Matplotlib

"A picture is worth a thousand words." rings true in data science. Data **visualization** can reveal patterns that are not obvious and communicate the insights more effectively. In this part, we will take a look at the Matplotlib library, one of the most popular data visualization tools, operating on numpy arrays as well as pandas Series and DataFrames. By convention, we import the library under a shorter name:

```python
import matplotlib as mpl
```

Specifically, the module matplotlib.pyplot, a collection of command style functions that make matplotlib work like MATLAB, will be used for the rest of the course:

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

The style can be changed from classic to ggplot, mimicking the aesthetic style used in the R package ggplot2.

```python
plt.style.use('ggplot')
```

matplotlib.pyplot is a collection of functions that make plotting in python work like MATLAB. Each function makes some change to a figure, e.g., creates a figure, creates a plotting area in a figure, plots lines, annotates the plots with labels, etc. as we will see in the following lessons.

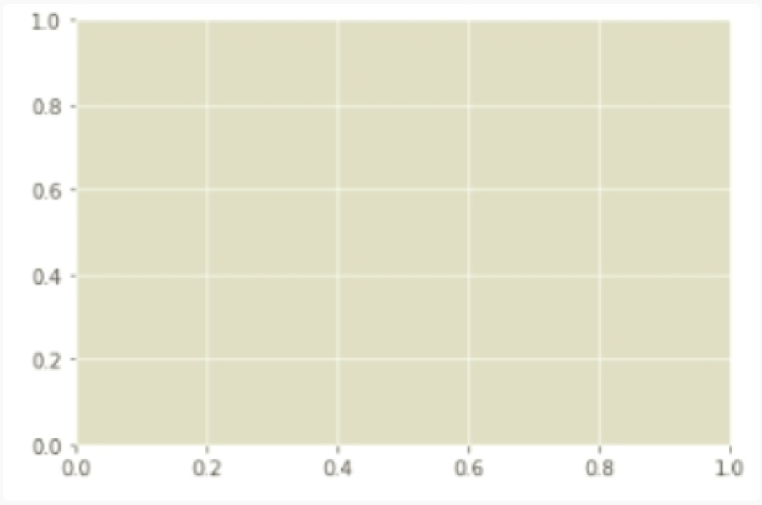
Basics

For all matplotlib plots, first create a figure and an axes object; to show the plot, call “plt.show()”. The figure contains all the objects, including axes, graphics, texts, and labels. The axes is a bounding box with ticks and labels. Think of **axes** as an individual plot.

```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes()
plt.show()
```

PY

Try it Yourself



matplotlib.pyplot provides many customization features as we will see in the following lessons.

Line Plot

Let's start with a beautiful wave function, sine function, sin(x), where x ranges from 0 to 10. We need to generate the sequence along the x-axis, an evenly spaced array, via **linspace()**

```
import numpy as np
x = np.linspace(0,10,1000) # x is a 1000 evenly spaced
numbers from 0 to 10
```

The second line generates an evenly spaced sequence of 1000 numbers o 0 to 10. You can view it as tides go up and down over time and the height of the tides are obtained by the sin function.

```
y = np.sin(x)
```

To plot, as before, we first create the figure and axes objects.

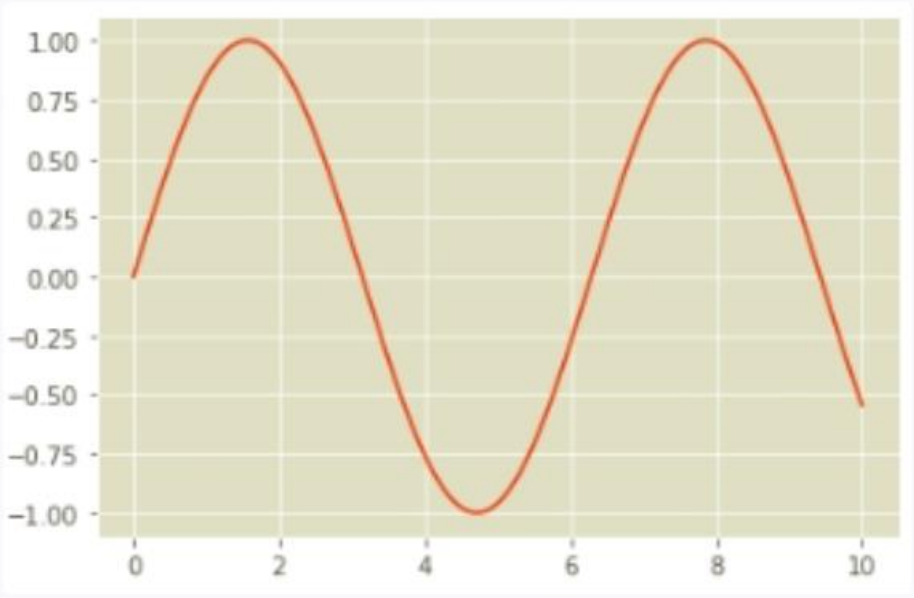
```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes()
```

We now make a plot directly from the Axes, "ax.plot()"; by default, it generates a Line2D object. To show the plot, we need to call show().

```
ax.plot(x, y)
plt.show()
```

PY

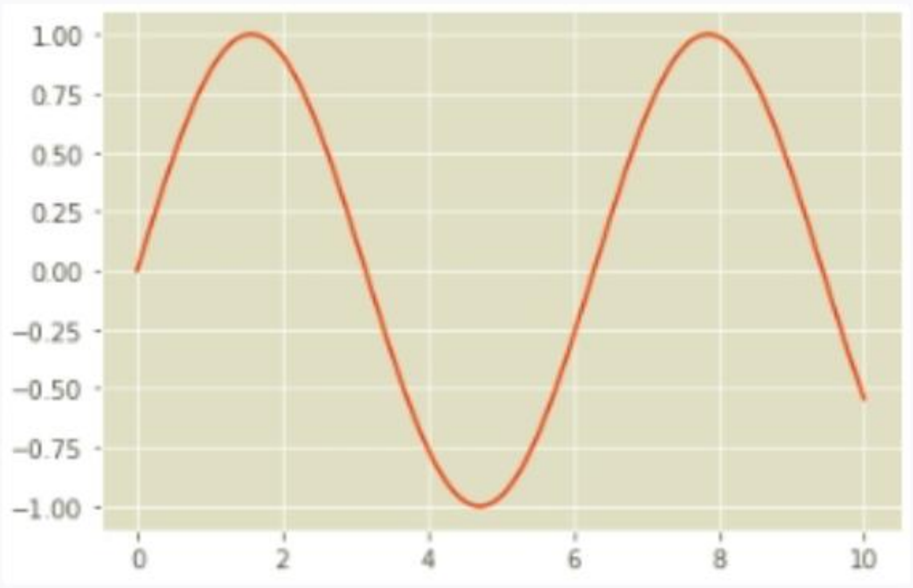
Try it Yourself



Alternatively, we can use the pylab interface and let the figure and axes be created for us in the background.

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

Try it Yourself



A line plot displays data along a number line, a useful tool to track changes over short and long periods of time.

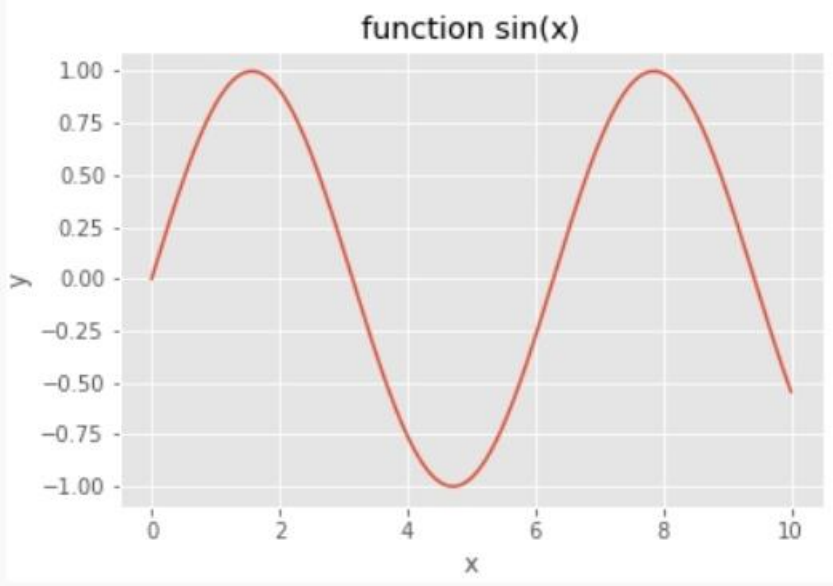
Labels and Titles

One critical component of every figure is the figure title. The job of the title is to accurately communicate what the figure is about. In addition, axes need titles, or more commonly referred to as axis labels. The axis labels explain what the plotted data values are. We can specify the x and y axis labels and a title using plt.xlabel(), plt.ylabel() and plt.title().

```
x = np.linspace(0,10,1000) # 1darray of length 1000
y = np.sin(x)

plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('function sin(x)')
plt.show()
```

Try it Yourself



One can also set the limits of x- and y-axis using plt.xlim() and plt.ylim(), respectively.

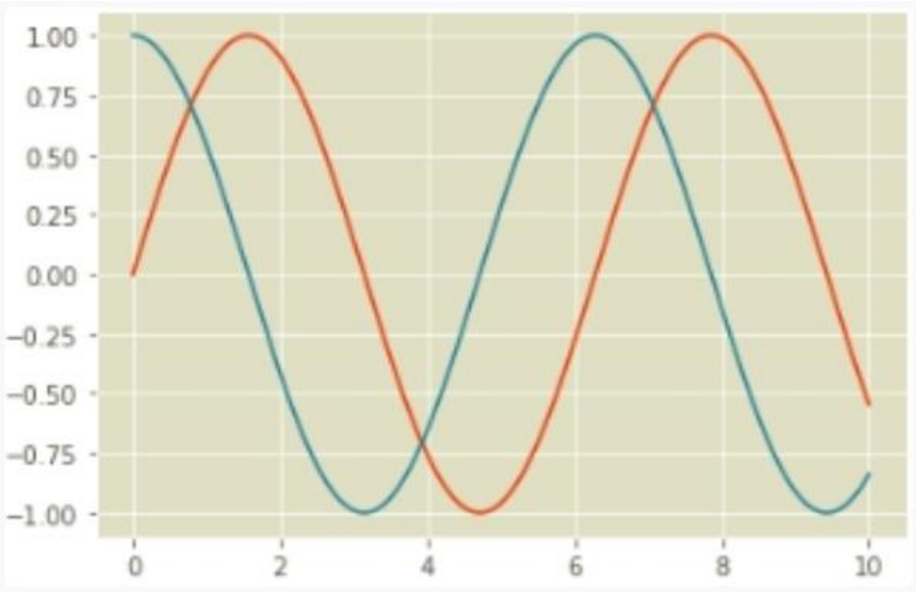
Multiple Lines

Usually there are various datasets of similar nature, and we would like to compare them and observe the differences. We can plot multiple lines on the same figure. Say, the sin function capture the tides on the east coast and cos function capture the tides on the west coast at the same time, we can plot them both on the same figure by calling the **.plot()** function multiple times.

```
x = np.linspace(0,10,1000) # 1darray of length 1000
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.show()
```

PY

Try it Yourself

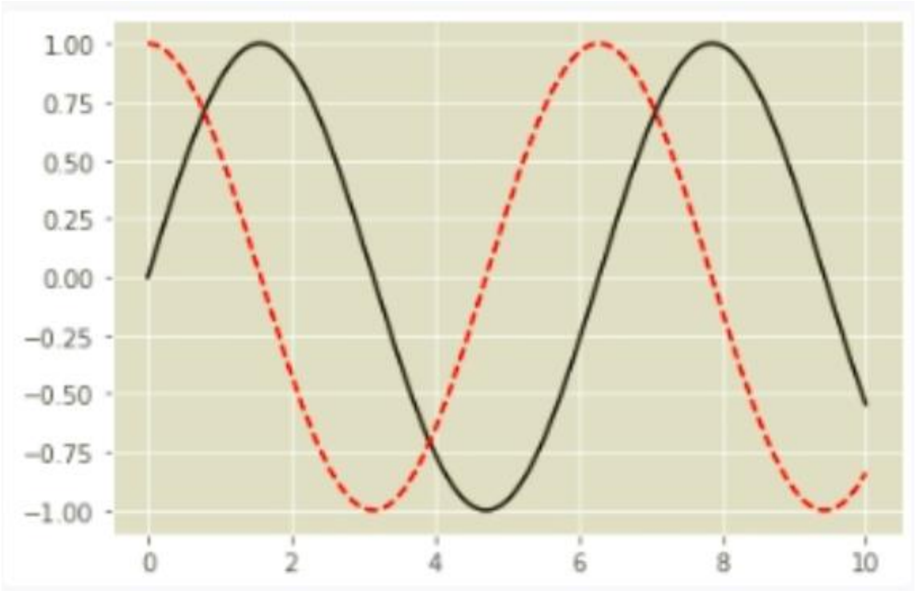


Colors and line styles can be specified to differentiate lines:

```
x = np.linspace(0,10,1000) # 1darray of length 1000
plt.plot(x, np.sin(x), color='k')
plt.plot(x, np.cos(x), color='r', linestyle ='--')
plt.show()
```

PY

Try it Yourself



Note that we specified basic colors using a single letter, that is, k for black and r for red. More examples include b for blue, g for green, c for cyan, etc. For more details on the use of colors in matplotlib, refer to its documentation.

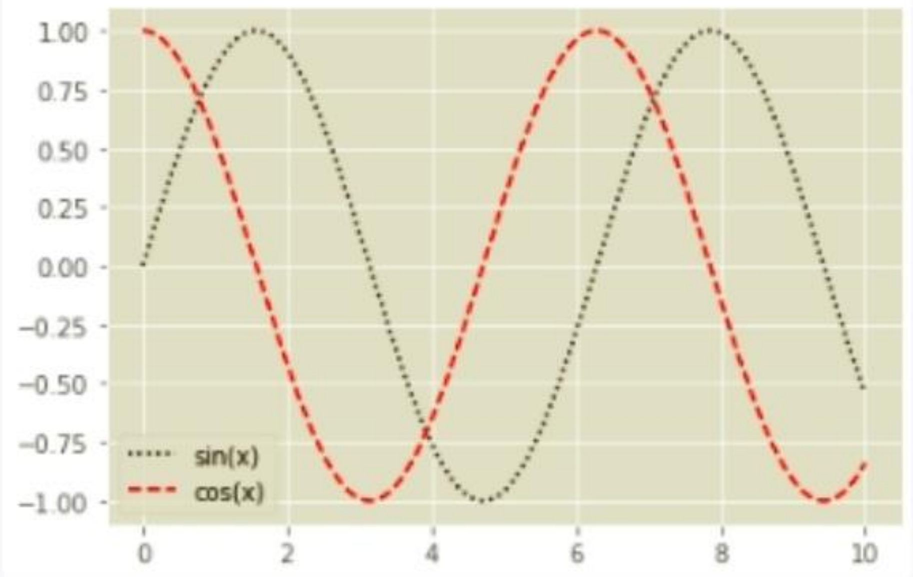
It is sometimes helpful to compare different datasets visually on the same figure and matplotlib makes it easy!

Legend

When there are multiple lines on a single axes, it's often useful to create a plot **legend** labeling each line. We can use the method plt.legend(), in conjunction with specifying labels in the plt.plot()

```
x = np.linspace(0,10,1000) # 1darray of length 1000
plt.plot(x, np.sin(x), 'k:', label='sin(x)')
plt.plot(x, np.cos(x), 'r--', label='cos(x)')
plt.legend()
plt.show()
```

Try it Yourself



Note here we use 'k:' to indicate the line of sin function to be black (indicated by k) and dotted (indicated by :). Line style and color codes can be combined into a single non-keyword argument in the plt.plot() function.

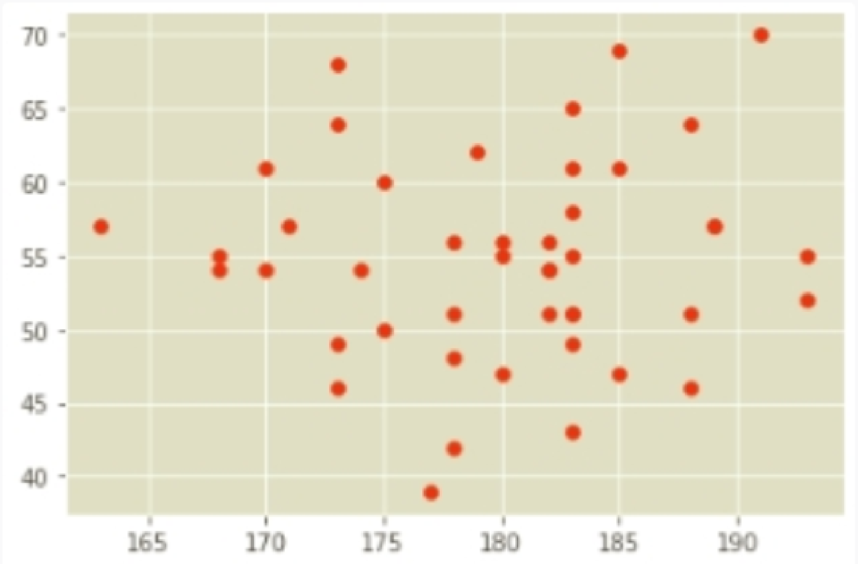
The color, line style (e.g., solid, dashed, etc.), and position, size, and style of labels can be modified using optional arguments to the function. For more details check the docstring of each function and the matplotlib documentation.

Scatter Plot

Another simple plot type is a **scatter plot**, instead of points being joined by line segments, points are represented individually with a dot, or another shape. Pass 'o' in the plt.plot(), or use plt.scatter() to show the relationship of heights and ages:

```
plt.scatter(presidents_df['height'],
presidents_df['age'])
plt.show()
```

Try it Yourself

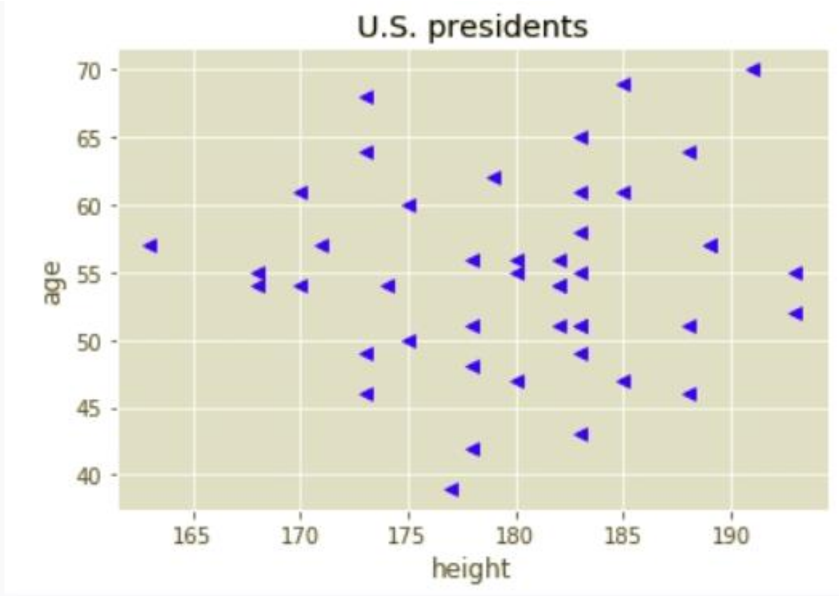


So we created a scatter plot demonstrating the correlation between heights and ages of the presidents. Note that Matplotlib infers the appropriate ranges for us from the data along both x- and y-axis.

By default, each data point is a full circle, and there is a good collection of other shapes. For example, we can pass '<' to draw a triangle pointing to the left, in addition we specify the color to blue:

```
plt.scatter(presidents_df['height'],
presidents_df['age'],
   marker='<',
   color='b')
plt.xlabel('height');
plt.ylabel('age')
plt.title('U.S. presidents')
plt.show()
```

Try it Yourself



Check the matplotlib documentation for the full list of symbols.

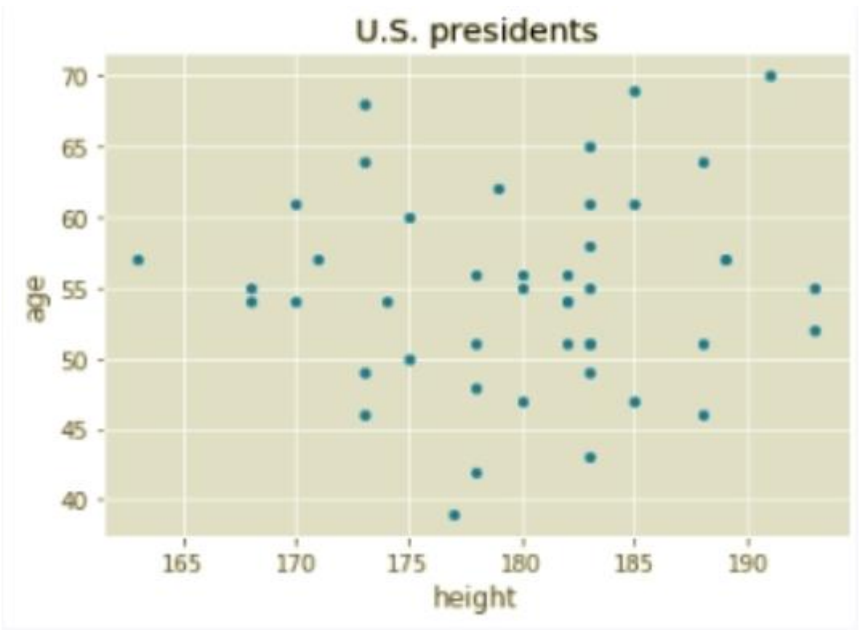
Scatter plot is a useful tool to display the relationship between two features; whereas a line plot puts more emphasis on the change between points as its slope depicts the rate of change.

Plotting with Pandas

A great thing about pandas is that it integrates well with matplotlib, so we can plot directly from DataFrames and Series. We specify the kind of plot as 'scatter', 'height' along x-axis, and 'age' along y-axis, and then give it a title:

```py
presidents_df.plot(kind = 'scatter',
  x = 'height',
  y = 'age',
  title = 'U.S. presidents')
plt.show()
```

Try it Yourself



We then created a scatter plot from the DataFrame, with both axis labels and title supplied.

As we specified the x-axis and y-axis with column names from the DataFrame, the labels were also annotated on the axes.

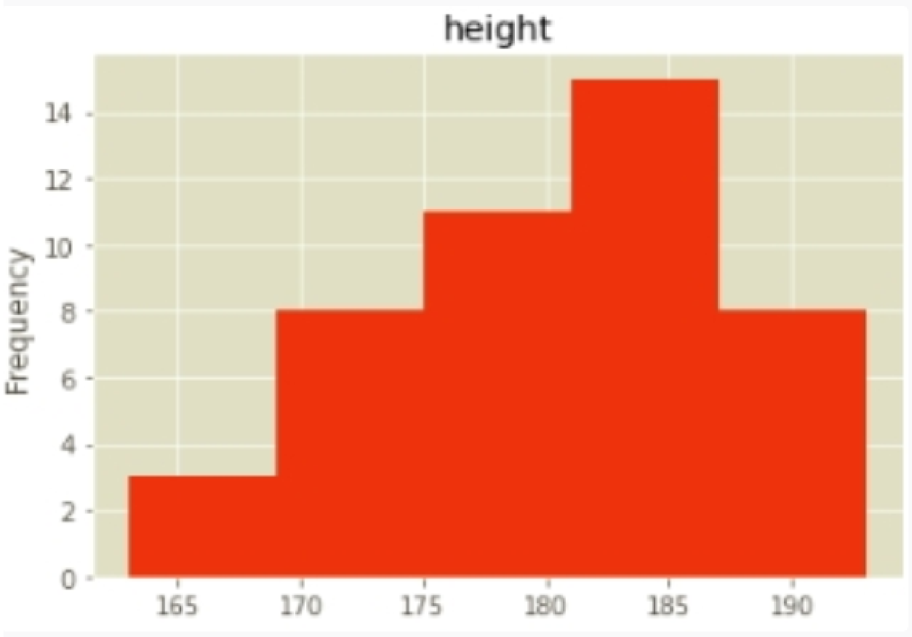
Histogram

A **histogram** is a diagram that consists of rectangles with width equal to the interval and area proportional to the frequency of a variable. For example, in the histogram below, there are five bins with equal length (i.e., [163, 169), [169, 175), [175, 181), [181, 187), and [187, 193)), and 3 presidents whose height is between 163 cm and 169 cm.

```
presidents_df['height'].plot(kind='hist',
  title = 'height',
  bins=5)
plt.show()
```

PY

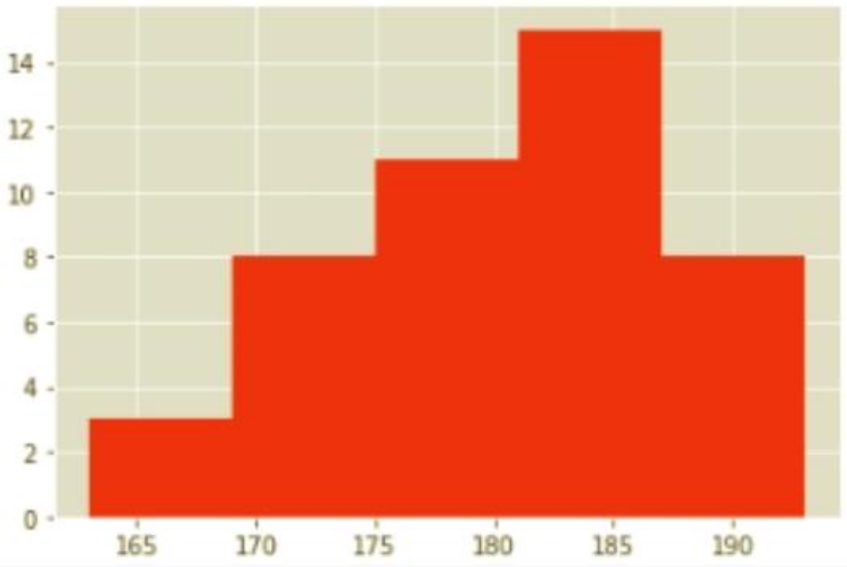
Try it Yourself



plt.hist(presidents_df['height'], bins=5) will create the same plot as above.

```
plt.hist(presidents_df['height'], bins=5)
plt.show()
```

```
(array([ 3.,  8., 11., 15.,  8.]),
 array([163., 169., 175., 181., 187., 193.])
 <a list of 5 Patch objects>)
```



In addition, plt.hist() outputs a 1darray with frequency, and bin end points.

Here we see it is a left skewed distribution (the tail of the distribution on the left hand side is longer than on the right hand side) indicating that the mean (180.0 cm) is slightly lower than the median (182.0 cm). The distribution isn't quite symmetric.

A histogram shows the underlying frequency distribution of a set of continuous data, allowing for inspection of the data for its shape, outliers, skewness, etc.

Boxplot

Remember the output from .describe()?

```
presidents_df['height'].describe()
```

PY

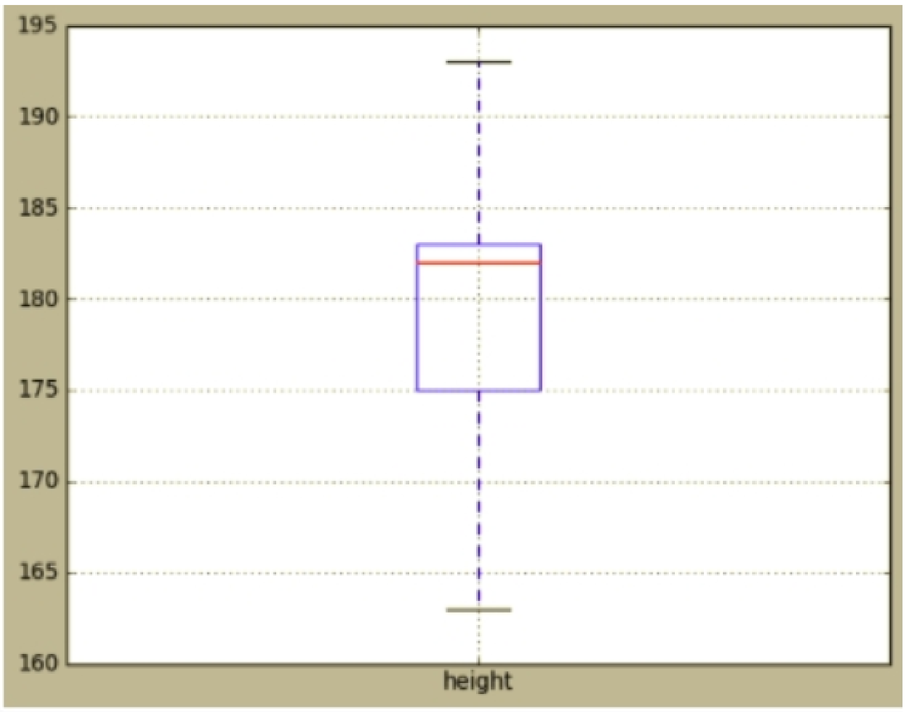
Try it Yourself

The corresponding **boxplot** is shown below. The blue box indicates the interquartile range (IQR, between the first quartile and the third), in other words, 50% data fall in this range. The red bar shows the median, and the lower and upper black whiskers are minimum and maximum.

```
plt.style.use('classic')
presidents_df.boxplot(column='height');
```

PY

Try it Yourself



As the red bar, the median, cuts the box into unequal parts, it means that the height data is skewed.

The box-and-whisker plot doesn't show frequency, and it doesn't display each individual statistic, but it clearly shows where the middle of the data lies and whether the data is skewed.

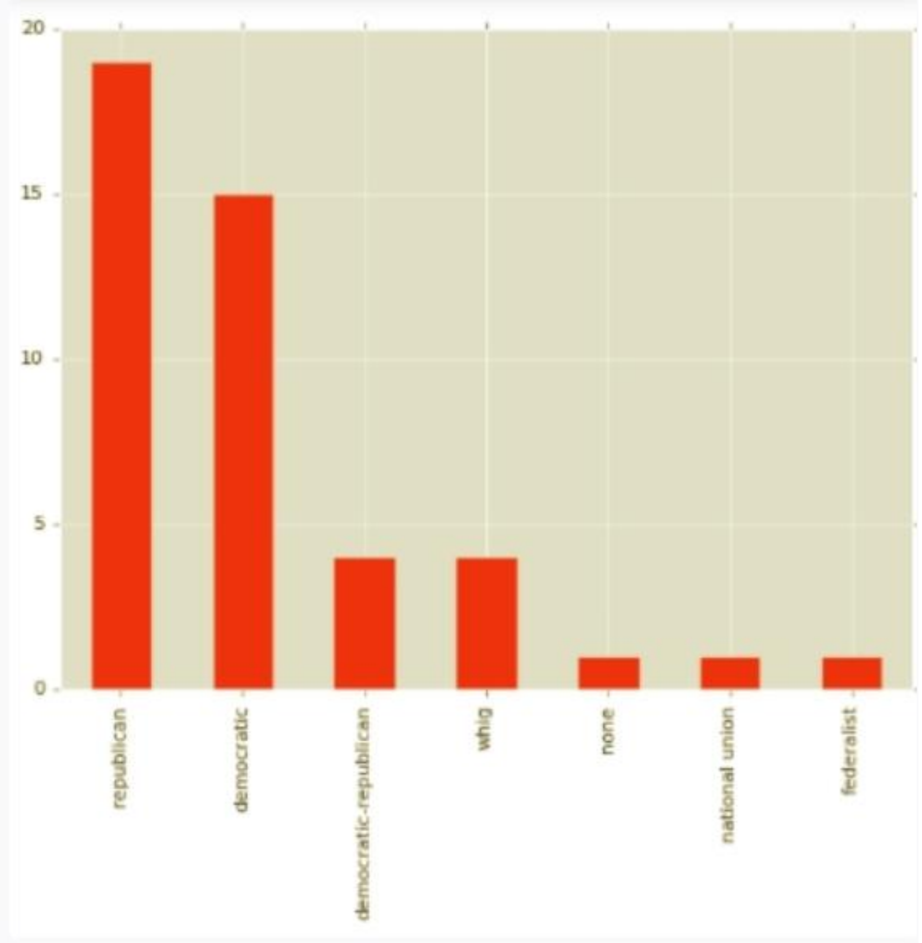
Bar Plot

Bar plots show the distribution of data over several groups. For example, a bar plot can depict the distribution of presidents by party.

```
party_cnt = presidents_df['party'].value_counts()

plt.style.use('ggplot')
party_cnt.plot(kind ='bar')
plt.show()
```

Try it Yourself



Bar plots are commonly confused with a histogram. **Histogram** presents numerical data whereas bar plot shows categorical data. The histogram is drawn in such a way that there is no gap between the bars, unlike in bar plots.

Making plots directly off pandas Series or DataFrame is powerful. It comes in handy when we explore data analysis in data science.