Experiment 11

Write Python programs to implement different types of plots using Numpy and Matplotlib.

Roll No.	01
Name	Aamir Ansari
Class	D10A
Subject	Python Lab
LO Mapped	LO1: Understand the structure, syntax, and semantics of the Python language. LO6: Design and Develop cost-effective robust applications using the latest Python trends and technologies.

Aim: Write Python programs to implement different types of plots using Numpy and Matplotlib.

Introduction:

Data visualization: It is the process of converting raw data into easily understandable pictorial representation, that enables fast and effective decisions. Data visualization is a strategy where we represent the quantitative information in a graphical form.

Data visualization is the discipline of trying to understand data by placing it in a visual context so that patterns, trends and correlations that might not otherwise be detected can be exposed. Python offers multiple great graphing libraries that come packed with lots of different features.

Data visualization techniques:

1. Pie chart
2. Bar graph
3. Histogram
4. Tree map
5. Scatter plot
6. Line chart
7. Bubble chart

Python provides many libraries for data visualization like matplotlib, seaborn, ggplot etc.

Matplotlib: Matplotlib library is a graph plotting library of python. Using matplotlib we can plot different scatter plots, line graphs, bar graphs, pie charts and histograms. Using these plots we can visualize our data. It provides an object-oriented APIs for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+.

Pyplot: It provides the state-machine interface to the plotting library in matplotlib. It means that figures and axes are implicitly and automatically created to achieve the desired plot. The pyplot interface is generally preferred for non-interactive plotting.

Plotting using Matplotlib: Matplotlib library is used for creating static, animated, and interactive 2D- plots or figures in Python. It can be installed using the following pip command from the command prompt

```
pip install matplotlib
```

For plotting using Matplotlib, we need to import its Pyplot module using the following command

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

The pyplot module of matplotlib contains a collection of functions that can be used to work on a plot. The plot() function of the pyplot module is used to create a figure. A figure is the overall window where the outputs of pyplot functions are plotted. A figure contains a plotting area,

legend, axis labels, ticks, title, etc. Each function makes some change to a figure. Below is the list of pyplot functions to plot different charts.

Function	Description
plot(*args[, scalex, scaley, data])	Plot x versus y as lines and/or markers.
bar(x, height[, width, bottom, align, data])	Make a bar plot.
boxplot(x[, notch, sym, vert, whis, ...])	Make a box and whisker plot.
hist(x[, bins, range, density, weights, ...])	Plot a histogram.
pie(x[, explode, labels, colors, autopct, ...])	Plot a pie chart.
scatter(x, y[, s, c, marker, cmap, norm, ...])	A scatter plot of x versus y.

Numpy: Another library which helps in the process of plotting graphs/charts using pyplot is NumPy. NumPy stands for numerical Python. NumPy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object and tools for working with these arrays.

We can install NumPy using the popular Python package installer, pip. Type the following command at the command prompt

pip install numpy

NumPy will get installed onto your system and will be ready to be used.

Pandas: Pandas is an open-source Python Library providing high-performance data manipulation and analysis tools using its powerful data structures. The name Pandas is derived from the word Panel Data, an Econometrics from Multidimensional data.

Standard Python distribution doesn't come bundled with Pandas modules. A lightweight alternative is to install NumPy using the popular Python package installer pip.

pip install pandas

Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze.

Creating different visualizations: We can create different types of visualization using matplotlib. There are different types of charts for analyzing & presenting data.

1. **Bar graph plot**: A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent. The bar plots can be plotted horizontally or vertically. A bar chart describes the comparisons

between the discrete categories. One of the axes of the plot represents the specific categories being compared, while the other axis represents the measured values corresponding to those categories.

Creating a bar plot: The matplotlib API in Python provides the **bar()** function which can be used in MATLAB style use or as an object-oriented API. The syntax of the **bar()** function to be used with the axes is as follows

plt.bar(x, height, width, bottom, align)

The function creates a bar plot bounded with a rectangle depending on the given parameters.

Customising a bar plot: A bar plot can be customized on the basis of several aspects.

1. **Multiple bar plots**: Multiple bar plots are used when comparison among the data set is to be done when one variable is changing. We can easily convert it as a stacked area bar chart, where each subgroup is displayed by one on top of the others. It can be plotted by varying the thickness and position of the bars.

```
IT = [12, 30, 1, 8, 22]
ECE = [28, 6, 16, 5, 10]
CSE = [29, 3, 24, 25, 17]

br1 = np.arange(len(IT))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]

plt.bar(br1, IT, color ='r', width = barWidth, edgecolor ='grey', label ='IT')
plt.bar(br2, ECE, color ='g', width = barWidth, edgecolor ='grey', label ='ECE')
plt.bar(br3, CSE, color ='b', width = barWidth, edgecolor ='grey', label ='CSE')
```

2. **Stacked bar plot**: Stacked bar plots represent different groups on top of one another. The height of the bar depends on the resulting height of the combination of the results of the groups. It goes from the bottom to the value instead of going from zero to value.

```
N = 5
boys = (20, 35, 30, 35, 27)
girls = (25, 32, 34, 20, 25)
boyStd = (2, 3, 4, 1, 2)
girlStd = (3, 5, 2, 3, 3)
ind = np.arange(N)
width = 0.35

fig = plt.subplots(figsize =(10, 7))
p1 = plt.bar(ind, boys, width, yerr = boyStd)
p2 = plt.bar(ind, girls, width, bottom = boys, yerr = girlStd)
```

```
plt.ylabel('Contribution')
plt.title('Contribution by the teams')
plt.xticks(ind, ('T1', 'T2', 'T3', 'T4', 'T5'))
plt.yticks(np.arange(0, 81, 10))
plt.legend((p1[0], p2[0]), ('boys', 'girls'))
```

3. **Horizontal bar charts**: If you want the bars to be displayed horizontally instead of vertically, use the **barh()** function.

```
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y)
```

4. **Bar color**: The **bar()** and **barh()** takes the keyword argument color to set the color of the bars.

```
x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "red")
```

2. **Pie chart**: A Pie Chart is a circular statistical plot that can display only one series of data. The area of the chart is the total percentage of the given data. The area of slices of the pie represents the percentage of the parts of the data. The slices of pie are called wedges. The area of the wedge is determined by the length of the arc of the wedge. The area of a wedge represents the relative percentage of that part with respect to whole data.

Creating a pie chart: Matplotlib API has a **pie()** function in its pyplot module which creates a pie chart representing the data in an array.

matplotlib.pyplot.pie(data, explode=None, labels=None, colors=None, autopct=None, shadow=False)

Customising a pie chart: A pie chart can be customized on the basis of several aspects.

1. **Labels**: Add labels to the pie chart with the label parameter. The label parameter must be an array with one label for each wedge.

```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
```

2. **Start angle**: As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a startangle parameter. The startangle parameter is defined with an angle in degrees, default angle is 0.

```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels, startangle = 90)
```

3. **Explode**: The explode parameter allows one of the wedges to stand out. The explode parameter, if specified, and not None, must be an array with one value for each wedge. Each value represents how far from the center each wedge is displayed.

```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0, 0]

plt.pie(y, labels = mylabels, explode = myexplode)
```

4. **Shadow**: Add a shadow to the pie chart by setting the shadows parameter to True.

```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0, 0]

plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)
```

5. **Colors**: You can set the color of each wedge with the colors parameter. The colors parameter, if specified, must be an array with one value for each wedge.

```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
mycolors = ["black", "hotpink", "b", "#4CAF50"]

plt.pie(y, labels = mylabels, colors = mycolors)
```

6. **Legend**: To add a list of explanations for each wedge, use the **legend()** function. To add a header to the legend, add the title parameter to the legend function.

```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend()
```

3. **Box plot**: A Box Plot is also known as Whisker plot is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum. In the box plot, a box is created from the first quartile to the third quartile, a vertical line is also there which goes through the box at the median. Here x-axis denotes the data to be plotted while the y-axis shows the frequency distribution.

Creating box plot: The matplotlib.pyplot module of matplotlib library provides the **boxplot()** function with the help of which we can create box plots.

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

The data values given to the **ax.boxplot()** method can be a Numpy array or Python list or Tuple of arrays. Let us create the box plot by using **numpy.random.normal()** to create some random data, it takes mean, standard deviation, and the desired number of values as arguments.

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

np.random.seed(10)
data = np.random.normal(100, 20, 200)

fig = plt.figure(figsize =(10, 7))

plt.boxplot(data)

plt.show()
```

Customising box plot: The data values given to the **ax.boxplot()** method can be a Numpy array or Python list or Tuple of arrays. Box plot is created by using **numpy.random.normal()** to create some random data, it takes mean, standard deviation, and the desired number of values as arguments.

```
np.random.seed(10)

data_1 = np.random.normal(100, 10, 200)
data_2 = np.random.normal(90, 20, 200)
data_3 = np.random.normal(80, 30, 200)
data_4 = np.random.normal(70, 40, 200)
data = [data_1, data_2, data_3, data_4]

fig = plt.figure(figsize =(10, 7))
ax = fig.add_axes([0, 0, 1, 1])
bp = ax.boxplot(data)
```

4. **Histogram**: A histogram is basically used to represent data provided in a form of some groups. It is an accurate method for the graphical representation of numerical data distribution. It is a type of bar plot where X-axis represents the bin ranges while Y-axis gives information about frequency.

Creating a histogram: To create a histogram the first step is to create a bin of the ranges, then distribute the whole range of the values into a series of intervals, and the count the values which fall into each of the intervals. Bins are clearly identified as consecutive, non-overlapping intervals of variables. The **matplotlib.pyplot.hist()** function is used to compute and create histograms of x.

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

x = np.random.normal(170, 10, 250)

plt.hist(x)
plt.show()
```

Customising a histogram: Matplotlib provides a range of different methods to customize histogram. **matplotlib.pyplot.hist()** function itself provides many attributes with the help of which we can modify a histogram.The **hist()** function provide a patches object which gives access to the properties of the created objects, using this we can modify the plot according to our will.

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import colors
from matplotlib.ticker import PercentFormatter

np.random.seed(23685752)
N_points = 10000
n_bins = 20

x = np.random.randn(N_points)
y = .8 ** x + np.random.randn(10000) + 25

fig, axs = plt.subplots(1, 1,
                        figsize =(10, 7),
                        tight_layout = True)

axs.hist(x, bins = n_bins)

plt.show()
```

5. **Line chart**: Line plot/chart is a type of plot which displays information as a series of data points called markers connected by straight lines. In this type of plot, we need the measurement points to be ordered (typically by their X-axis values). The line chart is represented by a series of data points connected by a straight line.

Creating a line chart: To create a line chart in Pandas, we can call

```
<dataframe>.plot.line()
```

To make a line plot with matplotlib, we call

```
plt.plot()
```

The first argument is used for the data on the horizontal axis, and the second is used for the data on the vertical axis. This function generates your plot but it doesn't display it. To display the plot, we need to call the **plt.show()** function.

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints)
plt.show()
```

Customising a line chart: A line chart can be customized on the basis of several aspects.

1. **Line styles**:

Style	Display
Solid (default)	**-**
Dotted	**:**
Dashed	**--**
Dashdot	**-.**
None	

```
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, linestyle = 'dashdot')
```

2. **Line color**: You can use the keyword argument color or the shorter c to set the color of the line.

```
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, color = 'r')
```

3. **Line width**: You can use the keyword argument linewidth or the shorter lw to change the width of the line. The value is a floating number, in points.

```
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, linewidth = '20.5')
```

4. **Multiple lines**: You can plot as many lines as you like by simply adding more **plt.plot()** functions. You can also plot many lines by adding the points for the x- and y-axis for each line in the same **plt.plot()** function.

```
y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])

plt.plot(y1)
plt.plot(y2)
```

6. **Subplots**: Matplotlib Pyplot API has a convenience function called **subplots()** which acts as a utility wrapper and helps in creating common layouts of subplots, including the enclosing figure object, in a single call.

Creating a subplot: The **subplots()** function takes three arguments that describes the layout of the figure. The layout is organized in rows and columns, which are represented by the first and second argument. The third argument represents the index of the current plot.

```
matplotlib.pyplot.subplots(nrows=1, ncols=1, sharex=False, sharey=False, squeeze=True, subplot_kw=None, gridspec_kw=None, **fig_kw)
```

With the **subplots()** function you can draw multiple plots in one figure.

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

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
```

Customising a subplot: A subplot can be customized on the basis of several aspects.

1. **Title**: You can add a title to each plot with the **title()** function.

```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")
```

2. **Super title**: You can add a title to the entire figure with the **suptitle()** function.

```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")

plt.suptitle("MY SHOP")
```

7. **Scatter plot**: Scatter plots are used to observe relationships between variables and use dots to represent the relationship between them. Scatter plots are used to plot data points on horizontal and vertical axis in the attempt to show how much one variable is affected by another.

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

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])

plt.scatter(x, y)
plt.show()
```

Each row in the data table is represented by a marker. The position depends on its values in the columns set on the X and Y axes. A third variable can be set to correspond to the color or size of the markers, thus adding yet another dimension to the plot.

Creating a scatter plot: With Pyplot, you can use the **scatter()** function to draw a scatter plot. The **scatter()** function plots one dot for each observation. It needs two arrays of the same length, one for the values of the x-axis, and one for values on the y-axis.

matplotlib.pyplot.scatter(x_axis_data, y_axis_data, s=None, c=None, marker=None, cmap=None, vmin=None, vmax=None, alpha=None, linewidths=None, edgecolors=None)

Customising a scatter plot: A scatter plot can be customized on the basis of several aspects.

1. **Compare plots**: Two plots can be compared using scatter function.

```
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)
```

2. **Colors**: You can set your own color for each scatter plot with the color or the c argument.

```
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y, color = '#88c999')
```

3. **Color each dot**: You can even set a specific color for each dot by using an array of colors as value for the c argument.

```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors =
np.array(["red","green","blue","yellow","pink","black","orange","purple","beige",
"brown","gray","cyan","magenta"])

plt.scatter(x, y, c=colors)
```

4. **Colormap**: The Matplotlib module has a number of available colormaps. A colormap is like a list of colors, where each color has a value that ranges from 0 to 100. You can specify the colormap with the keyword argument cmap with the value of the colormap.

```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')
```

5. **Size**: You can change the size of the dots with the s argument. Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis.

```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])
```

```
plt.scatter(x, y, s=sizes)
```

6. **Alpha**: You can adjust the transparency of the dots with the alpha argument. Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis.

```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])

plt.scatter(x, y, s=sizes, alpha=0.5)
```

7. **Combine color size and alpha**: You can combine a colormap with different sizes on the dots. This is best visualized if the dots are transparent.

```
x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))
colors = np.random.randint(100, size=(100))
sizes = 10 * np.random.randint(100, size=(100))

plt.scatter(x, y, c=colors, s=sizes, alpha=0.5, cmap='nipy_spectral')

plt.colorbar()
```

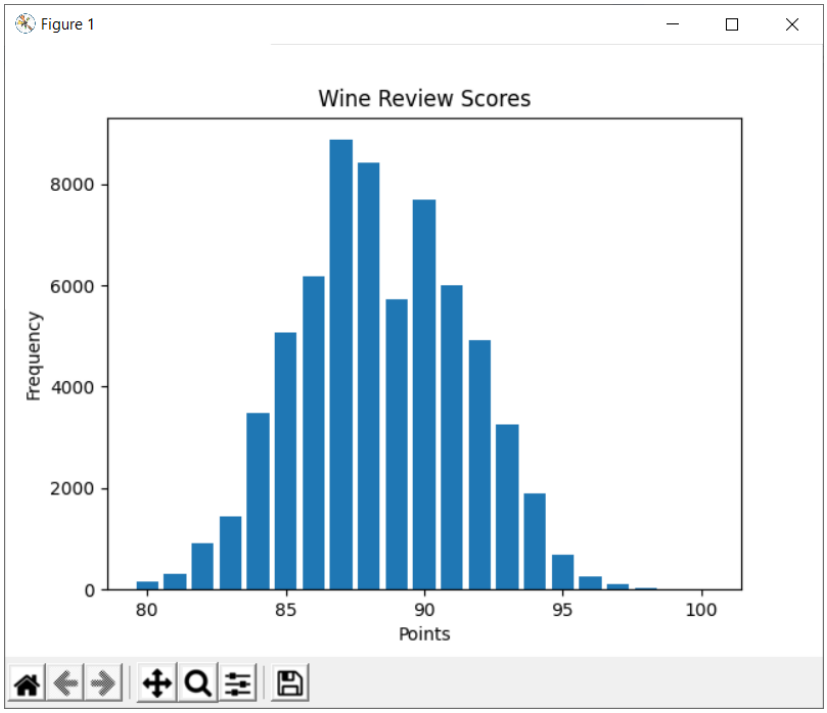
Results:

1. **Code**:

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

wine_reviews = pd.read_csv('winemag-data-130k-v2.csv', index_col=0)

fig, ax = plt.subplots()

data = wine_reviews['points'].value_counts()

points = data.index
frequency = data.values

ax.bar(points, frequency)

ax.set_title('Wine Review Scores')
ax.set_xlabel('Points')
ax.set_ylabel('Frequency')
```

plt.show()

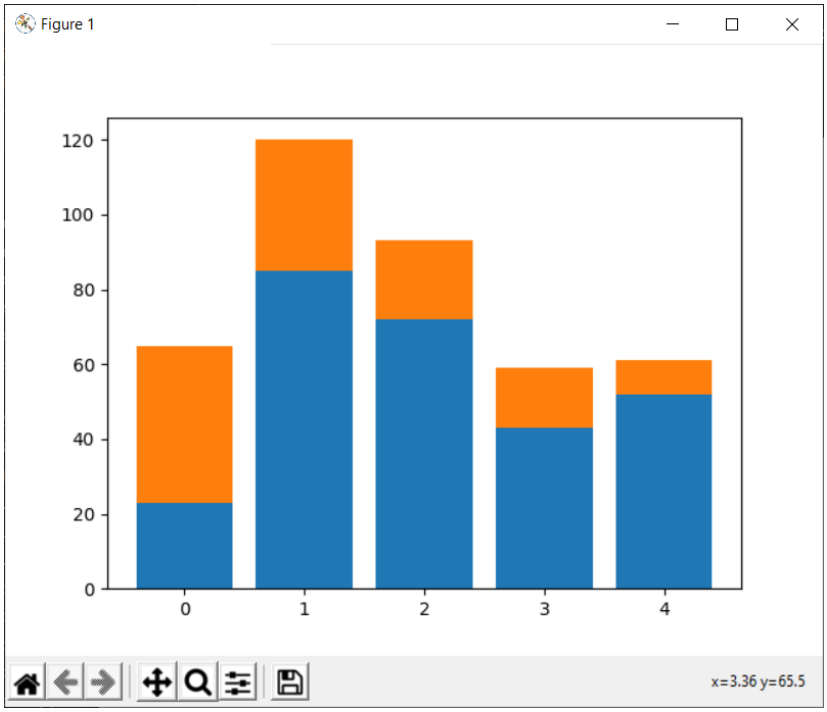
Output:



2. **Code**:

```
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
import matplotlib.pyplot as plt

data1 = [23,85, 72, 43, 52]
data2 = [42, 35, 21, 16, 9]

plt.bar(range(len(data1)), data1)
plt.bar(range(len(data2)), data2, bottom=data1)
plt.show()
```

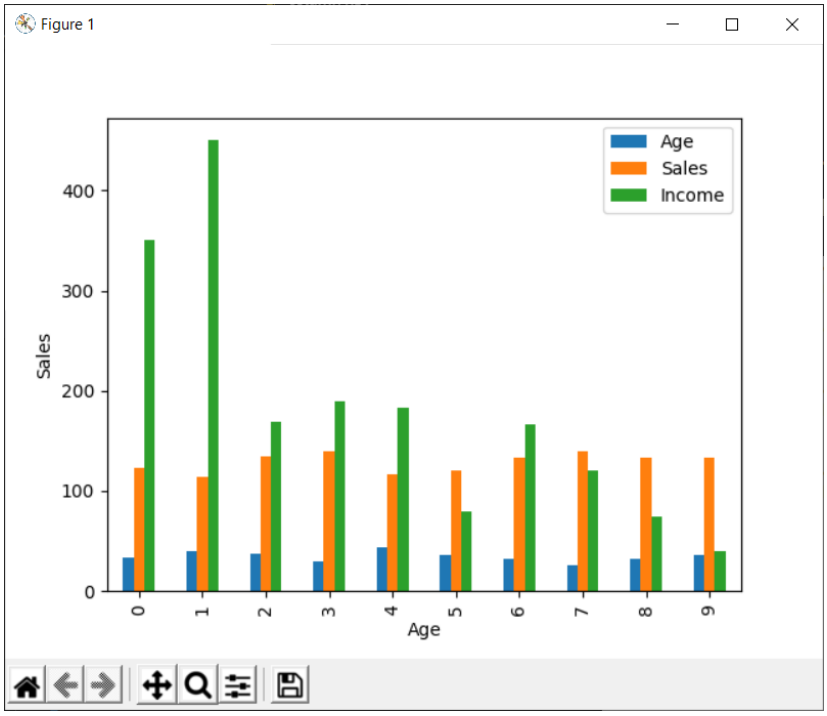
Output:



3. **Code**:

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

data = [['E001', 'M', 34, 123, 'Normal', 350],
        ['E002', 'F', 40, 114, 'Overweight', 450],
        ['E003', 'F', 37, 135, 'Obesity', 169],
        ['E004', 'M', 30, 139, 'Underweight', 189],
        ['E005', 'F', 44, 117, 'Underweight', 183],
        ['E006', 'M', 36, 121, 'Normal', 80],
        ['E007', 'M', 32, 133, 'Obesity', 166],
        ['E008', 'F', 26, 140, 'Normal', 120],
        ['E009', 'M', 32, 133, 'Normal', 75],
        ['E010', 'M', 36, 133, 'Underweight', 40] ]

df = pd.DataFrame(data, columns = ['EMPID', 'Gender', 'Age', 'Sales', 'BMI', 'Income'] )

df.plot.bar()

plt.bar(df['Age'], df['Sales'])
plt.xlabel("Age")
plt.ylabel("Sales")
plt.show()
```

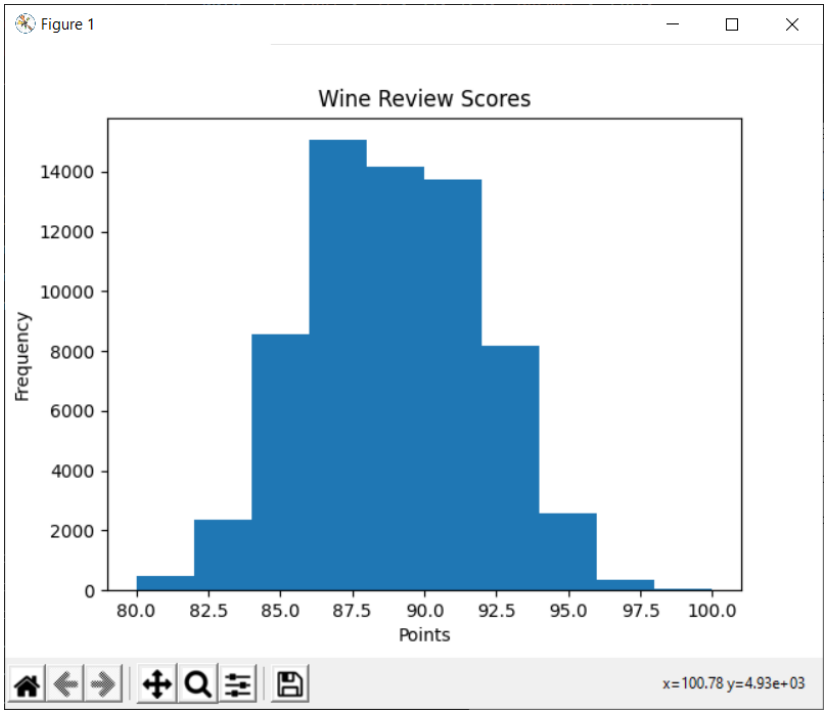
Output:



4. **Code**:

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

wine_reviews = pd.read_csv('winemag-data-130k-v2.csv', index_col=0)

fig, ax = plt.subplots()
ax.hist(wine_reviews['points'])

ax.set_title('Wine Review Scores')
ax.set_xlabel('Points')
ax.set_ylabel('Frequency')

plt.show()
```

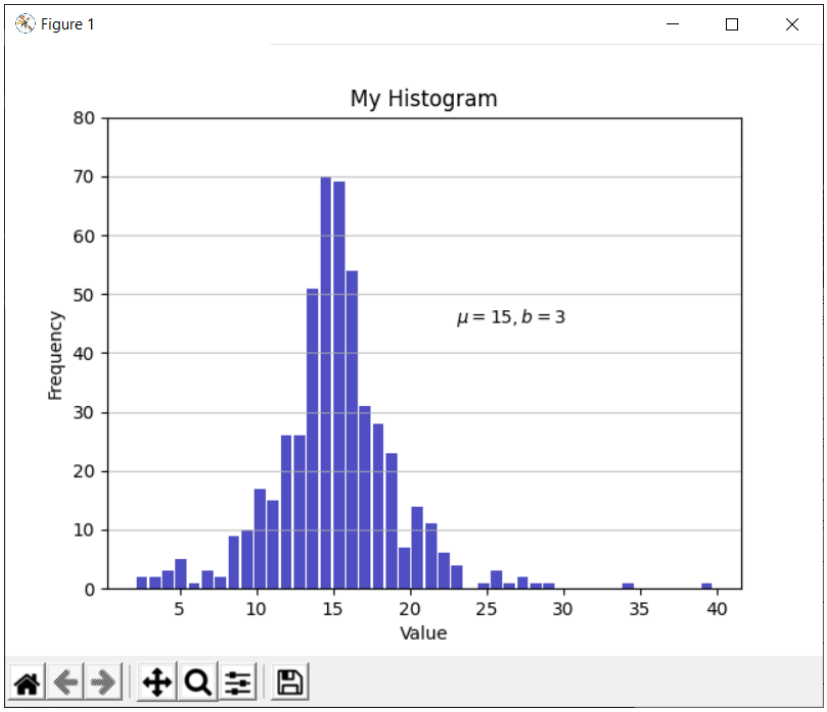
Output:



5. **Code**:

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

np.random.seed(444)
np.set_printoptions(precision=3)

d = np.random.laplace(loc=15, scale=3, size=500)

n, bins, patches = plt.hist(x=d, bins='auto', color='#0504aa', alpha=0.7, rwidth=0.85)

plt.grid(axis='y', alpha=0.75)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('My Histogram')
plt.text(23, 45, r'$\mu=15, b=3$')
maxfreq = n.max()

plt.ylim(ymax=np.ceil(maxfreq / 10) * 10 if maxfreq % 10 else maxfreq + 10)

plt.show()
```

Output:



6. **Code**:

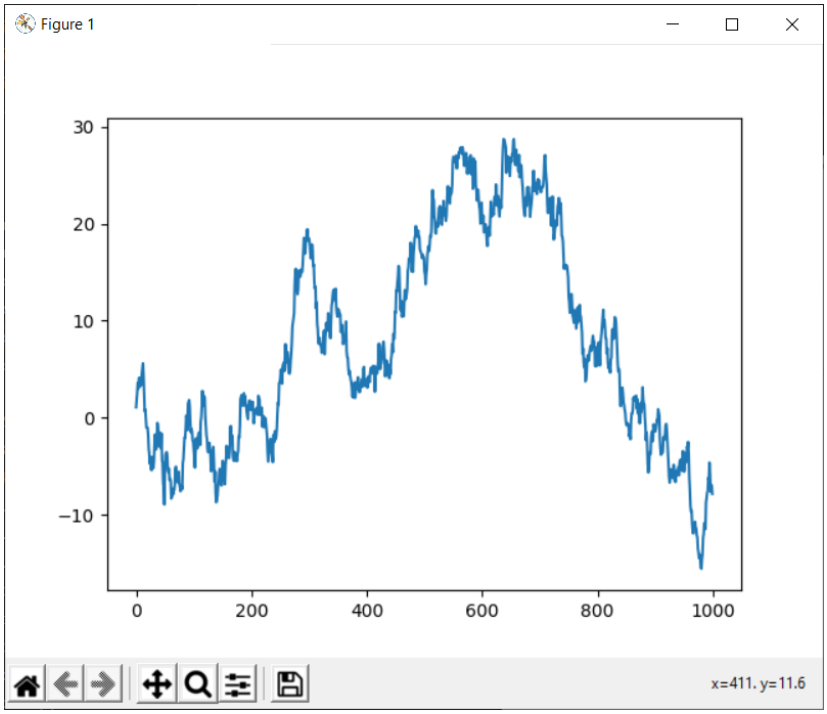
```
import pandas as pd
import matplotlib.pyplot as plt

data = [['E001', 'M', 34, 123, 'Normal', 350],
                ['E002', 'F', 40, 114, 'Overweight', 450],
                ['E003', 'F', 37, 135, 'Obesity', 169],
                ['E004', 'M', 30, 139, 'Underweight', 189],
                ['E005', 'F', 44, 117, 'Underweight', 183],
                ['E006', 'M', 36, 121, 'Normal', 80],
                ['E007', 'M', 32, 133, 'Obesity', 166],
                ['E008', 'F', 26, 140, 'Normal', 120],
                ['E009', 'M', 32, 133, 'Normal', 75],
                ['E010', 'M', 36, 133, 'Underweight', 40] ]

df = pd.DataFrame(data, columns = ['EMPID', 'Gender', 'Age', 'Sales', 'BMI', 'Income'] )

df.hist()
plt.show()
```

Output:

7. **Code**:

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

values=np.cumsum(np.random.randn(1000,1))

plt.plot(values)
plt.show()
```

Output:

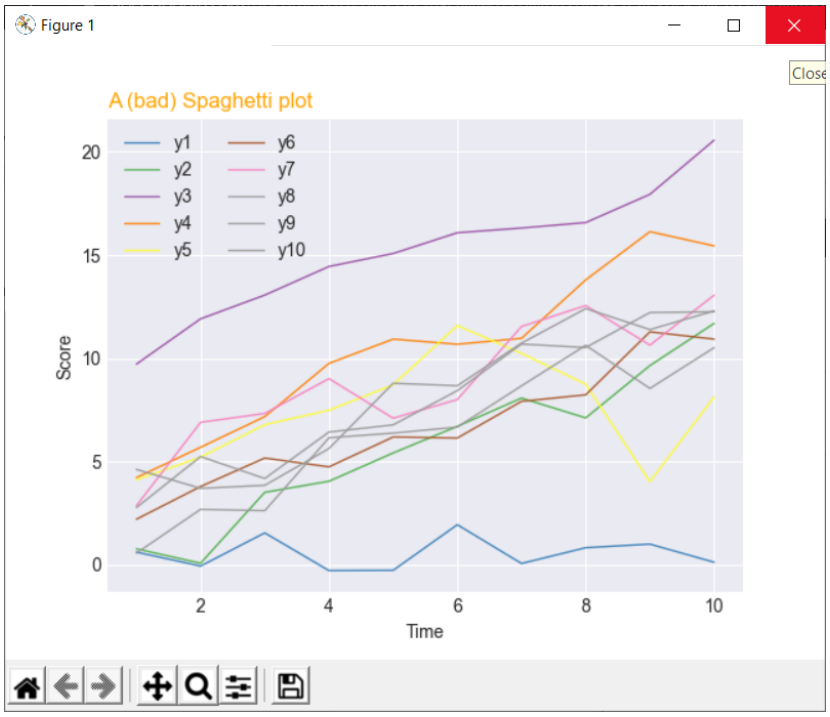


8. **Code**:

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

df=pd.DataFrame({'x': range(1,11), 'y1': np.random.randn(10), 'y2':
np.random.randn(10)+range(1,11), 'y3': np.random.randn(10)+range(11,21), 'y4':
np.random.randn(10)+range(6,16), 'y5':
np.random.randn(10)+range(4,14)+(0,0,0,0,0,0,0,-3,-8,-6), 'y6':
np.random.randn(10)+range(2,12), 'y7': np.random.randn(10)+range(5,15), 'y8':
np.random.randn(10)+range(4,14), 'y9': np.random.randn(10)+range(4,14), 'y10':
np.random.randn(10)+range(2,12) })

plt.style.use('seaborn-darkgrid')

palette = plt.get_cmap('Set1')

num=0
for column in df.drop('x', axis=1):
        num+=1
        plt.plot(df['x'], df[column], marker='', color=palette(num), linewidth=1,
        alpha=0.9, label=column)

plt.legend(loc=2, ncol=2)
```

```
plt.title("A (bad) Spaghetti plot", loc='left', fontsize=12, fontweight=0, color='orange')
plt.xlabel("Time")
plt.ylabel("Score")

plt.show()
```

Output:

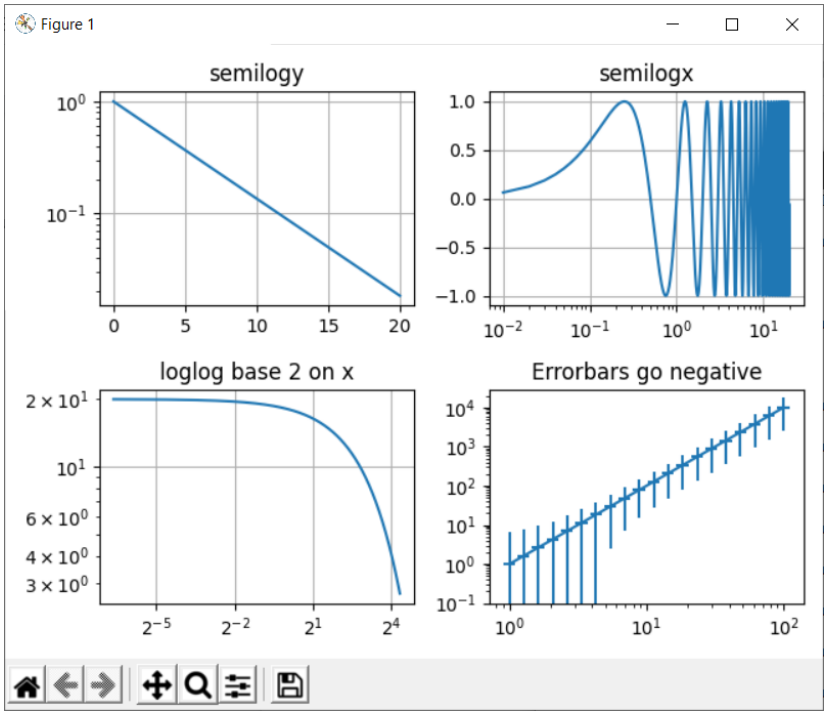


9. **Code**:

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

t = np.arange(0.01, 20.0, 0.01)

fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)

ax1.semilogy(t, np.exp(-t / 5.0))
ax1.set(title='semilogy')
ax1.grid()

ax2.semilogx(t, np.sin(2 * np.pi * t))
ax2.set(title='semilogx')
ax2.grid()

ax3.loglog(t, 20 * np.exp(-t / 10.0))
```

```
ax3.set_xscale('log', base=2)
ax3.set(title='loglog base 2 on x')
ax3.grid()

x = 10.0**np.linspace(0.0, 2.0, 20)
y = x**2.0

ax4.set_xscale("log", nonpositive='clip')
ax4.set_yscale("log", nonpositive='clip')
ax4.set(title='Errorbars go negative')
ax4.errorbar(x, y, xerr=0.1 * x, yerr=5.0 + 0.75 * y)

ax4.set_ylim(bottom=0.1)

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

Output:

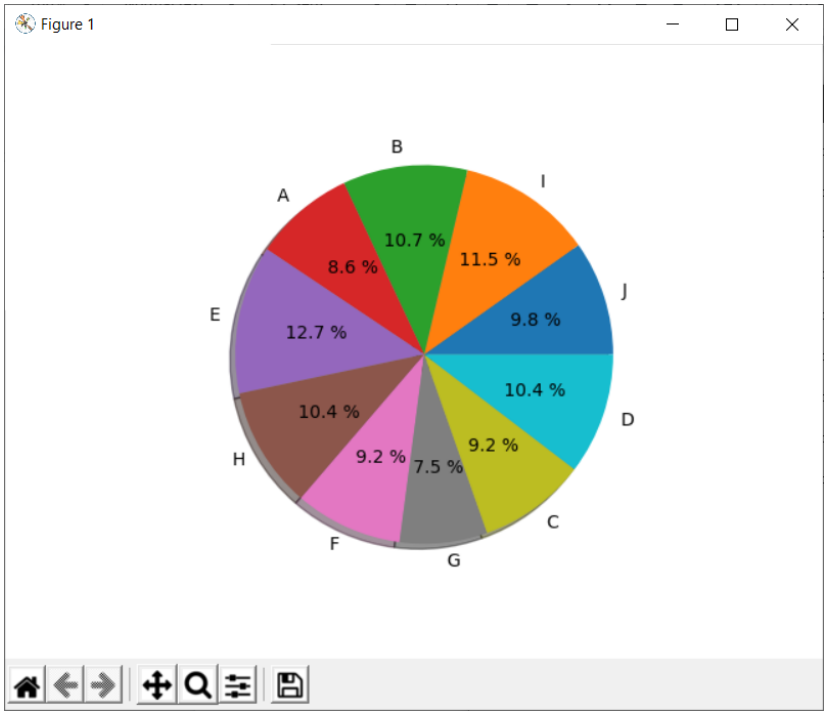


10. **Code**:

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

data = [['E001', 'M', 34, 123, 'Normal', 350],
        ['E002', 'F', 40, 114, 'Overweight', 450],
```

```
['E003', 'F', 37, 135, 'Obesity', 169],
['E004', 'M', 30, 139, 'Underweight', 189],
['E005', 'F', 44, 117, 'Underweight', 183],
['E006', 'M', 36, 121, 'Normal', 80],
['E007', 'M', 32, 133, 'Obesity', 166],
['E008', 'F', 26, 140, 'Normal', 120],
['E009', 'M', 32, 133, 'Normal', 75],
['E010', 'M', 36, 133, 'Underweight', 40] ]

df = pd.DataFrame(data, columns = ['EMPID', 'Gender', 'Age', 'Sales', 'BMI', 'Income'] )

plt.pie(df['Age'], labels = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"},

autopct ='% 1.1f %%', shadow = True)
plt.show()

plt.pie(df['Income'], labels = {"A", "B", "C", "D", "E", "F", "G", "H", "I", "J"}
autopct ='% 1.1f %%', shadow = True)
plt.show()
```

Output:

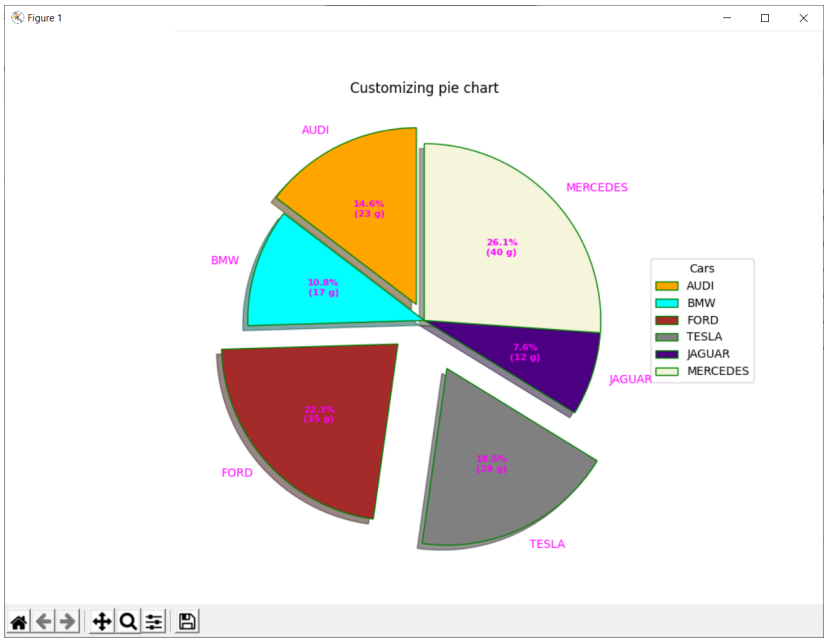


11. **Code**:

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

```
cars = ['AUDI', 'BMW', 'FORD', 'TESLA', 'JAGUAR', 'MERCEDES']

data = [23, 17, 35, 29, 12, 41]

explode = (0.1, 0.0, 0.2, 0.3, 0.0, 0.0)

colors = ( "orange", "cyan", "brown", "grey", "indigo", "beige")

wp = { 'linewidth' : 1, 'edgecolor' : "green" }

def func(pct, allvalues):
        absolute = int(pct / 100.*np.sum(allvalues))
        return "{:.1f}%\n({:d} g)".format(pct, absolute)

fig, ax = plt.subplots(figsize =(10, 7))
wedges, texts, autotexts = ax.pie(data,
                                  autopct = lambda pct: func(pct, data),
                                  explode = explode,
                                  labels = cars,
                                  shadow = True,
                                  colors = colors,
                                  startangle = 90,
                                  wedgeprops = wp,
                                  textprops = dict(color ="magenta"))

ax.legend(wedges, cars, title ="Cars", loc ="center left", bbox_to_anchor =(1, 0, 0.5, 1))

plt.setp(autotexts, size = 8, weight ="bold")
ax.set_title("Customizing pie chart")

plt.show()
```

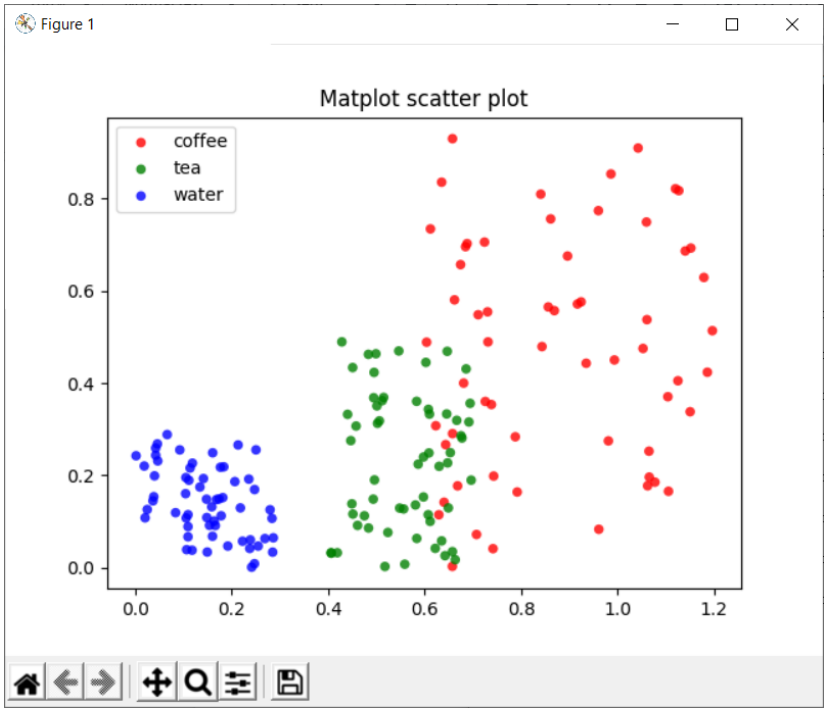
Output:



12. **Code**:

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

N = 60
g1 = (0.6 + 0.6 * np.random.rand(N), np.random.rand(N))
g2 = (0.4+0.3 * np.random.rand(N), 0.5*np.random.rand(N))
g3 = (0.3*np.random.rand(N),0.3*np.random.rand(N))

data = (g1, g2, g3)
colors = ("red", "green", "blue")
groups = ("coffee", "tea", "water")

fig = plt.figure()
ax = fig.add_subplot(1, 1, 1)

for data, color, group in zip(data, colors, groups):
        x, y = data
        ax.scatter(x, y, alpha=0.8, c=color, edgecolors='none', s=30, label=group)

plt.title('Matplot scatter plot')
plt.legend(loc=2)
plt.show()
```

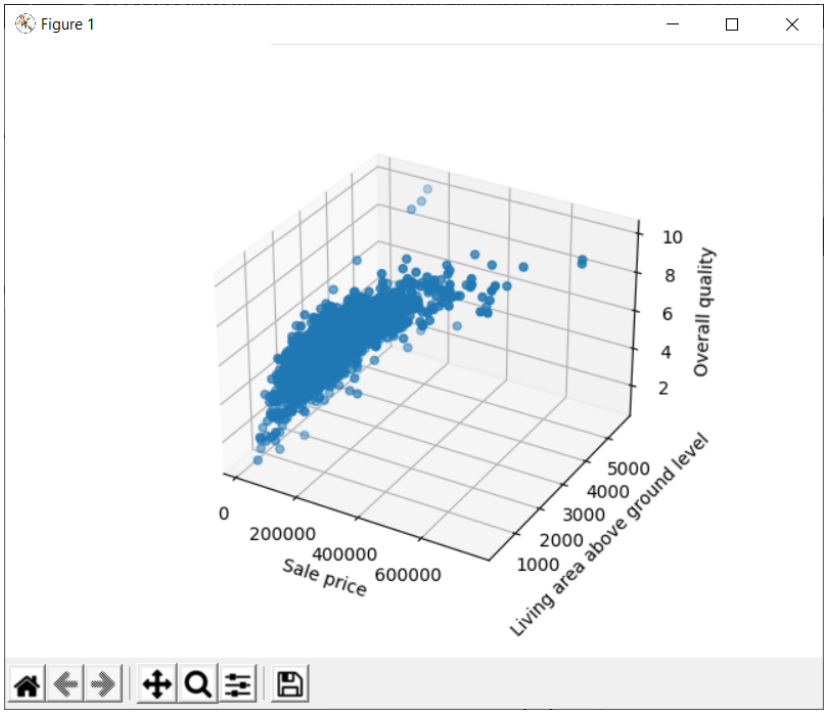
Output:



13. **Code**:

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

df = pd.read_csv('AmesHousing.csv')

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

x = df['SalePrice']
y = df['Gr Liv Area']
z = df['Overall Qual']

ax.scatter(x, y, z)
ax.set_xlabel("Sale price")
ax.set_ylabel("Living area above ground level")
ax.set_zlabel("Overall quality")

plt.show()
```

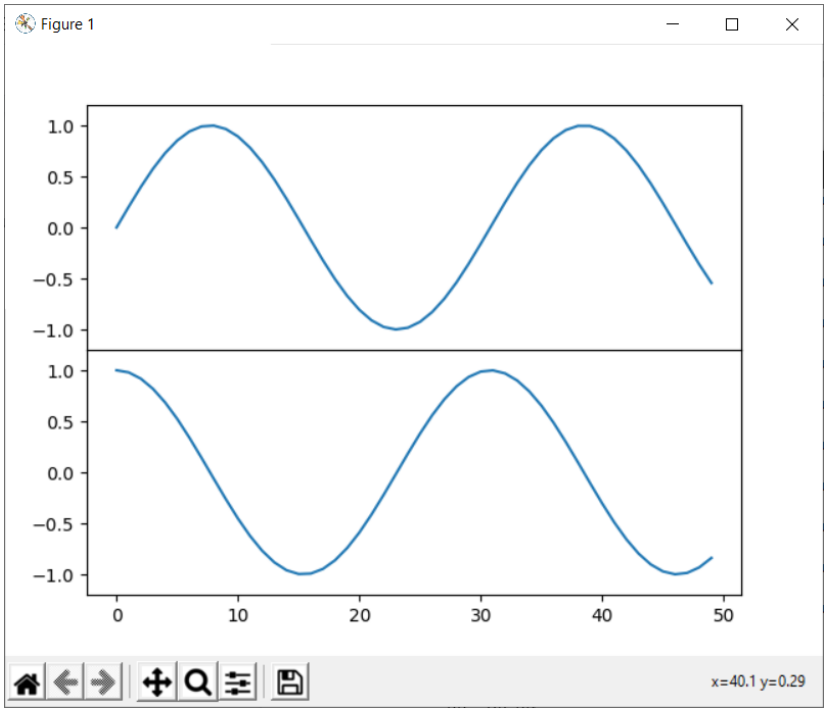
Output:



14. **Code**:

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

fig = plt.figure()
ax1 = fig.add_axes([0.1, 0.5, 0.8, 0.4], xticklabels=[], ylim=(-1.2, 1.2))
ax2 = fig.add_axes([0.1, 0.1, 0.8, 0.4], ylim=(-1.2, 1.2))

x = np.linspace(0, 10)
ax1.plot(np.sin(x))
ax2.plot(np.cos(x))

plt.show()
```

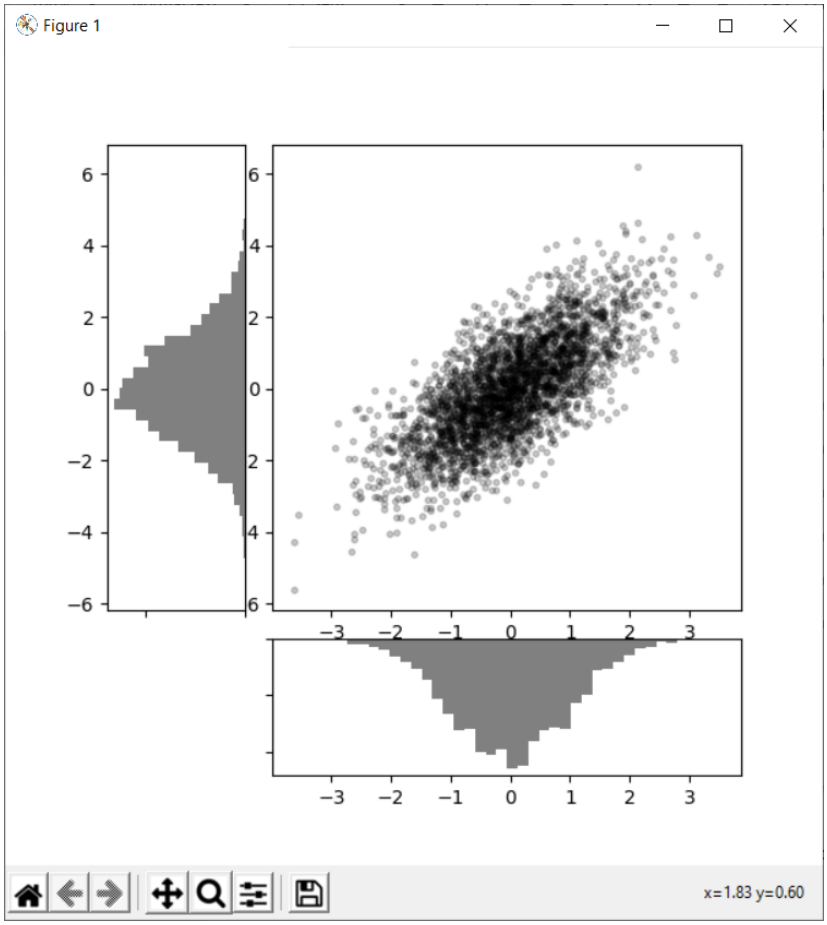
Output:



15. **Code**:

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

mean = [0, 0]
cov = [[1, 1], [1, 2]]
x, y = np.random.multivariate_normal(mean, cov, 3000).T

fig = plt.figure(figsize=(6, 6))
grid = plt.GridSpec(4, 4, hspace=0.2, wspace=0.2)
main_ax = fig.add_subplot(grid[:-1, 1:])
y_hist = fig.add_subplot(grid[:-1, 0], xticklabels=[], sharey=main_ax)
x_hist = fig.add_subplot(grid[-1, 1:], yticklabels=[], sharex=main_ax)

main_ax.plot(x, y, 'ok', markersize=3, alpha=0.2)

x_hist.hist(x, 40, histtype='stepfilled', orientation='vertical', color='gray')
x_hist.invert_yaxis()

y_hist.hist(y, 40, histtype='stepfilled', orientation='horizontal', color='gray')
y_hist.invert_xaxis()

plt.show()
```

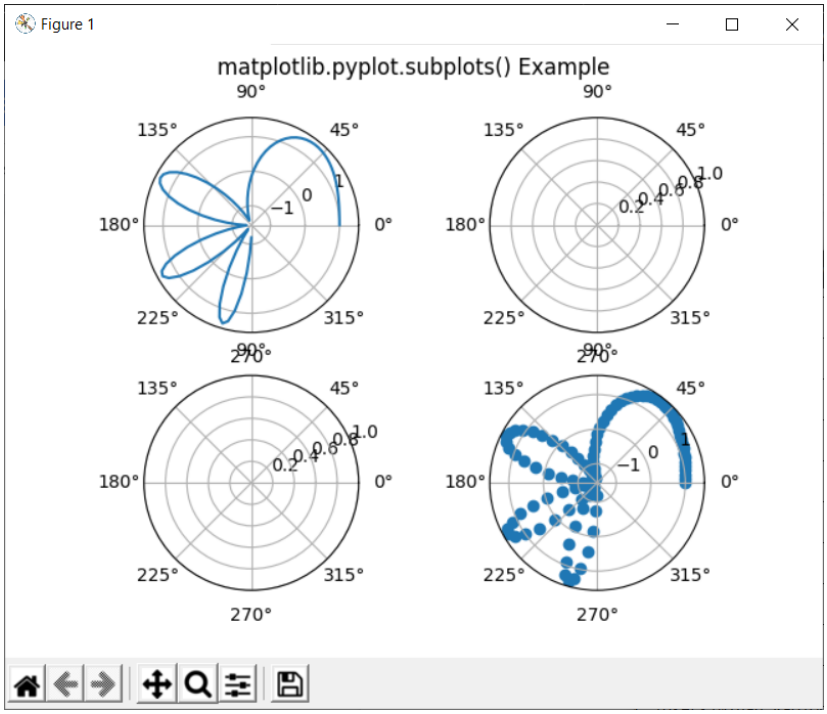
Output:



16. **Code**:

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

x = np.linspace(0, 1.5 * np.pi, 100)
y = np.sin(x**2)+np.cos(x**2)

fig, axs = plt.subplots(2, 2, subplot_kw = dict(polar = True))
axs[0, 0].plot(x, y)
axs[1, 1].scatter(x, y)

fig.suptitle('matplotlib.pyplot.subplots() Example')
plt.show()
```

Output:



Conclusion:
Hence, we have successfully understood the concept of data visualisation in Python using **numpy**, **pandas** and **matplotlib**. We studied to visualise a dataset as a **pie chart**, **bar graph**, **histogram**, **scatter plot**, **line chart** and **subplots**, along with their variations based on colour, labels, shadows.