Week 9 Matplotlib

Part 1- Intro to Matplotlib

Exercise 1 - Line Graph

1,2,3,4

In [3]:

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

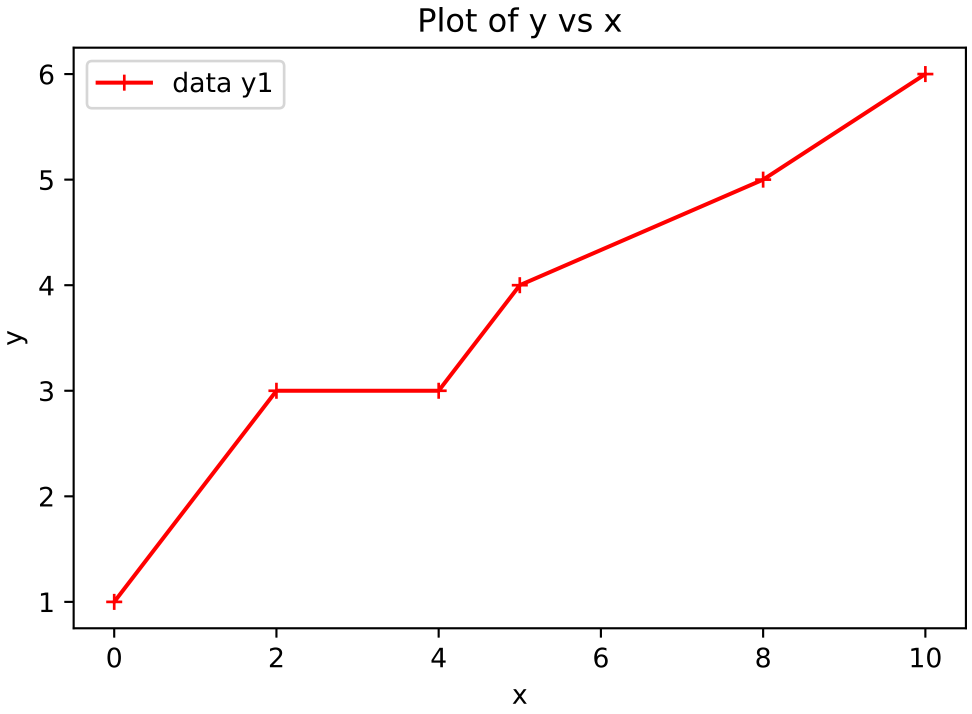
In [4]:

```
x = [0,2,4,5,8,10]
y = [1,3,3,3,4,5,6]
plt.plot(x,y) #error dimensions must agree
```

```
---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
<ipython-input-4-b514837aa0a2> in <module>
      1 x = [0,2,4,5,8,10]
      2 y = [1,3,3,3,4,5,6]
----> 3 plt.plot(x,y)

~\miniconda3\lib\site-packages\matplotlib\pyplot.py in plot(scalex, scale
y, data, *args, **kwargs)
   2838 @_copy_docstring_and_deprecators(Axes.plot)
   2839 def plot(*args, scalex=True, scaley=True, data=None, **kwargs):
-> 2840     return gca().plot(
   2841         *args, scalex=scalex, scaley=scaley,
   2842         **({"data": data} if data is not None else {}), **kwargs)

~\miniconda3\lib\site-packages\matplotlib\axes\_axes.py in plot(self, scal
ex, scaley, data, *args, **kwargs)
   1741         """
   1742         kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1743         lines = [*self._get_lines(*args, data=data, **kwargs)]
   1744         for line in lines:
   1745             self.add_line(line)

~\miniconda3\lib\site-packages\matplotlib\axes\_base.py in __call__(self,
 data, *args, **kwargs)
    271                 this += args[0],
    272                 args = args[1:]
--> 273             yield from self._plot_args(this, kwargs)
    274 
    275     def get_next_color(self):

~\miniconda3\lib\site-packages\matplotlib\axes\_base.py in _plot_args(sel
f, tup, kwargs)
    397 
    398         if x.shape[0] != y.shape[0]:
--> 399             raise ValueError(f"x and y must have same first dimens
ion, but "
    400                              f"have shapes {x.shape} and {y.shap
e}")
    401         if x.ndim > 2 or y.ndim > 2:

ValueError: x and y must have same first dimension, but have shapes (6,) a
nd (7,)
```

In [10]:

```
x = [0,2,4,5,8,10]
y = [1,3,3,4,5,6]
plt.plot(x,y,'r+-',label = "data y1")
plt.xlabel("x")
plt.ylabel("y")
plt.title("Plot of y vs x")
plt.legend()
plt.show()
```



For more options on plt.legend(), including positioning on the graph, please refer to the docs link can be found here (https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.legend.html)

Part 2 Saving plot and importing

Exercise 2 Histogram and csv

1,2

Below are two implementations of the function DiceRolls one returning an array with values between 1 and 6 for a specified array of length n and the other saving the result of the dice rolls in a CSV file.
More information on CSV files can be found here (https://en.wikipedia.org/wiki/Comma-separated_values)

In [4]:

```
def DiceRolls(n):
   return np.random.randint(1,7,n)
print(DiceRolls(10))
```

```
[5 2 1 4 6 4 5 1 3 5]
```

In [5]:

```
def DiceRollsCSV(n):
    with open('diceRolls.csv',mode='w') as file:
        write = csv.writer(file, delimiter = ',')# setting the delimeter to comma in ac
cordance with the csv format
        write.writerow(np.random.randint(1,7,n))
```

In [59]:

```
DiceRollsCSV(100000)
```

3,4

All the csv data is on 1 row so the for loop isn't really necessary; however, after the writerow operation we performed earlier the "cursor" has been moved to a new row we need to check if the row isn't empty to avoid adding an empty array to our data.
This data is in string format the map function allows us to typecast the entire list to integers

In [60]:

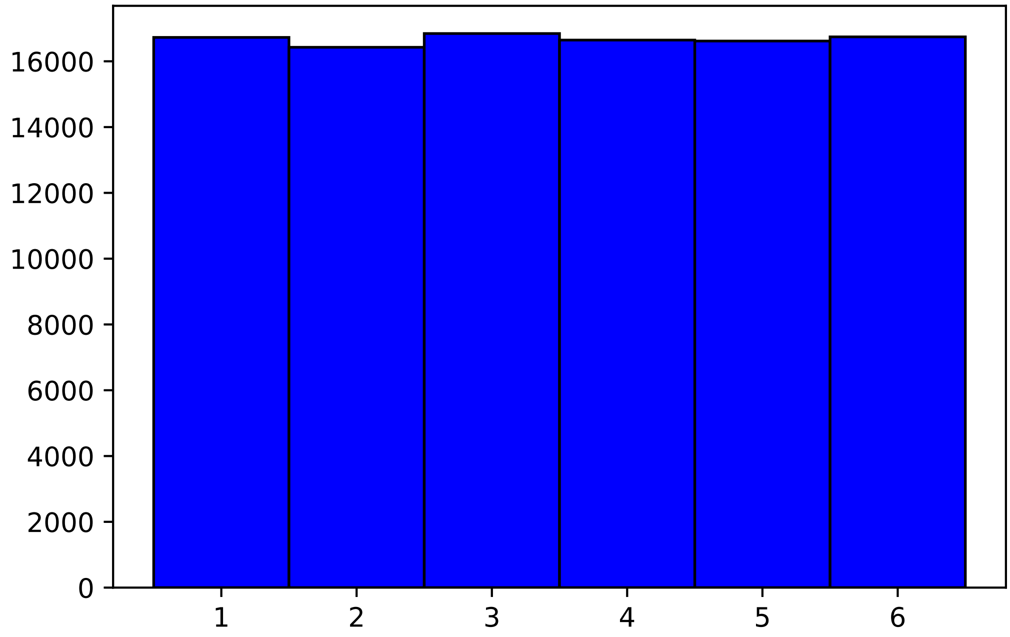
```
data = []
with open('diceRolls.csv', newline='') as csvfile:
    file = csv.reader(csvfile, delimiter=',')
    for row in file:
        if len(row) != 0:
            data = row
data = list(map(int,data))
```

In [61]:

```
plt.hist(data, bins=np.arange(1, 8), histtype = 'bar', rwidth=1, facecolor = 'blue', ed
gecolor="k",align ='left')
```

Out[61]:

```
(array([16726., 16426., 16844., 16645., 16616., 16743.]),
 array([1, 2, 3, 4, 5, 6, 7]),
 <BarContainer object of 6 artists>)
```



As expected the number distribution of dice rolls should approach a uniform distribution when more trials are performed. The number of bins and the align parameter allows us to center the graph neatly. You can play around with the setings and see how the graph display changes

Part 3 Curve Fitting

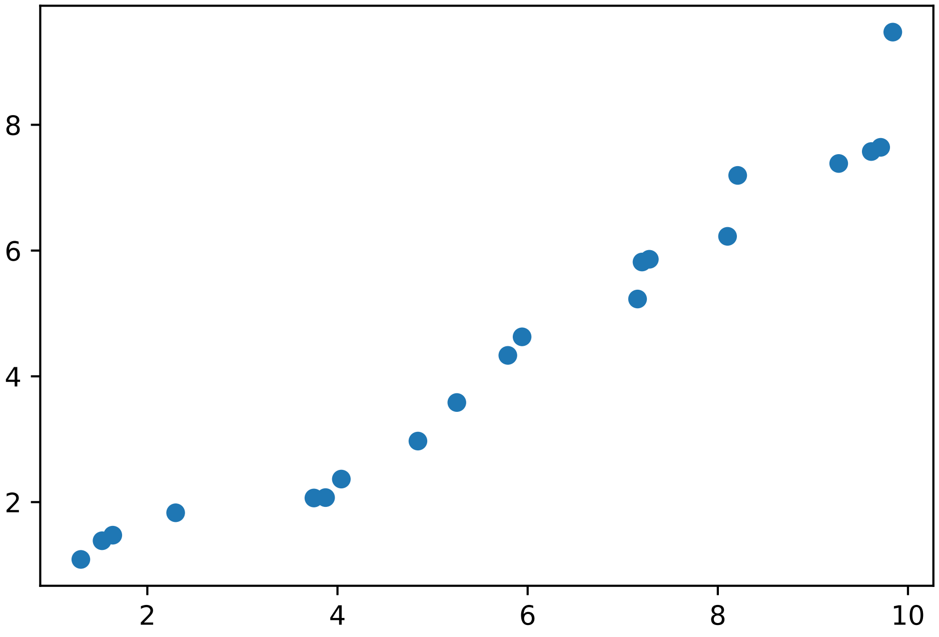
1,2,3,4,5

In [31]:

```
x = np.random.uniform(1.0,10,20)
y = np.random.uniform(1.0,10,20)
plt.scatter(np.sort(x),np.sort(y))
```

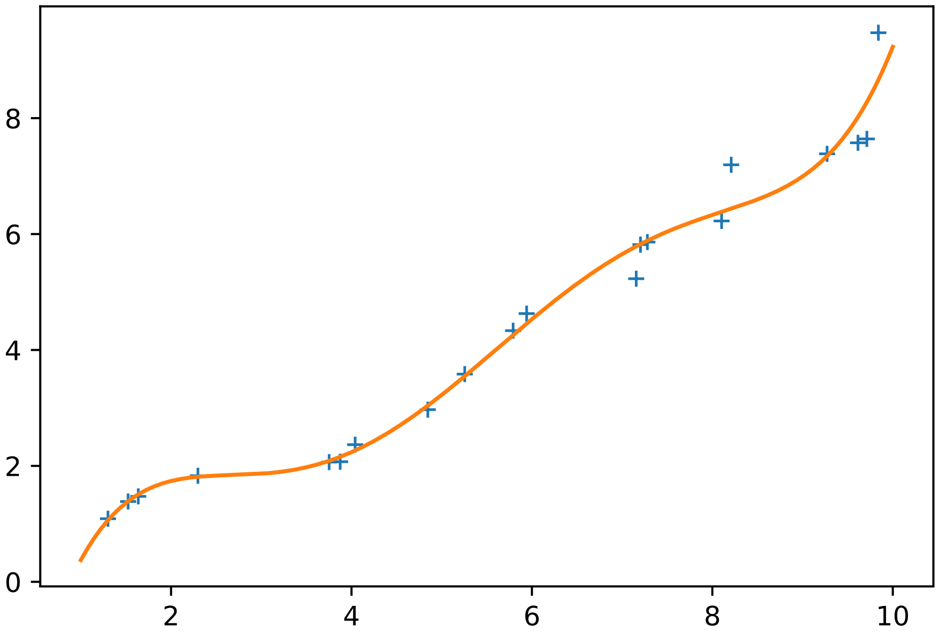
Out[31]:

```
<matplotlib.collections.PathCollection at 0x2c2bb1c9220>
```



In [32]:

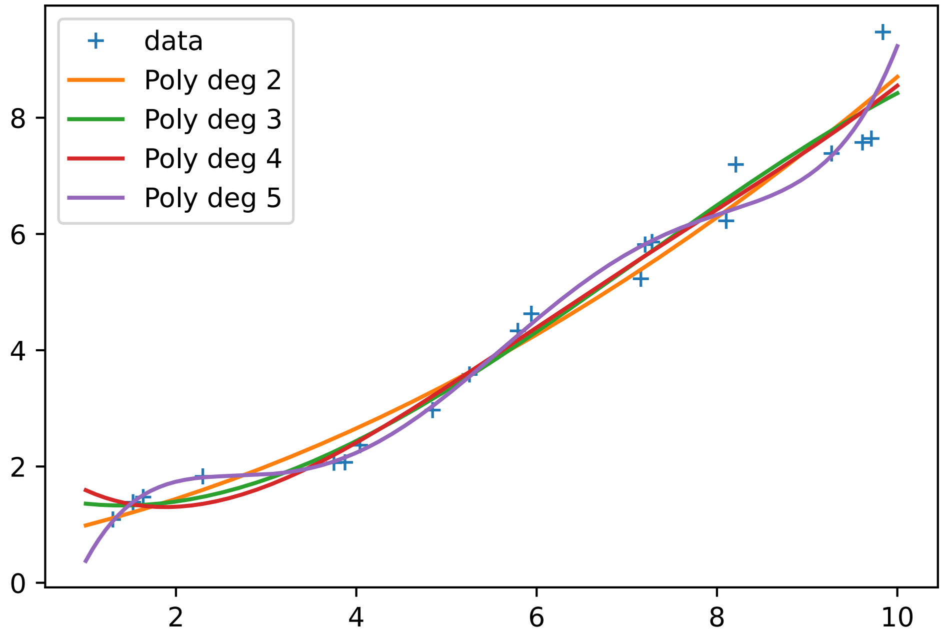
```
coefs  = np.polyfit(np.sort(x),np.sort(y),5) # a,b,c ...
x_range = np.linspace(1,10,500)
y_range = np.poly1d(coefs)(x_range)
plt.plot(np.sort(x),np.sort(y),'+',x_range,y_range)
plt.show()
```



Once you have obtained coefficents for the polynomial applying it to more points allows you to have a smoother line.
You may have noticed that increasing the degree of the polynomial makes for a curve approximating more points.This means that the standard deviation between the line and data points is low but does it represent the data accurately ? For those of you who will continue on and study data science you will find that this becomes a case of overfitting (https://en.wikipedia.org/wiki/Overfitting#:~:text=In%20statistics%2C%20overfitting%20is%20%22the,or%20pre

In [33]:

```
plt.plot(np.sort(x),np.sort(y),'+',label = "data")
for i in range(2,6):
    coefs  = np.polyfit(np.sort(x),np.sort(y),i) # a,b,c ...
    x_range = np.linspace(1,10,500)
    y_range = np.poly1d(coefs)(x_range)
    plt.plot(x_range,y_range, label = "Poly deg "+str(i))
    plt.legend()
plt.show()
```



In [ ]: