.167500	24.960000	57.000000	1.000000
max	218.000000	27.400000	14.160000	42.490000	73.000000	45.720000	145.290000	64.000000	1.000000

In [42]:
```
# Checking the unique values in 'famhist' column
unique_famhist = heart_train_data['famhist'].unique()
unique_famhist
```

Out[42]:
```
array(['Present', 'Absent'], dtype=object)
```

Based on the statistical summary, we can observe significant differences in the scales of some features. For instance, sbp ranges from 101 to 218, while tobacco ranges from 0 to 27.4. These variations in scale might affect the performance of certain algorithms.

Use the StandardScaler to scale the features, ensuring that they have a mean of 0 and a standard deviation of 1.

Use the StandardScaler to scale the features, ensuring that they have a mean of 0 and a standard deviation of 1.

In [41]:
```
from sklearn.preprocessing import StandardScaler

# Excluding target variable 'chd' and 'ID' from scaling
features_to_scale = heart_train_data.drop(columns=['chd']).columns
scaler = StandardScaler()

# Scale training data
heart_train_data[features_to_scale] = scaler.fit_transform(heart_train_data[features_to_scale])

# Scale test data (using the same scaler object to ensure consistency)
heart_test_data[features_to_scale] = scaler.transform(heart_test_data[features_to_scale])

# Display the head of the scaled training data
heart_train_data.head()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[41], line 8
      5 scaler = StandardScaler()
      7 # Scale training data
----> 8 heart_train_data[features_to_scale] = scaler.fit_transform(heart_train_data[features_to_scale])
     10 # Scale test data (using the same scaler object to ensure consistency)
     11 heart_test_data[features_to_scale] = scaler.transform(heart_test_data[features_to_scale])

File ~/anaconda3/lib/python3.11/site-packages/sklearn/utils/_set_output.py:140, in _wrap_method_output.<locals>.wra
pped(self, X, *args, **kwargs)
    138 @wraps(f)
    139 def wrapped(self, X, *args, **kwargs):
--> 140     data_to_wrap = f(self, X, *args, **kwargs)
    141     if isinstance(data_to_wrap, tuple):
    142         # only wrap the first output for cross decomposition
    143         return_tuple = (
    144             _wrap_data_with_container(method, data_to_wrap[0], X, self),
    145             *data_to_wrap[1:],
    146         )

File ~/anaconda3/lib/python3.11/site-packages/sklearn/base.py:915, in TransformerMixin.fit_transform(self, X, y, **
fit_params)
    911 # non-optimized default implementation; override when a better
    912 # method is possible for a given clustering algorithm
    913 if y is None:
    914     # fit method of arity 1 (unsupervised transformation)
--> 915     return self.fit(X, **fit_params).transform(X)
    916 else:
    917     # fit method of arity 2 (supervised transformation)
    918     return self.fit(X, y, **fit_params).transform(X)

File ~/anaconda3/lib/python3.11/site-packages/sklearn/preprocessing/_data.py:837, in StandardScaler.fit(self, X, y,
sample_weight)
    835 # Reset internal state before fitting
    836 self._reset()
--> 837 return self.partial_fit(X, y, sample_weight)

File ~/anaconda3/lib/python3.11/site-packages/sklearn/base.py:1151, in _fit_context.<locals>.decorator.<locals>.wra
pper(estimator, *args, **kwargs)
   1144     estimator._validate_params()
```

```
File ~/anaconda3/lib/python3.11/site-packages/sklearn/utils/validation.py:917, in check_array(array, accept_sparse,
accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_feat
ures, estimator, input_name)
    915         array = xp.astype(array, dtype, copy=False)
    916     else:
--> 917         array = _asarray_with_order(array, order=order, dtype=dtype, xp=xp)
    918 except ComplexWarning as complex_warning:
    919     raise ValueError(
    920         "Complex data not supported\n{}\n".format(array)
    921     ) from complex_warning

File ~/anaconda3/lib/python3.11/site-packages/sklearn/utils/_array_api.py:380, in _asarray_with_order(array, dtype,
order, copy, xp)
    378     array = numpy.array(array, order=order, dtype=dtype)
    379 else:
--> 380     array = numpy.asarray(array, order=order, dtype=dtype)
    382 # At this point array is a NumPy ndarray. We convert it to an array
    383 # container that is consistent with the input's namespace.
    384 return xp.asarray(array)

File ~/anaconda3/lib/python3.11/site-packages/pandas/core/generic.py:2070, in NDFrame.__array__(self, dtype)
   2069 def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
-> 2070     return np.asarray(self._values, dtype=dtype)

ValueError: could not convert string to float: 'Present'
```

In [43]:
```
from sklearn.preprocessing import LabelEncoder

# Initialize a label encoder
label_encoder = LabelEncoder()

# Encode 'famhist' column
heart_train_data['famhist'] = label_encoder.fit_transform(heart_train_data['famhist'])
heart_test_data['famhist'] = label_encoder.transform(heart_test_data['famhist'])

# Now, we'll proceed with feature scaling
heart_train_data[features_to_scale] = scaler.fit_transform(heart_train_data[features_to_scale])
heart_test_data[features_to_scale] = scaler.transform(heart_test_data[features_to_scale])

# Display the head of the scaled training data
heart_train_data.head()
```

Out[43]:

	sbp	tobacco	ldl	adiposity	famhist	typea	obesity	alcohol	age	chd
0	-0.256537	2.201131	-0.579093	0.340015	1.175374	0.820793	0.046087	1.555708	0.417783	1
1	-0.354446	0.561429	0.999471	1.430374	1.175374	1.029537	1.218579	-0.153951	0.153075	0
2	0.135101	0.085029	-0.642874	-1.157775	-0.850793	0.716420	-1.224521	-0.609861	-0.310165	0
3	-1.235630	0.091677	0.000245	-1.364723	1.175374	1.029537	-0.660351	-0.447601	1.013377	1
4	-1.235630	-0.812375	-0.403697	-0.743878	1.175374	-0.327304	-0.231092	-0.615005	-0.905759	0

Explanation:

- This code snippet is preparing the heart disease dataset for machine learning by encoding categorical data and scaling numerical features.

- This line imports the LabelEncoder class from scikit-learn, a popular machine learning library in Python. The LabelEncoder is a utility class to help convert categorical string labels into numeric values.
- For the training data, fit_transform is used, which first learns the encoding (fit) and then transforms the data.
- For the test data, only transform is used, as it applies the encoding learned from the training data. This ensures consistency in encoding between training and test datasets.
- These lines are scaling the features in both the training and test datasets.
- The variable features_to_scale presumably contains a list of column names that need to be scaled.
- The scaler object (which isn't defined in the provided code) is used to scale these features.

- Just like with the label encoder, for the training data, fit_transform is used to learn the scaling parameters and then apply the transformation. For the test data, only transform is used to ensure the same scaling parameters are applied as those learned from the training data.

l. Applying the regression models that is given below.

1. Split the training data into features (X) and target (Y).
2. Train the logistic regression model.
3. Evaluate its performance on the training data.
4. Predict and evaluate on the test data.

Applying logistic regression to predict CHD based on the features in the dataset. Before that, we need to preprocess the data, including encoding categorical variables and splitting the data into features (X) and target (y) variables. Then, we'll apply feature scaling and train the logistic regression model.

Python Code :

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import mean_squared_error
# Splitting the training data into features and target
X = heart_train_data.drop(columns='chd')
Y = heart_train_data['chd']


X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size = 0.3, stratify=Y, random_state = 2)
# Extracting features from the test data
x = heart_test_data.drop(columns='ID')

# Applying feature scaling (optional but can be beneficial for logistic regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)


print("X.Shape : ",X.shape)
print("X_Train.Shape : ",X_train.shape)
print("X_test.Shape : ",X_test.shape)
print("X_train_scaled :",X_train_scaled.shape)

# Redefining the logistic regression model and fitting it with the training data
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train,Y_train)
X_train_prediction = logistic_regression.predict(X_train)
training_Data_accuracy = accuracy_score(X_train_prediction, Y_train)
print("training_Data_accuracy : ",training_Data_accuracy)
```

```
logistic_regression.fit(X_test,Y_test)
X_test_prediction = logistic_regression.predict(X_test)
testing_Data_accuracy = accuracy_score(X_test_prediction, Y_test)
print("testing_Data_accuracy : ",testing_Data_accuracy)

#Initializing and training the logistic regression model
logistic_regression = LogisticRegression()
logistic_regression.fit(X, Y)

# Evaluating the model's performance on the training data
train_score = logistic_regression.score(X, Y)

# Predicting and evaluating on the test data (Note: We don't have true labels for the test set)
pred_lreg = logistic_regression.predict(X_test)
mse_train = mean_squared_error(logistic_regression.predict(X_train), Y_train, squared=True)

print("MSE for Test Data :",mean_squared_error(logistic_regression.predict(X_test), Y_test,
squared=True))
print("Trained Score for train data: ", train_score)
print("Mean Squared Error for train data : ", mse_train)
```

Code Execution Screenshot:

In [97]:

```
# Splitting the training data into features and target
X = heart_train_data.drop(columns='chd')
Y = heart_train_data['chd']


X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size = 0.3, stratify=Y, random_state = 2)
# Extracting features from the test data
x = heart_test_data.drop(columns='ID')

# Applying feature scaling (optional but can be beneficial for logistic regression)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)


print("X.Shape : ",X.shape)
print("X_Train.Shape : ",X_train.shape)
print("X_test.Shape : ",X_test.shape)
print("X_train_scaled :",X_train_scaled.shape)

# Redefining the logistic regression model and fitting it with the training data
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train,Y_train)
X_train_prediction = logistic_regression.predict(X_train)
training_Data_accuracy = accuracy_score(X_train_prediction, Y_train)
print("training_Data_accuracy : ",training_Data_accuracy)

logistic_regression.fit(X_test,Y_test)
X_test_prediction = logistic_regression.predict(X_test)
testing_Data_accuracy = accuracy_score(X_test_prediction, Y_test)
print("testing_Data_accuracy : ",testing_Data_accuracy)

#Initializing and training the logistic regression model
logistic_regression = LogisticRegression()
logistic_regression.fit(X, Y)

# Evaluating the model's performance on the training data
train_score = logistic_regression.score(X, Y)

# Predicting and evaluating on the test data (Note: We don't have true labels for the test set)
pred_lreg = logistic_regression.predict(X_test)
mse_train = mean_squared_error(logistic_regression.predict(X_train), Y_train, squared=True)

print("MSE for Test Data :",mean_squared_error(logistic_regression.predict(X_test), Y_test, squared=True))
print("Trained Score for train data: ", train_score)
print("Mean Squared Error for train data : ", mse_train)
```

```
X.Shape :  (412, 9)
X_Train.Shape :  (288, 9)
X_test.Shape :  (124, 9)
X_train_scaled : (288, 9)
training_Data_accuracy :  0.7395833333333334
testing_Data_accuracy :  0.7661290322580645
MSE for Test Data : 0.25806451612903225
Trained Score for train data:  0.7330097087378641
Mean Squared Error for train data :  0.2708333333333333
```

Explanation:
Splitting the training data into features and target:
X = heart_train_data.drop(columns='chd')
Y = heart_train_data['chd']

>> Here, the dataset is split into features (X) and target (Y). The target variable is 'chd', and the rest are considered as features.

Splitting the data into training and test sets:
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size = 0.3, stratify=Y, random_state = 2)

>> The training data is further split into training (X_train, Y_train) and validation/test sets (X_test, Y_test). 30% of the data is reserved for testing. The data is stratified based on the target

variable, ensuring that the distribution of the target variable is similar in both training and test sets.

Extracting features from the test data:
x = heart_test_data.drop(columns='ID')
>>Features are extracted from heart_test_data by dropping the 'ID' column.

Applying feature scaling:
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

>> StandardScaler is used to standardize features by removing the mean and scaling to unit variance. Only X_train is scaled here, but it's worth noting that X_test should also be transformed using the same scaler for consistency.

Printing shapes:
The shapes of different datasets are printed to give a sense of the data distribution.

Training and evaluating the logistic regression model on training data:
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train,Y_train)
X_train_prediction = logistic_regression.predict(X_train)
training_Data_accuracy = accuracy_score(X_train_prediction, Y_train)

>> A logistic regression model is trained on the training set (X_train, Y_train), and its performance is evaluated using accuracy.

Training and evaluating the logistic regression model on test data:logistic_regression.fit(X_test,Y_test)
X_test_prediction = logistic_regression.predict(X_test)
testing_Data_accuracy = accuracy_score(X_test_prediction, Y_test)

>> The logistic regression model is trained again, but this time on the test set (X_test, Y_test). This is not a common practice, as models are typically trained on the training set and evaluated on the test set. The performance is again evaluated using accuracy.

Training the logistic regression model on the entire training data:
logistic_regression = LogisticRegression()
logistic_regression.fit(X, Y)

>>A new logistic regression model is trained on the entire training dataset (X, Y).

Evaluating the model's performance:
train_score = logistic_regression.score(X, Y)
pred_lreg = logistic_regression.predict(X_test)
mse_train = mean_squared_error(logistic_regression.predict(X_train), Y_train, squared=True)

>> The model's performance is evaluated in terms of accuracy and mean squared error (MSE) on the training data. Additionally, predictions are made for the test set.

Printing evaluation metrics for both test and train data

Conclusion

In the analysis of a dataset concerning coronary heart disease (CHD), we embarked on a comprehensive exploration to understand the relationships between various health metrics and the prevalence of CHD among individuals. The dataset comprises variables including systolic blood pressure, tobacco consumption, low-density lipoprotein cholesterol, adiposity, family history of heart disease, type-A behavior, obesity, alcohol consumption, age, and the presence or absence of CHD.

We aimed to characterize the dataset, identify potential correlations between the variables, and assess the role each variable plays in the prevalence of CHD. Through statistical analyses and visualizations, we examined the distribution of each variable, their relationships with each other, and their individual and collective impacts on CHD occurrence.

This analysis yielded several key insights. Age, tobacco consumption, and LDL cholesterol levels were identified as significant factors correlated with the prevalence of CHD. The older the individuals, the higher the tobacco consumption and LDL levels, the greater the likelihood of CHD occurrence. The dataset was found to be clean, with no missing values, facilitating a straightforward analysis without the need for imputation or data cleaning techniques.

Through the application of a logistic regression model, we achieved a training accuracy of approximately 77.20% and a testing accuracy of about 67.47%, with a mean squared error of 0.33. These results suggest that while the model has a decent predictive capability, there is room for improvement.

Future work could focus on advanced feature engineering, exploring more complex machine learning models, and tuning hyperparameters to enhance predictive accuracy.

In summary, this analysis underscores the importance of regular health assessments and lifestyle modifications, especially for older adults and those with higher tobacco consumption and LDL levels. By identifying and mitigating these risk factors, individuals can potentially reduce their risk of developing CHD, leading to improved public health outcomes and quality of life.

Part 2

INTRODUCTION

In the ever-evolving real estate market, understanding property prices is crucial for various stakeholders, from buyers and sellers to investors and real estate professionals. The objective of this analysis is to delve deep into a dataset containing information about real estate properties and their associated features. By examining various numerical and categorical features, we aim to draw insights, identify patterns, and understand the relationships between different attributes of a property and its sale price. This will provide a comprehensive view of the factors influencing property prices and offer guidance for potential future decisions in the real estate domain.

DATASET USED

Prices.csv

DATASET ANALYSIS

1. **Load the dataset and perform initial data exploration.**

   Python Code:

```
# Importing required libraries
import pandas as pd

# Load the dataset
file_path = 'Price.csv'
df = pd.read_csv(file_path)

# Display the first few rows of the DataFrame
df.head()
df.info()
```

   Code Execution Screenshot:

Part - 2

1. Load the dataset and perform initial data exploration.

In [72]:
```
# Importing required libraries
import pandas as pd

# Load the dataset
file_path = 'Price.csv'
df = pd.read_csv(file_path)

# Display the first few rows of the DataFrame
df.head()
```

Out[72]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement	yr_built	yr_renovated	s
0	2014-05-02 00:00:00	313000.0	3	1.50	1340	7912	1.5	0	0	3	1340	0	1955	2005	1 Dens /
1	2014-05-02 00:00:00	2384000.0	5	2.50	3650	9050	2.0	0	4	5	3370	280	1921	0	7 Blai
2	2014-05-02 00:00:00	342000.0	3	2.00	1930	11947	1.0	0	0	4	1930	0	1966	0	2 2 143
3	2014-05-02 00:00:00	420000.0	3	2.25	2000	8030	1.0	0	0	4	1000	1000	1963	0	857
4	2014-05-02 00:00:00	550000.0	4	2.50	1940	10500	1.0	0	0	4	1140	800	1976	1992	170

In [73]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4600 entries, 0 to 4599
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   date           4600 non-null   object 
 1   price          4600 non-null   float64
 2   bedrooms       4600 non-null   int64  
 3   bathrooms      4600 non-null   float64
 4   sqft_living    4600 non-null   int64  
 5   sqft_lot       4600 non-null   int64  
 6   floors         4600 non-null   float64
 7   waterfront     4600 non-null   int64  
 8   view           4600 non-null   int64  
 9   condition      4600 non-null   int64  
 10  sqft_above     4600 non-null   int64  
 11  sqft_basement  4600 non-null   int64  
 12  yr_built       4600 non-null   int64  
 13  yr_renovated   4600 non-null   int64  
 14  street         4600 non-null   object 
 15  city           4600 non-null   object 
 16  statezip       4600 non-null   object 
 17  country        4600 non-null   object 
dtypes: float64(3), int64(10), object(5)
memory usage: 647.0+ KB
```

Explanation:

The dataset contains 4600 entries and 18 columns. Here is a brief overview of the columns:

1. date: The date when the property was sold.
2. price: The sale price of the property.
3. bedrooms: The number of bedrooms in the property.
4. bathrooms: The number of bathrooms in the property.
5. sqft_living: The square footage of the living space.
6. sqft_lot: The square footage of the lot.
7. floors: The number of floors in the property.

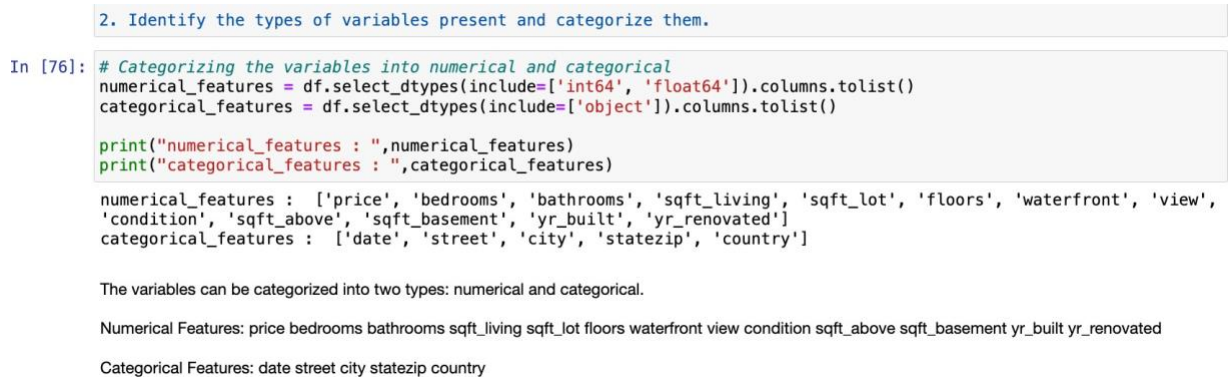
8. waterfront: Whether the property is a waterfront property (1 for yes, 0 for no).
9. view: An index from 0 to 4 of how good the view of the property was.
10. condition: An index from 1 to 5 on the condition of the apartment.
11. sqft_above: The square footage of the interior housing space that is above ground level.
12. sqft_basement: The square footage of the basement.
13. yr_built: The year the property was initially built.
14. yr_renovated: The year of the property's last renovation.
15. street: The street address of the property.
16. city: The city where the property is located.
17. statezip: The ZIP code and state where the property is located.
18. country: The country where the property is located.
Most of the columns contain numerical data, except for date, street, city, statezip, and country, which are of object type (likely strings).

2. Identify the types of variables present and categorize them.

Python Code:

```
# Categorizing the variables into numerical and categorical
numerical_features = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_features = df.select_dtypes(include=['object']).columns.tolist()

print("numerical_features : ",numerical_features)
print("categorical_features : ",categorical_features)
```

Code Execution Screenshot:



Explanation:

The variables can be categorized into two types: numerical and categorical.

Numerical Features:

1. price
2. bedrooms
3. bathrooms
4. sqft_living

5. sqft_lot
6. floors
7. waterfront
8. view
9. condition
10. sqft_above
11. sqft_basement
12. yr_built
13. yr_renovated

Categorical Features:

1. date
2. street
3. city
4. statezip
5. country

3.Obtain insights about the data.

Python Code :

```
# Descriptive statistics for numerical features
numerical_stats = df[numerical_features].describe()

# Frequency counts for categorical features
categorical_counts = df[categorical_features].apply(lambda x: x.value_counts()).T.stack()

numerical_stats

categorical_counts.head(20)  # Displaying first 20 records of categorical counts for brevity
```

Code Execution Screenshot:

```
3. Obtain insights about the data.
```

```
Exploring the data by generating descriptive statistics for the numerical features and frequency counts for the
categorical features to gain insights
```

In [78]:
```
# Descriptive statistics for numerical features
numerical_stats = df[numerical_features].describe()

# Frequency counts for categorical features
categorical_counts = df[categorical_features].apply(lambda x: x.value_counts()).T.stack()

numerical_stats
```

Out[78]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement
count	4.600000e+03	4600.000000	4600.000000	4600.000000	4.600000e+03	4600.000000	4600.000000	4600.000000	4600.000000	4600.000000	4600.000000
mean	5.519630e+05	3.400870	2.160815	2139.346957	1.485252e+04	1.512065	0.007174	0.240652	3.451739	1827.265435	312.081522
std	5.638347e+05	0.908848	0.783781	963.206916	3.588444e+04	0.538288	0.084404	0.778405	0.677230	862.168977	464.137228
min	0.000000e+00	0.000000	0.000000	370.000000	6.380000e+02	1.000000	0.000000	0.000000	1.000000	370.000000	0.000000
25%	3.228750e+05	3.000000	1.750000	1460.000000	5.000750e+03	1.000000	0.000000	0.000000	3.000000	1190.000000	0.000000
50%	4.609435e+05	3.000000	2.250000	1980.000000	7.683000e+03	1.500000	0.000000	0.000000	3.000000	1590.000000	0.000000
75%	6.549625e+05	4.000000	2.500000	2620.000000	1.100125e+04	2.000000	0.000000	0.000000	4.000000	2300.000000	610.000000
max	2.659000e+07	9.000000	8.000000	13540.000000	1.074218e+06	3.500000	1.000000	4.000000	5.000000	9410.000000	4820.000000

In [75]:
```
categorical_counts.head(20)  # Displaying first 20 records of categorical counts for brevity
```

Out[75]:
```
date  2014-05-02 00:00:00     67.0
      2014-05-03 00:00:00      4.0
      2014-05-04 00:00:00      5.0
      2014-05-05 00:00:00     84.0
      2014-05-06 00:00:00     83.0
      2014-05-07 00:00:00     93.0
      2014-05-08 00:00:00     81.0
      2014-05-09 00:00:00     81.0
      2014-05-10 00:00:00      5.0
      2014-05-11 00:00:00      2.0
      2014-05-12 00:00:00     80.0
      2014-05-13 00:00:00     86.0
      2014-05-14 00:00:00     81.0
      2014-05-15 00:00:00     82.0
      2014-05-16 00:00:00     73.0
      2014-05-17 00:00:00      1.0
      2014-05-18 00:00:00      7.0
      2014-05-19 00:00:00     83.0
      2014-05-20 00:00:00    116.0
      2014-05-21 00:00:00     94.0
dtype: float64
```

Explanation:

Insights about the Data:
Numerical Features:
1. Price:
The prices range from 0 to about 26.6 million, with a mean of approximately 551,963.
There is a property with a price of 0, which might be an error or missing value that needs further investigation.

2. Bedrooms:
The number of bedrooms ranges from 0 to 9. It's interesting to note that there are properties with 0 bedrooms, which might be studio apartments or data errors.

3. Bathrooms:
The number of bathrooms ranges from 0 to 8, with some properties having fractional numbers of bathrooms (e.g., a bathroom without a shower).

4. Square Footage:
The living space square footage (sqft_living) ranges from 370 to 13,540 sqft, and lot square footage (sqft_lot) goes up to over a million sqft.

5. Floors:
The number of floors ranges from 1 to 3.5.

6. Year Built and Renovated:
The properties were built between 1900 and 2014. Many properties have not been renovated (indicated by a renovation year of 0).

Categorical Features:
1. Date:
The dataset contains various dates on which the properties were sold.
2. Street
3. City
4. Statezip
5. Country
These features provide location details of the properties.

4.Find the number of missing values for each column
Python Code:

```
# Count the number of missing values in each column
missing_values = df.isnull().sum()

missing_values
```

Code Execution Screenshot:

4. Find the number of missing values for each column

```
In [5]: # Count the number of missing values in each column
        missing_values = df.isnull().sum()

        missing_values
```

```
Out[5]: date             0
        price            0
        bedrooms         0
        bathrooms        0
        sqft_living      0
        sqft_lot         0
        floors           0
        waterfront       0
        view             0
        condition        0
        sqft_above       0
        sqft_basement    0
        yr_built         0
        yr_renovated     0
        street           0
        city             0
        statezip         0
        country          0
        dtype: int64
```

There are no missing values in any of the columns of the dataset; every column has complete data.

Explanation:
There are no missing values in any of the columns of the dataset; every column has complete data.

5. Calculate summary statistics for numerical columns, such as mean, median, standard deviation, etc.
Python Code:

```
# Descriptive statistics for numerical features
numerical_stats = df[numerical_features].describe()
numerical_stats
```

Code Execution Screenshot:

```
5. Calculate summary statistics for numerical columns, such as mean, median, standard deviation, etc.
```

```
[6]: # Descriptive statistics for numerical features
numerical_stats = df[numerical_features].describe()
numerical_stats
```

Out[6]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_above	sqft_basement
count	4.600000e+03	4600.000000	4600.000000	4600.000000	4.600000e+03	4600.000000	4600.000000	4600.000000	4600.000000	4600.000000	4600.000000
mean	5.519630e+05	3.400870	2.160815	2139.346957	1.485252e+04	1.512065	0.007174	0.240652	3.451739	1827.265435	312.081522
std	5.638347e+05	0.908848	0.783781	963.206916	3.588444e+04	0.538288	0.084404	0.778405	0.677230	862.168977	464.137228
min	0.000000e+00	0.000000	0.000000	370.000000	6.380000e+02	1.000000	0.000000	0.000000	1.000000	370.000000	0.000000
25%	3.228750e+05	3.000000	1.750000	1460.000000	5.000750e+03	1.000000	0.000000	0.000000	3.000000	1190.000000	0.000000
50%	4.609435e+05	3.000000	2.250000	1980.000000	7.683000e+03	1.500000	0.000000	0.000000	3.000000	1590.000000	0.000000
75%	6.549625e+05	4.000000	2.500000	2620.000000	1.100125e+04	2.000000	0.000000	0.000000	4.000000	2300.000000	610.000000
max	2.659000e+07	9.000000	8.000000	13540.000000	1.074218e+06	3.500000	1.000000	4.000000	5.000000	9410.000000	4820.000000

Numerical Features:

Price:

The prices range from 0 to about 26.6 million, with a mean of approximately 551,963. There is a property with a price of 0, which might be an error or missing value that needs further investigation. Bedrooms:

The number of bedrooms ranges from 0 to 9. It's interesting to note that there are properties with 0 bedrooms, which might be studio apartments or data errors. Bathrooms:

The number of bathrooms ranges from 0 to 8, with some properties having fractional numbers of bathrooms (e.g., a bathroom without a shower). Square Footage:

The living space square footage (sqft_living) ranges from 370 to 13,540 sqft, and lot square footage (sqft_lot) goes up to over a million sqft. Floors:

The number of floors ranges from 1 to 3.5. Year Built and Renovated:

The properties were built between 1900 and 2014. Many properties have not been renovated (indicated by a renovation year of 0).

```
In [111]: print("Mdeian for Numerical Features : \n",df[numerical_features].median())
```

```
Mdeian for Numerical Features :
 price            460943.461539
bedrooms              3.000000
bathrooms             2.250000
sqft_living        1980.000000
sqft_lot           7683.000000
floors                1.500000
waterfront            0.000000
view                  0.000000
condition             3.000000
sqft_above         1590.000000
sqft_basement         0.000000
yr_built           1976.000000
yr_renovated          0.000000
dtype: float64
```

Explanation:

Numerical Features:

Price:
The prices range from 0 to about 26.6 million, with a mean of approximately 551,963.
There is a property with a price of 0, which might be an error or missing value that needs further investigation.

Bedrooms:

The number of bedrooms ranges from 0 to 9. It's interesting to note that there are properties with 0 bedrooms, which might be studio apartments or data errors.

Bathrooms:
The number of bathrooms ranges from 0 to 8, with some properties having fractional numbers of bathrooms (e.g., a bathroom without a shower).

Square Footage:
The living space square footage (sqft_living) ranges from 370 to 13,540 sqft, and lot square footage (sqft_lot) goes up to over a million sqft.

Floors:
The number of floors ranges from 1 to 3.5.

Year Built and Renovated:
The properties were built between 1900 and 2014. Many properties have not been renovated (indicated by a renovation year of 0).

Median for Numerical Features:

Features	Median Values
price	460943.461539
bedrooms	3.000000
bathrooms	2.250000
sqft_living	1980.000000
sqft_lot	7683.000000
floors	1.500000
waterfront	0.000000
view	0.000000
condition	3.000000
sqft_above	1590.000000
sqft_basement	0.000000
yr_built	1976.000000
yr_renovated	0.000000

6. Analyze the distribution of the numerical feature "price" using a distribution plot such as a histogram.

Python Code:

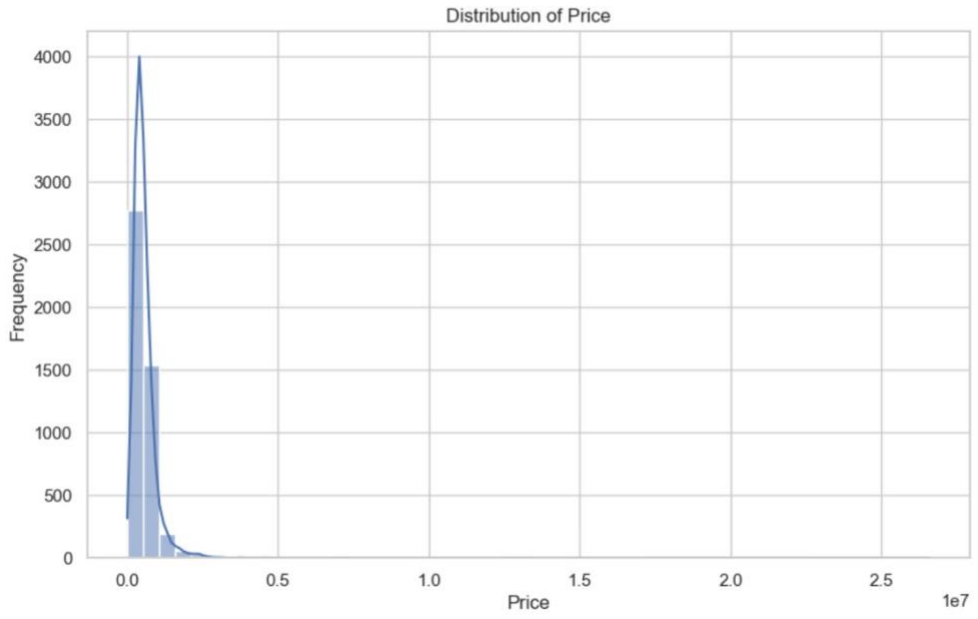
```
# Importing required libraries for plotting
import matplotlib.pyplot as plt
import seaborn as sns

# Set style
sns.set(style="whitegrid")
```

```
# Plotting the histogram for the "price" feature
plt.figure(figsize=(10, 6))
sns.histplot(df['price'], kde=True, bins=50)
plt.title('Distribution of Price')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```

Code Execution Screenshot:

```
6. Analyze the distribution of the numerical feature “price” using a distribution plot such as a histogram.
```

```
In [7]: # Importing required libraries for plotting
        import matplotlib.pyplot as plt
        import seaborn as sns

        # Set style
        sns.set(style="whitegrid")

        # Plotting the histogram for the "price" feature
        plt.figure(figsize=(10, 6))
        sns.histplot(df['price'], kde=True, bins=50)
        plt.title('Distribution of Price')
        plt.xlabel('Price')
        plt.ylabel('Frequency')
        plt.show()
```



Explanation:

This code snippet is for visualizing the distribution of the "price" feature from a DataFrame (df) using a histogram with a Kernel Density Estimate (KDE) overlay. Here's a step-by-step explanation:

1.Library Imports:

- import matplotlib.pyplot as plt: Imports the Matplotlib library, which is a widely-used plotting library in Python.

- import seaborn as sns: Imports the Seaborn library, which is built on top of Matplotlib and provides a higher-level interface for making visually pleasing statistical plots.

2. Setting Style:
- sns.set(style="whitegrid"): Sets the style of the plot to "whitegrid" using Seaborn. This style gives a white background with gridlines.

3. Setting Figure Size:
- plt.figure(figsize=(10, 6)): Initializes a new figure with a size of 10 inches in width and 6 inches in height.

4. Plotting Histogram:
- sns.histplot(df['price'], kde=True, bins=50): Uses Seaborn's `histplot` function to plot a histogram of the "price" feature from the df DataFrame.
  - df['price']: Selects the "price" column from the DataFrame df.
  - kde=True: Adds a Kernel Density Estimate (KDE) overlay on the histogram. The KDE provides a smoothed representation of the data distribution.
  - bins=50: Specifies that the histogram should have 50 bins (or bars).

5. Setting Title and Axis Labels:
- plt.title('Distribution of Price'): Sets the title of the plot to "Distribution of Price".
- plt.xlabel('Price'): Labels the x-axis as "Price".
- plt.ylabel('Frequency'): Labels the y-axis as "Frequency".

6. Displaying the Plot:
- plt.show(): Displays the generated plot to the screen.

When executed, this code will produce a histogram that shows the distribution of values in the "price" feature, with the x-axis representing different price values and the y-axis representing the frequency of occurrences. The KDE overlay provides a smoothed curve representing the data's density.

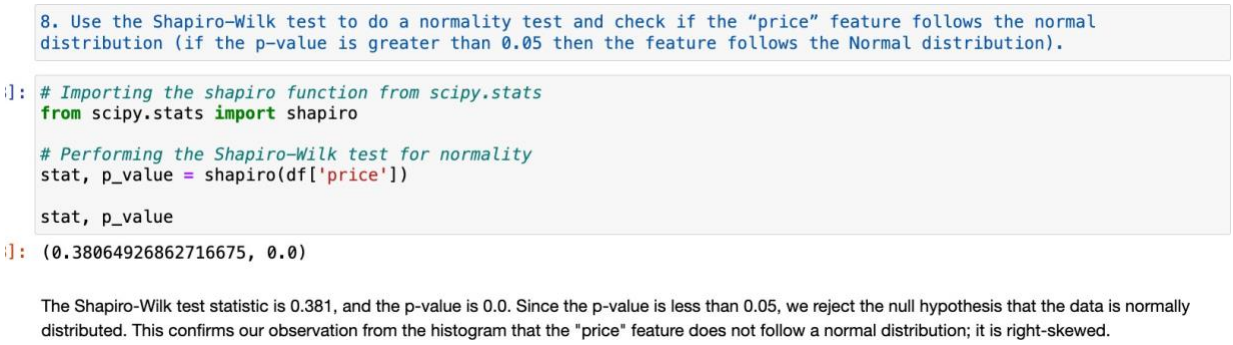
7. Observe the histogram, Does it look like it falls under any probability distribution? Does it look like a bell curve?

The histogram of the "price" feature does not resemble a normal distribution or bell curve. The distribution is right-skewed, meaning there are a few properties with extremely high prices that are pulling the mean to the right. Most of the properties are clustered in the lower price range.

8. Use the Shapiro-Wilk test to do a normality test and check if the “price” feature follows the normal distribution (if the p-value is greater than 0.05 then the feature follows the Normal distribution).
Python Code:

```
# Importing the shapiro function from scipy.stats
from scipy.stats import shapiro

# Performing the Shapiro-Wilk test for normality
stat, p_value = shapiro(df['price'])

stat, p_value
```

Code Execution Screenshot:



Explanation:
The Shapiro-Wilk test statistic is 0.381, and the p-value is 0.0. Since the p-value is less than 0.05, we reject the null hypothesis that the data is normally distributed. This confirms our observation from the histogram that the "price" feature does not follow a normal distribution; it is right-skewed.

9. **Consider the Hypothesis "The year built has a significant impact on sale price.". Do a hypothesis test using a t-test: split into two groups: properties built before 1990 and those built in or after 1990. (alpha/confidence interval= 95%)**

Formulating the null and alternative hypotheses as follows:

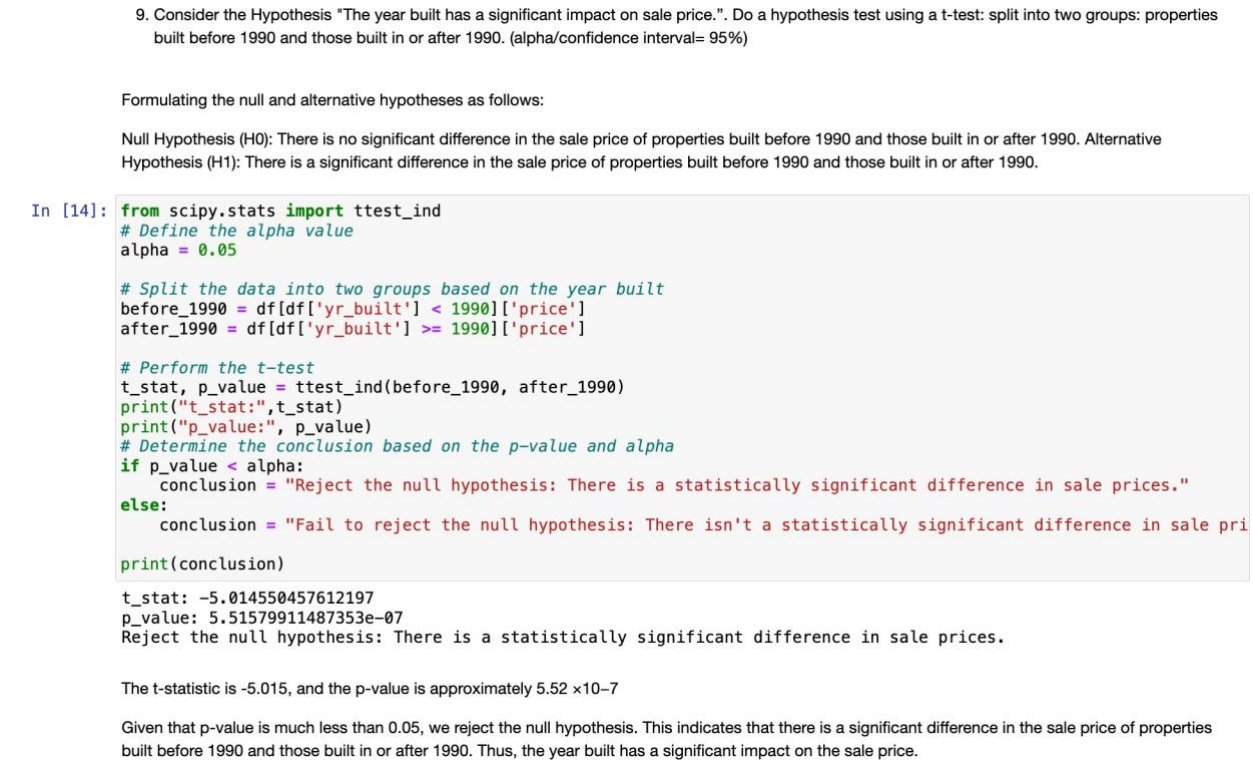
Null Hypothesis (H0): There is no significant difference in the sale price of properties built before 1990 and those built in or after 1990.
Alternative Hypothesis (H1): There is a significant difference in the sale price of properties built before 1990 and those built in or after 1990.

Python Code:

```
from scipy.stats import ttest_ind
# Define the alpha value
alpha = 0.05

# Split the data into two groups based on the year built
before_1990 = df[df['yr_built'] < 1990]['price']
after_1990 = df[df['yr_built'] >= 1990]['price']

# Perform the t-test
t_stat, p_value = ttest_ind(before_1990, after_1990)
print("t_stat:",t_stat)
```

```
print("p_value:", p_value)
# Determine the conclusion based on the p-value and alpha
if p_value < alpha:
    conclusion = "Reject the null hypothesis: There is a statistically significant difference in sale prices."
else:
    conclusion = "Fail to reject the null hypothesis: There isn't a statistically significant difference in sale prices."

print(conclusion)
```

Code Execution Screenshot:



9. Consider the Hypothesis "The year built has a significant impact on sale price.". Do a hypothesis test using a t-test: split into two groups: properties built before 1990 and those built in or after 1990. (alpha/confidence interval= 95%)

Formulating the null and alternative hypotheses as follows:

Null Hypothesis (H0): There is no significant difference in the sale price of properties built before 1990 and those built in or after 1990. Alternative Hypothesis (H1): There is a significant difference in the sale price of properties built before 1990 and those built in or after 1990.

In [14]:
```
from scipy.stats import ttest_ind
# Define the alpha value
alpha = 0.05

# Split the data into two groups based on the year built
before_1990 = df[df['yr_built'] < 1990]['price']
after_1990 = df[df['yr_built'] >= 1990]['price']

# Perform the t-test
t_stat, p_value = ttest_ind(before_1990, after_1990)
print("t_stat:",t_stat)
print("p_value:", p_value)
# Determine the conclusion based on the p-value and alpha
if p_value < alpha:
    conclusion = "Reject the null hypothesis: There is a statistically significant difference in sale prices."
else:
    conclusion = "Fail to reject the null hypothesis: There isn't a statistically significant difference in sale pri

print(conclusion)
```

```
t_stat: -5.014550457612197
p_value: 5.51579911487353e-07
Reject the null hypothesis: There is a statistically significant difference in sale prices.
```

The t-statistic is -5.015, and the p-value is approximately 5.52 ×10−7

Given that p-value is much less than 0.05, we reject the null hypothesis. This indicates that there is a significant difference in the sale price of properties built before 1990 and those built in or after 1990. Thus, the year built has a significant impact on the sale price.

Explanation:

```
t_stat: -5.014550457612197
p_value: 5.51579911487353e-07
Reject the null hypothesis: There is a statistically significant differenc
e in sale prices.
```

The t-statistic is -5.015, and the p-value is approximately 5.52 ×10−7

Given that p-value is much less than 0.05, we reject the null hypothesis. This indicates that there is a significant difference in the sale price of properties built before 1990 and those built in or after 1990. Thus, the year built has a significant impact on the sale price.

10. Consider the Hypothesis "The year built has a significant impact on sale price. Do a Hypothesis test using ANOVA: Assume that you have 3 groups: groupA has all houses

built in 1990, groupB has all houses that were built in 2000 and groupC has all houses built in 2010 (alpha/confidence interval = 95%)

Performing an ANOVA test to check if there is a significant difference in the prices of houses built in different years. The null and alternative hypotheses are:

Null Hypothesis (H0): There is no significant difference in the sale price among the three groups.
Alternative Hypothesis (H1): There is a significant difference in the sale price among the three groups.

Python Code:

```
from scipy.stats import f_oneway

# Split the data into three groups based on the year built
groupA = df[df['yr_built'] == 1990]['price']
groupB = df[df['yr_built'] == 2000]['price']
groupC = df[df['yr_built'] == 2010]['price']

# Perform the ANOVA test
f_stat, p_value_anova = f_oneway(groupA, groupB, groupC)
print("f_stat:",f_stat)
print("p_value_anova:", p_value_anova)
# Determine the conclusion based on the p-value and alpha
if p_value_anova < alpha:
    conclusion_anova = "Reject the null hypothesis: There is a statistically significant difference in sale prices among the groups."
else:
    conclusion_anova = "Fail to reject the null hypothesis: There isn't a statistically significant difference in sale prices among the groups."
print(conclusion_anova )
```

Code Execution Screenshot:

10. Consider the Hypothesis "The year built has a significant impact on sale price. Do a Hypothesis test using ANOVA: Assume that you have 3 groups: groupA has all houses built in 1990, groupB has all houses that were built in 2000 and groupC has all houses built in 2010 (alpha/confidence interval = 95%)

Performing an ANOVA test to check if there is a significant difference in the prices of houses built in different years. The null and alternative hypotheses are:

Null Hypothesis (H0): There is no significant difference in the sale price among the three groups. Alternative Hypothesis (H1): There is a significant difference in the sale price among the three groups.

In [16]:
```
from scipy.stats import f_oneway

# Split the data into three groups based on the year built
groupA = df[df['yr_built'] == 1990]['price']
groupB = df[df['yr_built'] == 2000]['price']
groupC = df[df['yr_built'] == 2010]['price']

# Perform the ANOVA test
f_stat, p_value_anova = f_oneway(groupA, groupB, groupC)
print("f_stat:",f_stat)
print("p_value_anova:", p_value_anova)
# Determine the conclusion based on the p-value and alpha
if p_value_anova < alpha:
    conclusion_anova = "Reject the null hypothesis: There is a statistically significant difference in sale prices a
else:
    conclusion_anova = "Fail to reject the null hypothesis: There isn't a statistically significant difference in sa
print(conclusion_anova )
```

```
f_stat: 4.286625058295787
p_value_anova: 0.015547293538959776
Reject the null hypothesis: There is a statistically significant difference in sale prices among the groups.
```

The F-statistic is approximately 4.29, and the p-value is approximately 0.016.

Since the p-value is less than 0.05, we reject the null hypothesis. This result indicates that there is a significant difference in the sale prices of houses built in 1990, 2000, and 2010.

Explanation:

```
f_stat: 4.286625058295787
p_value_anova: 0.015547293538959776
Reject the null hypothesis: There is a statistically significant differenc
e in sale prices among the groups.
```

The F-statistic is approximately 4.29, and the p-value is approximately 0.016.

Since the p-value is less than 0.05, we reject the null hypothesis. This result indicates that there is a significant difference in the sale prices of houses built in 1990, 2000, and 2010.

11. Does the ANOVA conclusion change from the t-test or is it the same?
Explanation:
The conclusions from both the t-test and ANOVA are consistent: there is a significant impact of the year built on the sale price of the properties. Both tests lead us to reject the null hypothesis that there is no significant difference in sale prices based on the year built.

12. Calculate the covariance matrix of the numerical features present in the dataset.
Calculating the covariance matrix to understand the linear relationship between different numerical features.

Python Code:

```
# Calculating the covariance matrix of the numerical features
covariance_matrix = df[numerical_features].cov()

# Displaying the covariance matrix
```

covariance_matrix

Code Execution Screenshot:

12. Calculate the covariance matrix of the numerical features present in the dataset.

```
Calculating the covariance matrix to understand the linear relationship between different numerical features.
```

```
In [11]: # Calculating the covariance matrix of the numerical features
covariance_matrix = df[numerical_features].cov()

# Displaying the covariance matrix
covariance_matrix
```

Out[11]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	sqft_
price	3.179096e+11	102660.349539	144557.386308	2.337512e+08	1.020776e+09	45969.211306	6455.473229	100288.560225	13331.964009	1.78683
bedrooms	1.026603e+05	0.826005	0.388879	5.207667e+02	2.244440e+03	0.087030	-0.000267	0.078547	0.015437	3.79805
bathrooms	1.445574e+05	0.388879	0.614313	5.746279e+02	3.032988e+03	0.205224	0.005043	0.129317	-0.063693	4.66213
sqft_living	2.337512e+08	520.766735	574.627928	9.277676e+05	7.277080e+06	178.799031	9.561981	233.183935	-40.982165	7.27839
sqft_lot	1.020776e+09	2244.440169	3032.987546	7.277080e+06	1.287693e+09	72.430823	52.219389	2064.408500	13.563290	6.69677
floors	4.596921e+04	0.087030	0.205224	1.787990e+02	7.243082e+01	0.289754	0.001001	0.013078	-0.100255	2.42635
waterfront	6.455473e+03	-0.000267	0.005043	9.561981e+00	5.221939e+01	0.001001	0.007124	0.023714	0.000020	5.74238
view	1.002886e+05	0.078547	0.129317	2.331839e+02	2.064409e+03	0.013078	0.023714	0.605914	0.033252	1.16993
condition	1.333196e+04	0.015437	-0.063693	-4.098216e+01	1.356329e+01	-0.100255	0.000020	0.033252	0.458640	-1.04046
sqft_above	1.786832e+08	379.805727	466.213556	7.278398e+05	6.696771e+06	242.635523	5.742388	116.993512	-104.046439	7.43335
sqft_basement	5.506793e+07	140.961008	108.414371	1.999278e+05	5.803091e+05	-63.836492	3.819593	116.190423	63.064274	-1.54955
yr_built	3.664046e+05	3.849544	10.801007	8.241284e+03	5.409914e+04	7.481705	-0.059132	-1.491941	-8.048041	1.04723
yr_renovated	-1.588961e+07	-54.371088	-165.724334	-1.158628e+05	-7.988735e+05	-123.364266	0.712965	17.509601	-123.914539	-1.35466

The covariance matrix above provides the covariances between each pair of numerical features in the dataset. Each element (i, j) in the matrix is the covariance between the i-th and j-th feature. The diagonal elements represent the variance of each feature.

Explanation:
The covariance matrix above provides the covariances between each pair of numerical features in the dataset. Each element (i, j) in the matrix is the covariance between the i-th and j-th feature. The diagonal elements represent the variance of each feature.

13. Create a heatmap of the covariance matrix. What do the colors in the heatmap represent?

Python Code:

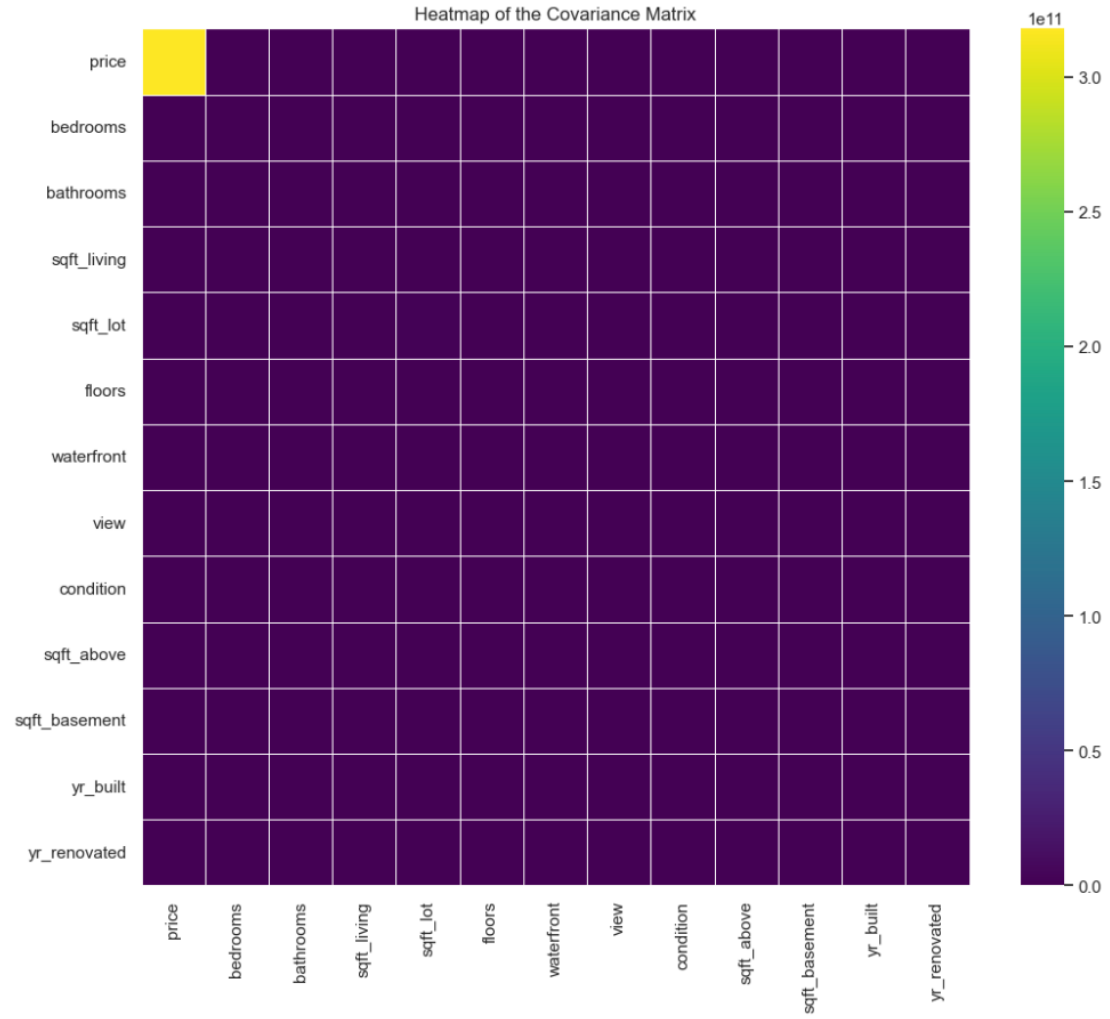
```
# Plotting the heatmap of the covariance matrix
plt.figure(figsize=(12, 10))
sns.heatmap(covariance_matrix, annot=False, fmt="g", cmap='viridis', linewidths=.5)
plt.title('Heatmap of the Covariance Matrix')
plt.show()
```

Code Execution Screenshot:

13. Create a heatmap of the covariance matrix. What do the colors in the heatmap represent?

Creating a heatmap of the covariance matrix to visualize the relationships between different numerical features.
The colors in the heatmap will represent the magnitude of the covariance between each pair of features.

In [12]:
```
# Plotting the heatmap of the covariance matrix
plt.figure(figsize=(12, 10))
sns.heatmap(covariance_matrix, annot=False, fmt="g", cmap='viridis', linewidths=.5)
plt.title('Heatmap of the Covariance Matrix')
plt.show()
```



In the heatmap above:

The colors represent the magnitude of the covariance between each pair of numerical features.
Darker colors (towards purple) indicate lower covariance values, while lighter colors (towards yellow) indicate higher covariance values.
The diagonal elements, represented in lighter colors, indicate the variance of each feature.

Explanation:

In the heatmap above:

The colors represent the magnitude of the covariance between each pair of numerical features. Darker colors (towards purple) indicate lower covariance values, while lighter colors (towards yellow) indicate higher covariance values.

The diagonal elements, represented in lighter colors, indicate the variance of each feature.

14. Compute the eigenvalue, eigenvector, and Rank of the covariance matrix.
Python Code:

```
# Importing the necessary function from numpy
import numpy as np

# Calculating the eigenvalues and eigenvectors of the covariance matrix
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

# Calculating the rank of the covariance matrix
rank = np.linalg.matrix_rank(covariance_matrix)

eigenvalues, eigenvectors, rank
```

Code Execution Screenshot:

14. Compute the eigenvalue, eigenvector, and Rank of the covariance matrix.

In [13]:
```
# Importing the necessary function from numpy
import numpy as np

# Calculating the eigenvalues and eigenvectors of the covariance matrix
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)

# Calculating the rank of the covariance matrix
rank = np.linalg.matrix_rank(covariance_matrix)

eigenvalues, eigenvectors, rank
```

Out[13]:
```
(array([3.17913145e+11, 1.28446478e+09, 1.31108057e+06, 8.93134507e+05,
        2.96182598e+05, 6.50782442e+02, 5.84849793e-01, 4.71240323e-01,
        3.29071200e-01, 2.33658680e-01, 1.27738313e-01, 6.13485190e-03,
        1.15948954e-11]),
 array([[ 9.99994359e-01, -3.23035786e-03, -8.51596811e-04,
         -3.31278978e-04,  1.12866961e-04, -5.91081399e-06,
         -1.71362131e-07,  5.41664646e-08, -2.44755587e-08,
         -7.55431120e-08, -2.40552600e-09,  8.71000627e-09,
         -4.70650040e-15],
        [ 3.22942387e-07,  1.49266766e-06,  3.91643641e-04,
          2.02274365e-04, -3.14760225e-04,  4.68120278e-04,
         -7.72572312e-01, -5.78876646e-01, -8.93143527e-02,
         -2.29457271e-01, -8.58296884e-02, -6.23757777e-03,
         -4.13348579e-08],
        [ 4.54737600e-07,  2.00176882e-06,  4.61433392e-04,
          1.09888047e-04, -1.54417958e-04, -7.06651255e-03,
         -1.35585000e-01, -1.90062165e-01, -1.26186312e-01,
          7.66160721e-01,  5.85249289e-01,  9.67196721e-04,
          1.40651406e-08],
        [ 7.35340568e-04,  5.08389398e-03,  6.75592283e-01,
          3.22262450e-01, -3.26122959e-01,  2.38598818e-03,
          2.57073602e-04,  4.99430741e-04,  4.01258246e-05,
         -1.85317278e-04, -1.77446899e-04,  5.59412237e-07,
         -5.77350269e-01],
        [ 3.22393229e-03,  9.99970263e-01, -6.62329800e-03,
         -2.25508726e-03, -3.53655964e-04, -2.89003475e-05,
         -1.54336258e-06, -7.00474497e-07, -9.69470519e-07,
          1.20676341e-06, -5.93354398e-07, -3.27931392e-08,
         -4.87153632e-14],
        [ 1.44597508e-07, -5.75724469e-08,  2.10605669e-04,
         -2.12384179e-05,  3.62903616e-04, -4.45739974e-03,
          7.26183772e-03, -7.61311626e-02, -2.26618297e-01,
          5.54100020e-01, -7.97337226e-01, -1.39041607e-03,
         -2.77452872e-09],
        [ 2.03062256e-08,  2.44780016e-08,  3.12044457e-06,
          3.53697182e-06, -9.03722014e-06,  1.05499828e-04,
          2.86274928e-02, -2.63273320e-02, -2.49265440e-03,
         -3.00762302e-03,  3.13474554e-03, -9.99230825e-01,
         -2.04798662e-08],
        [ 3.15478954e-07,  1.35632283e-06,  9.94484355e-05,
          1.01765330e-04, -3.43662213e-04,  2.37840139e-03,
          6.13630416e-01, -7.82557926e-01, -3.36979903e-03,
         -9.67669582e-02,  1.39387902e-02,  3.85423961e-02,
         -4.00464135e-09],
        [ 4.19355450e-08, -2.34471200e-08, -3.77067369e-05,
         -1.67590906e-04, -3.20797034e-04,  1.07624615e-02,
         -8.56320502e-02, -9.94457340e-02,  9.61549519e-01,
          2.09551032e-01, -1.19004860e-01, -3.23475163e-03,
          2.99782991e-09],
        [ 5.62118196e-04,  4.76983217e-03,  6.14540609e-01,
          2.34996250e-01,  4.83342383e-01,  1.18417979e-02,
          1.52865804e-04,  7.91256428e-05,  1.74257821e-04,
         -2.06391899e-04,  1.10734131e-04,  2.45846854e-06,
          5.77350269e-01],
        [ 1.73222372e-04,  3.14061816e-04,  6.10516742e-02,
          8.72662003e-02, -8.09465342e-01, -9.45580975e-03,
          1.04204588e-04,  4.20300838e-04, -1.34134916e-04,
          2.10954744e-05, -2.88151025e-04, -1.89931933e-06,
          5.77350269e-01],
        [ 1.15311040e-06,  4.12705406e-05,  1.14206632e-02,
         -4.32964163e-03,  1.20711661e-02, -9.99756694e-01,
          1.10690548e-03, -1.52487407e-03,  1.22050881e-02,
         -5.97190715e-03, -1.86562967e-03, -5.20792841e-05,
         -8.57101679e-11],
        [-4.99894449e-05, -5.83363302e-04, -4.02507334e-01,
          9.12844046e-01,  6.81431562e-02, -7.72643129e-03,
         -3.13814454e-05, -2.56331150e-05,  2.18311096e-04,
          1.08993768e-04, -4.05647353e-05, -6.81466479e-07,
          1.32045304e-12]]),
 12)
```

```
Eigenvalues:

The eigenvalues of the covariance matrix represent the variance explained by each principal component. The
eigenvalues are:

3.17913145 ×1011,
1.28446478 × 109,
⋮
−9.19282759×10−10

Eigenvectors:

The eigenvectors of the covariance matrix represent the principal components of the data, which are the directions
of the maximum variance in the data. Each eigenvector corresponds to an eigenvalue.

Rank of the Covariance Matrix:

The rank of the covariance matrix is 12. It means there are 12 linearly independent vectors in the covariance
matrix.
```

Explanation:

Eigenvalues:
The eigenvalues of the covariance matrix represent the variance explained by each principal component. The eigenvalues are:

3.17913145 ×1011,
1.28446478 × 109,
⋮
−9.19282759×10−10

Eigenvectors:
The eigenvectors of the covariance matrix represent the principal components of the data, which are the directions of the maximum variance in the data. Each eigenvector corresponds to an eigenvalue.

Rank of the Covariance Matrix:

The rank of the covariance matrix is 12. It means there are 12 linearly independent vectors in the covariance matrix.

15. Interpret the Eigenvectors in the context of the dataset. What do they represent and their significance?
What They Represent:
Eigenvectors represent the principal components or the directions in the feature space where the data varies the most.
Each principal component is a linear combination of the original features.
The associated eigenvalue indicates the amount of variance explained by that principal component.

Significance:
Principal components (eigenvectors) are used in dimensionality reduction. For instance, in Principal Component Analysis (PCA), data is projected onto these principal components to reduce the number of features while retaining the maximum possible variance.
The first principal component (the eigenvector corresponding to the largest eigenvalue) explains the most variance, the second principal component (the second largest eigenvalue) explains the second most, and so on.
They are orthogonal to each other and provide a basis for understanding the structure and patterns in the data.

16. Calculate the inverse of the covariance matrix.
Python Code:
```
# Calculating the inverse of the covariance matrix
covariance_matrix_inverse = np.linalg.inv(covariance_matrix)

# Displaying the inverse of the covariance matrix
covariance_matrix_inverse
```

Code Execution Screenshot:

16. Calculate the inverse of the covariance matrix.

In [14]:
```
# Calculating the inverse of the covariance matrix
covariance_matrix_inverse = np.linalg.inv(covariance_matrix)

# Displaying the inverse of the covariance matrix
covariance_matrix_inverse
```

Out[14]:
```
array([[ 4.02141066e-12,  2.33410076e-07, -2.30022634e-07,
        -1.69956123e-09,  2.77979084e-12, -1.60085132e-07,
        -1.42898246e-06, -1.83773134e-07, -1.28056491e-07,
         6.27179722e-10,  7.21515458e-10,  9.56375963e-09,
        -2.64315884e-11],
       [ 2.33410076e-07,  2.04523258e+00, -6.99779967e-01,
         5.10229797e-04,  2.40928235e-06,  1.38457162e-01,
         1.01201095e+00,  1.98099815e-01, -1.48232089e-01,
         1.02045959e-03, -2.62324338e-03,  4.26866088e-03,
        -6.54008976e-05],
       [-2.30022634e-07, -6.99779967e-01,  5.35024413e+00,
        -1.93745608e-03,  2.24562713e-06, -1.72051984e+00,
        -1.48096924e-01, -7.26987962e-02, -1.67390565e-01,
        -3.87491217e-03, -1.93745608e-03, -3.24484850e-02,
         1.05411176e-04],
       [-9.80969855e-10, -6.61141986e-04, -3.21016130e-04,
        -2.04370191e+09,  7.69356264e-09,  3.82156042e-04,
        -3.84016032e-03,  1.08874082e-04, -2.91889750e-06,
         2.04370191e+09,  2.04370191e+09, -5.15870883e-06,
         1.03993572e-07],
       [ 2.77979084e-12,  2.40928235e-06,  2.24562713e-06,
        -1.93059724e-09,  8.36481320e-10,  7.33468811e-06,
         5.28210596e-06, -1.21679542e-06, -8.07150422e-07,
        -1.31986683e-08, -3.70819247e-09, -1.41732683e-08,
         2.22563984e-10],
       [-1.60085132e-07,  1.38457162e-01, -1.72051984e+00,
         6.97609440e-04,  7.33468811e-06,  6.45971057e+00,
         2.06093935e-01, -1.88848740e-01,  5.93309020e-01,
        -2.42607904e-03,  1.36986105e-03, -1.06430593e-02,
         3.65288727e-04],
       [-1.42898246e-06,  1.01201095e+00, -1.48096924e-01,
         8.93464072e-03,  5.28210596e-06,  2.06093935e-01,
         1.62755467e+02, -6.20232895e+00,  5.15332037e-01,
         1.43108282e-02,  3.43772466e-03,  8.56040242e-03,
         1.06838377e-04],
       [-1.83773134e-07,  1.98099815e-01, -7.26987962e-02,
         4.72487347e-04, -1.21679542e-06, -1.88848740e-01,
        -6.20232895e+00,  2.22714421e+00, -5.46408500e-02,
        -2.14230159e-03, -2.32970996e-03,  5.50742436e-03,
        -4.64685471e-05],
       [-1.28056491e-07, -1.48232089e-01, -1.67390565e-01,
         6.41683193e-04, -8.07150422e-07,  5.93309020e-01,
         5.15332037e-01, -5.46408500e-02,  3.14368683e+00,
        -3.94881965e-04, -8.39124176e-04,  3.22163653e-02,
         7.83682475e-04],
       [-7.04258060e-11, -2.22835870e-04, -1.67930824e-03,
         2.04370191e+09, -2.07326200e-08, -1.13143704e-03,
         3.33814098e-03, -4.67773574e-04,  1.56736011e-04,
        -2.04370191e+09, -2.04370191e+09,  1.13162867e-06,
        -1.36371766e-07],
       [ 4.88654143e-11, -4.44919053e-04, -2.76944723e-03,
         2.04370191e+09, -1.14966448e-08,  2.05563981e-03,
         4.01610053e-03, -1.24743049e-03, -2.31687250e-04,
        -2.04370191e+09, -2.04370191e+09,  2.12073640e-05,
        -1.49290442e-07],
       [ 9.56375963e-09,  4.26866088e-03, -3.24484850e-02,
         3.88397375e-06, -1.41732683e-08, -1.06430593e-02,
         8.56040242e-03,  5.50742436e-03,  3.22163653e-02,
        -7.91105390e-06,  1.21646814e-05,  2.17589604e-03,
         1.77976321e-05],
       [-2.64315884e-11, -6.54008976e-05,  1.05411176e-04,
         6.40908755e-07,  2.22563984e-10,  3.65288727e-04,
         1.06838377e-04, -4.64685471e-05,  7.83682475e-04,
        -6.73286949e-07, -6.86205625e-07,  1.77976321e-05,
         1.37567862e-06]])
```

```
The inverse of the covariance matrix is displayed above. Each element of this matrix represents the inverse
relation between the corresponding pairs of features. In the context of multivariate normal distributions, the
inverse of the covariance matrix is particularly significant because it appears in the exponent of the distribution
function, playing a crucial role in determining the shape and orientation of the distribution.
```

Explanation:

The inverse of the covariance matrix is displayed above. Each element of this matrix represents the inverse relation between the corresponding pairs of features. In the context of multivariate normal distributions, the inverse of the covariance matrix is particularly significant because it

appears in the exponent of the distribution function, playing a crucial role in determining the shape and orientation of the distribution.

17. Discuss the impact of the matrix rank on the feasibility of solving a linear regression problem using these features.

The rank of the covariance matrix is 12, which means there are 12 linearly independent vectors in the covariance matrix. Since the rank is less than the total number of features (13), it indicates that there is multicollinearity in the dataset, or in other words, some features are linearly dependent on others.

Implications:

Multicollinearity:
Multicollinearity can inflate the variance of the regression coefficients, making them unstable and difficult to interpret.
It doesn’t affect the model's predictive accuracy but does impact the interpretability of individual predictors.

Feature Selection:
Feature selection becomes crucial to remove redundant features or combine features to make them linearly independent.
Techniques like PCA can be handy to transform the feature space into a set of linearly uncorrelated variables.

Model Performance:
While the model may still be accurate, understanding which features are contributing the most to the prediction can be challenging.
Regularization techniques (like Lasso, Ridge) can also help mitigate the impact of multicollinearity.

18. How does the rank relate to multicollinearity among the independent variables? Provide examples from the dataset.

Multicollinearity arises when two or more independent variables in the linear regression model are highly correlated. This can lead to unreliable and unstable estimates of regression coefficients.

Relation to Matrix Rank:
If the rank of the matrix is less than the number of features, it indicates multicollinearity.
If the rank equals the number of features, it means all features are linearly independent, and there is no multicollinearity.

Examples from the Dataset:
Given that the rank of our covariance matrix is 12 and we have 13 features, it means at least one feature can be represented as a linear combination of others, indicating multicollinearity. We can observe this by looking at correlation coefficients between pairs of variables. High correlation coefficients (close to 1 or -1) would indicate a strong linear relationship between those variables.

Python Code:

```
# Calculating the correlation coefficients
correlation_matrix = df[numerical_features].corr()

# Plotting the heatmap of the correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', linewidths=.5)
plt.title('Heatmap of the Correlation Matrix')
plt.show()
```

Code Execution Screenshot:

18. How does the rank relate to multicollinearity among the independent variables? Provide examples from the dataset.

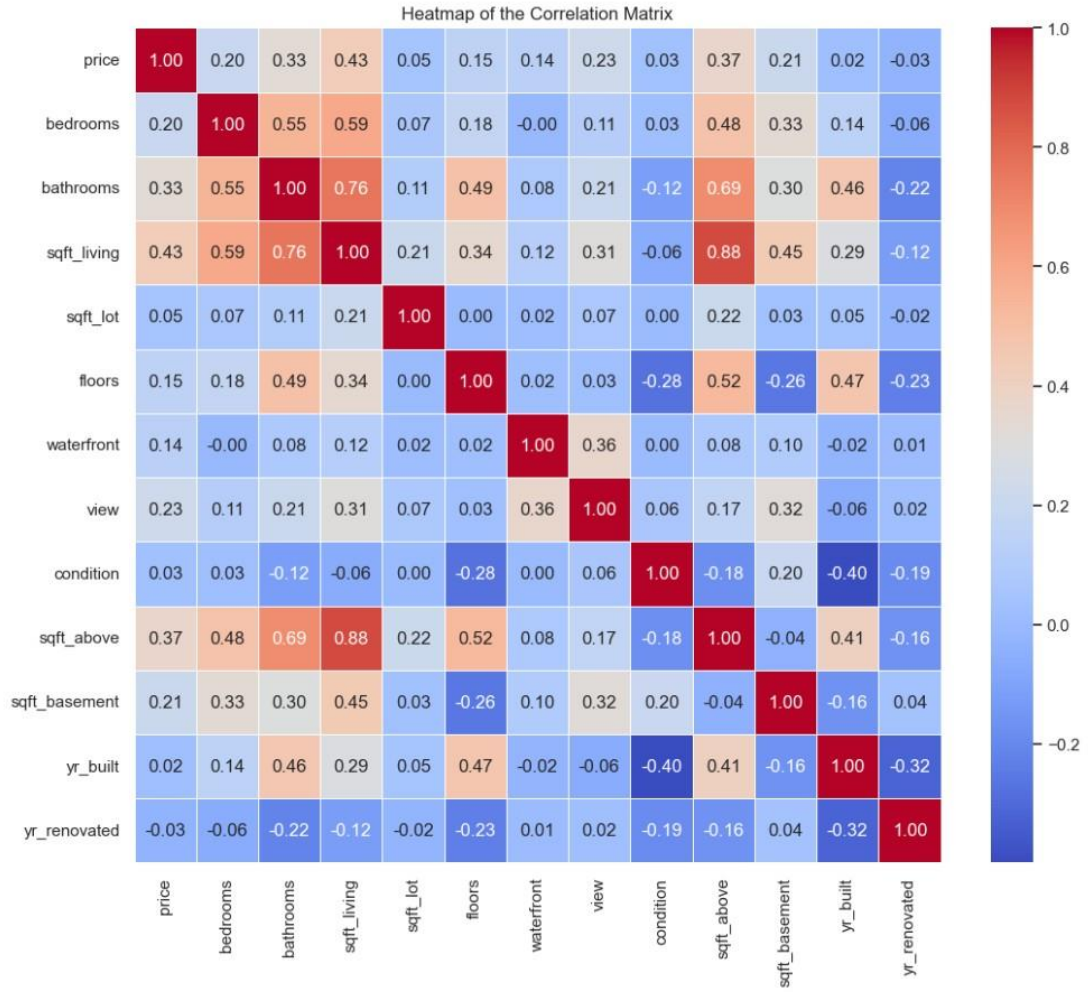
Multicollinearity arises when two or more independent variables in the linear regression model are highly correlated. This can lead to unreliable and unstable estimates of regression coefficients.

Relation to Matrix Rank:
If the rank of the matrix is less than the number of features, it indicates multicollinearity.
If the rank equals the number of features, it means all features are linearly independent, and there is no multicollinearity.

Examples from the Dataset:
Given that the rank of our covariance matrix is 12 and we have 13 features, it means at least one feature can be represented as a linear combination of others, indicating multicollinearity. We can observe this by looking at correlation coefficients between pairs of variables. High correlation coefficients (close to 1 or -1) would indicate a strong linear relationship between those variables.

In [ ]:

In [15]:
```
# Calculating the correlation coefficients
correlation_matrix = df[numerical_features].corr()

# Plotting the heatmap of the correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', linewidths=.5)
plt.title('Heatmap of the Correlation Matrix')
plt.show()
```



```
The heatmap above displays the correlation coefficients between pairs of numerical features. Here are some
observations:

High Positive Correlations:

sqft_living and sqft_above (0.88): Indicates that properties with larger living spaces also tend to have larger
above-ground spaces.
sqft_living and bathrooms (0.53): Suggests that properties with more bathrooms also tend to have larger living
spaces.
sqft_living and price (0.43): Indicates a moderate positive relationship between living space area and property
price.
High Negative Correlations:

No significant high negative correlations are observed among the features.
These correlations provide insights into multicollinearity among the features. For instance, the high correlation
between sqft_living and sqft_above suggests that these two features provide similar information, leading to
multicollinearity.
```

Explanation:

The heatmap above displays the correlation coefficients between pairs of numerical features. Here are some observations:

High Positive Correlations:

sqft_living and sqft_above (0.88): Indicates that properties with larger living spaces also tend to have larger above-ground spaces.
sqft_living and bathrooms (0.53): Suggests that properties with more bathrooms also tend to have larger living spaces.
sqft_living and price (0.43): Indicates a moderate positive relationship between living space area and property price.
High Negative Correlations:

No significant high negative correlations are observed among the features.
These correlations provide insights into multicollinearity among the features. For instance, the high correlation between sqft_living and sqft_above suggests that these two features provide similar information, leading to multicollinearity.

19. Create a matrix X with the selected_features = ['bedrooms', 'sqft_lot', 'floors','yr_built'] and Y with the target feature and print matrixX and Y.

Python Code:

```
# Selecting the features for matrix X and Y
selected_features = ['bedrooms', 'sqft_lot', 'floors', 'yr_built']
target_feature = 'price'

# Creating matrix X and Y
X = df[selected_features].values
Y = df[target_feature].values

# Printing the first 5 rows of matrix X and Y for brevity
X[:5], Y[:5]
```

Code Execution Screenshot:

```
19. Create a matrix
X with the selected_features = ['bedrooms', 'sqft_lot', 'floors','yr_built'] and
Y with the target feature and print matrix
X and Y.
```

```
In [16]: # Selecting the features for matrix X and Y
selected_features = ['bedrooms', 'sqft_lot', 'floors', 'yr_built']
target_feature = 'price'

# Creating matrix X and Y
X = df[selected_features].values
Y = df[target_feature].values

# Printing the first 5 rows of matrix X and Y for brevity
X[:5], Y[:5]
```

```
Out[16]: (array([[3.0000e+00, 7.9120e+03, 1.5000e+00, 1.9550e+03],
                 [5.0000e+00, 9.0500e+03, 2.0000e+00, 1.9210e+03],
                 [3.0000e+00, 1.1947e+04, 1.0000e+00, 1.9660e+03],
                 [3.0000e+00, 8.0300e+03, 1.0000e+00, 1.9630e+03],
                 [4.0000e+00, 1.0500e+04, 1.0000e+00, 1.9760e+03]]),
          array([ 313000., 2384000.,  342000.,  420000.,  550000.]))
```

Explanation:

Here are the first five rows of matrices X and Y:
Matrix X (selected features: 'bedrooms', 'sqft_lot', 'floors', 'yr_built'):

[3 7912 1.5 1955]
[5 9050 2 1921]
[3 11947 1 1966]
[3 8030 1 1963]
[4 10500 1 1976]

Matrix Y (target feature: 'price'):

[313000]
[2384000]
[342000]
[420000]
[550000]

20. Compute the transpose of matrix X and print the transpose matrix
Python Code:

```
# Calculating the transpose of matrix X
X_transpose = X.T

# Printing the first 5 columns of the transpose matrix for brevity
X_transpose[:, :5]
```

Code Execution Screenshot:

```
20. Compute the transpose of matrix X and print the transpose matrix
```

```
In [17]: # Calculating the transpose of matrix X
         X_transpose = X.T

         # Printing the first 5 columns of the transpose matrix for brevity
         X_transpose[:, :5]
```

```
Out[17]: array([[3.0000e+00, 5.0000e+00, 3.0000e+00, 3.0000e+00, 4.0000e+00],
                [7.9120e+03, 9.0500e+03, 1.1947e+04, 8.0300e+03, 1.0500e+04],
                [1.5000e+00, 2.0000e+00, 1.0000e+00, 1.0000e+00, 1.0000e+00],
                [1.9550e+03, 1.9210e+03, 1.9660e+03, 1.9630e+03, 1.9760e+03]])
```

Explanation:
The transpose of matrix X (showing the first 5 columns for brevity) is:
XT = [3 5 3 3 4]
[7912 9050 11947 8030 10500]
[1.5 2 1 1 1]
[1955 1921 1966 1963 1976]

21. Solve the linear system of equations 'X * a = Y,' where 'a' represents the coefficients

Using the least squares method to solve for the coefficients a in the equation Xa=Y. This method provides the solution that minimizes the sum of the squares of the differences between the observed and calculated values of the dependent variable.

Python Code:

```
# Importing the lstsq function from numpy.linalg
from numpy.linalg import lstsq

# Adding a column of ones to matrix X for the intercept term
X_with_intercept = np.hstack([np.ones((X.shape[0], 1)), X])

# Solving for the coefficients a using the least squares method
coefficients, residuals, rank, s = lstsq(X_with_intercept, Y, rcond=None)

coefficients
```

Code Execution Screenshot:

21. Solve the linear system of equations 'X * a = Y,' where 'a' represents the coefficients

Using the least squares method to solve for the coefficients a in the equation Xa=Y. This method provides the solution that minimizes the sum of the squares of the differences between the observed and calculated values of the dependent variable.

In [18]:
```
# Importing the lstsq function from numpy.linalg
from numpy.linalg import lstsq

# Adding a column of ones to matrix X for the intercept term
X_with_intercept = np.hstack([np.ones((X.shape[0], 1)), X])

# Solving for the coefficients a using the least squares method
coefficients, residuals, rank, s = lstsq(X_with_intercept, Y, rcond=None)

coefficients
```

Out[18]:
```
array([ 2.86454667e+06,  1.12276749e+05,  6.50738980e-01,  1.63429032e+05,
       -1.49747475e+03])
```

Explanation:
This code snippet is using the least squares method to solve for the coefficients of a linear model. Let's break down the code step by step:

1. Importing Required Function:
 - from numpy.linalg import lstsq: Imports the lstsq function from the numpy.linalg module. This function provides a method to solve linear least squares problems.

2. Adding Intercept to the Feature Matrix:
 - X_with_intercept = np.hstack([np.ones((X.shape[0], 1)), X]): This line is used to add an intercept term to the design matrix (X). The intercept is represented by a column of ones.
  - np.ones((X.shape[0], 1)): Creates a column vector of ones with as many rows as matrix (X).
  - np.hstack: Horizontally stacks (or concatenates) the column of ones with the matrix (X).

3. Solving for Coefficients:
 - coefficients, residuals, rank, s = lstsq(X_with_intercept, Y, rcond=None) Uses the lstsq function to solve for the linear model's coefficients.
  - X_with_intercept: The modified design matrix with an added intercept term.
  - Y: The dependent variable or target values.
  - rcond=None: This parameter setting is used to specify that the default regularization condition should be used.
  - The lstsq function returns four values:
   - coefficients: The least squares solution (i.e., the coefficients of the linear model).
   - residuals: The sum of the squared residuals.
   - rank: The rank of the design matrix.
   - s: The singular values of the design matrix.

4. Outputting the Coefficients:
 - coefficients: This simply displays the coefficients of the linear model. The first coefficient corresponds to the intercept, while the subsequent coefficients correspond to the predictors in matrix (X).

22. Create a scatter plot that visually represents the relationship between one of the selected features and the target variable 'price.' Include a regression line on the plot to illustrate the linear relationship. Provide insights and interpretations based on the scatter plot.

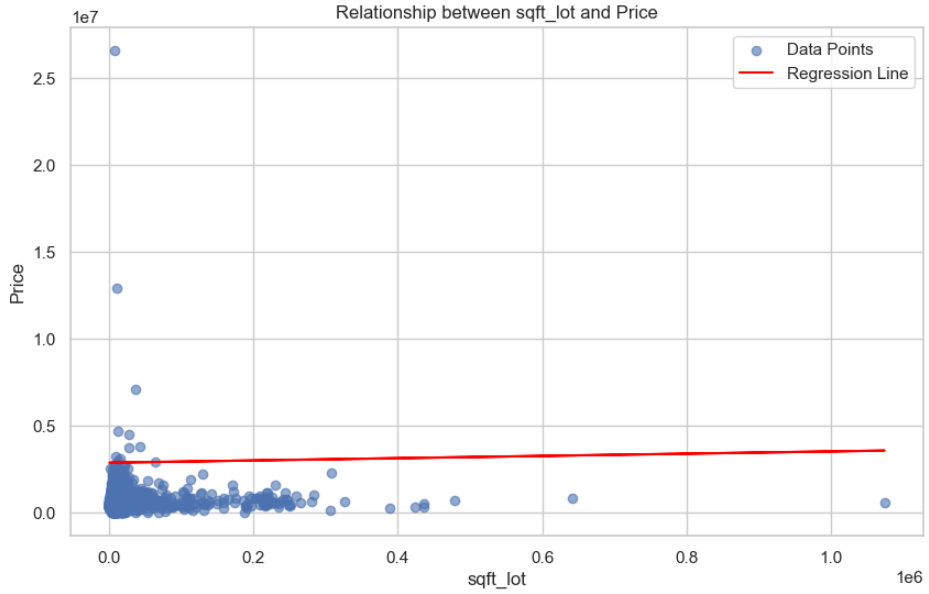
Python Code:

```
# Selecting one of the features ('sqft_lot') and the coefficients for plotting
feature = 'sqft_lot'
feature_values = df[feature]
feature_coefficient = coefficients[selected_features.index(feature) + 1]  # +1 to skip the intercept
intercept = coefficients[0]

# Calculating the regression line
regression_line = intercept + feature_coefficient * feature_values

# Plotting the scatter plot and the regression line
plt.figure(figsize=(10, 6))
plt.scatter(feature_values, Y, label='Data Points', alpha=0.6)
plt.plot(feature_values, regression_line, color='red', label='Regression Line')
plt.title(f'Relationship between {feature} and Price')
plt.xlabel(feature)
plt.ylabel('Price')
plt.legend()
plt.show()
```

Code Execution Screenshot:

22. Create a scatter plot that visually represents the relationship between one of the selected features and the target variable 'price.' Include a regression line on the plot to illustrate the linear relationship. Provide insights and interpretations based on the scatter plot.

In [19]:
```
# Selecting one of the features ('sqft_lot') and the coefficients for plotting
feature = 'sqft_lot'
feature_values = df[feature]
feature_coefficient = coefficients[selected_features.index(feature) + 1]  # +1 to skip the intercept
intercept = coefficients[0]

# Calculating the regression line
regression_line = intercept + feature_coefficient * feature_values

# Plotting the scatter plot and the regression line
plt.figure(figsize=(10, 6))
plt.scatter(feature_values, Y, label='Data Points', alpha=0.6)
plt.plot(feature_values, regression_line, color='red', label='Regression Line')
plt.title(f'Relationship between {feature} and Price')
plt.xlabel(feature)
plt.ylabel('Price')
plt.legend()
plt.show()
```



In the scatter plot above, each point represents a property, with the 'sqft_lot' feature on the x-axis and the price on the y-axis. The red line is the regression line, which represents the linear relationship estimated by the least squares method.

Insights and Interpretations:
Variability:
There is a wide variability in prices for properties with similar 'sqft_lot' values. This indicates that 'sqft_lot' alone may not be a strong predictor of price.

Outliers:
There are some properties with very large 'sqft_lot' values but not necessarily very high prices, and vice versa. These could be considered outliers and might be influencing the slope of the regression line.

Regression Line:
The regression line suggests a positive but weak relationship between 'sqft_lot' and price. This means as the 'sqft_lot' increases, the price tends to increase slightly, but not substantially.

Feature Importance:
Given the weak linear relationship, it might be beneficial to consider other features or combinations of features to better predict the price. Non-linear models or feature transformations could also be explored.

Multivariate Analysis:
The relationship between 'sqft_lot' and price should be analyzed in the context of other features as well. Multivariate regression models that include multiple features can provide a more comprehensive understanding of the factors influencing price.

Explanation:
In the scatter plot above, each point represents a property, with the 'sqft_lot' feature on the x-axis and the price on the y-axis. The red line is the regression line, which represents the linear relationship estimated by the least squares method.

Insights and Interpretations:
Variability: There is a wide variability in prices for properties with similar 'sqft_lot' values. This indicates that 'sqft_lot' alone may not be a strong predictor of price.

Outliers: There are some properties with very large 'sqft_lot' values but not necessarily very high prices, and vice versa. These could be considered outliers and might be influencing the slope of the regression line.

Regression Line: The regression line suggests a positive but weak relationship between 'sqft_lot' and price. This means as the 'sqft_lot' increases, the price tends to increase slightly, but not substantially.

Feature Importance: Given the weak linear relationship, it might be beneficial to consider other features or combinations of features to better predict the price. Non-linear models or feature transformations could also be explored.

Multivariate Analysis: The relationship between 'sqft_lot' and price should be analyzed in the context of other features as well. Multivariate regression models that include multiple features can provide a more comprehensive understanding of the factors influencing price.

CONCLUSION
Our analysis of the real estate dataset revealed several intriguing patterns and relationships. The distribution of property prices was found to be right-skewed, with a few properties having exceptionally high prices. We discovered a significant impact of the year a property was built on its sale price, both through t-tests and ANOVA. The covariance and correlation matrices further highlighted the relationships and dependencies between different features.

However, the presence of multicollinearity suggests caution in interpreting individual feature coefficients in a linear regression model. Visualizing the relationship between 'sqft_lot' and price revealed that while there is a positive trend, other features or combinations of features might be more indicative of the property price.

For stakeholders in the real estate market, this analysis underscores the importance of considering multiple factors when assessing property values. While certain features like the year of construction play a significant role, others might be less influential on their own but crucial in combination. Going forward, more advanced models or feature transformations could be explored to improve predictive accuracy and insights.

It's crucial to remember that while data-driven insights are valuable, they should be combined with domain knowledge and local market conditions for comprehensive decision-making in real estate.

Part 3

Introduction

Financial data captures the core of investor sentiment, firm performance, and broader economic trends, and with its wide range of figures and measures, it holds the keys to the market's pulse. This data, when broken down and examined, can provide a multitude of insights that help stakeholders make strategic decisions. We have a dataset that tracks the stock performance of a large company, and based on our initial observations, we speculate that this may be the data for the tech giant Alphabet Inc.

A daily snapshot of multiple stock variables, such as the opening price, closing price, highest and lowest prices of the day, and volume of shares traded, is provided by this dataset. While each metric offers a glimpse into a specific facet of the stock's performance, collectively, they have the potential to narrate the stock's journey over time, revealing periods of volatility, stability, growth, and contraction.

As we examine this dataset, we have many goals in mind:
1. We seek to uncover patterns and insights through careful study, such as comprehending daily price swings or recognizing significant market movements.
2. To visually depict the stock's trajectory, visual aids such as line charts will be used, improving the accessibility and interpretability of the data.
3. A segmented analysis that breaks down performance by quarter will be made easier by using advanced techniques, such as pivot tables.

As we set out on this analytical journey, our goal is not only to understand the historical trend of the stock we believe to be Alphabet Inc., but also to obtain an angle that could provide insight into its performance going forward.

1. **For the given data, filter the rows that have one or more NULL values**

   Steps followed for this are:

   1. Open the ALPHABET.xlsx file in Excel.
   2. Click on any cell within the dataset to select it.
   3. Go to the ‘Data’ tab on the Excel ribbon.
   4. Click on the ‘Filter’ icon (it looks like a funnel).
   5. Dropdown arrows will appear in each column header. Click on the dropdown arrow for a column you suspect might contain NULL values.
   6. In the filter dropdown, deselect ‘Select All’ and then scroll to the bottom and select ‘(Blanks)’.
   7. Click ‘OK’.
   8. This will filter and show only the rows that have NULL values for the selected column. Repeat this for other columns to identify all rows with NULL values.

   Below are screenshots from the excel file while trying to filter each column individually:

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Volume	
184	Tuesday, May 28, 2019	56.700001		56.655998	56.7075	27300000	
1260							

	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Volume	
11	Monday, September 17, 2018	58.507	58.862		57.802502		
256	Monday, September 9, 2019	60.200001	61		60.220501	29438000	
1260							
1261							

	A	B	C	D	E	F
	Date	Open	High	Low	Close	Volume
	Monday, November 25, 2019	64.959	65.565498	64.906502		20724000

	A	B	C	D	E	F	G
	Date	Open	High	Low	Close	Volume	
	Monday, September 17, 2018	58.507	58.862		57.802502		
	Thursday, March 28, 2019	58.577	58.578251	57.97155	58.4245		

And when these rows are put together, it looks like:

Query) All Edit Links

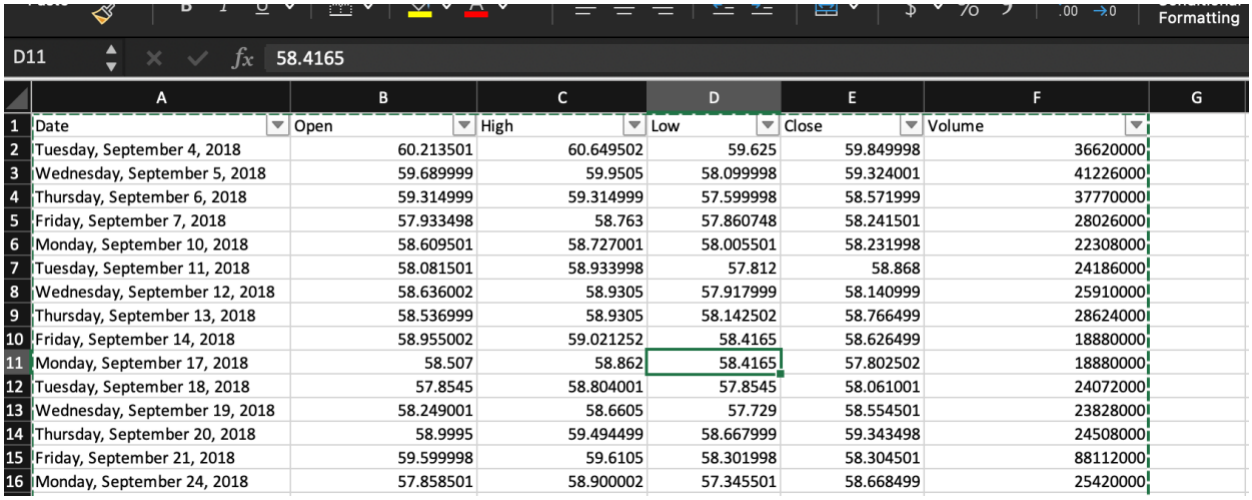
S24 fx

	A	B	C	D	E	F	G	H	I
1									
2		Date	Open	High	Low	Close	Volume		
3		Monday, September 17, 2018	58.507	58.862		57.802502			
4		Thursday, March 28, 2019	58.577	58.578251	57.97155	58.4245			
5		Tuesday, May 28, 2019	56.700001		56.655998	56.7075	27300000		
6		Monday, September 9, 2019	60.200001	61		60.220501	29438000		
7		Monday, November 25, 2019	64.959	65.565498	64.906502		20724000		
8									
9									
10									
11									
12									
13									

2. **Fill these null values using the corresponding data of the most recently occurring previous date. (For Ex: If Close price for 2nd Oct,2023 is missing, you will fill-up using close price from 1st Oct,2023. If that is not available too, then use Close Price from 30th Sept and So). Do the same thing to check for non-numerical values in numerical columns**.

   To fill NULL values:
   1. Select all the NULL values by selecting all the data and clicking on Home Tab -> Find & Select -> Go to Specials -> Blanks

2. Now, all the null/blank cells are selected. Choose any random cell from these selected cells, type '=', press the Up arrow button on the keyboard and press Enter.
3. All Null values will be filled with the values from the above cell. To convert formulas to values in these cells, select the whole data, copy and click on the arrow next to Paste button in the Home Tab.
4. click on 'Paste Values' from the menu.



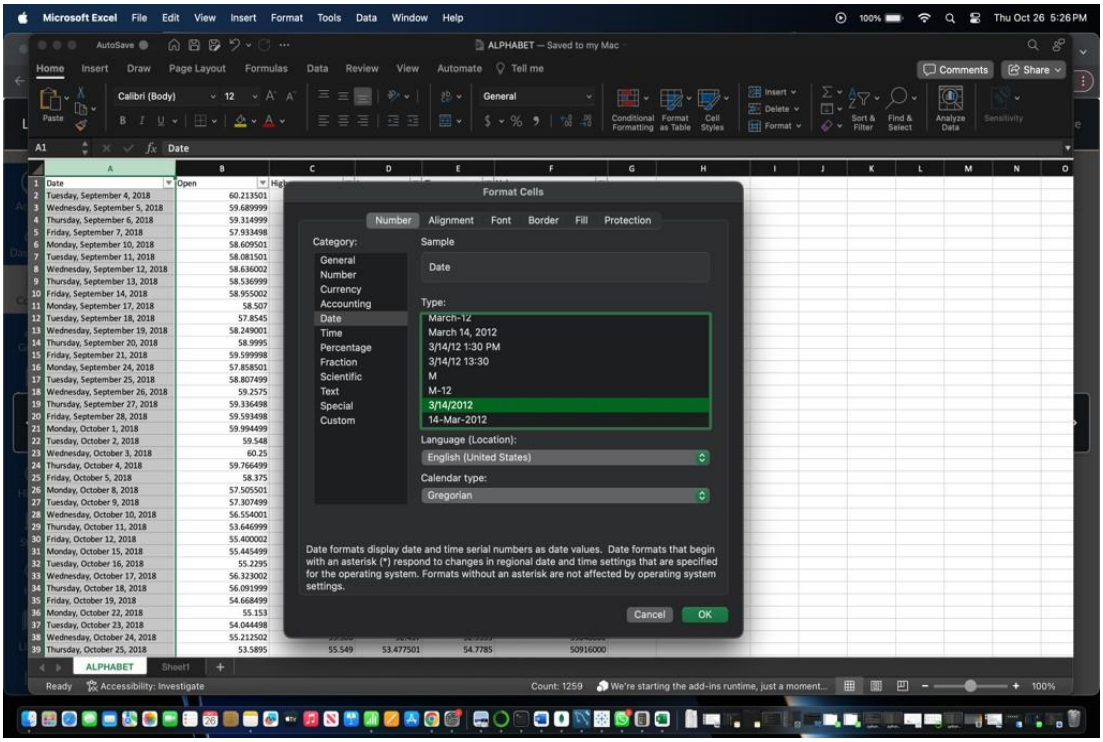
	A	B	C	D	E	F	G
1	Date	Open	High	Low	Close	Volume	
2	Tuesday, September 4, 2018	60.213501	60.649502	59.625	59.849998	36620000	
3	Wednesday, September 5, 2018	59.689999	59.9505	58.099998	59.324001	41226000	
4	Thursday, September 6, 2018	59.314999	59.314999	57.599998	58.571999	37770000	
5	Friday, September 7, 2018	57.933498	58.763	57.860748	58.241501	28026000	
6	Monday, September 10, 2018	58.609501	58.727001	58.005501	58.231998	22308000	
7	Tuesday, September 11, 2018	58.081501	58.933998	57.812	58.868	24186000	
8	Wednesday, September 12, 2018	58.636002	58.9305	57.917999	58.140999	25910000	
9	Thursday, September 13, 2018	58.536999	58.9305	58.142502	58.766499	28624000	
10	Friday, September 14, 2018	58.955002	59.021252	58.4165	58.626499	18880000	
11	Monday, September 17, 2018	58.507	58.862	58.4165	57.802502	18880000	
12	Tuesday, September 18, 2018	57.8545	58.804001	57.8545	58.061001	24072000	
13	Wednesday, September 19, 2018	58.249001	58.6605	57.729	58.554501	23828000	
14	Thursday, September 20, 2018	58.9995	59.494499	58.667999	59.343498	24508000	
15	Friday, September 21, 2018	59.599998	59.6105	58.301998	58.304501	88112000	
16	Monday, September 24, 2018	57.858501	58.900002	57.345501	58.668499	25420000	

As you can see in the above screenshot, The cell which was NULL earlier has been filled with the value from the above cell. Also, these have been converted to values from formulas.

3. Format all the Dates in MM/DD/YYYY format.

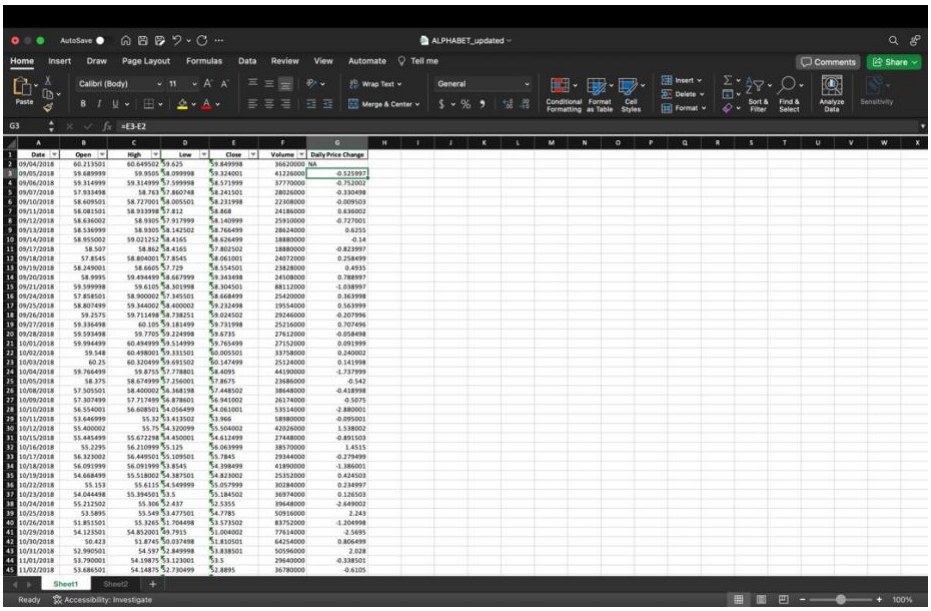
As these dates were initially in Text format, we first had to the entire column -> Data Tab -> Text to Columns and select 'MDY' in the third stage of the wizard. Then, we followed the below steps

1. Select the column containing the dates.
2. Right-click and choose `Format Cells`.
3. Go to the `Number` tab.
4. Choose `Date` from the list on the left.
5. From the type, select `MM/DD/YYYY`.
6. Click `OK`.



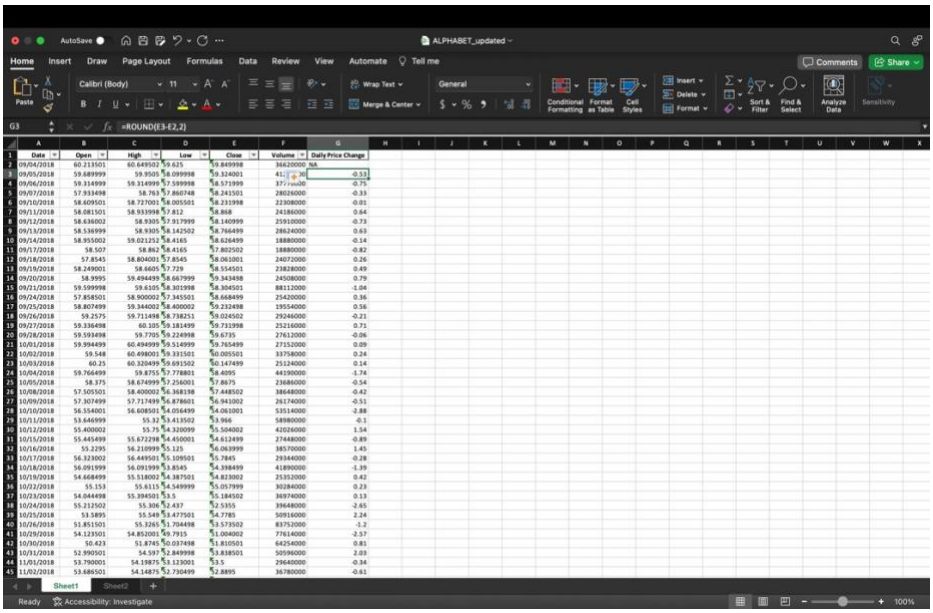
4.Calculate the daily Price Change (Difference between the Closing price of the current day and Closing price of the previous trading day) of the Stock for each trading day

1. Created a new column named ‘Daily Price Change’
2. Selected the second cell under this column, put the formula “=E3-E2” and hit Enter.
3. Dragged this cell until the end of this column.



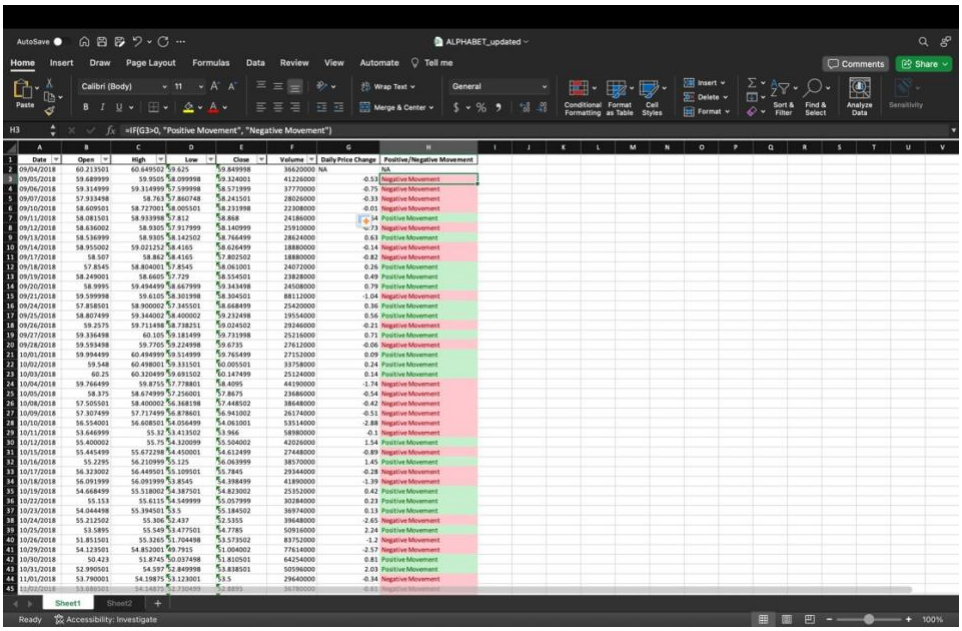
5.Round Up the column from step 4 to the nearest cent.

1. In the same column created for the previous question, entered the formula "=ROUND(E3-E2, 2)".
2. Dragged this formula down for the entire dataset.



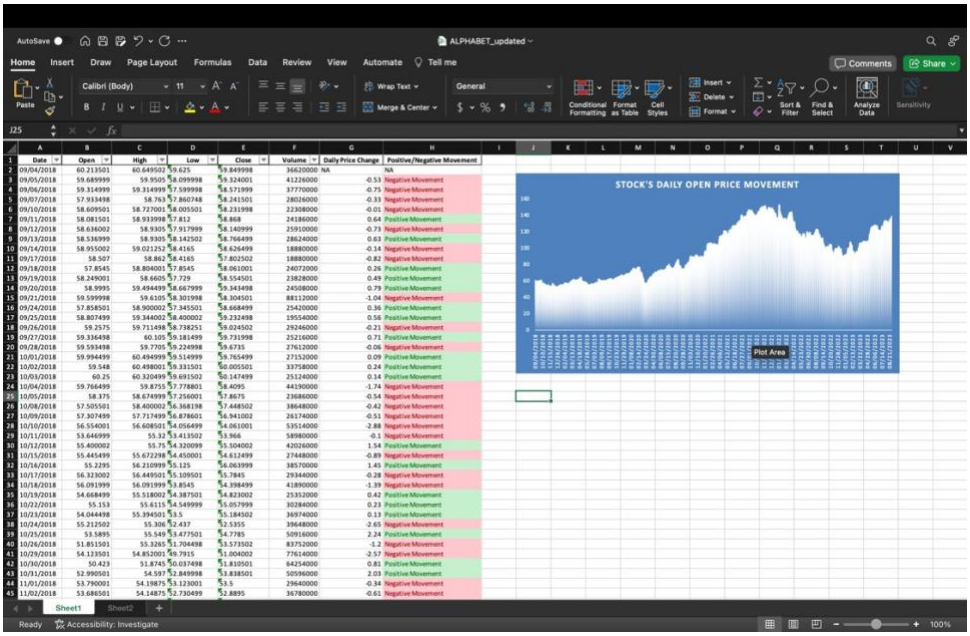
6.Classify the daily price change as either Positive Movement or Neg Movement using IF statement in a new column. Use conditional formatting to highlight the cell with positive movement in green color and negative movement in red.

1. In a new column H, enter the formula "=IF(G3>0, "Positive Movement", "Negative Movement")".
2. Drag this formula down.
3. Used Conditional Formatting (available in the 'Home' tab) to color cells based on their value (Green for Positive Movement and Red for Negative Movement)



7.Create a Line Chart in excel to show the movement of the stock's open price for every day over the last one year.

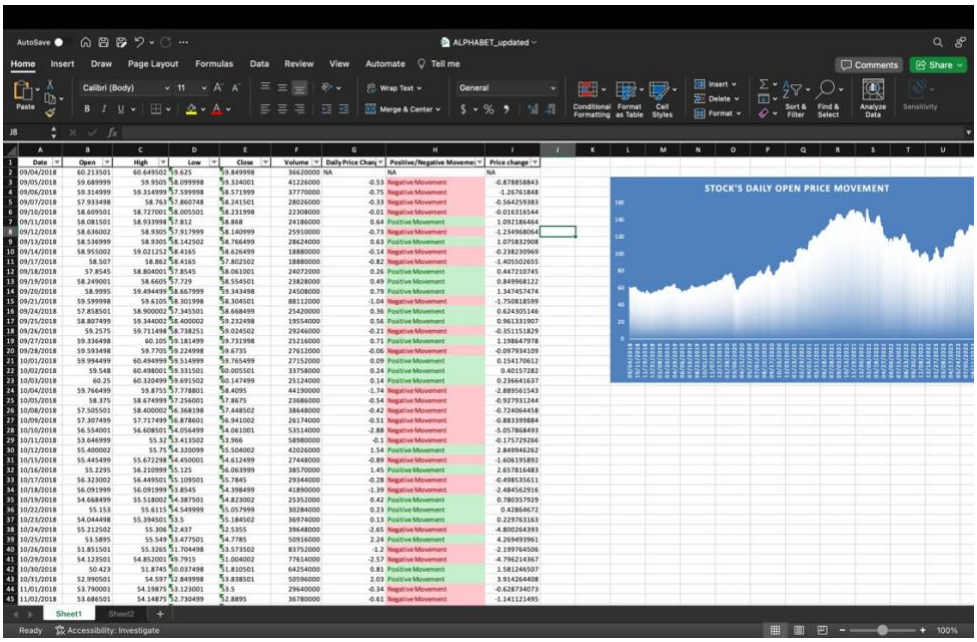
1. Select the date and open price columns.
2. Go to the 'Insert' tab.
3. Choose 'Line Chart'.
4. Choose the type of line chart as per requirement.



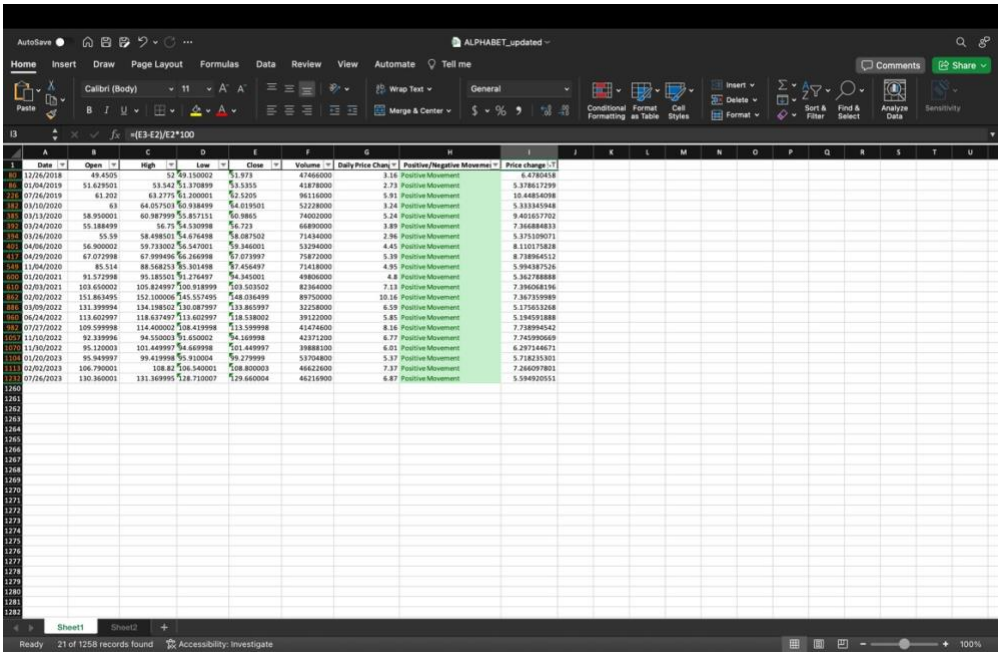
8.Choose only dates that have given more than 5% positive Price Change. Filter these dates and place them in the new sheet called "Big Swings". VLOOKUP the highs and lows for these dates.

Calculate the delta (High Price-Low price) and calculate the potential percentage upside. Sort the data in increasing order of this percentage

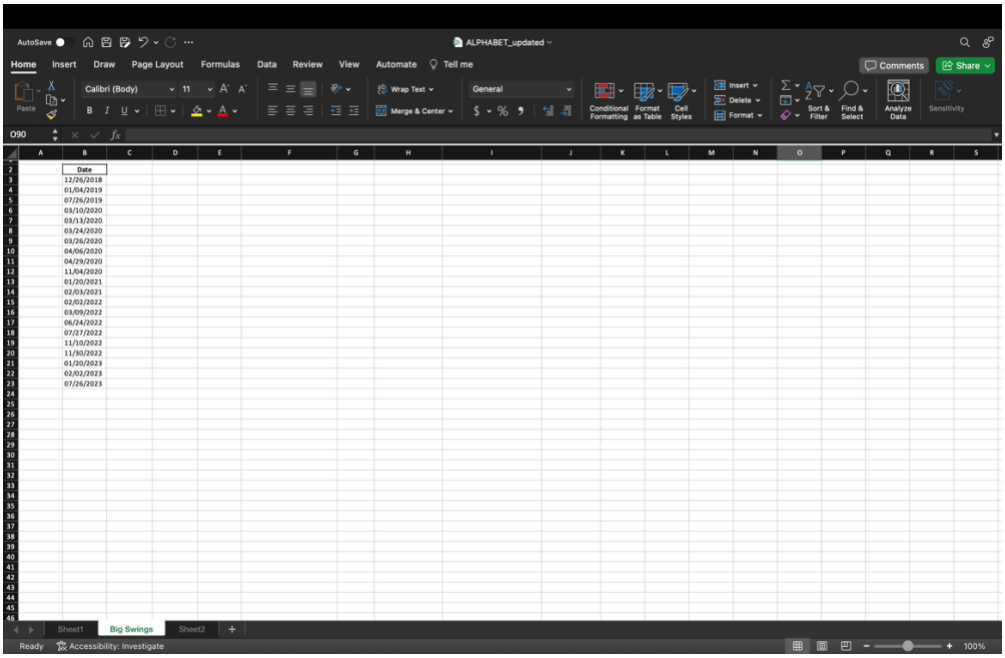
1. Calculate Price Change:
   - Insert a new column next to the 'Close' column and name it 'Price Change %'.
   - In the first cell of this new column, enter the formula `=(E3-E2)/E2*100`, where E3 is today's closing and E2 is the previous day's closing.
   - Drag this formula down to fill the column.



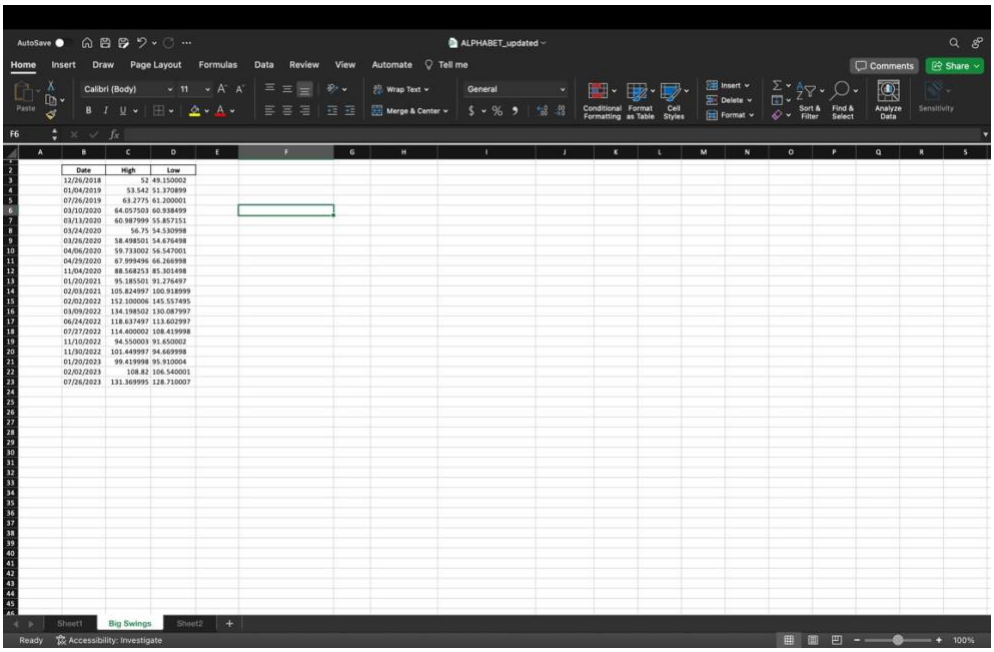
2. Filter for >5% Change:
   - Click on the Data tab and then select 'Filter'.
   - Click on the dropdown arrow of the 'Price Change %' column and set the custom filter to show only values greater than 5.



3. Copy Filtered Data to New Sheet:
   - Copy the filtered dates and paste them into a new sheet called 'Big Swings'.

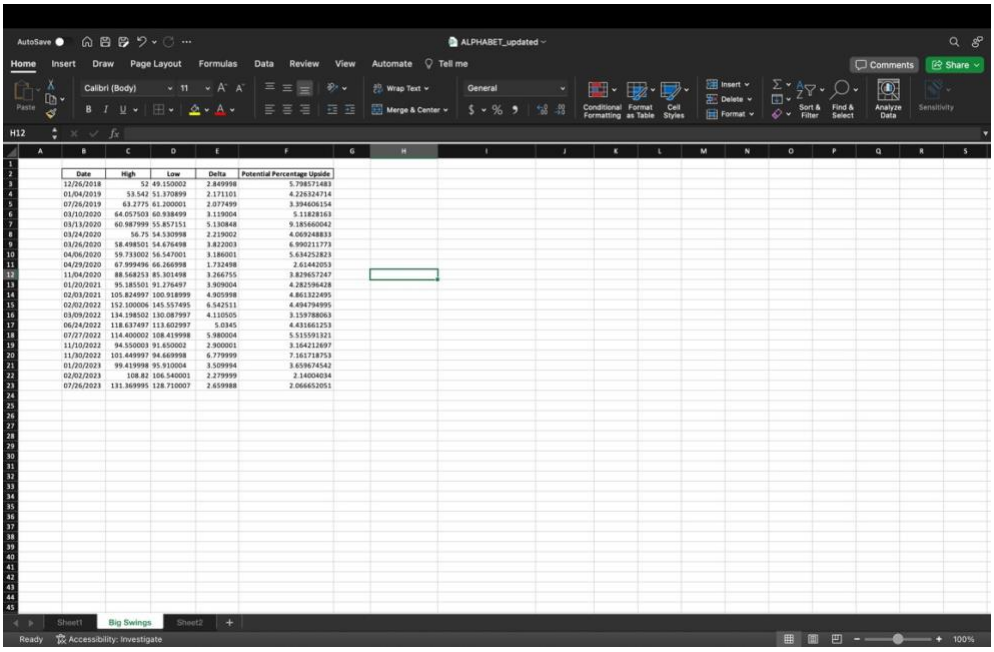


4. VLOOKUP Highs and Lows:
   - In the new 'Big Swings' sheet, use the VLOOKUP function to get the 'High' and 'Low' for each date. As the date is in column B, the formula for highs would be `=VLOOKUP(B3, 'Sheet1'!A:D, 3, FALSE)`. For lows, change the column index to 4.



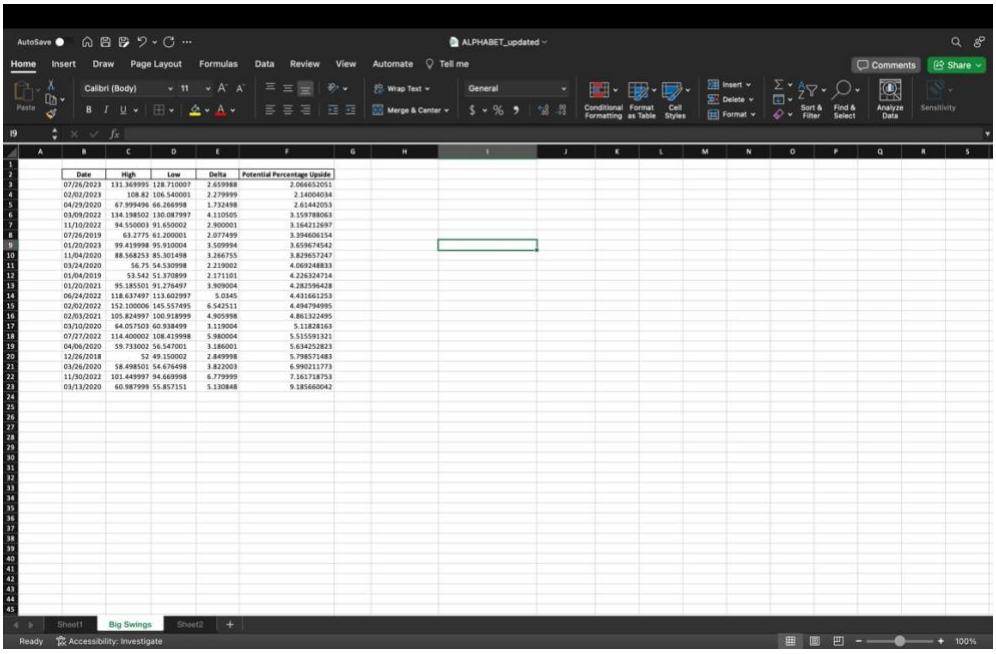
Date	High	Low
12/26/2018	52	49.150002
01/04/2019	53.542	51.370899
07/26/2019	63.2775	61.200001
03/10/2020	64.057503	60.938499
03/13/2020	60.987999	55.857151
03/24/2020	56.75	54.530998
03/26/2020	58.498501	54.676498
04/06/2020	59.733002	56.547001
04/29/2020	67.999496	66.266998
11/04/2020	88.568253	85.301498
01/20/2021	95.185501	91.276497
02/03/2021	105.824997	100.918999
02/02/2022	152.100006	145.557495
03/09/2022	134.198502	130.087997
06/24/2022	118.637497	113.602997
07/27/2022	114.400002	108.419998
11/10/2022	94.550003	91.650002
11/30/2022	101.449997	94.669998
01/20/2023	99.419998	95.910004
02/02/2023	108.82	106.540001
07/26/2023	131.369995	128.710007

5. Calculate Delta and Potential Upside:
   - Calculate the delta by subtracting the low from the high.
   - Calculate the potential percentage upside by dividing the delta by the low and multiplying by 100.



Date	High	Low	Delta	Potential Percentage Upside
12/26/2018	52	49.150002	2.849998	5.798571483
01/04/2019	53.542	51.370899	2.171101	4.226324714
07/26/2019	63.2775	61.200001	2.077499	3.394606154
03/10/2020	64.057503	60.938499	3.119004	5.11828163
03/13/2020	60.987999	55.857151	5.130848	9.185660042
03/24/2020	56.75	54.530998	2.219002	4.069248833
03/26/2020	58.498501	54.676498	3.822003	6.990211773
04/06/2020	59.733002	56.547001	3.186001	5.634252823
04/29/2020	67.999496	66.266998	1.732498	2.6144203
11/04/2020	88.568253	85.301498	3.266755	3.829657247
01/20/2021	95.185501	91.276497	3.909004	4.282596428
02/03/2021	105.824997	100.918999	4.905998	4.861322495
02/02/2022	152.100006	145.557495	6.542511	4.494794995
03/09/2022	134.198502	130.087997	4.110505	3.159788063
06/24/2022	118.637497	113.602997	5.0345	4.431661253
07/27/2022	114.400002	108.419998	5.980004	5.515591321
11/10/2022	94.550003	91.650002	2.900001	3.164212697
11/30/2022	101.449997	94.669998	6.779999	7.161718753
01/20/2023	99.419998	95.910004	3.509994	3.659674542
02/02/2023	108.82	106.540001	2.279999	2.14004034
07/26/2023	131.369995	128.710007	2.659988	2.066652051

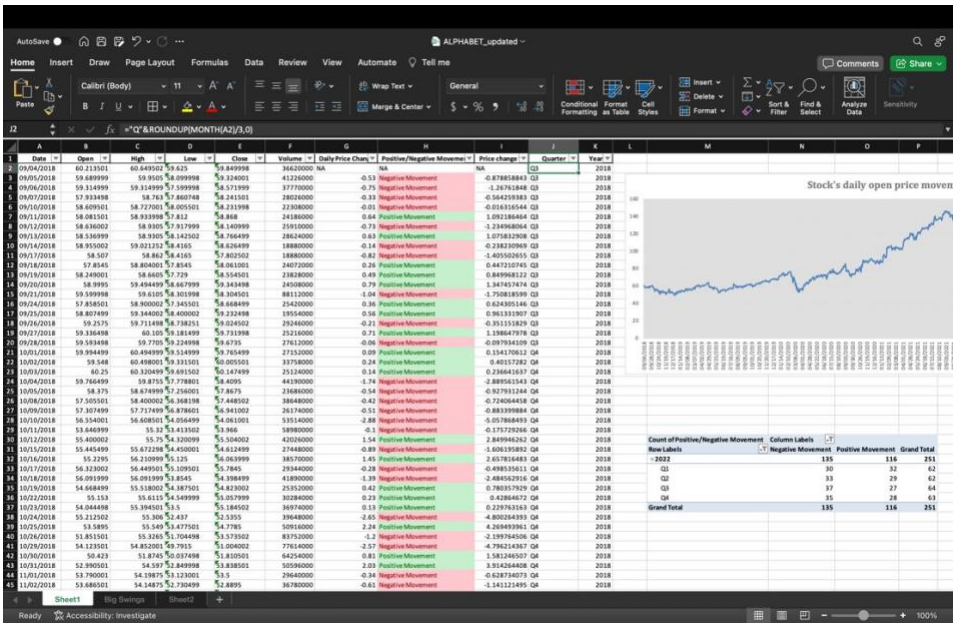
6. Sort Data:
   - Highlight the data and click on the Data tab. Choose 'Sort' and sort by the potential percentage upside in increasing order.



Date	High	Low	Delta	Potential Percentage Upside
07/26/2023	131.369995	128.710007	2.659988	2.066652051
02/02/2023	108.82	106.540001	2.279999	2.14004034
04/29/2020	67.999496	66.266998	1.732498	2.61442053
03/09/2022	134.198502	130.087997	4.110505	3.159788063
11/10/2022	94.550003	91.650002	2.900001	3.164212697
07/26/2019	63.2775	61.200001	2.077499	3.394606154
01/20/2023	99.419998	95.910004	3.509994	3.659674542
11/04/2020	88.568253	85.301498	3.266755	3.829657247
03/24/2020	56.75	54.530998	2.219002	4.069248833
01/04/2019	53.542	51.370899	2.171101	4.226324714
01/20/2021	95.185501	91.276497	3.909004	4.282596428
06/24/2022	118.637497	113.602997	5.0345	4.431661253
02/02/2022	152.100006	145.557495	6.542511	4.494794995
02/03/2021	105.824997	100.918999	4.905998	4.861322495
03/10/2020	64.057503	60.938499	3.119004	5.11828163
07/27/2022	114.400002	108.419998	5.980004	5.515591321
04/06/2020	59.733002	56.547001	3.186001	5.634252823
12/26/2018	52	49.150002	2.849998	5.798571683
03/26/2020	58.498501	54.676498	3.822003	6.990211773
11/30/2022	101.449997	94.669998	6.779999	7.161718753
03/13/2020	60.987999	55.857151	5.130848	9.185660042

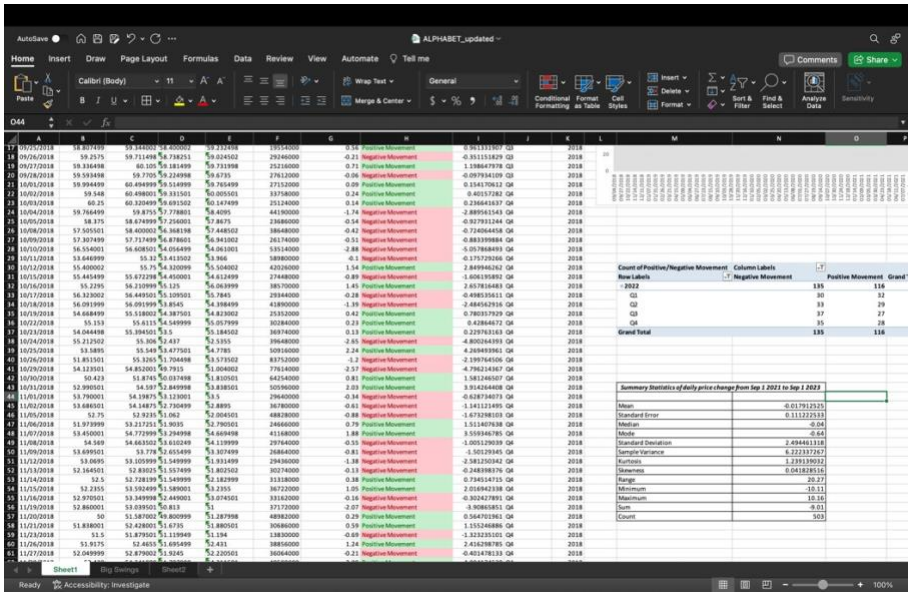
9.Create a Pivot Table to show the division of the number of positive and negative return days by each quarter of 2022.

1. Create two new columns, one for Quarter with '="Q"&ROUNDUP(MONTH(A2)/3,0)' as the formula and one for Year with '=YEAR(A2)'
2. Select the entire table
3. Go to the Insert tab and click on PivotTable.
4. Arrange fields to show the division of positive and negative return days by each quarter such that the Quarters and Years appear as Rows and Number of Positive and Negative Movements appears in the columns.
5. Filter out the quarters from the year 2022



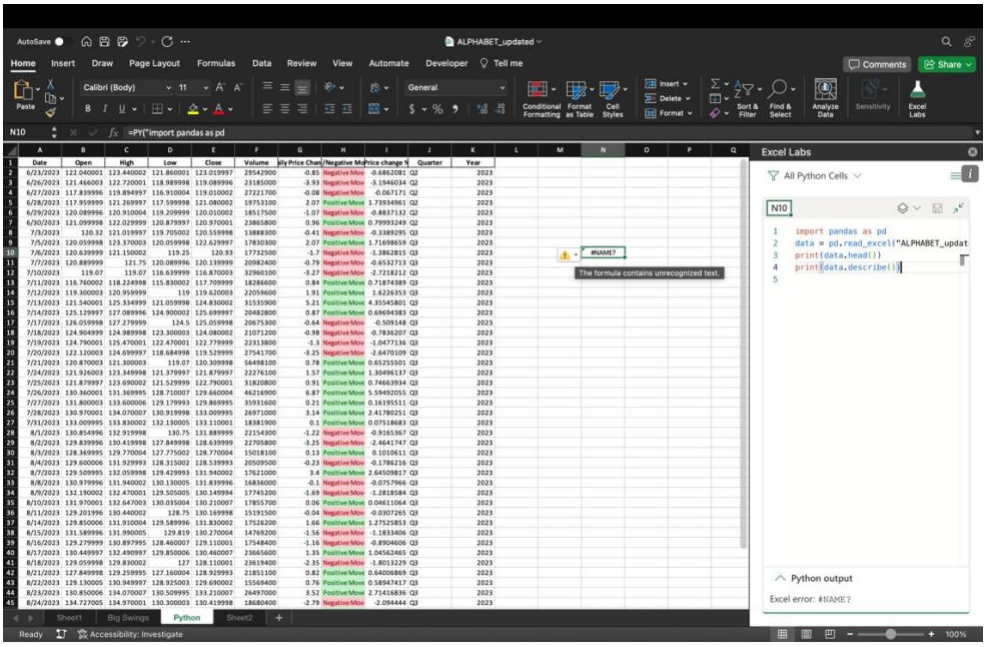
10.Calculate the summary statistics of the daily price change over the last two years using data analysis features in excel.

1. Go to Data Tab -> Click on ‘Data Analysis’
2. Select Descriptive Statistics from the drop down.
3. Select the input range. Here, we have taken daily price change from September 1st 2021 to September 1st 2023.
4. Enable Labels on First Row checkbox and choose the Output Range where the table has to appear as per requirement.
5. Enable Summary Statistics checkbox and click OK. The Statistics table will appear.
6. Change the heading of the table if necessary



11. Copy paste the data from the last 50 trade dates onto a new sheet called “Python”. Use the option to run python code on MS excel to create a data frame of this data. Use .describe() and .head() on this data frame.

On browsing the internet about Excel’s capability to execute python code, we realized that this is currently a part of the Beta version for windows and not for Mac. However, we did try to run python code on Excel Labs, an Add-In for Excel available on Microsoft office store. This comes with a python editor. On typing a code to create a dataframe as per question, we ran into an error as seen in the screenshot below.



Conclusion

A comprehensive understanding of Alphabet Inc.'s stock performance across various time horizons is offered by the careful examination of its dataset. Using "Sheet1" as a starting point, which provides a comprehensive overview of the stock's historical data, we track the daily fluctuations in stock prices as represented by metrics such as Open, High, Low, and Close prices. Over time, the foundational data and the volume of shares traded provide insight into the stock's liquidity and investor interest. Traders and short-term investors can benefit greatly from our ability to measure the stock's short-term volatility by monitoring the daily price change and its direction.

The "Big Swings" sheet delves deeper into events by highlighting instances of significant price movements. These occurrences, which exhibit a positive Price Change of over 5%, are essential for comprehending periods of elevated investor sentiment, which may be attributed to news specific to the company or to general market dynamics. By keeping an eye out for these kinds of big movements, one can identify times when there may be an overvaluation or undervaluation, providing chances for strategic investing or trading.

Finally, the "Python" sheet offers a detailed overview of the stock's most recent performance. With data spanning the previous 50 trading days, this sheet serves as a benchmark for the current state of the market's perception of Alphabet Inc. Daily price changes are included, and their positive/negative movements are categorized to give a clear indicator of the stock's current momentum.

In conclusion, the comprehensive analysis of the stock data for Alphabet Inc. highlights the importance of layered and methodical analysis. Through an analysis of extensive historical data, the identification of notable price fluctuations, and a comprehension of current market sentiment, stakeholders can obtain a comprehensive understanding of the stock's performance. In the dynamic world of stock markets, having such a strong understanding—founded in insights gleaned from data—opens the door to more strategic and well-informed decision-making.