Matplotlib

Matplotlib is a library used to create graphs, charts, and figures. It also provides functions to customize your figures by changing the colors, labels, etc.

To start using **matplotlib**, we first need to import it:

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

pyplot is the module we will be using to create our plots.

plt is a common name used for importing this module.

Matplotlib

Matplotlib works really well with **Pandas**!
To demonstrate the power of **matplotlib**, let's create a chart from dummy data.
We will create a pandas **Series** with some numbers and use it to create our chart:

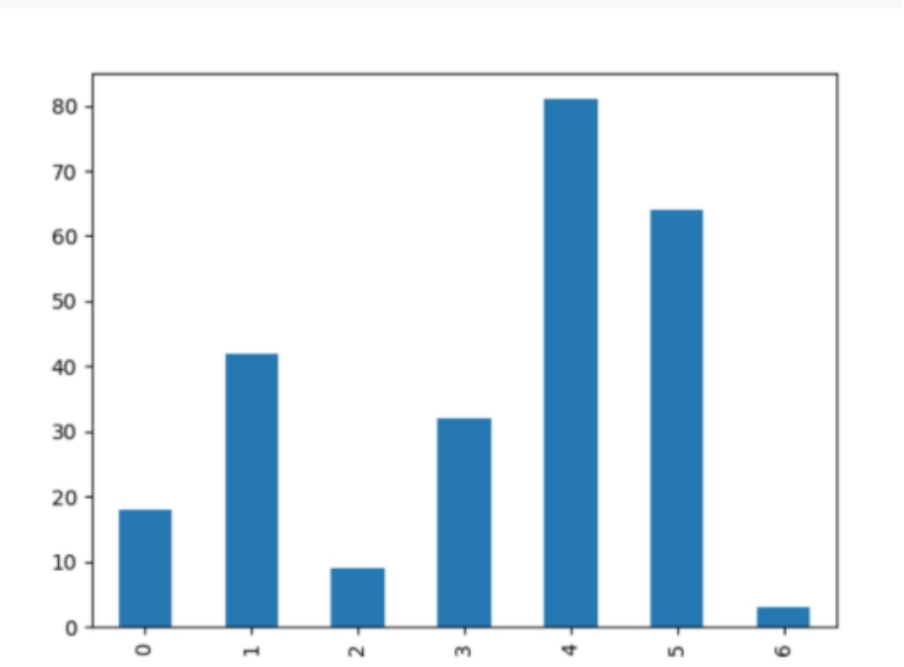
```
s = pd.Series([18, 42, 9, 32, 81, 64, 3])

s.plot(kind='bar')
plt.savefig('plot.png')
```
PY

Try it Yourself

The **.plot()** function is used to create a plot from the data in a Pandas Series or DataFrame.

Here is the result:



The data from the series is using the Y axis, while the index is plotted on the X axis.
As we have not provided a custom index for our data, the default numeric index is used.

plt.savefig('plot.png') is used to save and display the chart in our **Code Playground**.
In most environments this step is not needed, as calling the **plot**() function automatically displays the chart.

Line Plot

Matplotlib supports the creation of different chart types.
Let's start with the most basic one -- a **line chart**.
We will use the COVID-19 data from the previous module to create our charts.

Let's show the number of cases in the month of December.
To create a line chart we simply need to call the **plot**() function on our DataFrame, which contains the corresponding data:

```
df[df['month']==12]['cases'].plot()
```

PY

Try it Yourself

Run the code to see the result!

Line Plot

We can also include multiple lines in our chart.

For example, let's also include the **deaths** column in our DataFrame:

```
(df[df['month']==12])[['cases', 'deaths']].plot()
```

PY

Try it Yourself

Run the code to see the chart!

As you can see from the result, **matplotlib** automatically added a legend to show the colors of the lines for the columns.

Bar Plot

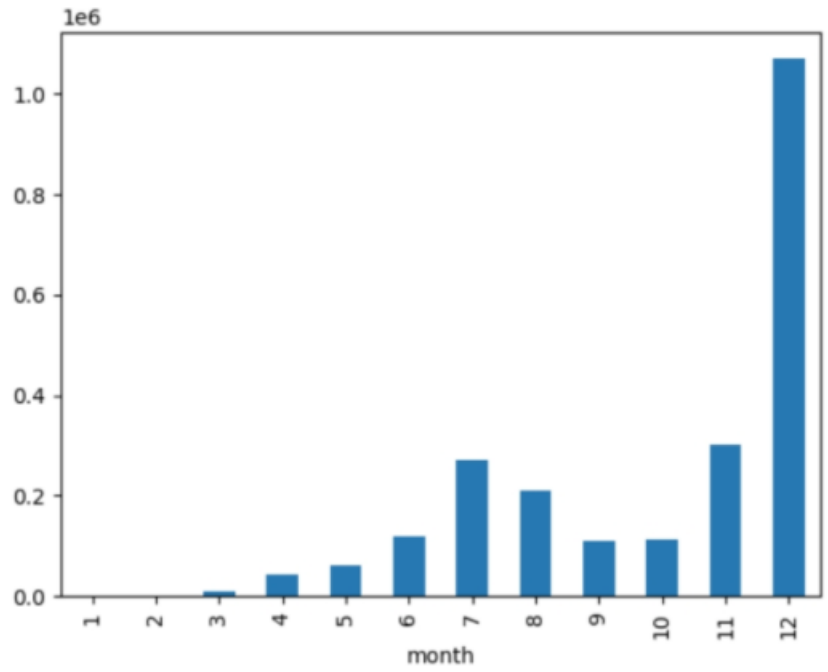
The **plot**() function can take a **kind** argument, specifying the type of the plot we want to produce.

For bar plots, provide **kind="bar"**.
Let's make a bar plot for the monthly infection cases:

```
(df.groupby('month')['cases'].sum()).plot(kind="bar")
```

PY

Try it Yourself



We first group the data by the **month** column, then calculate the sum of the cases in that month.

Bar Plot

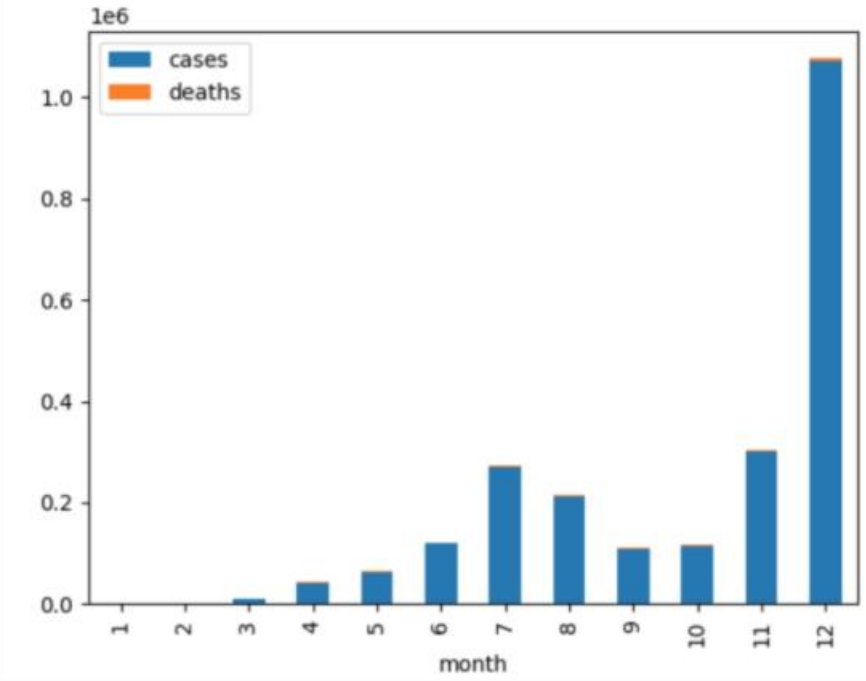
We can also plot multiple columns.
The **stacked** property can be used to specify if the bars should be stacked on top of each other.

For example:

```
df = df.groupby('month')[['cases', 'deaths']].sum()
df.plot(kind="bar", stacked=True)
```

PY

Try it Yourself



We have stacked the cases and deaths for each month.

kind="barh" can be used to create a horizontal bar chart.

Box Plot

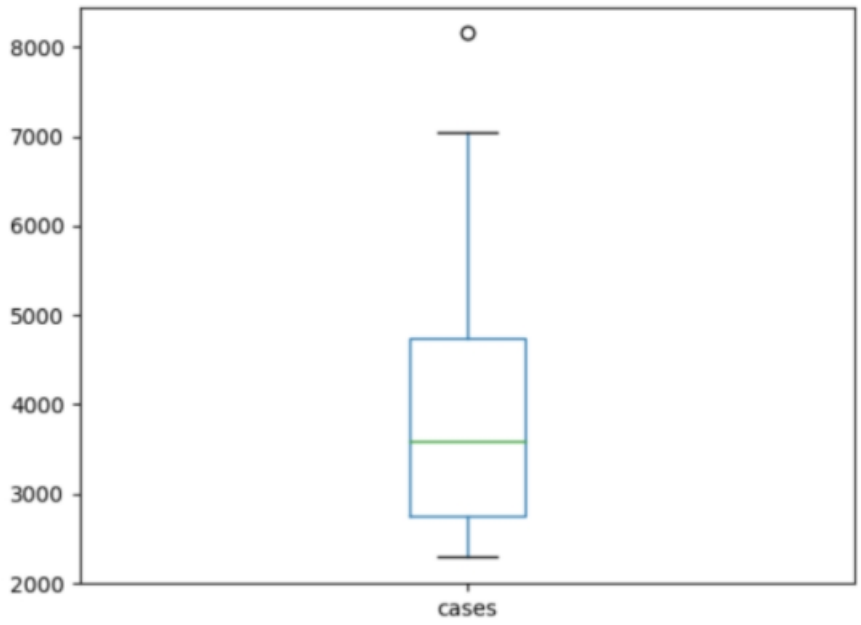
A box plot is used to visualize the distribution of values in a column, basically visualizing the result of the **describe**() function.

For example, let's create a box plot for the cases in June:

```
df[df["month"]==6]["cases"].plot(kind="box")
```

PY

Try it Yourself



The green line shows the **median** value.
The box shows the upper and lower quartiles (25% of the data is greater or less than these values).
The circles show the **outliers**, while the black lines show the min/max values excluding the outliers.

Check out the following article for more information on box plots:
Box plots

https://en.wikipedia.org/wiki/Box_plot

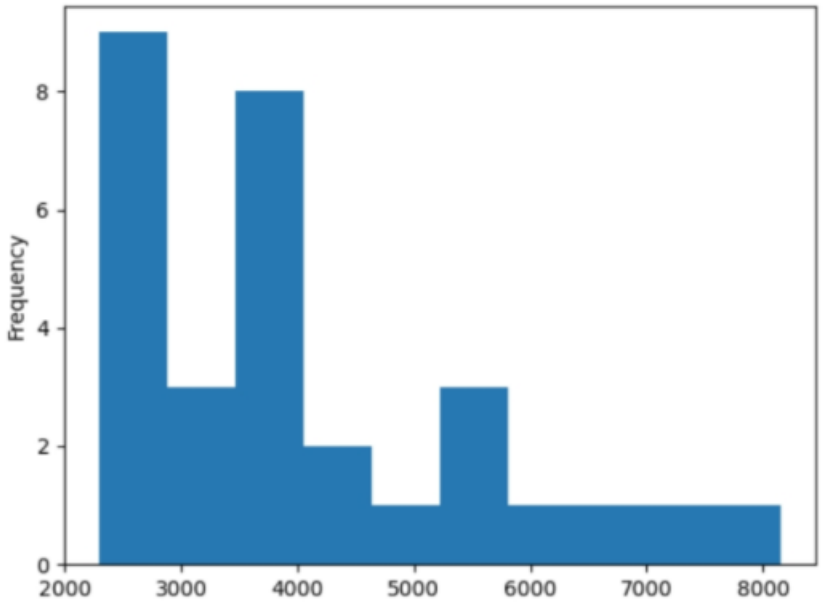
Histogram

Similar to box plots, **histograms** show the distribution of data.
Visually histograms are similar to bar charts, however, histograms display frequencies for a group of data rather than an individual data point; therefore, no spaces are present between the bars.

Typically, a histogram groups data into chunks (or bins).

For example:

```
df[df["month"]==6]["cases"].plot(kind="hist")
```
PY



The histogram grouped the data into 9 bins and shows their frequency. You can see that, for example, only single data points are greater than 6000.

You can manually specify the number of bins to use using the **bins** attribute: **plot(kind="hist", bins = 10)**

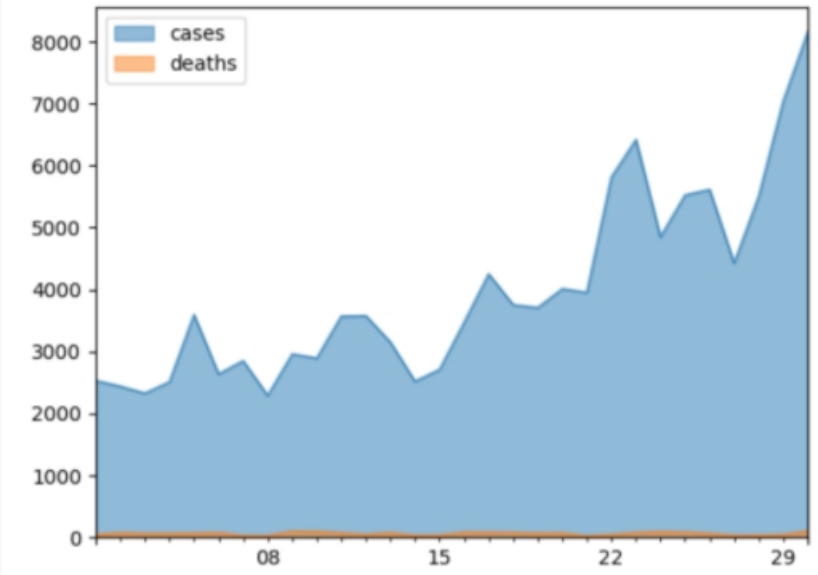
Area Plot

kind='area' creates an Area plot:

```
df[df["month"]==6][["cases",
"deaths"]].plot(kind="area", stacked=False)
```

PY

Try it Yourself



Area plots are stacked by default, which is why we provided **stacked=False** explicitly.

Scatter Plot

A **scatter plot** is used to show the relationship between two variables.

For example, we can visualize how the cases/deaths are related:

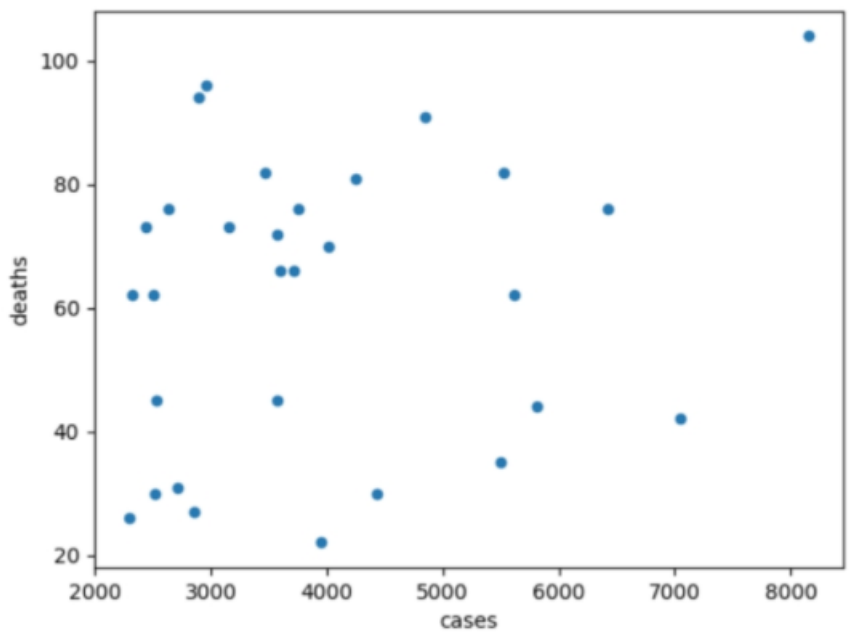
```
df[df["month"]==6][["cases",
"deaths"]].plot(kind="scatter", x='cases', y='deaths')
```

PY

Try it Yourself

We need to specify the **x** and **y** columns to be used for the plot.

We need to specify the **x** and **y** columns to be used for the plot.



The plot contains 30 points since we used the data for each day in June.

The data points look "scattered" around the graph, giving this type of data visualization its name.

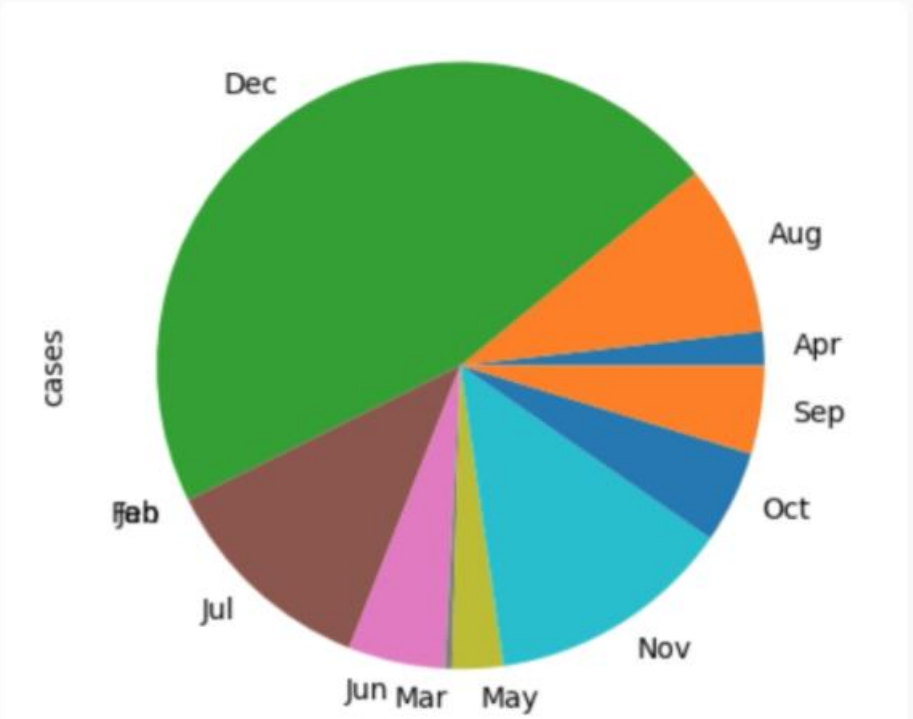
Pie Chart

We can create a pie chart using **kind="pie"**.
Let's create one for cases by month:

```
df.groupby('month')['cases'].sum().plot(kind="pie")
```

PY

Try it Yourself



Pie charts are generally used to show percentage or proportional data.

Pie charts are usually used when you have up to **6** categories.

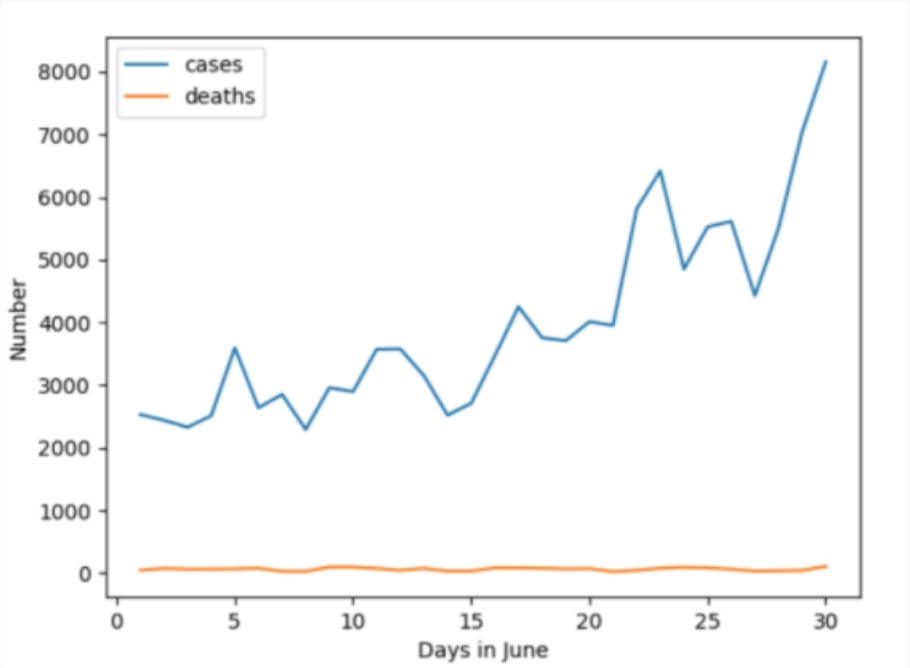
Plot Formatting

Matplotlib provides a number of arguments to customize your plot.
The **legend** argument specifies whether or not to show the legend.
You can also change the labels of the axis by setting the **xlabel** and **ylabel** arguments:

```
df[['cases', 'deaths']].plot(kind="line", legend=True)
plt.xlabel('Days in June')
plt.ylabel('Number')
```

PY

Try it Yourself



By default, pandas select the index name as **xlabel**, while leaving it empty for **ylabel**.

Plot Formatting

The **suptitle**() function can be used to set a plot title:

```
plt.suptitle("COVID-19 in June")
```

PY

Try it Yourself

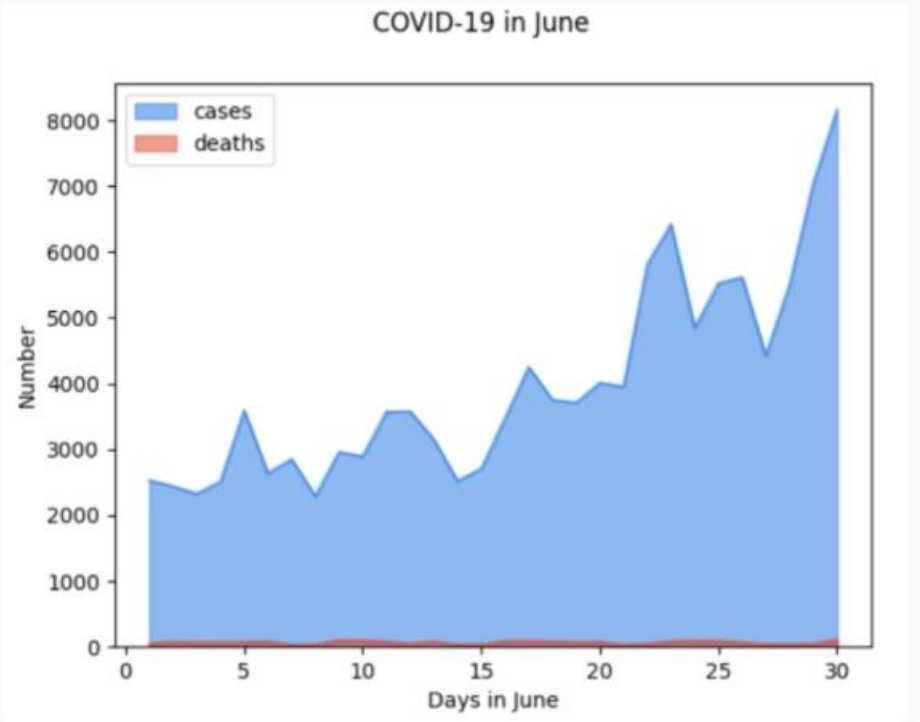
We can also change the colors used in the plot by setting the **color** attribute. It accepts a list of color hexes.

For example, let's set the cases to blue, deaths to red colors:

```
df[['cases', 'deaths']].plot(kind="area",
   legend=True,
   stacked=False,
   color=['#1970E7', '#E73E19'])
```

PY

Try it Yourself



These attributes work for almost all chart types.