Lab Report

Data Visualization Techniques

Submitted By

Name: Shashank Ranjan

Roll Number: 2354002

BATCH

M. Tech. (Data Science)

Department of Computer Science & Engineering
National Institute of Technology Patna (Bihar)
(JAN-JUN 2025)

List of experiments

S. No.	Assignment	Pre-requisites	Instructor Signature
1	Assignment 1	Python	
2	Assignment 2	Python	
3	Assignment 3	Python	
4	Assignment 4	Python	
5	Assignment 5	Python	
6	Assignment 6	Python	
7	Assignment 7	Python	
8	Assignment 8	Python	
9	Assignment 9	Python	
10	Assignment 10	Python	
11	Assignment 11	Python	

Lab Experiment 1

Objectives:

1. Get familiar with basic supported data structures including list, tuple, dictionary, etc.
2. Get familiar with basic python packages numpy and pandas packages.

Consider the given automobile data and do the followings:

1. Compute total records in the dataset

```
path = 'new_data.csv'
df = pd.read_csv(path)
# Get the total number of records (rows) in the DataFrame
total_records = len(df)
# Print or use the total_records as needed
print(f'Total Records: {total_records}')
```

```
Total Records: 205
```

2. Compute number of attributes and its naming list in the dataset.

```
df = pd.read_csv('new_data.csv')
num_attri = df.shape[1]
attri_names = df.columns.tolist()
print("no. of attri: ", num_attri)
print("List of attri: ", attri_names)
print(attri_names)
```

```
no. of attri:  27
List of attri:  ['Unnamed: 0', 'symboling', 'normalized-losses', 'make', 'fuel-type', 'aspirat
ion', 'num-of-doors', 'body-style', 'drive-wheels', 'engine-location', 'wheel-base', 'length',
'width', 'height', 'curb-weight', 'engine-type', 'num-of-cylinders', 'engine-size', 'fuel-syst
em', 'bore', 'stroke', 'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg', 'highway-mp
g', 'price']
['Unnamed: 0', 'symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration', 'num-of-do
ors', 'body-style', 'drive-wheels', 'engine-location', 'wheel-base', 'length', 'width', 'heigh
t', 'curb-weight', 'engine-type', 'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 's
troke', 'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'price']
```

3. Display top 25 records in the dataset.

```
df = pd.read_csv('new_data.csv')
top_25 = df.head(25)
last_25 = df.tail(25)
print("Top 25 records are")
print(top_25)
```

```
Top 25 records are
    Unnamed: 0  symboling normalized-losses         make fuel-type aspiration
0            0          3                 ?  alfa-romero       gas        std
1            1          3                 ?  alfa-romero       gas        std
2            2          1                 ?  alfa-romero       gas        std
3            3          2               164         audi       gas        std
4            4          2               164         audi       gas        std
5            5          2                 ?         audi       gas        std
6            6          1               158         audi       gas        std
7            7          1                 ?         audi       gas        std
8            8          1               158         audi       gas      turbo
9            9          0                 ?         audi       gas      turbo
10          10          2               192          bmw       gas        std
11          11          0               192          bmw       gas        std
12          12          0               188          bmw       gas        std
13          13          0               188          bmw       gas        std
14          14          1                 ?          bmw       gas        std
15          15          0                 ?          bmw       gas        std
16          16          0                 ?          bmw       gas        std
17          17          0                 ?          bmw       gas        std
18          18          2               121    chevrolet       gas        std
19          19          1                98    chevrolet       gas        std
20          20          0                81    chevrolet       gas        std
21          21          1               118        dodge       gas        std
22          22          1               118        dodge       gas        std
23          23          1               118        dodge       gas      turbo
24          24          1               148        dodge       gas        std
```

4. Display last 25 records in the dataset.

```
print("last 25 records are")
print(last_25)
```

```
last 25 records are
     Unnamed: 0  symboling normalized-losses        make fuel-type aspiration  \
180         180         -1                90      toyota       gas        std
181         181         -1                 ?      toyota       gas        std
182         182          2               122  volkswagen    diesel        std
183         183          2               122  volkswagen       gas        std
184         184          2                94  volkswagen    diesel        std
185         185          2                94  volkswagen       gas        std
186         186          2                94  volkswagen       gas        std
187         187          2                94  volkswagen    diesel      turbo
188         188          2                94  volkswagen       gas        std
189         189          3                 ?  volkswagen       gas        std
190         190          3               256  volkswagen       gas        std
191         191          0                 ?  volkswagen       gas        std
192         192          0                 ?  volkswagen    diesel      turbo
193         193          0                 ?  volkswagen       gas        std
194         194         -2               103       volvo       gas        std
195         195         -1                74       volvo       gas        std
196         196         -2               103       volvo       gas        std
197         197         -1                74       volvo       gas        std
198         198         -2               103       volvo       gas      turbo
199         199         -1                74       volvo       gas      turbo
200         200         -1                95       volvo       gas        std
201         201         -1                95       volvo       gas      turbo
202         202         -1                95       volvo       gas        std
203         203         -1                95       volvo    diesel      turbo
204         204         -1                95       volvo       gas      turbo
```

5. Display number of missing values in each attribute in the dataset.

```
df = pd.read_csv('new_data.csv')
missing_values = df.isnull()
print("Missing Values are: ")
print(missing_values)
```

```
Missing Values are:
     Unnamed: 0  symboling  normalized-losses   make  fuel-type  aspiration  \
0         False      False              False  False      False       False
1         False      False              False  False      False       False
2         False      False              False  False      False       False
3         False      False              False  False      False       False
4         False      False              False  False      False       False
..          ...        ...                ...    ...        ...         ...
200       False      False              False  False      False       False
201       False      False              False  False      False       False
202       False      False              False  False      False       False
203       False      False              False  False      False       False
204       False      False              False  False      False       False

     num-of-doors  body-style  drive-wheels  engine-location  ...  \
0           False       False         False            False  ...
1           False       False         False            False  ...
2           False       False         False            False  ...
3           False       False         False            False  ...
4           False       False         False            False  ...
..            ...         ...           ...              ...  ...
200         False       False         False            False  ...
201         False       False         False            False  ...
202         False       False         False            False  ...
203         False       False         False            False  ...
204         False       False         False            False  ...
```

6. Find total invalid records in the dataset.

```
df = pd.read_csv('new_data.csv')
invalid_records = df[df.isin(['?']).any(axis=1)]
print("No. of Invalid Records:")
print(len(invalid_records))
```

```
No. of Invalid Records:
46
```

7. Display all numeric and non-numeric attribute in the dataset.

```
numeric = df.select_dtypes(include=["number"]).columns
non_numeric = df.select_dtypes(exclude=["number"]).columns
print("numeric: ",numeric)
print("non Numeric: ",non_numeric)
```

```
numeric:  Index(['Unnamed: 0', 'symboling', 'wheel-base', 'length', 'width', 'height',
       'curb-weight', 'engine-size', 'compression-ratio', 'city-mpg',
       'highway-mpg'],
      dtype='object')
non Numeric:  Index(['normalized-losses', 'make', 'fuel-type', 'aspiration', 'num-of-doors',
       'body-style', 'drive-wheels', 'engine-location', 'engine-type',
       'num-of-cylinders', 'fuel-system', 'bore', 'stroke', 'horsepower',
       'peak-rpm', 'price'],
      dtype='object')
```

8. Create a new data frame of numeric attributes.

```
numeric_df = df.select_dtypes(include=['number'])
numeric_df
```

Unnamed: 0	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	hig
0	3	88.6	168.8	64.1	48.8	2548	130	9.0	21	
1	3	88.6	168.8	64.1	48.8	2548	130	9.0	21	
2	1	94.5	171.2	65.5	52.4	2823	152	9.0	19	
3	2	99.8	176.6	66.2	54.3	2337	109	10.0	24	
4	2	99.4	176.6	66.4	54.3	2824	136	8.0	18	
...	...	...	...	...	...	...	...	...	...	
200	-1	109.1	188.8	68.9	55.5	2952	141	9.5	23	
201	-1	109.1	188.8	68.8	55.5	3049	141	8.7	19	
202	-1	109.1	188.8	68.9	55.5	3012	173	8.8	18	
203	-1	109.1	188.8	68.9	55.5	3217	145	23.0	26	
204	-1	109.1	188.8	68.9	55.5	3062	141	9.5	19	

9. Delete the all invalid records from the new data frame.

```
invalid_records_index = df[df.isin(['?']).any(axis=1)].index
df_cleaned = df.drop(invalid_records_index)
print(df_cleaned)
```

```
Unnamed: 0  symboling normalized-losses   make fuel-type aspiration  \
         3          2               164   audi       gas        std
         4          2               164   audi       gas        std
         6          1               158   audi       gas        std
         8          1               158   audi       gas      turbo
        10          2               192    bmw       gas        std
       ...        ...               ...    ...       ...        ...
       200         -1                95  volvo       gas        std
       201         -1                95  volvo       gas      turbo
       202         -1                95  volvo       gas        std
       203         -1                95  volvo    diesel      turbo
       204         -1                95  volvo       gas      turbo

um-of-doors body-style drive-wheels engine-location  ...  engine-size  \
       four      sedan          fwd           front  ...          109
       four      sedan          4wd           front  ...          136
       four      sedan          fwd           front  ...          136
       four      sedan          fwd           front  ...          131
        two      sedan          rwd           front  ...          108
        ...        ...          ...             ...  ...          ...
       four      sedan          rwd           front  ...          141
       four      sedan          rwd           front  ...          141
       four      sedan          rwd           front  ...          173
       four      sedan          rwd           front  ...          145
       four      sedan          rwd           front  ...          141
```

10. Compute the average height width and length of all automobiles.

```
path = 'new_data.csv'
df = pd.read_csv(path)
# Choose the columns for which you want to compute the average
selected_columns = ['height', 'width', 'length']
# Check if all selected columns exist in the DataFrame
missing_columns = [col for col in selected_columns if col not in df.columns]
if not missing_columns:
    # Compute the average of the selected columns
    columns_average = df[selected_columns].mean()

    print("Average of selected columns:")
    print(columns_average)
else:
    print(f"Columns {missing_columns} not found in the DataFrame.")
```

```
Average of selected columns:
height     53.724878
width      65.907805
length    174.049268
dtype: float64
```

11. Compute the standard deviation height width and length of all automobiles.

```
path = 'new_data.csv'
df = pd.read_csv(path)
# Choose the columns for which you want to compute the standard deviation
selected_columns = ['height', 'width', 'length']
# Check if all selected columns exist in the DataFrame
missing_columns = [col for col in selected_columns if col not in df.columns]
if not missing_columns:
    # Compute the standard deviation of the selected columns
    columns_std_dev = df[selected_columns].std()
    print("Standard deviation of selected columns:")
    print(columns_std_dev)
else:
    print(f"Columns {missing_columns} not found in the DataFrame.")
```

```
Standard deviation of selected columns:
height     2.443522
width      2.145204
length    12.337289
dtype: float64
```

12. Describe each attribute in dataset by min, max, average, std and count.

```
path = 'new_data.csv'
df = pd.read_csv(path)
# Use the describe function
description = df.describe()
# Transpose the result for better readability
description = description.T
print("Attribute Descriptions:")
print(description)
```

```
Attribute Descriptions:
                   count         mean          std     min     25%     50%  \
Unnamed: 0         205.0   102.000000    59.322565     0.0    51.0   102.0
symboling          205.0     0.834146     1.245307    -2.0     0.0     1.0
wheel-base         205.0    98.756585     6.021776    86.6    94.5    97.0
length             205.0   174.049268    12.337289   141.1   166.3   173.2
width              205.0    65.907805     2.145204    60.3    64.1    65.5
height             205.0    53.724878     2.443522    47.8    52.0    54.1
curb-weight        205.0  2555.565854   520.680204  1488.0  2145.0  2414.0
engine-size        205.0   126.907317    41.642693    61.0    97.0   120.0
compression-ratio  205.0    10.142537     3.972040     7.0     8.6     9.0
city-mpg           205.0    25.219512     6.542142    13.0    19.0    24.0
highway-mpg        205.0    30.751220     6.886443    16.0    25.0    30.0

                     75%     max
Unnamed: 0         153.0   204.0
symboling            2.0     3.0
wheel-base         102.4   120.9
length             183.1   208.1
width               66.9    72.3
height              55.5    59.8
curb-weight       2935.0  4066.0
engine-size        141.0   326.0
compression-ratio    9.4    23.0
city-mpg            30.0    49.0
highway-mpg         34.0    54.0
```

13. Sort all the vehicles by their “bore”.

```
path = 'new_data.csv'
df = pd.read_csv(path)
# Choose the column for sorting
sort_column = 'bore'
# Check if the specified column exists in the DataFrame
if sort_column in df.columns:
    # Sort the DataFrame by the selected column
    sorted_df = df.sort_values(by=sort_column)
    # Display the sorted DataFrame
    print(sorted_df)
else:
    print(f"Column '{sort_column}' not found in the DataFrame.")
```

```
     Unnamed: 0  symboling normalized-losses         make fuel-type  \
134         134          3               150         saab       gas
2             2          1                 ?  alfa-romero       gas
31           31          2               137        honda       gas
30           30          2               137        honda       gas
34           34          1               101        honda       gas
..          ...        ...               ...          ...       ...
124         125          3               186      porsche       gas
54           55          3               150        mazda       gas
55           56          3               150        mazda       gas
56           57          3               150        mazda       gas
57           58          3               150        mazda       gas

    aspiration num-of-doors body-style drive-wheels engine-location  ...  \
134        std          two  hatchback          fwd           front  ...
2          std          two  hatchback          rwd           front  ...
31         std          two  hatchback          fwd           front  ...
30         std          two  hatchback          fwd           front  ...
34         std          two  hatchback          fwd           front  ...
..         ...          ...        ...          ...             ...  ...
124        std          two  hatchback          rwd           front  ...
54         std          two  hatchback          rwd           front  ...
55         std          two  hatchback          rwd           front  ...
56         std          two  hatchback          rwd           front  ...
57         std          two  hatchback          rwd           front  ...

     engine-size  fuel-system  bore  stroke  compression-ratio horsepower  \
134          121         mpfi  2.54    2.07               9.3        110
2            152         mpfi  2.68    3.47               9.0        154
31            92         1bbl  2.91    3.41               9.2         76
30            92         1bbl  2.91    3.41               9.6         58
34            92         1bbl  2.91    3.41               9.2         76
..           ...          ...   ...     ...               ...        ...
124          151         mpfi  3.94    3.11               9.5        143
54            70         4bbl     ?       ?               9.4        101
55            70         4bbl     ?       ?               9.4        101
56            70         4bbl     ?       ?               9.4        101
57            80         mpfi     ?       ?               9.4        135
```

14. Create sample dataset as “subset1” by considering the attributes: make, fuel-type,num-of-doors, drive-wheels and engine-location.

```
path = 'new_data.csv'
original_df = pd.read_csv(path)
# Specify the columns you want to include in the sample dataset
selected_columns = ['make', 'fuel-type', 'num-of-doors', 'drive-wheels', 'engine-location']
# Create the sample dataset 'sunset-1' by selecting specific columns
sunset_1 = original_df[selected_columns].copy()
output_csv = 'subset-1.csv'
sunset_1.to_csv(output_csv, index = False)
```

```
#print('new dataset created considering the above columns')
df=pd.read_csv('subset-1.csv')
df
```

	make	fuel-type	num-of-doors	drive-wheels	engine-location
0	alfa-romero	gas	two	rwd	front
1	alfa-romero	gas	two	rwd	front
2	alfa-romero	gas	two	rwd	front
3	audi	gas	four	fwd	front
4	audi	gas	four	4wd	front
...	...	...	...	...	...
200	volvo	gas	four	rwd	front
201	volvo	gas	four	rwd	front
202	volvo	gas	four	rwd	front
203	volvo	diesel	four	rwd	front
204	volvo	gas	four	rwd	front

205 rows × 5 columns

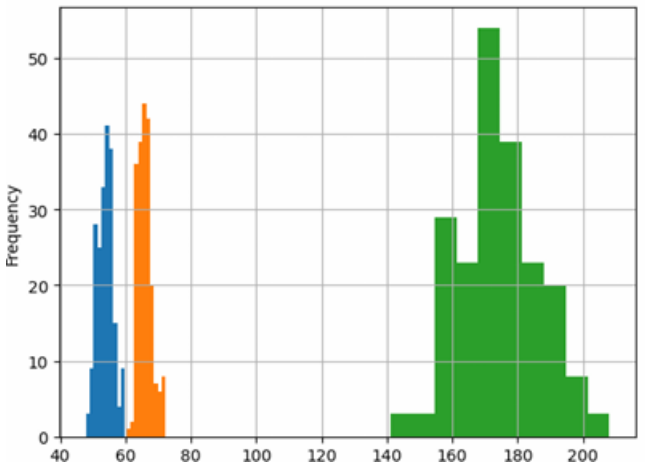
15. Create sample dataset as “subset2” by considering the attributes: make, height,length, width.

```
# Create sample dataset as “sunset-2” by considering the attributes: wheel-base,length, width,
path = 'new_data.csv'
original_df = pd.read_csv(path)
# Specify the columns you want to include in the sample dataset
selected_columns = ['wheel-base', 'length', 'width', 'height', 'curb-weight']
# Create the sample dataset 'sunset-2' by selecting specific columns
sunset_2 = original_df[selected_columns].copy()
output_csv_2 = 'subset-2.csv'
sunset_2.to_csv(output_csv_2)
#print('new dataset created considering the above columns')
df=pd.read_csv('subset-2.csv')
df
```

	Unnamed: 0	wheel-base	length	width	height	curb-weight
0	0	88.6	168.8	64.1	48.8	2548
1	1	88.6	168.8	64.1	48.8	2548
2	2	94.5	171.2	65.5	52.4	2823
3	3	99.8	176.6	66.2	54.3	2337
4	4	99.4	176.6	66.4	54.3	2824
..	...	...	...	...	...	...
0	200	109.1	188.8	68.9	55.5	2952
1	201	109.1	188.8	68.8	55.5	3049
2	202	109.1	188.8	68.9	55.5	3012
3	203	109.1	188.8	68.9	55.5	3217
4	204	109.1	188.8	68.9	55.5	3062

17. Plot histograms of height, width and length.

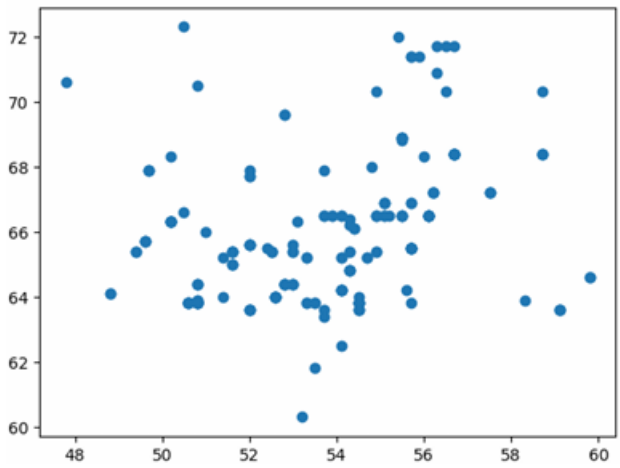
```
import matplotlib.pyplot as plt
df = pd.read_csv('subset-2.csv')
df['height'].hist(label='height')
df['width'].hist(label='width')
df['length'].hist(label='length')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```



18. Plot scatter plot between height and width attributes.

```
plt.scatter(df['height'],df['width'])
```

```
<matplotlib.collections.PathCollection at 0x1ea6ede2390>
```



Lab Experiment 2

Objective:

• Data preprocessing

• Basic data visualization using Matplotlib package.

Consider the given automobile data and do the following:

1. Load the dataset and create the new data frame as MyDF by considering the following attributes: number in () denotes the attribute index number in the data frame.

• Symbolling : (1) -3, -2, -1, 0, 1, 2, 3.

• fuel-type : (4) diesel, gas.

• body-style: (7) hardtop, wagon, sedan, hatchback, convertible.

• drive-wheels: (8) 4wd, fwd, rwd.

• wheel-base: (10) continuous from 86.6 120.9.

• length: (11) continuous from 141.1 to 208.1.

• width: (12) continuous from 60.3 to 72.3.

• height: (13) continuous from 47.8 to 59.8.

• num-of-cylinders: (16) eight, five, four, six, three, twelve, two.

• bore: (19) continuous from 2.54 to 3.94.

• horsepower: (22) continuous from 48 to 288.

• city-mpg: (24) continuous from 13 to 49.

• highway-mpg: (25) continuous from 16 to 54.

• price: (26) continuous from 5118 to 45400.

```
path = 'new_data.csv'
original_df = pd.read_csv(path)
selected_columns = ['symboling', 'fuel-type', 'body-style', 'drive-wheels', 'wheel-base','leng
Data = original_df[selected_columns].copy()
output_csv_2 = 'MyDF.csv'
Data.to_csv(output_csv_2)
```

```
df=pd.read_csv('MyDF.csv')
df
```

	symboling	fuel-type	body-style	drive-wheels	wheel-base	length	width	height	num-of-cylinders	bore	horsepow
0	3	gas	convertible	rwd	88.6	168.8	64.1	48.8	four	3.47	1
1	3	gas	convertible	rwd	88.6	168.8	64.1	48.8	four	3.47	1
2	1	gas	hatchback	rwd	94.5	171.2	65.5	52.4	six	2.68	1
3	2	gas	sedan	fwd	99.8	176.6	66.2	54.3	four	3.19	1
4	2	gas	sedan	4wd	99.4	176.6	66.4	54.3	five	3.19	1
...	...	...	...	...	...	...	...	...	...	...	
200	-1	gas	sedan	rwd	109.1	188.8	68.9	55.5	four	3.78	1
201	-1	gas	sedan	rwd	109.1	188.8	68.8	55.5	four	3.78	1
202	-1	gas	sedan	rwd	109.1	188.8	68.9	55.5	six	3.58	1
203	-1	diesel	sedan	rwd	109.1	188.8	68.9	55.5	six	3.01	1
204	-1	gas	sedan	rwd	109.1	188.8	68.9	55.5	four	3.78	1

205 rows × 14 columns

2. Fill the missing values in the dataset.

```
df = pd.read_csv('MyDF.csv')
invalid_records = df[df.isin(['?']).any(axis=1)]
print("No. of Invalid Records:")
print(len(invalid_records))
```

```
No. of Invalid Records:
10
```

3. Normalize the height attribute with min-max normalization.

```
df['normalize-height']=(df['height']-df['height'].min())/(df['height'].max()-df['height'].min
df
```

	symboling	fuel-type	body-style	drive-wheels	wheel-base	length	width	height	num-of-cylinders	bore	horsepow
0	3	gas	convertible	rwd	88.6	168.8	64.1	48.8	four	3.47	1
1	3	gas	convertible	rwd	88.6	168.8	64.1	48.8	four	3.47	1
2	1	gas	hatchback	rwd	94.5	171.2	65.5	52.4	six	2.68	1
3	2	gas	sedan	fwd	99.8	176.6	66.2	54.3	four	3.19	1
4	2	gas	sedan	4wd	99.4	176.6	66.4	54.3	five	3.19	1
...	...	...	...	...	...	...	...	...	...	...	
200	-1	gas	sedan	rwd	109.1	188.8	68.9	55.5	four	3.78	1
201	-1	gas	sedan	rwd	109.1	188.8	68.8	55.5	four	3.78	1
202	-1	gas	sedan	rwd	109.1	188.8	68.9	55.5	six	3.58	1
203	-1	diesel	sedan	rwd	109.1	188.8	68.9	55.5	six	3.01	1
204	-1	gas	sedan	rwd	109.1	188.8	68.9	55.5	four	3.78	1

205 rows × 15 columns

4. Normalize the horsepower attribute with z-score normalization.

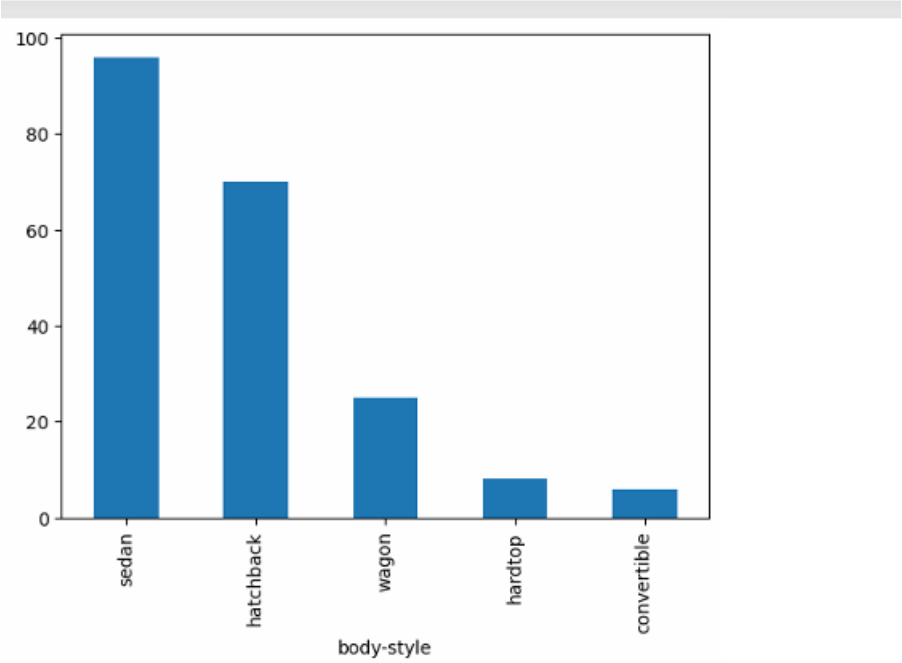
```
df = pd.read_csv("MyDF.csv")
df['horsepower'] = df['horsepower'].replace('?',np.nan)
df['horsepower'] = pd.to_numeric(df['horsepower'], errors = 'coerce')
mean_value = df['horsepower'].mean()
df['horsepower'].fillna(mean_value, inplace=True)
new_MyDF = df.copy()
new_MyDF.to_csv("New_MyDF")
new_MyDF
```

	symboling	fuel-type	body-style	drive-wheels	wheel-base	length	width	height	num-of-cylinders	bore	horsepow
0	3	gas	convertible	rwd	88.6	168.8	64.1	48.8	four	3.47	111
1	3	gas	convertible	rwd	88.6	168.8	64.1	48.8	four	3.47	111
2	1	gas	hatchback	rwd	94.5	171.2	65.5	52.4	six	2.68	154
3	2	gas	sedan	fwd	99.8	176.6	66.2	54.3	four	3.19	102
4	2	gas	sedan	4wd	99.4	176.6	66.4	54.3	five	3.19	115
...	...	...	...	...	...	...	...	...	...	...	
200	-1	gas	sedan	rwd	109.1	188.8	68.9	55.5	four	3.78	114
201	-1	gas	sedan	rwd	109.1	188.8	68.8	55.5	four	3.78	160
202	-1	gas	sedan	rwd	109.1	188.8	68.9	55.5	six	3.58	134
203	-1	diesel	sedan	rwd	109.1	188.8	68.9	55.5	six	3.01	106
204	-1	gas	sedan	rwd	109.1	188.8	68.9	55.5	four	3.78	114

205 rows × 14 columns

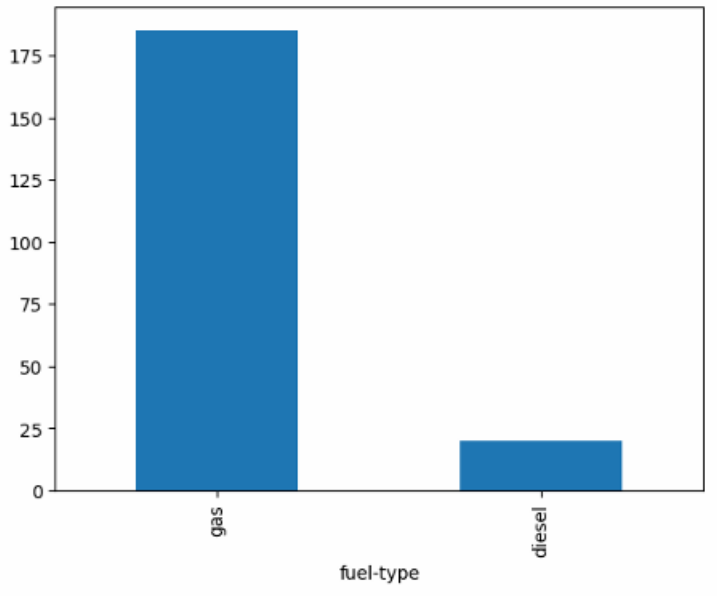
5. Plot the car distribution according to body style (Bar Graph)

```
new_MyDF['body-style'].value_counts().plot(kind='bar')
plt.show()
```



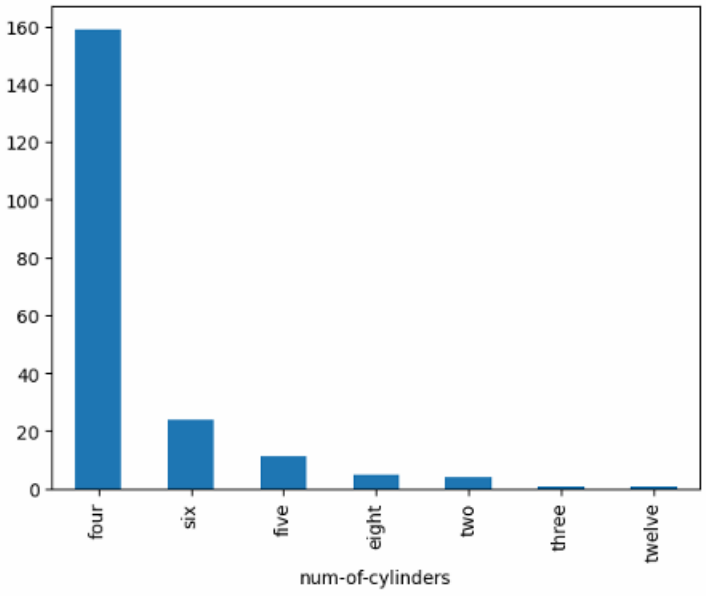
6. Plot a bar graph to show the distribution of cars by fuel type.

```
new_MyDF['fuel-type'].value_counts().plot(kind='bar')
plt.show()
```



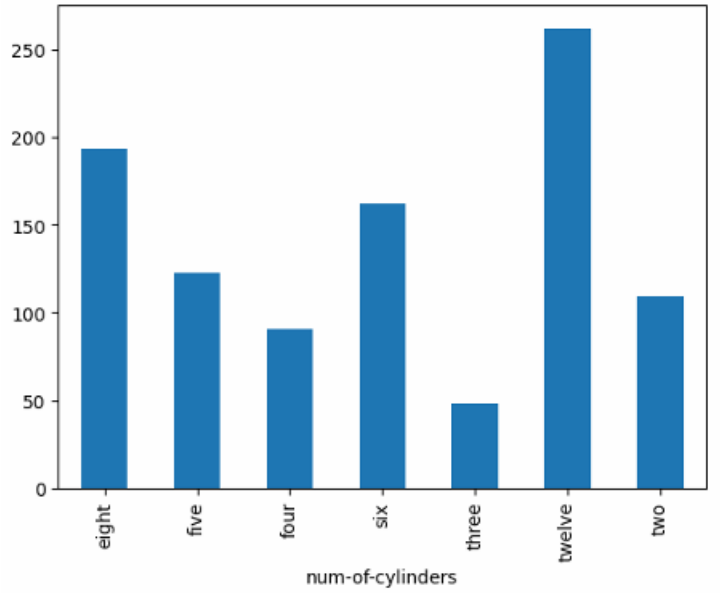
7. Plot a bar graph to show the distribution of cars by num-of-cylinders.

```
new_MyDF['num-of-cylinders'].value_counts().plot(kind='bar')
plt.show()
```



8. Plot the average horsepower of cars according to their number of cylinders.

```
df['horsepower'] = pd.to_numeric(df['horsepower'], errors='coerce')
avg_hrs = df.groupby('num-of-cylinders')['horsepower'].mean()
avg_hrs.plot(kind='bar')
plt.show()
```



9. Plot the average highway mpg, city mpg and price of cars according to their fuel type.

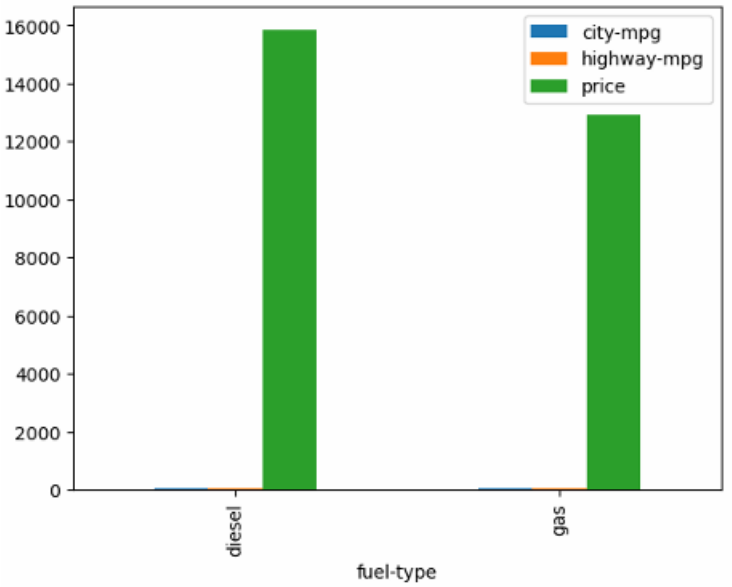
```
df = pd.read_csv("new_MyDF.csv")
df['price'] = df['price'].replace('?',np.nan)
df['price'] = pd.to_numeric(df['price'], errors = 'coerce')
mean_value = df['price'].mean()
df['price'].fillna(mean_value, inplace=True)
new_MyDF = df.copy()
new_MyDF.to_csv("cleaned_MyDF.csv")
```

```
df = pd.read_csv("cleaned_MyDF.csv")
df
```

	symboling	fuel-type	body-style	drive-wheels	wheel-base	length	width	height	num-of-cylinders	bore	horsepow
0	3	gas	convertible	rwd	88.6	168.8	64.1	48.8	four	3.47	111
1	3	gas	convertible	rwd	88.6	168.8	64.1	48.8	four	3.47	111
2	1	gas	hatchback	rwd	94.5	171.2	65.5	52.4	six	2.68	154
3	2	gas	sedan	fwd	99.8	176.6	66.2	54.3	four	3.19	102
4	2	gas	sedan	4wd	99.4	176.6	66.4	54.3	five	3.19	115
...	...	...	...	...	...	...	...	...	...	...	
200	-1	gas	sedan	rwd	109.1	188.8	68.9	55.5	four	3.78	114
201	-1	gas	sedan	rwd	109.1	188.8	68.8	55.5	four	3.78	160
202	-1	gas	sedan	rwd	109.1	188.8	68.9	55.5	six	3.58	134
203	-1	diesel	sedan	rwd	109.1	188.8	68.9	55.5	six	3.01	106
204	-1	gas	sedan	rwd	109.1	188.8	68.9	55.5	four	3.78	114

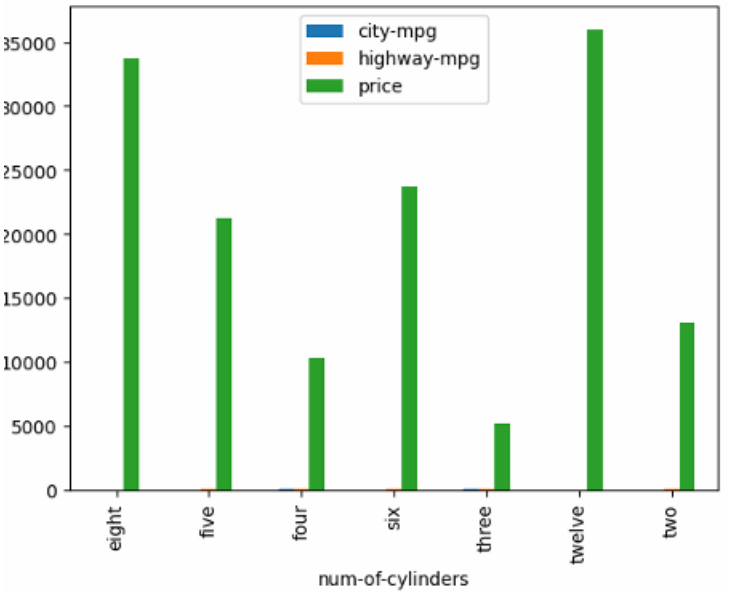
205 rows × 14 columns

```
df['city-mpg'] = pd.to_numeric(df['city-mpg'], errors='coerce')
df['highway-mpg'] = pd.to_numeric(df['highway-mpg'], errors='coerce')
df['price'] = pd.to_numeric(df['price'], errors='coerce')
avg_hrs = df.groupby('fuel-type')[['city-mpg','highway-mpg','price']].mean()
avg_hrs.plot(kind='bar')
plt.show()
```



10. Plot the average highway mpg, city mpg and price of cars according to their num-of-cylinders.

```
df['city-mpg'] = pd.to_numeric(df['city-mpg'], errors='coerce')
df['highway-mpg'] = pd.to_numeric(df['highway-mpg'], errors='coerce')
df['price'] = pd.to_numeric(df['price'], errors='coerce')
avg_hrs = df.groupby('num-of-cylinders')[['city-mpg','highway-mpg','price']].mean()
avg_hrs.plot(kind='bar')
plt.show()
```



Lab Experiment 3

Objective:

1.Basic data visualization using Matplotlib package.

2. To understand the concept of plot, subplot.

3. Bar graph

Query: What is the use of bar graphs? In what scenarios we plot the data using bar plot.

Exp-1

Consider the data set having 11 employees.

Name = [A,B,C,D,E,F,G,H,I,J,K]

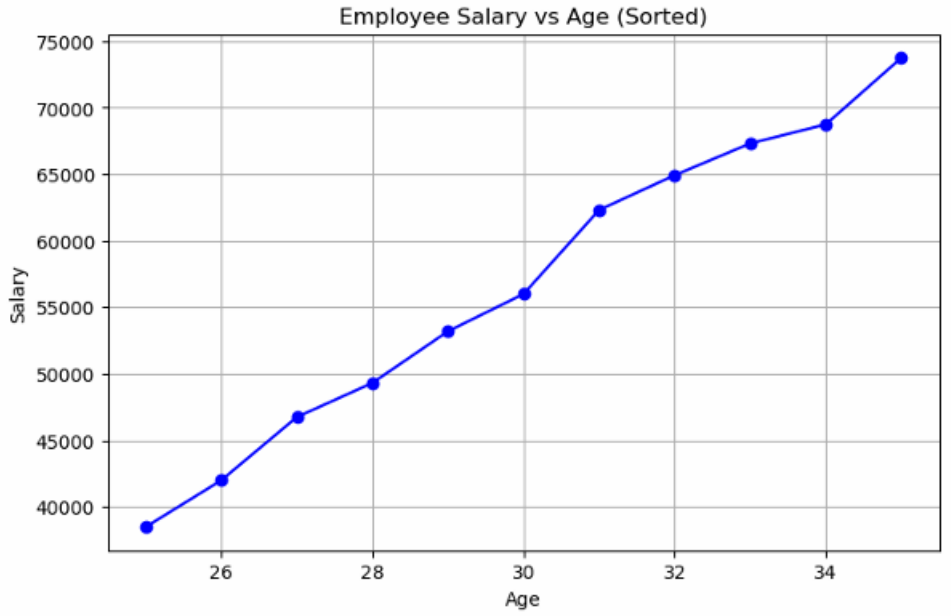
E_age = [25,26,27,28,29,30,31,32,33,34,35]

E_salary = [38496,42000,46752,49320,53200,56000,62316,64928,67317,68748,73752]

E_Overhead= [45372,48876,53850,57287,63016,65998,70003,71496,75370,83640,45872]

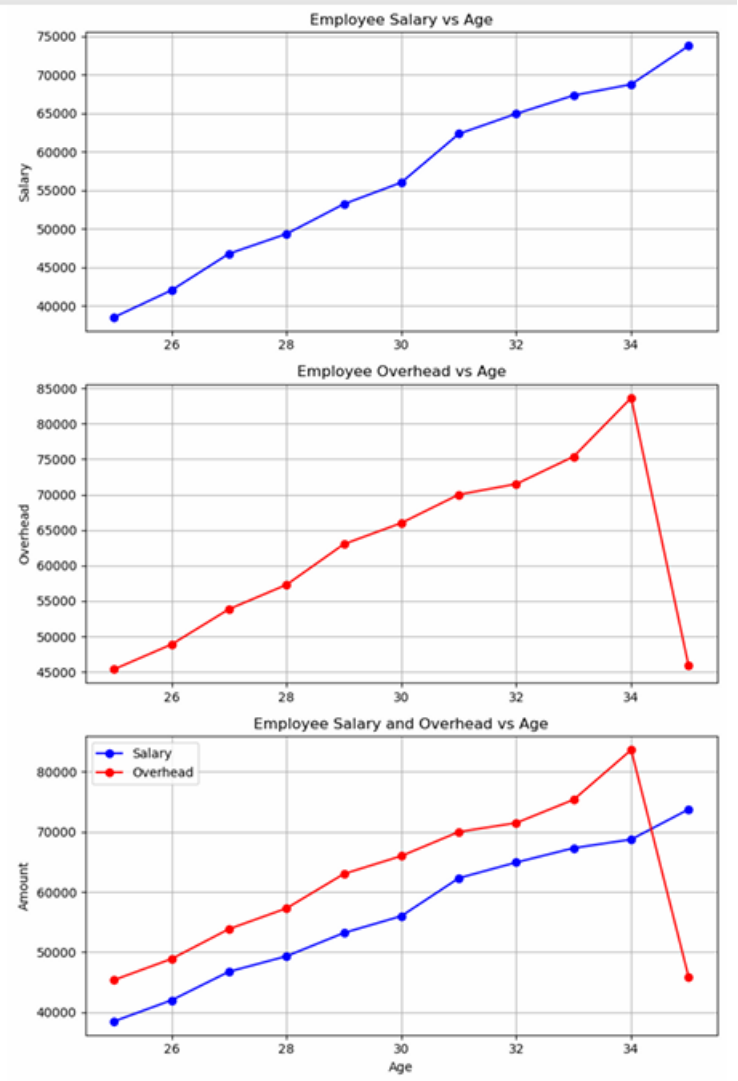
- Plot the employee salary according to their age in sorted order.
- Plot the employee age, salary and overhead in the form grid 3x1.
- Plot the employee salary and overhead in the form grid 2x1. The plot share the y-axis.
- Plot the employee salary and overhead in the form grid 1x2. The plot share the x-axis.
- Plot the employee salary and overhead in single plot.

```
# Sort data by age
sorted_indices = sorted(range(len(E_age)), key=lambda k: E_age[k])
E_age_sorted = [E_age[i] for i in sorted_indices]
E_salary_sorted = [E_salary[i] for i in sorted_indices]


plt.figure(figsize=(8, 5))
plt.plot(E_age_sorted, E_salary_sorted, marker='o', linestyle='-', color='b')
plt.title('Employee Salary vs Age (Sorted)')
plt.xlabel('Age')
plt.ylabel('Salary')
plt.grid(True)
plt.show()
```



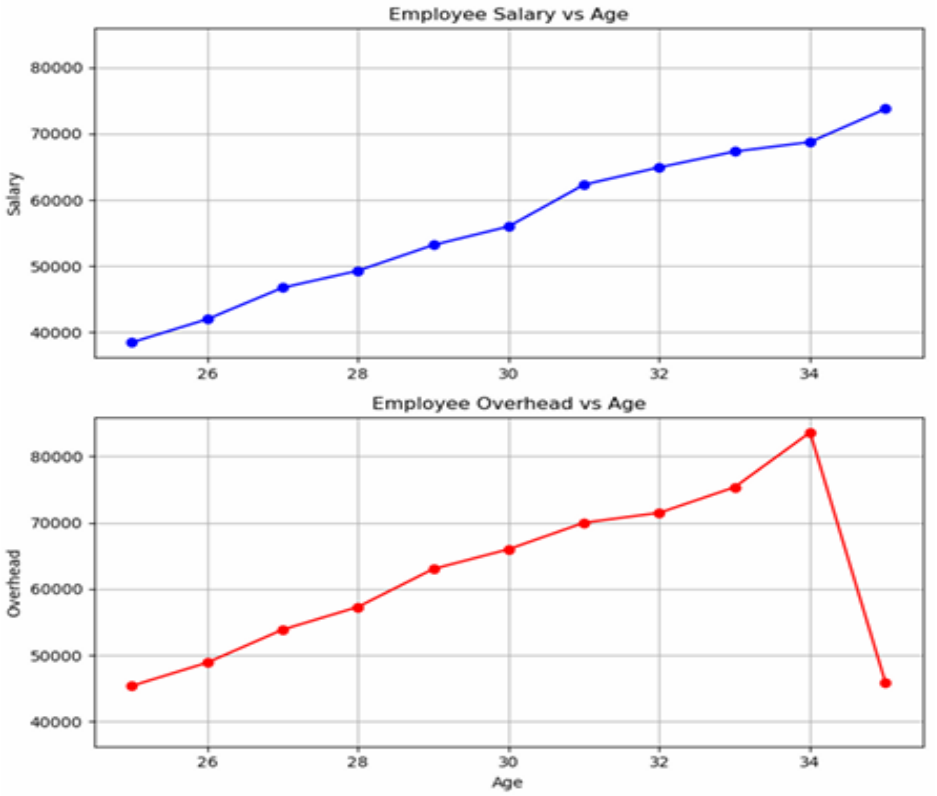
Query (ii) Plot the employee age, salary and overhead in the form grid 3x1.

```
fig, axes = plt.subplots(3, 1, figsize=(8, 12))

axes[0].plot(E_age, E_salary, marker='o', linestyle='-', color='b')
axes[0].set_title('Employee Salary vs Age')
axes[0].set_ylabel('Salary')
axes[0].grid(True)

axes[1].plot(E_age, E_Overhead, marker='o', linestyle='-', color='r')
axes[1].set_title('Employee Overhead vs Age')
axes[1].set_ylabel('Overhead')
axes[1].grid(True)

axes[2].plot(E_age, E_salary, marker='o', linestyle='-', color='b', label='Salary')
axes[2].plot(E_age, E_Overhead, marker='o', linestyle='-', color='r', label='Overhead')
axes[2].set_title('Employee Salary and Overhead vs Age')
axes[2].set_xlabel('Age')
axes[2].set_ylabel('Amount')
axes[2].legend()
axes[2].grid(True)

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



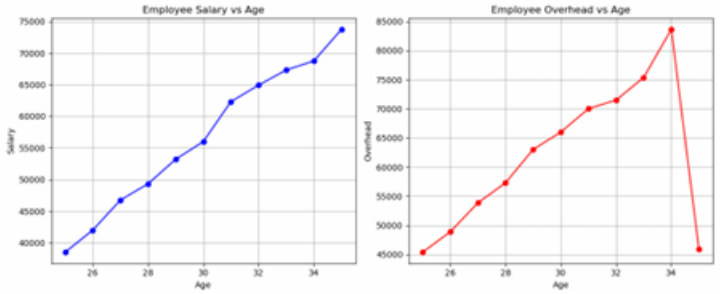
Query (iii) Plot the employee salary and overhead in the form grid 2x1. The plot share the y-axis.

```
fig, axes = plt.subplots(2, 1, figsize=(8, 8), sharey=True)

axes[0].plot(E_age, E_salary, marker='o', linestyle='-', color='b')
axes[0].set_title('Employee Salary vs Age')
axes[0].set_ylabel('Salary')
axes[0].grid(True)

axes[1].plot(E_age, E_Overhead, marker='o', linestyle='-', color='r')
axes[1].set_title('Employee Overhead vs Age')
axes[1].set_xlabel('Age')
axes[1].set_ylabel('Overhead')
axes[1].grid(True)

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



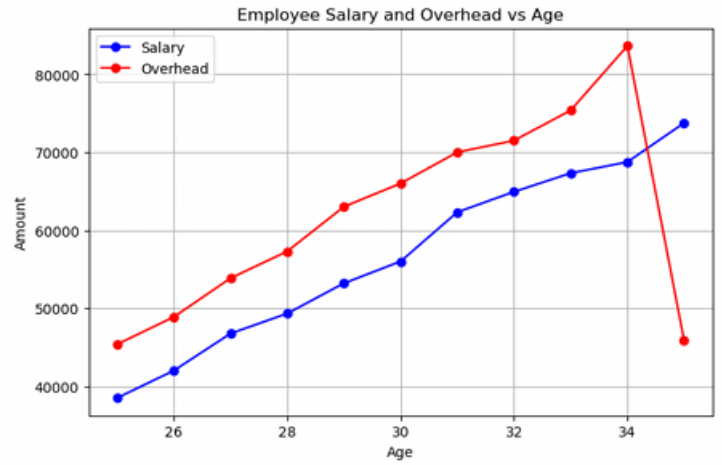
Query (iv) Plot the employee salary and overhead in the form grid 1x2. The plot share the x-axis.

```
fig, axes = plt.subplots(1, 2, figsize=(12, 5), sharex=True)

axes[0].plot(E_age, E_salary, marker='o', linestyle='-', color='b')
axes[0].set_title('Employee Salary vs Age')
axes[0].set_xlabel('Age')
axes[0].set_ylabel('Salary')
axes[0].grid(True)

axes[1].plot(E_age, E_Overhead, marker='o', linestyle='-', color='r')
axes[1].set_title('Employee Overhead vs Age')
axes[1].set_xlabel('Age')
axes[1].set_ylabel('Overhead')
axes[1].grid(True)

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



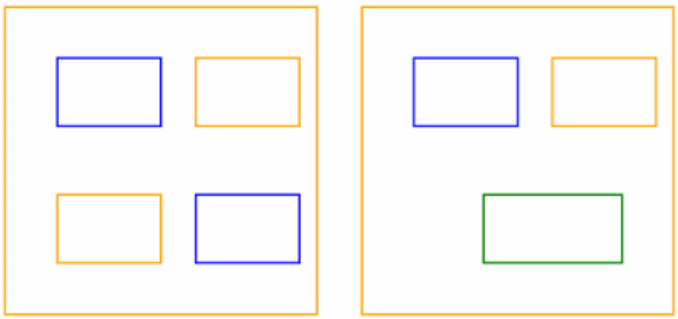
Query (v) Plot the employee salary and overhead in single plot.

```
plt.figure(figsize=(8, 5))
plt.plot(E_age, E_salary, marker='o', linestyle='-', color='b', label='Salary')
plt.plot(E_age, E_Overhead, marker='o', linestyle='-', color='r', label='Overhead')
plt.title('Employee Salary and Overhead vs Age')
plt.xlabel('Age')
plt.ylabel('Amount')
plt.legend()
plt.grid(True)
plt.show()
```



Exp :- 2 Plot the dummy graph in the following grid format.

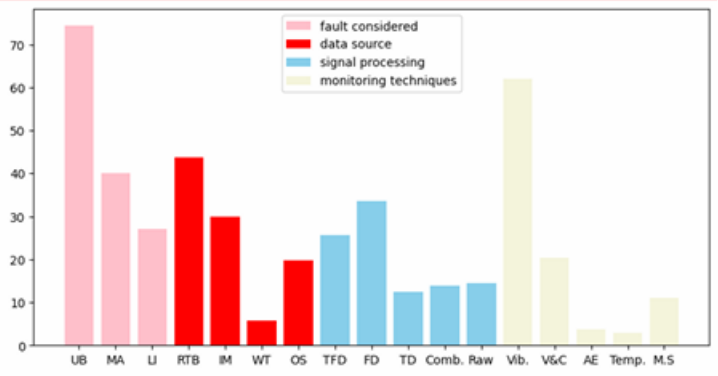
```
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Create a figure
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# Function to draw a box inside an axis
def draw_box(ax, x, y, width, height, color="blue"):
    rect = patches.Rectangle((x, y), width, height, linewidth=2, edgecolor=color, facecolor='
    ax.add_patch(rect)

# **Left Side Grid (First Box)**
axes[0].set_xlim(0, 10)
axes[0].set_ylim(0, 10)
axes[0].set_xticks([])
axes[0].set_yticks([])
axes[0].spines[:].set_visible(False)

# Draw 4 boxes in a grid layout
draw_box(axes[0], 2, 6, 3, 2, "blue")
draw_box(axes[0], 6, 6, 3, 2, "orange")
draw_box(axes[0], 2, 2, 3, 2, "orange")
draw_box(axes[0], 6, 2, 3, 2, "blue")

# **Right Side Grid (Second Box)**
axes[1].set_xlim(0, 10)
axes[1].set_ylim(0, 10)
axes[1].set_xticks([])
axes[1].set_yticks([])
axes[1].spines[:].set_visible(False)

# Draw boxes in different positions
draw_box(axes[1], 2, 6, 3, 2, "blue")
draw_box(axes[1], 6, 6, 3, 2, "orange")
draw_box(axes[1], 4, 2, 4, 2, "green")

# **Outer Box Boundaries** (To mimic the image layout)
for ax in axes:
    ax.add_patch(patches.Rectangle((0.5, 0.5), 9, 9, linewidth=2, edgecolor="orange", facecol

# Show the plot
plt.tight_layout()
plt.show()
```



Exp-3: Re generate the figure given below.

```
fault_considered=[74.45,40.15,27.01]
data_source=[43.8,29.93,5.84,19.71]
signal_processing=[25.55,33.58,12.41,13.87,14.50]
monitoring_techniques=[62.04,20.44,3.65,2.92,10.95]
label_01=['UB','MA','LI']
label_02=['RTB','IM','WT','OS']
label_03=['TFD','FD','TD','Comb.','Raw']
label_04=['Vib.','V&C','AE','Temp.','M.S']

plt.figure(figsize=(10,5))
plt.bar(label_01,fault_considered,color='pink',label='fault considered')
plt.bar(label_02,data_source,color='r',label='data source')
plt.bar(label_03,signal_processing,color='skyblue',label='signal processing')
plt.bar(label_04,monitoring_techniques,color='beige',label='monitoring techniques')
plt.legend(loc='upper center')

# Function to add labels on top of bars
def add_labels(bars):
    for bar in bars:
        height = bar.get_height()
        plt.text(bar.get_x() + bar.get_width() / 2, height, '%.2f' % height, ha='center', va=

# Add labels on top of each bar
add_labels(bars1)
add_labels(bars2)
add_labels(bars3)
add_labels(bars4)

plt.legend(loc='upper center')
```

```
plt.xlabel('Methods')
plt.ylabel('Values')
plt.title('Comparison of Different Methods')
plt.show()
```



Lab Experiment 4

Objective:

• Basic data visualization using Matplotlib package.

Exp-1: Analyse the given dataset using line graphs.

Consider the data set having 11 employees.

E_age = [25,26,27,28,29,30,31,32,33,34,35]

E_salary = [38496,42000,46752,49320,53200,56000,62316,64928,67317,68748,73752]

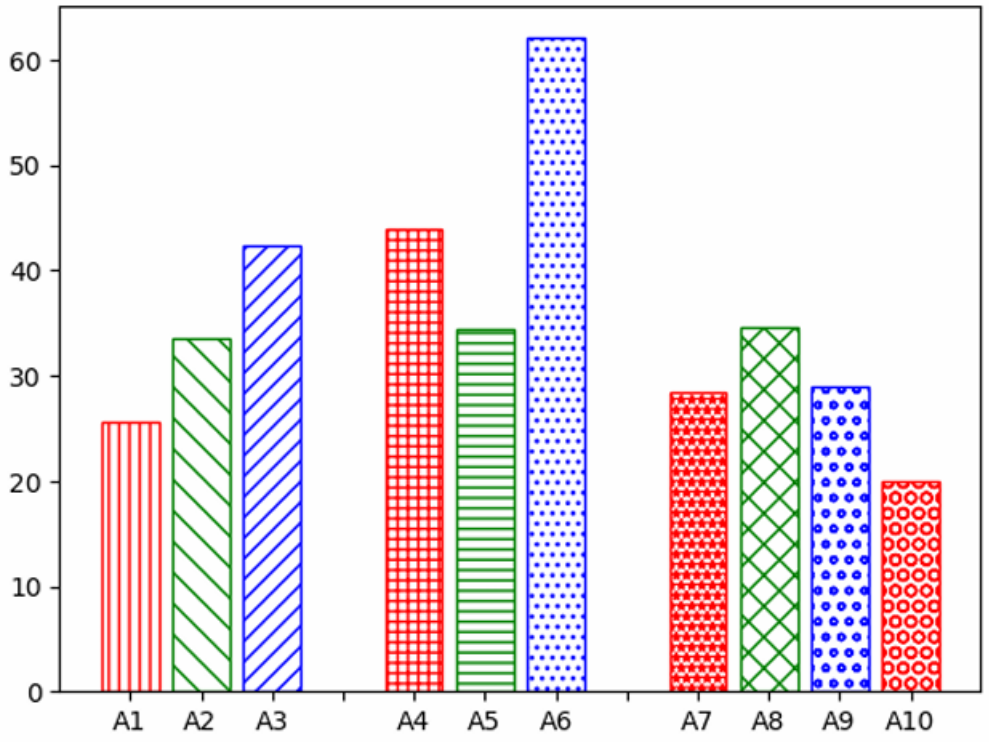
E_Overhead= [45372,48876,53850,57287,63016,65998,70003,71496,75370,83640,45872]

```
E_age = [25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35]
E_salary = [38496, 42000, 46752, 49320, 53200, 56000, 62316, 64928, 67317, 68748, 73752]
E_Overhead = [45372, 48876, 53850, 57287, 63016, 65998, 70003, 71496, 75370, 83640, 45872]

plt.plot(E_age, E_salary, label='Salary')
plt.plot(E_age, E_Overhead, label='Overhead')

plt.xlabel('Employee Age')
plt.ylabel('Values')
plt.title('Employee Data')
plt.legend()

plt.show()
```

Exp-2: Regenerate this graph using the matplotlib library.

```
patterns = ["|||", "\\\\", "///","oo", "+++", "---", "...","oo", "***", "\/\/", "oo", "OO"]
bars_ticks = ['A1', 'A2', 'A3',' ','A4', 'A5', 'A6',' ','A7', 'A8', 'A9', 'A10']
bar_height = [25.55, 33.58, 42.41,0,43.87, 34.5, 62.04,0, 28.44, 34.65, 28.92, 19.95]
hatch_colors = ['red', 'green', 'blue','white', 'red', 'green', 'blue','white', 'red', 'green
y_pos = np.arange(len(bars_ticks))
width = 0.8  # width of bars
fig, ax = plt.subplots()
```

```
bars = ax.bar(y_pos, bar_height, width=width, color='none', edgecolor='black')

for i, bar in enumerate(bars):
    bar.set_hatch(patterns[i])
    bar.set_edgecolor(hatch_colors[i])  # Set the hatch color to the edge color
ax.set_xticks(y_pos)
ax.set_xticklabels(bars_ticks)
plt.show()
```



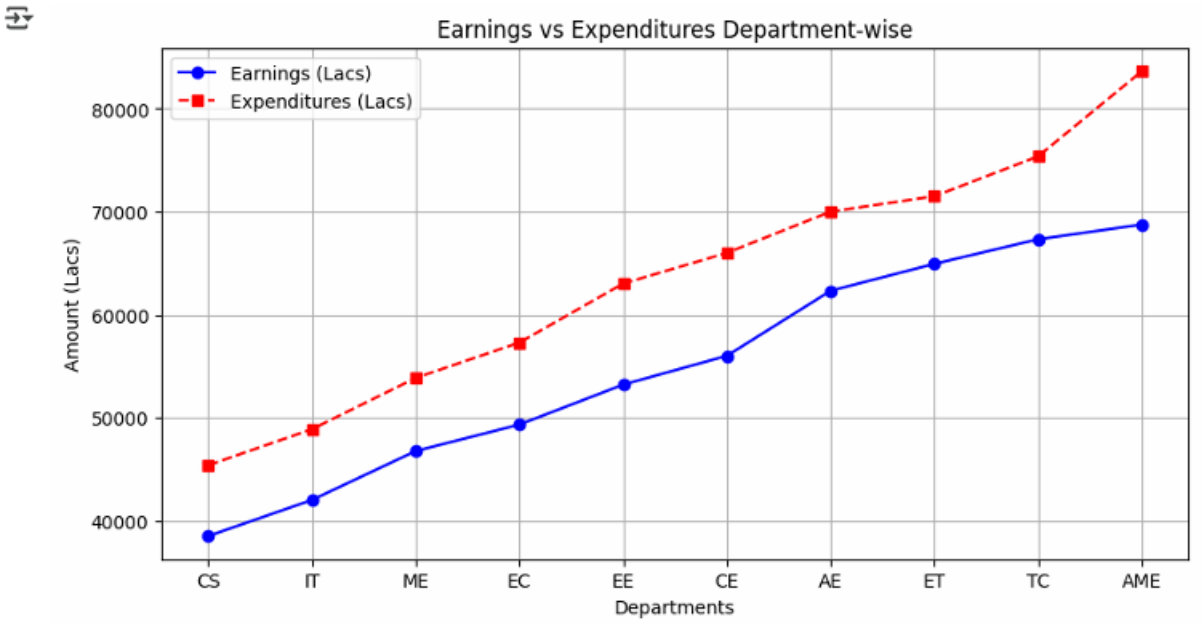
Lab Experiment 5

Exp-1 Consider the data set having 11 employees.

Dept. = [CS, IT, ME, EC, EE, CE, AE, ET, TC, AME] Earning (Lacs)= [38496,42000,46752,49320,53200,56000,62316,64928,67317,68748] Expenditure (Lacs)= [45372,48876,53850,57287,63016,65998,70003,71496,75370,83640]

1. Visualize the earning and expenditure department wise. (Line graph)

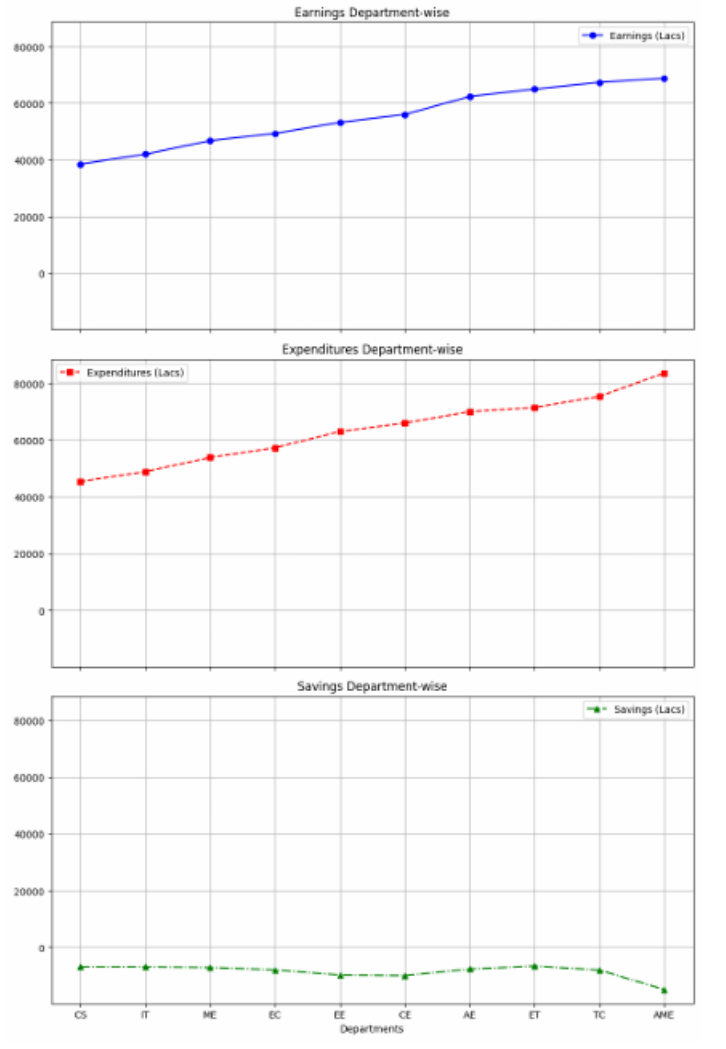
```
import matplotlib.pyplot as plt

departments = ['CS', 'IT', 'ME', 'EC', 'EE', 'CE', 'AE', 'ET', 'TC', 'AME']
earnings = [38496, 42000, 46752, 49320, 53200, 56000, 62316, 64928, 67317, 68748]
expenditures = [45372, 48876, 53850, 57287, 63016, 65998, 70003, 71496, 75370, 83640]

plt.figure(figsize=(10, 5))
plt.plot(departments, earnings, marker='o', linestyle='-', color='b', label='Earnings (Lacs)')
plt.plot(departments, expenditures, marker='s', linestyle='--', color='r', label='Expenditures (Lacs)')

plt.xlabel("Departments")
plt.ylabel("Amount (Lacs)")
plt.title("Earnings vs Expenditures Department-wise")
plt.legend()
plt.grid(True)
plt.show()
```



2. Visualize the earning, expenditure and saving department wise in three graphs. These graphs share the y-axis scale. (Line graph)

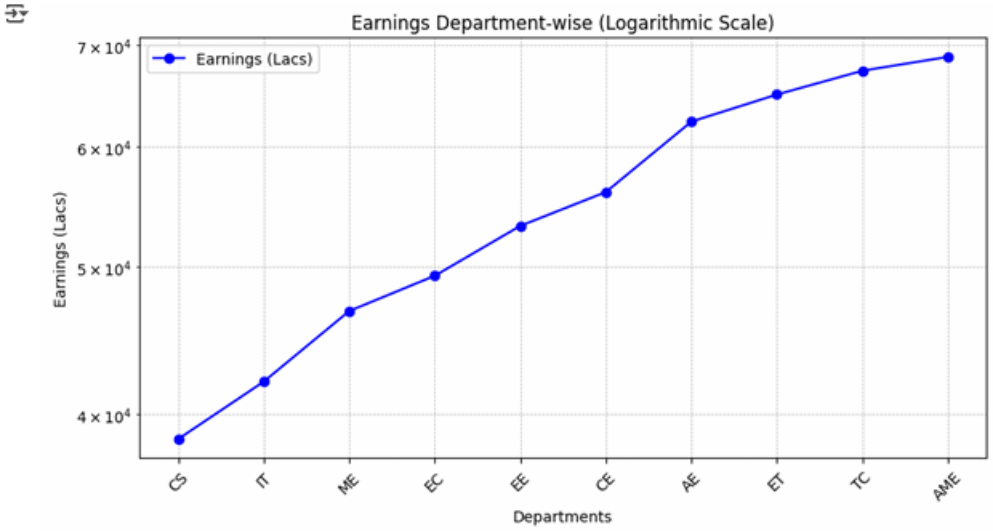
```
savings = [earnings[i] - expenditures[i] for i in range(len(earnings))]
fig, axes = plt.subplots(3, 1, figsize=(10, 15), sharex=True, sharey=True)

axes[0].plot(departments, earnings, marker='o', linestyle='-', color='b', label='Earnings (Lacs)')
axes[0].set_title("Earnings Department-wise")
axes[0].legend()
axes[0].grid(True)

axes[1].plot(departments, expenditures, marker='s', linestyle='--', color='r', label='Expenditures (Lacs)')
axes[1].set_title("Expenditures Department-wise")
axes[1].legend()
axes[1].grid(True)

axes[2].plot(departments, savings, marker='^', linestyle='-.', color='g', label='Savings (Lacs)')
axes[2].set_title("Savings Department-wise")
axes[2].legend()
axes[2].grid(True)
```

```
plt.xlabel("Departments")
plt.tight_layout()
plt.show()
```



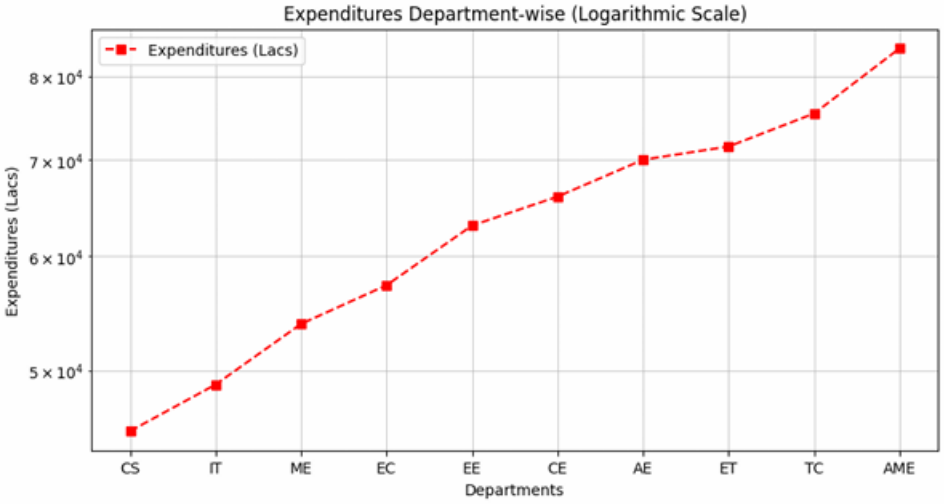
3. Visualize the department's earning using a line graph. Consider the y-axis as a logarithmic scale.

```
import numpy as np

departments = ['CS', 'IT', 'ME', 'EC', 'EE', 'CE', 'AE', 'ET', 'TC', 'AME']
earnings = [38496, 42000, 46752, 49320, 53200, 56000, 62316, 64928, 67317, 68748]

plt.figure(figsize=(10, 5))
plt.plot(departments, earnings, marker='o', color='b', label='Earnings (Lacs)')
plt.yscale('log')

plt.xlabel("Departments")
plt.ylabel("Earnings (Lacs)")
plt.title("Earnings Department-wise (Logarithmic Scale)")
plt.legend()
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()
```



4. Visualize the department's Expenditure using a line graph. Consider the y-axis as a logarithmic scale. Put minor scale on y-axis.

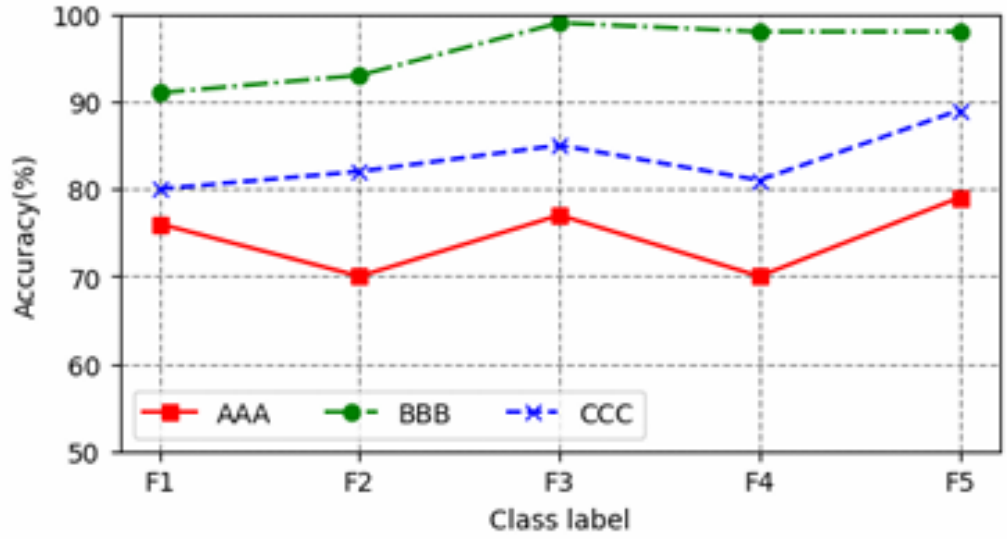
```
departments = ['CS', 'IT', 'ME', 'EC', 'EE', 'CE', 'AE', 'ET', 'TC', 'AME']
expenditures = [45372, 48876, 53850, 57287, 63016, 65998, 70003, 71496, 75370, 83640]

plt.figure(figsize=(10, 5))
plt.plot(departments, expenditures, marker='s', linestyle='--', color='r', label='Expenditures (Lacs)')
plt.yscale('log')


plt.xlabel("Departments")
plt.ylabel("Expenditures (Lacs)")
plt.title("Expenditures Department-wise (Logarithmic Scale)")
plt.legend()
plt.grid(True, which='both', linewidth=0.5)
plt.show()
```



Exp-2: Regenerate this graph using the matplotlib library.

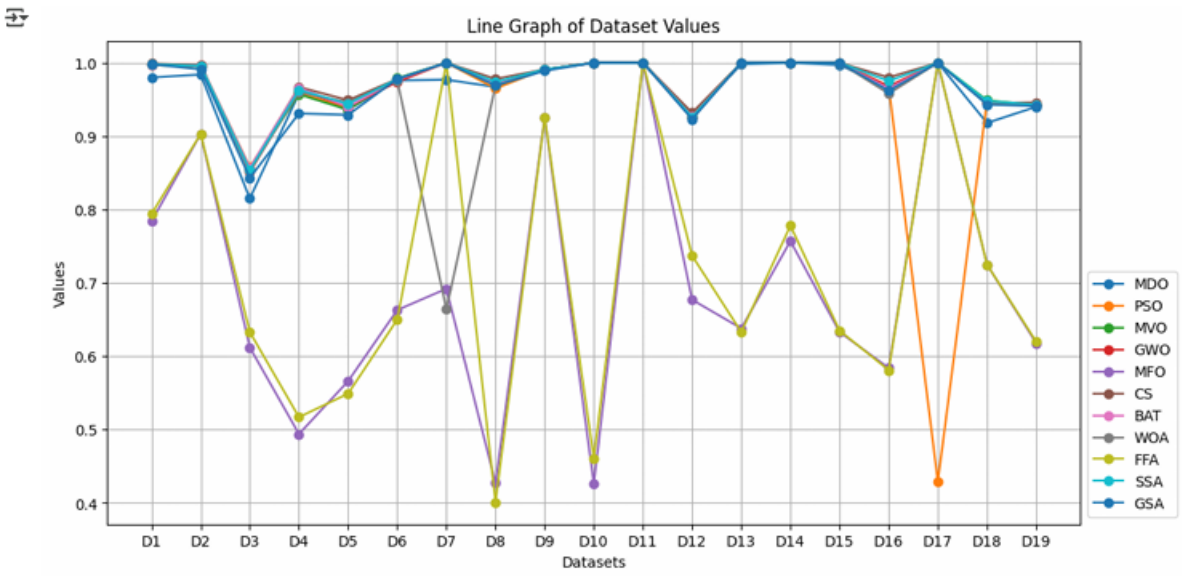
```
AAA = [76,70,77,70,79]
BBB = [91,93,99,98,98]
CCC = [80,82,85,81,89]

plt.figure(figsize=(6,3))
plt.ylim(50,100)

plt.plot(AAA,label='AAA',color='red',marker='s')
plt.plot(BBB,label='BBB',color='green',marker='o',linestyle='dashdot')
plt.plot(CCC,label='CCC',color='blue',marker='x',linestyle='dashed')

plt.grid(True)
plt.legend(ncol=3)

plt.xticks(np.arange(5), ('F1', 'F2', 'F3', 'F4', 'F5'))
plt.grid(linestyle='dashed',color='black',alpha=0.5)
plt.ylabel("Accuracy(%)")
plt.xlabel("Class label")
```

```
Text(0.5, 0, 'Class label')
```



Exp-3: Consider the dataset below ad plot the line graph below.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Dataset from the provided image
data = {
    "MDO": [0.98, 0.984, 0.815, 0.958, 0.936, 0.976, 0.977, 0.967, 0.991, 1, 1, 0.927, 0.998, 1, 0.997, 0.966, 1, 0.918, 0.94],
    "PSO": [0.998, 0.996, 0.858, 0.959, 0.938, 0.978, 1, 0.965, 0.991, 1, 1, 0.926, 1, 1, 1, 0.968, 0.429, 0.948, 0.942],
    "MVO": [0.997, 0.995, 0.856, 0.957, 0.936, 0.979, 1, 0.97, 0.991, 1, 1, 0.929, 1, 1, 1, 0.965, 1, 0.949, 0.942],
    "GWO": [0.998, 0.997, 0.851, 0.963, 0.94, 0.974, 1, 0.971, 0.991, 1, 1, 0.925, 1, 1, 1, 0.968, 1, 0.945, 0.942],

    "MFO": [0.784, 0.903, 0.612, 0.494, 0.566, 0.663, 0.692, 0.427, 0.926, 0.426, 1, 0.677, 0.638, 0.757, 0.633, 0.584, 1, 0.725, 0.618],
    "CS": [0.999, 0.997, 0.856, 0.967, 0.949, 0.978, 1, 0.978, 0.991, 1, 1, 0.932, 1, 1, 1, 0.98, 1, 0.945, 0.946],
    "BAT": [0.997, 0.996, 0.857, 0.966, 0.94, 0.977, 1, 0.971, 0.991, 1, 1, 0.927, 1, 1, 1, 0.965, 1, 0.945, 0.942],
    "WOA": [0.998, 0.995, 0.852, 0.96, 0.946, 0.976, 0.664, 0.969, 0.991, 1, 1, 0.926, 1, 1, 1, 0.958, 1, 0.945, 0.943],
    "FFA": [0.794, 0.903, 0.633, 0.517, 0.549, 0.65, 1, 0.401, 0.926, 0.461, 1, 0.737, 0.633, 0.778, 0.635, 0.581, 1, 0.725, 0.62],
    "SSA": [0.998, 0.996, 0.854, 0.963, 0.944, 0.978, 1, 0.974, 0.991, 1, 1, 0.926, 1, 1, 1, 0.975, 1, 0.948, 0.943],
    "GSA": [0.998, 0.991, 0.843, 0.931, 0.929, 0.978, 1, 0.969, 0.989, 1, 1, 0.923, 1, 1, 1, 0.962, 1, 0.943, 0.941]
}

# Convert dictionary to DataFrame
df = pd.DataFrame(data, index=[f"D{i}" for i in range(1, 20)])

# Plot the line graph
plt.figure(figsize=(12, 6))
for column in df.columns:
    plt.plot(df.index, df[column], marker='o', linestyle='-', label=column)

plt.xlabel("Datasets")
plt.ylabel("Values")
plt.title("Line Graph of Dataset Values")
plt.legend(loc='lower left', bbox_to_anchor=(1, 0))
plt.grid(True)
plt.show()
```



Lab Experiment 6

Objective:

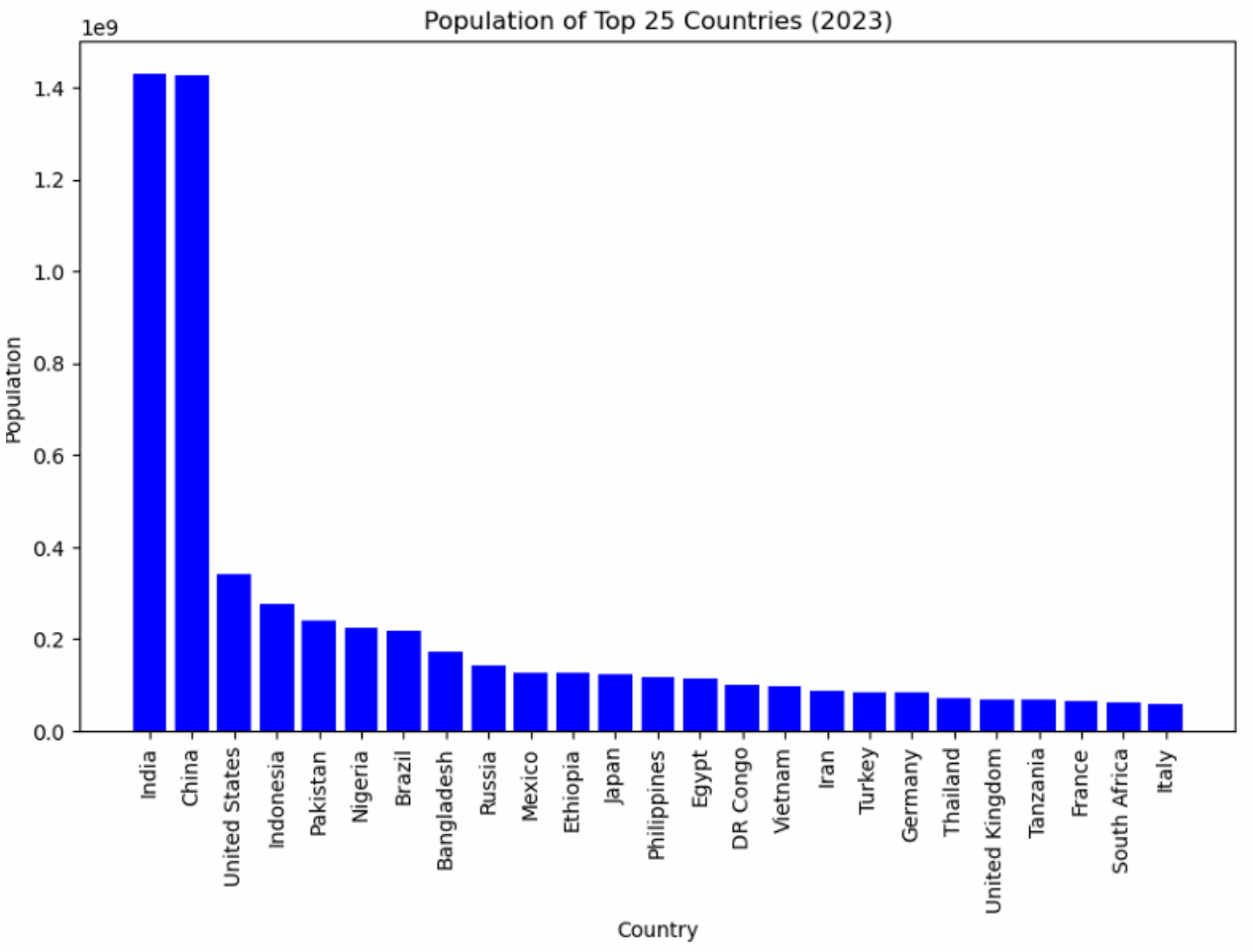
- Basic data visualization using Matplotlib package.

Visualise the data by drawing a valid graph

Consider the world population data. (population.csv)

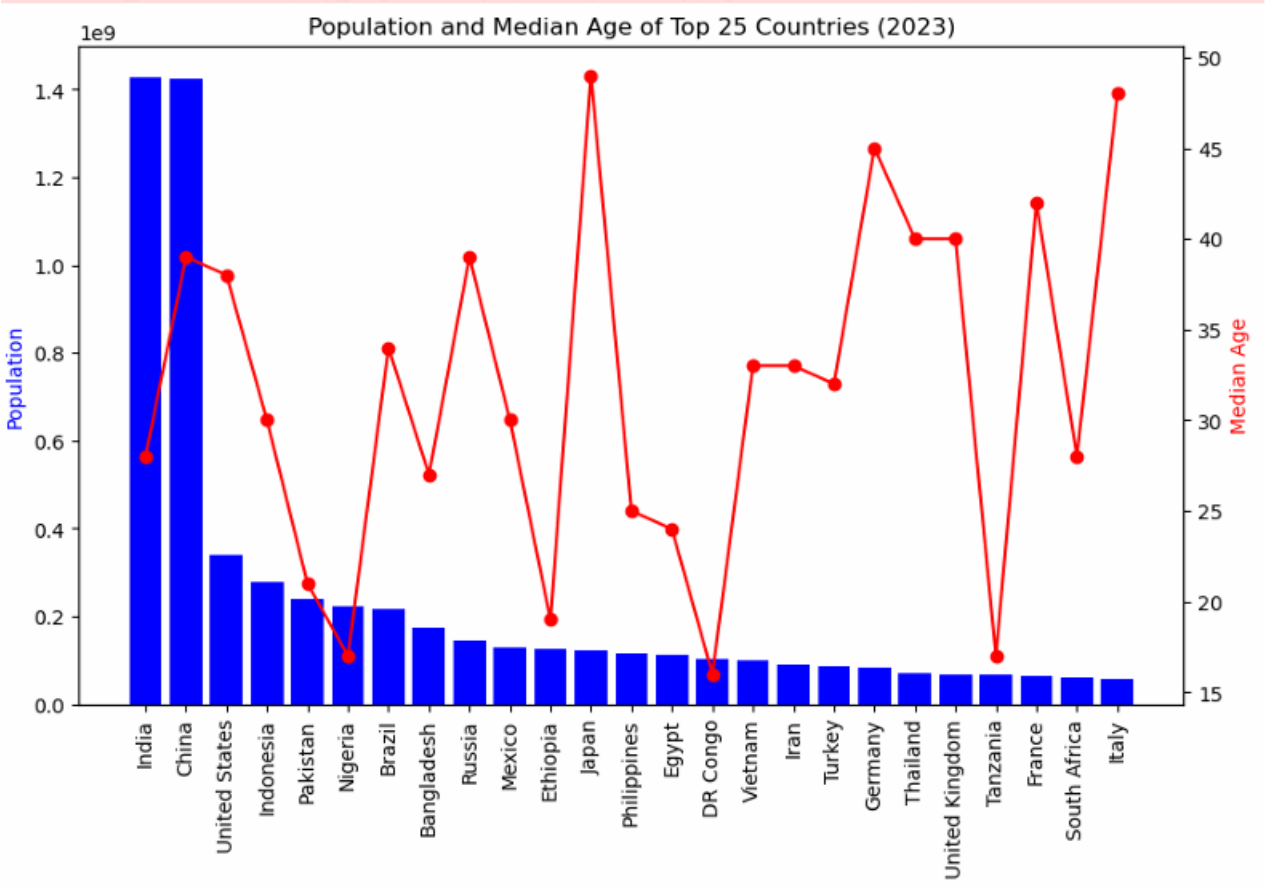
1.Visualize the population of the top 25 countries (population-wise)

```
top_25 = df.sort_values(by='Population (2023)', ascending=False).head(25)
plt.figure(figsize=(10, 6))
plt.bar(top_25['Country (or dependency)'], top_25['Population (2023)'], color='blue')
plt.xticks(rotation=90)
plt.title('Population of Top 25 Countries (2023)')
plt.xlabel('Country')
plt.ylabel('Population')
plt.show()
```



2.Visualize the population and middle age of the top 25 countries (population-wise)

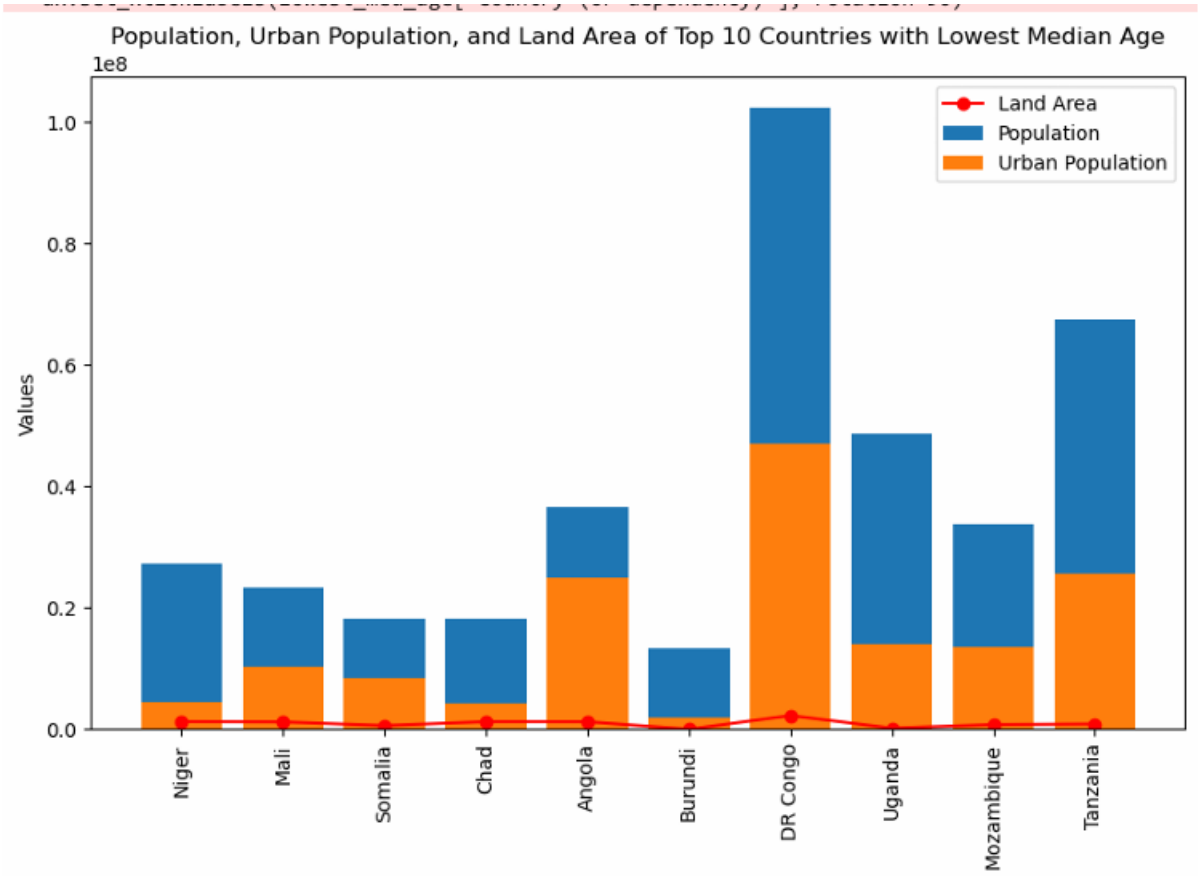
```
fig, ax1 = plt.subplots(figsize=(10, 6))
ax1.bar(top_25['Country (or dependency)'], top_25['Population (2023)'], color='blue')
ax1.set_xticklabels(top_25['Country (or dependency)'], rotation=90)
ax1.set_ylabel('Population', color='blue')
ax2 = ax1.twinx()
ax2.plot(top_25['Country (or dependency)'], top_25['Med. Age'], color='red', marker='o')
ax2.set_ylabel('Median Age', color='red')
plt.title('Population and Median Age of Top 25 Countries (2023)')
plt.show()
```



3.Visualize the total population, urban population, and Land area of the top 10 countries having the lowest middle age.

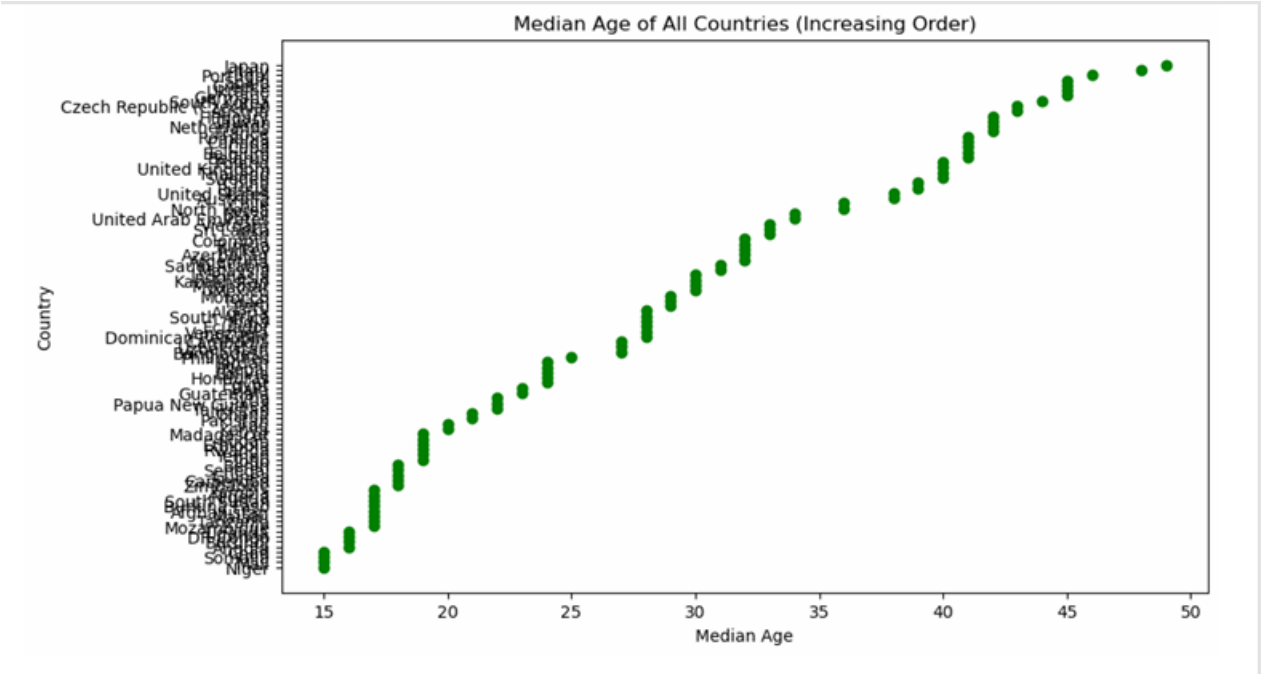
```
lowest_med_age = df.sort_values(by='Med. Age').head(10)
fig, ax = plt.subplots(figsize=(10, 6))
ax.bar(lowest_med_age['Country (or dependency)'], lowest_med_age['Population (2023)'], label=
ax.bar(lowest_med_age['Country (or dependency)'], lowest_med_age['Urban Pop %'] * lowest_med_
ax.plot(lowest_med_age['Country (or dependency)'], lowest_med_age['Land Area(Km²)'], color='r
ax.set_xticklabels(lowest_med_age['Country (or dependency)'], rotation=90)
ax.set_ylabel('Values')
ax.legend()
```

```
plt.title('Population, Urban Population, and Land Area of Top 10 Countries with Lowest Median
plt.show()
```



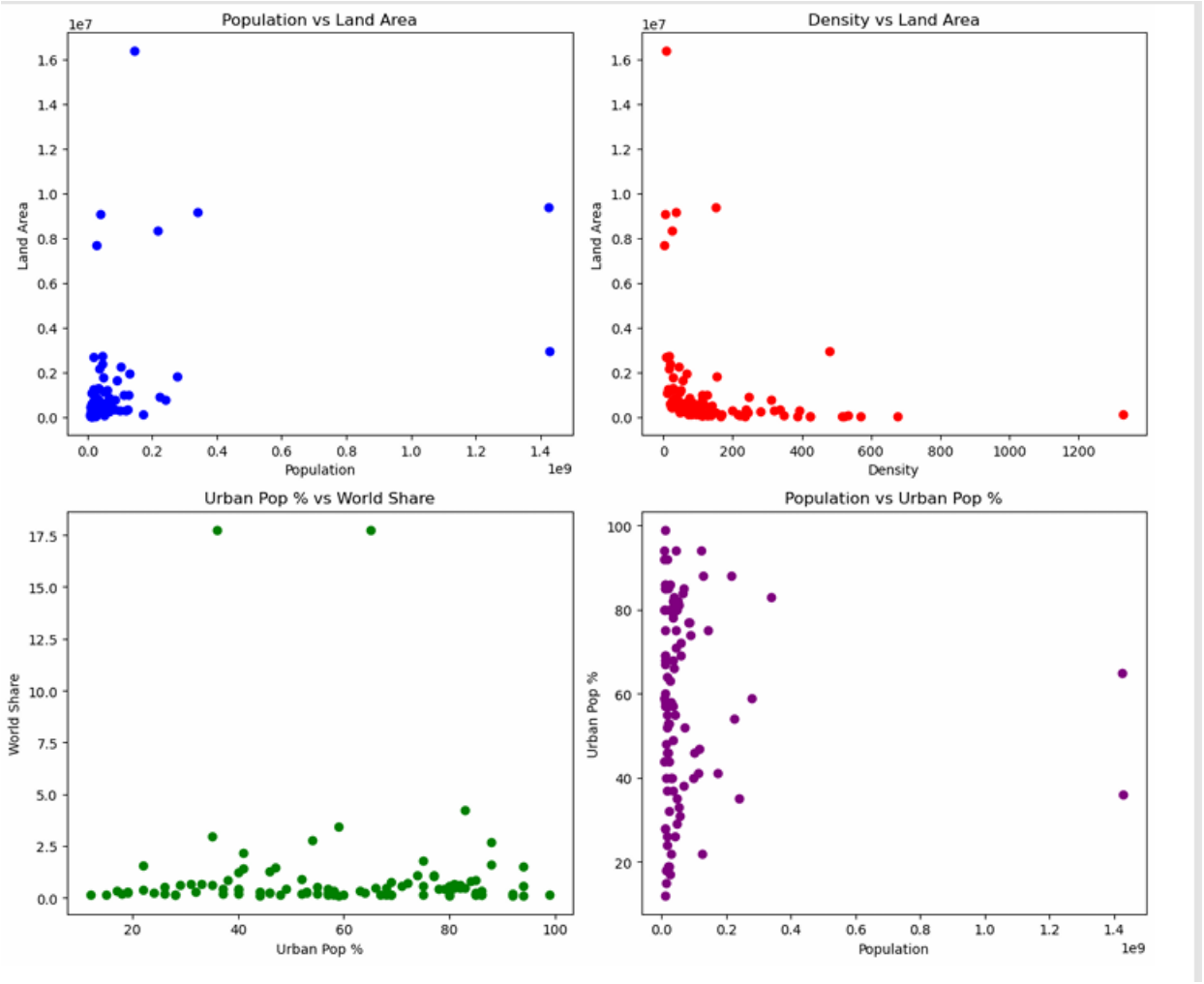
4.Visualize the middle age (increasing order) of all the countries using a dot plot.

```
sorted_med_age = df.sort_values(by='Med. Age')
plt.figure(figsize=(10, 6))
plt.scatter(sorted_med_age['Med. Age'], range(len(sorted_med_age)), color='green')
plt.yticks(range(len(sorted_med_age)), sorted_med_age['Country (or dependency)'])
plt.title('Median Age of All Countries (Increasing Order)')
plt.xlabel('Median Age')
plt.ylabel('Country')
plt.show()
```



5.Visualize the correlation among the following attributes by creating 4 subplots (2x2 Grid): (a). Population and Land area (b). Density and land Area (c). Urban Pop and world share (d). Population and Urban Pop

```
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# a. Population and Land area
axes[0, 0].scatter(df['Population (2023)'], df['Land Area(Km²)'], color='blue')
axes[0, 0].set_title('Population vs Land Area')
axes[0, 0].set_xlabel('Population')
axes[0, 0].set_ylabel('Land Area')

# b. Density and Land Area
axes[0, 1].scatter(df['Density(P/Km²)'], df['Land Area(Km²)'], color='red')
axes[0, 1].set_title('Density vs Land Area')
axes[0, 1].set_xlabel('Density')
axes[0, 1].set_ylabel('Land Area')

# c. Urban Pop and World Share
axes[1, 0].scatter(df['Urban Pop %'], df['World Share'], color='green')
axes[1, 0].set_title('Urban Pop % vs World Share')
axes[1, 0].set_xlabel('Urban Pop %')
axes[1, 0].set_ylabel('World Share')

# d. Population and Urban Pop
axes[1, 1].scatter(df['Population (2023)'], df['Urban Pop %'], color='purple')
axes[1, 1].set_title('Population vs Urban Pop %')
axes[1, 1].set_xlabel('Population')
axes[1, 1].set_ylabel('Urban Pop %')

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



6.Consider the data given below and plot in the polar coordinate system.

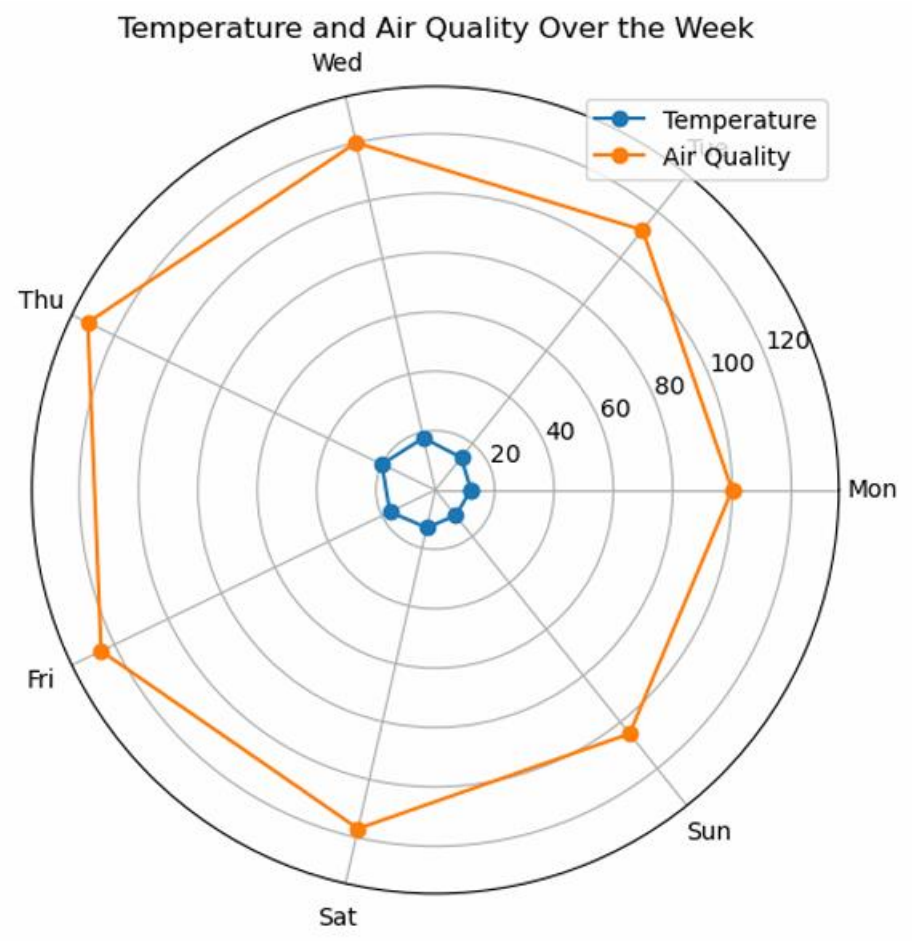
Days = [Mon, Tue, Wed, Thu, Fri, Sat, Sun]

Temp= [12, 14, 18, 20, 17, 13, 11]

Air Quality = [100, 112, 120, 130, 125, 117, 105]

- List item
- List item

```
days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
temp = [12, 14, 18, 20, 17, 13, 11]
air_quality = [100, 112, 120, 130, 125, 117, 105]

angles = np.linspace(0, 2 * np.pi, len(days), endpoint=False)
angles = np.concatenate((angles, [angles[0]]))  # Close the plot

fig, ax = plt.subplots(subplot_kw={'projection': 'polar'}, figsize=(6, 6))
ax.plot(angles, temp + [temp[0]], marker='o', label='Temperature')
ax.plot(angles, air_quality + [air_quality[0]], marker='o', label='Air Quality')
ax.set_xticks(angles[:-1])
ax.set_xticklabels(days)
ax.set_title('Temperature and Air Quality Over the Week')
ax.legend(loc='upper right')
plt.show()
```



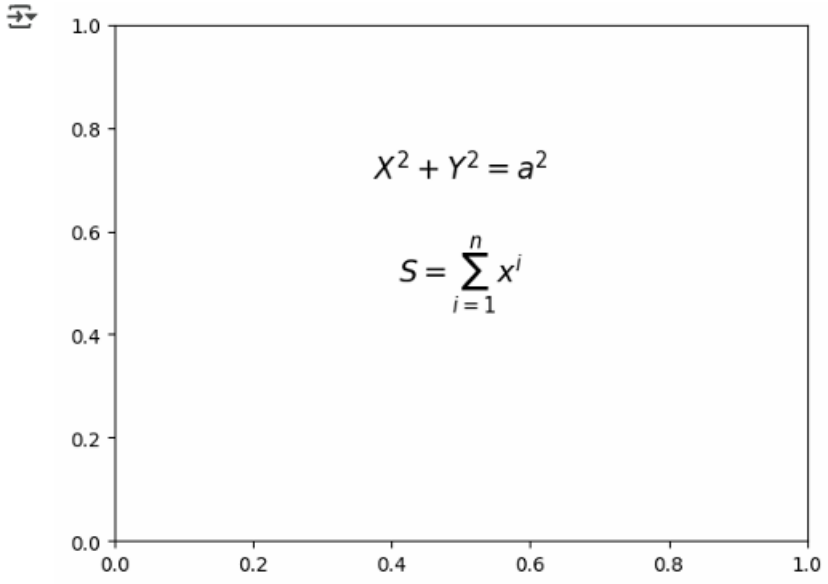
Lab Experiment 7

Objective:

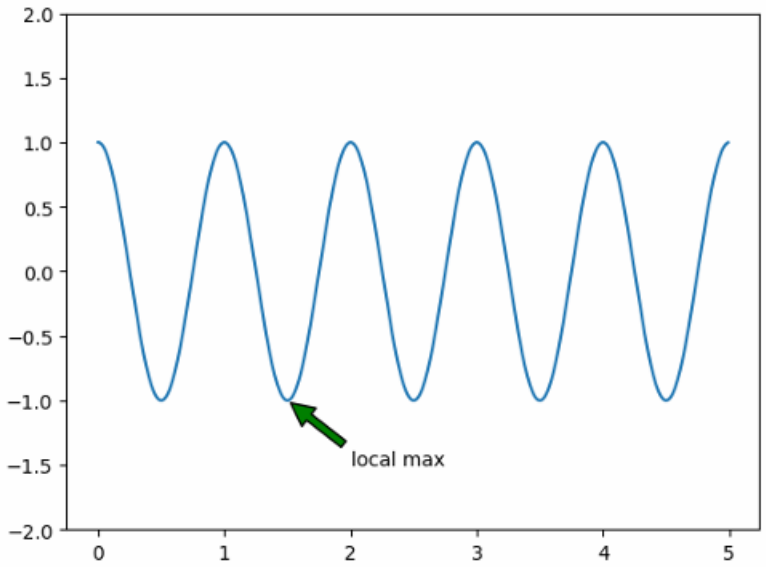
• Use of color, text, and annotation.

Q1: Regenerate the following visualizations:

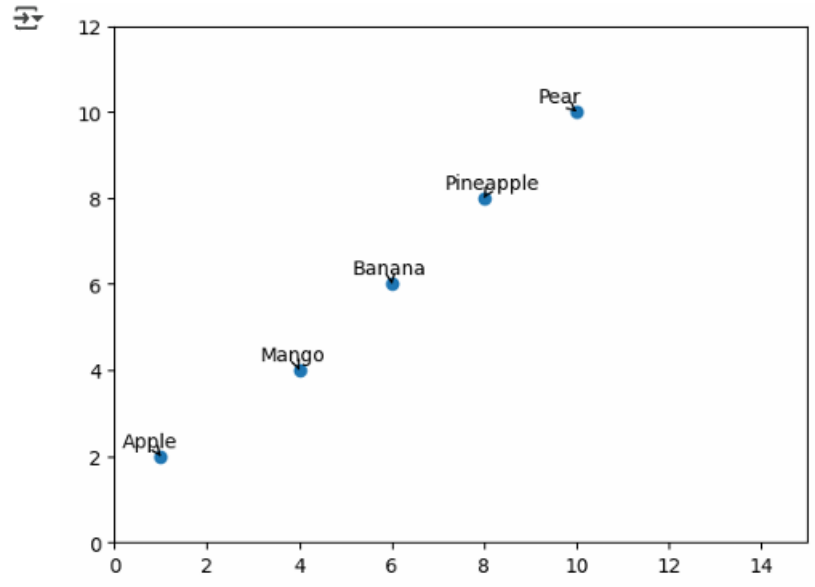
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()


ax.axis('on')


ax.text(0.5, 0.7, r'$X^2 + Y^2 = a^2$', fontsize=15, ha='center')
ax.text(0.5, 0.5, r'$S = \sum_{i=1}^n x^i$', fontsize=15, ha='center')


plt.show()
```



```
import numpy as np
import matplotlib.pyplot as plt

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2 * np.pi * t)

plt.plot(t, s)

plt.annotate('local max', xy=(1.5, -1), xytext=(2, -1.5),
             arrowprops=dict(facecolor='green', shrink=0.05))


plt.ylim(-2, 2)


plt.show()
```

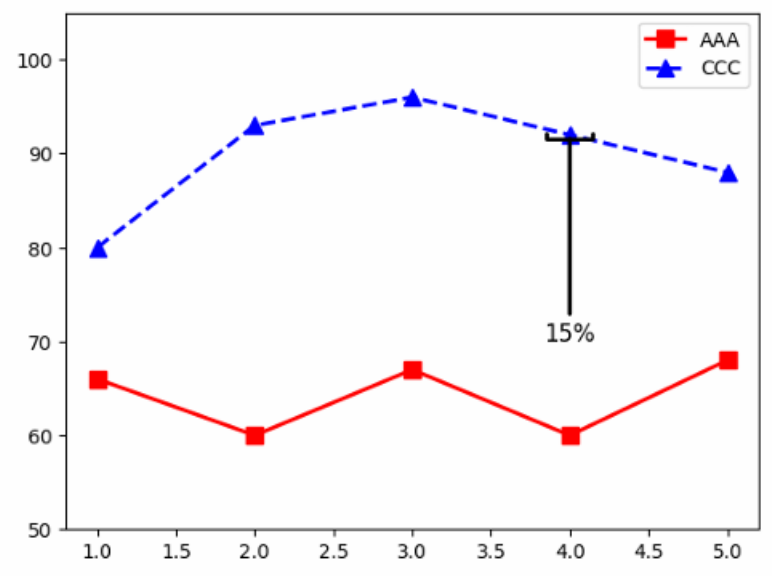


```
import matplotlib.pyplot as plt

x = [1, 4, 6, 8, 10]
y = [2, 4, 6, 8, 10]
labels = ['Apple', 'Mango', 'Banana', 'Pineapple', 'Pear']

plt.scatter(x, y)

for i, label in enumerate(labels):
    plt.annotate(label, (x[i], y[i]), xytext=(-20,5), textcoords='offset points',
                 arrowprops=dict(arrowstyle='->'))

plt.xlim(0, 15)
plt.ylim(0, 12)


plt.show()
```



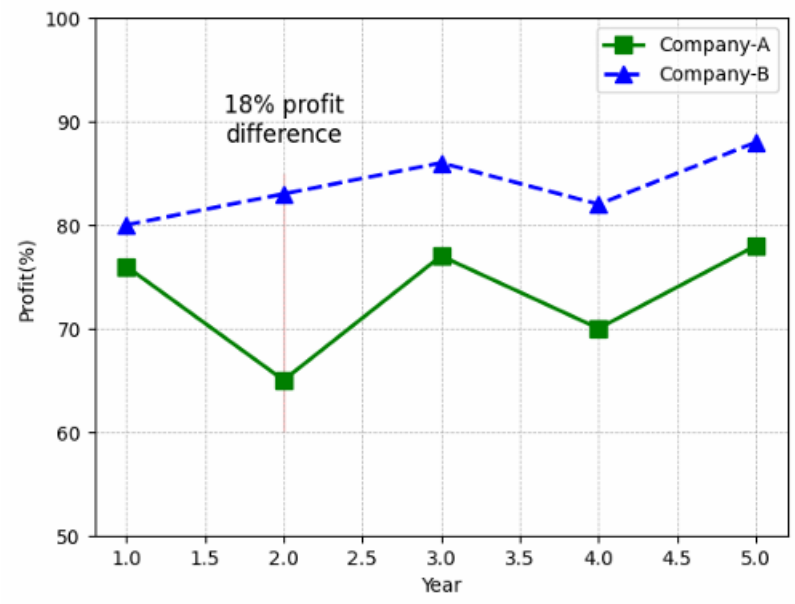
Q4: data = [[66, 60, 67, 60, 68],[80, 93, 96, 92, 88]]

```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([1, 2, 3, 4, 5])
data = [[66, 60, 67, 60, 68], [80, 93, 96, 92, 88]]


plt.plot(x, data[0], 'r-s', label="AAA", linewidth=2, markersize=8)


plt.plot(x, data[1], 'b--^', label="CCC", linewidth=2, markersize=8)


plt.annotate('15%', xy=(4, 92), xytext=(4, 70),
             arrowprops=dict(facecolor='black', arrowstyle='-[', lw=2),
             fontsize=12, ha='center')


plt.legend()
plt.ylim(50, 105)
plt.show()
```



Q5: data = [[76, 65, 77, 70, 78],[80, 83, 86, 82, 88]]

```
import matplotlib.pyplot as plt
import numpy as np


x = np.array([1, 2, 3, 4, 5])
data = [[76, 65, 77, 70, 78], [80, 83, 86, 82, 88]]

plt.plot(x, data[0], 'g-s', label="Company-A", linewidth=2, markersize=8)


plt.plot(x, data[1], 'b--^', label="Company-B", linewidth=2, markersize=8)


plt.fill_between([2], 60, 85, color='red', alpha=0.3)


plt.text(2, 88, "18% profit\ndifference", fontsize=12, ha='center')


plt.xlabel("Year")
plt.ylabel("Profit(%)")
plt.grid(True, linestyle="--", linewidth=0.5)
plt.legend()
plt.ylim(50, 100)

plt.show()
```



Lab Experiment 8

Objective:

Analyse the dataset features through visualization. The visualization will demonstrate the concepts like Figure, figure size, subplots, ticks, grid, markers, legends, axes, scale, save quality image.

Figure 1: Class distribution of survived and died

```
# 1. Class distribution of died and survived
plt.subplot(2, 2, 1)
sns.countplot(x='Pclass', hue='Survived', data=titanic, palette='viridis')
plt.title('Survival by Passenger Class', fontsize=12)
plt.xlabel('Passenger Class', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.legend(['Died', 'Survived'], title='Survival Status')
```

```
<matplotlib.legend.Legend at 0x79031e557890>
```

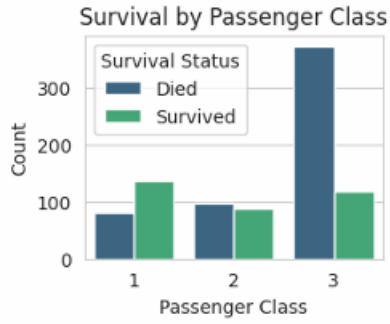


Figure 2: Age distribution by survival and sex

```
# 2. Age distribution by Survival and Sex
plt.subplot(2, 2, 2)
sns.boxplot(x='Sex', y='Age', hue='Survived', data=titanic, palette='viridis')
plt.title('Age Distribution by Sex and Survival', fontsize=12)
plt.xlabel('Sex', fontsize=10)
plt.ylabel('Age', fontsize=10)
plt.legend(title='Survival Status')
```

```
<matplotlib.legend.Legend at 0x79031e4e2990>
```

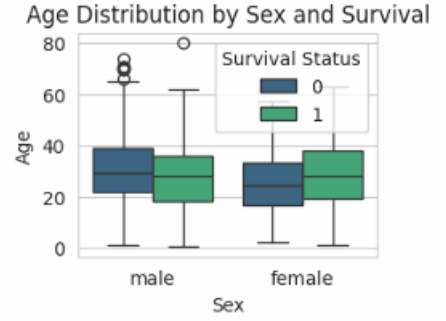


Figure 3: Histogram of Age

```
# 3. Histogram of Age
plt.subplot(2, 2, 3)
sns.histplot(titanic['Age'].dropna(), bins=30, kde=False, color='darkblue')
plt.title('Age Distribution (Histogram)', fontsize=12)
plt.xlabel('Age', fontsize=10)
plt.ylabel('Count', fontsize=10)
```

```
Text(0, 0.5, 'Count')
```

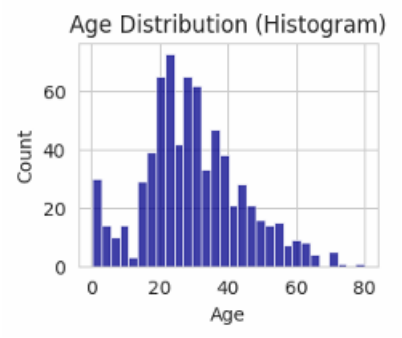
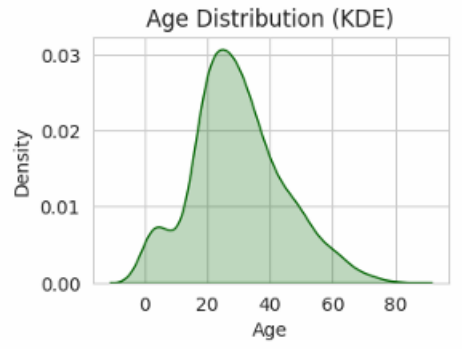


Figure 4: Kernel Density Plot of Age

```
# 4. Kernel density plot of age
plt.subplot(2, 2, 4)
sns.kdeplot(titanic['Age'].dropna(), shade=True, color='darkgreen')
plt.title('Age Distribution (KDE)', fontsize=12)
plt.xlabel('Age', fontsize=10)
plt.ylabel('Density', fontsize=10)

plt.tight_layout()
plt.savefig('titanic_analysis_part1.png', dpi=300, bbox_inches='tight')
plt.show()

# Create a new figure for additional visualizations
plt.figure(figsize=(15, 10))
```

```
<ipython-input-7-2953e704bb3b>:3: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(titanic['Age'].dropna(), shade=True, color='darkgreen')
```




```
<Figure size 1500x1000 with 0 Axes>
<Figure size 1500x1000 with 0 Axes>
```

Figure 5: Density Plot of Age by Sex

```
# 5. Comparison of density plots of Age with different Sex
plt.subplot(2, 2, 1)
sns.kdeplot(titanic[titanic['Sex'] == 'male']['Age'].dropna(), label='Male', shade=True)
sns.kdeplot(titanic[titanic['Sex'] == 'female']['Age'].dropna(), label='Female', shade=True)
plt.title('Age Distribution by Sex', fontsize=12)
plt.xlabel('Age', fontsize=10)
plt.ylabel('Density', fontsize=10)
plt.legend()
```

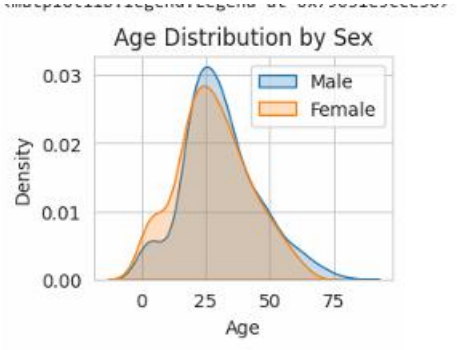


Figure 6: Fare distribution by Passenger Class

```
# 6. Fare distribution by class
plt.subplot(2, 2, 2)
sns.boxplot(x='Pclass', y='Fare', data=titanic, palette='viridis')
plt.title('Fare Distribution by Passenger Class', fontsize=12)
plt.xlabel('Passenger Class', fontsize=10)
plt.ylabel('Fare', fontsize=10)
```

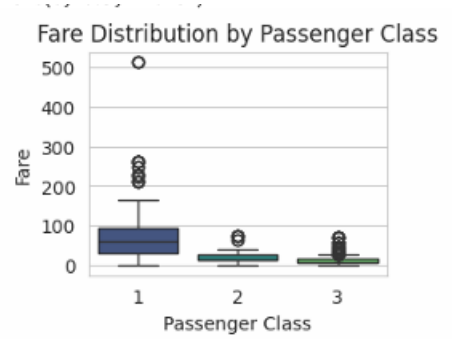
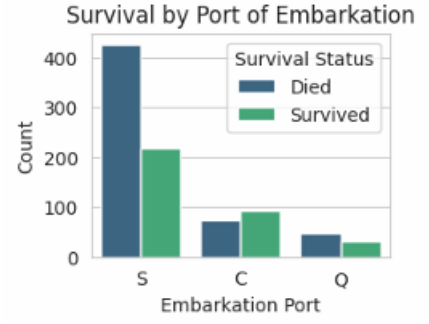


Figure 7: Port of Embarkation distribution

```
# 7. Survival rate by port of embarkation
plt.subplot(2, 2, 3)
sns.countplot(x='Embarked', hue='Survived', data=titanic, palette='viridis')
plt.title('Survival by Port of Embarkation', fontsize=12)
plt.xlabel('Embarkation Port', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.legend(['Died', 'Survived'], title='Survival Status')
```

```
<matplotlib.legend.Legend at 0x790319abc290>
```



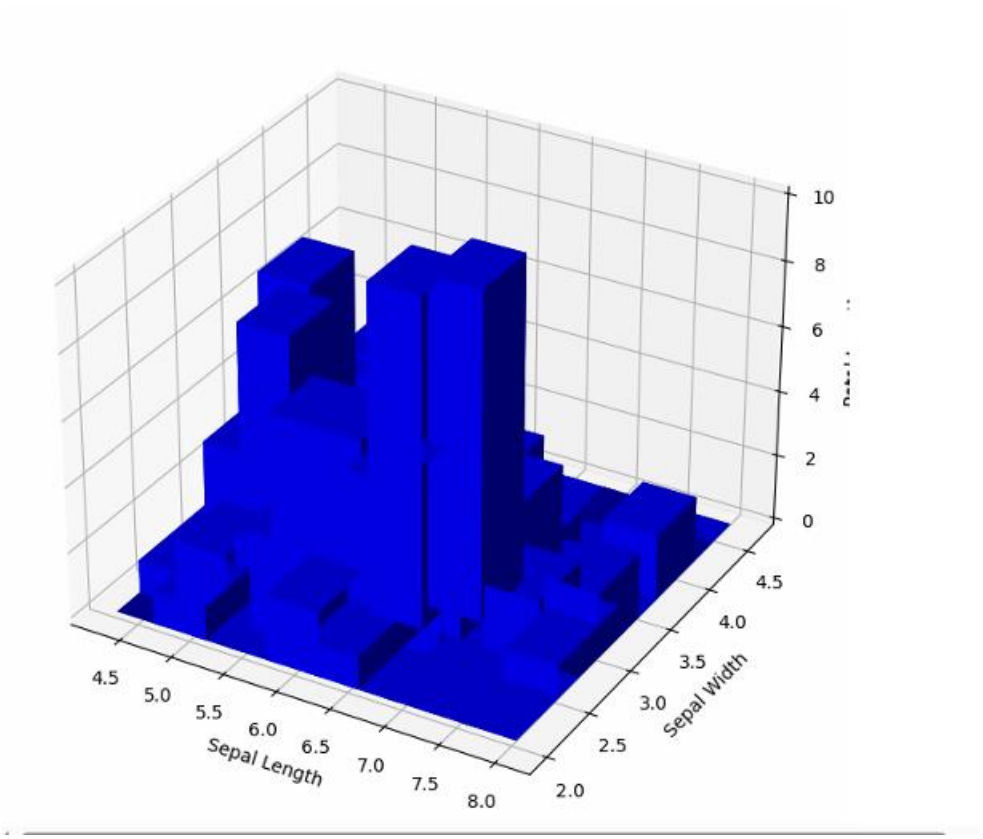
Lab Experiment 9

Objective:

• Create 3-D plots using the matplotlib library.

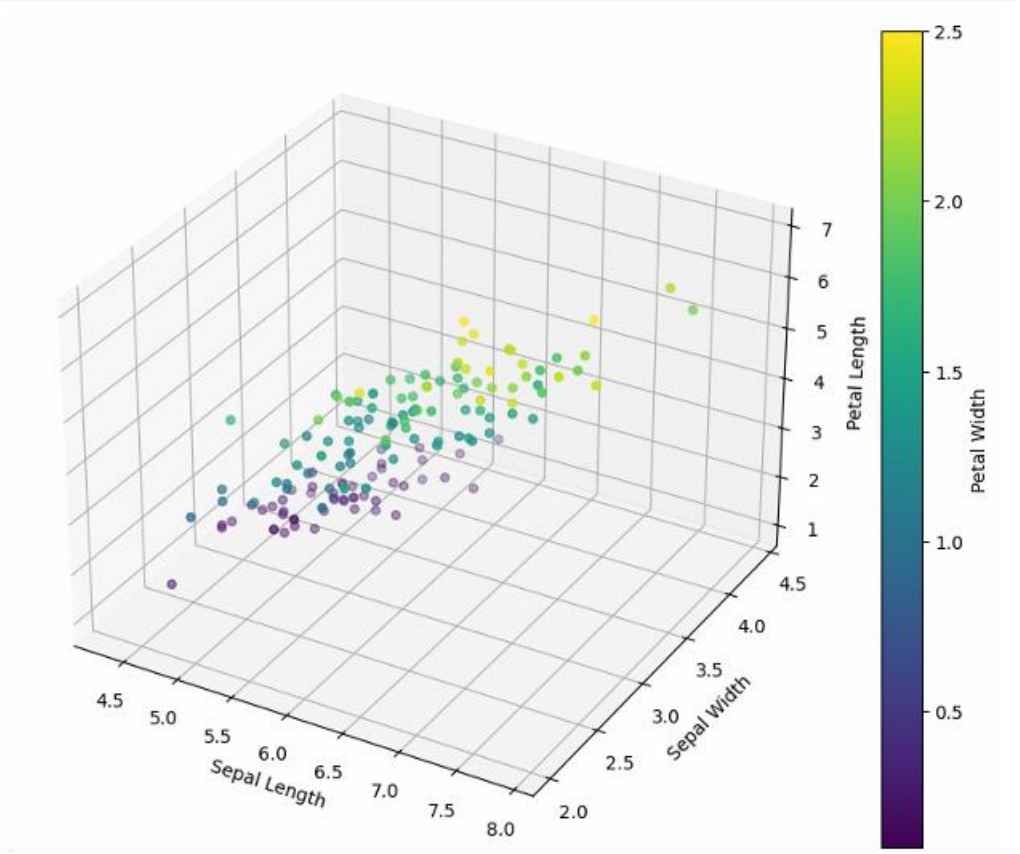
Consider the iris dataset having 150 samples with four features (sepal.length, sepal.width petal.length, petal.width) and three classes ('Setosa', 'Versicolor', 'Virginica'). Analyze the data through visualization.

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd

df = pd.read_csv('/content/iris.csv')

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

# Extract features
x = df['sepal.length']
y = df['sepal.width']
z = df['petal.length']

hist, xedges, yedges = np.histogram2d(x, y, bins=10)
xpos, ypos = np.meshgrid(xedges[:-1], yedges[:-1], indexing="ij")
xpos = xpos.flatten()
ypos = ypos.flatten()
zpos = np.zeros_like(xpos)
dx = dy = 0.5 * np.ones_like(zpos)
dz = hist.flatten()

ax.bar3d(xpos, ypos, zpos, dx, dy, dz, color='b', zsort='average')

ax.set_xlabel('Sepal Length')
ax.set_ylabel('Sepal Width')
ax.set_zlabel('Petal Length')

plt.show()
```



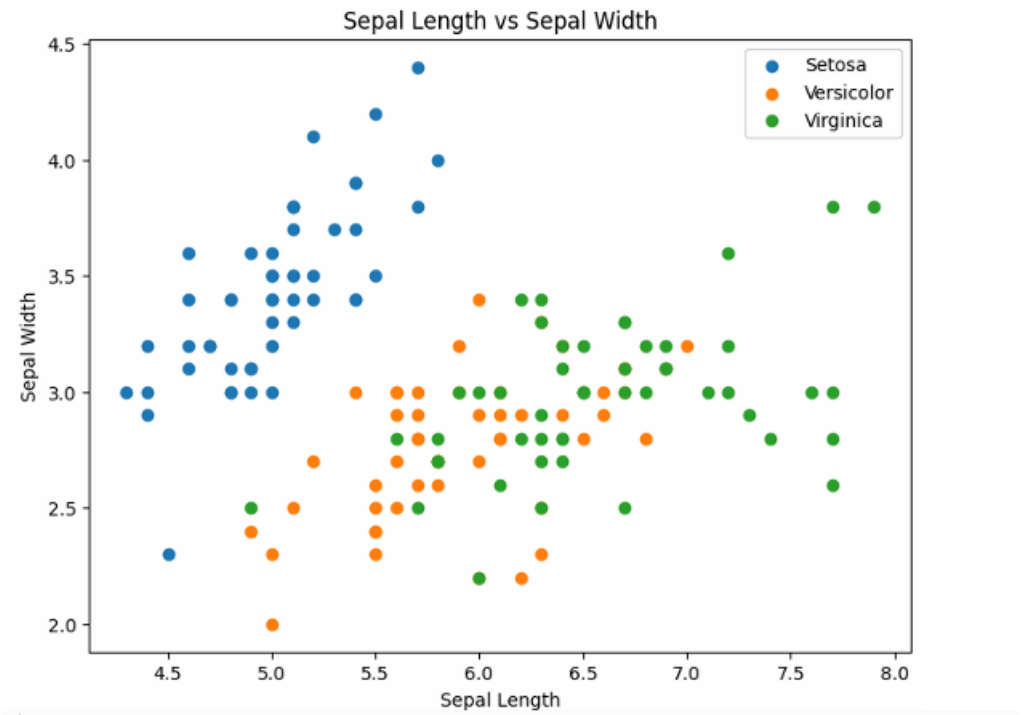
Q2. Plot the distribution of all four feature in single 3D plot. (Density estimation)

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


df = pd.read_csv('/content/iris.csv')

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

x = df['sepal.length']
y = df['sepal.width']
z = df['petal.length']
c = df['petal.width']

scatter = ax.scatter(x, y, z, c=c, cmap='viridis')
fig.colorbar(scatter, ax=ax, label='Petal Width')

ax.set_xlabel('Sepal Length')
ax.set_ylabel('Sepal Width')
ax.set_zlabel('Petal Length')

plt.show()
```



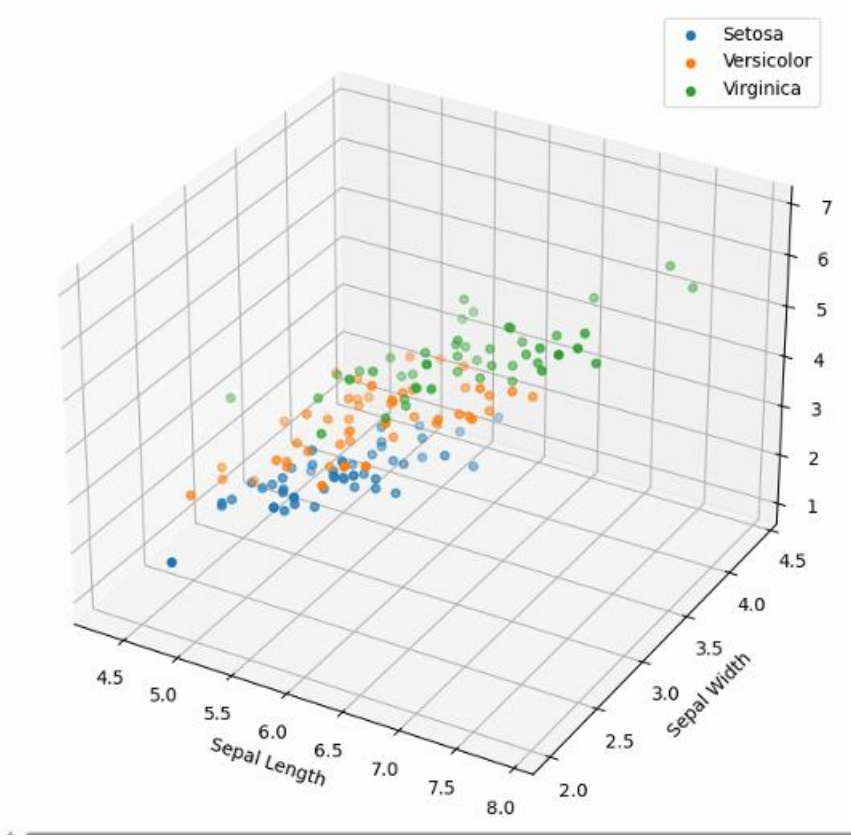
Q3. Visualize the distribution with respect to all three classes by considering two features.

```
import matplotlib.pyplot as plt


df = pd.read_csv('/content/iris.csv')

plt.figure(figsize=(8, 6))

for variety in df['variety'].unique():
    subset = df[df['variety'] == variety]
    plt.scatter(subset['sepal.length'], subset['sepal.width'], label=variety)

plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Sepal Length vs Sepal Width')
plt.legend()
plt.show()
```



Q4. Visualize the distribution with respect to all three classes by considering three features.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

df = pd.read_csv('/content/iris.csv')

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

for variety in df['variety'].unique():
    subset = df[df['variety'] == variety]
    ax.scatter(subset['sepal.length'], subset['sepal.width'], subset['petal.length'], label=variety)

ax.set_xlabel('Sepal Length')
ax.set_ylabel('Sepal Width')
ax.set_zlabel('Petal Length')
plt.legend()
plt.show()
```



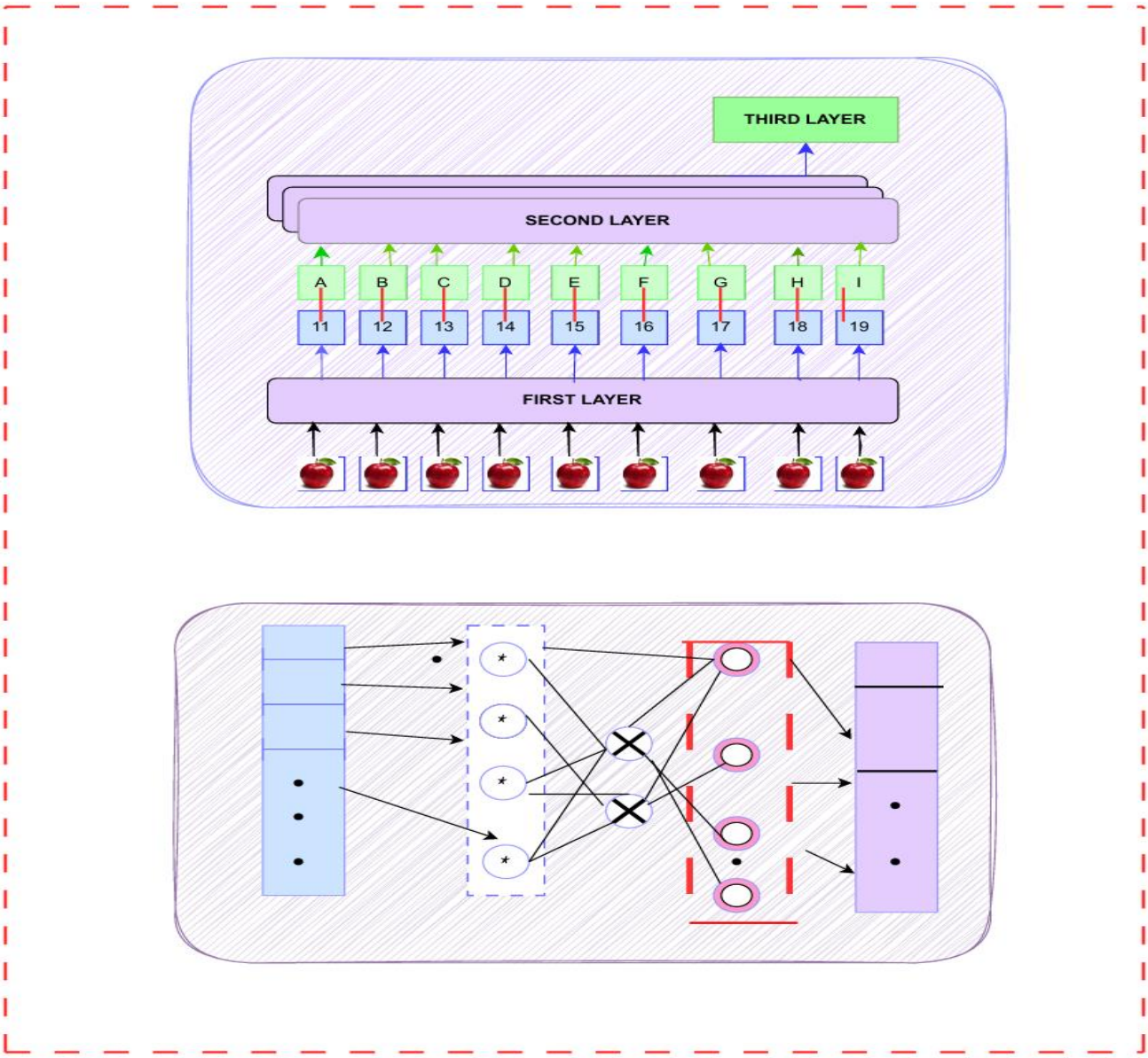
Q5. Visualize the data distribution using violinplot with respect all the classes by considering the feature separately (sepal.length, sepal.width petal.length, petal.width).

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv('/content/iris.csv')

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

features = ['sepal.length', 'sepal.width', 'petal.length', 'petal.width']
titles = ['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width']

for i, (feature, title) in enumerate(zip(features, titles)):
    ax = axes[i // 2, i % 2]
    for variety in df['variety'].unique():
        subset = df[df['variety'] == variety]
        ax.violinplot(subset[feature], positions=[list(df['variety'].unique()).index(variety)], showmeans=True)
    ax.set_xticks(range(len(df['variety'].unique())))
    ax.set_xticklabels(df['variety'].unique())
    ax.set_title(title)

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

Lab Experiment 10

Objective: To design a graphical image.

Q1. Create the given graphical image below using the graphical tool draw.io



Lab Experiment 11

Objective: To understand complex data visualization

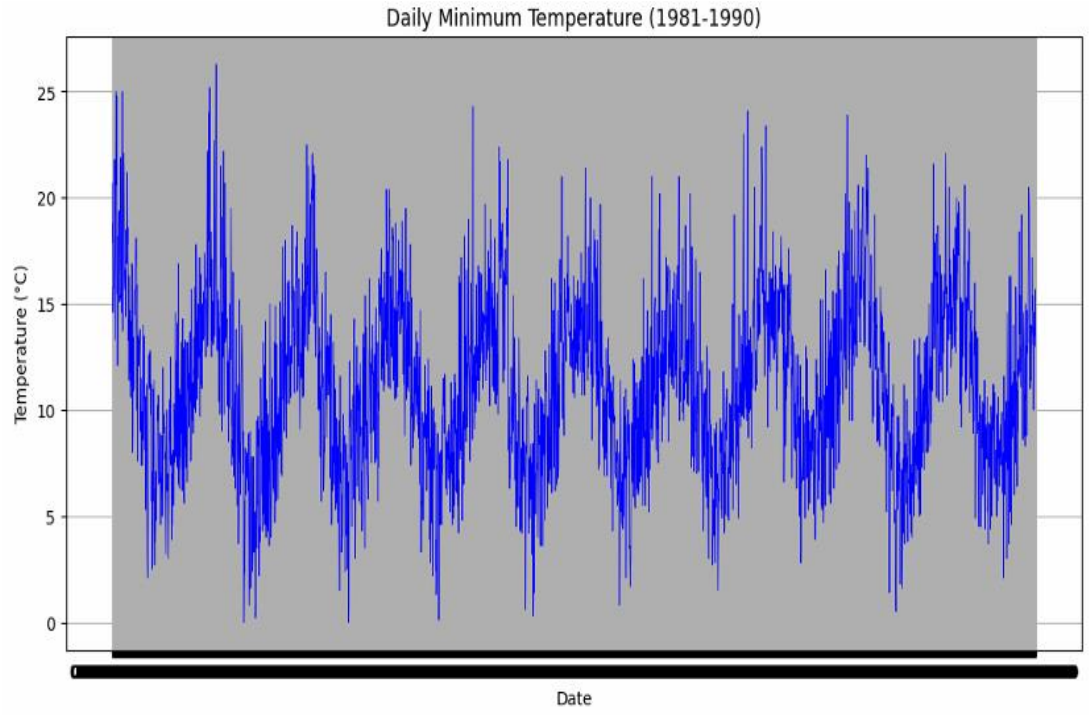
- Time series data
- Images data
- Geospatial Data

Q1. Consider the daily minimum temperature dataset and do the following:

Data source: https://raw.githubusercontent.com/jbrownlee/Datasets/master/daily-min-temperatures.csv

- Visualize the day-wise data from 1981 to 1990.

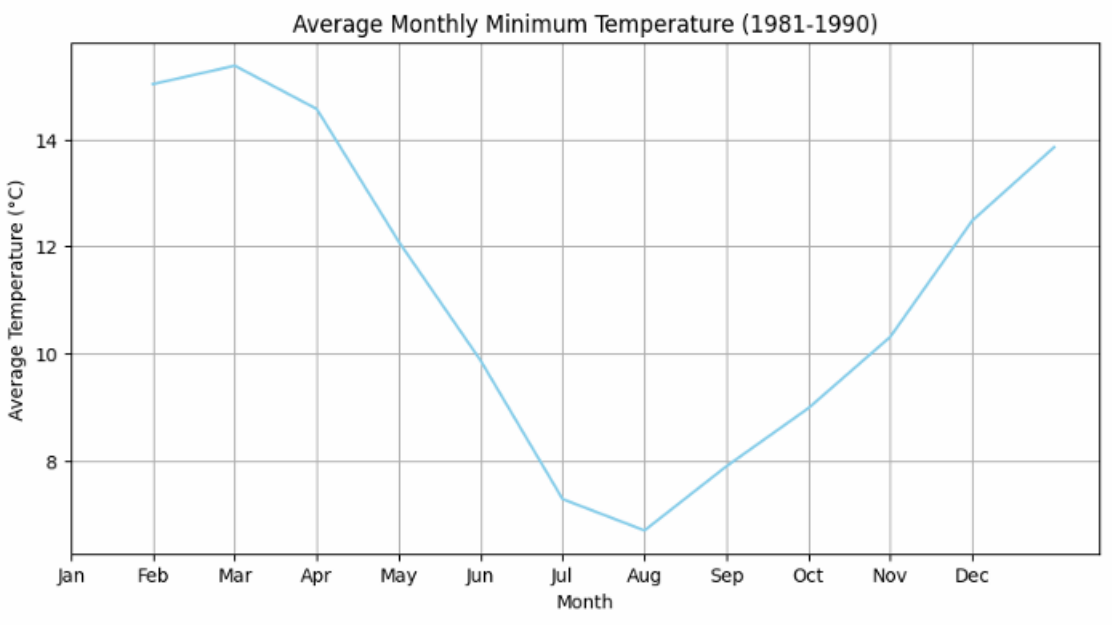
```
plt.figure(figsize=(12, 6))
plt.plot(df['Date'], df['Temp'], color='blue', linewidth=0.5)
plt.title('Daily Minimum Temperature (1981-1990)')
plt.xlabel('Date')
plt.ylabel('Temperature (°C)')
plt.grid(True)
plt.show()
```



- Visualize the average month-wise data from 1981 to 1990.

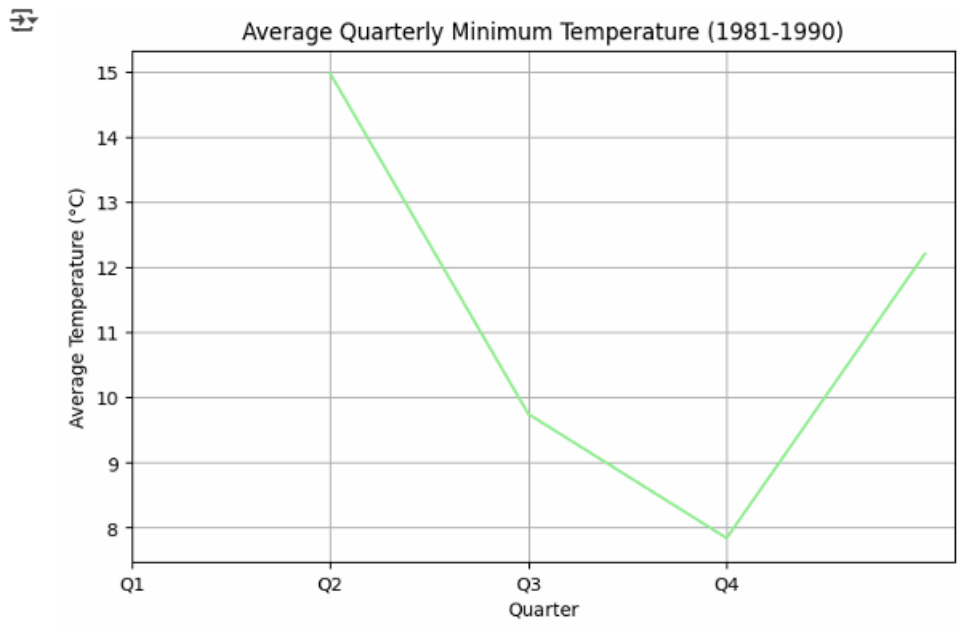
```
df['Month'] = df['Date'].dt.month
monthly_avg = df.groupby('Month')['Temp'].mean()
```

```
plt.figure(figsize=(10, 5))
monthly_avg.plot(kind='line', color='skyblue')
plt.title('Average Monthly Minimum Temperature (1981-1990)')
plt.xlabel('Month')
plt.ylabel('Average Temperature (°C)')
plt.xticks(range(12), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.grid(True)
plt.show()
```



- Visualize the average quarter-wise data from 1981 to 1990 and highlight the highest

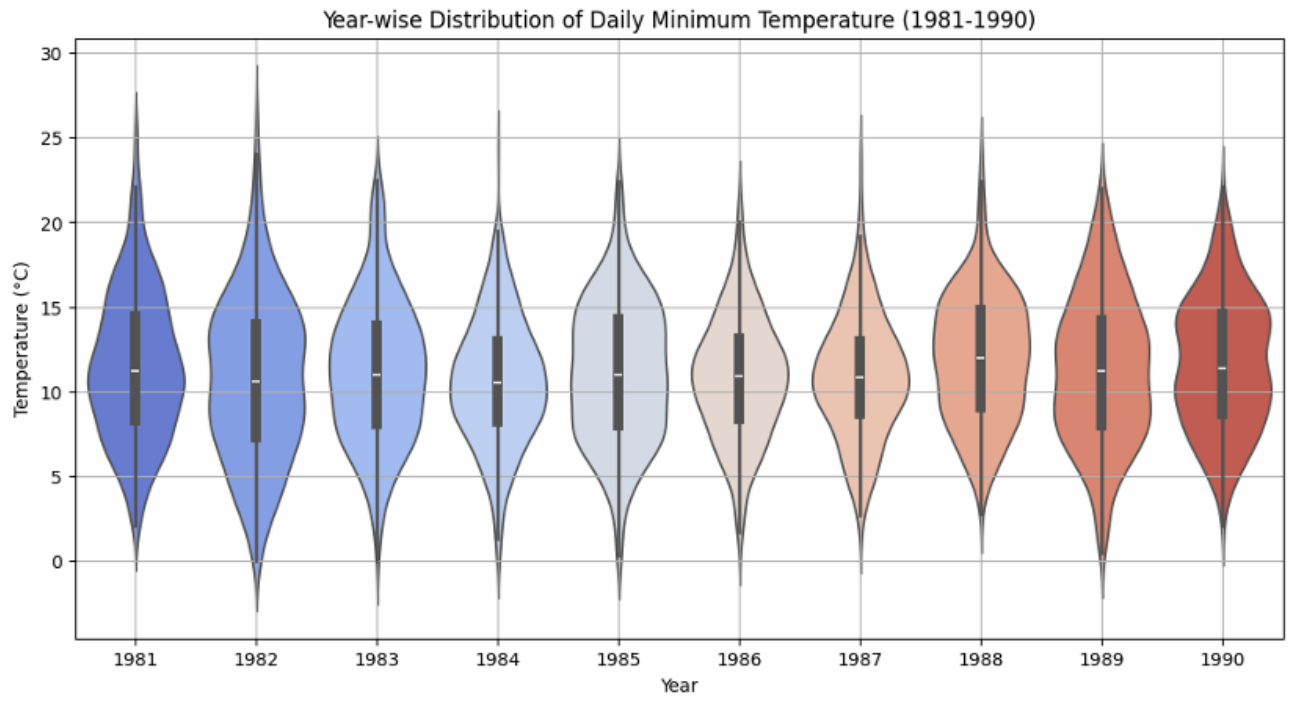
```
df['Quarter'] = df['Date'].dt.quarter
quarterly_avg = df.groupby('Quarter')['Temp'].mean()

plt.figure(figsize=(8, 5))
quarterly_avg.plot(kind='line', color='lightgreen')
plt.title('Average Quarterly Minimum Temperature (1981-1990)')
plt.xlabel('Quarter')
plt.ylabel('Average Temperature (°C)')
plt.xticks([0, 1, 2, 3], ['Q1', 'Q2', 'Q3', 'Q4'], rotation=0)

plt.grid(True)
plt.show()
```



- Visualize the minimum daily temperature distribution through violin plot year-wise from 1981 to 1990.

```
# 4. Visualize the minimum daily temperature distribution through violin plot year-wise
df['Year'] = df['Date'].dt.year
plt.figure(figsize=(12, 6))
sns.violinplot(x='Year', y='Temp', data=df, palette='coolwarm')
plt.title('Year-wise Distribution of Daily Minimum Temperature (1981-1990)')
plt.xlabel('Year')
plt.ylabel('Temperature (°C)')

plt.grid(True)
plt.show()
```

```
<ipython-input-8-ee3501b2d597>:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `lege

  sns.violinplot(x='Year', y='Temp', data=df, palette='coolwarm')
```

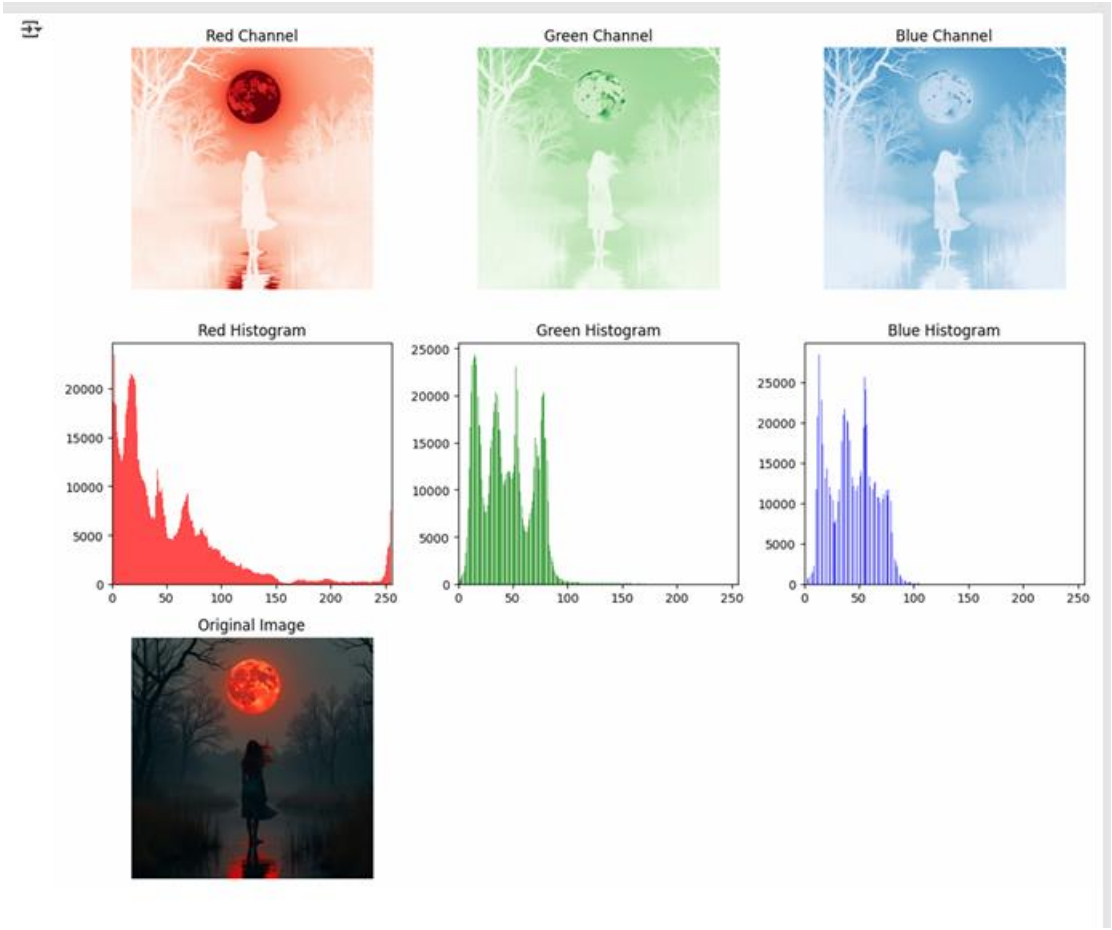


Q2. Do the following to visualize the image:

- Read a colour image and display it in the notebook file.
- Next, create a subplot of grid size 3x3. The first row will display the red, green, and blue channels of an image. The second row will display the histogram of each channel.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read a color image (replace 'your_image.jpg' with your image path)
image = cv2.imread('/content/flux.png')  # Reads in BGR format
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert to RGB for correct display
```

```
# Split into individual channels
red_channel = image[:, :, 0]
green_channel = image[:, :, 1]
blue_channel = image[:, :, 2]

# Create a 3x3 subplot grid
plt.figure(figsize=(12, 10))

# First row: Display RGB channels
plt.subplot(3, 3, 1)
plt.imshow(red_channel, cmap='Reds')
plt.title('Red Channel')
plt.axis('off')

plt.subplot(3, 3, 2)
plt.imshow(green_channel, cmap='Greens')
plt.title('Green Channel')
plt.axis('off')

plt.subplot(3, 3, 3)
plt.imshow(blue_channel, cmap='Blues')
plt.title('Blue Channel')
plt.axis('off')

# Second row: Histograms of each channel
plt.subplot(3, 3, 4)
plt.hist(red_channel.ravel(), bins=256, color='red', alpha=0.7)
plt.title('Red Histogram')
plt.xlim([0, 256])

plt.subplot(3, 3, 5)
plt.hist(green_channel.ravel(), bins=256, color='green', alpha=0.7)
plt.title('Green Histogram')
plt.xlim([0, 256])

plt.subplot(3, 3, 6)
plt.hist(blue_channel.ravel(), bins=256, color='blue', alpha=0.7)
plt.title('Blue Histogram')
plt.xlim([0, 256])

# Third row: Original image (optional)
plt.subplot(3, 3, 7)
plt.imshow(image)
plt.title('Original Image')
plt.axis('off')

plt.tight_layout()  # Adjust spacing between subplots
plt.show()
```



Q3. Display the map of India and highlight the region of Bihar state.

```
import geopandas as gpd
import matplotlib.pyplot as plt

states = gpd.read_file("/content/India-States.shp")

fig, ax = plt.subplots(figsize=(10 / 3 * 2, 12 / 3 * 2))
states.plot(ax=ax, color='lightgray', edgecolor='black')
states[states['ST_NM'].str.lower() == 'bihar'].plot(ax=ax, color='orange', edgecolor='black')
plt.title('Map of India Highlighting Bihar', fontsize=1)
ax.axis('off')

plt.show()
```