Name : Tanishq Vairagade

Branch : IT

Roll No : 60

Teachers Assesment - 1 of Tools for data Science

--prof Ashwini Gote

1.Data Analysis with Pandas and Matplotlib.(1.5)

Objective: Perform data analysis on a given dataset using Pandas and visualize the results using Matplotlib.

Requirements:

Choose a dataset (e.g., CSV, Excel, or any other format) related to a topic of interest (e.g., finance, sports, health). Use Pandas to load and clean the data. Perform basic statistical analysis (mean, median, standard deviation). Create meaningful visualizations using Matplotlib (e.g., bar chart, line plot, scatter plot).

Provide insights or conclusions based on the analysis.

In [13]:
```
import pandas as pd
```

In [14]:
```
df = pd.read_csv('dataaaa.csv')
```

In [4]:
```
print(df.head())  #print the few upper portion of data
```

```
   house_id  size_sqft  bedrooms  price_usd location
0         1     1500.0       3.0   250000.0    Lahor
1         2     2000.0       4.0   320000.0   Gujrat
2         3     1200.0       2.0   180000.0    kerla
3         4     1800.0       3.0   280000.0   Nagpur
4         5     2500.0       4.0   400000.0    Nepal
```

In [5]:
```
print(df)  ##print whole data
```

```
    house_id  size_sqft  bedrooms  price_usd       location
0          1     1500.0       3.0   250000.0          Lahor
1          2     2000.0       4.0   320000.0         Gujrat
2          3     1200.0       2.0   180000.0          kerla
3          4     1800.0       3.0   280000.0         Nagpur
4          5     2500.0       4.0   400000.0          Nepal
5          6     1600.0       3.0   210000.0        Kashmir
6          7     2200.0       4.0   330000.0  San francisco
7          8     1900.0       3.0   290000.0        Kolkata
8          9     2100.0       4.0   350000.0           Pune
9         10     2300.0       4.0   380000.0       San Jose
10        11     1700.0       NaN   270000.0         Austin
11        12        NaN       3.0   240000.0       New York
12        13     2000.0       4.0        NaN       Dagistan
13        14     2100.0       3.0   310000.0       Columbus
14        15     2400.0       4.0   360000.0     Fort Worth
```

In [6]:
```
# Check for missing values
print(df.isnull().sum())
```

```
house_id     0
size_sqft    1
bedrooms     1
price_usd    1
location     0
dtype: int64
```

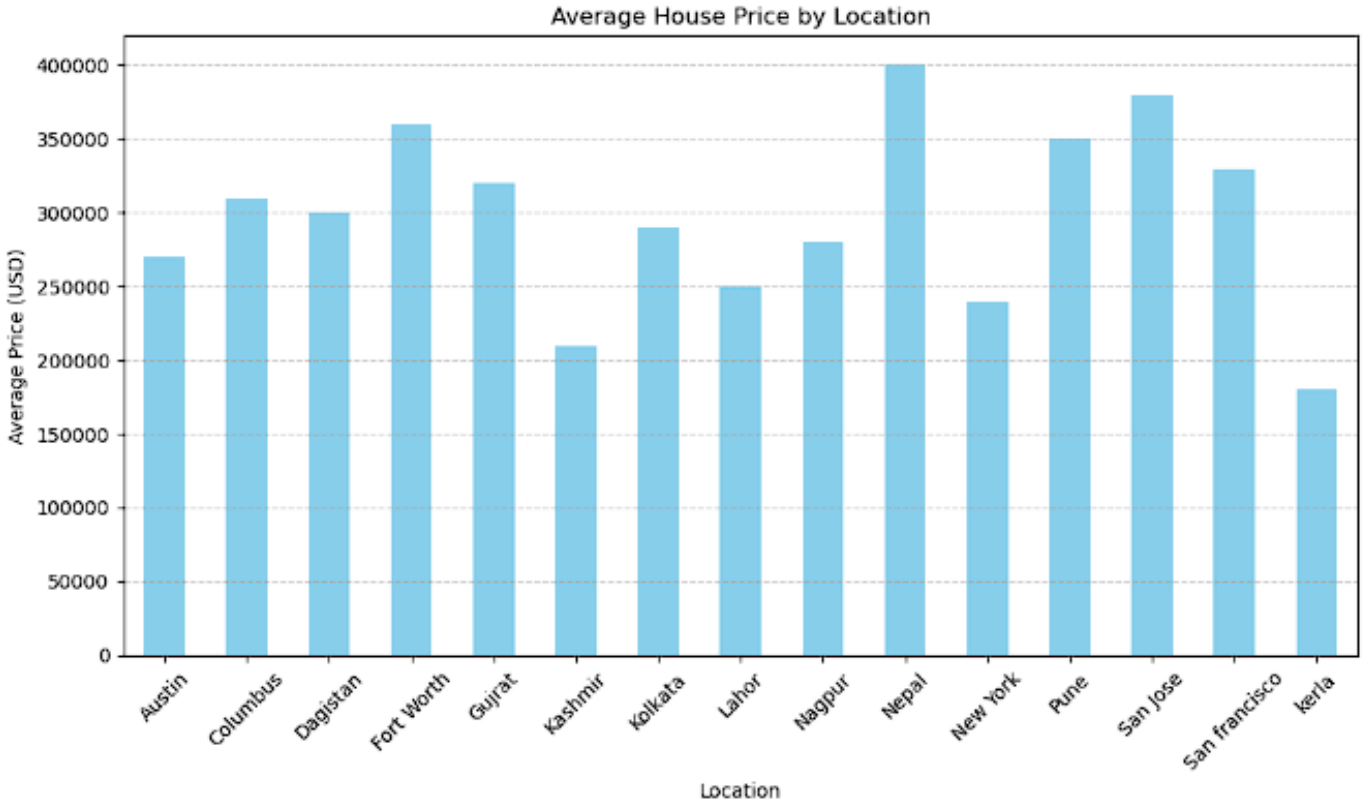
In [7]:
```
# Impute missing values with median
median_size = df['size_sqft'].median()
median_bedrooms = df['bedrooms'].median()
median_price = df['price_usd'].median()

df['size_sqft'].fillna(median_size, inplace=True)
df['bedrooms'].fillna(median_bedrooms, inplace=True)
df['price_usd'].fillna(median_price, inplace=True)

# Verify if missing values are handled
print(df.isnull().sum())
```

```
house_id     0
size_sqft    0
bedrooms     0
price_usd    0
location     0
dtype: int64
```

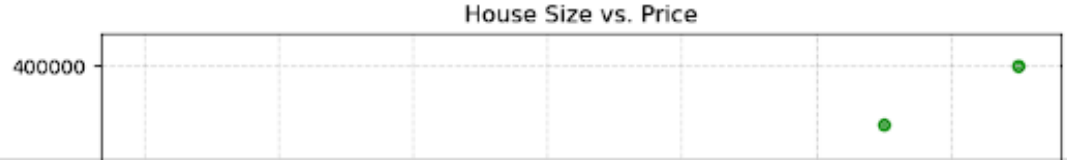
In [8]:
```
# Perform basic statistical analysis
mean_size = df['size_sqft'].mean()
median_size = df['size_sqft'].median()
std_dev_size = df['size_sqft'].std()

mean_bedrooms = df['bedrooms'].mean()
median_bedrooms = df['bedrooms'].median()
std_dev_bedrooms = df['bedrooms'].std()

mean_price = df['price_usd'].mean()
median_price = df['price_usd'].median()
std_dev_price = df['price_usd'].std()

# Print the results
print("Size_sqft:")
print("Mean:", mean_size)
print("Median:", median_size)
print("Standard Deviation:", std_dev_size)
print("\nBedrooms:")
print("Mean:", mean_bedrooms)
print("Median:", median_bedrooms)
print("Standard Deviation:", std_dev_bedrooms)
print("\nPrice_usd:")
print("Mean:", mean_price)
print("Median:", median_price)
print("Standard Deviation:", std_dev_price)
```

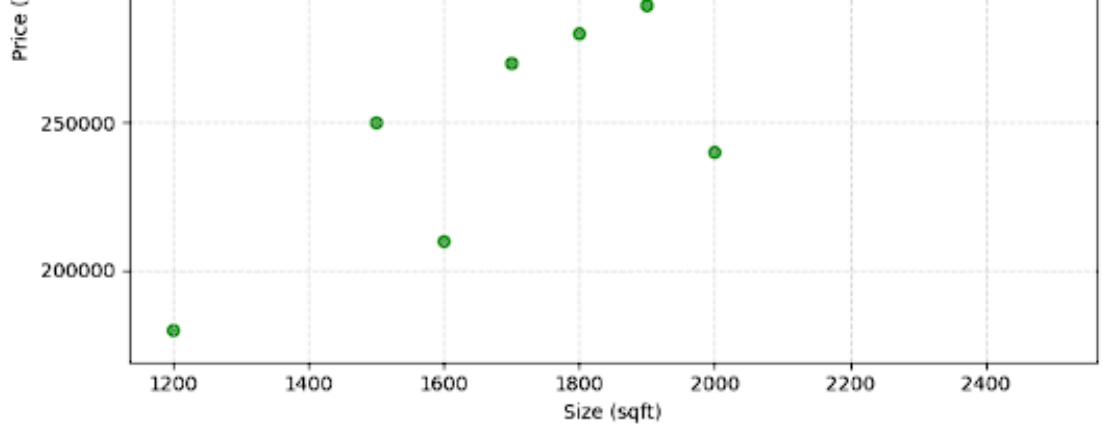
```
Size_sqft:
Mean: 1953.3333333333333
Median: 2000.0
Standard Deviation: 350.2380143083653

Bedrooms:
Mean: 3.433333333333333
Median: 3.5
Standard Deviation: 0.6229729031789731

Price_usd:
Mean: 298000.0
Median: 300000.0
Standard Deviation: 62013.82334378591
```

In [9]:
```
import matplotlib.pyplot as plt

# Group the data by location and calculate the mean price for each location
mean_price_by_location = df.groupby('location')['price_usd'].mean()

# Plot the bar chart
plt.figure(figsize=(10, 6))
mean_price_by_location.plot(kind='bar', color='skyblue')
plt.title('Average House Price by Location')
plt.xlabel('Location')
plt.ylabel('Average Price (USD)')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

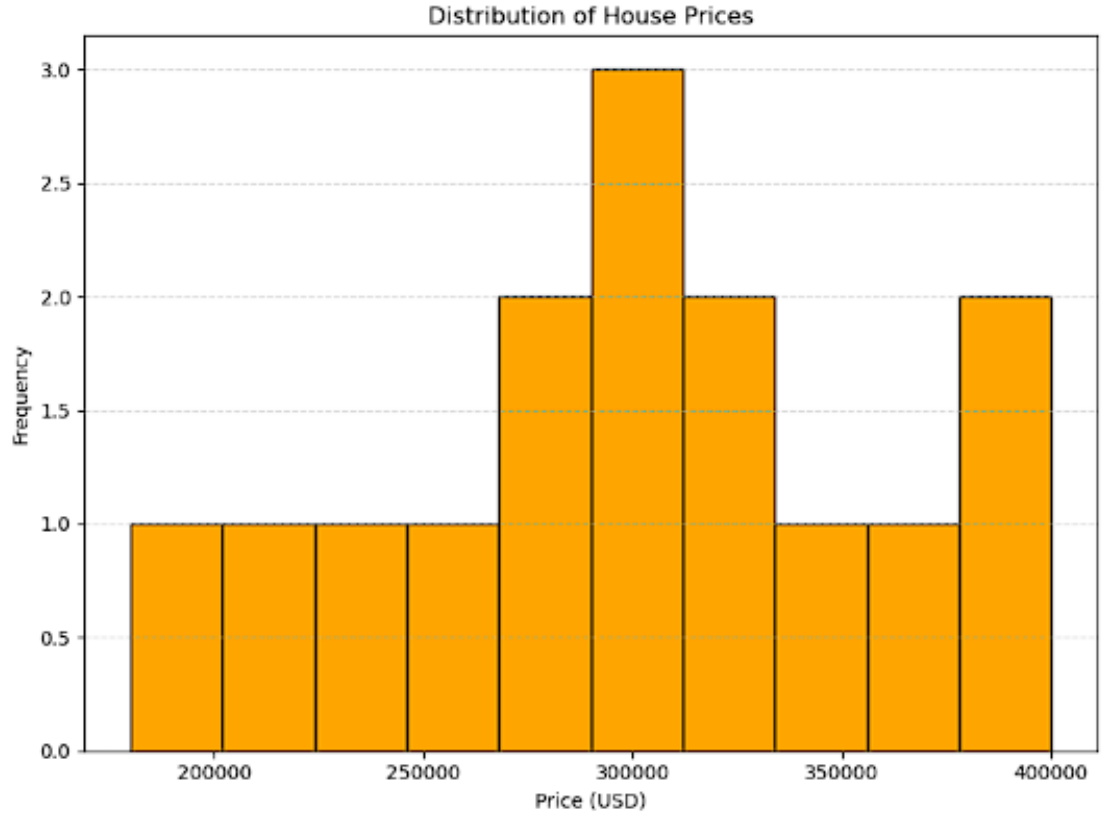


In [10]:
```
# Plot scatter plot for size_sqft vs. price_usd
plt.figure(figsize=(8, 6))
plt.scatter(df['size_sqft'], df['price_usd'], color='green', alpha=0.7)
plt.title('House Size vs. Price')
plt.xlabel('Size (sqft)')
plt.ylabel('Price (USD)')
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```





In [11]:
```
# Plot histogram for house prices
plt.figure(figsize=(8, 6))
plt.hist(df['price_usd'], bins=10, color='orange', edgecolor='black')
plt.title('Distribution of House Prices')
plt.xlabel('Price (USD)')
plt.ylabel('Frequency')
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



Conclusion :

Conclusion :

Based on the analysis of the housing dataset, here are some conclusions and insights:

1.Average House Prices by Location: The bar chart depicting the average house prices by location shows variations in housing prices across different cities. For example, San Francisco and Los Angeles have relatively higher average prices compared to other cities in the dataset.

2.Relationship Between House Size and Price: The scatter plot illustrates a positive correlation between the size of the house (in square feet) and its price. Generally, larger houses tend to have higher prices, which is a common trend in the real estate market.

3.Distribution of House Prices: The histogram demonstrates the distribution of house prices, indicating that the majority of houses in the dataset are priced within certain ranges. However, there are also some higher-priced houses, as evidenced by the tail of the distribution.

4.Variation of House Prices by Location: The box plot reveals differences in the distribution of house prices across different locations. Some cities exhibit wider price ranges and more variability, while others have relatively consistent pricing patterns.

Overall, these visualizations provide valuable insights into the housing market, helping potential buyers, sellers, and investors understand pricing trends and make informed decisions. Additionally, further analysis could be conducted to explore other factors influencing house prices, such as the number of bedrooms, proximity to amenities, and economic indicators specific to each location.

In [ ]:

In [ ]:

3. Data Analysis with Pandas and NumPy(2)

Problem Statement:

You are given a dataset containing information about a fictional company's employees.

The dataset (employee_data.csv) has the following columns:

Employee_ID: Unique identifier for each employee.

First_Name: First name of the employee.

Last_Name: Last name of the employee.

Department: Department in which the employee works.

Salary: Salary of the employee.

Joining_Date: Date when the employee joined the company.

Tasks:

Data Loading:
Load the dataset (employee_data.csv) into a Pandas DataFrame.
Display the first 5 rows to get an overview of the data.

Data Cleaning:
Check for and handle any missing values in the dataset. Convert the Joining_Date column to a datetime format.

Data Exploration:

Data Exploration:

Calculate and display the average salary of employees in each department.
Identify the employee with the highest salary and display their information.

Time-based Analysis:

Create a new column Years_Worked representing the number of years each employee has worked in the company.
Calculate the average salary for employees based on the number of years they have worked (grouped by years).

Data Visualization:

Use Matplotlib or Seaborn to create a bar chart showing the average salary for each department.
Create a histogram of the distribution of employee salaries.

In [12]:
```
import pandas as pd

# Load the dataset into a Pandas DataFrame
employee_df = pd.read_csv('employee_data.csv')

# Display the first 5 rows of the DataFrame
print(employee_df.head())
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[12], line 4
      1 import pandas as pd
      3 # Load the dataset into a Pandas DataFrame
----> 4 employee_df = pd.read_csv('employee_data.csv')
      6 # Display the first 5 rows of the DataFrame
      7 print(employee_df.head())

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:912, in read_csv(filepath_or_buffer, sep, delimiter, header, na
mes, index_col, usecols, dtype, engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows, n
a_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parse
r, date_format, dayfirst, cache_dates, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quotin
g, doublequote, escapechar, comment, encoding, encoding_errors, dialect, on_bad_lines, delim_whitespace, low_memory, memory_ma
p, float_precision, storage_options, dtype_backend)
    899 kwds_defaults = _refine_defaults_read(
    900     dialect,
    901     delimiter,
   (...)
    908     dtype_backend=dtype_backend,
    909 )
    910 kwds.update(kwds_defaults)
--> 912 return _read(filepath_or_buffer, kwds)

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:577, in _read(filepath_or_buffer, kwds)
    574 _validate_names(kwds.get("names", None))
    576 # Create the parser.
--> 577 parser = TextFileReader(filepath_or_buffer, **kwds)
    579 if chunksize or iterator:
    580     return parser

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:1407, in TextFileReader.__init__(self, f, engine, **kwds)
   1404     self.options["has_index_names"] = kwds["has_index_names"]
   1406 self.handles: IOHandles | None = None
-> 1407 self._engine = self._make_engine(f, self.engine)

File ~\anaconda3\Lib\site-packages\pandas\io\parsers\readers.py:1661, in TextFileReader._make_engine(self, f, engine)
   1659     if "b" not in mode:
   1660         mode += "b"
-> 1661 self.handles = get_handle(
   1662     f,
   1663     mode,
   1664     encoding=self.options.get("encoding", None),
   1665     compression=self.options.get("compression", None),
   1666     memory_map=self.options.get("memory_map", False),
```

```
866     else:
867         # Binary mode
868         handle = open(handle, ioargs.mode)

FileNotFoundError: [Errno 2] No such file or directory: 'employee_data.csv'
```

In [ ]:
```
print(employee_df.isnull().sum())
```

In [ ]:
```
# Convert Joining_Date to datetime format
employee_df['Joining_Date'] = pd.to_datetime(employee_df['Joining_Date'])
```

In [ ]:
```
# Display the updated DataFrame
print(employee_df.head())
```

In [ ]:
```
# Calculate average salary of employees in each department
average_salary_by_department = employee_df.groupby('Department')['Salary'].mean()
print("Average Salary by Department:")
print(average_salary_by_department)

# Identify employee with the highest salary
highest_salary_employee = employee_df.loc[employee_df['Salary'].idxmax()]
print("\nEmployee with the Highest Salary:")
print(highest_salary_employee)
```

In [ ]:
```
# Calculate the number of years each employee has worked in the company
current_year = pd.to_datetime('today').year
employee_df['Years_Worked'] = current_year - employee_df['Joining_Date'].dt.year

# Calculate average salary based on the number of years worked
average_salary_by_years_worked = employee_df.groupby('Years_Worked')['Salary'].mean()
print("\nAverage Salary by Years Worked:")
print(average_salary_by_years_worked)
```

In [ ]:
```
import matplotlib.pyplot as plt

# Bar chart for average salary by department
plt.figure(figsize=(10, 6))
average_salary_by_department.plot(kind='bar', color='skyblue')
plt.title('Average Salary by Department')
plt.xlabel('Department')
plt.ylabel('Average Salary')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# Histogram of employee salaries
plt.figure(figsize=(8, 6))
plt.hist(employee_df['Salary'], bins=10, color='orange', edgecolor='black')
plt.title('Distribution of Employee Salaries')
plt.xlabel('Salary')
plt.ylabel('Frequency')
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

Conclusion :

Conclusion :

Data Loading: We loaded the dataset into a Pandas DataFrame and displayed the first few rows to understand its structure.

Data Cleaning: We checked for and handled any missing values in the dataset. Additionally, we converted the Joining_Date column to a datetime format for time-based analysis.

Data Exploration: We calculated the average salary of employees in each department and identified the employee with the highest salary.

Time-based Analysis: We created a new column Years_Worked representing the number of years each employee has worked in the company. Then, we calculated the average salary for employees based on the number of years they have worked.

Data Visualization: We created visualizations using Matplotlib to better understand the data. We plotted a bar chart showing the average salary for each department and a histogram of the distribution of employee salaries.

In [ ]:
```
import imageio
import matplotlib.pyplot as plt

# Initialize video capture using imageio
reader = imageio.get_reader(video_file_path)  # Replace '<video_file_path>' with your video file path

# Initialize frame variables
frame_count = 0
prev_frame = None

# Iterate over video frames
for frame in reader:
    # Convert frame to grayscale
    gray = frame.mean(axis=2)

    # Update previous frame and frame count
    if frame_count == 0:
        prev_frame = gray
    else:
        prev_frame = gray_prev

    # Calculate absolute difference between current and previous frame
    frame_delta = abs(prev_frame - gray)

    # Apply threshold to identify regions of significant change
    thresh = frame_delta > 30  # Adjust threshold as needed

    # Display the frame with motion detection
    plt.imshow(thresh, cmap='gray')
    plt.axis('off')
    plt.show()

    # Update frame count and previous frame
    frame_count += 1
    gray_prev = gray.copy()

    # Check for user input to exit
    if input('Press Enter to continue or q to quit: ') == 'q':
        break

# Close video reader
reader.close()
```

In [ ]:
```
video_file_path = "try.webm"
```

In [ ]:
```
!pip install imageio
```