Data Visualization using Matplotlib Library

> **What is Data visualization?**
>
> - Representation of the data in a pictorial or graphical format
> - First step of data analysis
> - Allow us to get the intuitive understanding of the data
> - Helps to visualize the patterns in the data.
>
> **Types of Comparison**
>
> - **Univariate**: Univariate plots visualize a single variable at a time, showing the distribution or variation of that variable.
> - **Bivariate Plots**: Bivariate plots visualize the relationship between two variables, showing how they are related or correlated.
> - **Multivariate Plots**: Multivariate plots involve visualizing more than two variables simultaneously, providing insights into complex relationships and interactions among multiple variables in the dataset.
>
> **Introduction to Matplotlib**
>
> - It is a Python's 2D plotting library
> - 'pyplot' is a subpackage of matplotlib that provides a MATLAB-like way of plotting
> - Provides a simple way of plotting the various plots like histogram, bar plot, scatter plot" **Installation**
> - run ->**pip install matplotlib**<- in command prompt or Jupyter Notebook.

Import Libraries

In [2]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

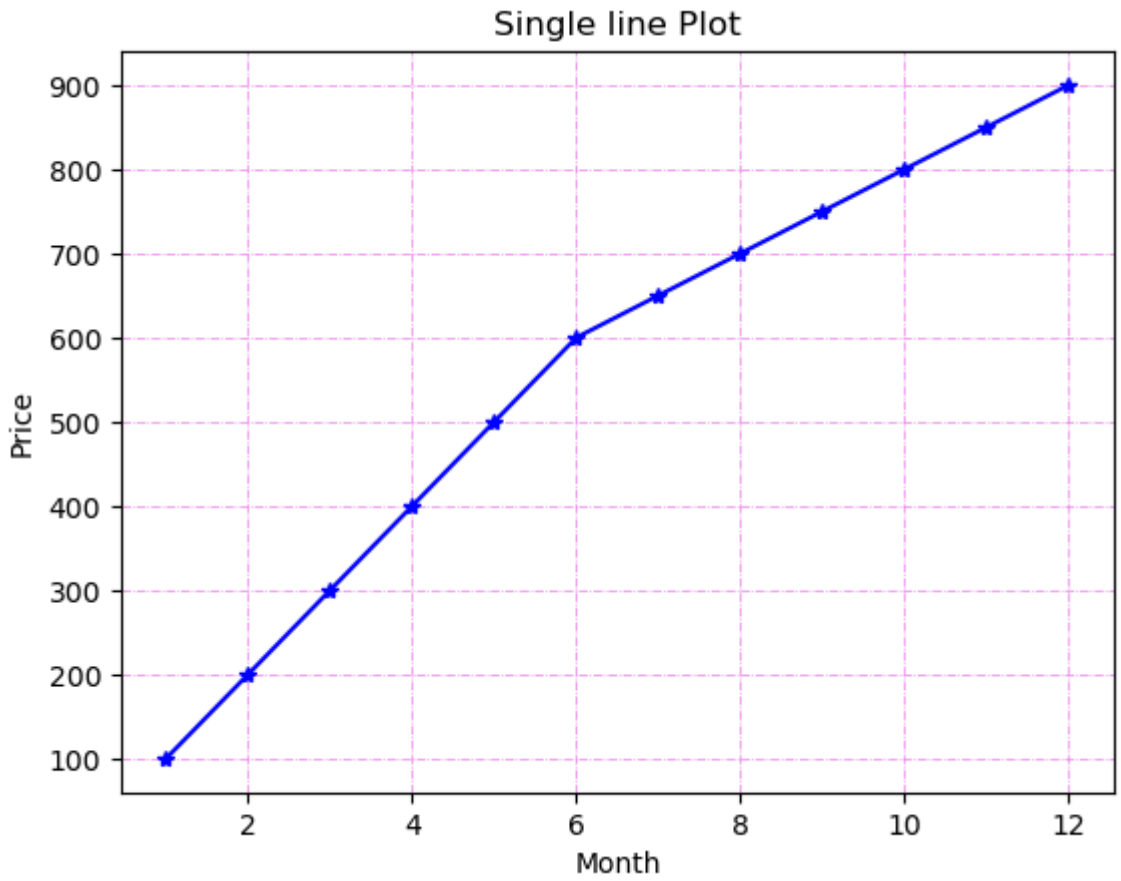
Single Line Plot

- It is a simple plot that displays the relationship between two variables and helps in finding the trend over a time period.

In [3]:
```
Month = np.arange(1,13) #1 - 12 months
```

In [5]:
```
Price = [100,200,300,400,500,600,650,700,750,800,850,900] # price for each month
```

In [15]:
```
#plot code
plt.plot(Month, Price, marker = '*', color = 'blue')

#customizing plot
plt.title('Single line Plot')
plt.xlabel('Month')
plt.ylabel('Price')
plt.grid(linestyle = '-.',linewidth=0.5, color='violet') #adding gridlines
plt.show()
```



Insights:

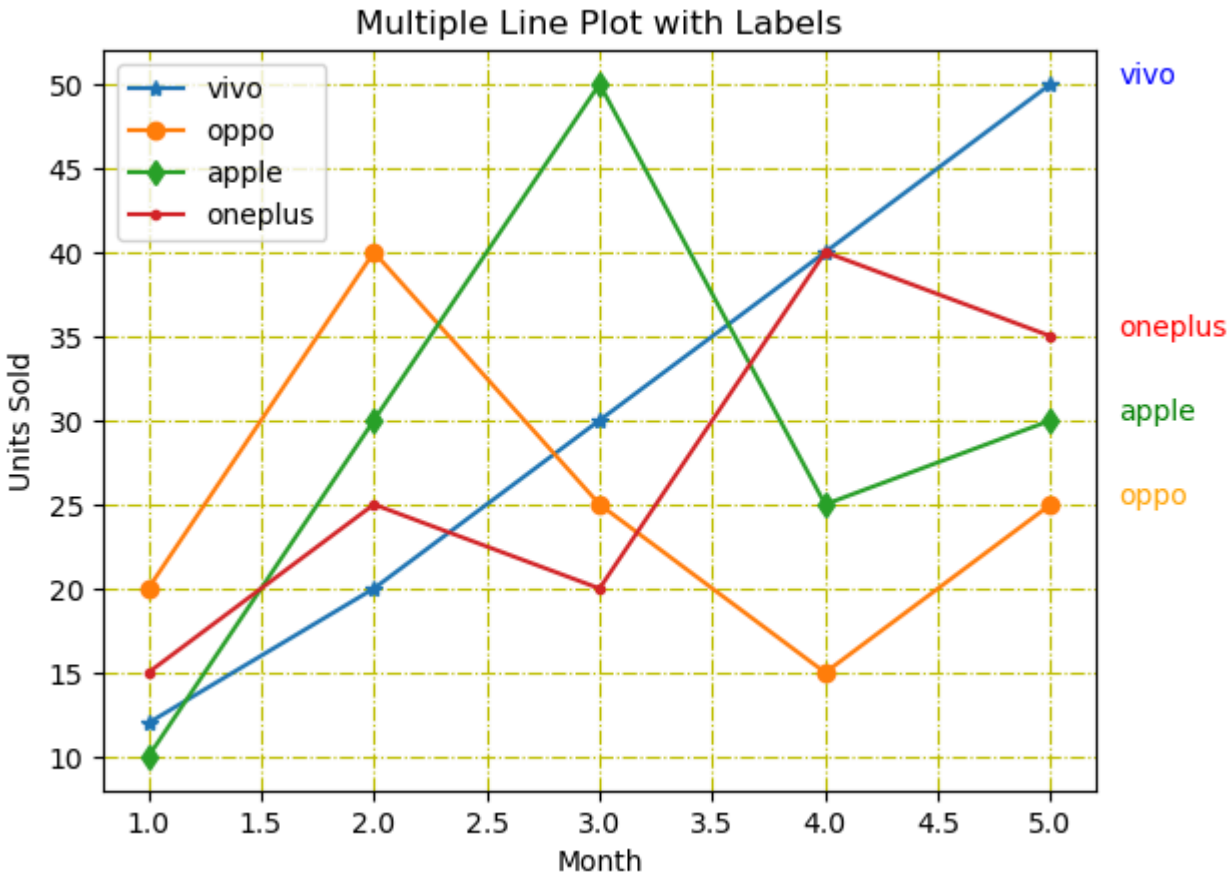
- As the month increases the price also increases.

Multiple Line Plot

- In matplotlib, a multiple line plot refers to a plot where multiple lines are displayed on the same graph.
- Each line represents a different dataset or variable, allowing for easy comparison between them.
- This type of plot is useful for visualizing trends, patterns, or relationships between multiple sets of data on a single graph.
- It helps in comparing the behavior of different variables over the same range of values.

In [17]:
```
# Data
month = [1, 2, 3, 4, 5] #months
vivo = [12, 20, 30, 40, 50]#units sold
oppo = [20, 40, 25, 15, 25]
apple = [10, 30, 50, 25, 30]
oneplus = [15, 25, 20, 40, 35]

# Plot the lines
plt.plot(month, vivo, label='vivo',marker = '*')
plt.plot(month, oppo, label='oppo',marker = 'o')
plt.plot(month, apple, label='apple',marker = 'd')
plt.plot(month, oneplus, label='oneplus',marker = '.')

# Add labels near the lines
plt.text(5.3, vivo[-1], 'vivo', color='blue', fontsize=10)
plt.text(5.3, oppo[-1], 'oppo', color='orange', fontsize=10)
plt.text(5.3, apple[-1], 'apple', color='green', fontsize=10)
plt.text(5.3, oneplus[-1], 'oneplus', color='red', fontsize=10)

# Customize the plot
plt.xlabel('Month')
```

```
plt.ylabel('Units Sold')
plt.title('Multiple Line Plot with Labels')

plt.legend()
plt.grid(linestyle = '-.',color = 'y')
# Show the plot
plt.show()
```



Insights:

- The sales of vivo branded phone is best selling phone.
- Apple has acheived the highest sales in 3rd month comparing with all 5 months.
- Oppo's sales has negative trend which shows decrease in sales over the period.

Scatter Plot

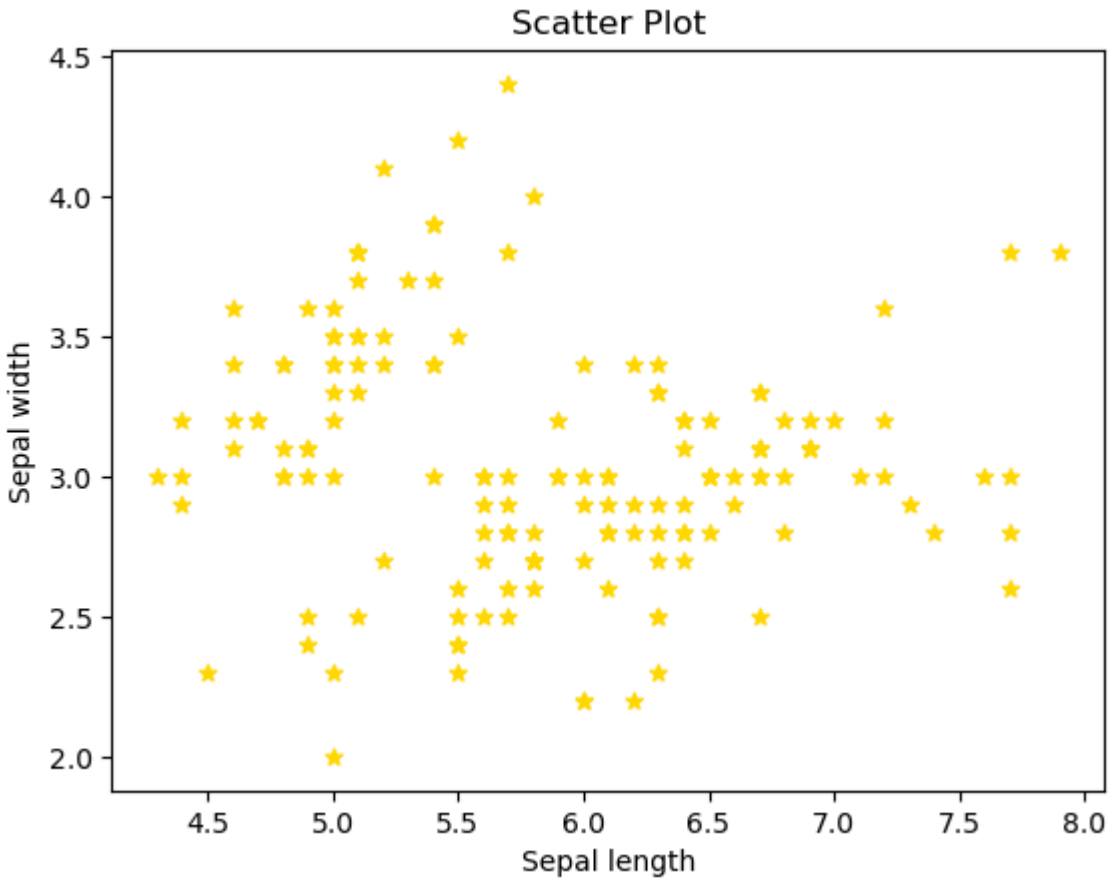
- A scatter plot in matplotlib is a type of plot that displays individual data points as markers on a two-dimensional plane.
- Each point on the plot represents the values of two variables, one plotted along the x-axis and the other along the y-axis.
- Scatter plots are useful for visualizing the relationship between two continuous variables and identifying patterns such as correlations, clusters, or outliers in the data.

In [18]:
```
import seaborn as sns #Here, importing seaborn for loading dataset.
df = sns.load_dataset('iris') #iris flower classification dataset
df.head() #display first five rows
```

Out[18]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [19]:
```
plt.scatter(x=df['sepal_length'],y = df['sepal_width'], marker = '*', color = 'gold')
plt.title('Scatter Plot')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.show()
```



Insights:

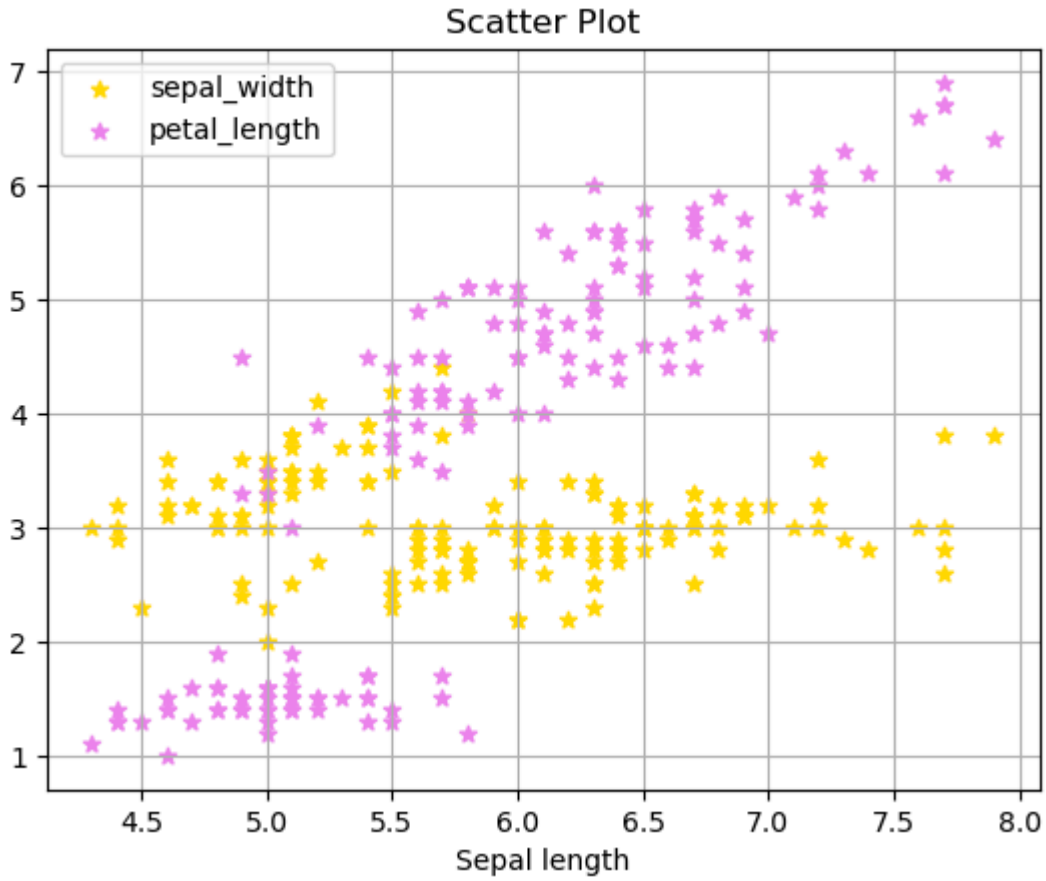
- There is no significant correlation between 'sepal length' and 'sepal width'.

Multiple Scatter Plot

- A Multiple Scatter Plot involves plotting multiple sets of data points on the same graph, where each set is represented by a different set of markers.
- This allows for the comparison of relationships between multiple pairs of variables.
- Each set of data points is visualized with distinct markers, colors, or sizes to differentiate between them, making it easier to analyze and compare the relationships between various pairs of variables simultaneously.

In [13]:
```
plt.scatter(x=df['sepal_length'],y = df['sepal_width'], marker = '*', color = 'gold', label =
plt.scatter(x=df['sepal_length'],y = df['petal_length'], marker = '*', color = 'violet', labe
plt.title('Scatter Plot')
```

```
plt.xlabel('Sepal length')
plt.grid()
plt.legend()
plt.show()
```



Insights:

- The above plot shows the positive relationship between sepal length and petal length.
- The sepal width and sepal length has no correlation.

What is Correlation?

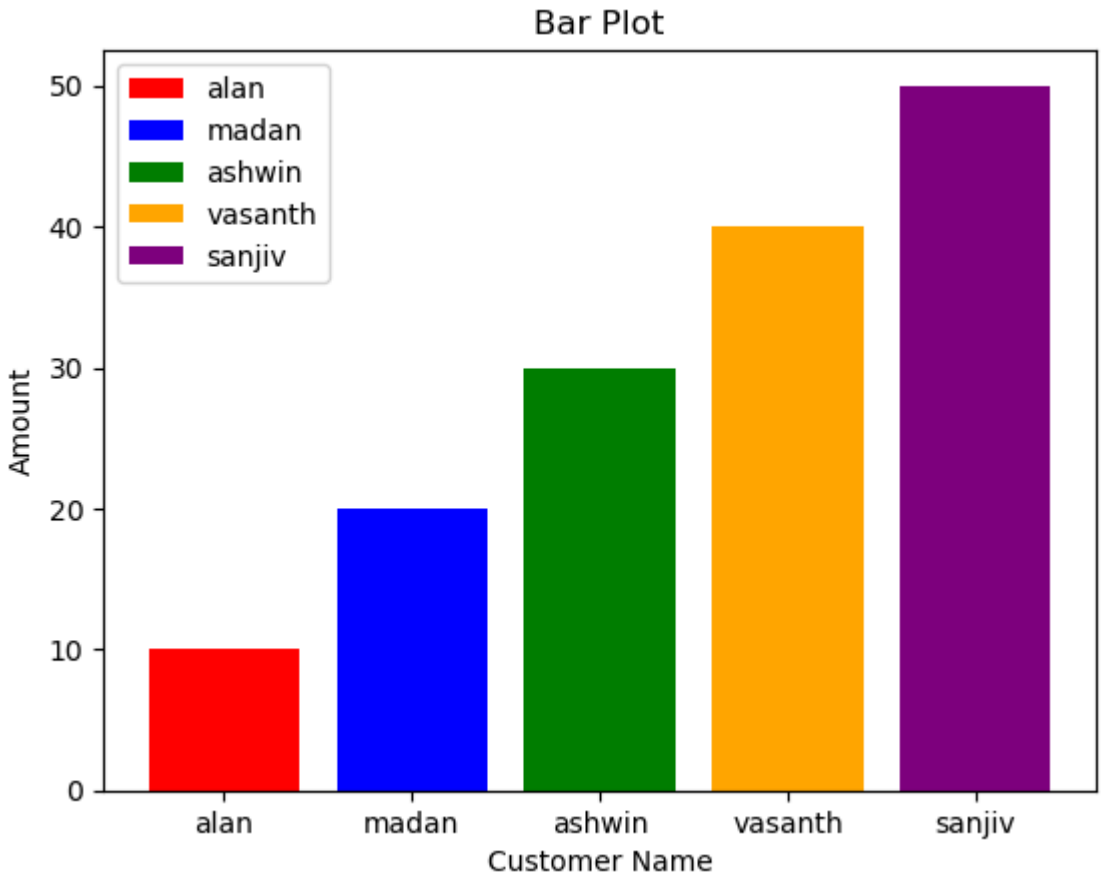
- The correlation between two variables can be determined by the pattern that the data points form on the plot.
- **Positive Correlation**: Data points tend to form a pattern that slopes upwards from left to right. As one variable increases, the other variable also tends to increase.
- **Negative Correlation**: Data points tend to form a pattern that slopes downwards from left to right. As one variable increases, the other variable tends to decrease.
- **No Correlation**: Data points appear randomly scattered across the plot with no clear pattern or trend. There is no apparent relationship between the variables.

Bar Plot

- A Bar plot in matplotlib is a type of chart that represents categorical data with rectangular bars.
- The length of each bar corresponds to the value of the data it represents.
- Each bar typically represents a category, and the height of the bar indicates the value associated with that category.

In [20]:
```
# Data
customer = ['alan', 'madan', 'ashwin', 'vasanth', 'sanjiv']
amount = [10,20,30,40,50]
colors = ['red', 'blue', 'green', 'orange', 'purple']
```

In [21]:
```
plt.bar(x = customer, height = amount, label = customer,color = colors)
plt.title('Bar Plot')
plt.xlabel('Customer Name')
plt.ylabel('Amount')
plt.legend()
plt.show()
```



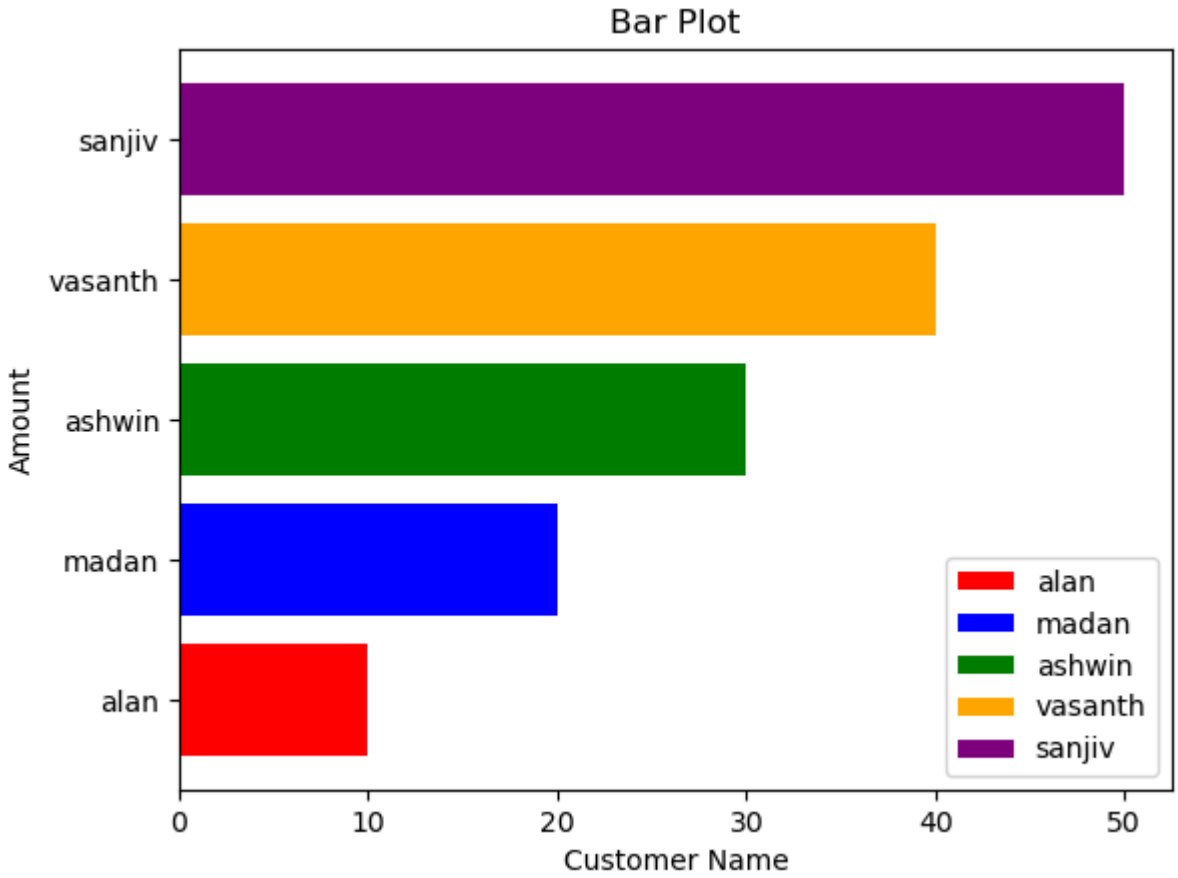
Insights:

- Sanjiv has paid the highest amount.
- Alan is the customer who paid less amount.

Horizontal Bar Plot

- A horizontal bar plot in matplotlib is similar to a standard bar plot but with the bars oriented horizontally instead of vertically.
- Horizontal bar plots are useful when you have long category names that may not fit well along the horizontal axis.

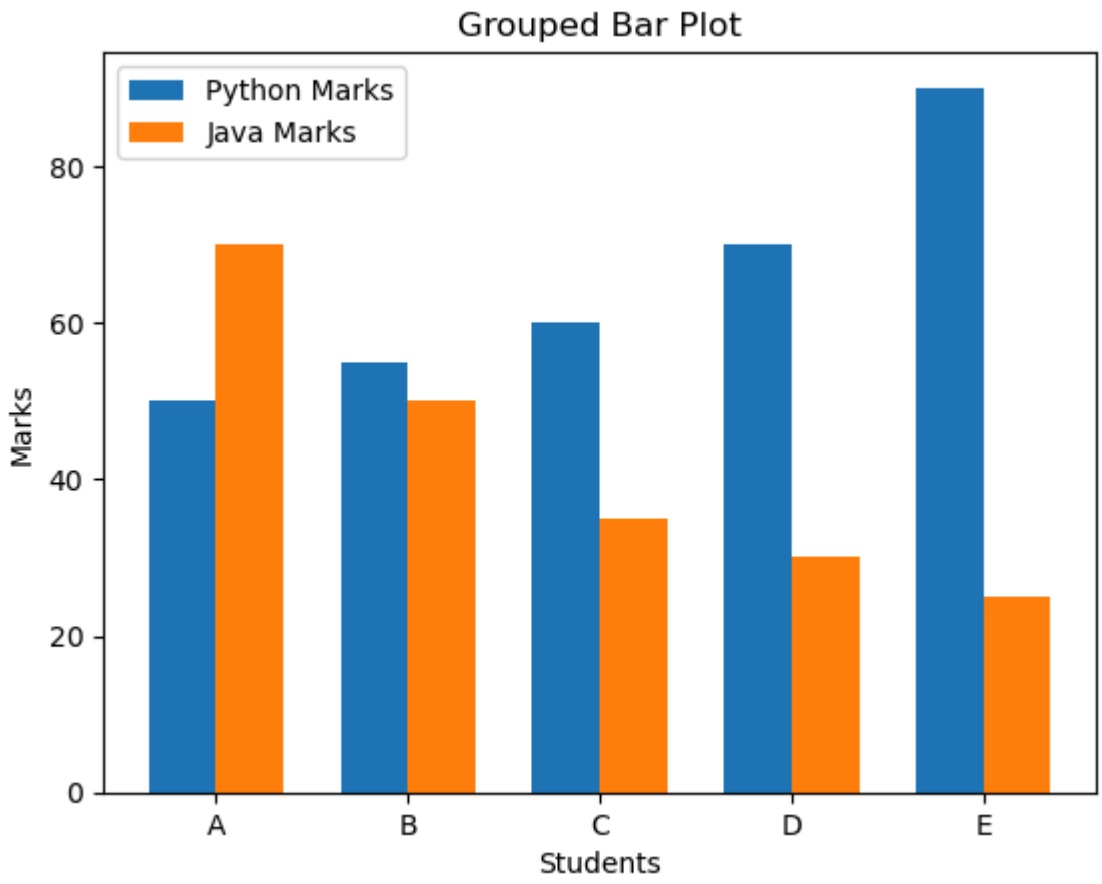
In [16]:
```
plt.barh(y = customer, width = amount, label = customer,color = colors)
plt.title('Bar Plot')
plt.xlabel('Customer Name')
plt.ylabel('Amount')
plt.legend()
plt.show()
```



Grouped Bar Plot

- Creating a grouped bar plot in matplotlib involves plotting bars for different groups of data side by side.
- You need to specify the categories and values for each group, then use the bar() function to plot the bars accordingly.
- Adjust the width and positioning of the bars to visually group them together.

In [22]:
```
python = [50,55,60,70,90]
java = [70,50,35,30,25]
X = np.arange(5)
plt.bar(x = X, height= python, width = 0.35, label = 'Python Marks')
plt.bar(x = X + 0.35, height= java, width = 0.35 , label = 'Java Marks')
plt.legend()
plt.xticks(ticks=X+0.35/2,labels = ('A','B','C','D','E'))
plt.title('Grouped Bar Plot')
plt.xlabel('Students')
plt.ylabel('Marks')
plt.show()
```



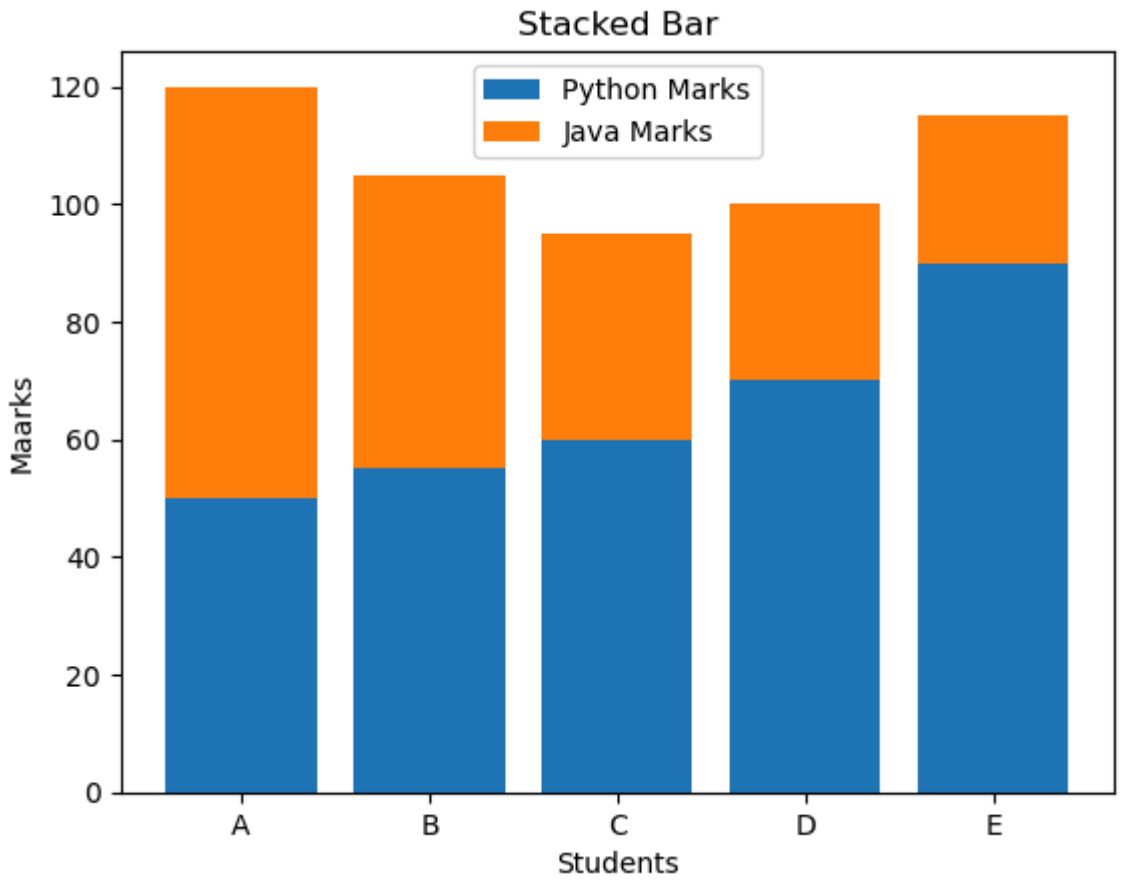
Insights:

- From Student A to E, the java marks are decreasing and the python marks are increasing.
- E scored the highest scores in python and least score in java.

Stacked Bar Plot

- A stacked bar plot in matplotlib is a type of chart where multiple sets of data are stacked on top of each other within each category on the x-axis.
- Each bar in the plot represents a category, and the height of the bar is divided into segments, each corresponding to a different dataset.

In [23]:
```
python = [50,55,60,70,90]
java = [70,50,35,30,25]
X = np.arange(5)
plt.bar(x = X, height= python, label = 'Python Marks')
plt.bar(x = X, height= java, bottom = python, label = 'Java Marks')
plt.xticks(ticks = X, labels = ('A','B','C','D','E'))
plt.title('Stacked Bar')
plt.xlabel('Students')
plt.ylabel('Maarks')
plt.legend()
plt.show()
```



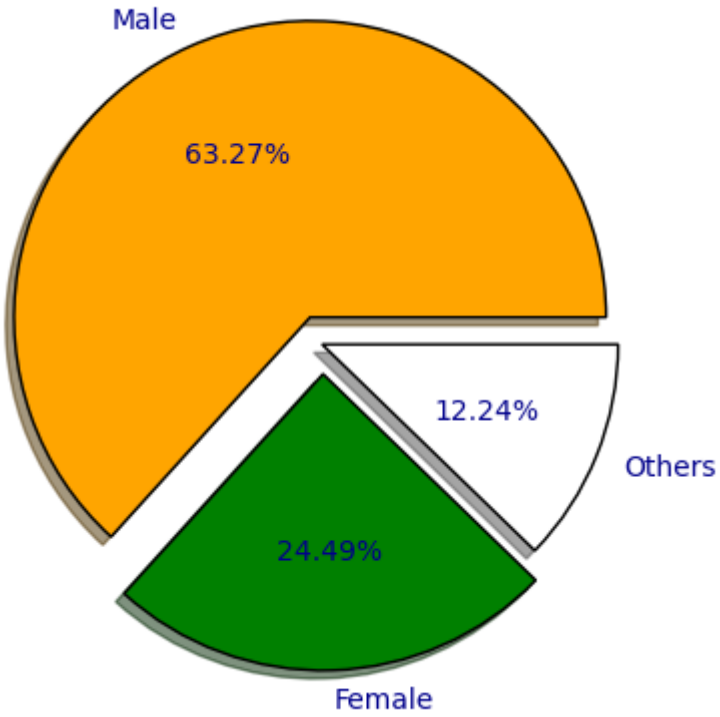
Insights:

- We can clearly see that that the students performed well in python and performed poor in java.

Pie Plot

- A pie plot in matplotlib is a circular statistical graphic that is divided into slices to illustrate numerical proportions.
- It is used to display the univariate data.
- Each section of the pie plot represents a single category in the data

In [19]:
```
gender = ['Male','Female','Others']
population = [2584,1000,500]
plt.pie(x = population, labels = gender, autopct = '%.2f%%' ,colors=['orange','green','white'
        shadow=True,explode = [0.1,0.1,0],
        textprops={'color': 'darkblue'},wedgeprops={'edgecolor': 'black'})
plt.show()
```



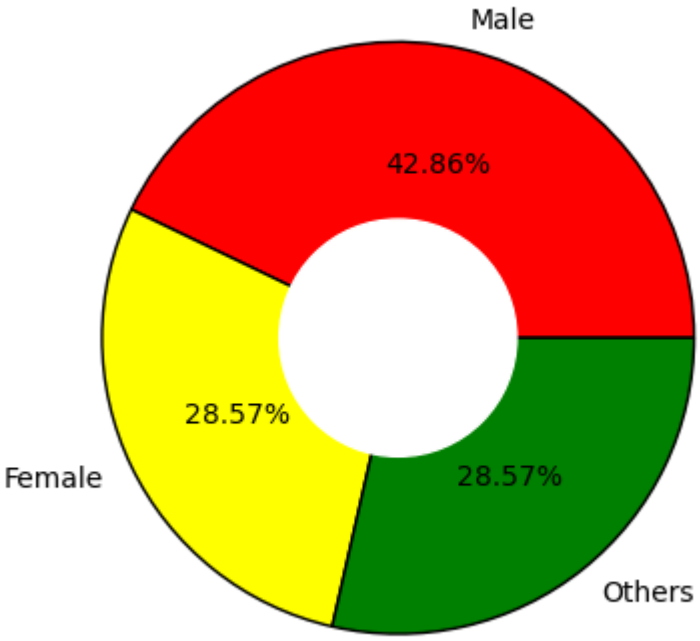
Insights:

- The male is the majority gender in the data.
- The female holds the second place and others holds the last place.

Donut Pie

- A donut plot, also known as a donut chart, is a variation of a pie chart where the center of the chart is removed, creating a "hole" in the middle. The outer ring of the donut chart represents the whole dataset, similar to a pie chart, but the center is empty.

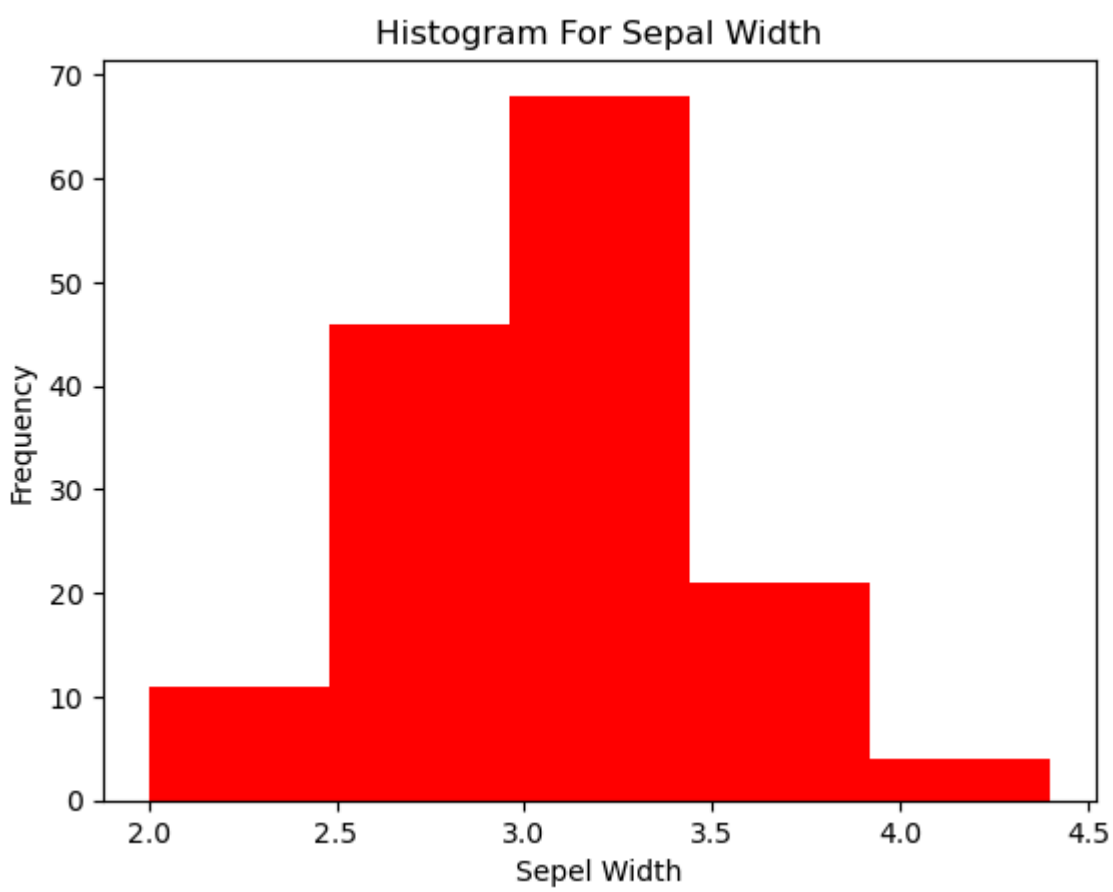
In [27]:
```
gender = ['Male','Female','Others']
population = [1500,1000,1000]
plt.pie(x = population, labels = gender, autopct = '%.2f%%' ,colors=['red','yellow','green'],
        textprops={'color': 'black'},wedgeprops={'edgecolor': 'black'})
circle = plt.Circle(xy = (0,0),radius=.4,color = 'white')
plt.gcf()
plt.gca().add_artist(circle)
plt.show()
```



Histogram

- A histogram is a type of plot that is used to represent the distribution of a numeric variable.
- It provides an estimate of the probability distribution of continuous data by dividing the data into intervals or bins and displaying the frequency of observations in each bin with bars.
- One axis of the histogram represents the variable in the form of bars, while the other axis represents the frequency of each bar.
- There are no gaps between the bars of the histogram.

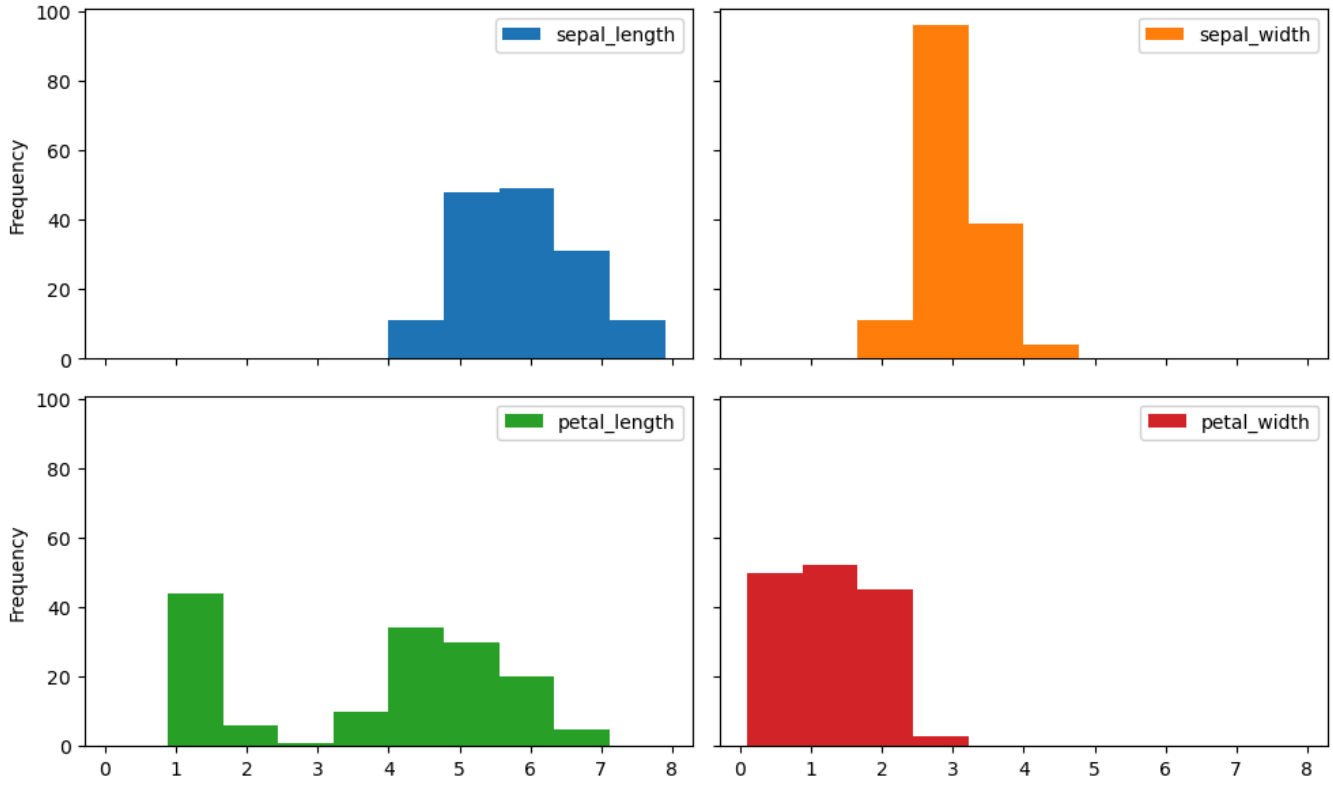
In [28]:
```
plt.hist(x = df['sepal_width'],color = 'red', bins = 5) # Bins is used to return the specifie
plt.title('Histogram For Sepal Width')
plt.xlabel('Sepel Width')
plt.ylabel('Frequency')
plt.show()
```



Multiple Histogram

- Displaying Histogram Plot of various variables in the same graph.

In [29]:
```
df.plot.hist(subplots = True, layout = (2,2), figsize = (10,6), sharex = True, sharey = True)
plt.tight_layout()
plt.show()
```



Box Plot

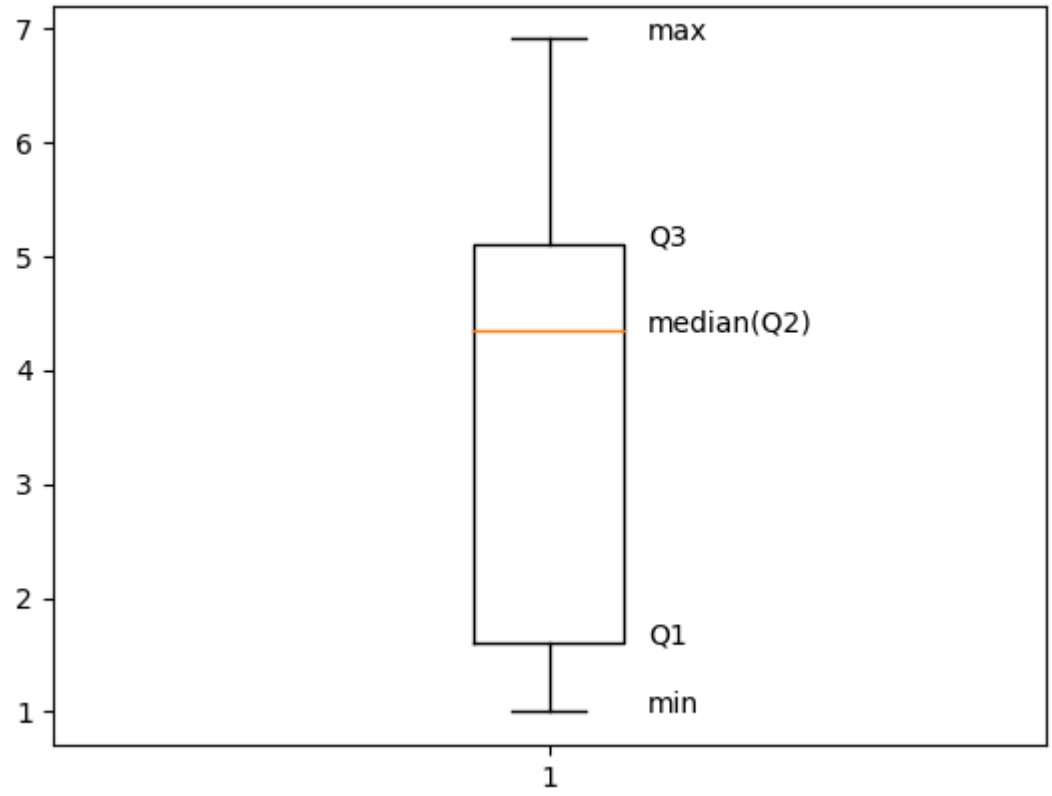
- A box plot, also known as a box-and-whisker plot, is a graphical representation of the distribution of a dataset based on five summary statistics: minimum, first quartile (Q1), median (Q2), third quartile (Q3), and maximum.

In a box plot:

- The box represents the interquartile range (IQR) between the first and third quartiles (Q1 and Q3).
- The line inside the box represents the median (Q2).
- The "whiskers" extend from the box to the minimum and maximum values within a certain range.
- Outliers are often identified as individual points beyond the whiskers.

Box plots are useful for visually summarizing the distribution of data, identifying outliers, and comparing the spread and central tendency of different datasets.

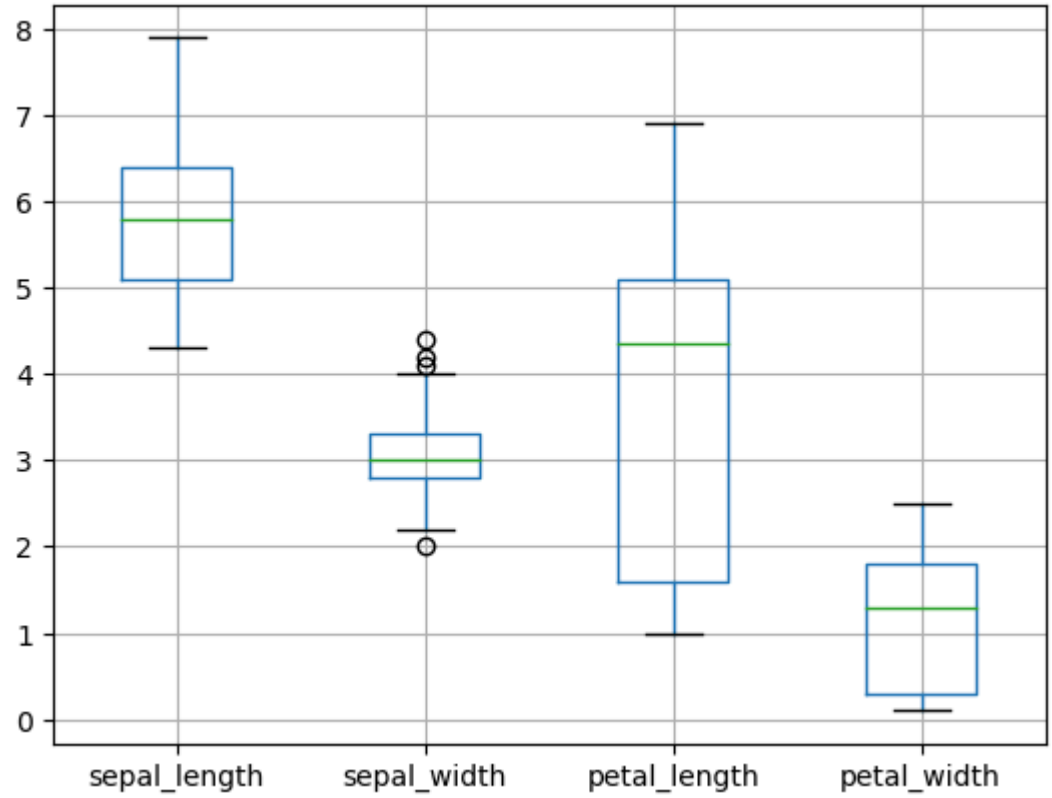
In [26]:
```
plt.boxplot(x = df['petal_length'])
plt.text(x = 1.1, y = df['petal_length'].min(), s = 'min')
plt.text(x = 1.1, y = df['petal_length'].quantile(0.25), s = 'Q1')
plt.text(x = 1.1, y = df['petal_length'].median(), s = 'median(Q2)')
plt.text(x = 1.1, y = df['petal_length'].quantile(0.75), s = 'Q3')
plt.text(x = 1.1, y = df['petal_length'].max(), s = 'max')
plt.show()
```



Example:-

In [30]:
```
df.boxplot()
```

Out[30]:
```
<Axes: >
```



Insights:

- The sepal width has outliers.
- The median sepal width is 3.
- The sepal length has the highest length range.
- The petal width has the lowest range.

Area Plot

An area plot, also known as a filled line plot, is a type of plot where the area under the line is filled with color to represent the cumulative effect of the values being plotted.

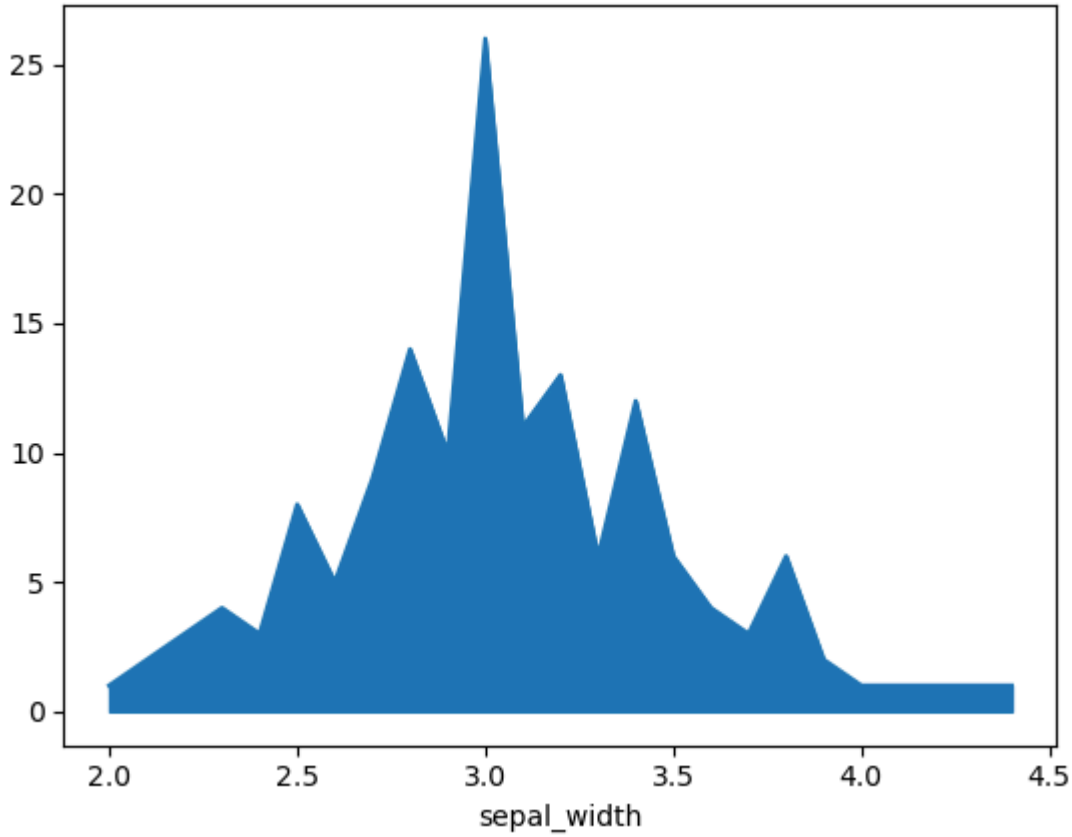
In an area plot:

- The x-axis typically represents time or another continuous variable.
- The y-axis represents the value of the variable being plotted.
- The area under the curve is filled with color to emphasize the magnitude or cumulative effect of the data.

Area plots are useful for visualizing the trend and cumulative effect of data over time or across different categories. They are effective in highlighting the overall pattern and variations in the data series.

In [28]:
```
df['sepal_width'].value_counts().sort_index().plot.area()
```

Out[28]:
```
<Axes: xlabel='sepal_width'>
```



Insights:

- The highest sepal_width frequency is 3 i.e, the median sepal_width is 3.

In this Matplotlib Jupyter Notebook documentation, we have explored a variety of plots and visualizations to analyze and interpret data effectively. From single line plots to histograms and box plots, Matplotlib has allowed us to represent data in insightful ways, uncovering patterns, trends, and relationships within the datasets.