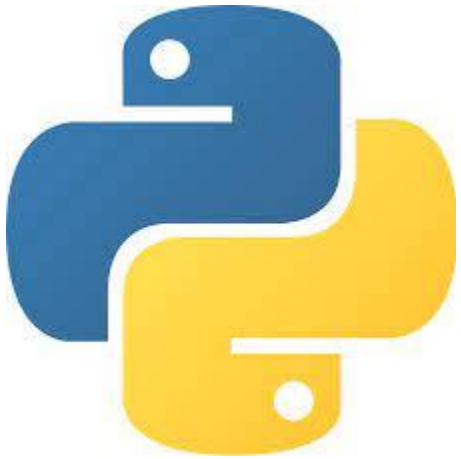
Data Processing and Visualizations using Python

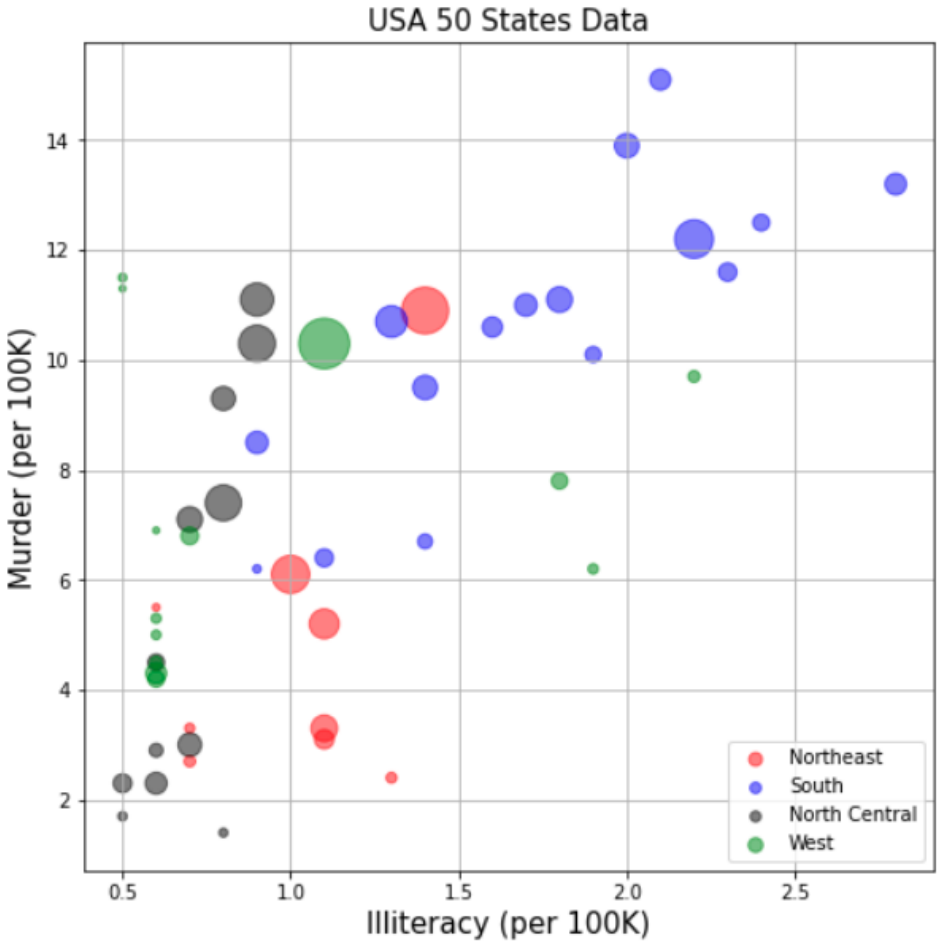
Day 2 – Basic Visualization

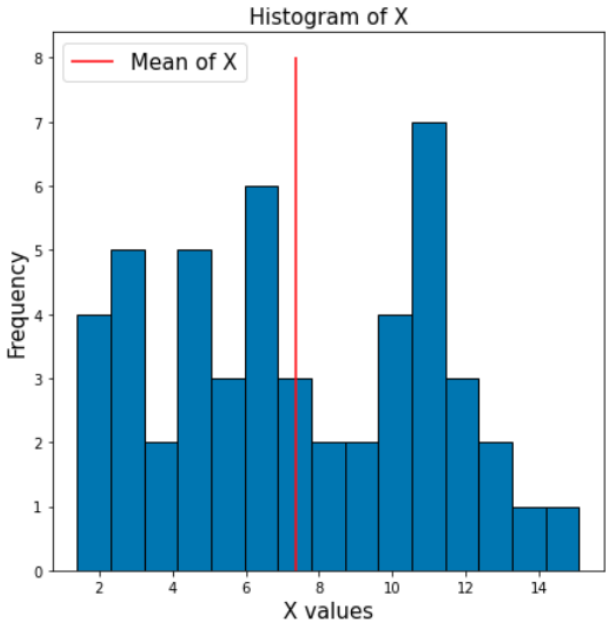


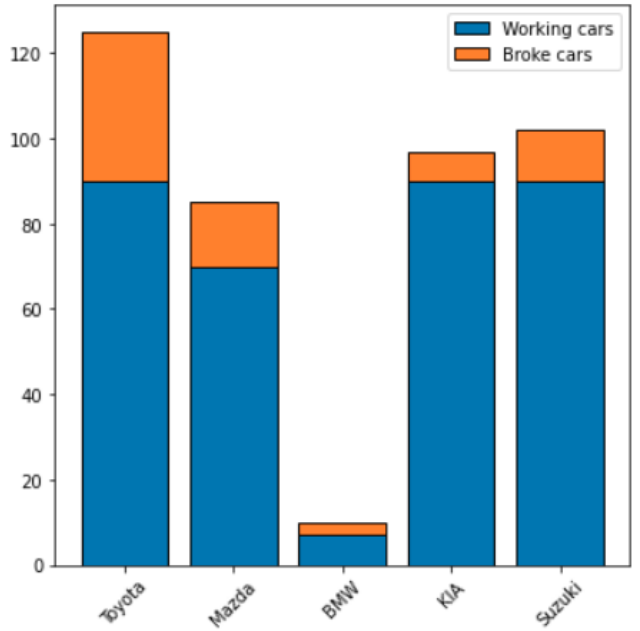
SICSS 2022 – Haifa University

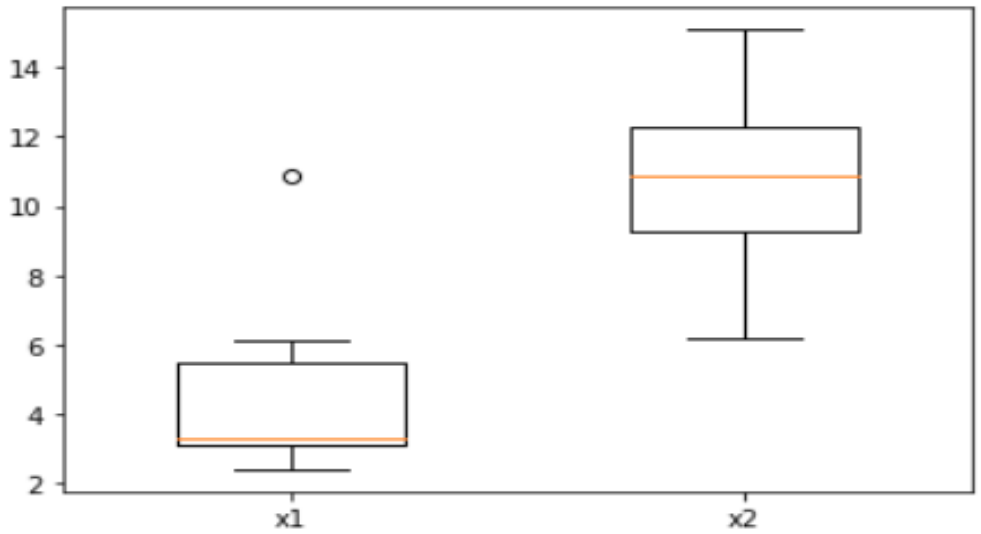
Amit Donner

Visualization









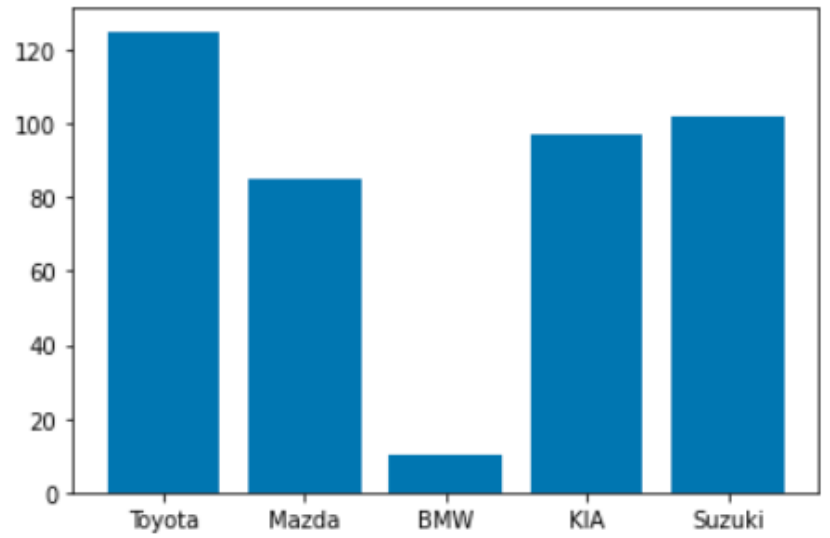
- Maybe the strongest tool we have in terms of explaining and communicating our data/results.
- A picture is worth a thousand words.
- A good plot should has a clear message that pops out immediately.
- Using the *matplotlib.pyplot* module, we will explore and compare different visualization methods for numerical (discrete and continuous) and categorical data, 2-dimensional data (and even 3 and 4 dimensional).
- Later, we will visually demonstrate 2 important results in Statistics – The *Central Limit Theorem* and the *Law of Large Numbers*

Discrete and Categorical Data

- Variables such as *Hair color (Categorical)*, *Number of kids per household (Numerical)*, etc. are usually visualized similarly using **bar plots** and **pie charts***.
- For example, if my data consists of counts of different cars In the university parking.

```
Category = ['Toyota', 'Mazda', 'BMW', 'KIA', 'Suzuki']
Freq = [125, 85, 10, 97, 102]
```

```
import matplotlib.pyplot as plt
plt.bar(Category, Freq)
plt.show()
```

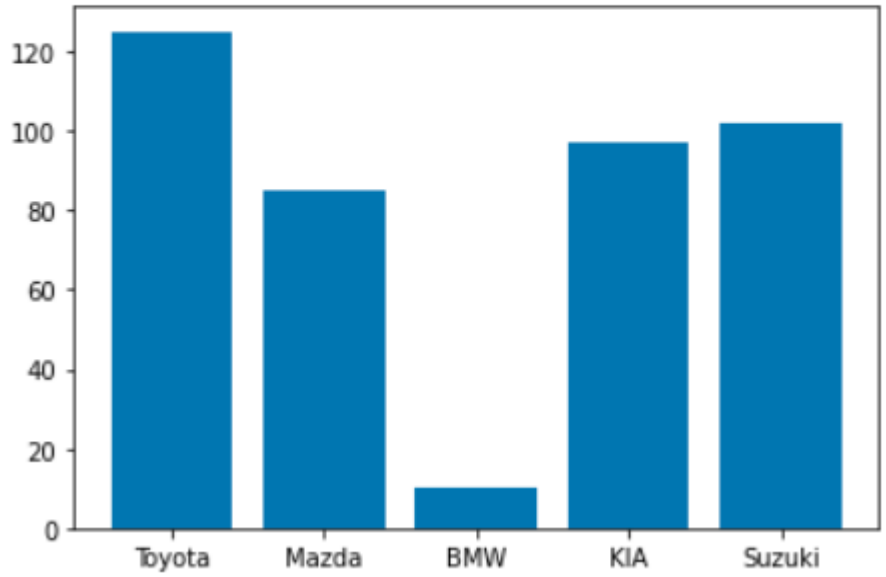


- Our first plot will be the bar plot.
- The *plt.show()* at the end presents the resulted plot

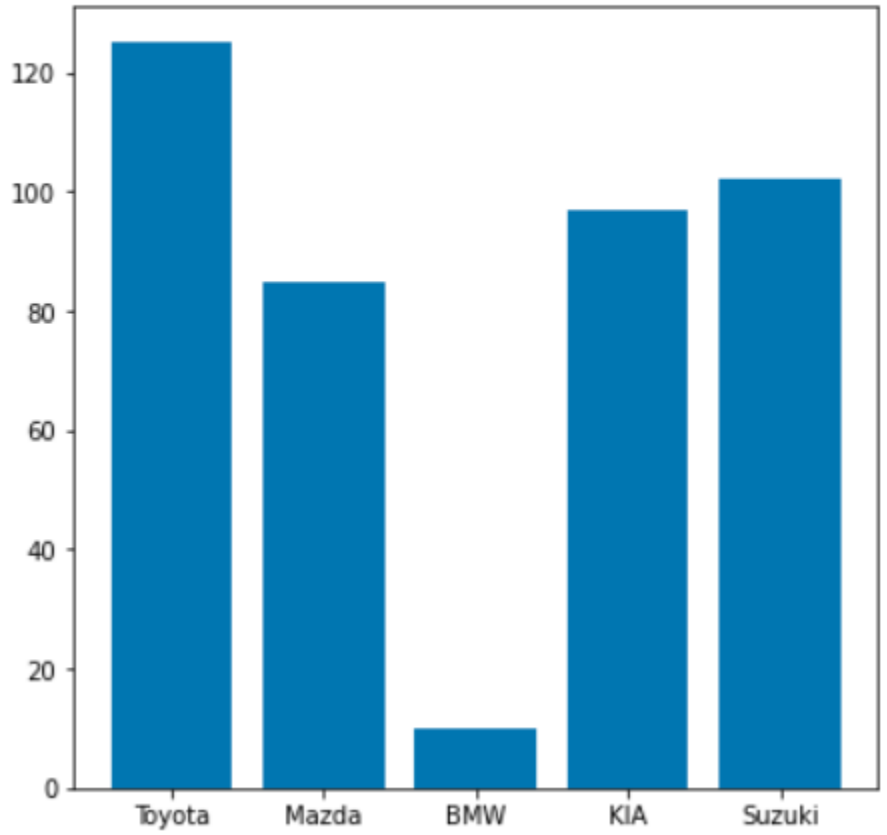
* Don’t use pie charts

- Using the *plt.figure()* function we can adjust the plot size.

```
import matplotlib.pyplot as plt
plt.bar(Category, Freq)
plt.show()
```

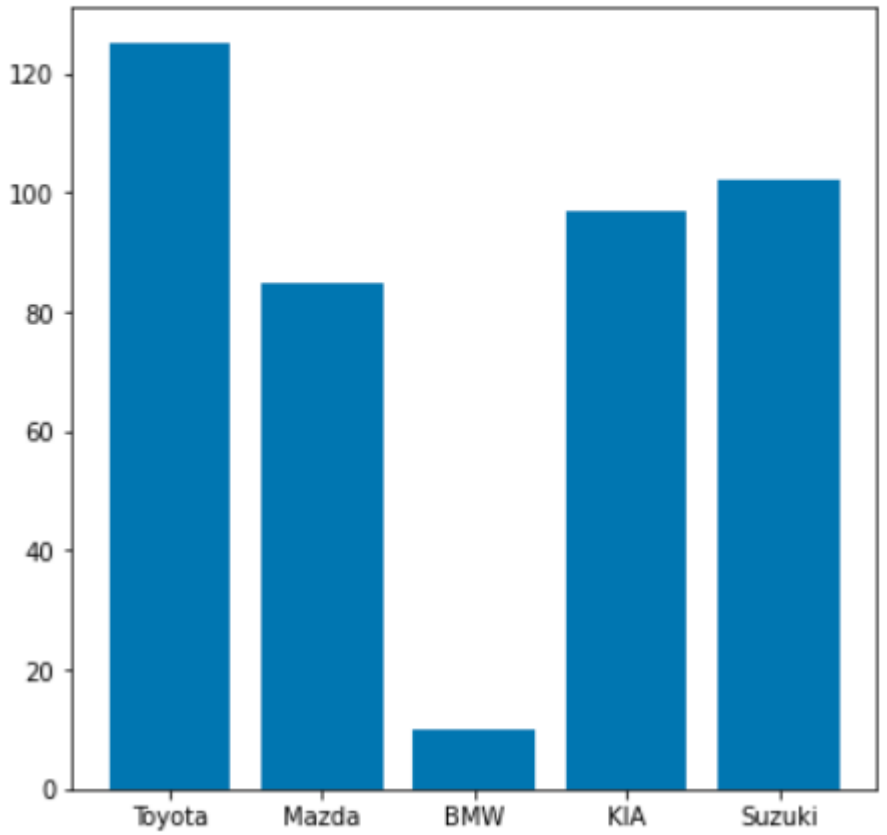



```
plt.figure(figsize=(6,6))
plt.bar(Category, Freq)
plt.show()
```

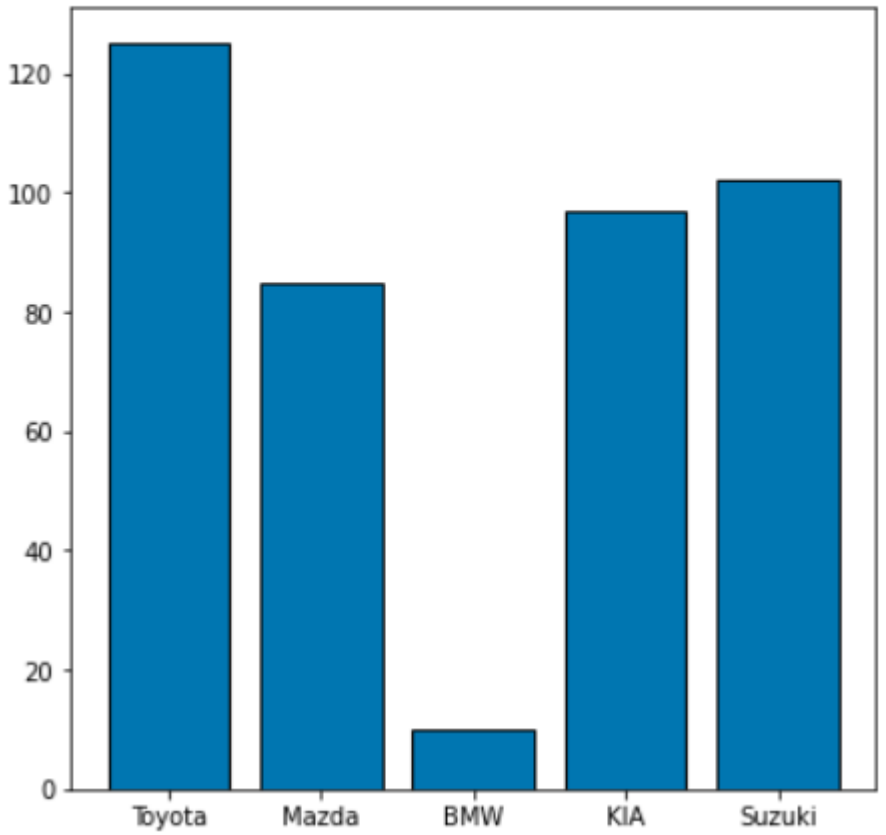


- Using *edgecolor* argument of the *plt.bar* function we can emphasize the bars' edges.

```
plt.figure(figsize=(6,6))
plt.bar(Category, Freq)
plt.show()
```

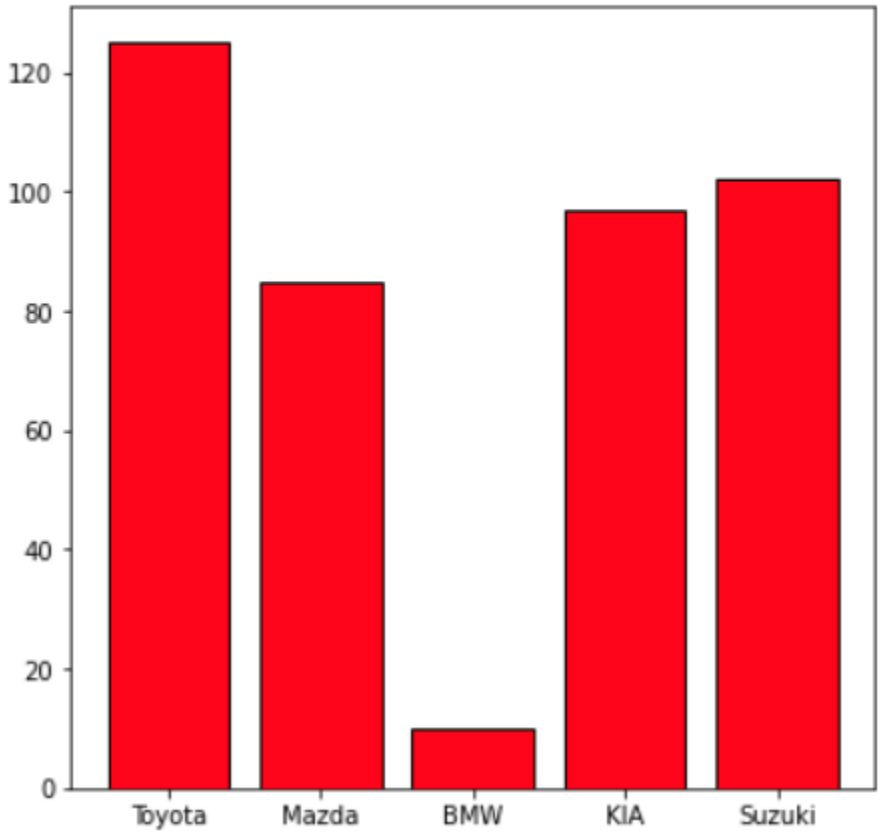



```
plt.figure(figsize=(6,6))
plt.bar(Category, Freq, edgecolor = "black")
plt.show()
```



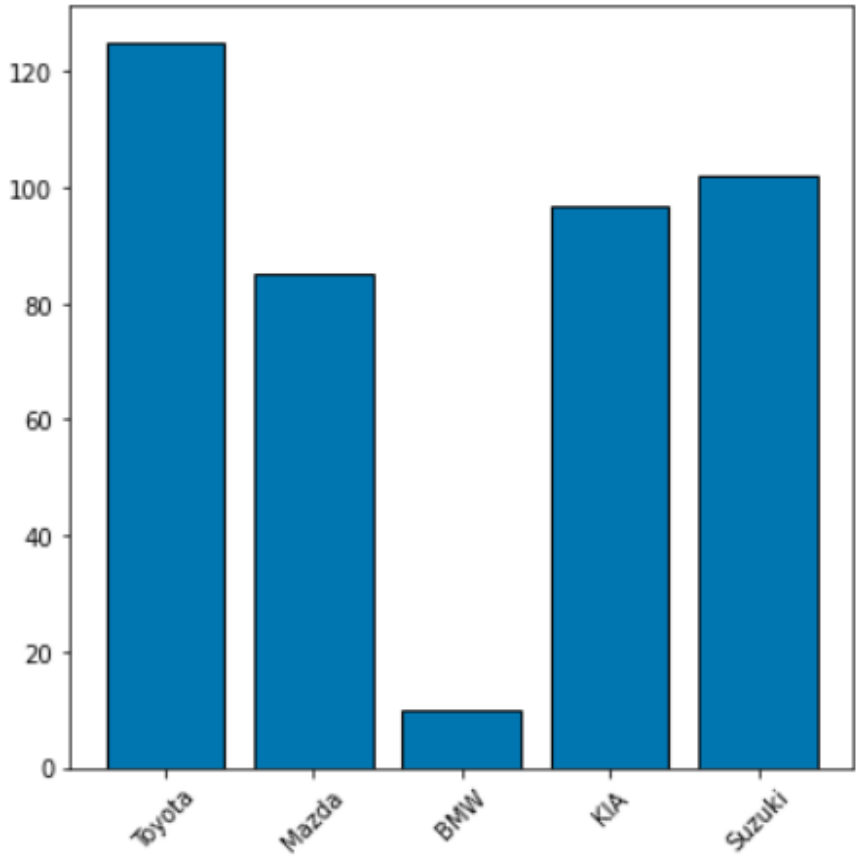
- We can also change the bars colors using the *color* argument.

```
plt.figure(figsize=(6,6))
plt.bar(Category, Freq, edgecolor = "black", color = "red")
plt.show()
```



- We can control the ticks on the x axis using the *xticks* function. For example, we can rotate the labels using the *rotation* argument.

```
plt.figure(figsize=(6,6))
plt.bar(Category, Freq, edgecolor = "black")
plt.xticks(rotation = 45)
plt.show()
```

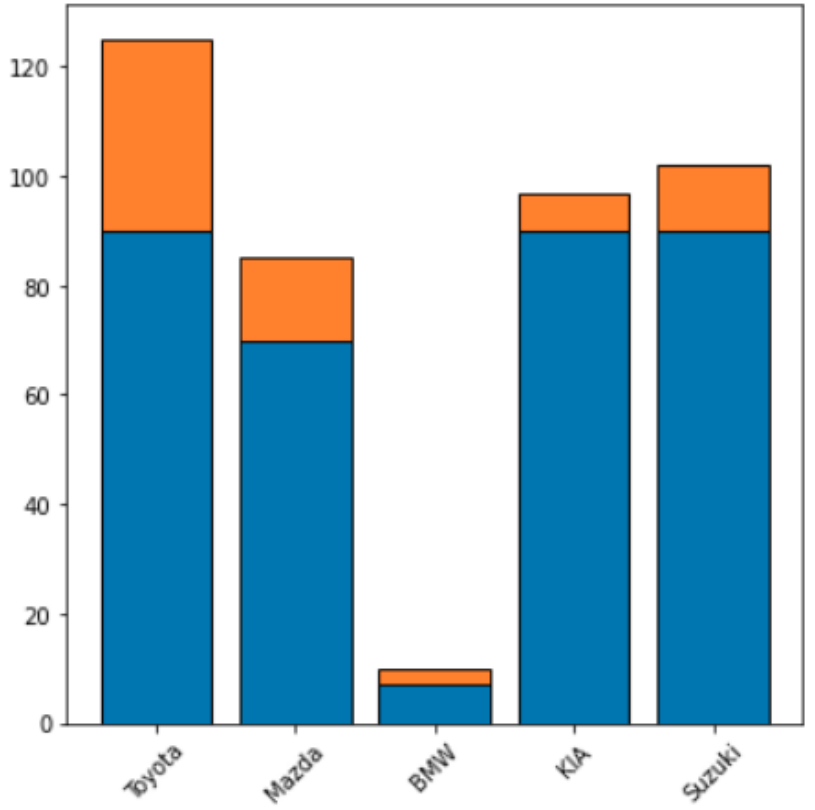


- In case we want to split the number of cars to 2 levels (working and broke cars) we can divide the bar plots to two, using the *bottom* argument.

```
Category = ['Toyota', 'Mazda', 'BMW', 'KIA', 'Suzuki']
Freq_work = [90, 70, 7, 90, 90]
Freq_broke = [35, 15, 3, 7, 12]
```

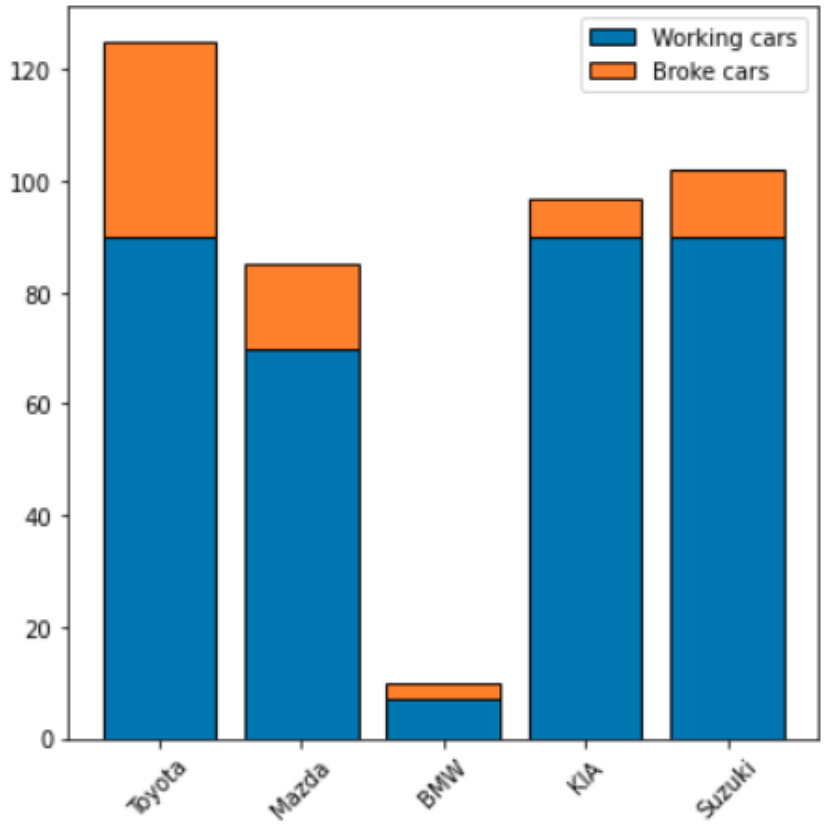
```
plt.figure(figsize=(6,6))
plt.bar(Category, Freq_work,edgecolor = "black")
plt.bar(Category, Freq_broke, bottom = Freq_work,edgecolor = "black")
plt.xticks(rotation = 45)
plt.show()
```

- How can we distinguish between broke and working cars?



- We can add legends any time using the function *plt.legend().*
- Legends create more informative plots and help us distinguish between subgroups, categories and many more.

```
plt.figure(figsize=(6,6))
plt.bar(Category, Freq_work,edgecolor = "black")
plt.bar(Category, Freq_broke, bottom = Freq_work,edgecolor = "black")
plt.xticks(rotation = 45)
plt.legend(loc = 'best', labels = ['Working cars', 'Broke cars'])
plt.show()
```



- More on bar plots can be found in the *matplotlib* module website.

matplotlib.pyplot.bar

`matplotlib.pyplot.bar(x, height, width=0.8, bottom=None, *, align='center', data=None, **kwargs)` [source]

Make a bar plot.

The bars are positioned at *x* with the given *align*ment. Their dimensions are given by *height* and *width*. The vertical baseline is *bottom* (default 0).

Many parameters can take either a single value applying to all bars or a sequence of values, one for each bar.

Parameters:

x : *float or array-like*

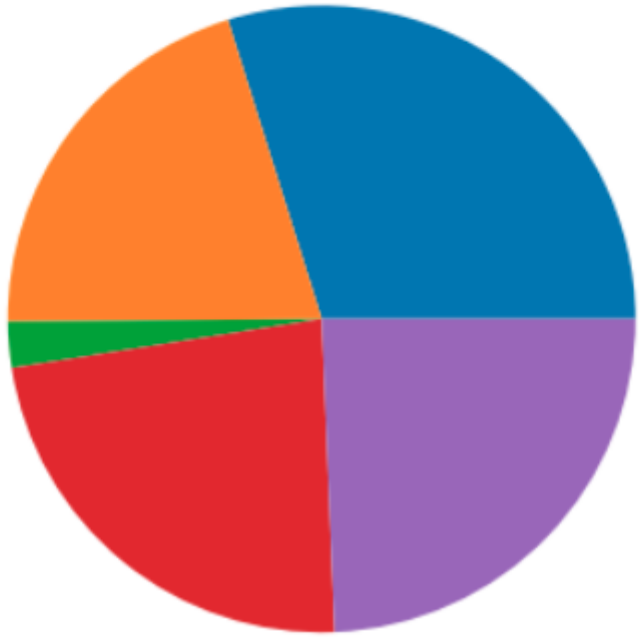
The x coordinates of the bars. See also *align* for the alignment of the bars to the coordinates.

height : *float or array-like*

The height(s) of the bars.

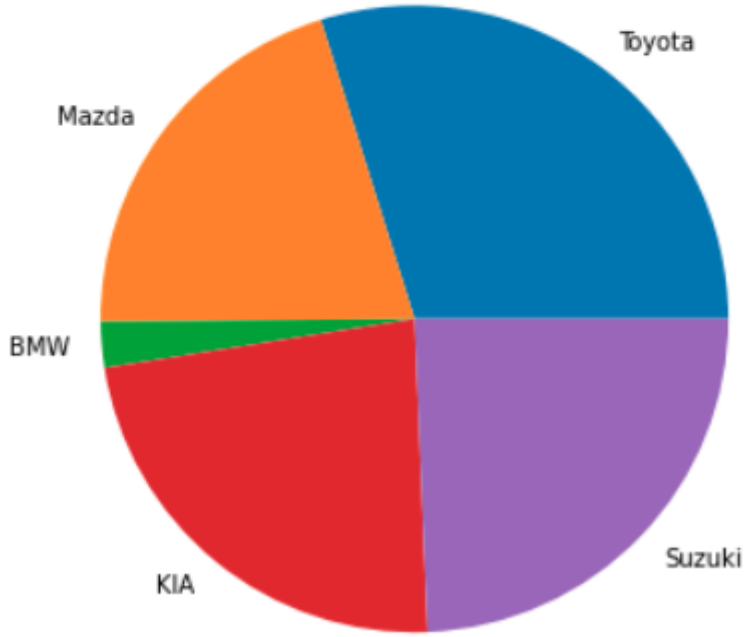
- Another option is the pie chart, which shows the proportion of each category in the data. We can do so using the *plt.pie* function.

```
plt.figure(figsize=(6,6))
plt.pie(Freq)
plt.show()
```



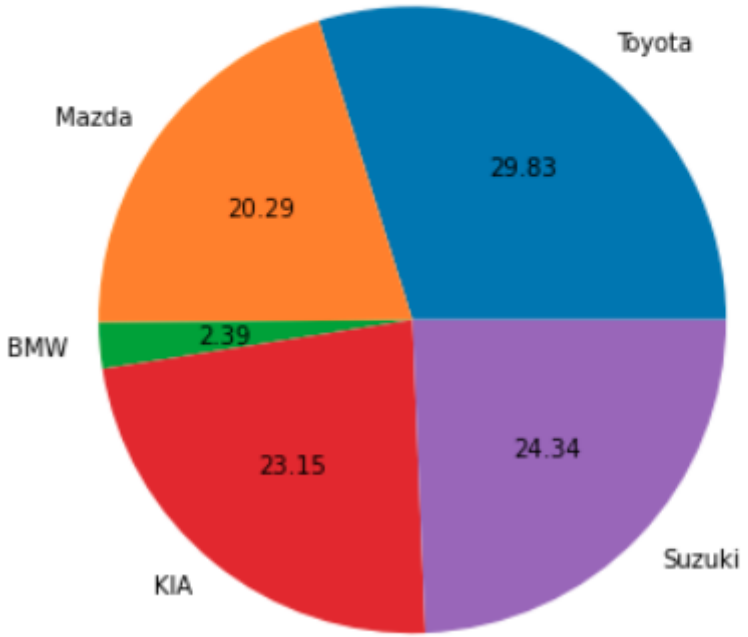
- By specifying the labels' names using the argument *labels* we get a more informative plot.

```
plt.figure(figsize=(6,6))
plt.pie(Freq, labels = Category)
plt.show()
```



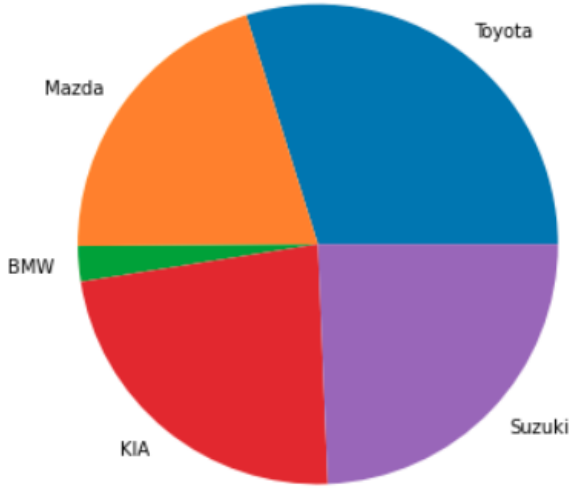
- An even more informative plot will include the percentage of each category. Fortunately, it can be done by the *autopct* argument.

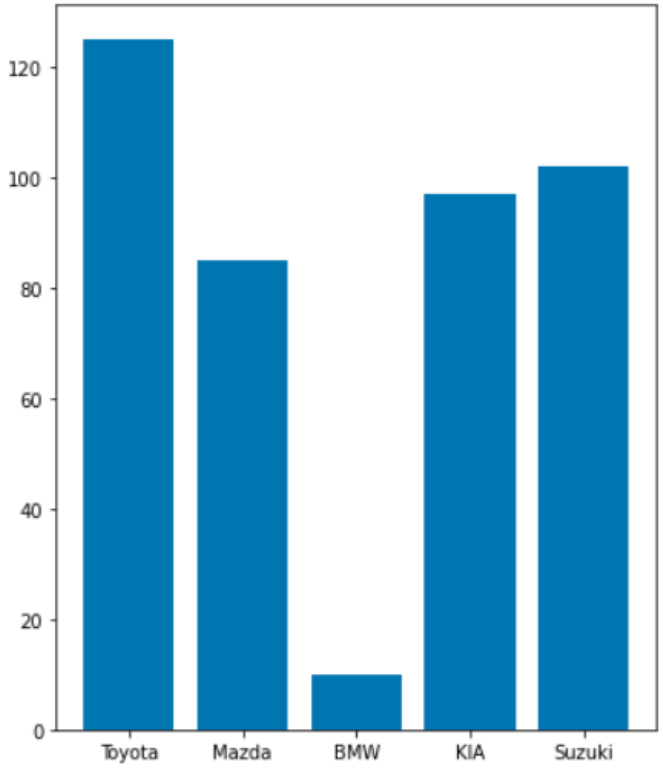
```
plt.figure(figsize=(6,6))
plt.pie(Freq, labels = Category, autopct="%.2f")
plt.show()
```



Pie charts Vs. Bar plots

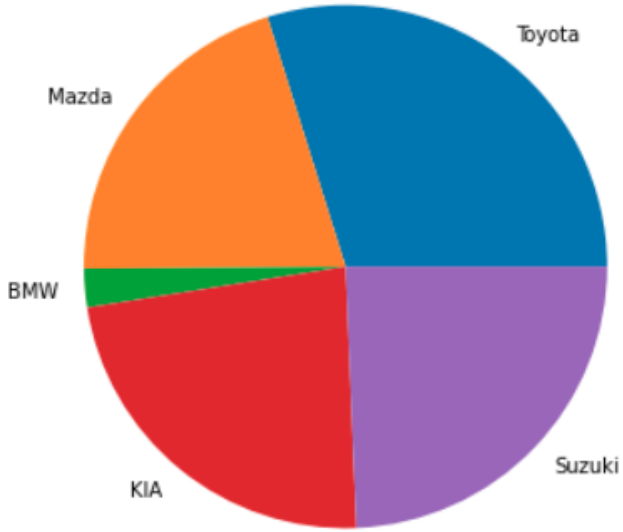
- Which plot makes the differences clearer?

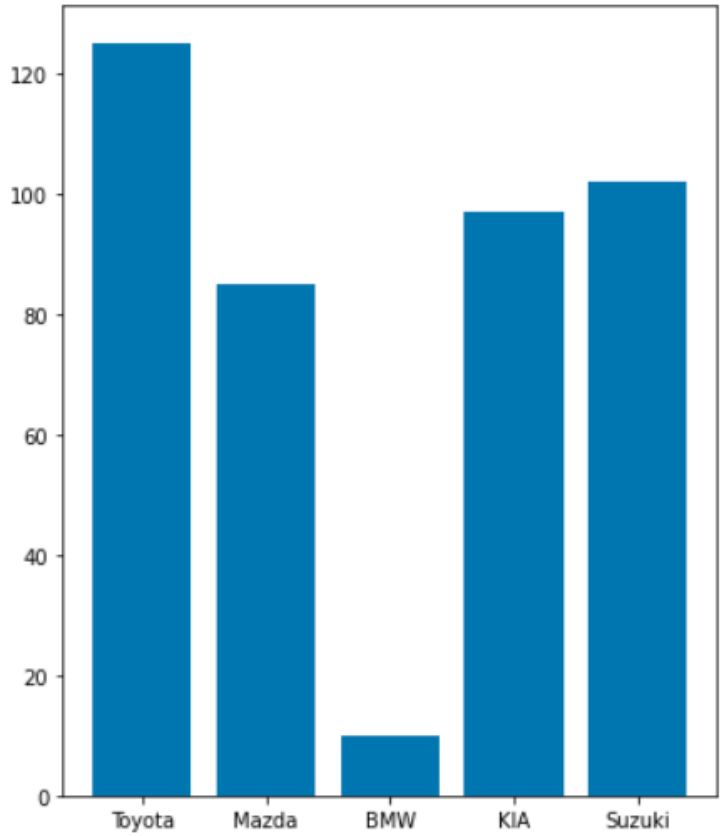




- The slide above included two plots beside each other. Using the plt.subplot() we can create a *n by m* grid of plots. For example, the plot before was generated using the code:

```
plt.figure(figsize=(10,6))
plt.subplot(1,2,1)
plt.pie(Freq, labels = Category)
plt.subplot(1,2,2)
plt.bar(Category, Freq)
plt.tight_layout()
plt.show()
```





Visualizing Discrete Distributions

- *Bernoulli* trial – test with 2 outcomes, success (1) and failure (0) where the probabilities of success and failures are p and $q = 1 - p$ respectively. Also called the *Bernoulli distribution*.
- Now consider a sequence of n independent and identically distributed (IID) Bernoulli trials, and we are interested with the number of successes out of the n trials.
- Obviously, the number of successes lies between 0 to n, but how can we compute for example the probability of k successes out of n trials?

- The *Binomial* distribution does exactly that. Specifically, if we conducted n IID Bernoulli trials with success probability p, the probability of *k* successes is

$$\binom{n}{k} p^k (1-p)^{n-k}$$

- We denote this distribution by Bin(n, p).
- The module *scipy.stats* consists of many functions which are related to different distributions (such as Bernoulli, Binomial, Normal and many others). More precisely, the above probability can be computed using the following code:

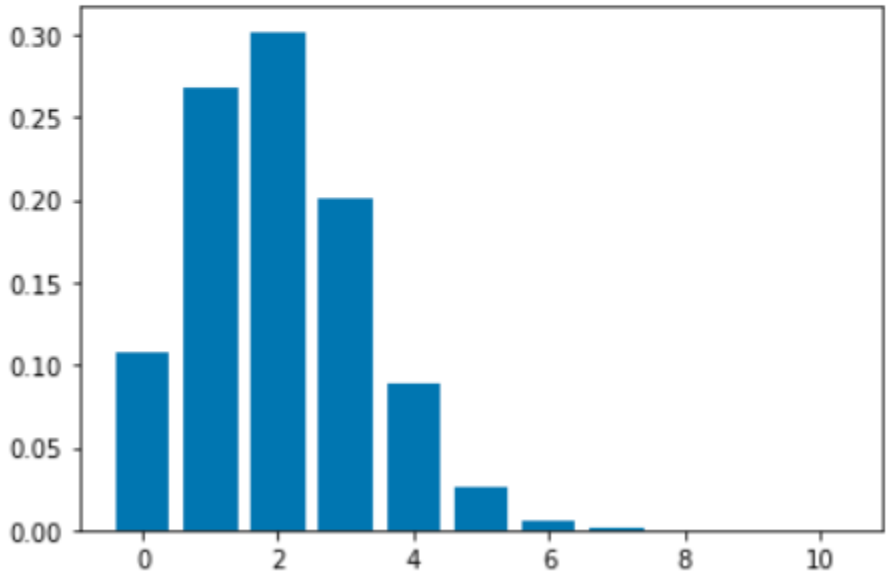
```
from scipy.stats import binom
n = 10
p = 0.2
binom.pmf(3, n, p)
```

```
0.20132659200000022
```

- First, let's visualize the Binomial distribution.

- For example, a Binomial distribution with $n = 10, p = 0.2$ looks like this:

```
n = 10 # The number of trials
p = 0.2 # The probability of success
X_values = [i for i in range(n+1)] # Possible values are 0,1,2,...,n
probs = [binom.pmf(x, n, p) for x in X_values]
plt.bar(X_values, probs)
plt.show()
```

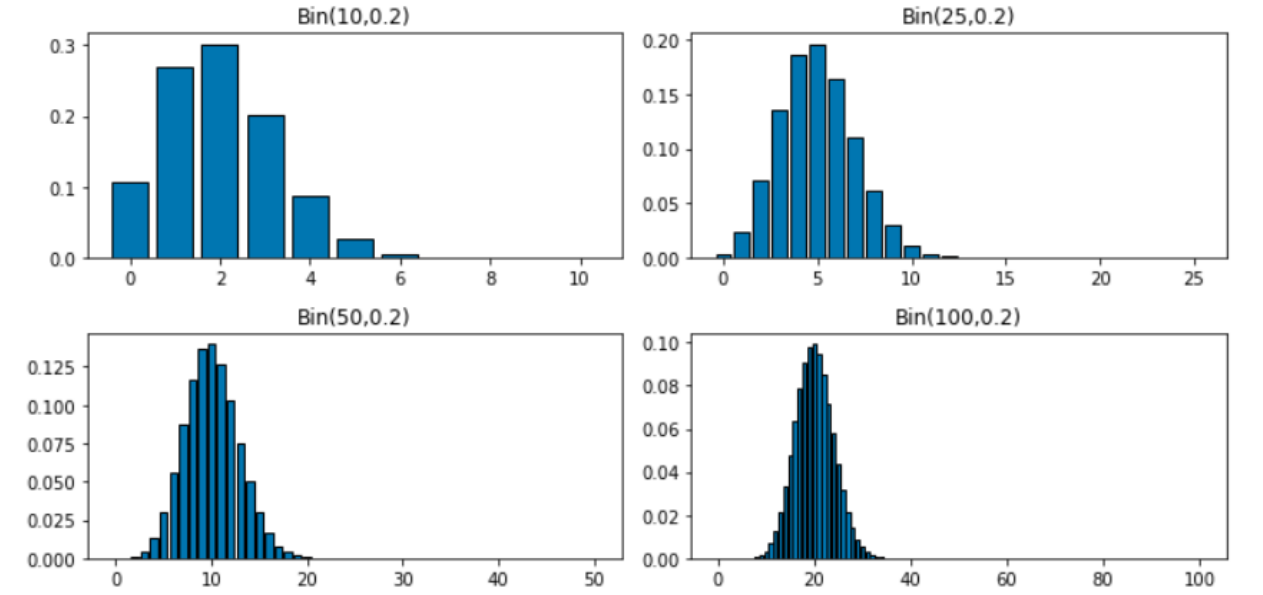


- What happens if we will change the values of n and p?

- First, I'll define 2 function, the first computes the probabilities of the Binomial distribution given values of n, p, and the second plots the distribution for different n values which are pre-specified.

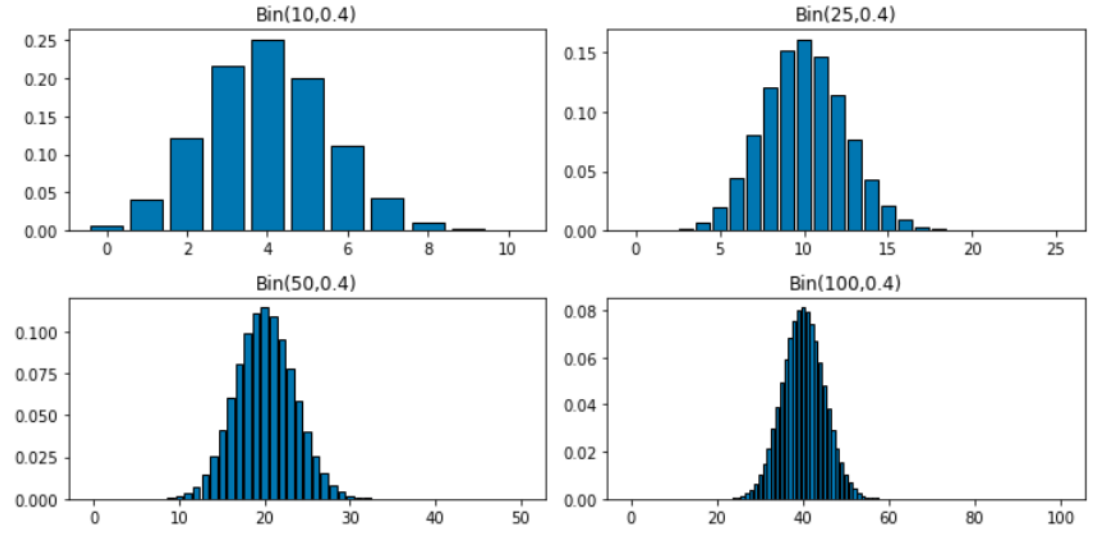
```
n = [10, 25, 50, 100]
p = [0.2, 0.4, 0.5, 0.9]

def BinomDist(N, P):
    values = [x for x in range(N+1)]
    probs = [binom.pmf(x, N, P) for x in values]
    return values, probs

def PlotBinom(n, P):
    plt.figure(figsize=(10,5))
    i = 1
    for N in n:
        x, prob = BinomDist(N, P)
        plt.subplot(2, 2, i)
        plt.bar(x, prob, edgecolor = "black")
        plt.title(f'Bin({N},{P})')
        i += 1
    plt.tight_layout()
    plt.show()
```

```
PlotBinom(n, p[0])
```

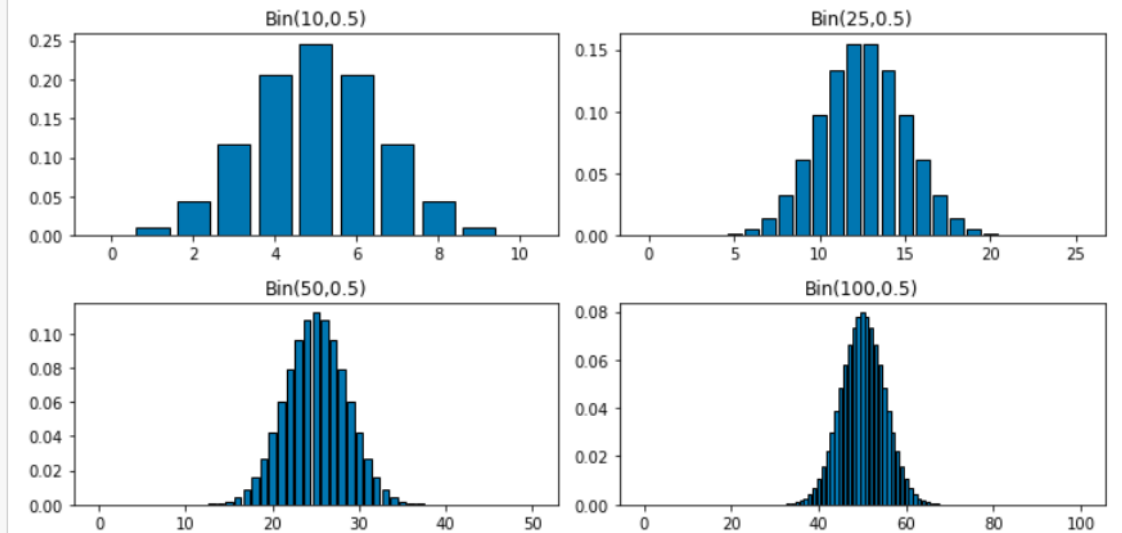


- We can try different values of p:

```
PlotBinom(n, p[1])
```

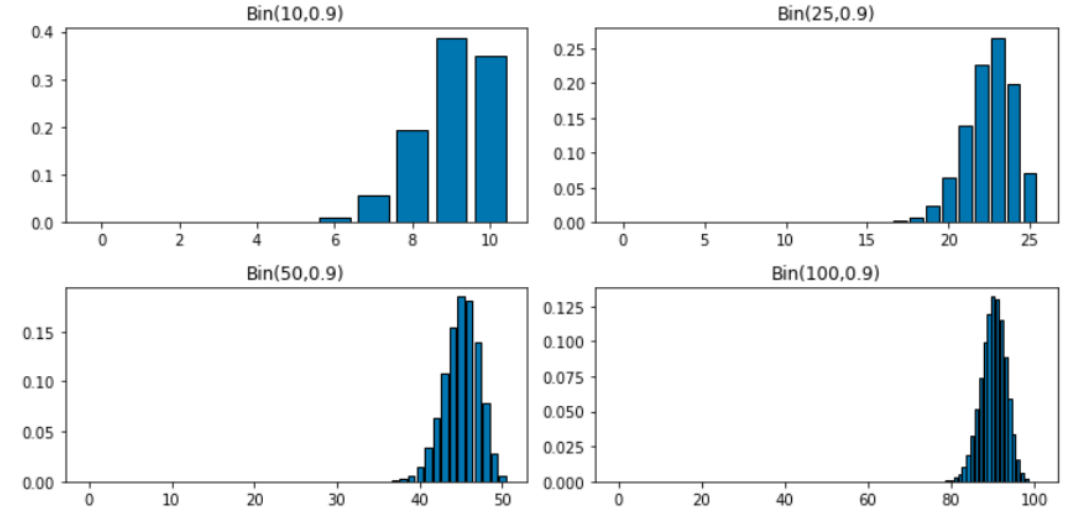



```
PlotBinom(n, p[2])
```



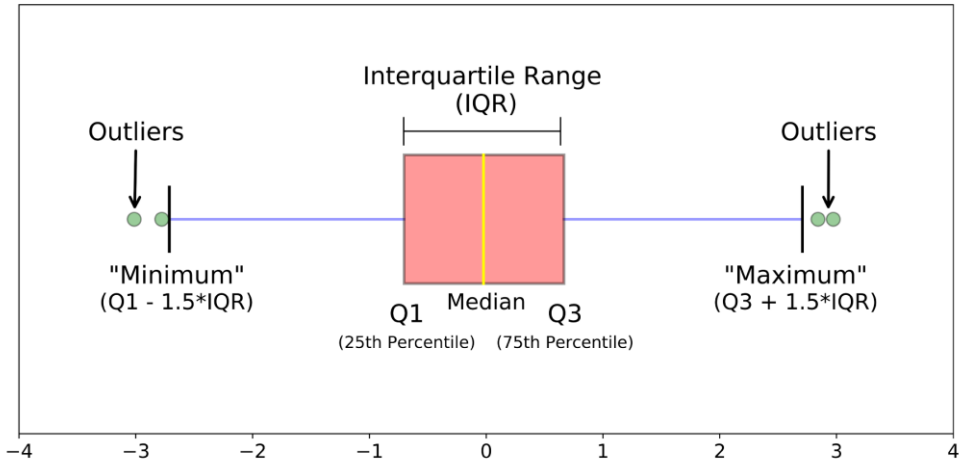
- Looks familiar?

```
PlotBinom(n, p[3])
```



Continuous Variables

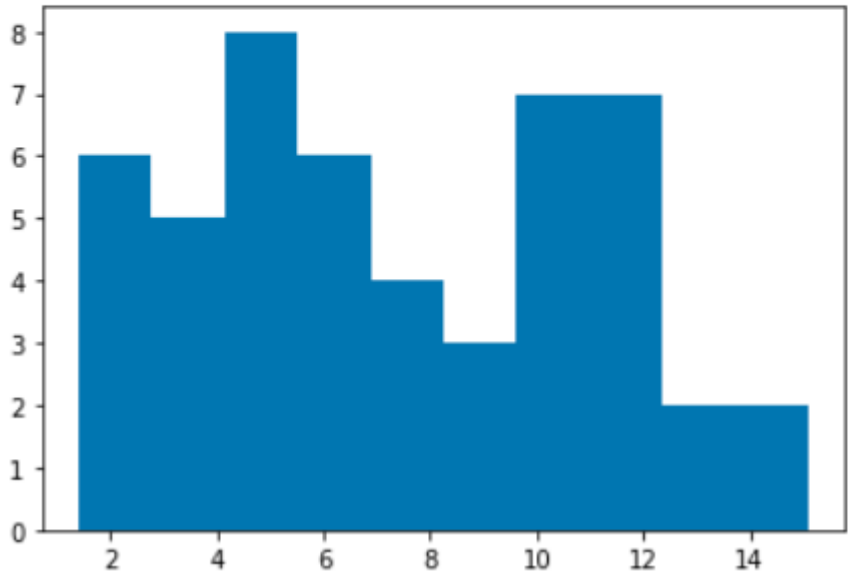
- Height, income, distance, etc. are handled differently and visualized using histograms and boxplots.
- When plotting a histogram, we split the range of the variable of interest into equally length and disjoint (i.e. without overlap) intervals, and we count how many observations fall within each bin.
- Histograms are non-parametric density estimators.
- Boxplots present the quartiles of the distribution.



Histograms

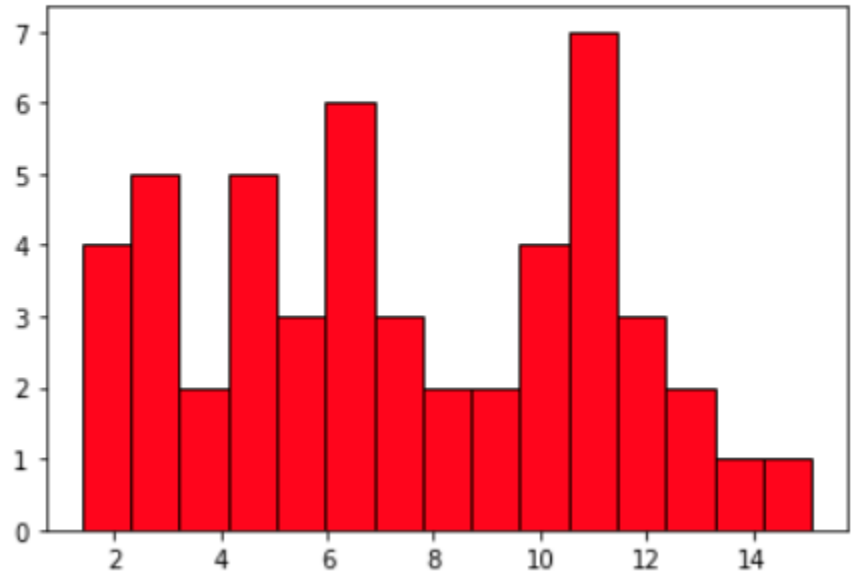
- For histograms we use the function plt.hist()

```
x = [3.1, 2.7, 3.3, 3.3, 5.2, 10.9, 6.1, 2.4, 5.5, 15.1, 10.1, 6.2, 10.7,
     13.9, 10.6, 13.2, 8.5, 12.5, 11.1, 6.4, 11.6, 11, 12.2, 9.5, 6.7,
     10.3, 7.1, 2.3, 4.5, 11.1, 2.3, 9.3, 2.9, 1.4, 7.4, 1.7, 3,
     11.3, 7.8, 10.3, 6.8, 6.2, 5.3, 5, 11.5, 9.7, 4.2, 4.5, 4.3, 6.9]
plt.hist(x)
plt.show()
```

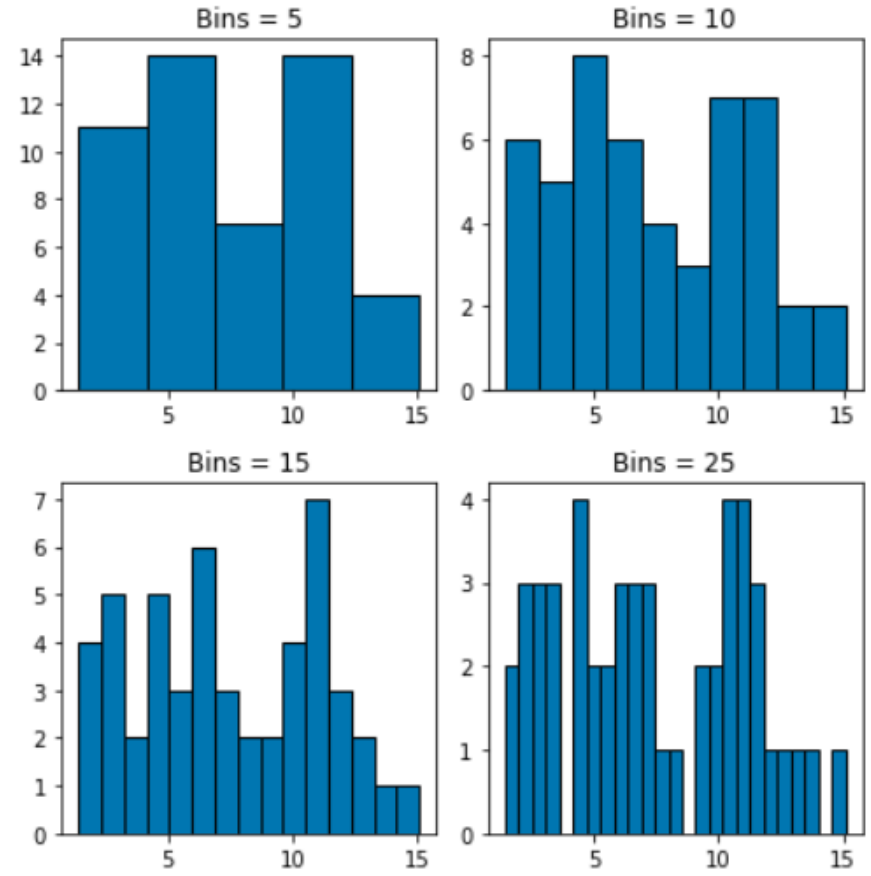


- Python selects the number bins automatically, but we can choose different values.
- We can also change the color and The edge color like in bar plots.

```
plt.hist(x, bins = 15, edgecolor = "black", color = "red")
plt.show()
```

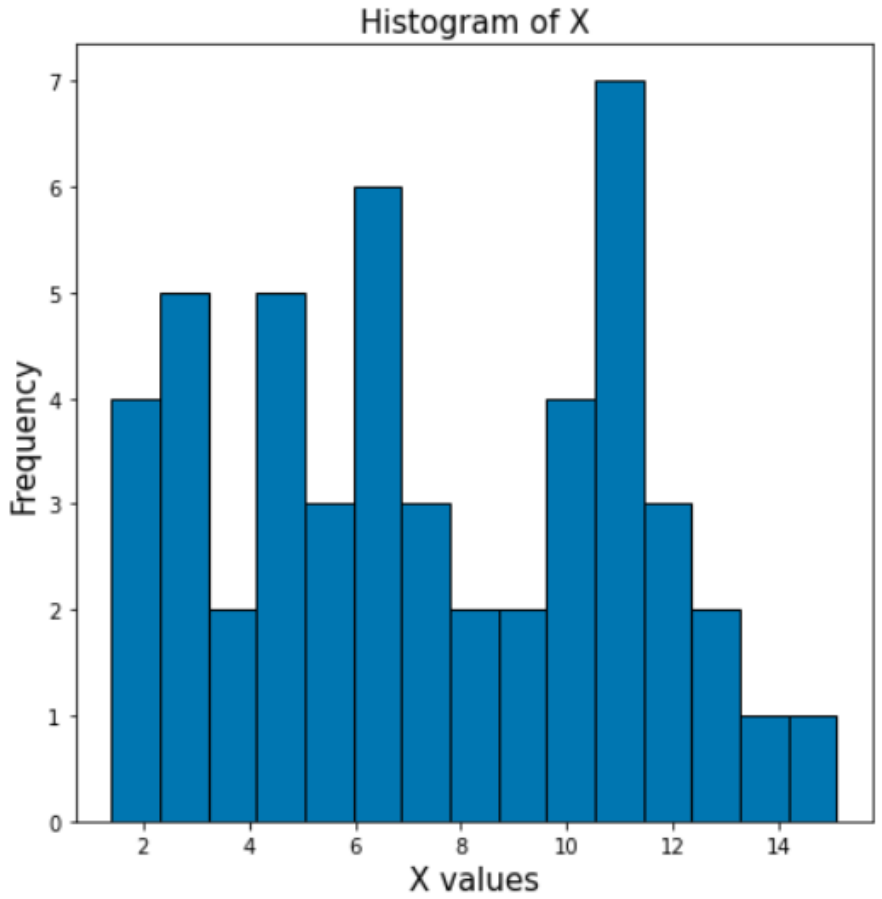



```
Bins = [5, 10, 15, 25]
plt.figure(figsize=(6,6))
for i in range(4):
    plt.subplot(2, 2, i+1)
    plt.hist(x, bins = Bins[i], edgecolor = "black")
    plt.title(f'Bins = {Bins[i]}')
plt.tight_layout()
plt.show()
```

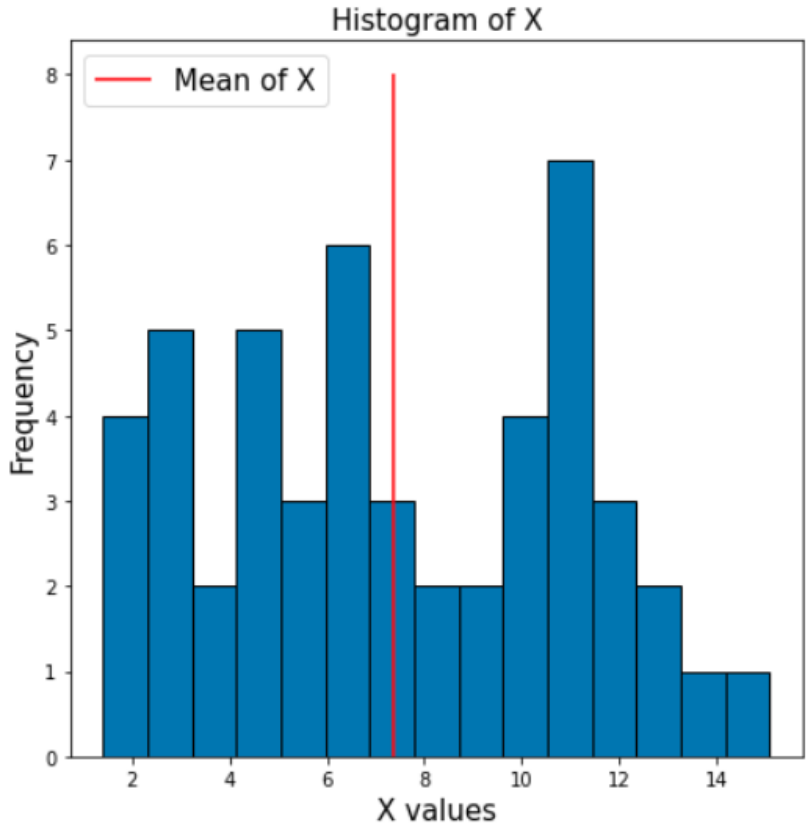


- We can also change the axis labels, title, size and more.

```
plt.figure(figsize=(7,7))
plt.hist(x, bins = 15, edgecolor = "black")
plt.xlabel('X values', size = 15)
plt.ylabel('Frequency', size = 15)
plt.title('Histogram of X', size = 15)
plt.show()
```



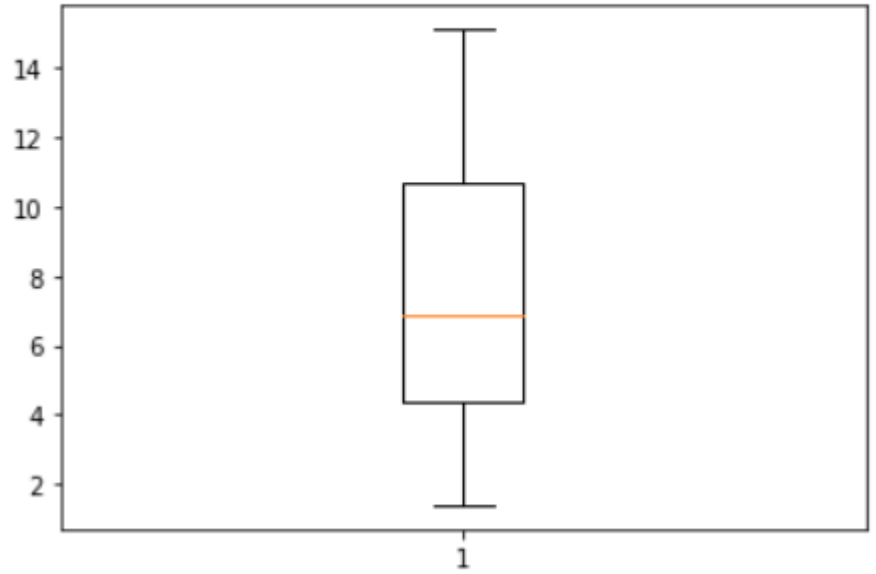

```
plt.figure(figsize=(7,7))
plt.hist(x, bins = 15, edgecolor = "black")
plt.vlines(sum(x)/len(x), 0, 8, color = "red")
plt.legend(labels = ['Mean of X'], loc = 'upper left', fontsize = 15)
plt.xlabel('X values', size = 15)
plt.ylabel('Frequency', size = 15)
plt.title('Histogram of X', size = 15)
plt.show()
```



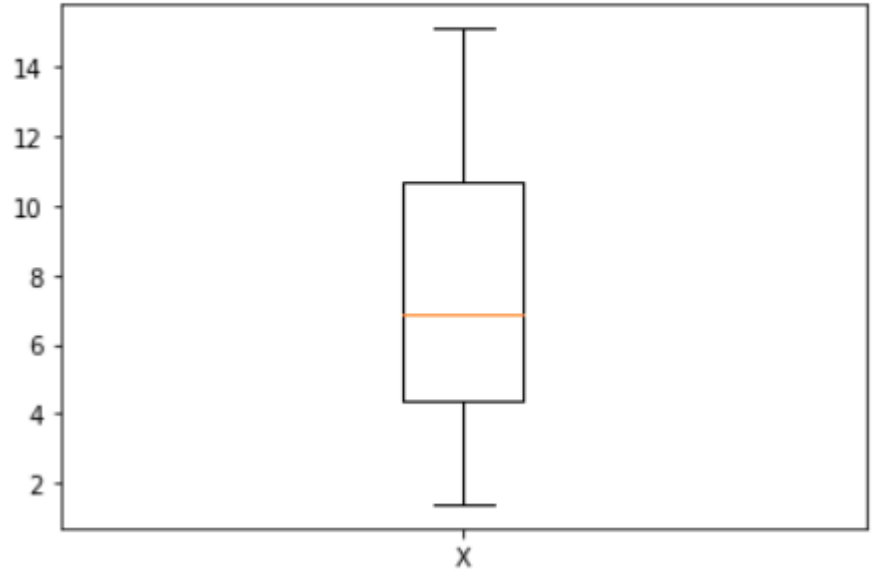
Boxplots

- Boxplots can be plotted using the *plt.boxplot()* function (Shocking right?).

```
plt.boxplot(x)
plt.show()
```

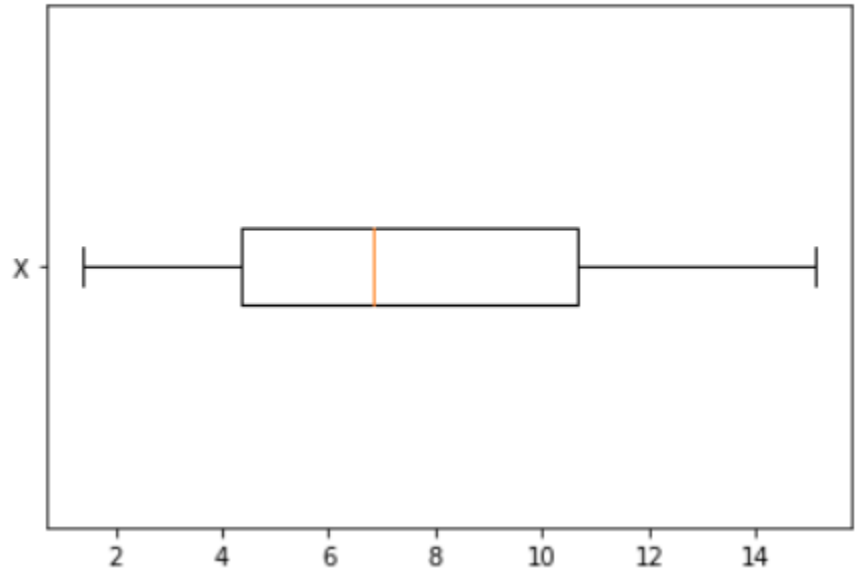



```
plt.boxplot(x, labels = "X")
plt.show()
```

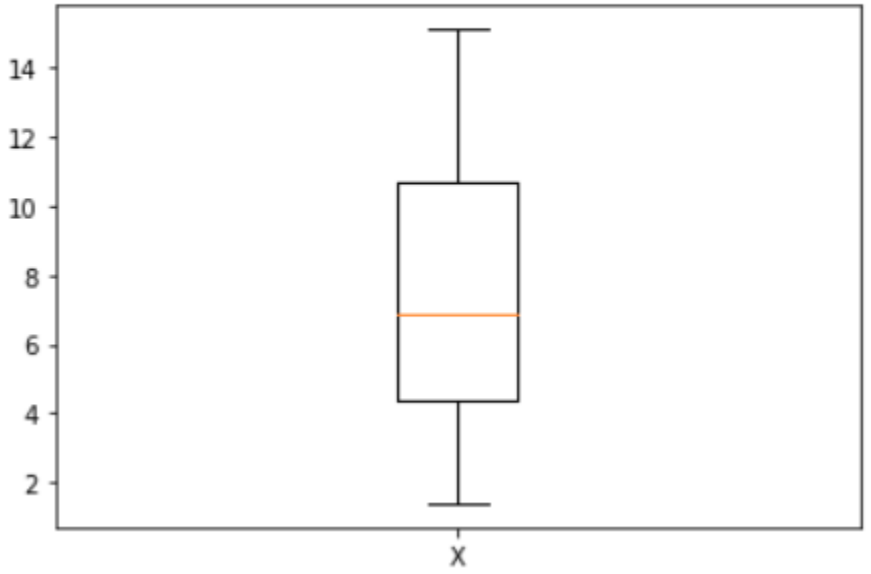


- The boxplots can be plotted horizontally as well, using the *vert* argument.
- In addition, we can show the mean instead if the median, using the *meanline* argument.

```
plt.boxplot(x, vert = False, labels = "X")
plt.show()
```

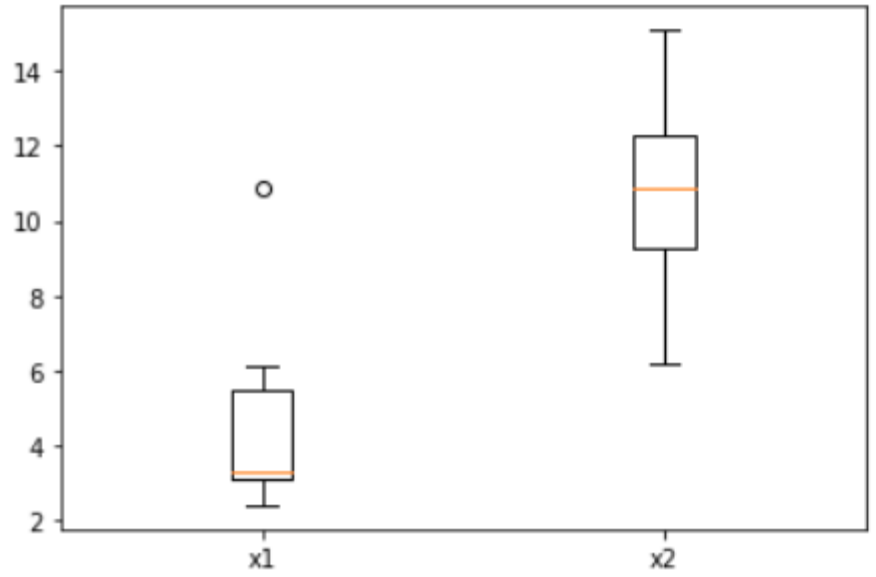



```
plt.boxplot(x, labels = "X", meanline=True)
plt.show()
```

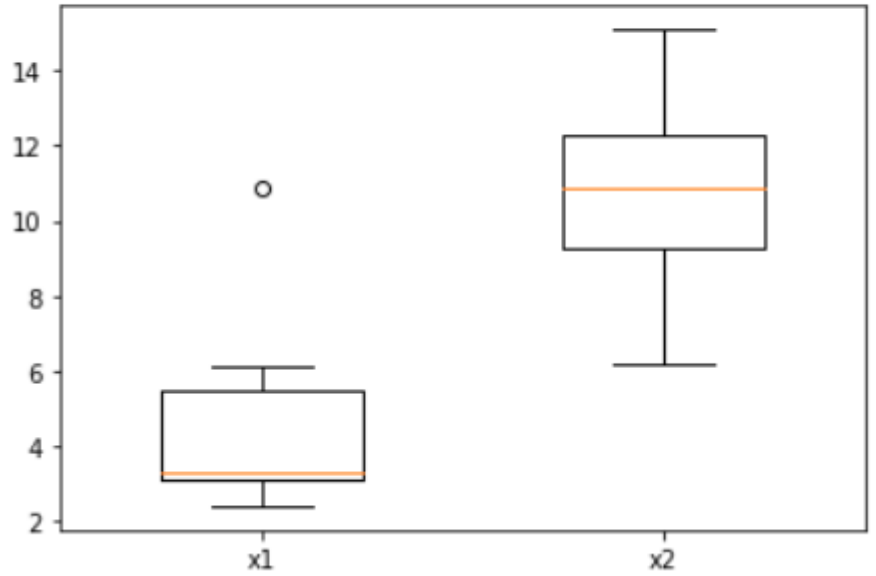


- We can even have 2 (and more than 2) boxplots beside each other and change their widths.

```
x1 = [3.1, 2.7, 3.3, 3.3, 5.2, 10.9, 6.1, 2.4, 5.5]
x2 = [15.1, 10.1, 6.2, 10.7, 13.9, 10.6, 13.2, 8.5, 12.5, 11.1, 6.4, 11.6, 11, 12.2, 9.5, 6.7]
plt.boxplot([x1,x2], labels = ["x1", "x2"])
plt.show()
```

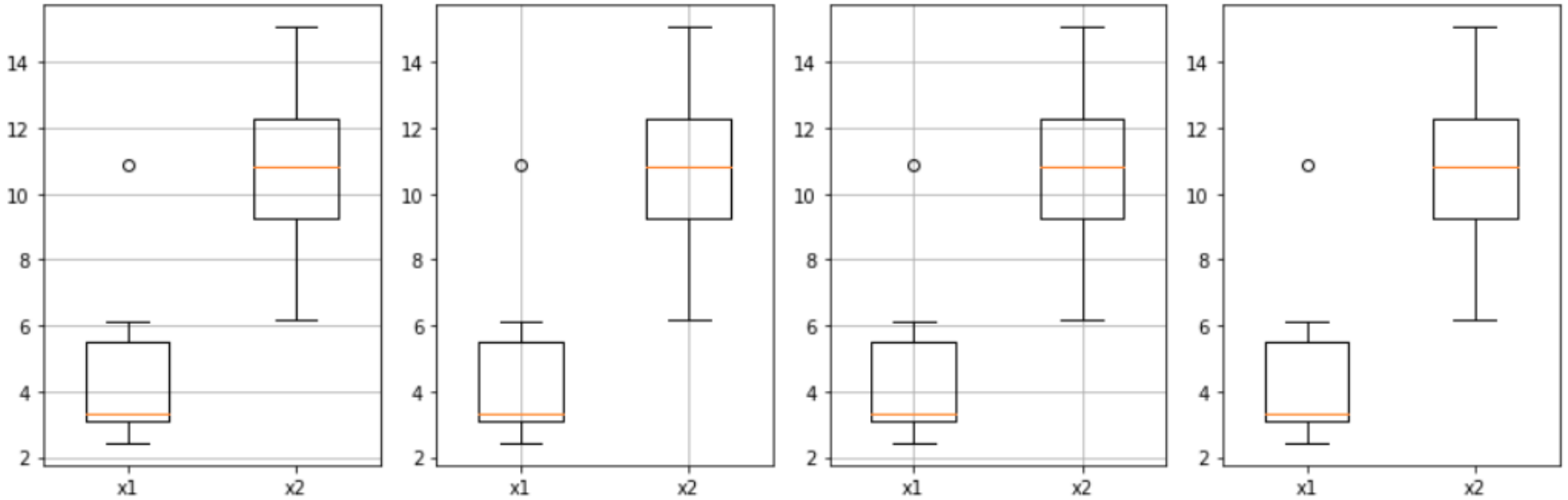



```
plt.boxplot([x1,x2], labels = ["x1", "x2"], widths=0.5)
plt.show()
```



- Adding grid lines is can be helpful and yield a more informative plot.

```
plt.figure(figsize=(12,4))
plt.subplot(1,4,1)
plt.boxplot([x1,x2], labels = ["x1", "x2"], widths=0.5)
plt.grid(axis="y")
plt.subplot(1,4,2)
plt.boxplot([x1,x2], labels = ["x1", "x2"], widths=0.5)
plt.grid(axis="x")
plt.subplot(1,4,3)
plt.boxplot([x1,x2], labels = ["x1", "x2"], widths=0.5)
plt.grid()
plt.subplot(1,4,4)
plt.boxplot([x1,x2], labels = ["x1", "x2"], widths=0.5)
plt.tight_layout()
plt.show()
```



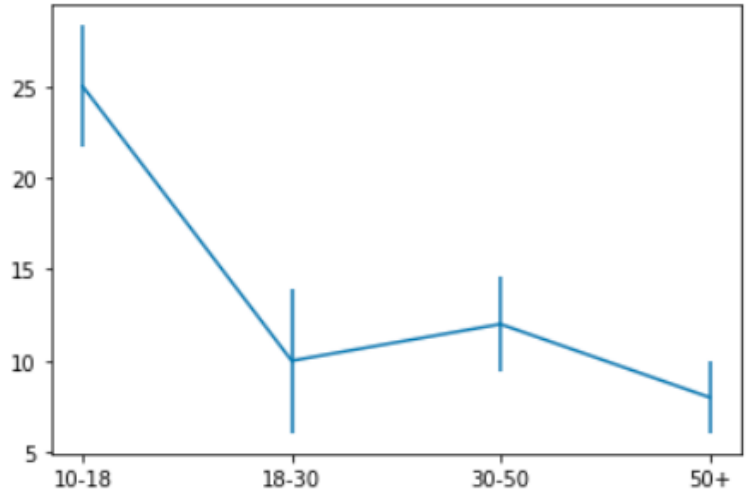
Error Bars

- Assume that we have observations of weekly hours spent on Instagram across different age groups, and we want to visualize the differences between the groups' means.

```
groups = ['10-18', '18-30', '30-50', '50+']
M = [25, 10, 12, 8] # Groups means
SE = [1.7, 2, 1.3, 1] # Means standard error
CI = [1.96*se for se in SE] # Upper limit of the mean 95% CI
```

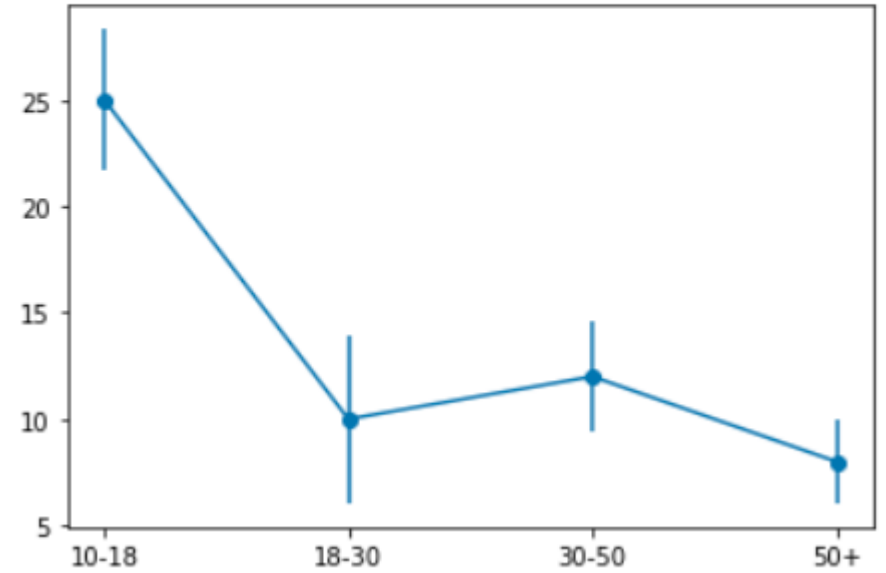
- A suitable candidate is the error bar.
  Using the function plt.errorbar()
  we can plot the mean with its confidence - interval (or any other interval).

```
plt.errorbar(groups, M, yerr=CI)
plt.show()
```



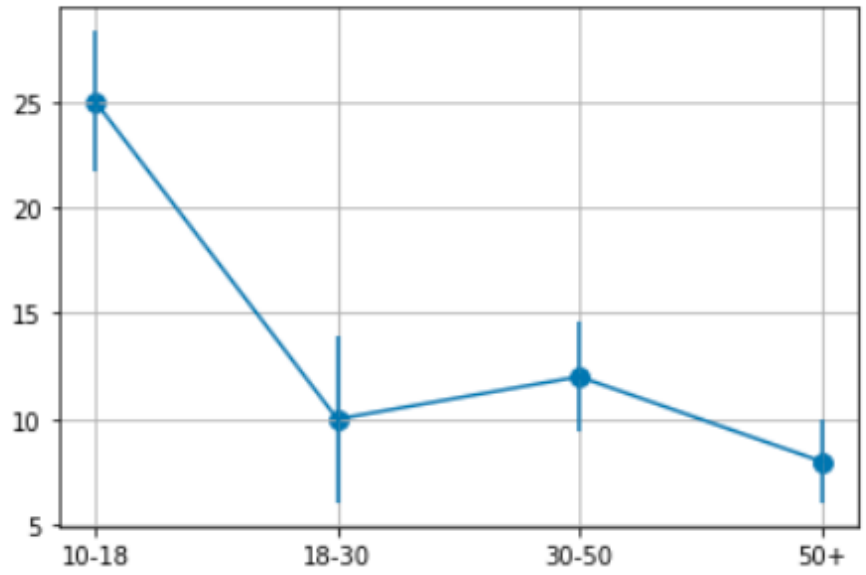
- More ‘improvements’ can be made:
- Adding points to show the exact position of the mean using the *plt.scatter()* function (more on it later).

```
plt.scatter(groups, M)
plt.errorbar(groups, M, yerr=CI)
plt.show()
```



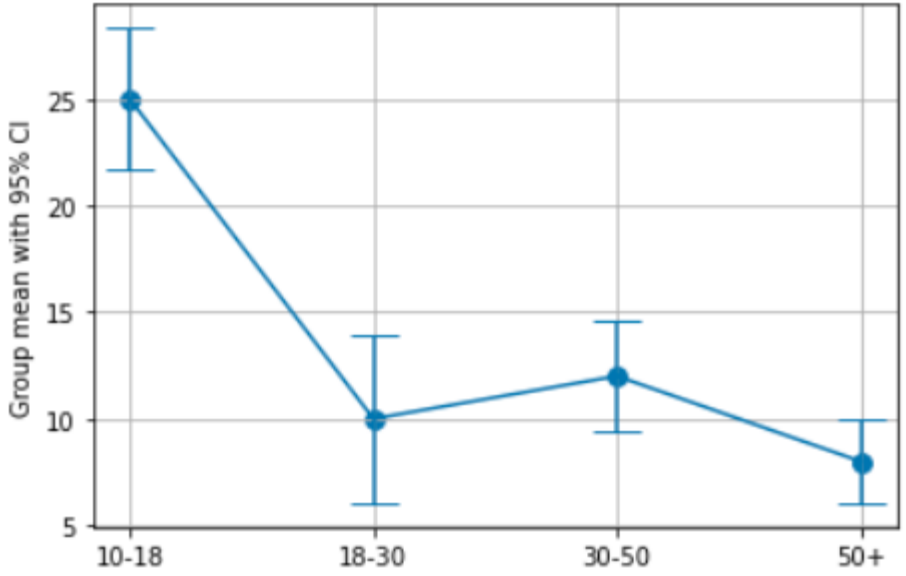
- Grid lines

```
plt.errorbar(groups, M, yerr=CI)
plt.scatter(groups, M, s=60)
plt.grid()
plt.show()
```



- Whiskers, and y axis label.

```
plt.errorbar(groups, M, yerr=CI, capsize=10)
plt.scatter(groups, M, s=60)
plt.grid()
plt.ylabel('Group mean with 95% CI')
plt.show()
```

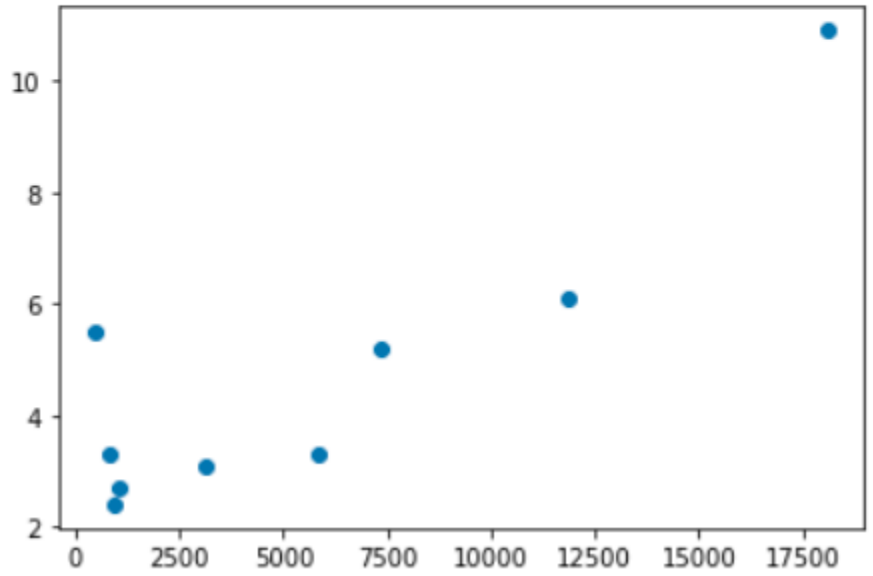


2-Dimensional Continuous Data

- In case we have 2 continuous variables, usually the optimal candidate for visualizing their relation is the good old *scatter plot.*

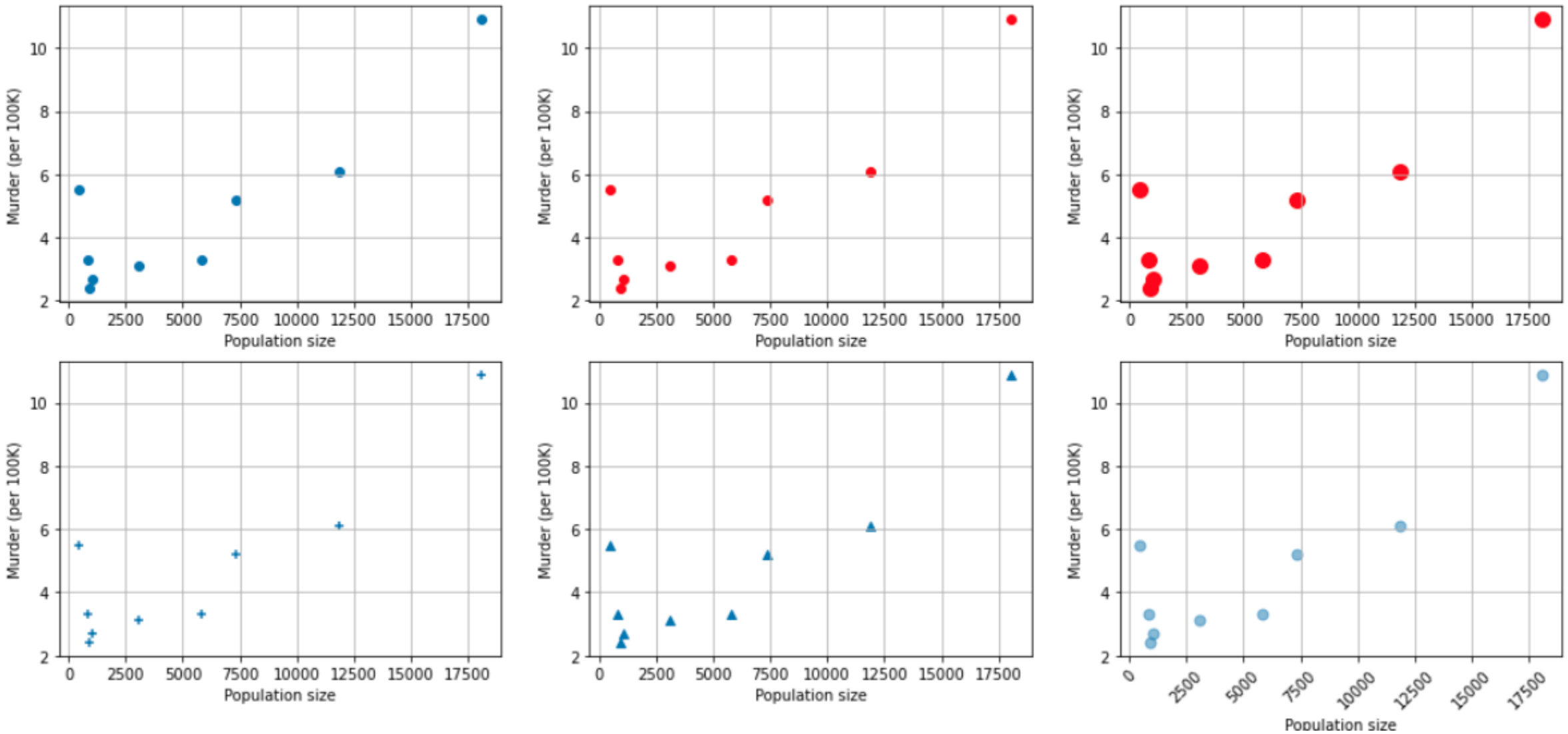
```
Pop1 = [3100, 1058, 5814, 812, 7333, 18076, 11860, 931, 472]
Murder1 = [3.1, 2.7, 3.3, 3.3, 5.2, 10.9, 6.1, 2.4, 5.5]
```

```
plt.scatter(Pop1, Murder1)
plt.show()
```



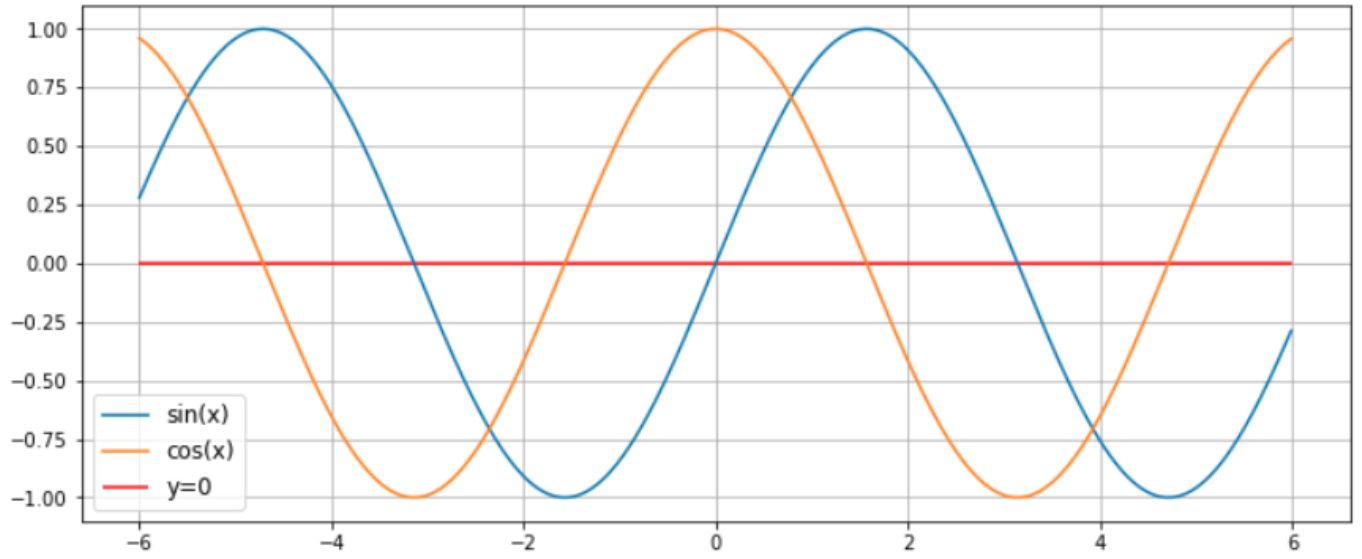
- As we saw we can control many aspects of the plot. In scatter plots we can change the dots marker colors, size, shape, transparency,
and many more

```
plt.figure(figsize=(18,8))
plt.subplot(2,3,1)
plt.scatter(Pop1, Murder1)
plt.xlabel('Population size')
plt.ylabel('Murder (per 100K)')
plt.grid()
plt.subplot(2,3,2)
plt.scatter(Pop1, Murder1, c = 'r') # c - changing the markers' color
plt.xlabel('Population size')
plt.ylabel('Murder (per 100K)')
plt.grid()
plt.subplot(2,3,3)
plt.scatter(Pop1, Murder1, c = 'r', s = 100) # s - markers' size
plt.xlabel('Population size')
plt.ylabel('Murder (per 100K)')
plt.grid()
plt.subplot(2,3,4)
plt.scatter(Pop1, Murder1, marker="+") # marker - changing the markers' shape
plt.xlabel('Population size')          # In this case we will se + sign
plt.ylabel('Murder (per 100K)')
plt.grid()
plt.subplot(2,3,5)
plt.scatter(Pop1, Murder1, marker="^") # Here we will have triangles
plt.xlabel('Population size')
plt.ylabel('Murder (per 100K)')
plt.grid()
plt.subplot(2,3,6)
plt.scatter(Pop1, Murder1, s=50, alpha = 0.5) # alpha - markers transparency
plt.xlabel('Population size')
plt.ylabel('Murder (per 100K)')
plt.grid()
plt.xticks(rotation = 45) # rotating the labels on the x axis
plt.tight_layout()
plt.show()
```


Murder (per 100K)
Population size
2
4
6
8
10
0
2500
5000
7500
10000
12500
15000
17500

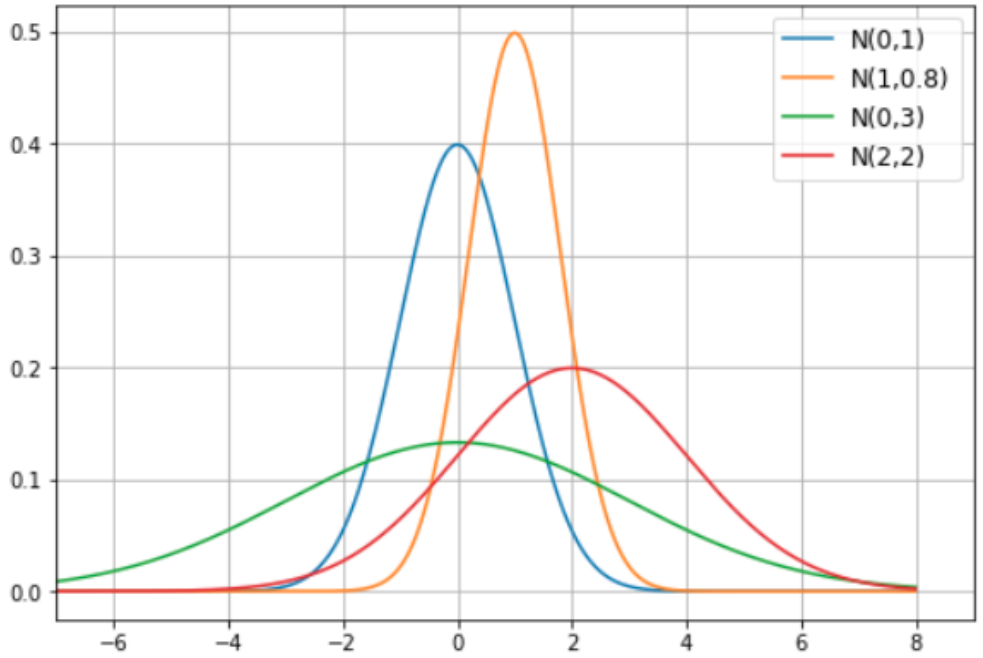
- We can also plot mathematical functions such as sin(x) and cos(x) (and any other function).

```
from math import sin, cos
x = [i/100 for i in range(-600, 600)]
y_sin = [sin(Y) for Y in x]
y_cos = [cos(Y) for Y in x]
plt.figure(figsize=(12,5))
plt.plot(x,y_sin)
plt.plot(x,y_cos)
plt.grid()
plt.hlines(0, -6, 6, color = "red")
plt.legend(loc = "best", labels = ['sin(x)', 'cos(x)', 'y=0'], fontsize = 12)
plt.show()
```



- And our beloved Normal distribution density.

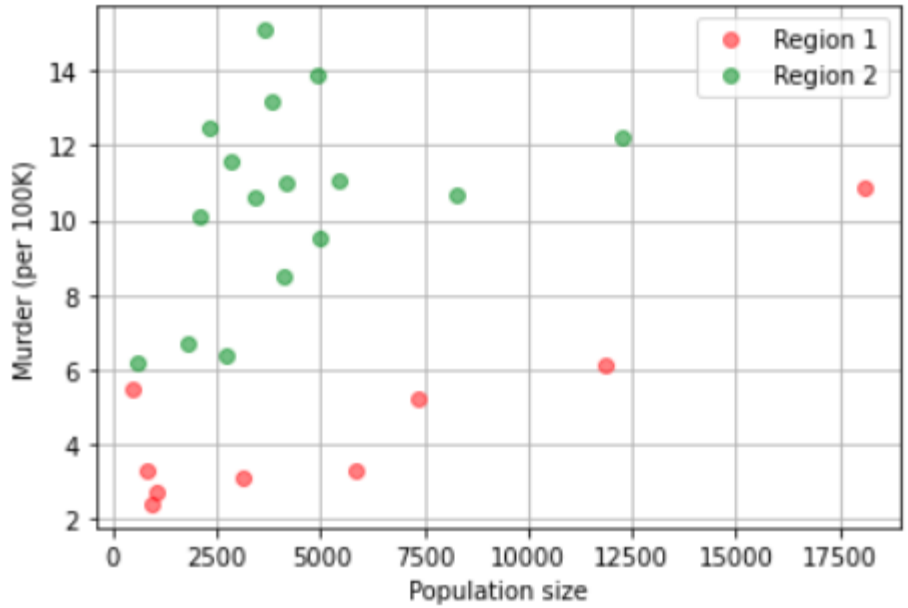
```
from scipy.stats import norm # Importing the normal distribution module
mu = [0, 1, 0, 2]
sd = [1, 0.8, 3, 2]
x = [i/100 for i in range(-800, 800)] # x value from -8 to 8 with steps of 0.01

plt.figure(figsize=(8,5.5))
for i in range(4):
    y = [norm.pdf(X, loc = mu[i], scale = sd[i]) for X in x]
    plt.plot(x,y)
    plt.xlim(-7, 9)

plt.grid()
plt.legend(loc = 'best', labels = ['N('+str(mu[i])+','+str(sd[i])+')' for i in range(4)], fontsize = 12)
plt.show()
```



- Back to scatter plots - Consider we have the same data as before (Population size and proportion of murders to 100K) but now from 2 different regions in the USA.

```
Pop1 = [3100, 1058, 5814, 812,
        7333, 18076, 11860, 931, 472]
Murder1 = [3.1, 2.7, 3.3, 3.3,
           5.2, 10.9, 6.1, 2.4, 5.5]

Pop2 = [3615, 2110, 579, 8277, 4931,
        3387, 3806, 4122, 2341, 5441,
        2715, 2816, 4173, 12237, 4981, 1799]
Murder2 = [15.1, 10.1, 6.2, 10.7, 13.9,
           10.6, 13.2, 8.5, 12.5, 11.1,
           6.4, 11.6, 11, 12.2, 9.5, 6.7]
```

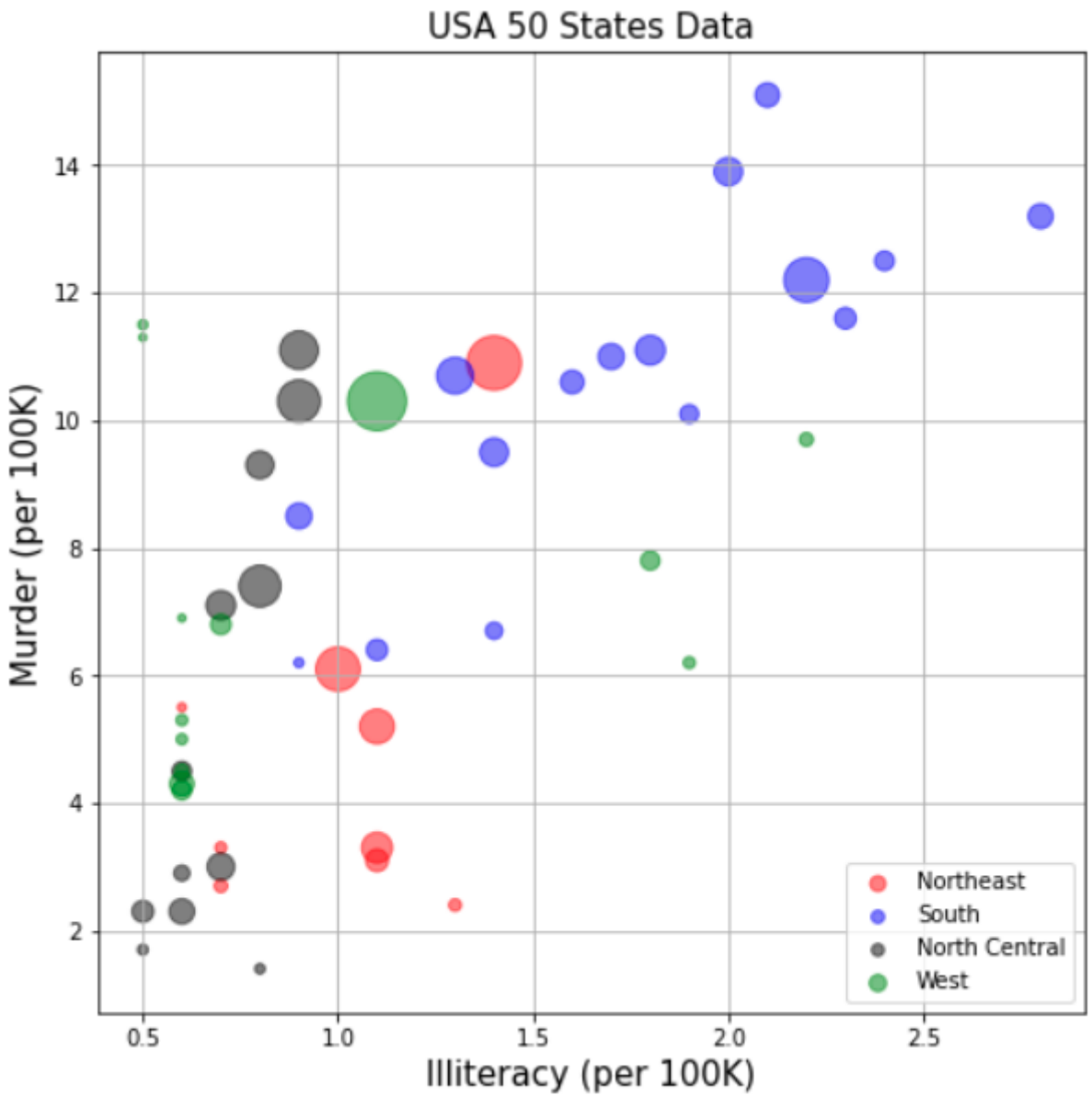
```
plt.plot(Pop1, Murder1, 'ro', alpha = 0.5)
plt.plot(Pop2, Murder2, 'go', alpha = 0.5)
plt.xlabel('Population size')
plt.ylabel('Murder (per 100K)')
plt.legend(loc = "best", labels = ['Region 1', 'Region 2'])
plt.grid()
plt.show()
```



- What about data of murder and illiteracy per 100K and population size for 4 different regions?

```
Pop1 = [3100, 1058, 5814, 812, 7333, 18076, 11860, 931, 472]
Pop2 = [3615, 2110, 579, 8277, 4931, 3387, 3806, 4122, 2341, 5441, 2715, 2816, 4173, 12237, 4981, 1799]
Pop3 = [11197, 5313, 2861, 2280, 9111, 3921, 4767, 1544, 637, 10735, 681, 4589]
Pop4 = [365, 2212, 21198, 2541, 868, 813, 746, 590, 1144, 2284, 1203, 3559, 376]

Murder1 = [3.1, 2.7, 3.3, 3.3, 5.2, 10.9, 6.1, 2.4, 5.5]
Murder2 = [15.1, 10.1, 6.2, 10.7, 13.9, 10.6, 13.2, 8.5, 12.5, 11.1, 6.4, 11.6, 11, 12.2, 9.5, 6.7]
Murder3 = [10.3, 7.1, 2.3, 4.5, 11.1, 2.3, 9.3, 2.9, 1.4, 7.4, 1.7, 3]
Murder4 = [11.3, 7.8, 10.3, 6.8, 6.2, 5.3, 5, 11.5, 9.7, 4.2, 4.5, 4.3, 6.9]

Illiteracy1 = [1.1, 0.7, 1.1, 0.7, 1.1, 1.4, 1, 1.3, 0.6]
Illiteracy2 = [2.1, 1.9, 0.9, 1.3, 2, 1.6, 2.8, 0.9, 2.4, 1.8, 1.1, 2.3, 1.7, 2.2, 1.4, 1.4]
Illiteracy3 = [0.9, 0.7, 0.5, 0.6, 0.9, 0.6, 0.8, 0.6, 0.8, 0.8, 0.5, 0.7]
Illiteracy4 = [.5, 1.8, 1.1, 0.7, 1.9, 0.6, 0.6, 0.5, 2.2, 0.6, 0.6, 0.6, 0.6]

# 1 - Northeast
# 2 - South
# 3 - North Central
# 4 - West
```

```
plt.figure(figsize=(8,8))
plt.scatter(Illiteracy1, Murder1,
            s = [pop/30 for pop in Pop1],
            c = 'red',
            alpha = 0.5)
plt.scatter(Illiteracy2, Murder2,
            s = [pop/30 for pop in Pop2],
            c = 'blue',
            alpha = 0.5)
plt.scatter(Illiteracy3, Murder3,
            s = [pop/30 for pop in Pop3],
            c = 'black',
            alpha = 0.5)
plt.scatter(Illiteracy4, Murder4,
            s = [pop/30 for pop in Pop4],
            c = 'green',
            alpha = 0.5)

plt.xlabel('Illiteracy (per 100K)', size = 15)
plt.ylabel('Murder (per 100K)', size = 15)
plt.title('USA 50 States Data', size = 15)
plt.grid()
plt.legend(labels = ['Northeast', 'South', 'North Central', 'West'], loc = 'lower right', markerscale = 0.4)
plt.show()
```


USA 50 States Data
Murder (per 100K)
Illiteracy (per 100K)
14
12
10
8
6
4
2
0.5
1.0
1.5
2.0
2.5
Northeast
South
North Central
West

Jupyter Time!