EDUNET FOUNDATION-Class Exrercise Notebook

LAB 9 - Implementing Matplotlib Concepts in Python

Matplotlib Basics

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

import matplotlib.pyplot as plt

Now the Pyplot package can be referred to as plt.

Draw a line in a diagram from position (0,0) to position (6,250):

Plotting x and y points The `plot( )` function is used to draw points (markers) in a diagram.

By default, the `plot()` function draws a line from point to point.

The function takes parameters for specifying points in the diagram.

Parameter 1 is an array containing the points on the x-axis.

Parameter 2 is an array containing the points on the y-axis.

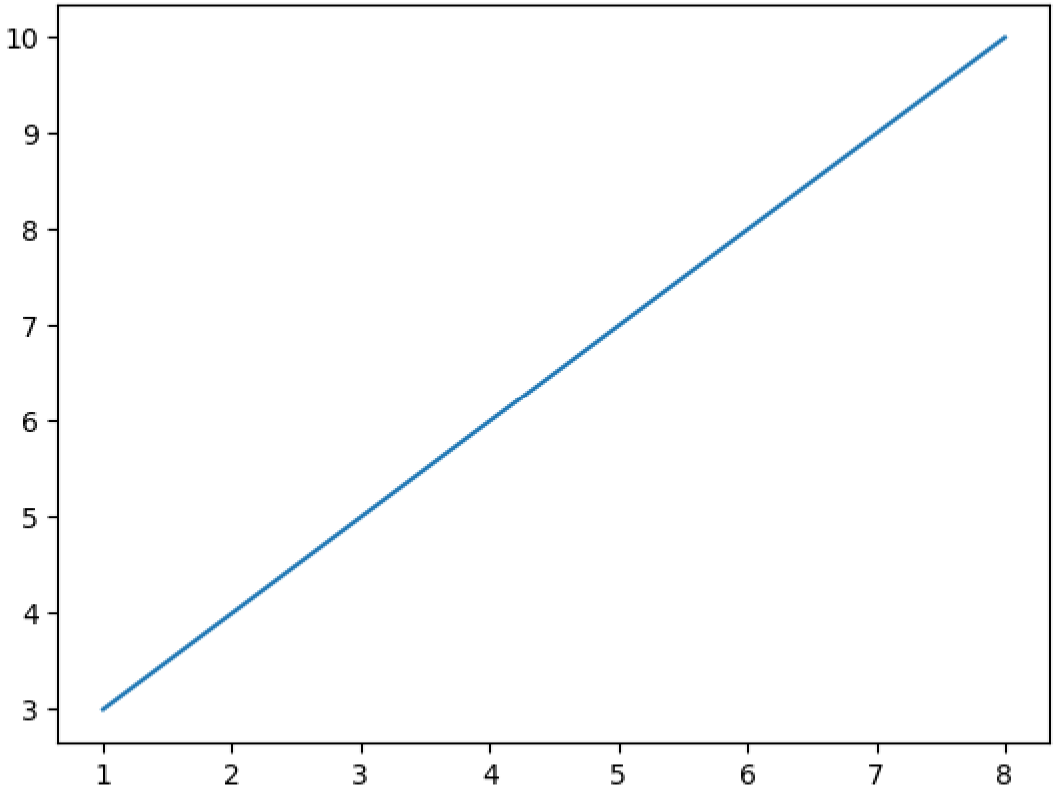
If we need to plot a line from (1, 3) to (8, 10), we have to pass two arrays [1, 8] and [3, 10] to the plot function.

In [1]:
```
pip install matplotlib
```

```
Markdown                      3.4.1
MarkupSafe                    2.1.1
matplotlib                    3.7.0
matplotlib-inline             0.1.6
mccabe                        0.7.0
menuinst                      1.4.19
mistune                       0.8.4
mkl-fft                       1.3.1
mkl-random                    1.2.2
mkl-service                   2.4.0
mock                          4.0.3
mpmath                        1.2.1
msgpack                       1.0.3
multidict                     6.0.4
multipledispatch              0.6.0
multitasking                  0.0.11
munkres                       1.1.4
murmurhash                    1.0.10
mypy-extensions               0.4.3
mysql-connector-python        8.1.0
```

In [2]:

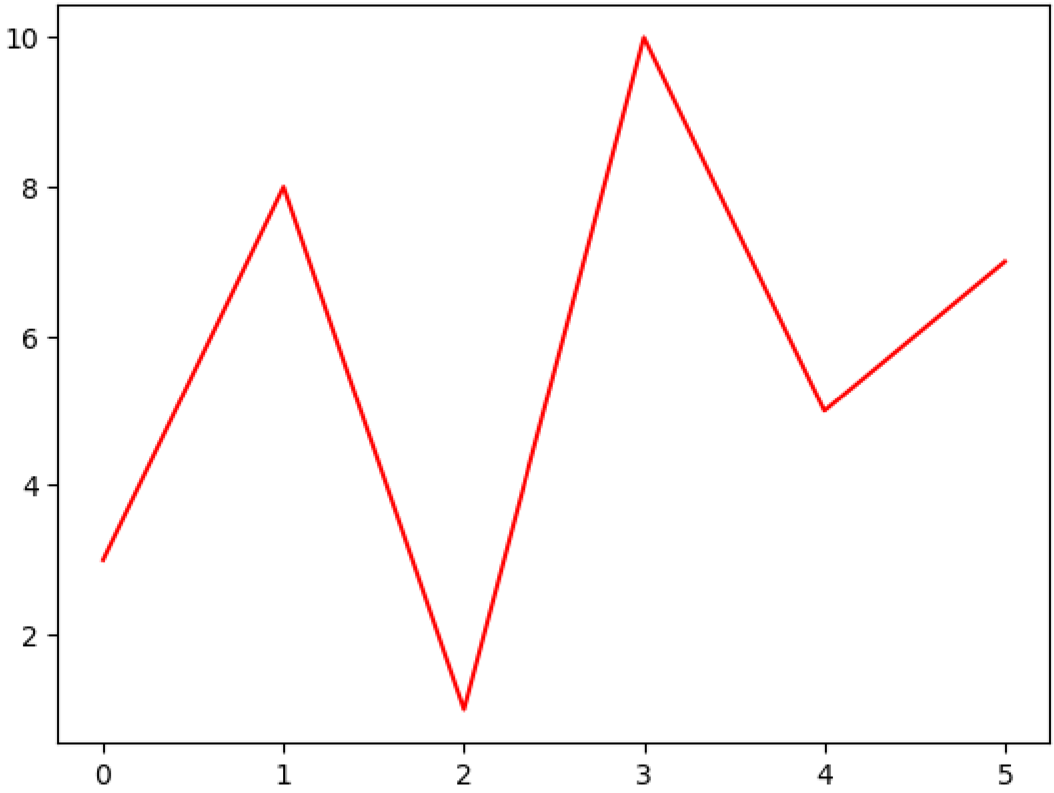
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([1, 8])
ypoints = np.array([3, 10])

plt.plot(xpoints, ypoints)
plt.show()
```



Multiple Points You can plot as many points as you like, just make sure you have the same number of points in both axis.

Draw a line in a diagram from position (1, 3) to (2, 8) then to (6, 1) and finally to position (8, 10):

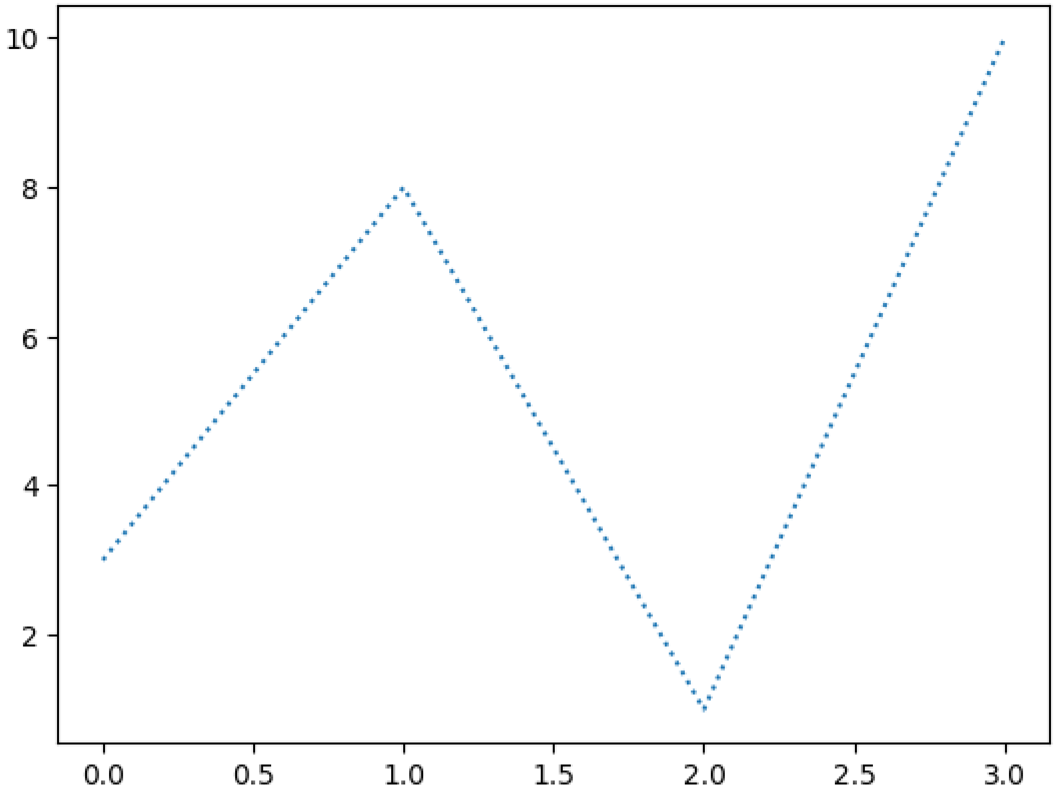
In [3]:
```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10, 5, 7])

plt.plot(ypoints, c='red')
plt.show()
```



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

In [27]:

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

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, linestyle = 'dotted')
plt.show()
```

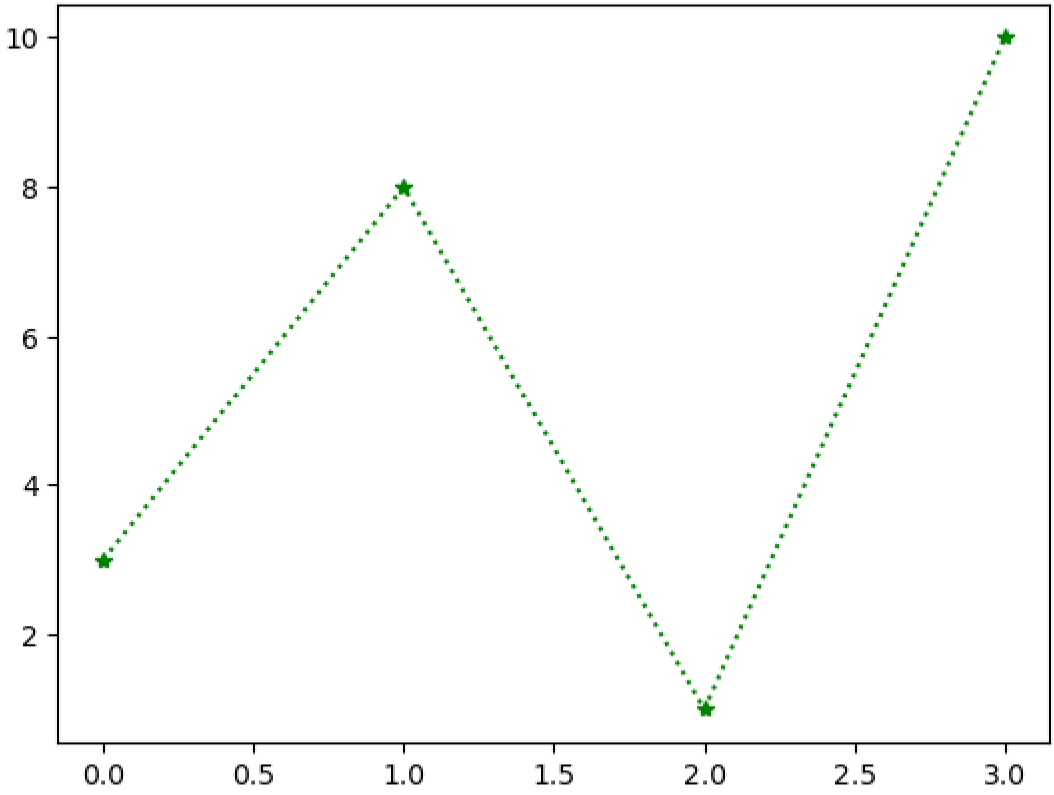


Format Strings fmt You can also use the shortcut string notation parameter to specify the marker.

This parameter is also called fmt, and is written with this syntax:

marker|line|color

In [7]:
```
import matplotlib.pyplot as plt
import numpy as np

ypoints = np.array([3, 8, 1, 10])

plt.plot(ypoints, '*:g')
plt.show()
```

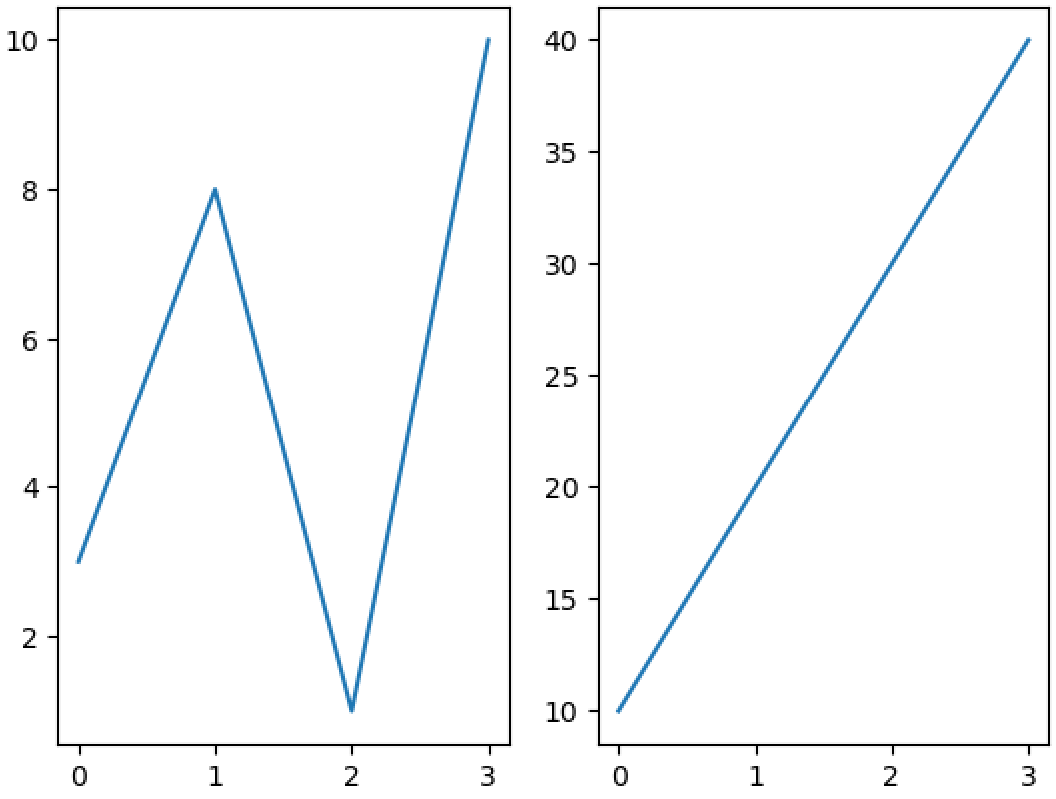


Display Multiple Plots

With the `subplot()` function you can draw multiple plots in one figure:

In [8]:

```
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()
```



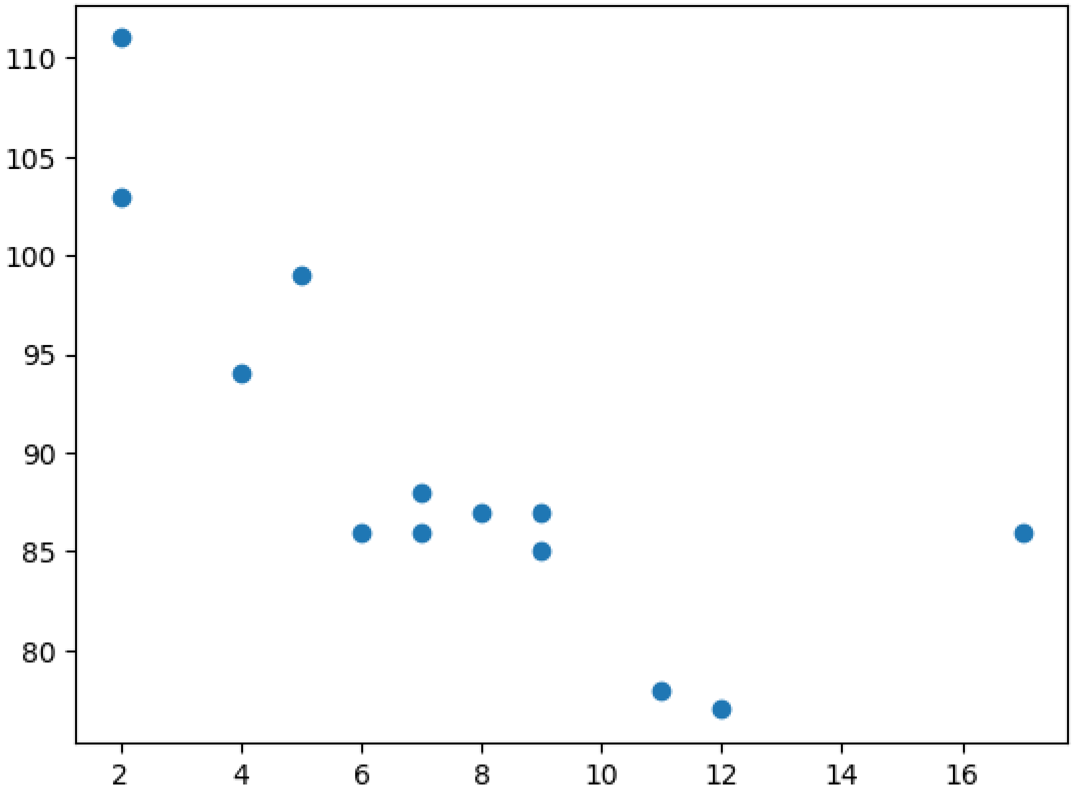
Creating Scatter Plots

With Pyplot, you can use the `scatter()` function to draw a scatter plot.

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:

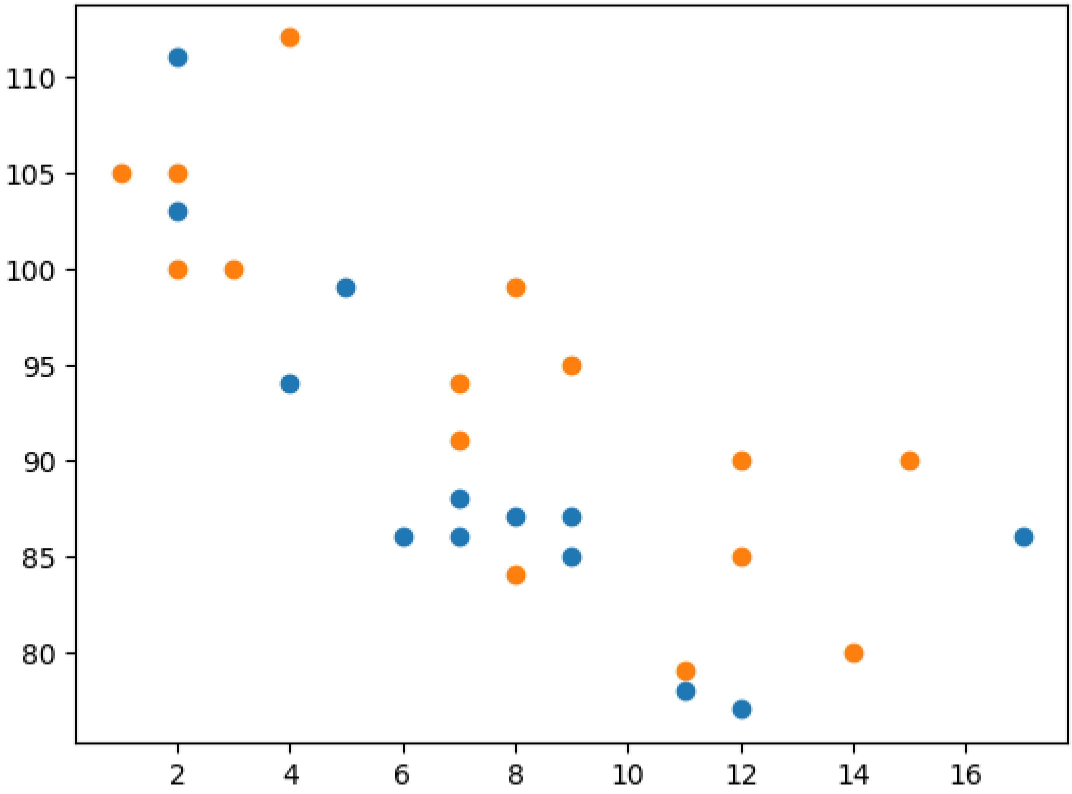
In [30]:

```
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])

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



In [31]:

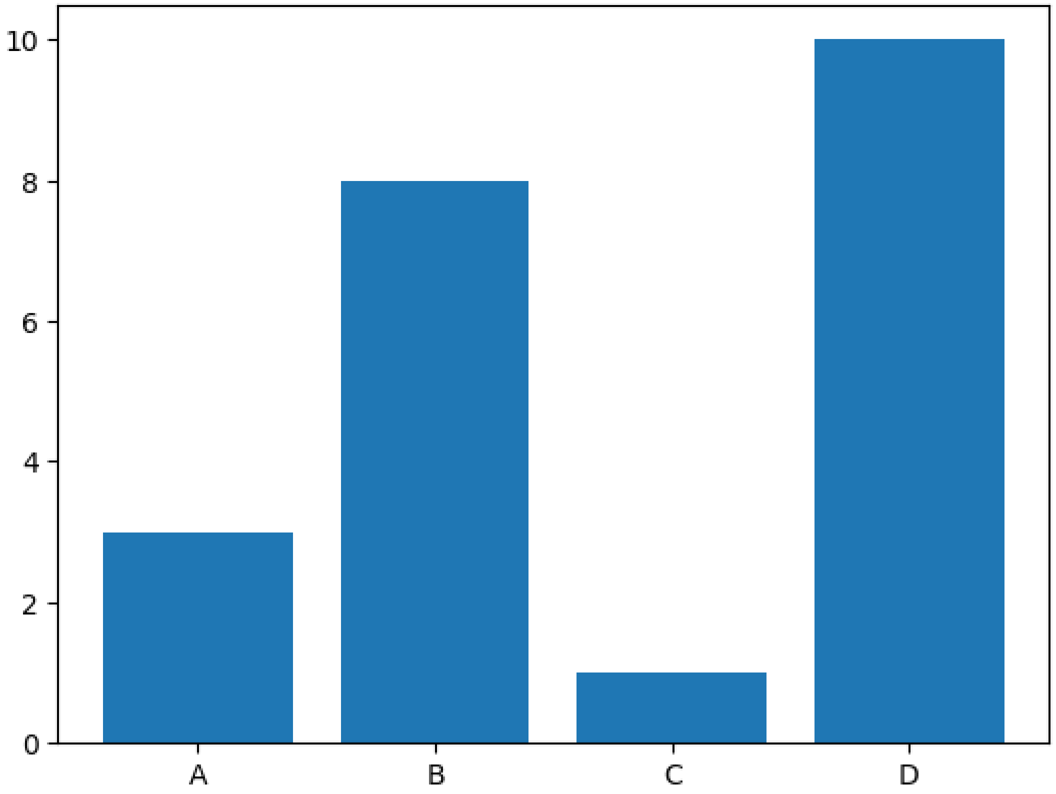
```
import matplotlib.pyplot as plt
import numpy as np

#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)

plt.show()
```



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

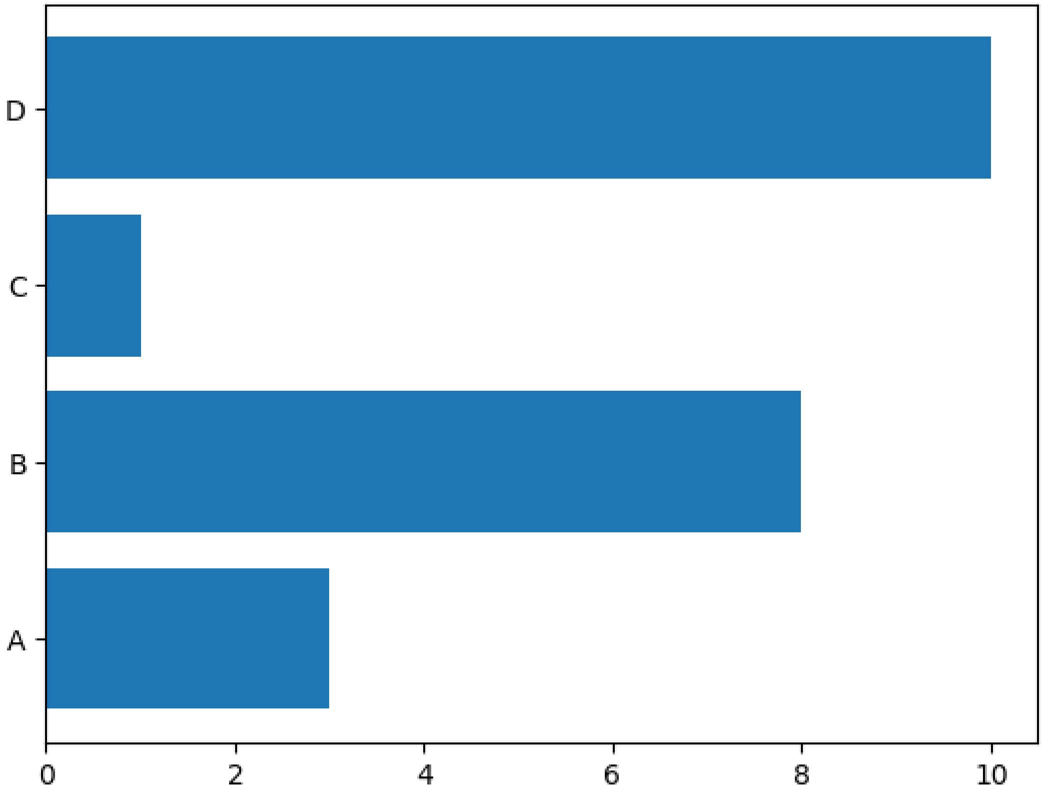
In [32]:

```
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()
```



In [33]:

```
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)
plt.show()
```

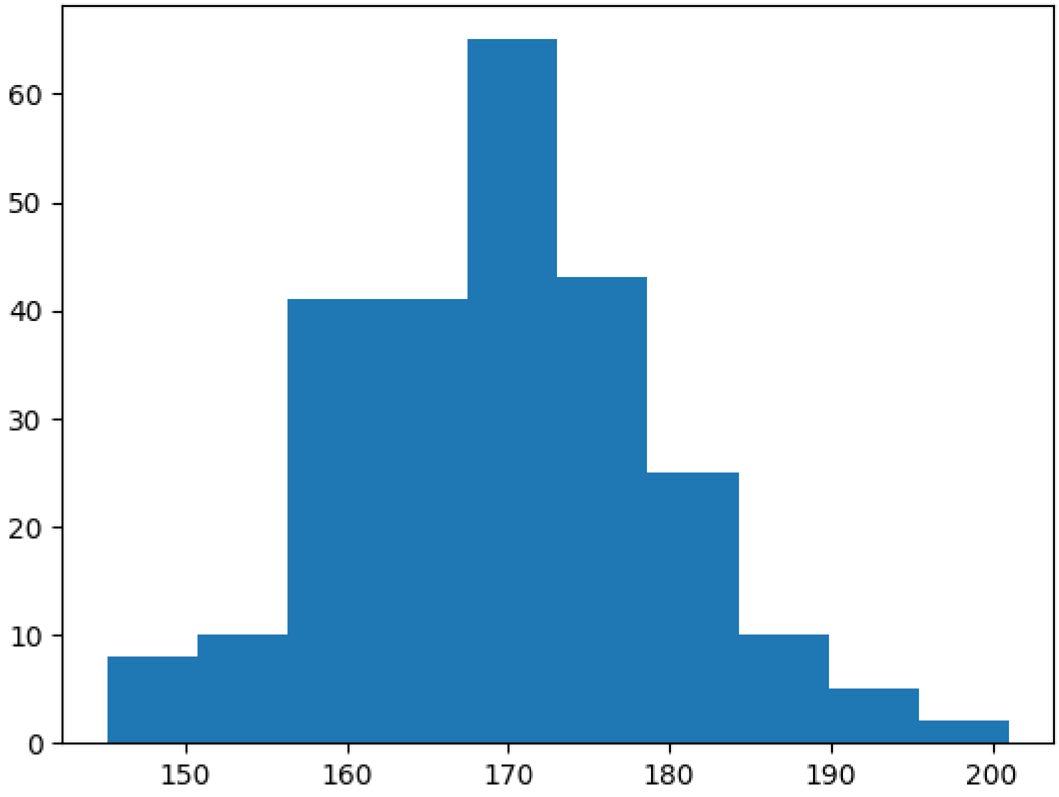


Histogram

A histogram is a graph showing frequency distributions. It is a graph showing the number of observations within each given interval. Example: Say you ask for the height of 250 people, you might end up with a histogram like this:

In [34]:

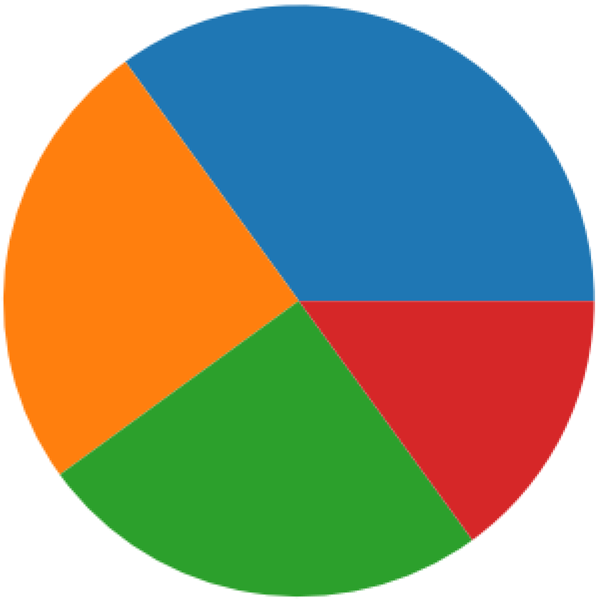
```
import matplotlib.pyplot as plt
import numpy as np

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

plt.hist(x)
plt.show()
```



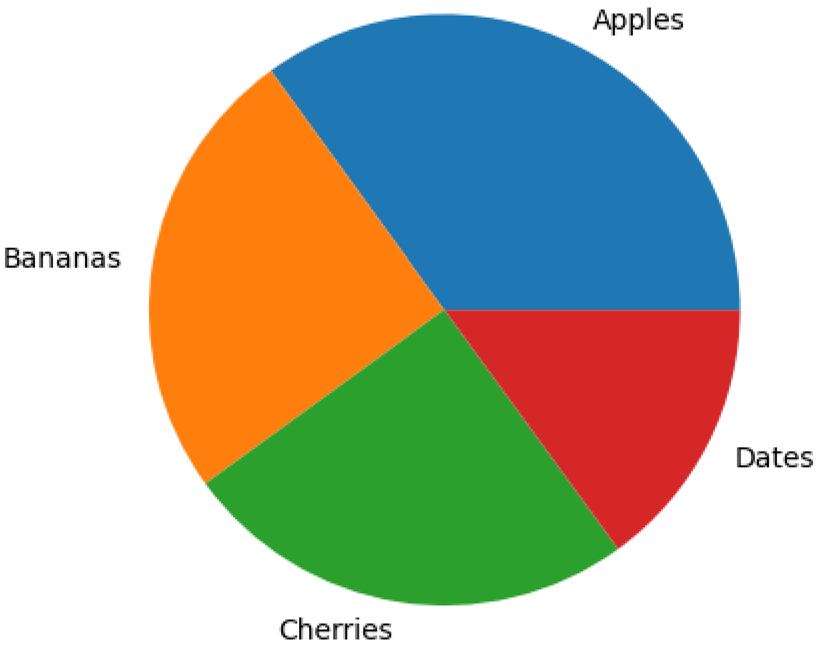
Creating Pie Charts

With Pyplot, you can use the `pie()` function to draw pie charts:

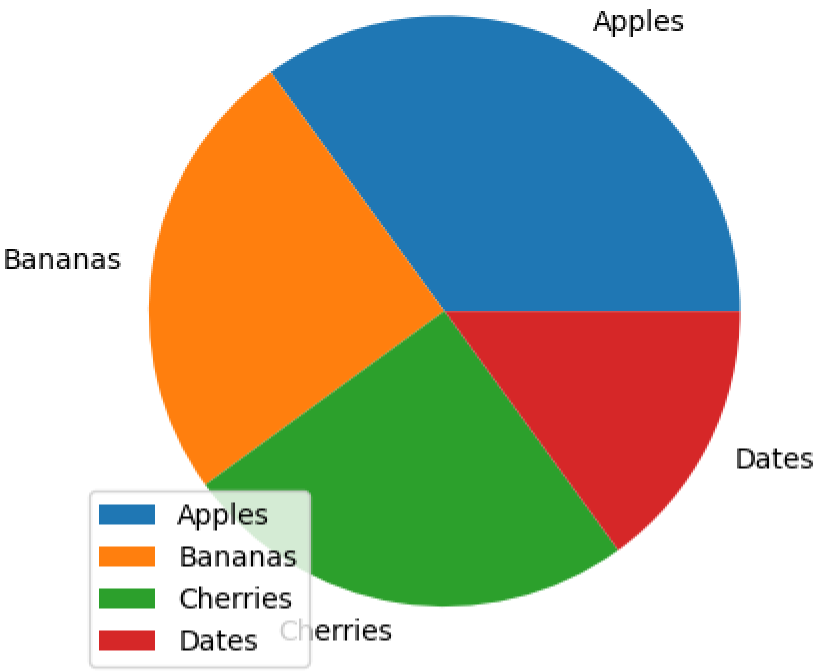
In [35]:
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```



In [36]:
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.show()
```



In [37]:
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.legend()
plt.show()
```



Pandas Introduction

What is Pandas?

Pandas is a Python library used for working with data sets.

It has functions for analyzing, cleaning, exploring, and manipulating data.

The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.

Why Use Pandas?

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.

Create an alias with the as keyword while importing:

import pandas as pd Now the Pandas package can be referred to as pd instead of pandas.

In [38]:
```
import pandas

mydataset = {
  'cars': ["BMW", "Volvo", "Ford"],
  'passings': [3, 7, 2]
}

myvar = pandas.DataFrame(mydataset)

print(myvar)
```

```
    cars  passings
0    BMW         3
1  Volvo         7
2   Ford         2
```

What is a Series?

A Pandas Series is like a column in a table. It is a one-dimensional array holding data of any type.

In [39]:
```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

```
x    1
y    7
z    2
dtype: int64
```

What is a DataFrame?

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

In [40]:
```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

```
   calories  duration
0       420        50
1       380        40
2       390        45
```

Named Indexes

With the index argument, you can name your own indexes.

In [41]:

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df.loc[0])
```

```
calories    420
duration     50
Name: 0, dtype: int64
```

Read CSV Files

A simple way to store big data sets is to use CSV files (comma separated files). CSV files contains plain text and is a well know format that can be read by everyone including Pandas. In our examples we will be using a CSV file called 'data.csv'.

Info About the Data The DataFrames object has a method called `info()` , that gives you more information about the data set.

In [47]:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Duration  169 non-null    int64  
 1   Pulse     169 non-null    int64  
 2   Maxpulse  169 non-null    int64  
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

In [ ]:

In [43]:
```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.to_string())
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
5          60    102       127     300.0
6          60    110       136     374.0
7          45    104       134     253.3
8          30    109       133     195.1
9          60     98       124     269.0
10         60    103       147     329.3
11         60    100       120     250.7
12         60    106       128     345.3
13         60    104       132     379.3
14         60     98       123     275.0
15         60     98       120     215.2
16         60    100       120     300.0
17         45     90       112       NaN
```

Viewing the Data
One of the most used method for getting a quick overview of the DataFrame, is the head() method.

The `head()` method returns the headers and a specified number of rows, starting from the top.

In [44]:
```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.head(10))
```

```
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
2        60    103       135     340.0
3        45    109       175     282.4
4        45    117       148     406.0
5        60    102       127     300.0
6        60    110       136     374.0
7        45    104       134     253.3
8        30    109       133     195.1
9        60     98       124     269.0
```

Remove Rows
One way to deal with empty cells is to remove rows that contain empty cells.

This is usually OK, since data sets can be very big, and removing a few rows will not have a big impact on the result.

In [45]:

```
import pandas as pd

df = pd.read_csv('data.csv')

new_df = df.dropna()

print(new_df.to_string())
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
5          60    102       127     300.0
6          60    110       136     374.0
7          45    104       134     253.3
8          30    109       133     195.1
9          60     98       124     269.0
10         60    103       147     329.3
11         60    100       120     250.7
12         60    106       128     345.3
13         60    104       132     379.3
14         60     98       123     275.0
15         60     98       120     215.2
16         60    100       120     300.0
18         60    103       123     323.0
```

f you want to change the original DataFrame, use the inplace = True argument:

In [46]:

```
import pandas as pd

df = pd.read_csv('data.csv')

df.dropna(inplace = True)

print(df.to_string())
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
5          60    102       127     300.0
6          60    110       136     374.0
7          45    104       134     253.3
8          30    109       133     195.1
9          60     98       124     269.0
10         60    103       147     329.3
11         60    100       120     250.7
12         60    106       128     345.3
13         60    104       132     379.3
14         60     98       123     275.0
15         60     98       120     215.2
16         60    100       120     300.0
18         60    103       123     323.0
```

In [48]:
```python
import pandas as pd

df = pd.read_csv('data.csv')

new_df = df.dropna()

print(new_df.to_string())
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
5          60    102       127     300.0
6          60    110       136     374.0
7          45    104       134     253.3
8          30    109       133     195.1
9          60     98       124     269.0
10         60    103       147     329.3
11         60    100       120     250.7
12         60    106       128     345.3
13         60    104       132     379.3
14         60     98       123     275.0
15         60     98       120     215.2
16         60    100       120     300.0
18         60    103       123     323.0
```

Replacing Values

One way to fix wrong values is to replace them with something else.

In our example, it is most likely a typo, and the value should be "45" instead of "450", and we could just insert "45" in row 7:

In [51]:
```
import pandas as pd

df = pd.read_csv('data.csv')

df.loc[7,'Duration'] = 45

print(df.to_string())
```

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
5          60    102       127     300.0
6          60    110       136     374.0
7          45    104       134     253.3
8          30    109       133     195.1
9          60     98       124     269.0
10         60    103       147     329.3
11         60    100       120     250.7
12         60    106       128     345.3
13         60    104       132     379.3
14         60     98       123     275.0
15         60     98       120     215.2
16         60    100       120     300.0
17         45     90       112       NaN
18         60    103       123     323.0
```

In [52]:
```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.duplicated())
```

```
0      False
1      False
2      False
3      False
4      False
       ...  
164    False
165    False
166    False
167    False
168    False
Length: 169, dtype: bool
```

Finding Relationships

A great aspect of the Pandas module is the `corr()` method. The `corr( )` method calculates the relationship between each column in your data set. The examples in this page uses a CSV file called: 'data.csv'.

In [53]:

```
import pandas as pd

df = pd.read_csv('data.csv')

print(df.corr())
```

```
          Duration     Pulse  Maxpulse  Calories
Duration  1.000000 -0.155408  0.009403  0.922717
Pulse    -0.155408  1.000000  0.786535  0.025121
Maxpulse  0.009403  0.786535  1.000000  0.203813
Calories  0.922717  0.025121  0.203813  1.000000
```

Result Explained The Result of the `corr()` method is a table with a lot of numbers that represents how well the relationship is between two columns.

The number varies from -1 to 1.

1 means that there is a 1 to 1 relationship (a perfect correlation), and for this data set, each time a value went up in the first column, the other one went up as well.

0.9 is also a good relationship, and if you increase one value, the other will probably increase as well.

-0.9 would be just as good relationship as 0.9, but if you increase one value, the other will probably go down.

0.2 means NOT a good relationship, meaning that if one value goes up does not mean that the other will.

In [ ]:

```

```