Experiment 01

1. What is data visualization? List and discuss the various tools for data visualization. Explain the various libraries available in python for data visualization. Write the syntax and describe the parameters used for the following:
   - Box Plot
   - Scatter Plot
   - Histogram
   - Pie Chart
   - Facet Plot
   - Pair Plot
   - Area Chart
   - Violin Plot
   - Bar Chart

Aim :

To write the syntax and coding for various plots using for data visualization.

What is Data Visualization?

Graphical representation of any information or data is known as Data Visualization. This helps in segregating the data in an efficient manner by using various types of visuals such as graphs, maps, charts, maps, and visualization tools. In addition to this, with the help of a data visualization tool, the data can be presented in a very unique and understandable manner so that people who are not from a technical background can understand everything easily.

What are Data Visualization Tools?

Data Visualization Tools are software platforms that provide information in a visual format such as a graph, chart, etc to make it easily understandable and usable. Data Visualization tools are so popular as they allow analysts and statisticians to create visual data models easily according to their specifications by conveniently providing an interface, database connections, and Machine Learning tools all in one place!

Best Data Visualization Tools

This article demonstrates the **12 most popular Data Visualization tools** that are commonly used the

1. Tableau

Tableau is a data visualization tool that can be used to create interactive graphs, charts, and maps. It allows you to connect to different data sources and **create visualizations in minutes.** You can also share your work with others and collaborate on projects. Tableau Desktop is the original product. It’s made for creating static visualizations that can be published on one or more web pages, but it doesn’t create interactive maps. Tableau Public is the free distribution of the Desktop product with some limitations. As a data scientist, Tableau has to be the number one tool for you to learn and use in your everyday tasks.

2. Looker

Looker is a data visualization tool that can go in-depth into the data and analyze it to obtain useful insights. It provides real-time dashboards of the data for more in-depth analysis so that businesses can make instant decisions based on the data visualizations obtained. Looker also provides connections with Redshift, Snowflake, and BigQuery, as well as more than 50 SQL-supported dialects so you can connect to multiple databases without any issues.

Looker data visualizations can be shared with anyone using any particular tool. Also, you can export these files in any format immediately. It also provides customer support wherein you can ask any question and it shall be answered. A price quote can be obtained by submitting a form.

3. Zoho Analytics

Zoho Analytics is a Business Intelligence and Data Analytics software that can help you create wonderful-looking data visualizations based on your data in a few minutes. You can obtain data from multiple sources and mesh it together to create multidimensional data visualizations that allow you to view your business data across departments. In case you have any questions, you can use Zia which is a smart assistant created using artificial intelligence, machine learning, and natural language processing.

4. Sisense

Sisense is a business intelligence-based data visualization system and it provides various tools that allow data analysts to simplify complex data and obtain insights for their organization and outsiders. Sisense believes that eventually, every company will be a data-driven company and every product will be related to data in some way. Therefore it tries its best to provide various data analytics tools to business teams and data analytics so that they can help make their companies the data-driven companies of the future.

It is very easy to set up and learn Sisense. It can be easily installed within a minute and data analysts can get their work done and obtain results instantly. Sisense also allows its users to export their files in multiple formats such as PPT, Excel, MS Word, PDF, etc. Sisense also provides full-time customer support services whenever users face any issues. A price quote can be obtained by submitting a form.

5. IBM Cognos Analytics

IBM Cognos Analytics is an Artificial Intelligence-based business intelligence platform that supports data analytics among other things. You can visualize as well as analyze your data and share actionable insights with anyone in your organization. Even if you have limited or no knowledge about data analytics, you can use IBM Cognos Analytics easily as it interprets the data for you and presents you with actionable insights in plain language.

You can also share your data with multiple users if you want on the cloud and share visuals over email or Slack. You can also import data from various sources like spreadsheets, cloud, CSV files, or on-premises databases and combine related data sources into a single data module.

6. **Google Charts**

One of the major players in the data visualization market space, Google Charts, coded with SVG and HTML5, is famed for its capability to produce graphical and pictorial data visualizations. Google Charts offers zoom functionality, and it provides users with unmatched cross-platform compatibility with iOS, Android, and even the earlier versions of the Internet Explorer browser. It provides:

- User-friendly platform
- Easy to integrate data
- Visually attractive data graphs
- Compatibility with Google products.

7. **Excel**

Microsoft Excel is a data visualization tool that has an easy interface, so it doesn’t have to be difficult to work with. There are many different ways of visualizing data in Excel. One of them is by using scatter plots, Scatter plots display the relationship between two datasets that you would want to compare. You can also see how different variables are related to one another in order to determine if they’re connected or not. Scatter plots are used to analyze statistical, scientific, medical, and economic data for purposes such as market research or financial planning

8. Microsoft Power BI

Microsoft Power BI is a Data Visualization platform focused on creating a data-driven business intelligence culture in all companies today. To fulfill this, it offers self-service analytics tools that can be used to analyze, aggregate, and share data in a meaningful fashion.

Microsoft Power BI offers hundreds of data visualizations to its customers along with built-in Artificial Intelligence capabilities and Excel integration facilities.

9.**QlikView**
QlikView is not just another data visualization tool, It is a data discovery platform that empowers the users to make faster, more informed decisions by accelerating analytics, revealing new business insights, and increasing the accuracy of results. It has been an intuitive software development kit that has been used in organizations around the world for many years. It can combine various kinds of data sources with visualizations in color-coded tables, bar charts, line graphs, pie charts, and sliders. It has been developed on a "drag and drops" visualization interface, allowing users to easily add data from many different sources, such as databases or spreadsheets, without having to write any code. These characteristics also make it a relatively easier tool to learn and grasp.

10. SAP Analytics Cloud

SAP Analytics Cloud uses business intelligence and data analytics capabilities to help you evaluate your data and create visualizations in order to predict business outcomes. It also provides you with the latest modeling tools that help you by alerting you of possible errors in the data and categorizing different data measures and dimensions. SAP Analytics Cloud also suggests Smart Transformations to the data that lead to enhanced visualizations.

In case you have any doubts or business questions related to data visualization, SAP Analytics Cloud provides you with complete customer satisfaction by handling your queries using conversational artificial intelligence and natural language technology.

11. Yellowfin

Yellowfin is a worldwide famous analytics and business software vendor that has a well-suited automation product that is specially created for people who have to take decisions within a short period of time. This is an easy-to-use data visualization tool that allows people to understand things and act according to them in the form of collaboration, data storytelling, and stunning action-based dashboards.

12. Whatagraph

Whatagraph is a seamless integration that provides marketing agencies with an easy and useful way of sharing or sending marketing campaign data with clients. With this platform, you can create the data in a way that the result is easy to understand and comprehend. This Data visualization tool has numerous customization options which can be picked virtually and help in creating reporting widgets or creating your own methods of presenting data.

Explain the various libraries available in python for data visualization.

Top 10 Python Libraries for Data Visualization that are commonly used these days.

Matplotlib

Matplotlib is a data visualization library and 2-D plotting library of Python It was initially released in 2003 and it is the most popular and widely-used plotting library in the Python community. It comes with an interactive environment across multiple platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, etc. It can be used to embed plots into applications using various GUI toolkits like Tkinter, GTK+, wxPython, Qt, etc. So you can use Matplotlib to create plots, bar charts, pie charts, histograms, scatterplots, error charts, power spectra, stemplots.

Seaborn

Seaborn is a Python data visualization library that is based on Matplotlib and closely integrated with the NumPy and pandas data structures. Seaborn has various dataset-oriented plotting functions that operate on data frames and arrays that have whole datasets within them. Then it internally performs the necessary statistical aggregation and mapping functions to create informative plots that the user desires. It is a high-level interface for creating beautiful and informative statistical graphics that are integral to exploring and understanding data. The Seaborn data graphics can include bar charts, pie charts, histograms, scatterplots, error charts, etc. Seaborn also has various tools for choosing color palettes that can reveal patterns in the data.

Geoplotlib

It supports the creation of geographical maps in particular with many different types of maps available such as dot-density maps, choropleths, symbol maps, etc. One thing to keep in mind is that requires NumPy and pyglet as prerequisites before installation but that is not a big disadvantage.

Plotly

Plotly is a free open-source graphing library that can be used to form data visualizations. Plotly (plotly.py) is built on top of the Plotly JavaScript library (plotly.js) and can be used to create web-based data visualizations that can be displayed in Jupyter notebooks or web applications using Dash or saved as individual HTML files. Plotly provides more than 40 unique chart types like scatter plots, histograms, line charts, bar charts, pie charts, error bars, box plots, multiple axes, sparklines, dendrograms, 3-D charts, etc. Plotly also provides contour plots, which are not that common in other data visualization libraries. In addition to all this, Plotly can be used offline with no internet connection.

GGplot

Ggplot is a Python data visualization library that is based on the implementation of ggplot2 which is created for the programming language R. Ggplot can create data visualizations such as bar charts, pie charts, histograms, scatterplots, error charts, etc. using high-level API. It also allows you to add different types of data visualization components or layers in a single visualization. Once ggplot has been told which variables to map to which aesthetics in the plot, it does the rest of the work so that the user can focus on interpreting the visualizations and take less time in creating them.

Altair

Altair is a statistical data visualization library in Python. It is based on Vega and Vega-Lite which are a sort of declarative language for creating, saving, and sharing data visualization designs that are also interactive. Altair can be used to create beautiful data visualizations of plots such as bar charts, pie charts, histograms, scatterplots, error charts, power spectra, stemplots, etc. using a minimal amount of coding. Altair has dependencies which include python 3.6, entrypoints, jsonschema, NumPy, Pandas, and Toolz which are automatically installed with the Altair installation commands.

Bokeh

Bokeh is a data visualization library that provides detailed graphics with a high level of interactivity across various datasets, whether they are large or small. Bokeh is based on The Grammar of Graphics like ggplot but it is native to Python while ggplot is based on ggplot2 from R. Data visualization experts can create various interactive plots for modern web browsers using bokeh which can be used in interactive web applications, HTML documents, or JSON objects. Bokeh has 3 levels that can be used for creating visualizations. The first level focuses only on creating the data plots quickly, the second level controls the basic building blocks of the plot while the third level provides full autonomy for creating the charts with no pre-set defaults. This level is suited to the data analysts and IT professionals that are well versed in the technical side of creating data visualizations.

Pygal

Pygal is a Python data visualization library. While Pygal is similar to Plotly or Bokeh in that it creates data visualization charts that can be embedded into web pages and accessed using a web browser, a primary difference is that it can output charts in the form of SVG's or Scalable Vector Graphics. These SVG's ensure that you can observe your charts clearly without losing any of the quality even if you scale them. However, SVG's are only useful with smaller datasets as too many data points are difficult to render and the charts can become sluggish.

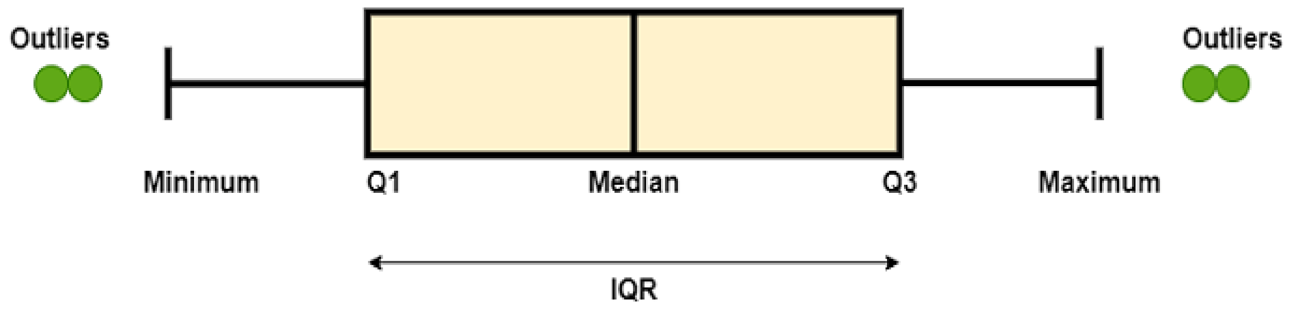
A BOX Plot:

A Box plot is a way to visualize the distribution of the data by using a box and some vertical lines. It is known as the whisker plot. The data can be distributed between five key ranges, which are as follows:

1. **Minimum**: Q1-1.5*IQR
2. **1st quartile** (Q1): 25th percentile
3. **Median**:50th percentile
4. **3rd quartile**(Q3):75th percentile
5. **Maximum**: Q3+1.5*IQR

Here IQR represents the **Inter Quartile Range** which starts from the first quartile (Q1) and ends at the third quartile (Q3).

Box Plot visualization



In the box plot, those points which are out of range are called outliers. We can create the box plot of the data to determine the following:

- The number of outliers in a dataset
- Is the data skewed or not
- The range of the data

The range of the data from minimum to maximum is called the whisker limit. In Python, we will use the matplotlib module's pyplot module, which has an inbuilt function named boxplot() which can create the box plot of any data set.

Syntax:

matplotlib.pyplot.boxplot(data,notch=none,vert=none,patch_artist,widths=none)

In the boxplot() function, we have a lot of attributes which can be used to create a more attractive and amazing box plot of the data set.

Parameters:

Data	The data should be an array or sequence of arrays which will be plotted.
Notch	This parameter accepts only Boolean values, either true or false.
Vert	This attribute accepts a Boolean value. If it is set to true, then the graph will be vertical. Otherwise, it will be horizontal.
Position	It accepts the array of integers which defines the position of the box.
Widths	It accepts the array of integers which defines the width of the box.
Patch_Artist	This parameter accepts Boolean values, either true or false, and this is an optional parameter.
Labels	This accepts the strings which define the labels for each data point
Meanline	It accepts a boolean value, and it is optional.
Order	It sets the order of the boxplot.
Bootstrap	It accepts the integer

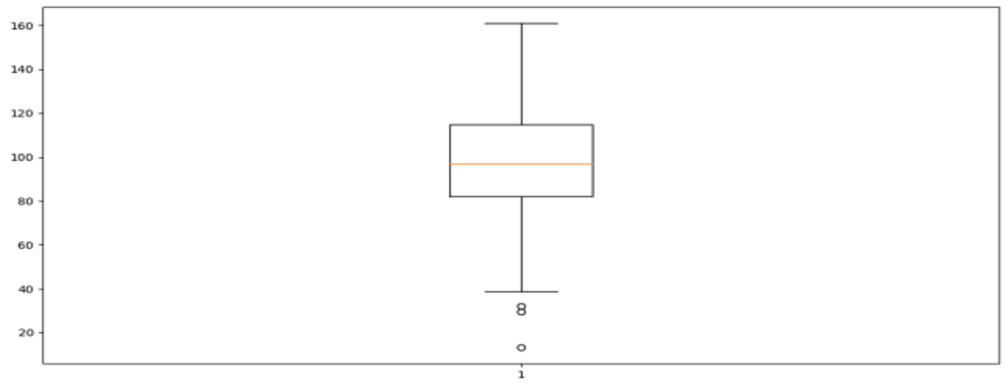
value, which specifies the range of the notched boxplot.

Example1:

We will create the random data set of the numpy array and create the box plot.

```
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(15)
dataSet = np.random.normal(100, 25, 200)
print(dataSet)
figure = plt.figure(figsize =(10, 8))
plt.boxplot(dataSet)
plt.show()
```

Output:



By using csv file:

```
import pandas as pd
# load the dataset
df = pd.read_csv("D:\iris.csv")
# display 5 rows of dataset
print(df.head())
```

Output:-

```
  sepal  sepal.width  petal.length  petal.width variety
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

Process finished with exit code 0
```

Scatter() plot pandas in Python

A data visualization method that displays the relationship between two numerical variables is called a scatter plot. In Python, there is a class named *DataFrame* that can be used to plot to scatter plots using pandas, and this class's member is called plot. By using the scatter() function on the plot component, a pandas DataFrame plot is drawn between two variables or two columns.

Utilizing a Pandas Data Frame for a scatter plot

The *plot* is a member of the Python pandas Data Frame class.

Drawing a scatter plot between two specified columns of a pandas Data Frame requires calling the scatter() function on the plot component.

Multiple columns are possible in a pandas Data Frame.

The scatter() method's X and Y inputs can be any two columns.

Syntax

Here, the syntax shows how to write code for the scatter() method:

DataFrame.plot.scatter(x_axis, y_axis, s = none, c = none)

The following are the syntax parameters for the scatter() method:

X_axis value - An array holding the scatter plot's x-axis data.
Y_ axis value - a data array for the y-axis.
S stands for the marker's size (can be scalar or array of size equal to the size of x or y)
c- the order of the markers' colors
Marker -Marker style
Cmap – Cmap name
Line Width – Width of marker border
Edgecolor – Marker border color
Alpha – Blending value ,between 0 & 1

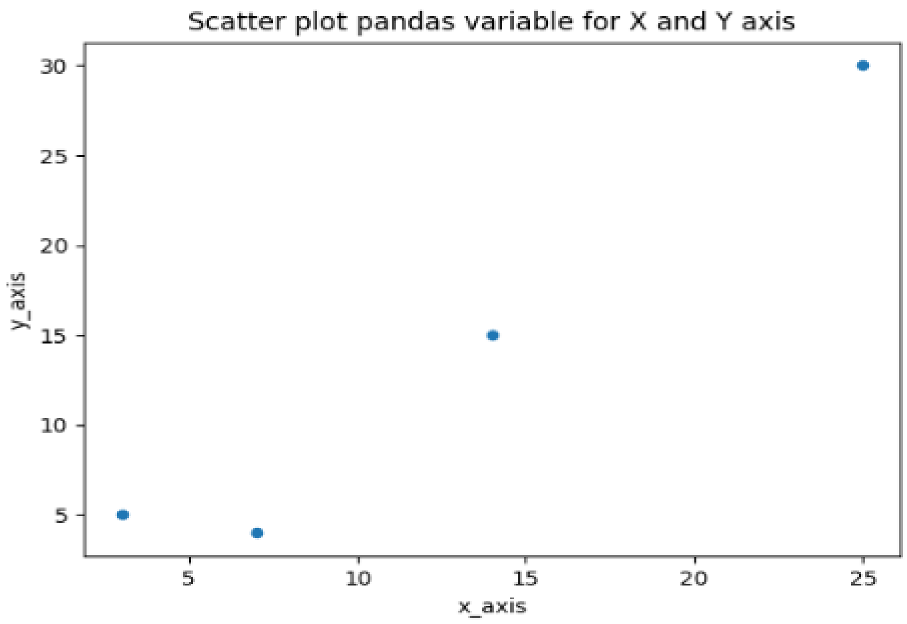
Example

the following example shows a scatter plot using the pandas. We can see the plot values in the x and y-axis position with essential tuples.

```
# Example to draw a scatter plot in pandas
# use two columns to create a scatter plot pandas DataFrame
import pandas as pd
import matplotlib.pyplot as plot
# List of values
data_value = [(3, 5),
        (25, 30),
       (7, 4),
        (14, 15)]
# Load input data in the pandas DataFrame
dataFrame = pd.DataFrame(data = data_value, columns = ['x_axis', 'y_axis']);
# Draw a scatter plot using pandas
dataFrame.plot.scatter(x = 'x_axis', y = 'y_axis', title = "Scatter plot pandas variable for X and
 Y axis");
```

```
plot.show();
```

Output:



Histogram:

A histogram is a graphical representation of the distribution of a continuous dataset. It divides the data into intervals (bins) and shows the frequency of data points falling into each bin. Histograms are useful for visualizing the shape and spread of the data and identifying any patterns or outliers.

Syntax: matplotlib.pyplot.hist(x)

Below is a general description of the parameters commonly used in histogram functions across various programming languages and plotting libraries:

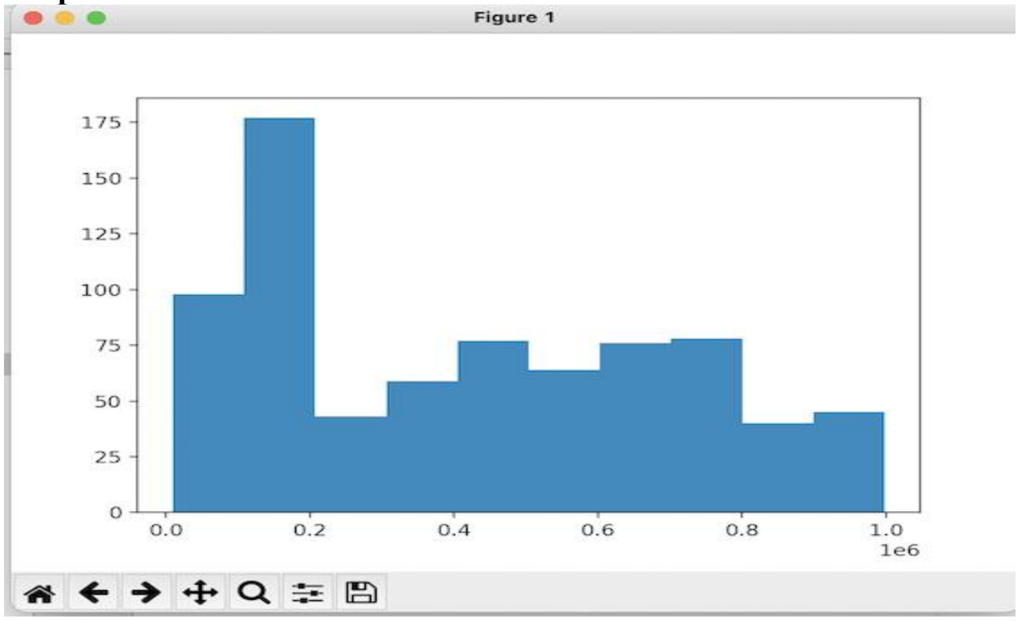
1.Data: This is the input dataset for which you want to create a histogram. It can be a list, array, or a column from a data frame.
2.X: Array or sequence of array.
3.Bins: Optional parameters contains integer or sequence or strings.
4.Range: The range parameter allows you to specify the minimum and maximum values for the bins. Data points outside this range are not included in the histogram.
5.**Density** (Normalization): The density parameter, if set to `True`, normalizes the histogram such that the total area under the histogram is equal to 1. This is useful when comparing histograms with different sample sizes or data ranges.

6. **Color**: You can specify the color of the bars in the histogram to differentiate it from other plots or to match a specific color scheme.
7.**Edgecolor**: The edgecolor parameter allows you to set the color of the edges of the histogram bars.
8.**Alpha** (Transparency): The alpha parameter controls the transparency of the histogram bars. A lower alpha value makes the bars more transparent, while a higher value makes them more opaque.
9. **Label**: Labels can be provided for the X and Y axes, as well as for the title of the histogram. This helps to provide context and understanding for the data being presented.
9. **Axis Limits**: You can set specific limits for the X and Y axes to control the range of values displayed on the histogram.
10. **Grid**: The grid parameter allows you to turn on or off grid lines on the histogram plot, aiding in reading and interpreting the data.
11. **Cumulative**: If set to `True`, the cumulative parameter displays a cumulative histogram, showing the cumulative count of data points up to each bin.

Here's a simple example of creating a histogram using Python's `matplotlib` library:

```
# import all modules
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Read in the DataFrame
df = pd.read_csv("D:\schoolimprovement2010grants.csv")

# creating a histogram
plt.hist(df['Award_Amount'])
plt.show()
```

Output:

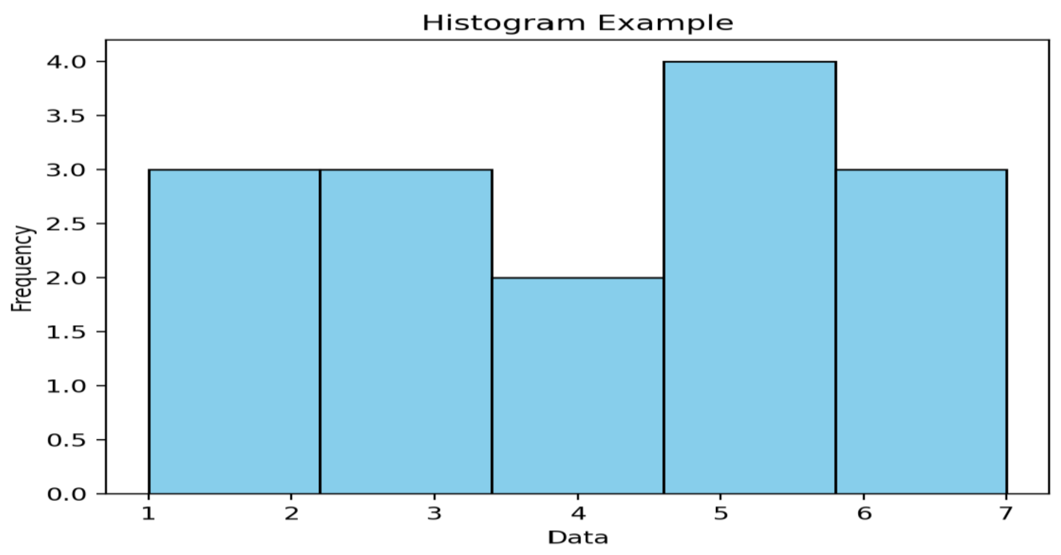


Prepared by T.SWETHA, MITS (CSE-DS)

python Program:

```
import matplotlib.pyplot as plt
# Sample data
data = [1, 2, 2, 3, 3, 3, 4, 4, 5, 5, 5, 5, 6, 6, 7]
# Create the histogram
plt.hist(data, bins=5, color='skyblue', edgecolor='black')
# Add labels and title
plt.xlabel('Data')
plt.ylabel('Frequency')
plt.title('Histogram Example')
# Display the plot
plt.show()
```

Output:



This code will create a histogram with 5 bins, displaying the frequency of data points in each bin. The bars will be colored sky blue with black edges, and the X and Y axes will be labeled accordingly. The title "Histogram Example" will be shown above the plot.

<u>Pie Chart:</u>

A pie chart is a circular data visualization that displays the proportion of different categories in a dataset as slices of a pie. The size of each slice represents the percentage of data points belonging to each category relative to the whole. Below is a general description of the parameters commonly used in pie chart functions across various programming languages and plotting libraries:

Syntax:

Matplotlib.pyplot.pie(dat,explode=none,labels=none,colors=none,autoput=none,shodow=false)

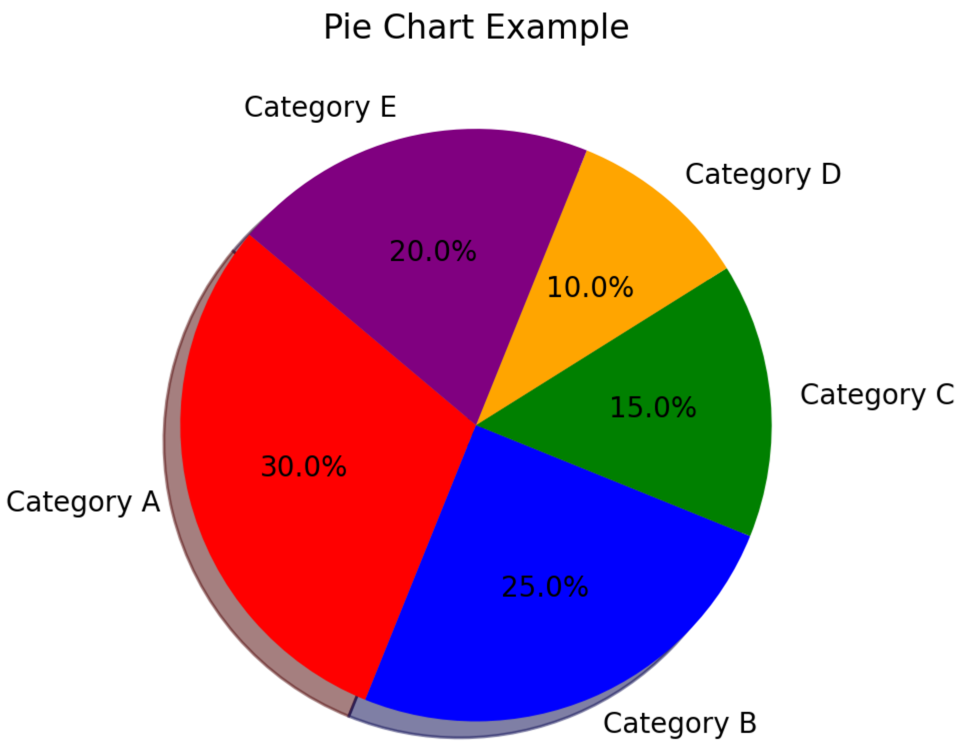
Here's a simple example of creating a pie chart using Python's `matplotlib` library:

Data: This is the input dataset that you want to visualize using the pie chart. It is typically a list or an array containing numerical values representing the proportions or frequencies of each category.
2. **Labels**: Labels are the names or descriptions of each category in the dataset. They are displayed near the corresponding slice in the pie chart.
3. **Colors**: You can specify a list of colors to represent each slice in the pie chart. This helps differentiate the categories visually.
4. **Startangle**: The startangle parameter allows you to rotate the pie chart by a specified angle. It determines the starting position of the first slice. By default, the first slice starts at the 0-degree angle, which corresponds to the positive x-axis in a Cartesian coordinate system.
5.**Explode**: If you want to emphasize a particular slice or separate it from the rest of the pie, you can use the explode parameter. It is a list of values that determines how much each slice is "exploded" from the center of the pie.
6. **Shadow**: Setting the shadow parameter to `True` adds a shadow effect behind the pie chart, giving it a three-dimensional appearance.
7. **Autopct**: This parameter controls the format of the percentage labels displayed on each slice. You can use string formatting to customize how the percentages are presented.
8.**Pctdistance**: If you use the autopct parameter to display percentages, the pct distance parameter allows you to set the distance of the percentage labels from the center of the pie.
9. **Label distance**: The label distance parameter sets the distance of the category labels from the center of the pie.
10. **Radius**: You can specify the radius of the pie chart to control its size. By default, the radius is set to 1.
11. **Counterclock**: If set to `True`, the counterclock parameter makes the slices of the pie chart drawn counterclockwise.
12. **Wedgeprops**: This parameter allows you to set properties for the wedges (slices) in the pie chart, such as edge color and width.

~~~
```python
import matplotlib.pyplot as plt

# Sample data
data = [30, 25, 15, 10, 20]

# Labels for the categories
labels = ['Category A', 'Category B', 'Category C', 'Category D', 'Category E']

# Colors for the slices
colors = ['red', 'blue', 'green', 'orange', 'purple']
~~~

```
# Create the pie chart
plt.pie(data, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=140)

# Add a title
plt.title('Pie Chart Example')

# Display the plot
plt.show()
```

OutPut:



This code will create a pie chart with five slices representing the proportions of different categories in the data. Each slice will be colored according to the specified colors, and the percentages of each category will be displayed with one decimal point. The pie chart will have a shadow effect and start at the 140-degree angle. The title "Pie Chart Example" will be shown above the chart.

<u>FACET PLOT:</u>

The term "facet plot" typically refers to a type of data visualization that displays multiple subsets of a dataset as separate plots arranged in a grid or matrix. Each plot in the grid represents a subset of the data, often based on one or more categorical variables. Facet plots are also known as small multiple plots or trellis plots. The specific syntax and available parameters for creating facet plots can vary depending on the programming language or plotting library you are using. Here, I'll provide a general description of the parameters commonly used in facet plot functions:

Syntax:

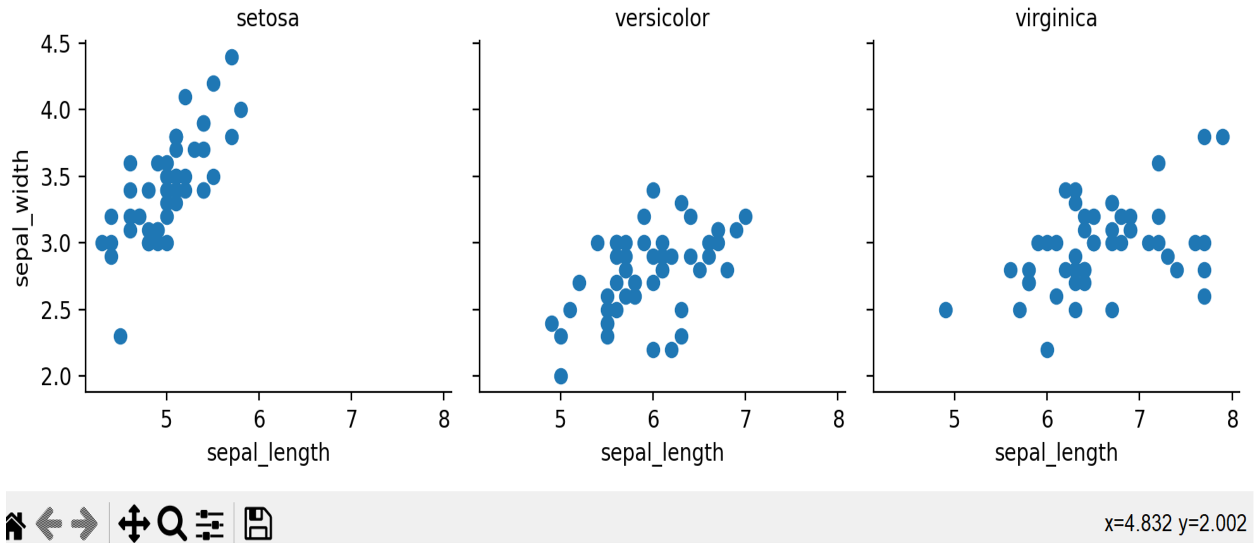
Class selection.fact grid(data,*,row=none)

1. **Data**: This is the input dataset that contains the variables you want to visualize in the facet plot.
2. **Facet Grid/Grid Layout**: The facet grid or grid layout specifies how the plots will be arranged in the grid. It typically involves specifying the categorical variables used for rows and columns to divide the data into subsets.
3. **Plot Type**: You can choose the type of plot for each facet, such as line plots, bar plots, scatter plots, etc. The specific parameter to set the plot type may vary based on the plotting library.
4. **X and Y Variables**: These parameters specify which variables from the dataset will be used as the x-axis and y-axis variables for each plot.
5. **Color, Size, and other Aesthetics**: You can set aesthetics such as color, size, shape, etc., to further differentiate data points or lines in the plots based on other variables in the dataset.
6. **Facet Labels**: You can add labels or titles for each facet plot in the grid to provide context and understanding of the data.
7. **Axis Labels and Titles**: Labels and titles for the X and Y axes can be specified to describe the data being visualized.
8.**Legends**: If the plot involves multiple groups or categories, you can add a legend to identify each group in the facet grid.
9.**Facet Scales**: Some plotting libraries allow you to set scales independently for each facet, controlling the range of values displayed on the axes.
10.**Facet Themes**: You may have the option to apply pre-defined themes or customize the appearance of the facet grid, such as background color, grid lines, font sizes, etc.

For example, in Python's `seaborn` library, you can create facet plots using the `FacetGrid` or `catplot` functions. Here's a simple example of creating a facet grid of scatter plots using `seaborn`:

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

# Sample data
data = sns.load_dataset("iris")

# Create a facet grid of scatter plots
g = sns.FacetGrid(data, col="species")
g.map(plt.scatter, "sepal_length", "sepal_width")

# Add a title for each facet
g.set_titles("{col_name}")

# Display the plot
plt.show()
```

Output:



In this example, the `FacetGrid` function divides the data into subsets based on the "species" column, creating separate scatter plots for each species. The `map` function is used to specify the scatter plot and the variables to be plotted on the X and Y axes. The `set_titles` function adds a title to each facet based on the species name.

PAIR PLOT:

A pair plot, also known as a scatter plot matrix, is a data visualization that shows pairwise relationships between multiple variables in a dataset. It displays scatter plots for each combination of variables, allowing you to examine how different variables relate to one another. Pair plots are particularly useful for exploring correlations and identifying patterns in multivariate data. The syntax and available parameters for creating a pair plot can vary depending on the programming language or plotting library you are using.

Syntax:

seaborn.pairplot(data, **kwargs)

Here, provides a general description of the parameters commonly used in pair plot functions:

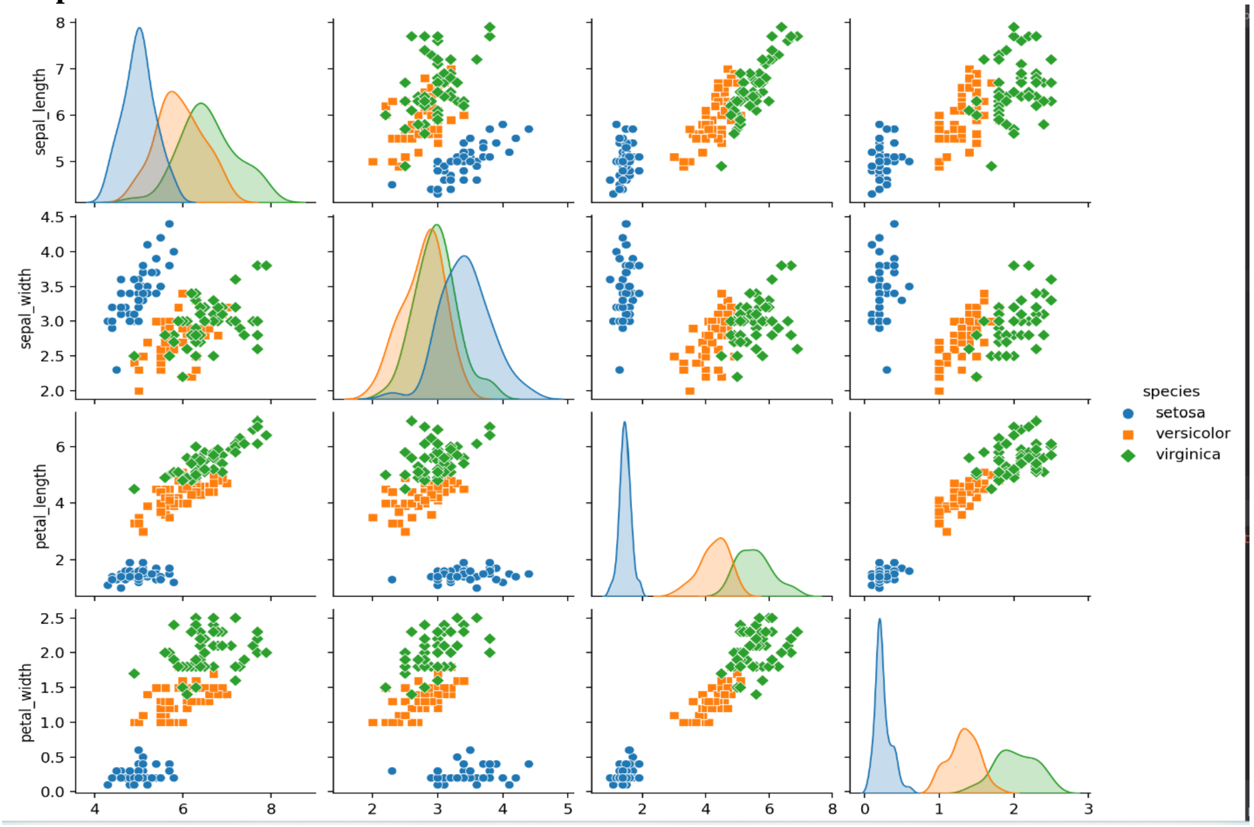
1.**Data**: This is the input dataset that contains the variables you want to visualize in the pair plot.
2.**Hue**: If you have a categorical variable in the dataset that you want to use for color differentiation, you can specify the `hue` parameter. This will color the data points in the scatter plots based on the categories of the specified variable.
3.**Markers and Colors**: You can set specific markers and colors for the data points in the scatter plots to differentiate between different groups or categories.
4.**Diagonal Plot Type**: The diagonal plot type refers to the type of plot that will be displayed on the diagonal of the pair plot. It is common to use histograms or kernel density plots to visualize the distribution of each variable on the diagonal.

5.**Palette**: If the `hue` parameter is specified, the `palette` parameter allows you to choose a color palette for the different categories.
6.**Scatter Plot Aesthetics**: You can set various aesthetics for the scatter plots, such as transparency (alpha), marker size, etc.
7.**Labels**: You can add labels or titles for the X and Y axes to describe the data being visualized.
8.**Axis Limits**: You can set specific limits for the X and Y axes to control the range of values displayed in the scatter plots.
9.**Correlation Coefficients**: Some libraries provide options to display correlation coefficients or statistical metrics on the pair plot, which can be useful to quantify the relationships between variables.

Here's a simple example of creating a pair plot using Python's `seaborn` library:

```python
import seaborn as sns
import matplotlib.pyplot as plt
# Sample data
data = sns.load_dataset("iris")
# Create the pair plot
sns.pairplot(data, hue="species", markers=["o", "s", "D"])
# Display the plot
plt.show()
```

Output:



In this example, the `pairplot` function from `seaborn` creates a pair plot of the variables in the "iris" dataset. The `hue` parameter is set to "species," which means that the data points will be colored based on the different species of iris flowers. The `markers` parameter specifies different markers for each species to differentiate them in the scatter plots. The pair plot will display scatter plots for all combinations of variables and histograms on the diagonal.

AREA CHART:

An area chart is a type of data visualization that displays the data as a series of filled areas, with each area representing the cumulative total of a variable over a continuous interval or time period. It is often used to show the trend and magnitude of changes in one or more variables over time or another continuous domain.

Syntax:

```
plotly.express.area(data_frame=None, x=None, y=None, line_group=None, color=None, hover_name=None, hover_data=None, custom_data=None, text=None, facet_row=None, facet_col=None, facet_col_wrap=0, animation_frame=None, animation_group=None, category_orders={}, labels={}, color_discrete_sequence=None, color_discrete_map={}, orientation=None, groupnorm=None, log_x=False, log_y=False, range_x=None, range_y=None, line_shape=None, title=None, template=None, width=None, height=None)
```

Below is a general description of the parameters commonly used in area chart functions across various programming languages and plotting libraries:

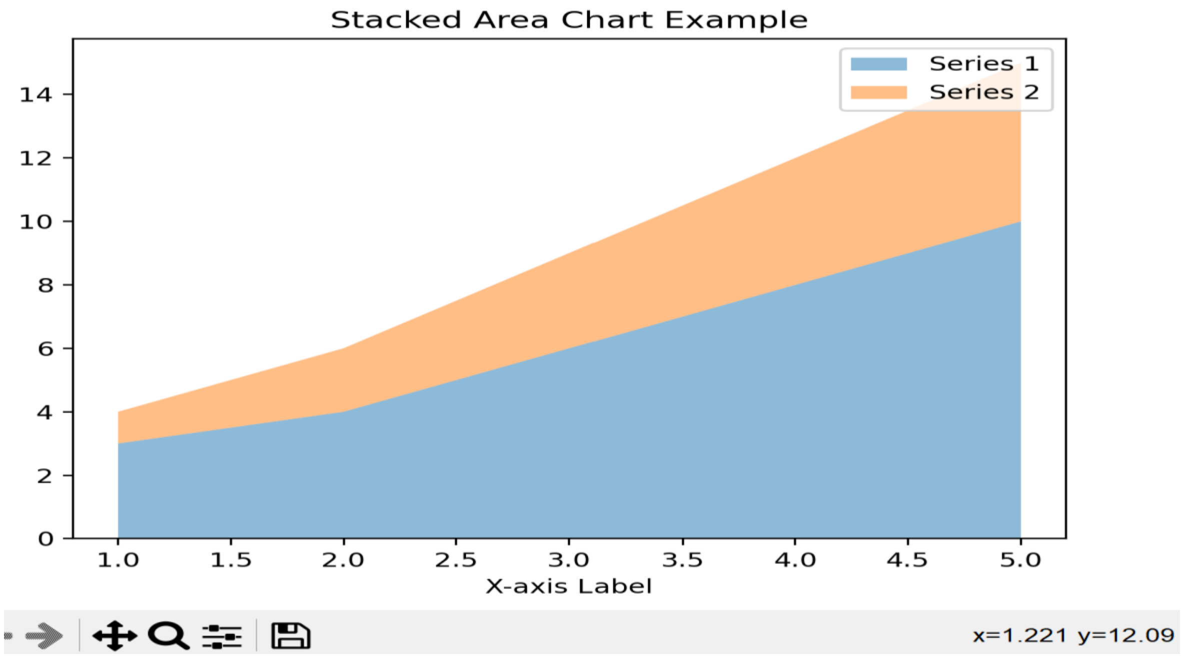
1.**X and Y Data**: These are the input datasets for the area chart. The X data represents the independent variable, which is typically time or another continuous domain, and the Y data represents the dependent variable(s) whose cumulative totals will be displayed as filled areas.
2. **Stacked**: The stacked parameter determines whether the areas will be stacked on top of each other (stacked area chart) or overlaid (unstacked area chart). In a stacked area chart, the areas sum up to the total, while in an unstacked area chart, they are simply overlaid.
3.**Colors**: You can specify the colors for the areas to differentiate between different groups or categories in the data.
4.**Alpha** (Transparency): The alpha parameter controls the transparency of the filled areas. A lower alpha value makes the areas more transparent, while a higher value makes them more opaque.
5.**Line Style**: Some plotting libraries allow you to customize the appearance of the line connecting the points on the area chart.
6.**Labels**: Labels can be added for the X and Y axes, as well as for the title of the area chart to provide context and understanding of the data.
7.**Legend**: If the area chart involves multiple groups or categories, you can add a legend to identify each group in the chart.
8.**Axis Limits**: You can set specific limits for the X and Y axes to control the range of values displayed on the area chart.
9.**Fill Between**: In some libraries, there might be a parameter to specify whether to fill between the area chart and a specific reference line or value.

10.**Baseline**: The baseline parameter allows you to set a specific value for the baseline of the area chart, around which the areas will be filled.

Here's a simple example of creating a stacked area chart using Python's `matplotlib` library:

```python

import matplotlib.pyplot as plt
# Sample data
x_data = [1, 2, 3, 4, 5]
y_data1 = [3, 4, 6, 8, 10]
y_data2 = [1, 2, 3, 4, 5]

# Create the stacked area chart
plt.stackplot(x_data, y_data1, y_data2, labels=['Series 1', 'Series 2'], alpha=0.5)

# Add labels and title
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')
plt.title('Stacked Area Chart Example')

# Show the legend
plt.legend()

# Display the plot
plt.show()
```

OUTPUT:



Prepared by T.SWETHA, MITS (CSE-DS)

This code will create a stacked area chart with two filled areas representing the cumulative totals of two series of data points. The areas will be labeled with "Series 1" and "Series 2" in the legend. The X and Y axes will be labeled accordingly, and the title "Stacked Area Chart Example" will be shown above the chart.

VIOLIN PLOT:

A violin plot is a data visualization that combines elements of a box plot with a kernel density plot. It displays the distribution of a continuous variable or numeric data across different categories or groups. Violin plots are particularly useful for comparing the distribution of data between multiple groups and identifying patterns such as data concentration and skewness.

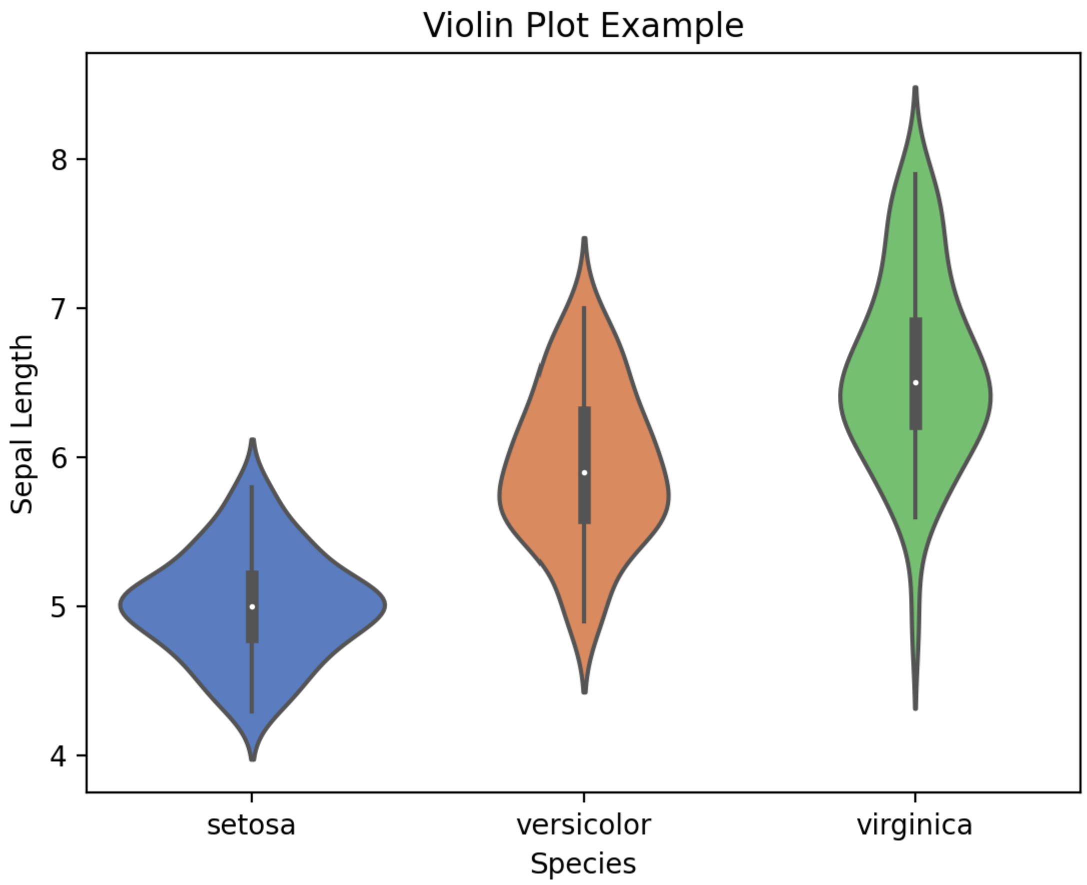
Below is a general description of the parameters commonly used in violin plot functions across various programming languages and plotting libraries:

1.**Data**: This is the input dataset that contains the variables you want to visualize in the violin plot. It can be a list, array, or a column from a data frame.
2.**x/y**: Depending on the library, you can specify whether you want the violin plots to be displayed horizontally (y-axis) or vertically (x-axis). The variable you want to compare between groups typically goes here.
3.**Grouping/Category Variable**: If you have a categorical variable in the dataset that you want to use for grouping the data, you can specify it as a parameter. The violin plots will be created for each category separately.
4.**Orientation**: Some plotting libraries offer the option to explicitly set the orientation of the violin plots (e.g., horizontal or vertical).
5.**Color and Style**: You can set the color and style of the violin plots to differentiate between different groups or categories.
6.**Width**: The width parameter controls the width of the individual violin plots. It allows you to make them wider or narrower depending on your preference.
7.**Inner Quartile Range (IQR)**: The IQR parameter controls the range of the data used to draw the box plot part of the violin. By default, it is set to 1.5, but you can change it to highlight different parts of the data distribution.
8.**Kernel Density Estimation (KDE)**: Violin plots include a kernel density plot that visualizes the distribution of the data. You may have the option to control the kernel bandwidth or other parameters related to KDE.
9.**Trim:** The trim parameter allows you to trim the violin tails. Trimming removes extreme data points from the kernel density plot, which can be useful in reducing visual clutter.
10.**Scale**: The scale parameter allows you to scale the width of the violin plots based on the number of data points in each group. Setting it to 'width' ensures that each group's violin has the same width, while 'area' scales them based on the number of data points.
11.**Inner/Outer Points**: Some libraries allow you to show individual data points within the violin plot to highlight the data distribution.
12.**Labels and Titles**: Labels can be added for the X and Y axes, as well as for the title of the violin plot to provide context and understanding of the data.

Here's a simple example of creating a violin plot using Python's `seaborn` library:

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

# Sample data
data = sns.load_dataset("iris")

# Create the violin plot
sns.violinplot(x="species", y="sepal_length", data=data, palette="muted")

# Add labels and title
plt.xlabel('Species')
plt.ylabel('Sepal Length')
plt.title('Violin Plot Example')

# Display the plot
plt.show()
```

OutPUT:



In this example, the `violinplot` function from `seaborn` creates a violin plot of the "sepal_length" variable for each species of iris flower in the "iris" dataset. The X-axis represents the species, and the Y-axis represents the sepal length. The violin plots are colored using the "muted" color palette, and the title "Violin Plot Example" is shown above the chart.

<u>BAR CHART:</u>

A bar chart is a common type of data visualization that represents categorical data with rectangular bars. Each bar's length or height corresponds to the value of the data it represents. Bar charts are effective for comparing discrete categories and showing the magnitude of different data points in a visually clear manner.

Syntax:

Plt.bar(X,height,width,bottom,align)

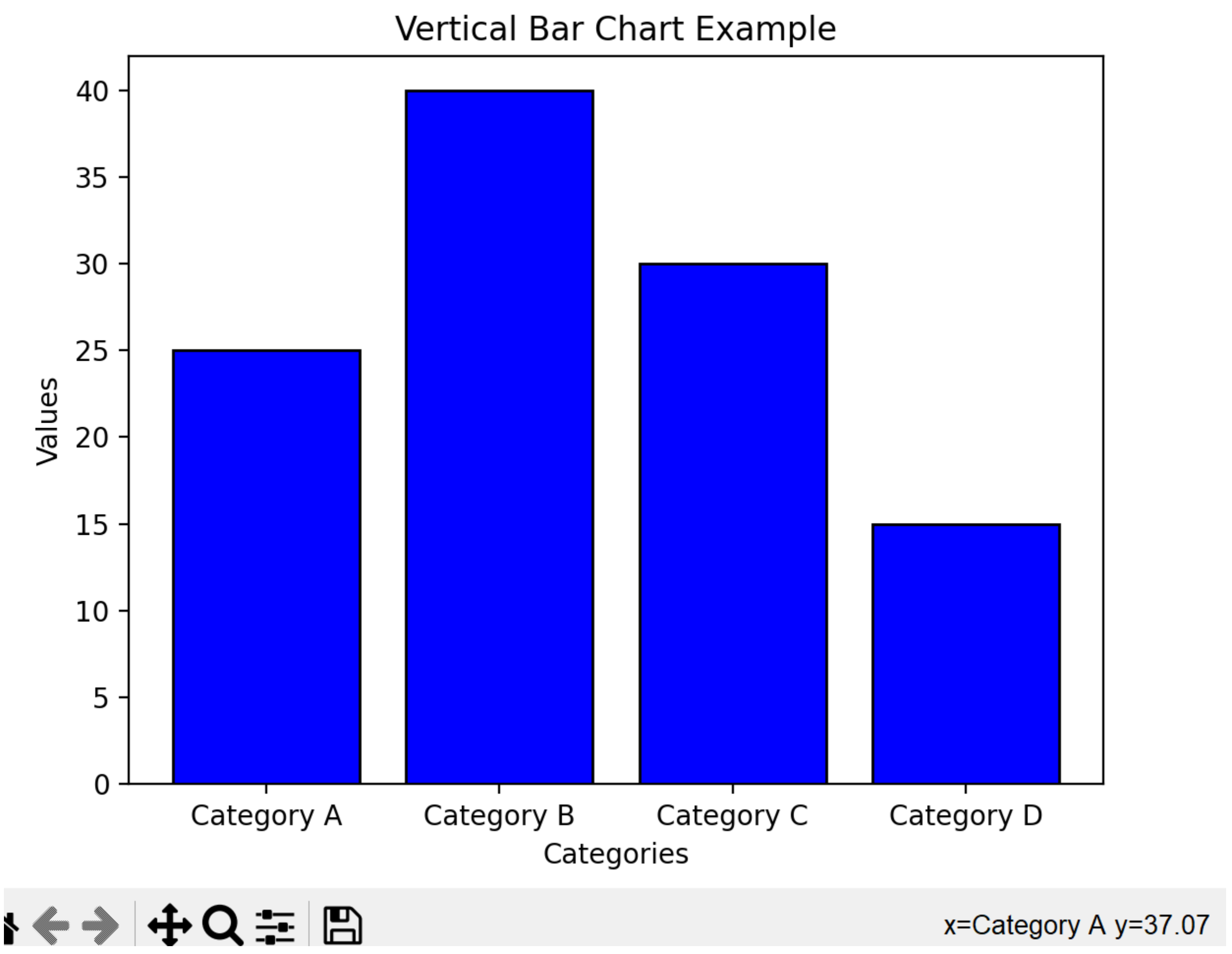
Below is a general description of the parameters commonly used in bar chart functions across various programming languages and plotting libraries:

1.**Data**: This is the input dataset that contains the variables you want to visualize in the bar chart. It typically consists of categorical variables and their corresponding numerical values.
2.**X and Y Variables**: The X variable represents the categorical data, usually displayed on the horizontal axis, while the Y variable represents the numerical values, displayed on the vertical axis.
3.**Bar Width**: The bar width parameter allows you to set the width of the bars in the bar chart. This can be useful for customizing the appearance of the chart.
4.**Color**: You can specify the color of the bars to differentiate between different groups or categories in the data.
5.**Orientation**: Depending on the plotting library, you may have the option to create either vertical or horizontal bar charts. The orientation parameter allows you to specify the desired orientation.
6.**Bar Alignment**: For vertical bar charts, you can set the bar alignment parameter to control whether the bars are centered on the X-axis labels or aligned to the left or right.
7.**Edgecolor**: The edgecolor parameter allows you to set the color of the edges of the bars.
8.**Labels**: Labels can be added for the X and Y axes, as well as for the title of the bar chart to provide context and understanding of the data.
9.**Legend**: If the bar chart involves multiple groups or categories, you can add a legend to identify each group in the chart.
10.**Axis Limits**: You can set specific limits for the X and Y axes to control the range of values displayed on the bar chart.
11.**Error Bars**: Some libraries allow you to add error bars to the bar chart, which can represent uncertainties or variations in the data.

Here's a simple example of creating a vertical bar chart using Python's `matplotlib` library:

~~~
```python
import matplotlib.pyplot as plt

# Sample data
categories = ['Category A', 'Category B', 'Category C', 'Category D']
values = [25, 40, 30, 15]

# Create the bar chart
plt.bar(categories, values, color='blue', edgecolor='black')

# Add labels and title
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Vertical Bar Chart Example')

# Display the plot
plt.show()
~~~

OUTPUT:



x=Category A y=37.07

In this example, the `bar` function from `matplotlib` creates a vertical bar chart with four bars representing the values of different categories. The bars are colored blue with black edges, and the X and Y axes are labeled accordingly. The title "Vertical Bar Chart Example" is shown above the chart.

Result

Thus the various types of plots are drawn successfully.