Module -5 (Data Processing)

Plotting and visualization. Matplotlib - Basic plot, Ticks, Labels, and Legends. Working with CSV files. – Pandas - Reading, Manipulating, and Processing Data.

Plotting and Visualization

Visualization libraries

- **matplotlib**
- **Seaborn**

Plotting and Visualization

matplotlib:

- python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- a set of functionalities similar to those of MATLAB
- line plots, scatter plots, barcharts, histograms, pie charts etc.
- relatively low-level; some effort needed to create advanced visualization

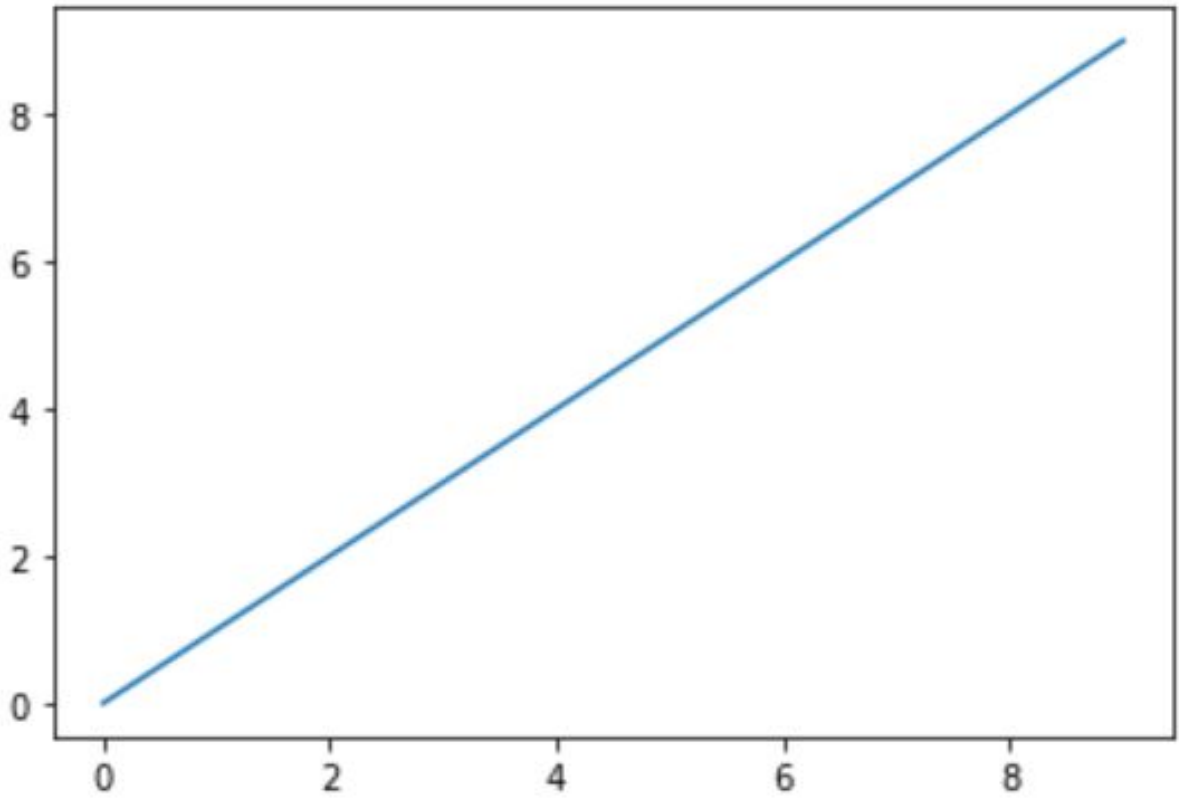
Plotting and Visualization

Seaborn:

- based on matplotlib
- provides high level interface for drawing attractive statistical graphics
- Similar (in style) to the popular ggplot2 library in R

Simple line plot

```
import matplotlib.pyplot as plt
import numpy as np
data = np.arange(10)
plt.plot(data)
```



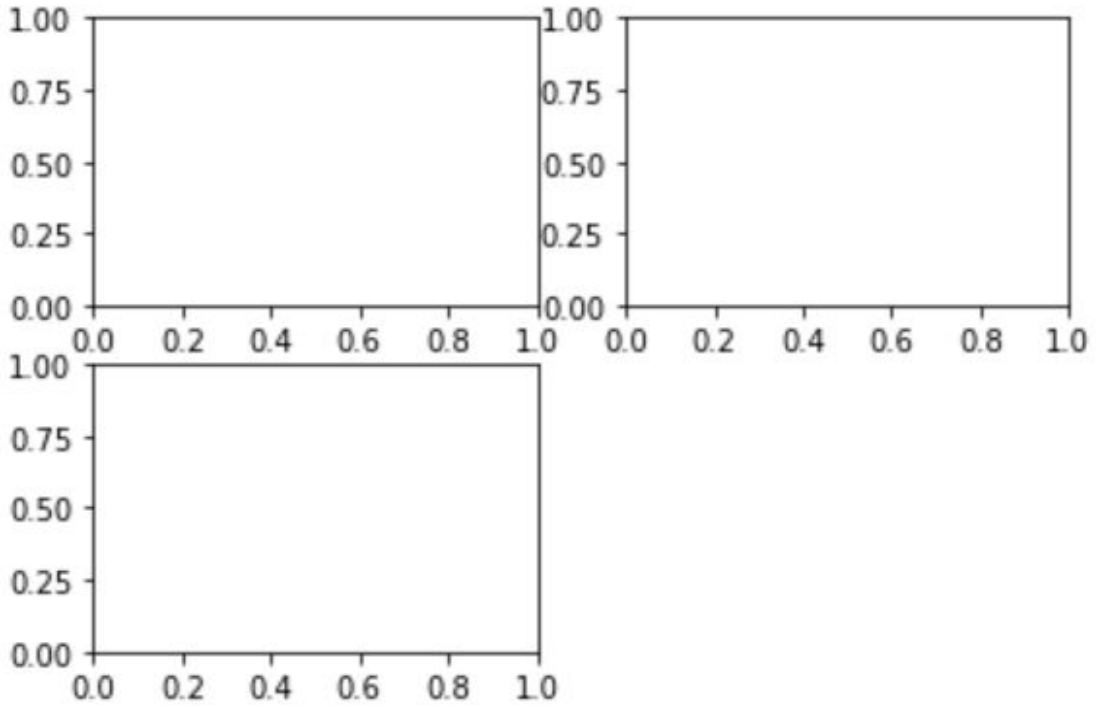
Figures and Subplots

- Plots in matplotlib reside within a Figure object.
- A new figure can be created with plt.figure

In IPython, an empty plot window will appear, but in Jupyter nothing will be shown

```
fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
```

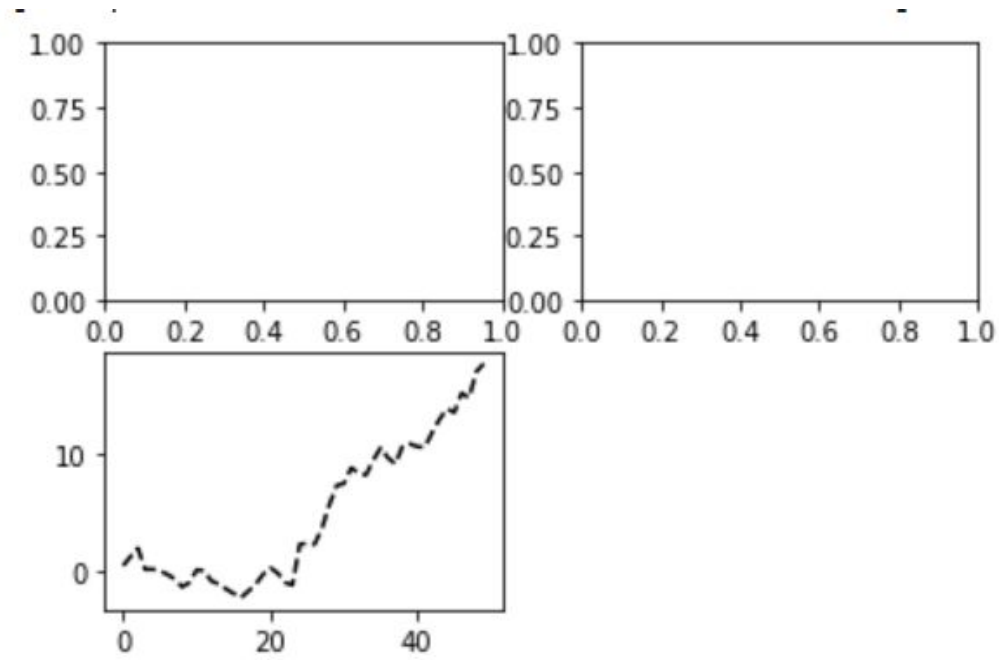
This means that the figure should be 2 × 2 (so up to four plots in total), and we're selecting the first of four subplots (numbered from 1).



An empty matplotlib

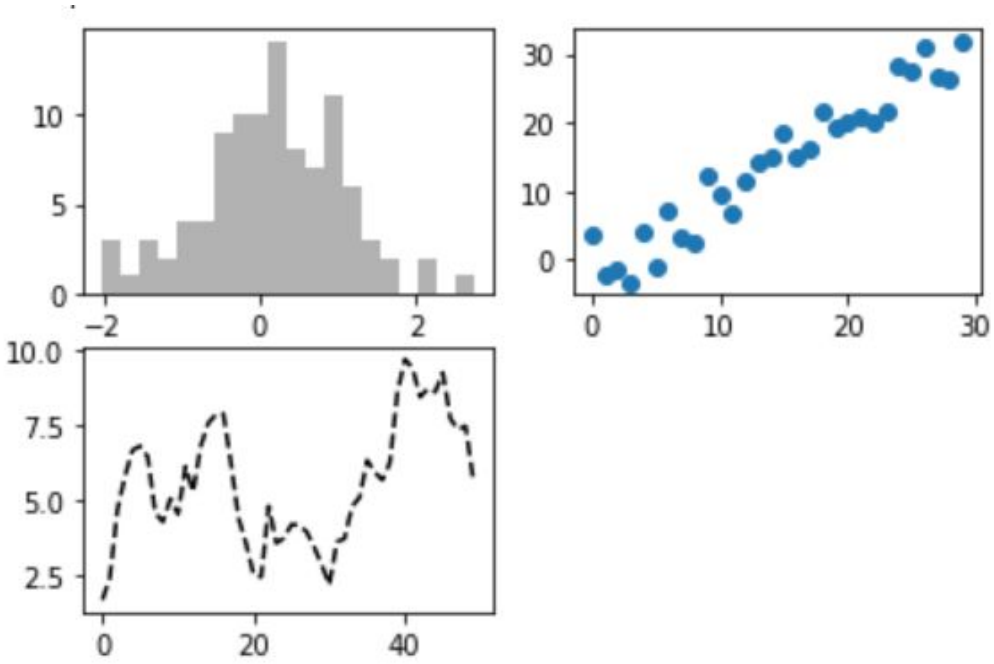
- When you issue a plotting command like plt.plot(), matplotlib draws on the last figure and subplot

```
fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
plt.plot(np.random.randn(50).cumsum(), 'k-
```



- 'k--' is a style option instructing matplotlib to plot a black dashed line.

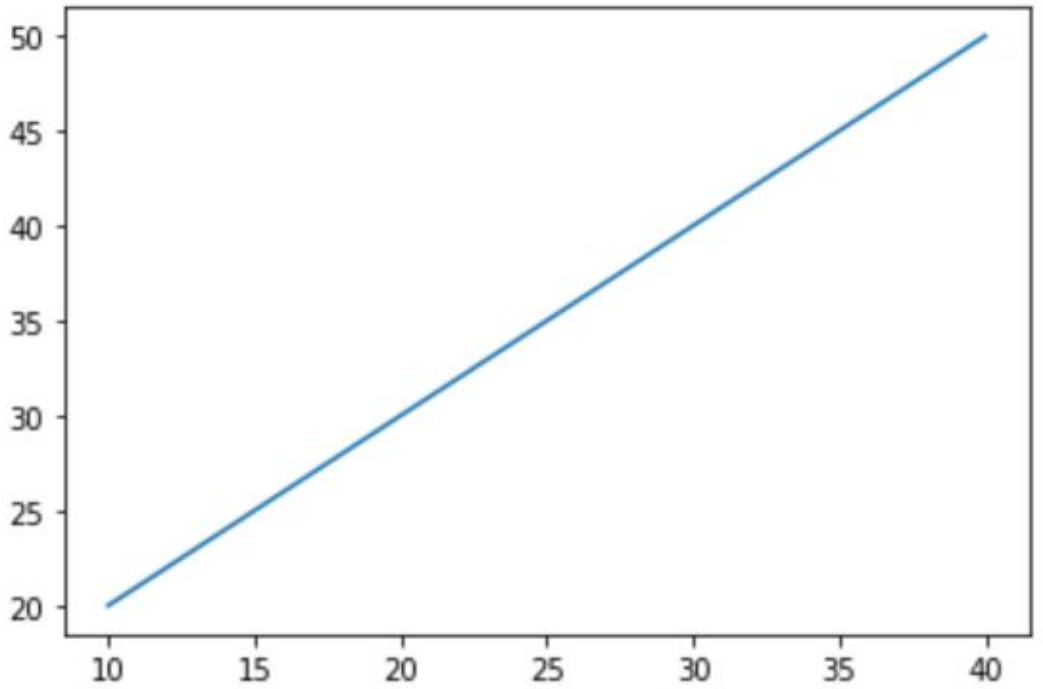
```
fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
plt.plot(np.random.randn(50).cumsum(), 'k--')
ax1.hist(np.random.randn(100), bins=20, color='k', alpha=0.3)
ax2.scatter(np.arange(30), np.arange(30) + 3 * np.random.randn(30))
```



Basic plots:

- Line Plots:

```
import matplotlib.pyplot as plt
# initializing the data
x = [10, 20, 30, 40]
y = [20, 30, 40, 50]
# plotting the data
plt.plot(x, y)
plt.show()
```

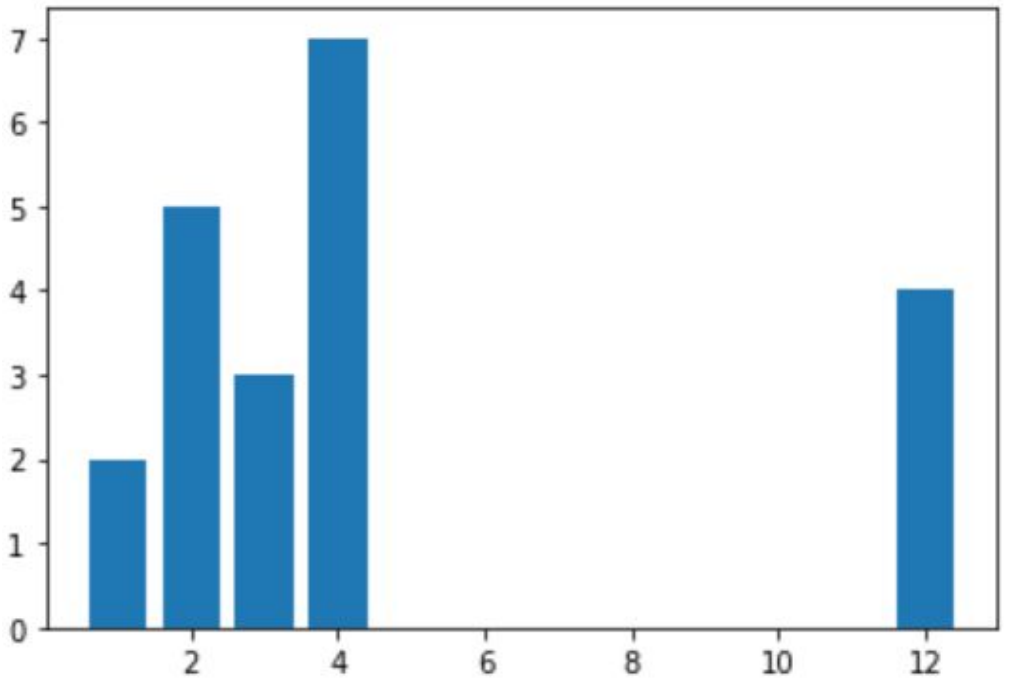


Bar chart

- A bar plot or bar chart is a graph that represents the category of data with rectangular bars with lengths and heights that is proportional to the values which they represent.
- The bar plots can be plotted horizontally or vertically.
- It can be created using the bar() method.

plt.bar(x, height, width, bottom, align)

```
import matplotlib.pyplot as plt
# data to display on plots
x = [3, 1, 3, 12, 2, 4, 4]
y = [3, 2, 1, 4, 5, 6, 7]
# This will plot a simple bar chart
plt.bar(x, y)
```



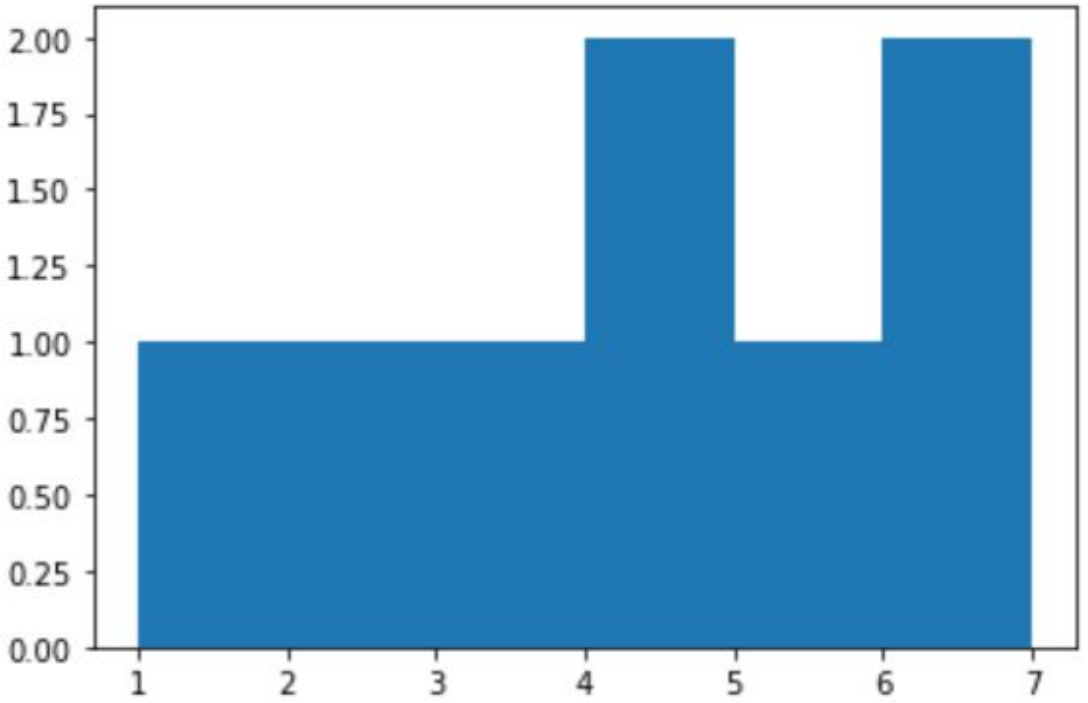
Histograms

- A histogram is basically used to represent data in the form of some groups.
- It is a type of bar plot where the X-axis represents the bin ranges while the Y-axis gives information about frequency.

```
matplotlib.pyplot.hist(x, bins=None, range=None, density=False, weights=None,
cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical',
rwidth=None, log=False, color=None, label=None, stacked=False, \*, data=None, \*\*kwargs)
```

Histograms

```
import matplotlib.pyplot as plt
# data to display on plots
x = [1, 2, 3, 4, 5, 6, 7, 4]
# This will plot a simple histogram
plt.hist(x, bins = [1, 2, 3, 4, 5, 6, 7])
plt.show()
```

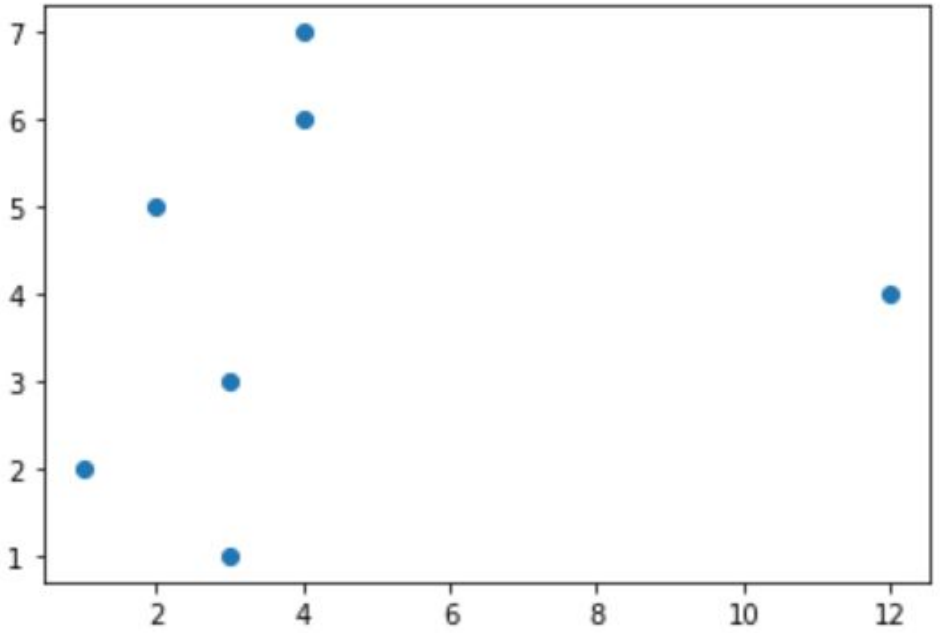


Scatter Plot

- Scatter plots are used to observe the relationship between variables and use dots to represent the relationship between them.
- The **scatter()** method in the matplotlib library is used to draw a scatter plot.

matplotlib.pyplot.scatter(x_axis_data, y_axis_data, s=None, c=None, marker=None, cmap=None, vmin=None, vmax=None, alpha=None, linewidths=None, edgecolors=None)

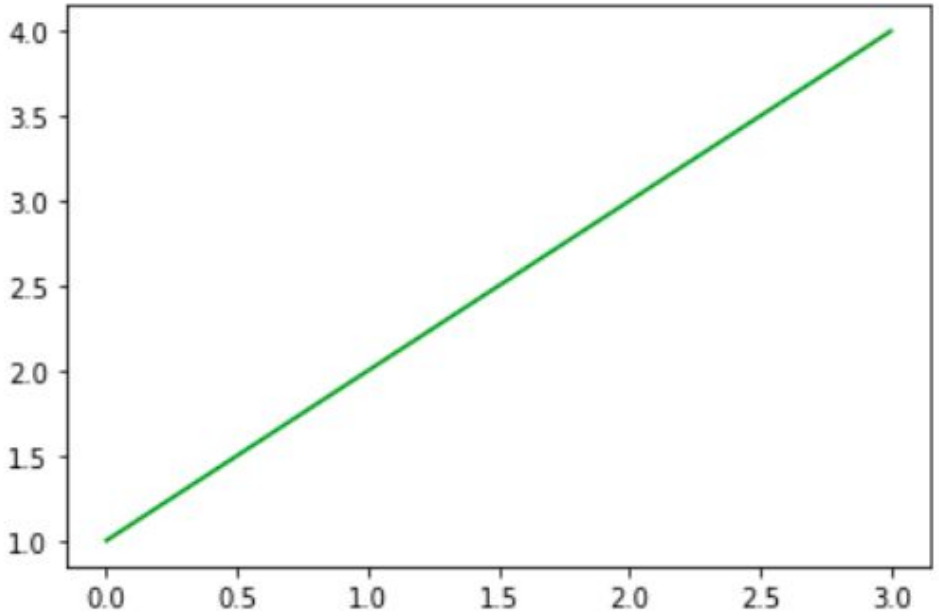
```
import matplotlib.pyplot as plt
# data to display on plots
x = [3, 1, 3, 12, 2, 4, 4]
y = [3, 2, 1, 4, 5, 6, 7]
# This will plot a simple scatter chart
plt.scatter(x, y)
```



Colors, Markers, and Line Styles

Matplotlib.pyplot.colors() : This function is used to specify the color. It is **do-nothing** function.

```
import matplotlib.pyplot as plt
# Define the Color
color = 'green'
plt.plot([1, 2, 3, 4], color =color)
  plt.show()
```

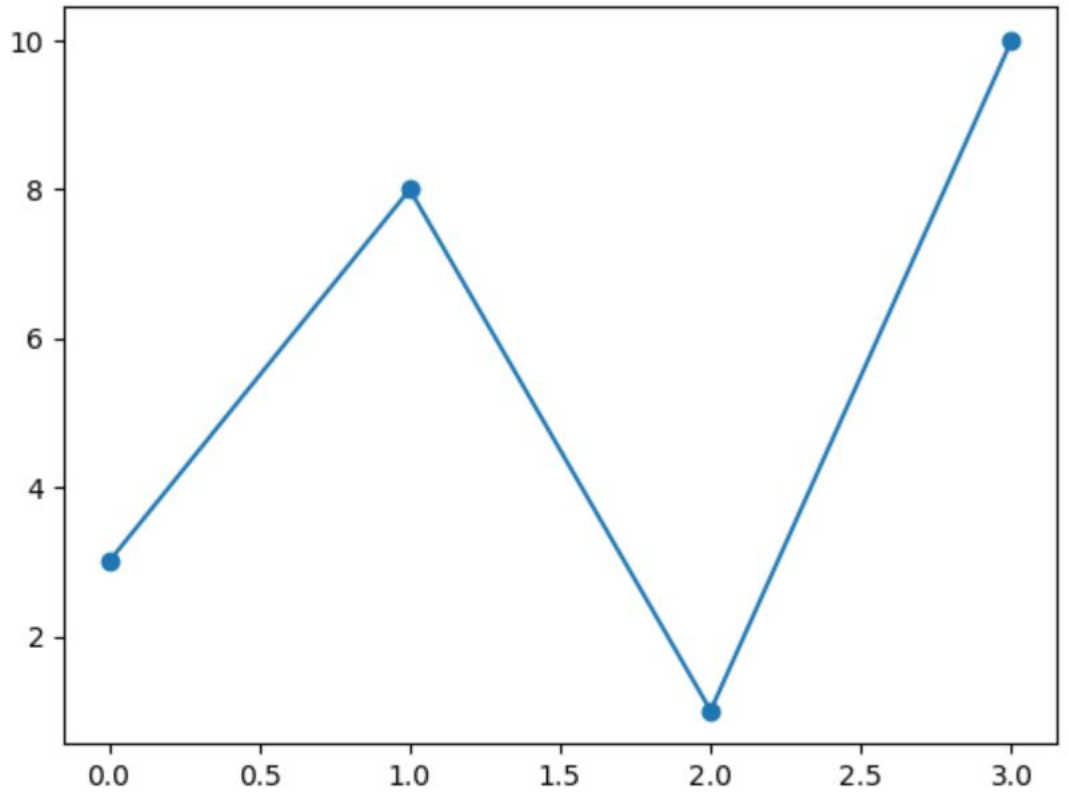


Alias	Color
'b'	Blue
'r'	Red
'g'	Green
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Colors, Markers, and Line Styles

- the keyword argument **marker** to emphasize each point with a specified marker:

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, marker = 'o')
plt.show()
```



Mark each point with a star:

```
plt.plot(ypoints, marker = '*')
```

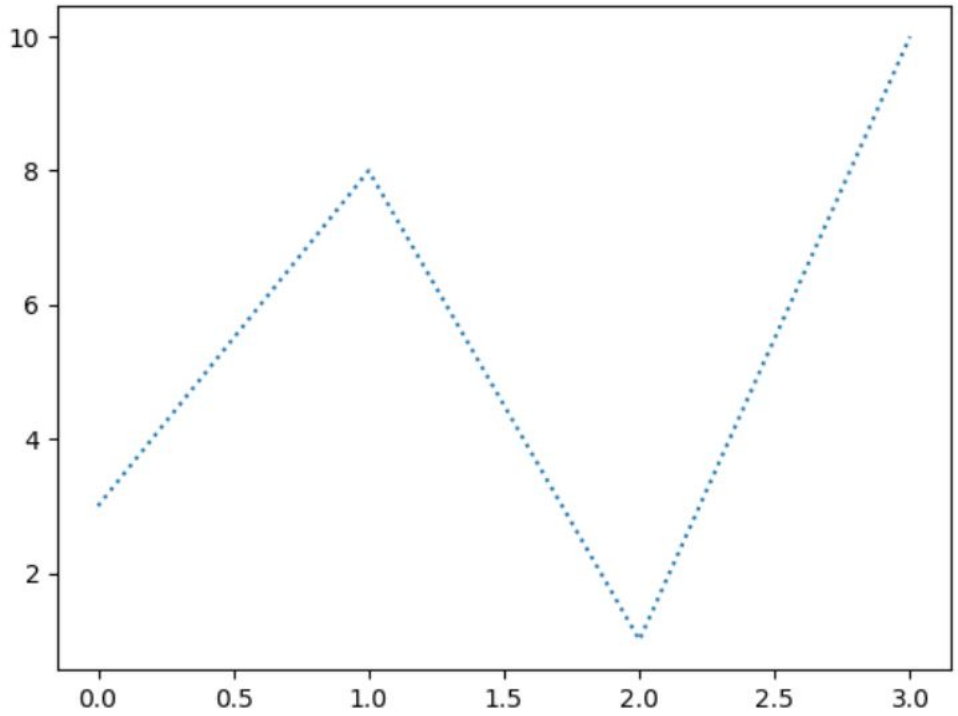
Colors, Markers, and Line Styles

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

Colors, Markers, and Line Styles

use the keyword argument **linestyle**, or shorter ls, to **change the style of the plotted line:**

```
import matplotlib.pyplot as plt
import numpy as np
ypoints = np.array([3, 8, 1, 10])
plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```



Shorter syntax:

```
plt.plot(ypoints, ls = ':')
```

Colors, Markers, and Line Styles

Style	Or
'solid' (default)	'-'
'dotted'	':'
'dashed'	'--'
'dashdot'	'-.'
'None'	'' or ' '

Ticks, Labels, and Legends

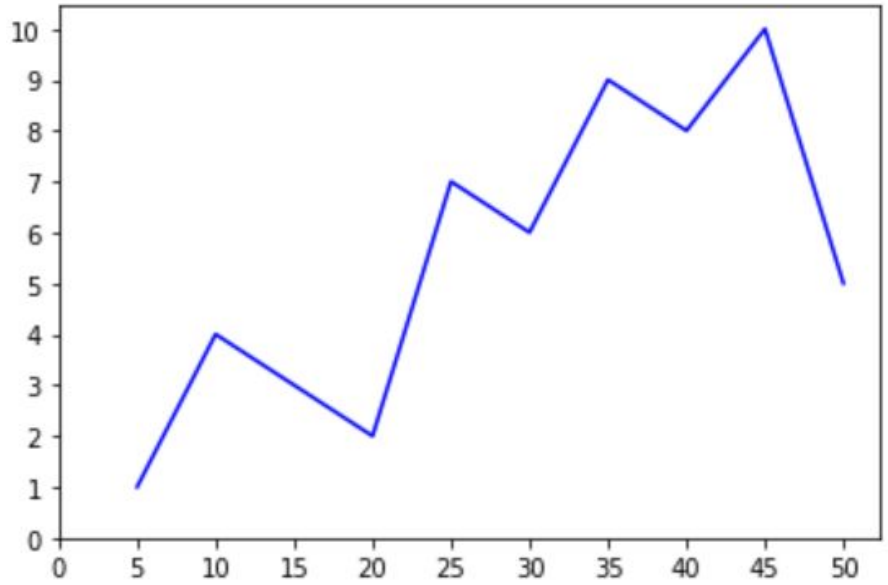
- **Ticks** are the values used to show **specific points on the coordinate axis**. It can be a **number or a string**. Whenever we plot a graph, the axes adjust and take the default ticks.

Parameter	Value	Use
axis	x, y, both	Tells which axis to operate
reset	True, False	If True, set all parameters to default
direction	in, out, inout	Puts the ticks inside or outside or both
length	Float	Sets tick's length
width	Float	Sets tick's width
rotation	Float	Rotates ticks wrt the axis
colors	Color	Changes tick color
pad	Float	Distance in points between tick and label

Ticks, Labels, and Legends

- The **xticks()** and **yticks()** function takes a list object as argument. The elements in the list denote the positions on corresponding action where ticks will be displayed.

```
# importing libraries
import matplotlib.pyplot as plt
import numpy as np
# values of x and y axes
x = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
y = [1, 4, 3, 2, 7, 6, 9, 8, 10, 5]
plt.plot(x, y, 'b')
# 0 is the initial value, 51 is the final value
# (last value is not taken) and 5 is the differenc
e
# of values between two consecutive ticks
plt.xticks(np.arange(0, 51, 5))
plt.yticks(np.arange(0, 11, 1))
plt.show()
```



Ticks, Labels, and Legends

- use the **xlabel() and ylabel()** functions to set a label for the x- and y-axis.

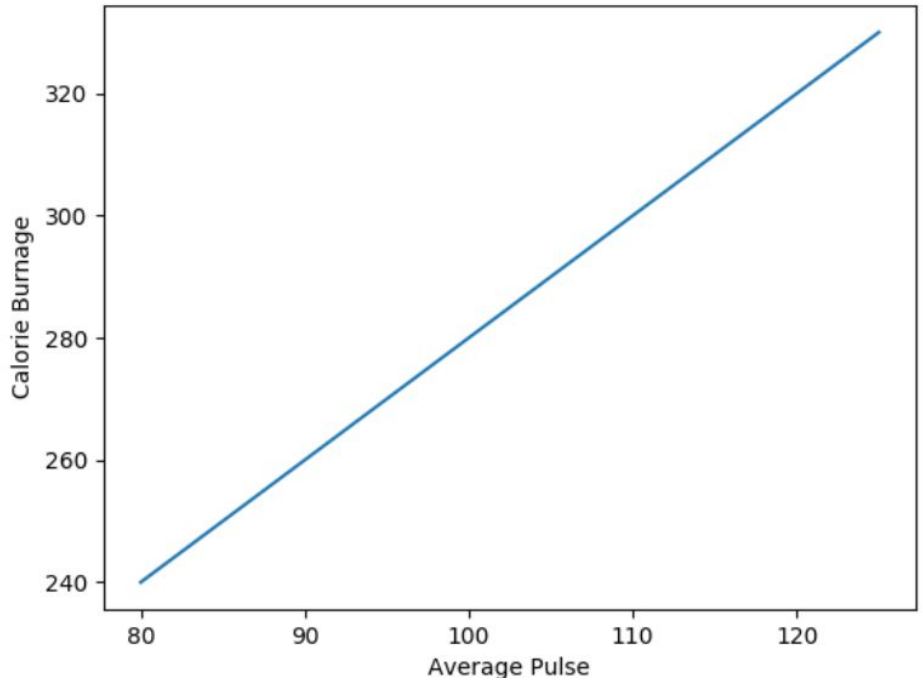
```
import numpy as np
import matplotlib.pyplot as plt

x =np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

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

plt.show()
```

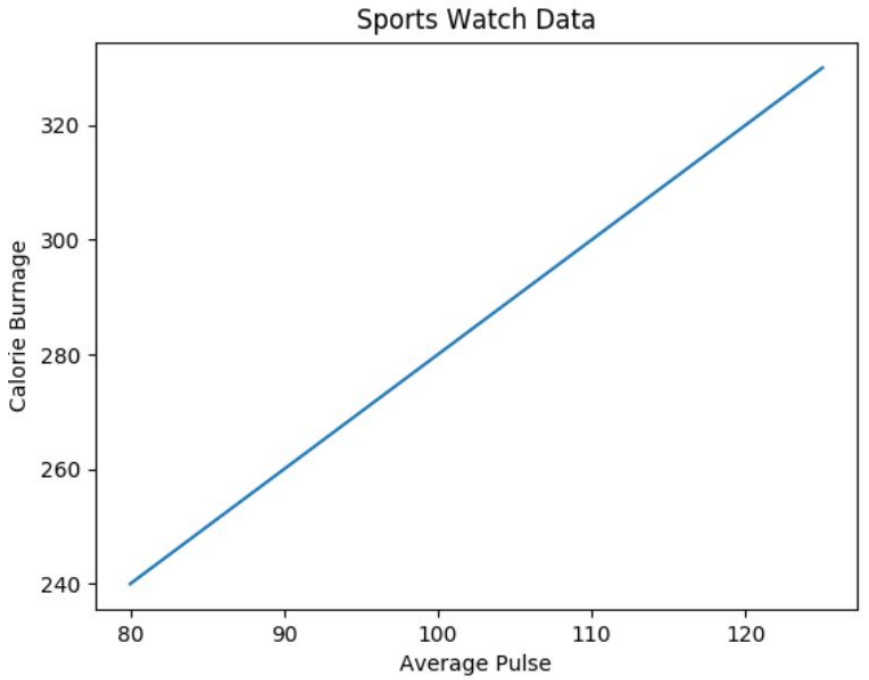


Ticks, Labels, and Legends

use the **title()** function to set a **title for the plot**.

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

x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])
plt.plot(x, y)
plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")
plt.show()
```



use the `loc` parameter in `title()` to position the title. Legal values are: 'left', 'right', and 'center'. Default value is 'center'.

```
plt.title("Sports Watch Data", loc = 'left')
```

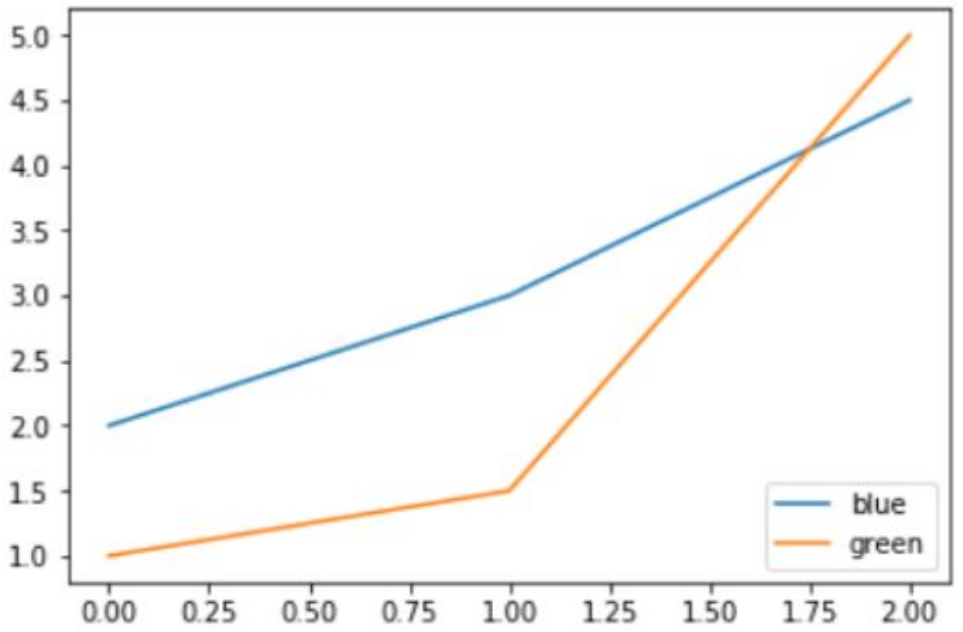
Ticks, Labels, and Legends

- Matplotlib.pyplot.legend() : A legend is an area describing the elements of the graph. In the matplotlib library, there's a function called **legend()** which is used to Place a legend on the axes.
- 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
  - *the attribute **ncol** represents the number of columns that the legend has.*
  - *t's default value is 1.*

Ticks, Labels, and Legends

- **matplotlib.pyplot.legend(["blue", "green"], bbox_to_anchor=(0.75, 1.15), ncol=2)**

```
# importing modules
import numpy as np
import matplotlib.pyplot as plt
# Y-axis values
y1 = [2, 3, 4.5]
# Y-axis values
y2 = [1, 1.5, 5]
# Function to plot
plt.plot(y1)
plt.plot(y2)
# Function add a legend
plt.legend(["blue", "green"], loc ="lower right")
# function to show the plot
plt.show()
```



Ticks, Labels, and Legends

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.

- Working with CSV files: Pandas - Reading, Manipulating, and Processing Data.
- Introduction to Micro services using Flask.

CSV file format

- CSV stands for “Comma Separated Values.”
- It is the simplest form of storing tabular data, such as a spreadsheet or database in tabular form as plain text.
- .Each line of the file is a data record.
- Each record consists of one or more fields, separated by commas.
- The use of the comma as a field separator is the source of the name for this file format.
- For working CSV files in python, there is an inbuilt module called csv.
- A simple way to store big data sets is to use CSV files.

pandas

- pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
- It contains data structures and data manipulation tools designed to make data cleaning and analysis fast and easy in Python.
- pandas is designed for working with tabular or heterogeneous data. NumPy, by contrast, is best suited for working with homogeneous numerical array data.

```
import pandas as pd
```

pandas Data Structures

- **Series** : A Series is a one-dimensional array-like object containing a sequence of values and an associated array of data labels, called its index.

```
obj = pd.Series([4, 7, -5, 3])
```

- create a Series with an index identifying each data point with a label:

```
obj2 = pd.Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c’])
obj2['a’]
obj2[['c', 'a', 'd']]
```

DataFrame

- A DataFrame represents a rectangular table of data and contains an ordered collection of columns, each of which can be a different value type (numeric, string, boolean, etc.).
- The DataFrame has both a row and column index
- construct a DataFrame:

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
'year': [2000, 2001, 2002, 2001, 2002, 2003],
'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}

frame = pd.DataFrame(data)
```

- If you specify a sequence of columns, the DataFrame's columns will be arranged in that order:

```
pd.DataFrame(data, columns=['year', 'state', 'pop'])
```

	state	year	pop
0	Ohio	2000	1.5
1	Ohio	2001	1.7
2	Ohio	2002	3.6
3	Nevada	2001	2.4
4	Nevada	2002	2.9
5	Nevada	2003	3.2

A column in a DataFrame can be retrieved as a Series by attribute:

frame2['state’]

frame2.year

frame2[column] works for any column name, but frame2.column only works when the column name is a valid Python variable name.

Rows can also be retrieved by position or name with the special loc attribute

frame2.loc['three']

Columns can be modified by assignment.

frame2['debt'] = 16.5

frame2['debt'] = np.arange(6)

- When you are assigning lists or arrays to a column, the value’s length must match the length of the DataFrame.
- Assigning a column that doesn’t exist will create a new column
- del method can then be used to remove column: **del frame2[‘column name']**

- transpose the DataFrame (swap rows and columns) with similar syntax to a NumPy array:

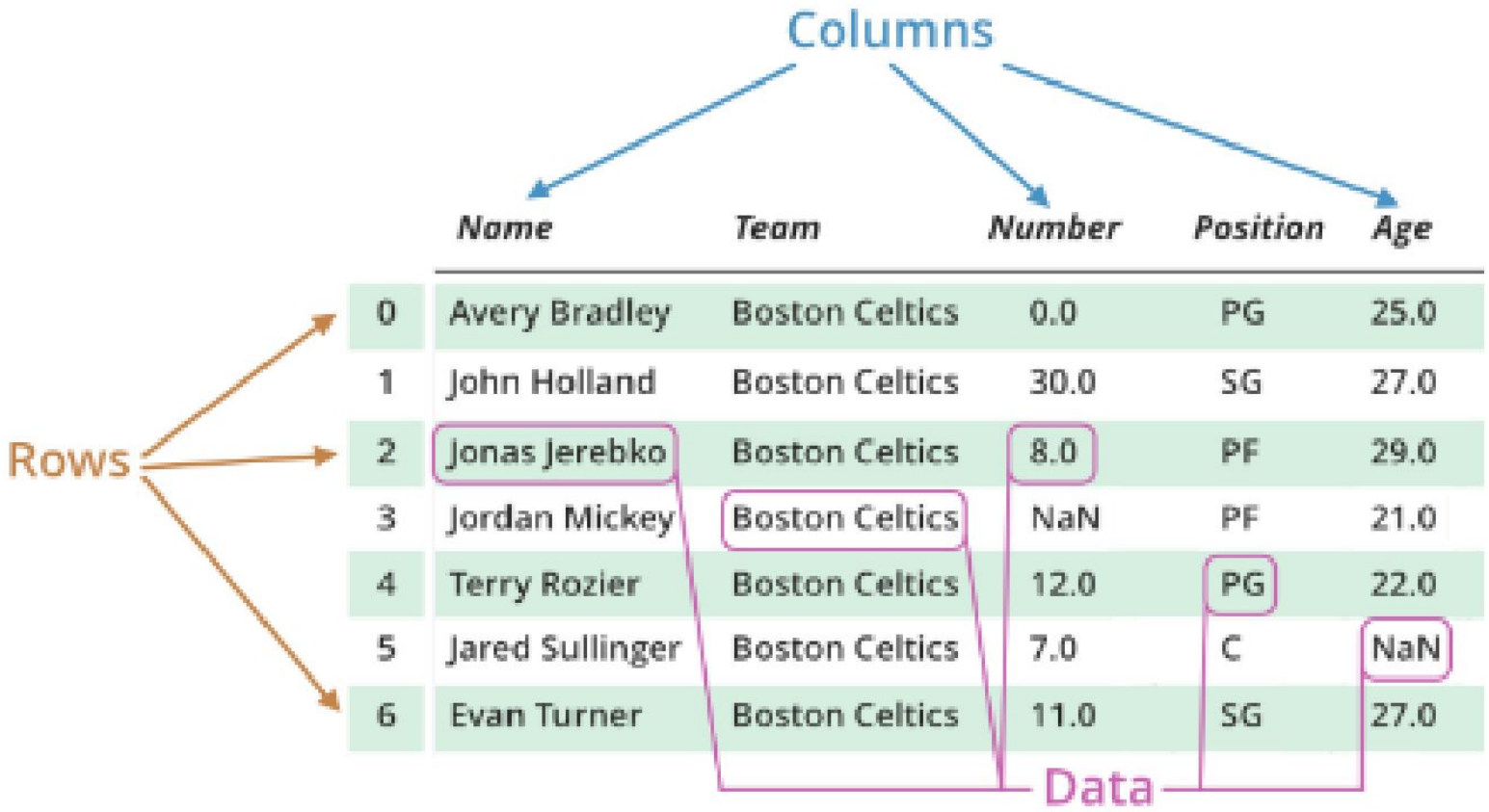
```
frame3.T
```

Possible data inputs to DataFrame constructor

Type	Notes
2D ndarray	A matrix of data, passing optional row and column labels
dict of arrays, lists, or tuples	Each sequence becomes a column in the DataFrame; all sequences must be the same length
NumPy structured/record array	Treated as the "dict of arrays" case
dict of Series	Each value becomes a column; indexes from each Series are unioned together to form the result's row index if no explicit index is passed
dict of dicts	Each inner dict becomes a column; keys are unioned to form the row index as in the "dict of Series" case
List of dicts or Series	Each item becomes a row in the DataFrame; union of dict keys or Series indexes become the DataFrame's column labels
List of lists or tuples	Treated as the "2D ndarray" case
Another DataFrame	The DataFrame's indexes are used unless different ones are passed
NumPy MaskedArray	Like the "2D ndarray" case except masked values become NA/missing in the DataFrame result

Working with CSV files pandas

```
import pandas as pd
df = pd.read_csv('sample.csv', encoding= 'unicode_escape')
print(df.to_string())
```



	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Index Objects

- pandas's Index objects are responsible for holding the axis labels and other metadata (like the axis name or names). Any array or other sequence of labels you use when constructing a Series or DataFrame is internally converted to an Index:

```
obj = pd.Series(range(3), index=['a', 'b', 'c'])
```

```
Index
```

Each Index has a number of methods and properties

Method	Description
`append`	Concatenate with additional Index objects, producing a new Index
`difference`	Compute set difference as an Index
`intersection`	Compute set intersection
`union`	Compute set union
`isin`	Compute boolean array indicating whether each value is contained in the passed collection
`delete`	Compute new Index with element at index `i` deleted
`drop`	Compute new Index by deleting passed values
`insert`	Compute new Index by inserting element at index `i`
`is_monotonic`	Returns `True` if each element is greater than or equal to the previous element
`is_unique`	Returns `True` if the Index has no duplicate values
`unique`	Compute the array of unique values in the Index

Selection with loc and iloc

special indexing operators loc and iloc. They enable you to select a subset of the rows and columns from a DataFrame

select a single row and multiple columns by label:

```
data = pd.DataFrame(np.arange(16).reshape((4, 4)), index=['Ohio', 'Colorado', 'Utah', 'New York'],columns=['one', 'two', 'three', 'four'])
data.loc['Colorado', ['two', 'three']]
```

Type	Notes
`df[val]`	Select single column or sequence of columns from the DataFrame; special case conveniences: boolean array (filter rows), slice (slice rows), or boolean DataFrame (set values based on some criterion)
`df.loc[val]`	Selects single row or subset of rows from the DataFrame by label
`df.loc[:, val]`	Selects single column or subset of columns by label
`df.loc[val1, val2]`	Select both rows and columns by label
`df.iloc[where]`	Selects single row or subset of rows from the DataFrame by integer position

DataFrame functions

- abs () Return a Series/ DataFrame with absolute numeric value of each element.

```
>>> s = pd.Series([-1.10, 2, -3.33, 4])
>>> s.abs()
0    1.10
1    2.00
2    3.33
3    4.00
dtype: float64
```

Checking for missing values using isnull() and notnull() :

```
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}
# creating a dataframe from list
df = pd.DataFrame(dict)
# using isnull() function
df.isnull()
```

	First Score	Second Score	Third Score
0	False	False	True
1	False	False	False
2	True	False	False
3	False	True	False

Dropping missing values using dropna() :

```
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, np.nan, 45, 56],
        'Third Score':[52, 40, 80, 98],
        'Fourth Score':[np.nan, np.nan, np.nan, 65]}
df = pd.DataFrame(dict)
df.dropna()
```

- Agg fun([axis]) Aggregate using one or more operations over the specified axis.

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.agg.html#pandas.DataFrame.agg

```
>>> df = pd.DataFrame([[1, 2, 3],
...                    [4, 5, 6],
...                    [7, 8, 9],
...                    [np.nan, np.nan, np.nan]],
...                   columns=['A', 'B', 'C'])
```

Aggregate these functions over the rows.

```
>>> df.agg(['sum', 'min'])
        A     B     C
sum  12.0  15.0  18.0
min   1.0   2.0   3.0
```

Different aggregations per column.

```
>>> df.agg({'A' : ['sum', 'min'], 'B' : ['min', 'max']})
        A    B
sum  12.0  NaN
min   1.0  2.0
max   NaN  8.0
```