Matplotlib Tutorial

José Luis Higuera Caraveo

March 20 2022

1 Matplotlib Tutorial

Have you ever thought what reports would be like without visualizations to represent and understand the data with which you work?

Reading articles would be tedious and unfriendly to readers. Visualizations are very important, in the world of Machine Learning, where few people understand how the data is being processed, it is important to communicate them in a friendly way, where anyone, even non-technical people, can understand what is being done.

The visualizations play a very important role, with them we can observe trends, even before applying any algorithm.

In this tutorial we will learn how to perform the following visualization step by step, understanding each line of code.

We will use the Matplotlib library along with its `pyplot` module.

According to the documentation, matplotlib is a library for creating static and animated displays with Python. Matplotlib makes things easy even if they seem hard. This library helps us:

- Create quality visualizations.
- Make graphs and interactive figures in which you can zoom, scroll and update them.
- Custom views.
- Export them to different formats.

And much more, all available in the Documentation.

1.1 IImporting the libraries.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

For this tutorial we will not need to import datasets, with the help of `Numpy` we will generate the necessary data for the graph.

[2]:
```
# We create 300 numbers ranging from the range -pi to pi
x = np.linspace(-np.pi, np.pi, 300, endpoint=True)
# We will use the function sine and cosine
sine, cosine = np.sin(x), np.cos(x)
```

1.2 Begining with the tutorial

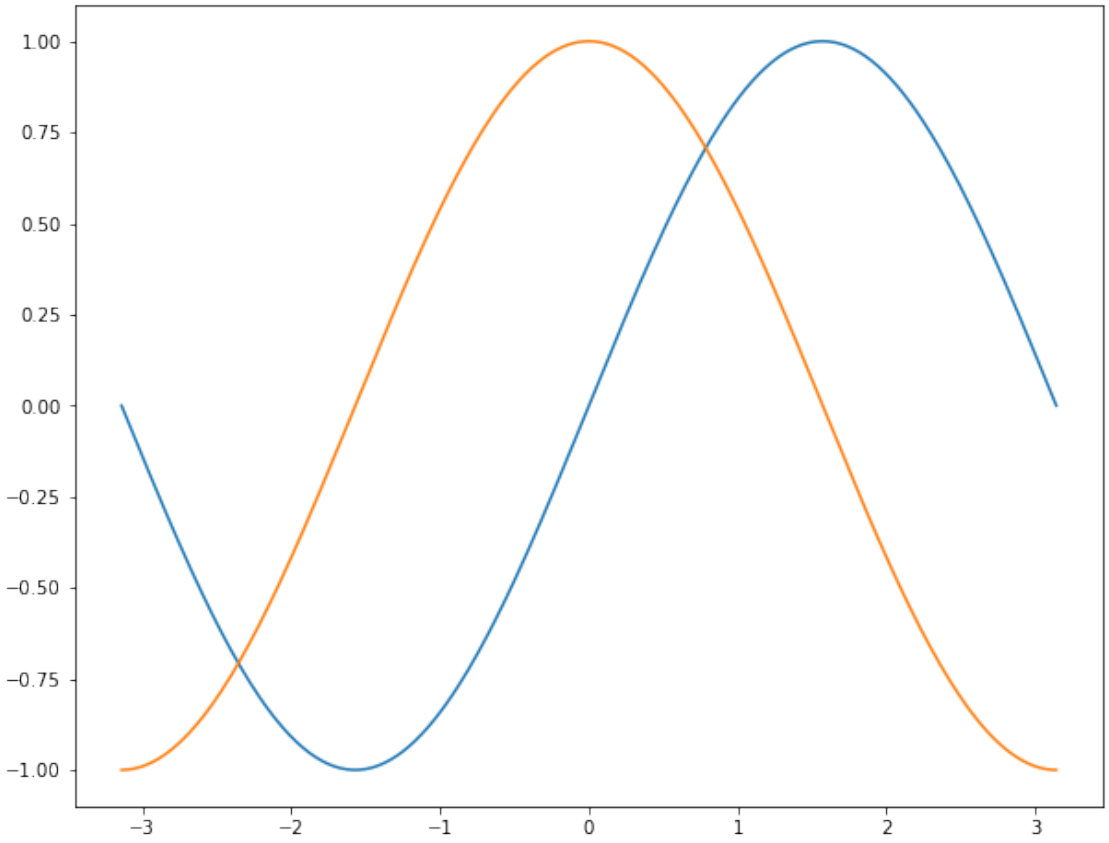
1.2.1 Understanding the figure and plot methods

In matplotlib we work with figures, to start with the tutorial, we will create the one that is going to be needed for the project. If you want to read more about it, I invite you to visit the documentation. matplotlib.pyplot.figure.

Likewise, we will create the sine and cosine plot with the data that was generated using the `plot` method. For more information you can visit matplotlib.pyplot.plot

> **Note:** We will use the `show` method to prevent matplotlib from printing a series of information related to the plot stored in memory.

[3]:
```
# Creating the figure
plt.figure(figsize=(10,8))
plt.plot(x, sine)
plt.plot(x, cosine)
plt.show()
```

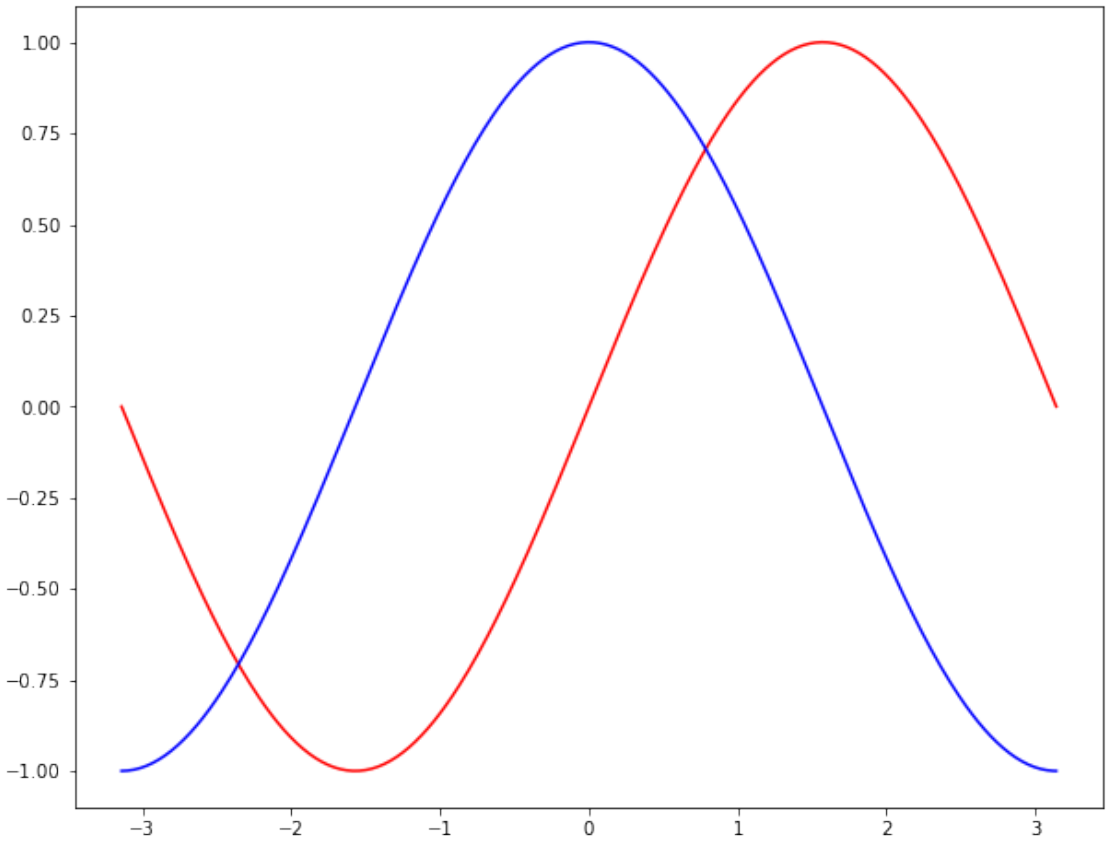


We start with some graphs that seem simple, but this is where your imagination has no limits, we can format the plots, for this matplotlib offers us a series of parameters that we can configure to our liking, such as:

- **Color:** Choose the color of the graph.
- **Linewidth:** Choose the thickness of the line.
- **Linestyle:** We can choose whether to use solid, dotted, dashed lines, among others available Here
- **Label:** Assign a name or description to the line.
- For more parameters, visit the documentation.

If we look at our example, the sine line is Red, and the cosine line is Blue, and the lines are thicker and each graph has a label assigned to it. Let's add this.

[4]:
```
plt.figure(figsize=(10,8))
plt.plot(x, sine, color='red', linewidth=1.5, linestyle='-', label='Seno')
plt.plot(x, cosine, color='blue', linewidth=1.5, linestyle='-', label='Coseno')
plt.show()
```



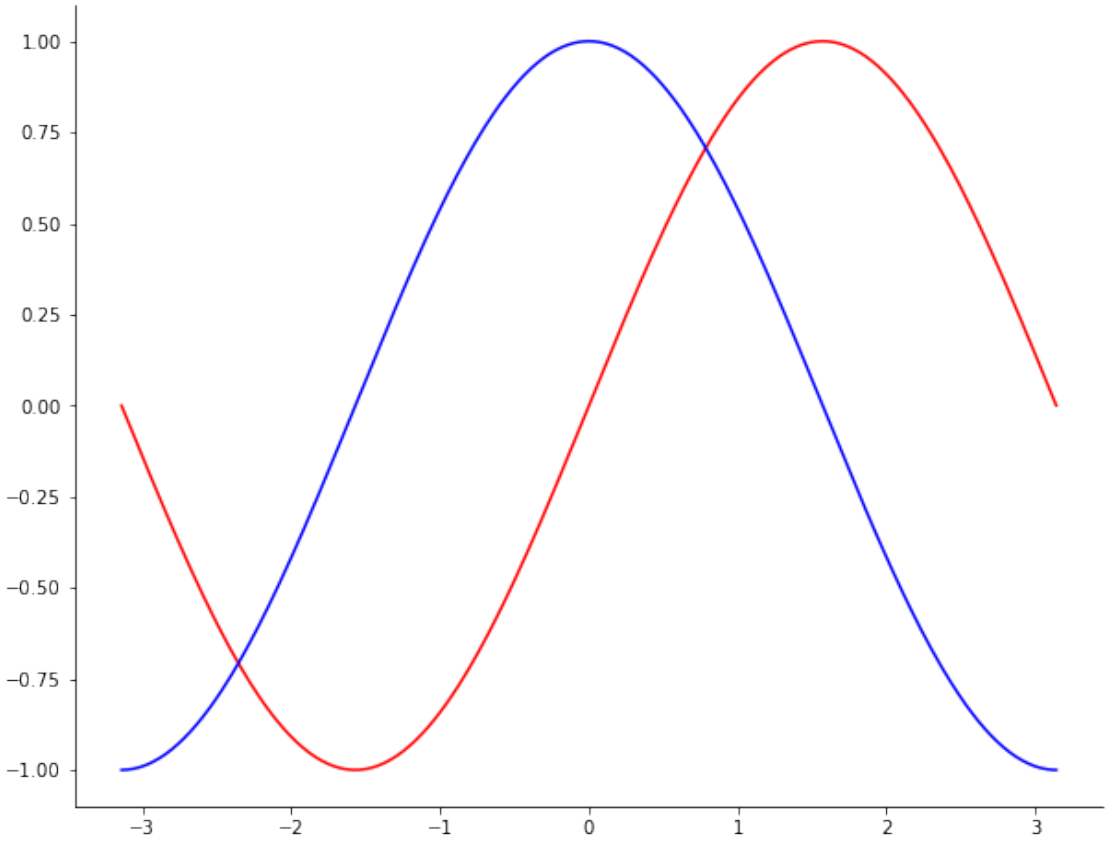
1.2.2 Working with the axes.

Note that the example has no axes, we have to remove them from our graph. To do this, we can access them with the `gca` method of `pyplot`.

A figure is made up of 4 axes. Top, Bottom, Left and Right. In the example, the right axis and the one Top are not used, in addition the left and the one below are found in a different position.

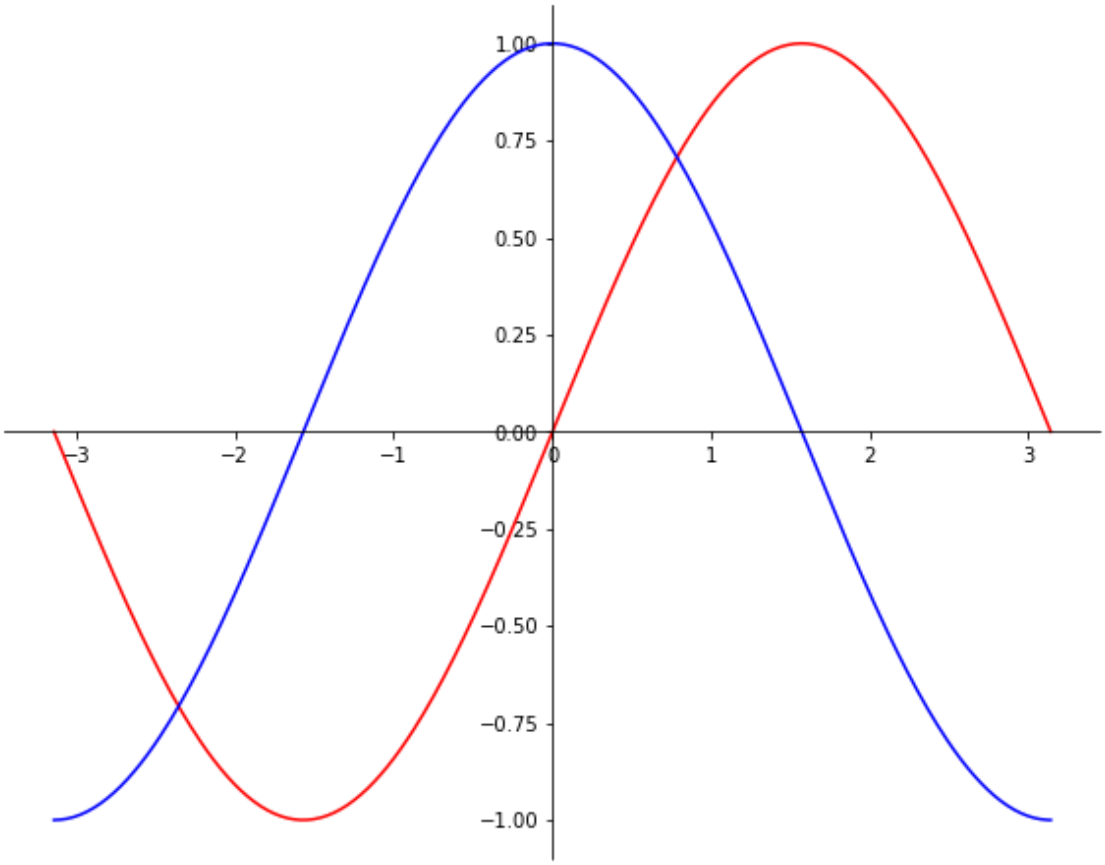
We will proceed to apply these changes with the `spines` method by accessing the axes of the figure with the `gca` method of `pyplot`. `Spines` provides us with a series of parameters that we can add using the `set` function, and the `set_color` subfunction, we will rely on this to assign the axes that we are going to remove a `set_color` none.

[6]:
```
plt.figure(figsize=(10,8))
plt.plot(x, sine, color='red', linewidth=1.5, linestyle='-', label='Seno')
plt.plot(x, cosine, color='blue', linewidth=1.5, linestyle='-', label='Coseno')

axes = plt.gca()
# Deleting the right axe
axes.spines['right'].set_color('none')
# Deleting the top axe
axes.spines['top'].set_color('none')
plt.show()
```



We have eliminated two axes, the other two if we are going to need them, but in a different position, for this, with the same `spines` method and with the `set_position` parameter we will achieve this result. Find more information Here

[7]:
```
plt.figure(figsize=(10,8))
plt.plot(x, sine, color='red', linewidth=1.5, linestyle='-', label='Seno')
plt.plot(x, cosine, color='blue', linewidth=1.5, linestyle='-', label='Coseno')
axes = plt.gca()
axes.spines['right'].set_color('none')
axes.spines['top'].set_color('none')

# We position the axis according to the data and centered on zero
axes.spines['bottom'].set_position(('data', 0))
axes.spines['left'].set_position(('data', 0))
plt.show()
```

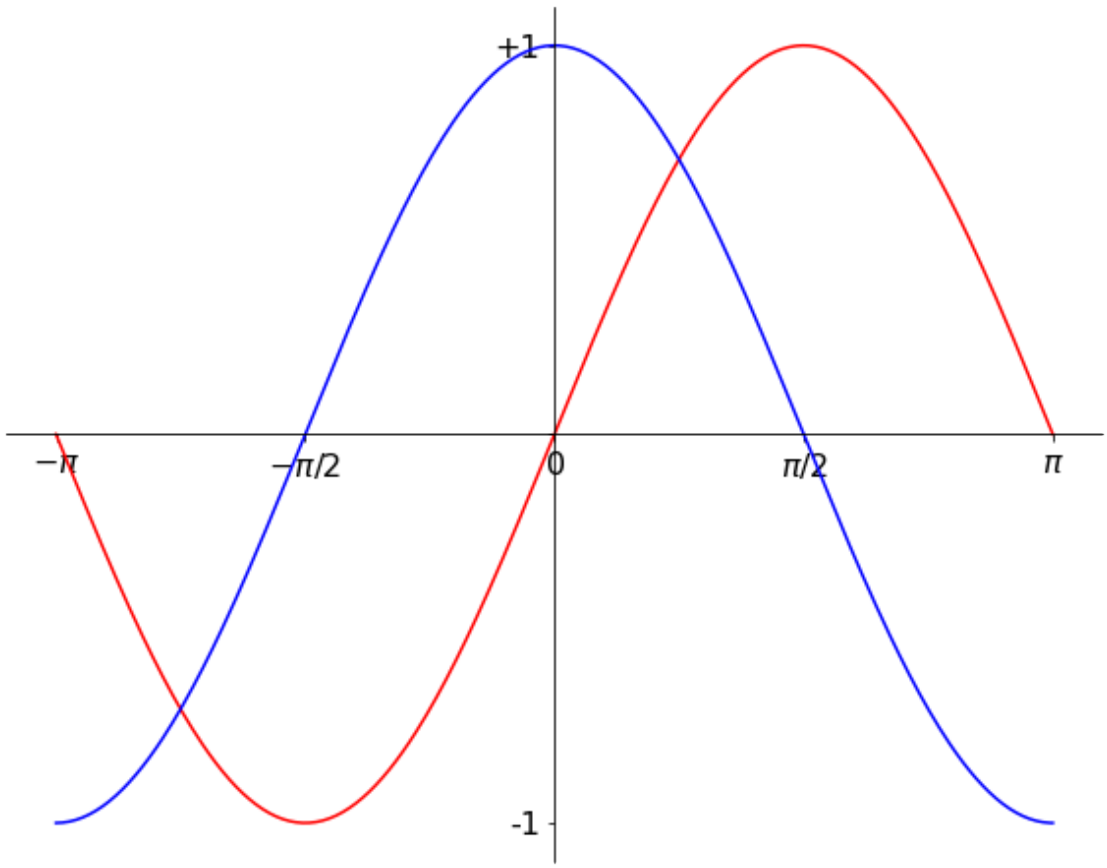


1.2.3 Changing axis data x and y

The axes data is different from the one shown in the example, we have to show them the same. The `xticks` and `yticks` method will help us.

In addition, thanks to a `for` loop we can go through that data and give it the format we want.

[8]:
```
plt.figure(figsize=(10,8))
plt.plot(x, sine, color='red', linewidth=1.5, linestyle='-', label='Seno')
plt.plot(x, cosine, color='blue', linewidth=1.5, linestyle='-', label='Coseno')
axes = plt.gca()
axes.spines['right'].set_color('none')
axes.spines['top'].set_color('none')
axes.spines['bottom'].set_position(('data', 0))
axes.spines['left'].set_position(('data', 0))

# We add the information we want to appear on the axes
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
           [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$\pi/2$', r'$\pi$'])
plt.yticks(np.linspace(-1,1,3, endpoint=True),
           ['-1', '', '+1'])

for label in axes.get_xticklabels() + axes.get_yticklabels():
    label.set_fontsize(16)

plt.show()
```

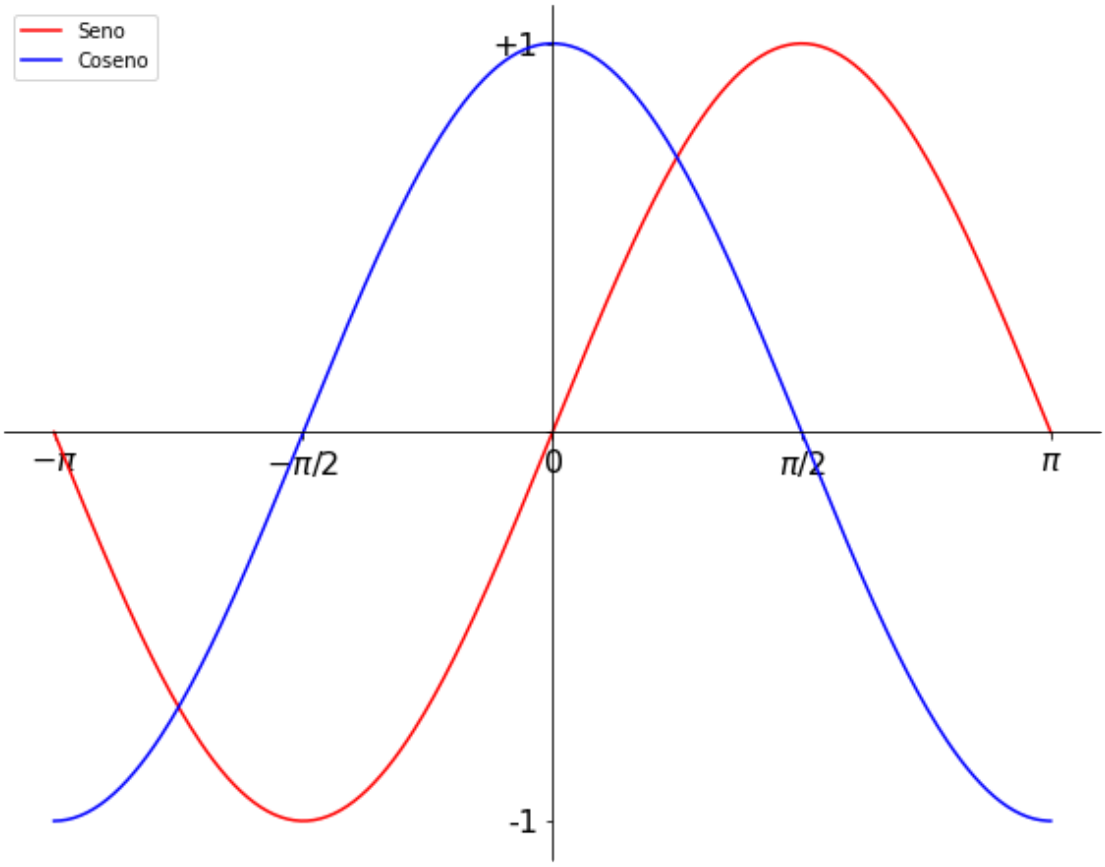


1.2.4 Visual aid, the legend method

We see that the example graph provides us with a visual aid to identify which line corresponds to which function.

To add this help we can use the `legend` method, giving it a location specified by us or simply letting matplotlib decide which is best for the example.

[9]:
```
plt.figure(figsize=(10,8))
plt.plot(x, sine, color='red', linewidth=1.5, linestyle='-', label='Seno')
plt.plot(x, cosine, color='blue', linewidth=1.5, linestyle='-', label='Coseno')
axes = plt.gca()
axes.spines['right'].set_color('none')
axes.spines['top'].set_color('none')
axes.spines['bottom'].set_position(('data', 0))
axes.spines['left'].set_position(('data', 0))
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
           [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$\pi/2$', r'$\pi$'])
plt.yticks(np.linspace(-1,1,3, endpoint=True),
           ['-1', '', '+1'])
for label in axes.get_xticklabels() + axes.get_yticklabels():
    label.set_fontsize(16)

# Adding the legend visual aid
plt.legend(loc='upper left')

plt.show()
```



1.2.5 Multigraphics, Scatter and Plot together in one figure.

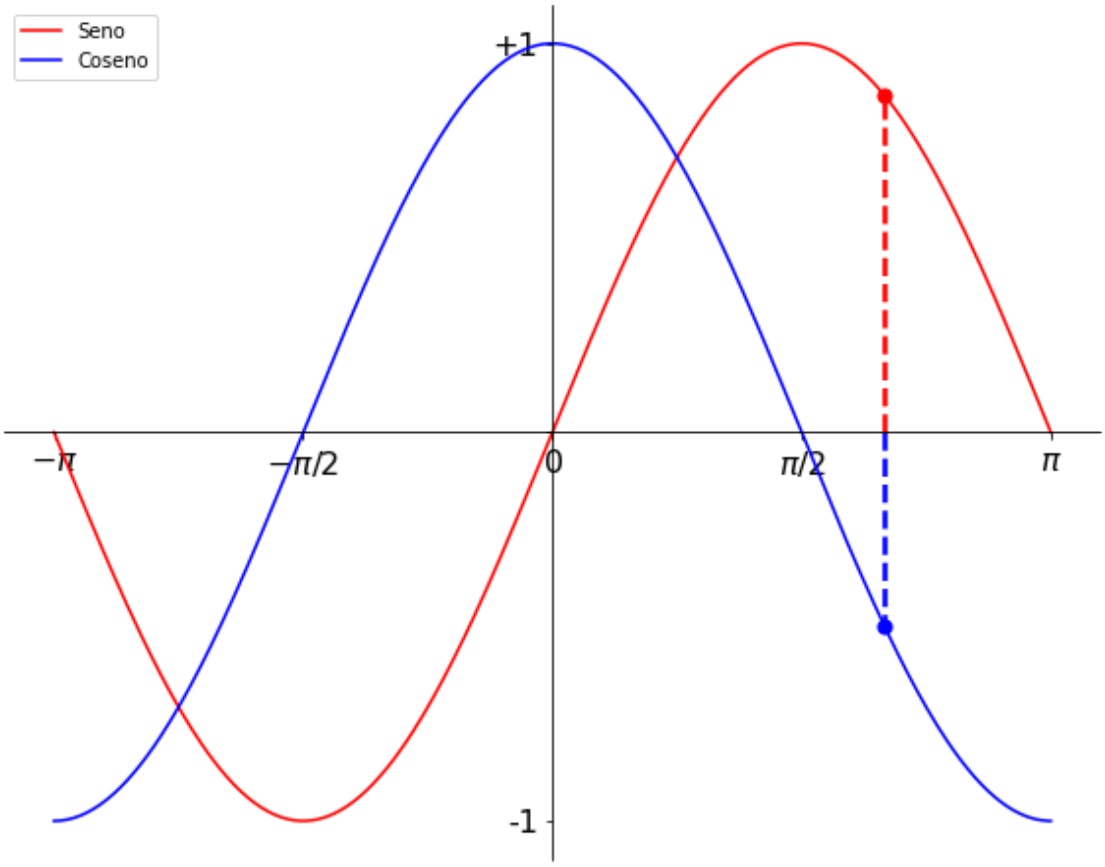
Now let's add the blue and red point that appears on the graph. We will use a chart type called `scatter`. The points are at $\frac{2}{3}$ of π in X, and at y the corresponding sine and cosine value for that $\frac{2}{3}$ of π.

The dotted lines can be added with the `plot` method used earlier to plot the graph of the sine and cosine function. The coordinates we will use are the same ones that will be used for the points added with `scatter`. We have to indicate two coordinates, point where the line starts, point where it ends. In our example:

- Starts at $(x = \frac{2\pi}{3}, y = 0)$.
- Ends in $(x = \frac{2\pi}{3}, y = sin/cos(x))$.

[9]:
```
plt.figure(figsize=(10,8))
plt.plot(x, sine, color='red', linewidth=1.5, linestyle='-', label='Seno')
plt.plot(x, cosine, color='blue', linewidth=1.5, linestyle='-', label='Coseno')
axes = plt.gca()
axes.spines['right'].set_color('none')
axes.spines['top'].set_color('none')
axes.spines['bottom'].set_position(('data', 0))
```

```
axes.spines['left'].set_position(('data', 0))
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
          [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$\pi/2$', r'$\pi$'])
plt.yticks(np.linspace(-1,1,3, endpoint=True),
          ['-1', '', '+1'])
for label in axes.get_xticklabels() + axes.get_yticklabels():
    label.set_fontsize(16)
plt.legend(loc='upper left')

# We calculate the value of 2/3 of pi
x_0 = 2*np.pi/3
# We paint the red dotted line
plt.plot([x_0,x_0], [0, np.sin(x_0)], color='red', linewidth=2.5,␣
 ↪linestyle='--')
# We paint the blue dotted line
plt.plot([x_0,x_0], [0, np.cos(x_0)], color='blue', linewidth=2.5,␣
 ↪linestyle='--')
# We paint the red point of size 50
plt.scatter(x_0, np.sin(x_0), 50, color='red')
# We paint the blue point of size 50
plt.scatter(x_0, np.cos(x_0), 50, color='blue')

plt.show()
```



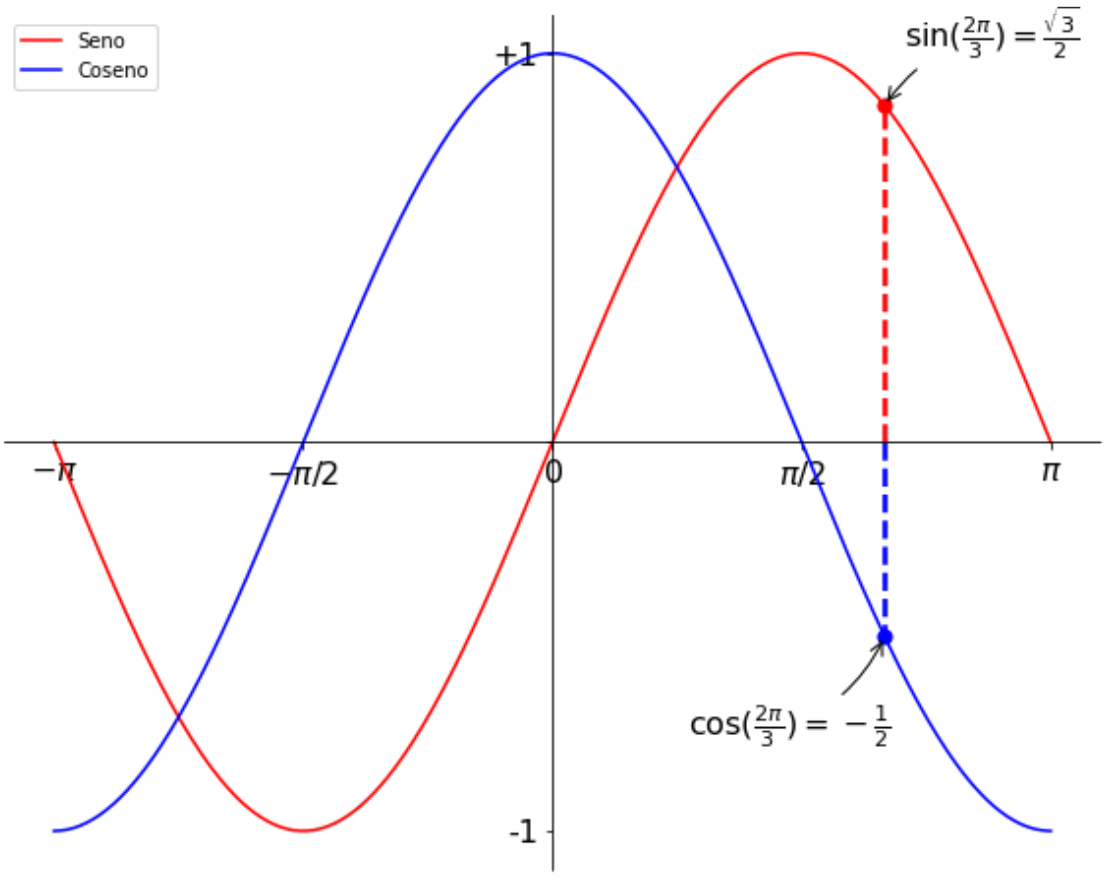
1.2.6 Add value to your visualizations by placing annotations.

We only have to add some annotations, for this we use the `annotate` method, with which, with the help of coordinates, we can indicate a certain point with an arrow.

- First parameter: Annotation.
- Second parameter: Coordinates
- Rest of parameters are configurable and according to the style that each user wants to give it.

[10]:
```
plt.figure(figsize=(10,8))
plt.plot(x, sine, color='red', linewidth=1.5, linestyle='-', label='Seno')
plt.plot(x, cosine, color='blue', linewidth=1.5, linestyle='-', label='Coseno')
axes = plt.gca()
axes.spines['right'].set_color('none')
axes.spines['top'].set_color('none')
axes.spines['bottom'].set_position(('data', 0))
axes.spines['left'].set_position(('data', 0))
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
           [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$\pi/2$', r'$\pi$'])
```

```
plt.yticks(np.linspace(-1,1,3, endpoint=True),
           ['-1', '', '+1'])
for label in axes.get_xticklabels() + axes.get_yticklabels():
    label.set_fontsize(16)
plt.legend(loc='upper left')
x_0 = 2*np.pi/3
plt.plot([x_0,x_0], [0, np.sin(x_0)], color='red', linewidth=2.5,␣
 ↪linestyle='--')
plt.plot([x_0,x_0], [0, np.cos(x_0)], color='blue', linewidth=2.5,␣
 ↪linestyle='--')
plt.scatter(x_0, np.sin(x_0), 50, color='red')
plt.scatter(x_0, np.cos(x_0), 50, color='blue')

#Annotation for the point of sine
plt.annotate(r'$\sin(\frac{2\pi}{3}) = \frac{\sqrt{3}}{2}$',
             xy = (x_0, np.sin(x_0)),
             xycoords = 'data',
             xytext = (+10,+30),
             textcoords= 'offset points',
             fontsize = 16,
             arrowprops = dict(arrowstyle = '->', connectionstyle = 'arc3,rad=.
 ↪2'))

# Annotation for the point of cosine
plt.annotate(r'$\cos(\frac{2\pi}{3}) = -\frac{1}{2}$',
             xy = (x_0, np.cos(x_0)),
             xycoords = 'data',
             xytext = (-100,-50),
             textcoords= 'offset points',
             fontsize = 16,
             arrowprops = dict(arrowstyle = '->', connectionstyle = 'arc3,rad=.
 ↪2'))

plt.show()
```



With this we have finished our tutorial, and I hope you have learned a lot about how visualizations are handled with `matplotlib`.

Finally, a tip that can be very useful, save your visualizations, you can do this with a simple line of code.

```
plt.savefig("File_Name.extension", bbox_inches='tight')
```

1.3 Keep learning

I invite you to continue investigating and investigating within the documentation, `Matplotlib` is a very powerful tool and you can take great advantage of it to visualize your data and make sense of your research related to Data Science.

Remember that a visualization says more than a thousand words and is the best means by which you can communicate your research and projects, this to facilitate understanding for technicians and non-technicians.