Gallery of Graphs

This lesson discusses the variety of commonly used graphs in Python.

WE'LL COVER THE FOLLOWING

The plotting package `matplotlib` allows you to make very fancy graphs. We will discuss some of them in this lesson.

Pie Charts

A pie chart uses pie slices to show the percentage of each constituent in the whole unit. It is commonly used in scientific publications to represent data. Let's look at its implementation using `matplotlib`:

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

genre = ['Drama', 'Comedy', 'Thriller', 'Sci-Fi', 'History']
amount = [100, 25, 30, 70, 75]
pieColor = ['MediumBlue', 'SpringGreen', 'BlueViolet'];

plt.axis('equal')
plt.pie(amount, labels=genre, autopct='%1.1f%%', colors=pieColor)
```

The pie chart is created using the `plt.pie()` command. There are two main arguments of `plt.pie()`. First is the size of each section, which is stored in `amount`. Second is their corresponding labels, which is stored in `genre`. These are plotted sequentially in the counterclockwise direction.

The `autopct = '%1.1f%%'` command shows the percentages to one decimal place. To format the percentage to the two decimal places, use `autopct = '%1.2f%%'`, for three decimal places, use `autopct = '%1.3f%%'` and so on.

You may want to use `plt.axis('equal')` to make the scales along the horizontal and vertical axes equal so that the pie actually looks like a circle rather than an ellipse. We will use the `colors` keyword in your pie chart to specify a sequence of colors. The sequence must be between square brackets, each color must be between quotes preserving upper and lower cases, and they must be separated by commas. The sequence will be repeated if it is not long enough.

> 💡 Use the `help('matplotlib.pyplot.pie')` command to explore more properties.

Histograms

Histograms are another widely used 2-D graphs. Let's see its basic implementation:

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

n = np.random.randn(100000)
fig, axes = plt.subplots(figsize=(12, 8))
plt.hist(n)
axes.set_title("Histogram")
axes.set_xlim((min(n), max(n)))
```

We have generated an array of random numbers using NumPy's random library. Using the `randn` function, we are ensuring that the array has a

library. Using the `randn` function, we are ensuring that the array has a

standard normal distribution. We then plot this array using the `axes.hist()` function with the data serving as the input argument.

> 💡 Use the `help('matplotlib.pyplot.hist')` command to explore more properties.

Bar graphs

Bar graphs are easy to read representations of discrete data and `matplotlib` is used to implement it in Python. Let's see an example of it below:

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

y = np.random.randint(low=1, high=100, size=10)
x = np.arange(10)
fig, axes = plt.subplots(figsize=(12, 8))
axes.bar(x, y)
axes.set_xticks(np.arange(10))
axes.set_title("Bar Chart")
```

In line 4, we have used the `randint` from `np.random` class to generate 10 random integers between 1 and 100. Then we generated an array of the first 9 integers. We then plotted them using the `axes.bar()` function and set the ticks for the x-axis accordingly using the `set_xticks` method.

> 💡 Use the `help('matplotlib.pyplot.bar')` command to explore more properties.

Box plot

A boxplot presents information from a five-number summary. It does not show the distribution in as much detail but is useful for indicating whether a distribution is skewed and whether there are unusual observations in the

data set. Boxplots are very useful when large numbers of observations are involved and two or more data sets are being compared.

The boxplot function creates the graph and returns a lot of stuff such as *boxes*. What you see in the graph is an orange line at the median of the data. The black box spans the IQR ranging from the lower quartile (25%) to the upper quartile (75%). The whiskers are the black lines that are connected to the 50% box with black lines. Let's look at its implementation:

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

data1 = np.random.randint(low=1, high=100, size=50)
data2 = np.random.randint(low=1, high=100, size=50)
data = [data1, data2]
fig, axes = plt.subplots(figsize=(12, 8))

axes.boxplot(data)
axes.set_title("Box Plot")
```

In lines 4 and 5, we have used `randint` from `np.random` class to generate two data sets of random integers named `data1` and `data2`. We then store these data sets in the list `data` and plot it using the `axes.boxplot` command.

> 💡 Use the `help('matplotlib.pyplot.boxplot')` command to explore more properties.

Polar plot

In mathematics, the polar coordinate system is a two-dimensional coordinate system in which each point on a plane is determined by the distance from a reference point and the angle from a reference direction.

The reference point (analogous to the origin of a Cartesian system) is called the pole, and the ray from the pole in the reference direction is the polar axis. The distance from the pole is called the radial coordinate or radius, and the angle is called the angular coordinate, polar angle, or azimuth. Let's look at its implementation in Python:

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


theta = np.linspace(0, 2 * np.pi , 100)
r = np.sin(50 * theta) * np.cos(50 * theta)
fig = plt.figure()
ax = fig.add_subplot(111, polar=True)
ax.plot(theta, r);
```

In the example above, `r` is the radial coordinate and `theta` is the polar coordinate. It is important to note that in line 7, we have created an axis and set the value of `polar` to `True`. This creates a polar axis and we have then plotted `r` and `theta` using the `plot()` function.

> 💡 Use the `help('matplotlib.pyplot.plot')` command to explore more properties.

Fill between

To color the area between curves or a curve and one of the axes, we use the `fill_between` function. Let's see its implementation below:

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

x = np.linspace(0, 10, 100)
fig, axes = plt.subplots(figsize=(12, 8))
axes.fill_between(x, 2 * x**2, x**2)
```

> 💡 Use the `help('matplotlib.pyplot.fill_between')` command to explore more properties.

Scatter plot

To plot individual points without joining them, we can use scatter plots. There

To plot individual points without joining them, we can use scatter plots. There are two ways to make a scatter plot:

```
plt.plot(x, y, 's')
```

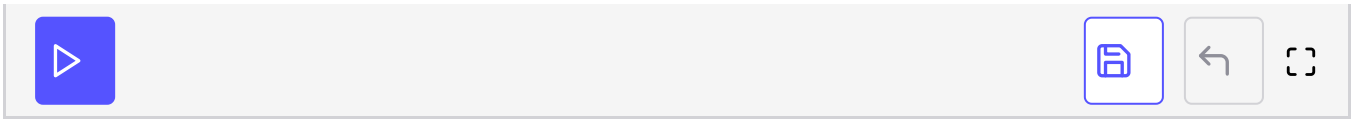
or

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

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

x = np.linspace(0, 10, 100)
y = np.random.randint(low=1, high=100, size=100)    # creates a random array of 100 intergers
                                                    # ranging from 1 - 100

fig, axes = plt.subplots(figsize=(12, 8))
axes.scatter(x, y, marker='.')
```



> 💡 Use the `help('matplotlib.pyplot.scatter')` command to explore more properties.

We will learn about 3-D plots in the next lesson.