Unit-5: Data Visualization using dataframe:

5.1 importing matplotlib.pyplot and plotting: (only two dimensional Plots)

5.1.1 range() , subplot() , legend(), columns(), len() functions.

5.2 scatter plot: concept of Scatter plot, set title, xlabel and ylabel)

5.3 Line chart : concept of line plot: plot(), set_title(), legend()

5.4 histogram chart : Concepts of histogram hist(),set title, xlabel and ylabel

5.5 Bar Chart : Concepts of Bar chart, bar(),set title, xlabel and ylabel.

5.1 importing matplotlib.pyplot and plotting: (only two dimensional Plots)

Data visualization is the most important step in the life cycle of data science, data analytics, or in data engineering.

It is more impressive, interesting and understanding when we represent our study or analysis with the help of colors and graphics.

Using visualization elements like graphs, charts, maps, etc., it becomes easier for clients to understand the underlying structure, trends, patterns and relationships among variables within the dataset.

Simply explaining the data summary and analysis using plain numbers becomes complicated for both, people coming from technical and non-technical backgrounds.

Data visualization gives us a clear idea of what the data wants to convey to us. It makes data neutral for us to understand the data insights.

Data visualization involves operating a huge amount of data and converts it into meaningful and knowledgeable visuals using various tools.

For visualizing data we need the best software tools to handle various types of data in structured or unstructured format from different sources such as files, web API, databases, and many more.

We must choose the best visualization tool that fulfils all our requirements.

The tool should support interactive plots generation, connectivity to data sources, combining data sources, automatically refresh the data, secured access to data sources, and exporting widgets.

All these features allow us to make the best visuals of our data and also save time.

Advantages of Data Visualization:

Python's popular data analysis library, pandas, provides several different options for visualizing your data with .plot().

Python offers multiple great graphing libraries that come packed with lots of different features. No matter if you want to create interactive, live or highly customized plots python has an excellent library for you.

To get a little overview here are a few popular plotting libraries:

- **Matplotlib:** low level, provides lots of freedom
- **Pandas Visualization:** easy to use interface, built on Matplotlib
- **Seaborn:** high-level interface, great default styles
- **ggplot:** based on R's ggplot2, uses Grammar of Graphics
- **Plotly:** can create interactive plots

Matplotlib

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

To install Matplotlib pip and conda can be used.

```
pip install matplotlib
or
conda install matplotlib
```

Matplotlib is specifically good for creating basic graphs like line charts, bar charts, histograms and many more.

Basic plots in Matplotlib :

Matplotlib comes with a wide variety of plots. Plots helps to understand trends, patterns, and to make correlations.

They're typically instruments for reasoning about quantitative information.

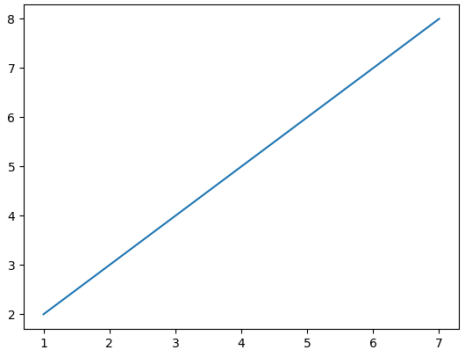
It can be imported by typing:

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

Example : unit-5_exapme/ex1.py

```
import matplotlib.pyplot as pt

x=[1,2,3,4,5,6,7]
y=[2,3,4,5,6,7,8]

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

Output :



5.1.1 range() , subplot() , legend(), columns(), len() functions.

1. subplot() :

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

Example : Draw 2 plots: unit-5_exapme/ex2_sunplot.py

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

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

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

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

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

The subplots() Function

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.

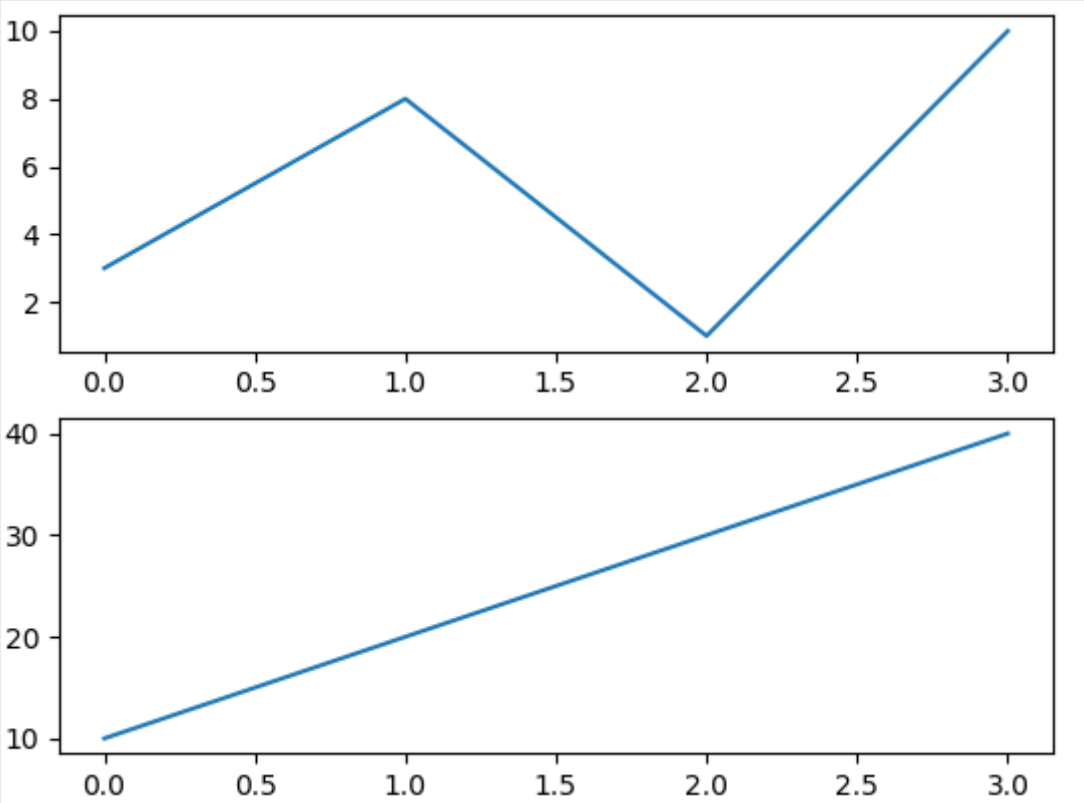
```
plt.subplot(1, 2, 1)

#the figure has 1 row, 2 columns, and this plot is the first plot.
```

```
plt.subplot(1, 2, 2)
#the figure has 1 row, 2 columns, and this plot is the second plot.
```

So, if we want a figure with 2 rows an 1 column (meaning that the two plots will be displayed on top of each other instead of side-by-side), we can write the syntax like this:

Example: **Draw 2 plots on top of each other:** unit-5_exapme/ex3_subplot.py

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

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

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

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

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

plt.show()
```

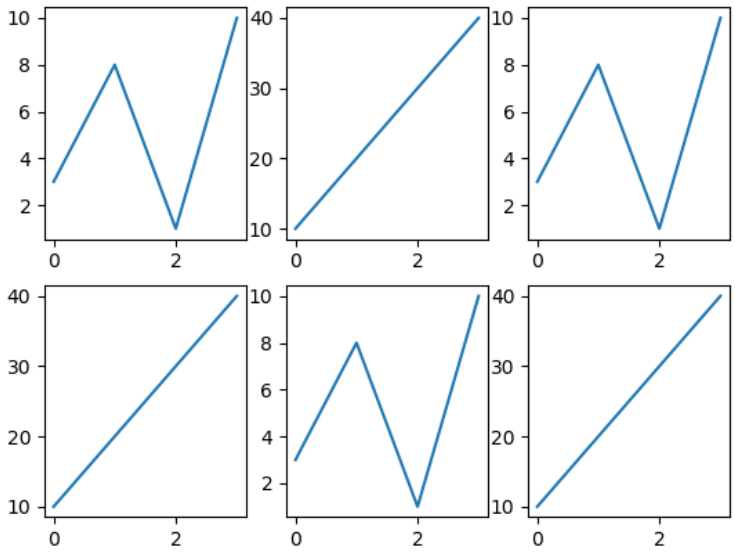


You can draw as many plots you like on one figure, just describe the number of rows, columns, and the index of the plot.

Example **Draw 6 plots:**

```
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(2, 3, 1)
plt.plot(x,y)

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

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

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(2, 3, 5)
```

```
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(2, 3, 6)
plt.plot(x,y)

plt.show()
```



Marker Reference

Marker	Description
'o'	Circle
'*'	Star
'.'	Point
','	Pixel
'x'	X
'X'	X (filled)
'+'	Plus
'P'	Plus (filled)
's'	Square
'D'	Diamond
'd'	Diamond (thin)
'p'	Pentagon
'H'	Hexagon
'h'	Hexagon
'v'	Triangle Down
'^'	Triangle Up
'<'	Triangle Left
'>'	Triangle Right
'1'	Tri Down
'2'	Tri Up
'3'	Tri Left
'4'	Tri Right
'|'	Vline

'_'	Hline

Line Reference

Line Syntax	Description
'-'	Solid line
':'	Dotted line
'--'	Dashed lin
'-.'	Dashed/dotted line

Color Reference

Color Syntax	Description
'r'	Red
'g'	Green
'b'	Blue
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

https://www.w3schools.com/python/matplotlib_markers.asp
https://www.w3schools.com/python/matplotlib_line.asp

for title x and y axis

```
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
```

2. legend()

Matplotlib has native support for legends.
Legends can be placed in various positions:
A legend can be placed inside or outside the chart and the position can be moved

The legend() method adds the legend to the plot.

Matplotlib.pyplot.legend()

Syntax:

matplotlib.pyplot.legend([“blue”, “green”], bbox_to_anchor=(0.75, 1.15), ncol=2)

The attribute Loc in legend() is used to specify the location of the legend.
Default value of **loc** is loc=”best” (upper left).
The strings ‘upper left’, ‘upper right’, ‘lower left’, ‘lower right’ place the legend at the corresponding corner of the axes/figure.

The attribute bbox_to_anchor=(x, y) of legend() function is used to specify the coordinates of the legend, and the attribute ncol represents the number of columns that the legend has.It’s default value is 1.

The Following are some more attributes of function legend() :

- **shadow**: [None or bool] Whether to draw a shadow behind the legend.It’s Default value is None.
- **markerscale**: [None or int or float] The relative size of legend markers compared with the originally drawn ones.The Default is None.

- **numpoints**: [None or int] The number of marker points in the legend when creating a legend entry for a Line2D (line).The Default is None.
- **fontsize**: The font size of the legend.If the value is numeric the size will be the absolute font size in points.
- **facecolor**: [None or “inherit” or color] The legend’s background color.
- **edgecolor**: [None or “inherit” or color] The legend’s background patch edge color.

Example 1 : unit-5_exapme/ex4_legend.py

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

x = [1, 2, 3, 4, 5]
y = [1, 4, 9, 16, 25]

pt.plot(x, y)
pt.legend(['single element legend'])

pt.show()
```

Example 2 : unit-5_exapme/ex4_legend.py

```
import matplotlib.pyplot as pt
import numpy as np
x = [2, 3, 4.5]
y = [1, 1.5, 5]
pt.plot(x)
pt.plot(y)

pt.title('Legend inside')
pt.legend(["blue", "green"], loc ="upper right")
 pt.show()
```

3. columns()
4. len()

Scatter Plot

To create a scatter plot in Matplotlib we can use the scatter method.
We will also create a figure and an axis using plt.subplots so we can give our plot a title and labels.
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

Example : unit-5_exapme/ex1.py

```
import matplotlib.pyplot as pt

x=[1,2,3,4,5,6,7]
y=[2,3,4,5,6,7,8]

#scatter plot
```

```
pt.scatter(x,y)
pt.sow()
```

Example :Draw two plots on the same figure:

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

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2])
y = np.array([99,86,87,88,111])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12])
y = np.array([100,105,84,105,90,99,90])
plt.scatter(x, y)
plt.show()
```

Colors: Example

```
plt.scatter(x, y, color = pink)
OR
plt.scatter(x, y, color = '#88c999')
```

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.
Here is an example of a colormap:

This colormap is called 'viridis' and as you can see it ranges from 0, which is a purple color, and up to 100, which is a yellow color.

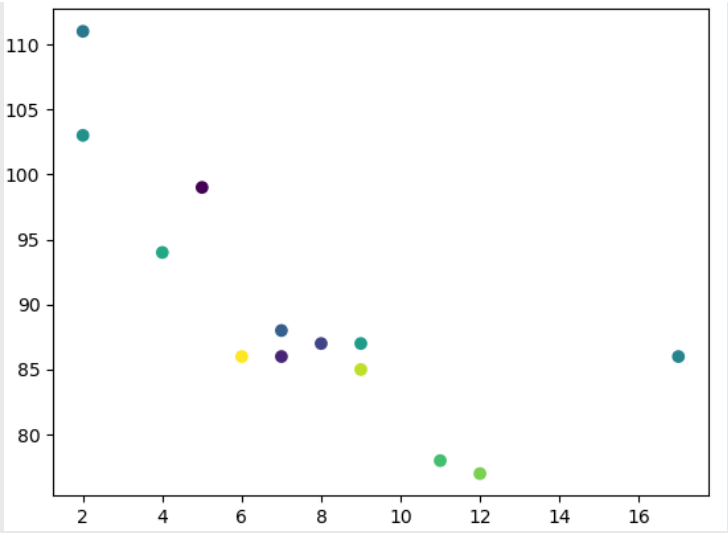
How to Use the ColorMap

You can specify the colormap with the keyword argument cmap with the value of the colormap, in this case 'viridis' which is one of the built-in colormaps available in Matplotlib.

In addition you have to create an array with values (from 0 to 100), one value for each of the point in the scatter plot:

Example :**Create a color array, and specify a colormap in the scatter plot:**

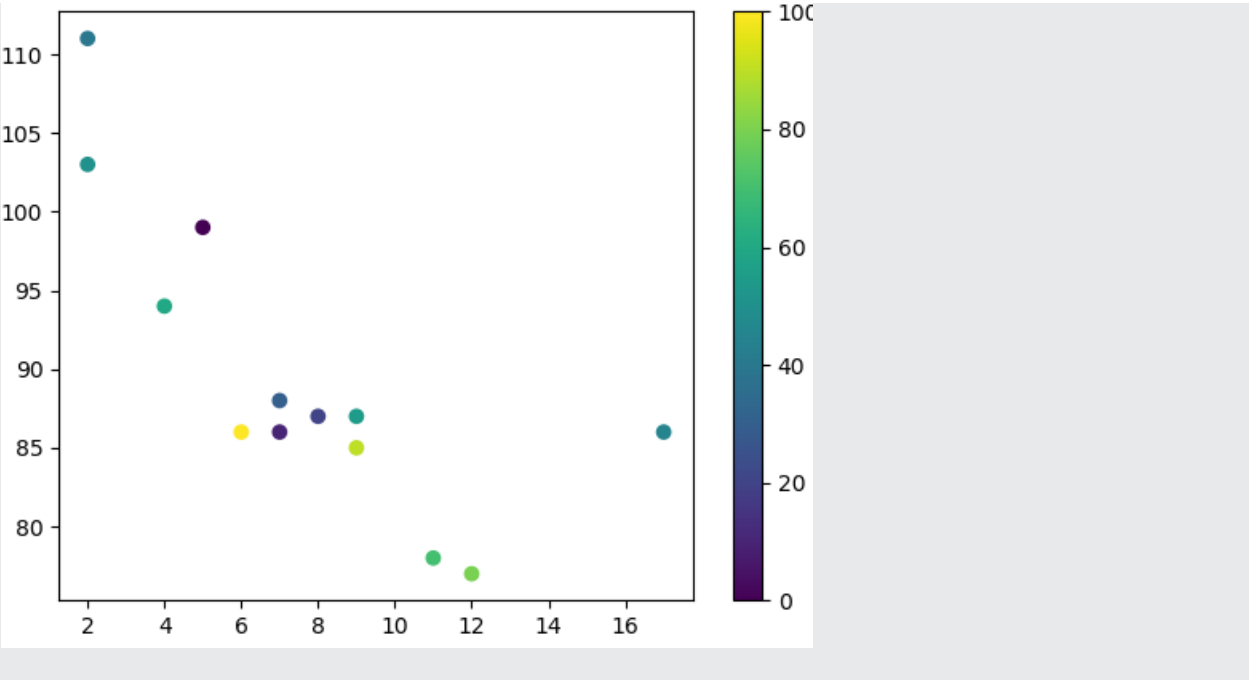
```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])
plt.scatter(x, y, c=colors, cmap='viridis')
plt.show()
```

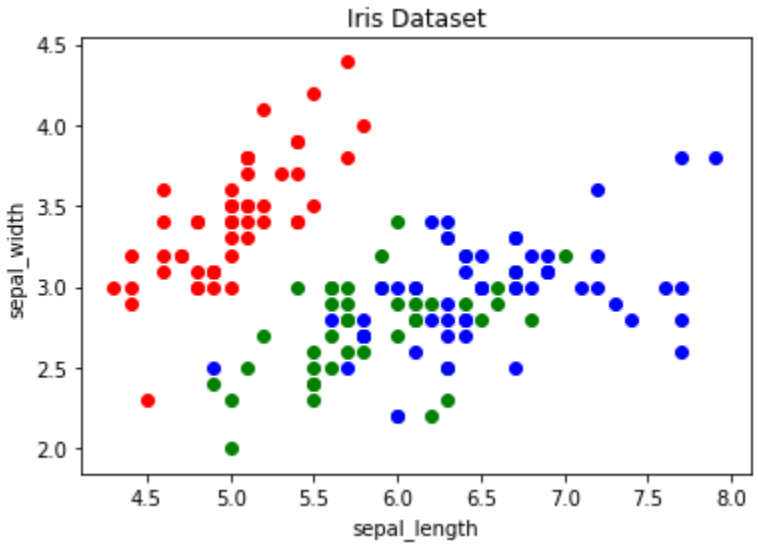


You can include the colormap in the drawing by including the plt.colorbar() statement:

Example:Include the actual colormap:

```
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])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])
plt.scatter(x, y, c=colors, cmap='viridis')
plt.colorbar()
plt.show()
```





Line Chart

In Matplotlib we can create a line chart by calling the plot method.
Line charts are used to represent the relation between two data X and Y on a different axis.
Here we will see some of the examples of a line chart in Python :
First **import Matplotlib.pyplot** library for plotting functions.

Also, import the Numpy library as per requirement.

Then define data values x and y.

We can also plot multiple columns in one graph, by looping through the columns we want and plotting each column on the same axis.

Example : unit-5_exapme/ex5_line.py

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

x = np.array([1, 2, 3, 4])  # X-axis points
```

```
y = x*2  # Y-axis points

plt.plot(x, y)  # Plot the chart
plt.show()  # display
```

Example : unit-5_exapme/ex5_line.py

```
x = np.array([1, 2, 3, 4])
y = x*2

pt.plot(x, y)
pt.xlabel("X-axis")  # add X-axis label
pt.ylabel("Y-axis")  # add Y-axis label
pt.title("This is Line Chart")  # add title
pt.show()
```

Multiple plots on the same axis

Here we will see how to add 2 plots within the same axis.

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

x = np.array([1, 2, 3, 4])
y = x*2

# first plot with X and Y data
plt.plot(x, y)

x1 = [2, 4, 6, 8]
y1 = [3, 5, 7, 9]

# second plot with x1 and y1 data
plt.plot(x1, y1, '-.')

plt.xlabel("X-axis data")
plt.ylabel("Y-axis data")
plt.title('multiple plots')
plt.show()
```

Matplotlib.axes.Axes.set_title() in Python

Matplotlib is a library in Python and it is numerical – mathematical extension for NumPy library.
The **Axes Class** contains most of the figure elements: Axis, Tick, Line2D, Text, Polygon, etc., and sets the coordinate system.

And the instances of Axes supports callbacks through a callbacks attribute.

matplotlib.axes.Axes.set_title() Function

The **Axes.set_title() function** in axes module of matplotlib library is used to set a title for the axes.
Syntax: Axes.set_title(self, label, fontdict=None, loc='center', pad=None, **kwargs)

Parameters: This method accepts the following parameters.

- **label :** This parameter is the Text to use for the title.
- **fontdict :** This parameter is the dictionary controlling the appearance of the title text.
- **loc :** This parameter is used to set the location of the title {'center', 'left', 'right'}.
- **pad :** This parameter is the offset of the title from the top of the axes, in points.
- **Returns:**This method returns the matplotlib text instance representing the title.

Example 1: matplotlib.axes.Axes.set_title() function in matplotlib.axes

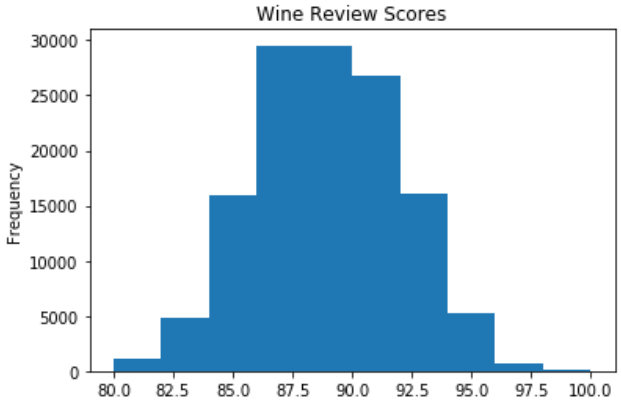
```
import matplotlib.pyplot as plt
#plot 1:
x = [0, 1, 2, 3]
y = [3, 8, 1, 10]

ax = plt.subplot(1, 2, 1)

plt.plot(x, y, 'o',linestyle = 'dotted', color='green')
ax.set_title('Title with special font',fontsize = 14)
plt.show()
```

Histogram

A histogram is a graph showing *frequency* distributions.
It is a graph showing the number of observations within each given interval.

If we pass it categorical data like the points column from the wine-review dataset it will automatically calculate how often each class occurs.

Example: Say you ask for the height of 250 people, you might end up with a histogram like this:



You can read from the histogram that there are approximately:

2 people from 140 to 145cm
5 people from 145 to 150cm
15 people from 151 to 156cm
31 people from 157 to 162cm
46 people from 163 to 168cm
53 people from 168 to 173cm
45 people from 173 to 178cm
28 people from 179 to 184cm
21 people from 185 to 190cm
4 people from 190 to 195cm

Create Histogram

In Matplotlib, we use the hist() function to create histograms.
The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10.

Example :**A Normal Data Distribution by NumPy:**

```
import numpy as np
x = np.random.normal(170, 10, 250)
print(x)

=====
#usinghist()

import matplotlib.pyplot as pt
import numpy as np

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

pt.hist(x)
pt.show()
```

Example

```
# create figure and axis
fig, ax = plt.subplots()

# plot histogram
ax.hist(wine_reviews['points'])

# set title and labels
ax.set_title('Wine Review Scores')
ax.set_xlabel('Points')
ax.set_ylabel('Frequency')
```

Bar Chart

A bar chart can be created using the bar method.
The bar-chart isn't automatically calculating the frequency of a category so we are going to use pandas value_counts function to do this.

The bar-chart is useful for categorical data that doesn't have a lot of different categories (less than 30) because else it can get quite messy.

Creating Bars

With Pyplot, you can use the bar() function to draw bar graphs:

Example :Draw 4 bars:

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

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

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

The bar() function takes arguments that describes the layout of the bars.

The categories and their values represented by the *first* and *second* argument as arrays.

Horizontal Bars

If you want the bars to be displayed horizontally instead of vertically, use the barh() function:

Example :Draw 4 horizontal bars

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

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

plt.barh(x, y)

# The bar() and barh() takes the keyword argument color to set the color of the bars:
# plt.bar(x, y, color = "red")
plt.show()
```