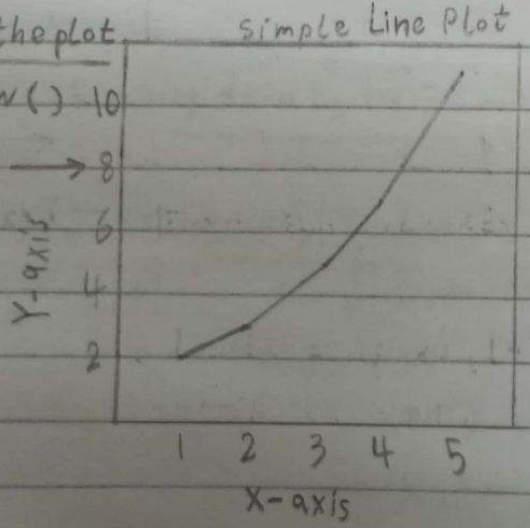
Matplotlib

Pyplot

* Matplotlib :- graph plotting library that serves as a visualization utility

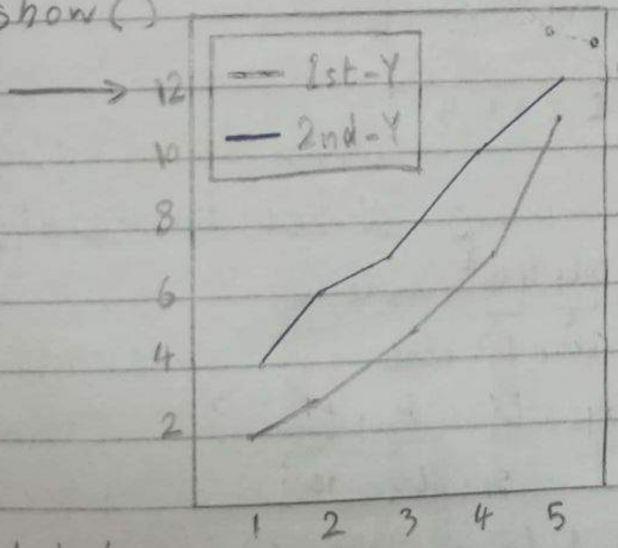
* Pyplot

```
import matplotlib.pyplot asplt
    (or)
From matplotlib import pyplot as plt
* create some data
X = [1, 2, 3, 4, 5]
Y = [2, 3, 5, 7, 11]
* plot the data
plt.plot(x,y)
* Add labels and titles
plt.xlabel('x-axis')
plt.ylabel('Y-axis')
plt.title('simple Line Plot',fontdict={'Fontname':'....'
                                       ,'Fontsize': 20})
*Display the plot
plt.show()
```



* To Add another Line to the plot

```
x=[1,2,3,4,5]
Y_1=[2,3,5,7,11]

plt.plot(x,Y_1)

Y_2=[4,6,7,10,12]
plt.plot(x,Y_2)
plt.show()
```



* Plot Labels

```
plt Legend(['1st_Y', '2nd-Y']) → —1st-Y  لإضافة
                                  —2nd-Y
```

لتسهيل التمييز بين الخطين بإضافتم بالترتيب

(or)

```
plt.plot(x,Y-1, Label='1st_Y')
plt.plot(x,Y_2, Label='2nd-Y')
plt.Legend()
```

`plt.axis([0, 6, 0, 20])` → The **`axis()`** takes a list of [xmin, xmax, ymin, ymax]

Bar plots

* plt styles

print(plt.style.available) → لطباعة الـ styles المتاحة

مثلاً

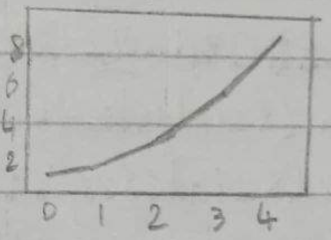
plt.style.use('Fivethirtyeight')

← بعض الـ style تحتوي على Format و grid و Linewidth

* x & y ticks

→ إذا مثلاً الـ x-axis كـ (0.5, 1.0, 1.5) وأردنا التحكم في الـ values الظاهرة

```
plt.xticks([0,1,2,3,4])
plt.yticks([0,2,4,6,8])
```

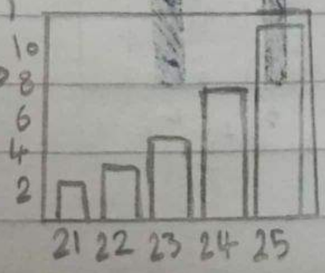


* Save as PNG

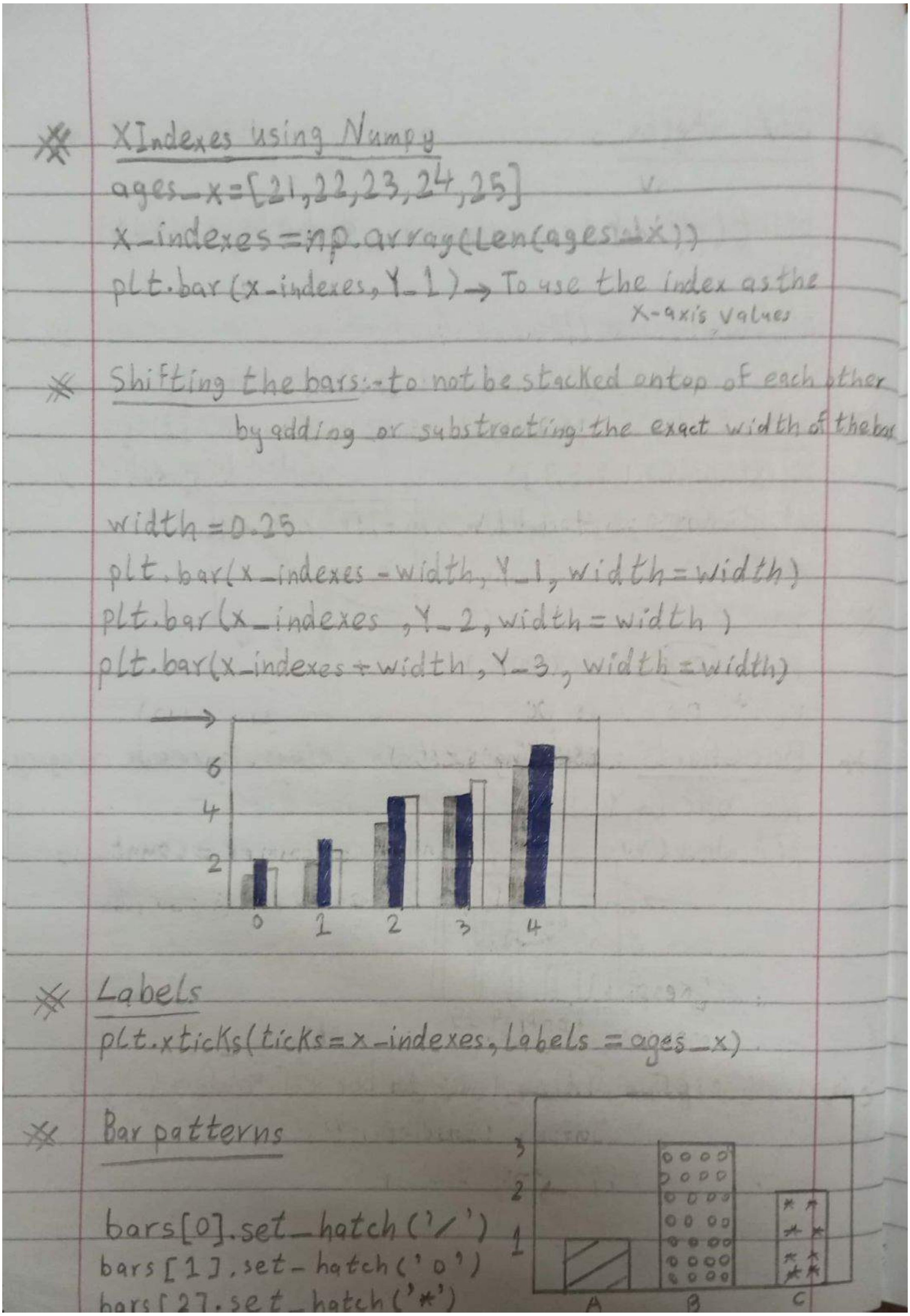
```
plt.savefig('plot.png')
```

* Bar Charts : plotting certain values For each category

```
plt.bar(x, Y_1)
plt.show()
```



← ويمكن طباعة كذه bar على نفس الرسمة لكن ممكن مش يظهروا بوضوح


6
4
2
0
1
2
3
4
3
2
1
A
B
C

※ XIndexes using Numpy

```
ages_x=[21,22,23,24,25]
x_indexes=np.array(Len(ages_x))
plt.bar(x-indexes,Y_1) → To use the index as the
                          X-axis values
```

※ Shifting the bars:- to not be stacked ontop of each other by adding or substracting the exact width of the bar

```
width=0.25
plt.bar(x_indexes - width, Y_1, width=width)
plt.bar(x_indexes ,Y_2, width=width )
plt.bar(x_indexes + width , Y_3 , width=width)
```

※ Labels

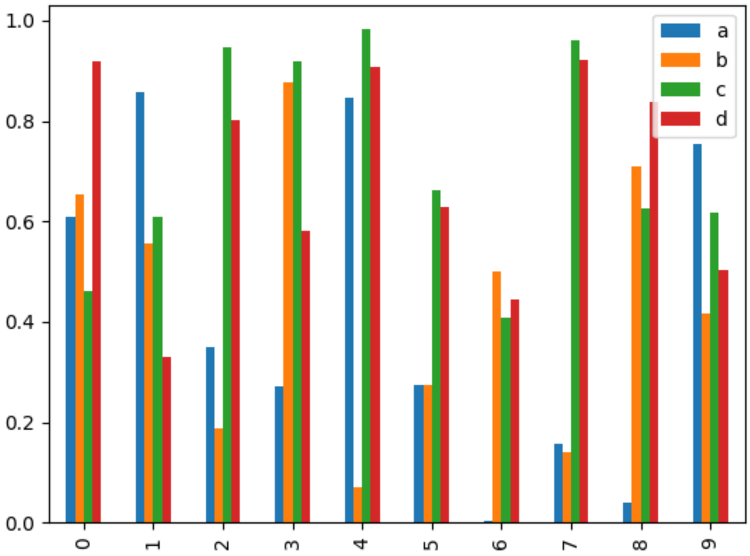
```
plt.xticks(ticks=x-indexes, Labels =ages_x)
```

※ Bar patterns

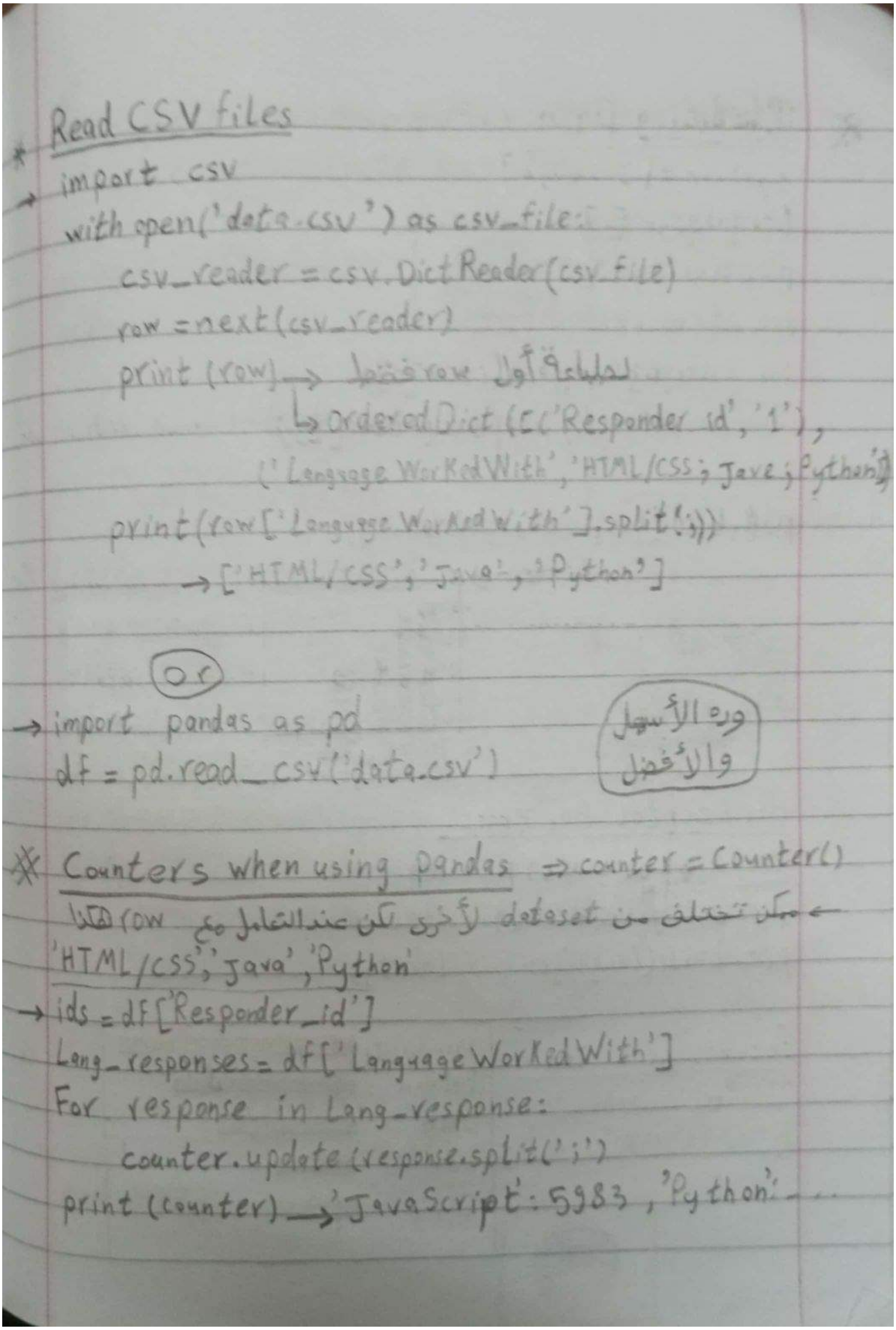
```
bars[0].set_hatch('/')
bars[1].set-hatch('o')
bars[2].set_hatch('*')
```

#Another way to plot several bars

```
X = np.random.rand(10, 4)
df = pd.DataFrame(X, columns=['a', 'b', 'c', 'd'])
df.plot(kind='bar')
```



Read CSV files



Read CSV files

```
* import csv
→ with open('data.csv') as csv_file:
      csv_reader = csv.DictReader(csv_file)
      row = next(csv_reader)
      print (row) → هيطلعلي أول row فقط
                 ↳ orderedDict ([('Responder id', '1'),
             ('Language WorKedWith', 'HTML/CSS; Java; Python')])
     print(row['Language WorKed With'].split(';'))
         → ['HTML/CSS', 'Java', 'Python']
```

or

```
→ import pandas as pd
df = pd.read_csv('data.csv')
```

وده الأسهل
والأفضل

* Counters when using pandas ⇒ counter = Counter()

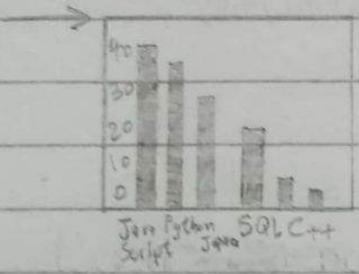
ممكن تختلف من dataset لأخرى لكن عند التعامل مع row هكذا ←

'HTML/CSS', 'Java', 'Python'

```
→ ids = df['Responder_id']
Lang_responses = df['LanguageWorKedWith']
For response in Lang_response:
    counter.update(response.split(';'))
print (counter) → 'JavaScript': 5983, 'Python': ...
```

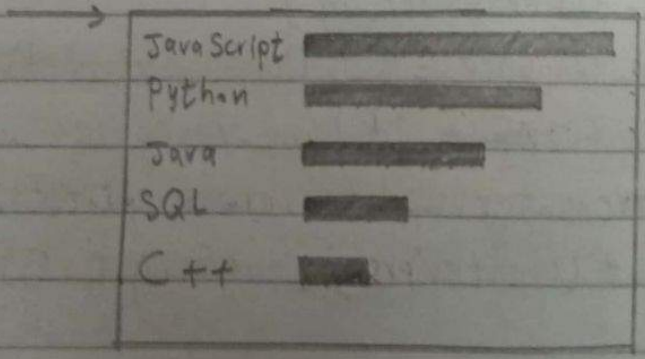
※ Plotting Data (if we continue on the previous dataset)

```
Language = []
popularity = []
For item in counter:
    Languages.append(item[0])
    popularity.append(item[1])

plt.bar(Languages, popularity)
```



※ Horizontal Bar charts

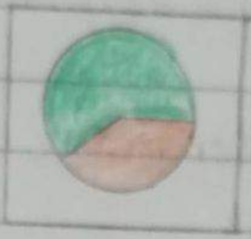
```
Languages.reverse()
popularity.reverse()
plt.barh(Languages, popularity)
```



Pie Charts

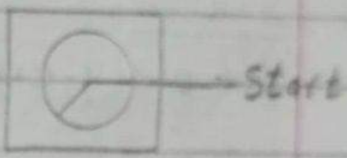
⁂ Pie Charts : shows the propotions that each category contribute to a whole

```
y=np.array([60,40])
plt.pie(y)
```



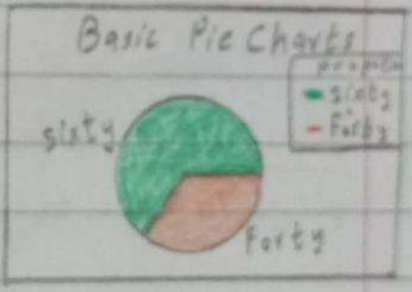
• Notes: بداية رسم أول widge تكون من ال x-axis وتمشي عكس عقارب الساعة

$$\text{size of each wedge} = \frac{x}{\text{sum}(x)}$$



⁂ Labels, titles and Legend

```
y=np.array([60,40])   or ([120,80])
Labels=['Sixty', 'Forty']
plt.pie(y,Labels=Labels)
plt.title("Basic Pie Chart")
plt.Legend(title="propotions")
```



⁂ Wedge

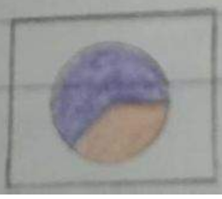
```
plt.pie(y,Labels=Labels, wedgeprops={'edgecolor'
                                     :'black'})
```

لعمل حواف سوده حول كل slice

⁂ Colors

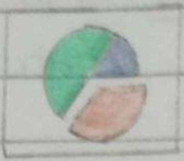
```
y=np.array([60,40])
mycolors=['purple','orange']   or ['b', "#4CAF50"]
plt.pie(y, colors= mycolors]
```

b → blue, #4CAF50 → Green



※ Explode

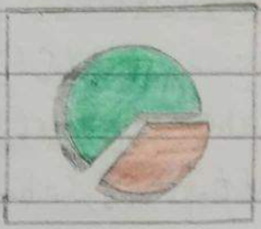
```
explode =[0,0,0.2]
plt.pie(y, explode = explode)
```



- explode parameter must be an array with one value For each wedge

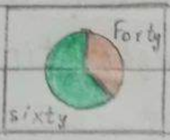
※ Shadow

```
plt.pie(y, explode=explode, shadow = True)
```



※ Start angle

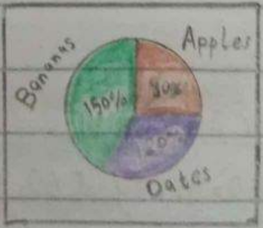
```
plt.pie(y, Label =Labels, startangle= 90)
```



※ Percentage

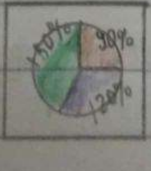
```
plt.pie(y, Label =Labels, autopct='%1.1f%%')
```

Format

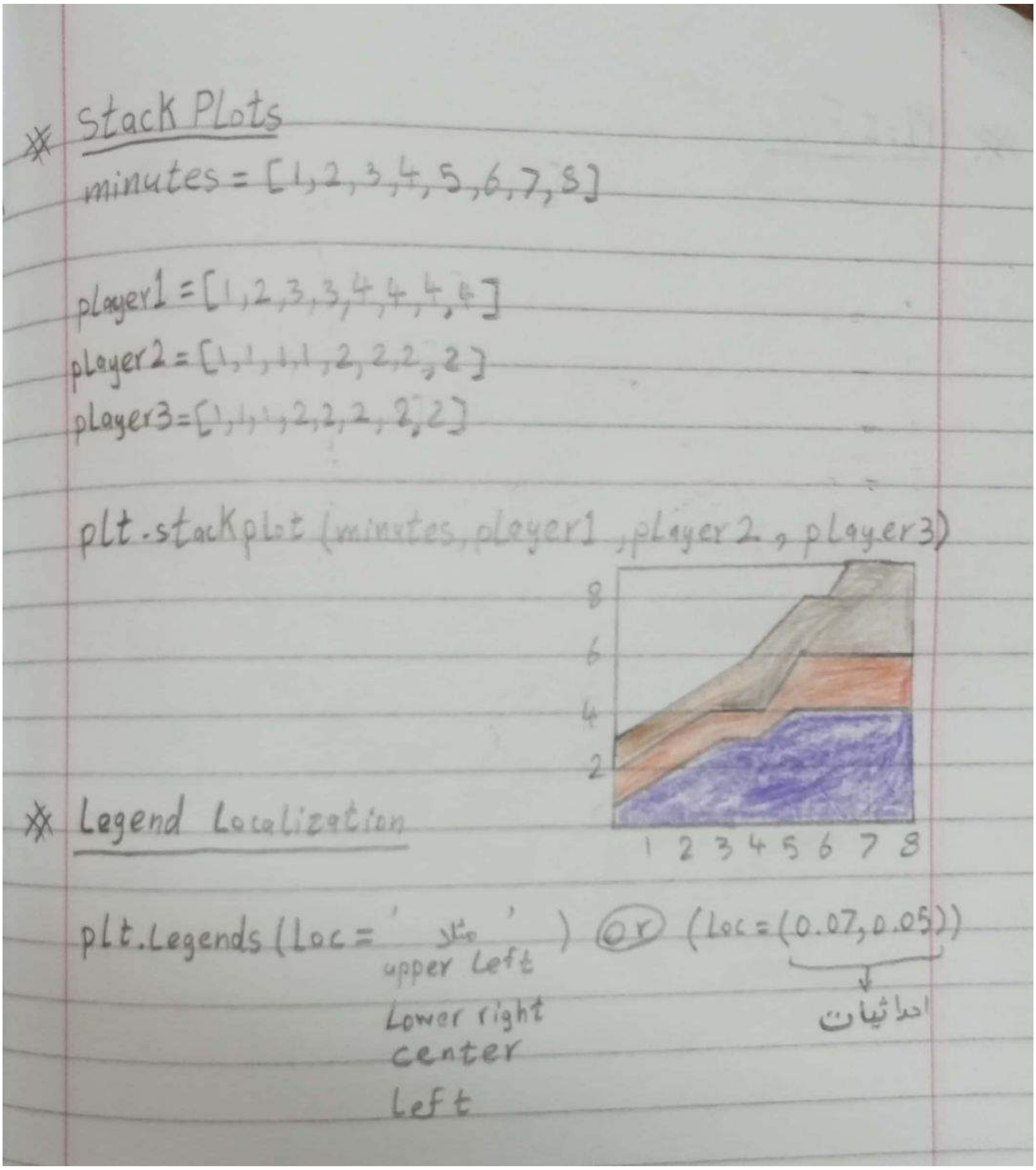


※ Percentage distance

```
plt.pie(y, autopct='%1.1f%%'
        , pctdistance=0.9)
```



Stack Plots



※ Stack Plots

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

player1 = [1,2,3,3,4,4,4,4]

player2 = [1,1,1,1,2,2,2,2]

player3 = [1,1,1,2,2,2,2,2]

plt.stackplot (minutes, player1 ,player 2 , player3)

8
6
4
2
1 2 3 4 5 6 7 8

※ Legend Localization

plt.Legends (Loc= ' مثلا ') or (Loc=(0.07,0.05))

upper Left

Lower right

center

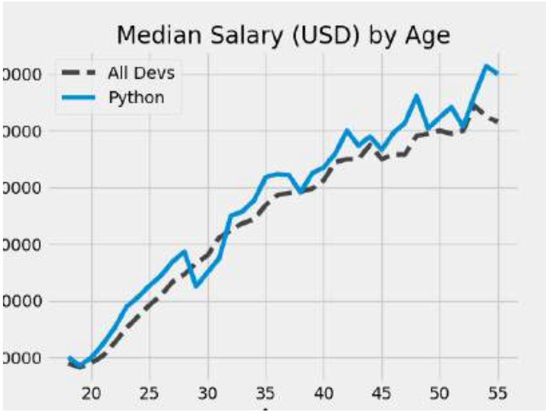
Left

احداثيات

Plot Fills

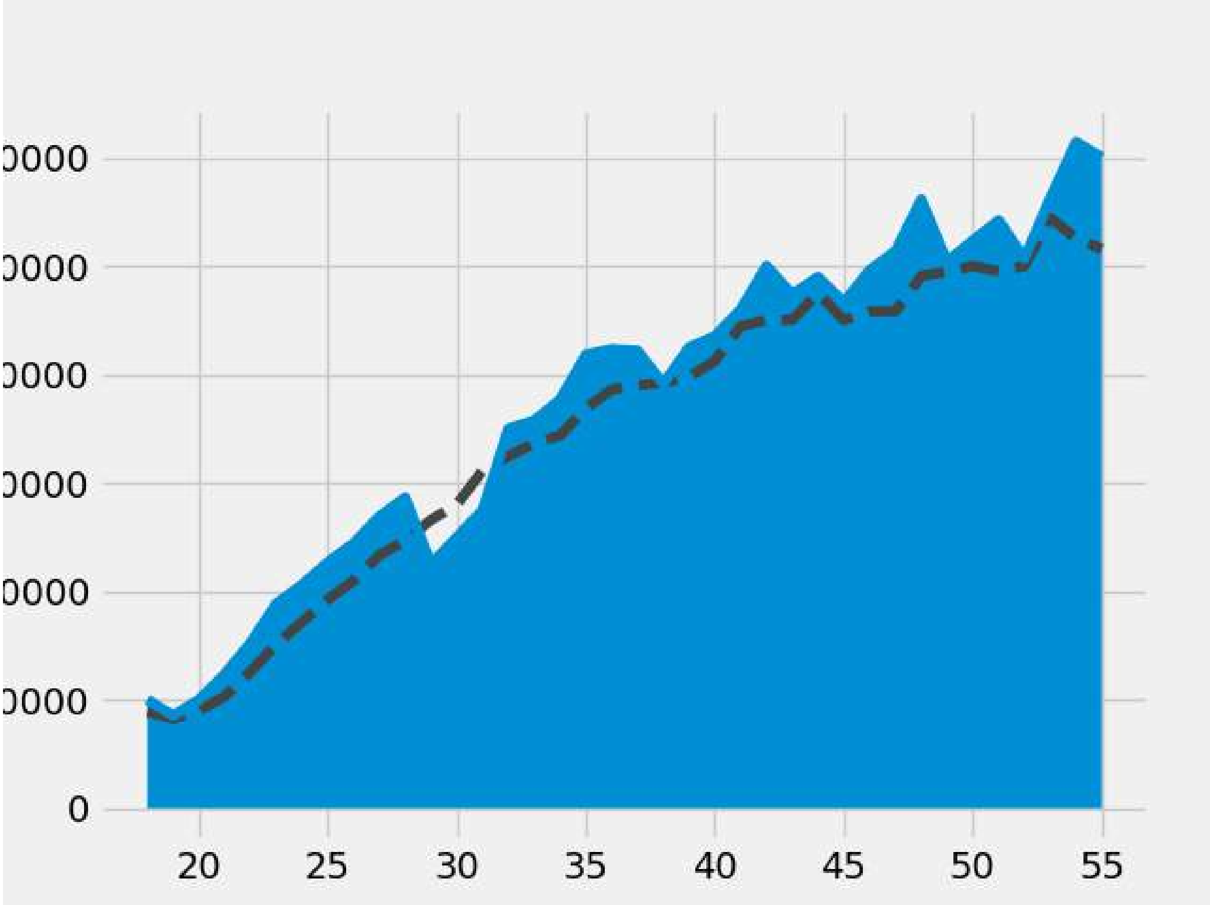
```
import pandas as pd
from matplotlib import pyplot as plt

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

```
ages = data['Age']
dev_salaries = data['All_Devs']
py_salaries = data['Python']
js_salaries = data['JavaScript']

plt.plot(ages, dev_salaries, color='#444444',
         linestyle='--', label='All Devs')

plt.plot(ages, py_salaries, label='Python')

overall_median = 57287

plt.legend()

plt.title('Median Salary (USD) by Age')
plt.xlabel('Ages')
plt.ylabel('Median Salary (USD)')

plt.tight_layout()

plt.show()
```



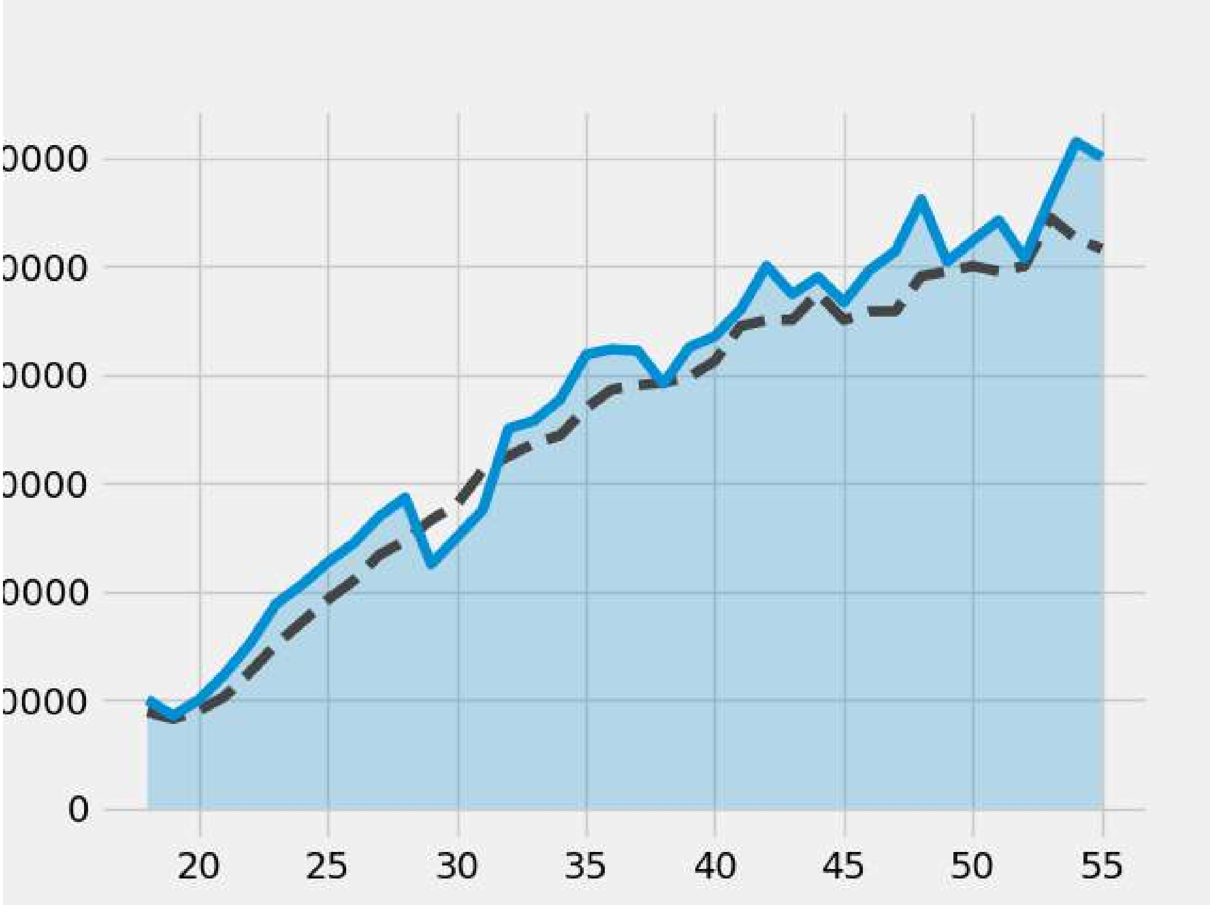
```
plt.fill_between(ages, py_salaries)
```

→ To fill between **py_salaries** all the way to the bottom but may interfere with other lines



```
plt.fill_between(ages, py_salaries, alpha=0.25)
```

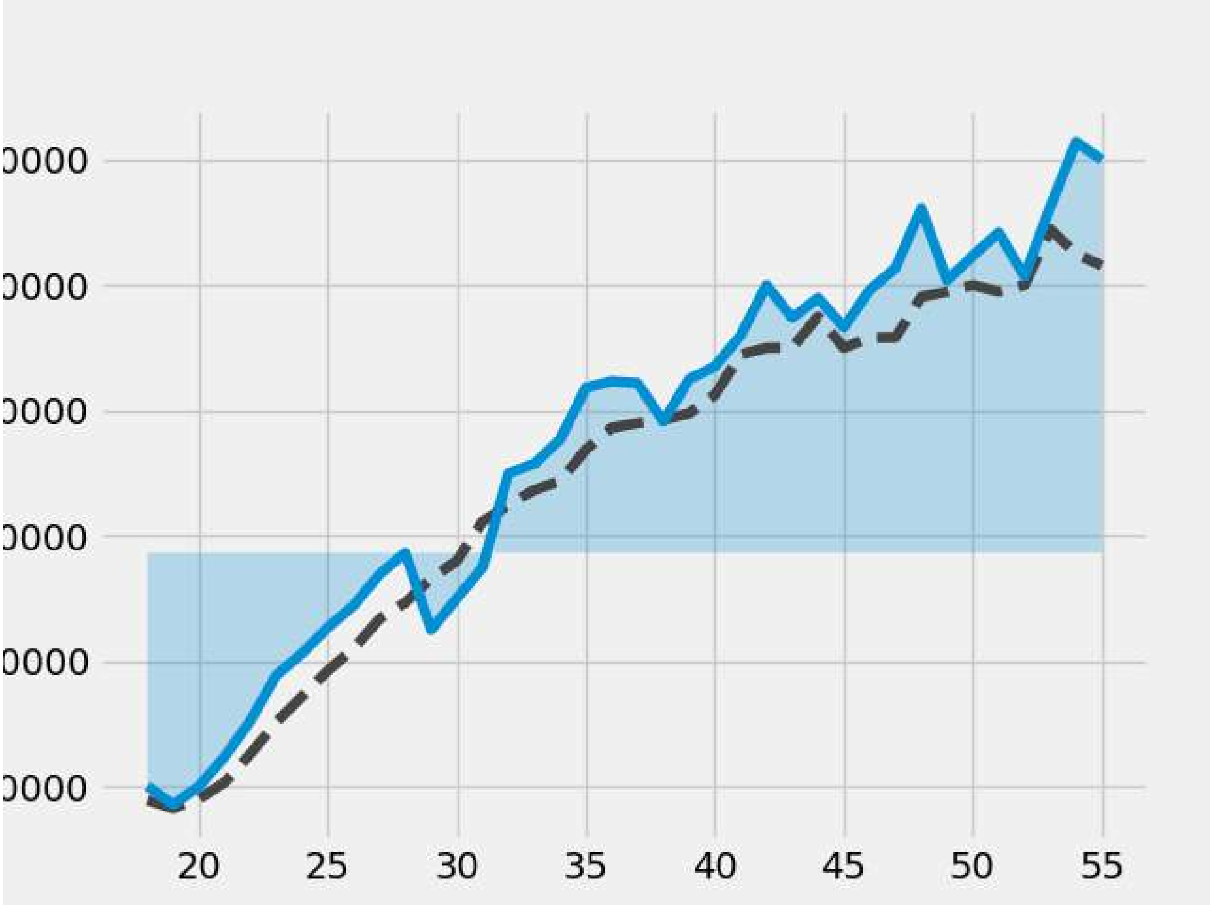
→ To see through the filling a bit better



```
plt.fill_between(ages, py_salaries, overall_median, alpha=0.2
```

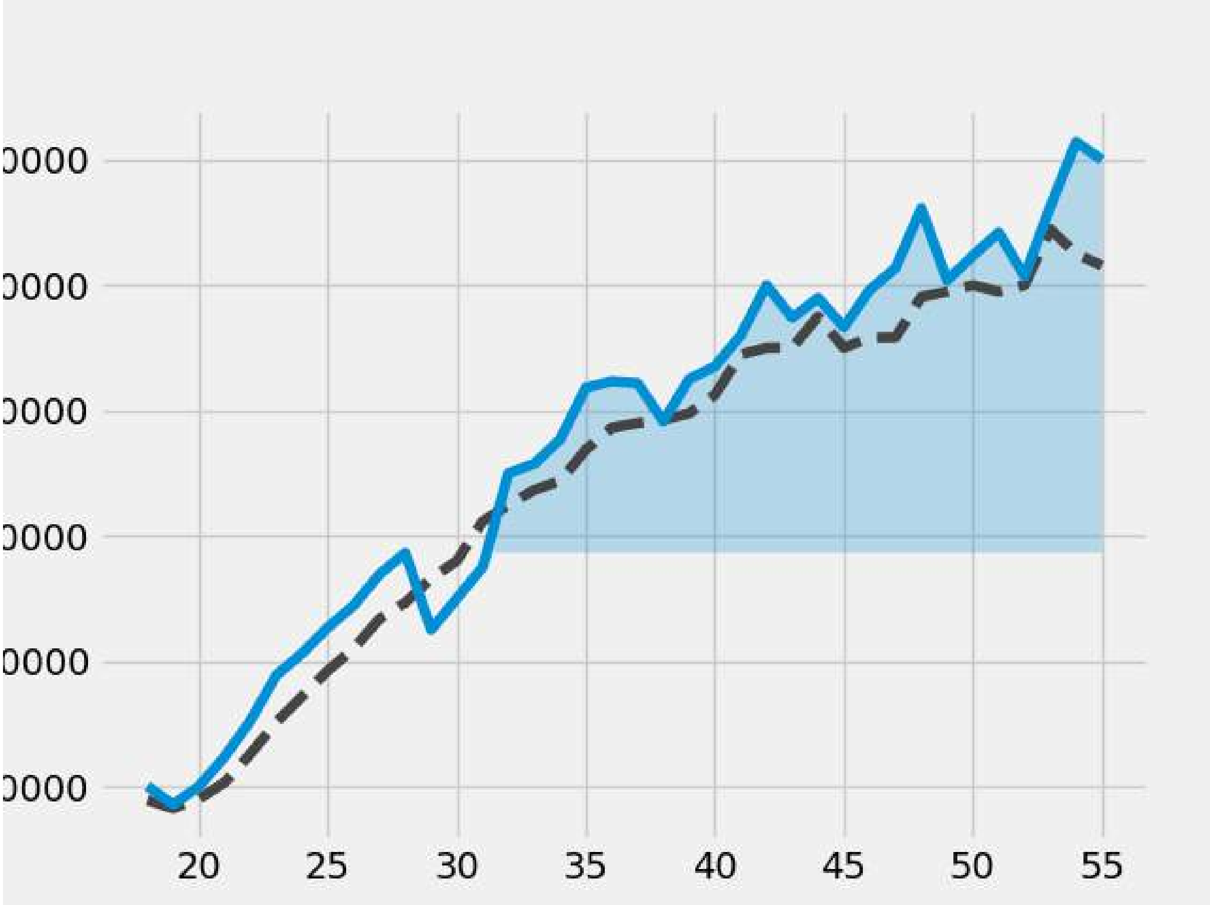
`overall_median` as the third parameter → To fill to a specific number(ex. to show where it crossed the median salary)

```
plt.fill_between(ages, py_salaries, overall_median,
                                 where=(py_salaries > overall_
                                 interpolate=True, alpha=0.25
```



`where = (py_salaries > dev_salaries)` → to fill and no longer plot below the **overall_median point**

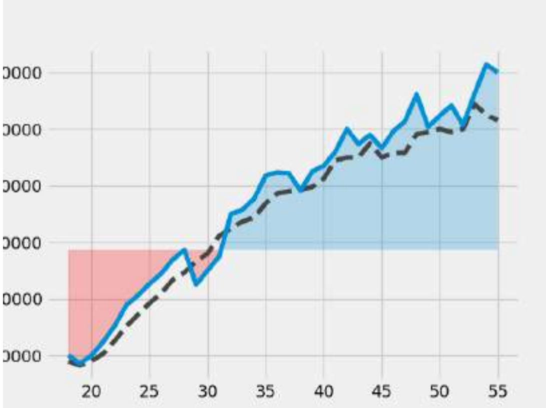
`interpolate= True` → **to make sure that certain x intersections don’t get clicked and all of the regions are filled correctly**



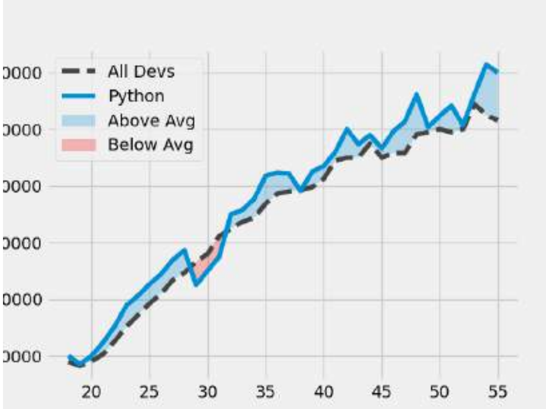
```
plt.fill_between(ages, py_salaries, overall_median,
                                 where=(py_salaries > overall_
                                 interpolate=True, alpha=0.25
                                 label='Above Avg')

plt.fill_between(ages, py_salaries, overall_median,
                                 where=(py_salaries <= overal
                                 interpolate=True, color='red
                                 label='Below Avg')
```

→ to also fill above the lower half

`color` `=”` `ex.(red)` `“` → to specify a certain color



#When changing `where=(py_salaries > dev_salaries)` to `where=(py_salaries > dev_salaries)` and the same to the lower half



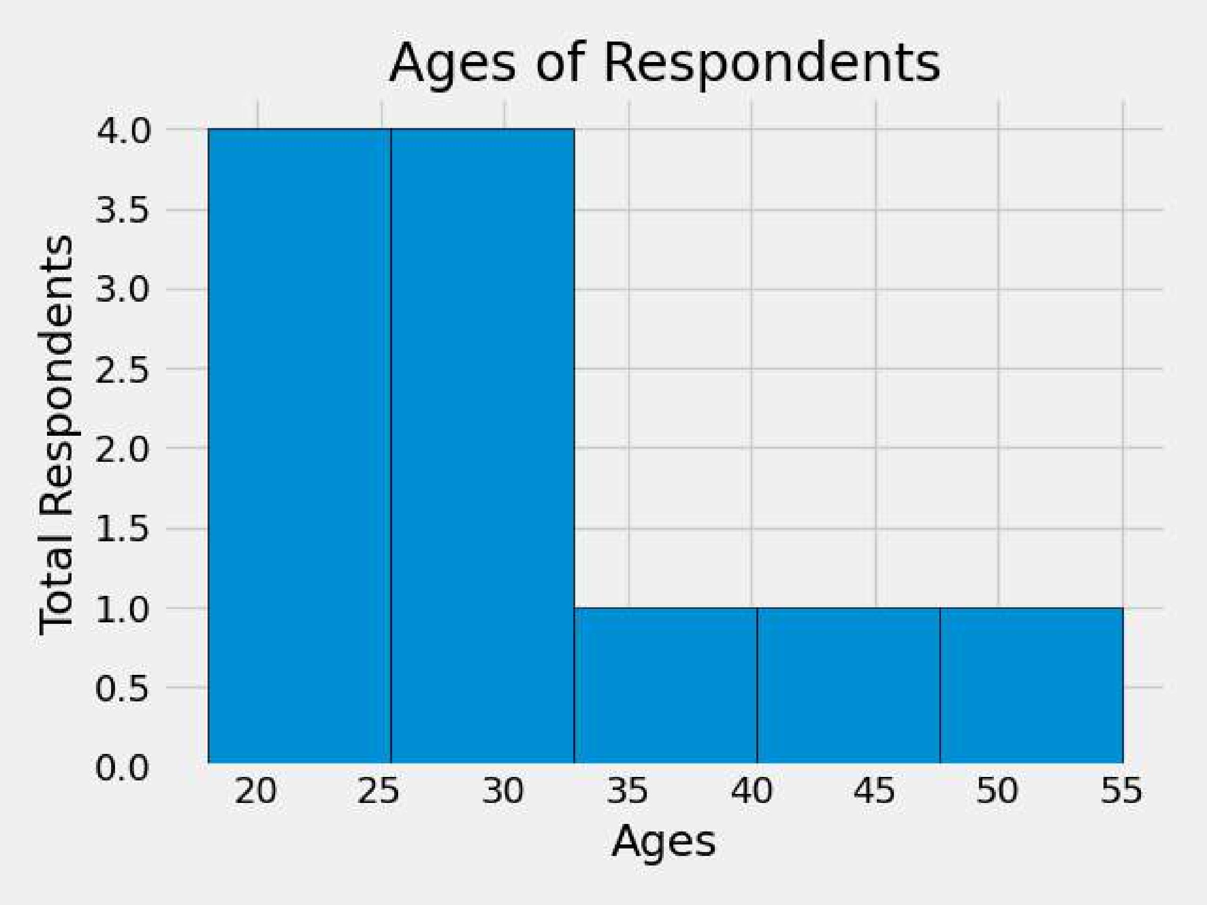
Histograms

```
ages = [18, 19, 21, 25, 26, 26, 30, 32, 38, 45, 55]

plt.hist(ages, bins=5, edgecolor='black')
```

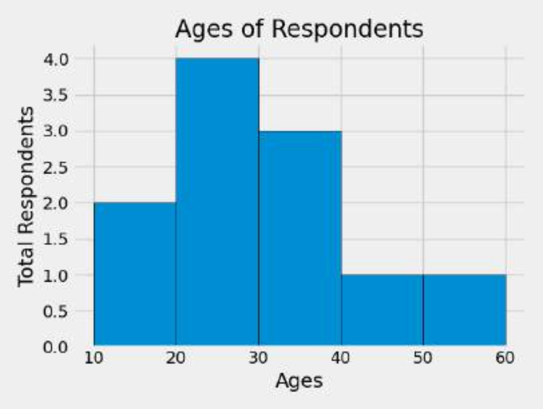
`.hist(……)` → To plot a Histogram

`bins` `=` `integer or a list of values`

`edgecolor` `=` `'black'` → To see the bins a bit more clearly



```
ages = [18, 19, 21, 25, 26, 26, 30, 32, 38, 45, 55]
bins = [10, 20, 30, 40, 50, 60]
plt.hist(ages, bins=bins, edgecolor='black')
```



2 people from 10-20

4 people from 20-30

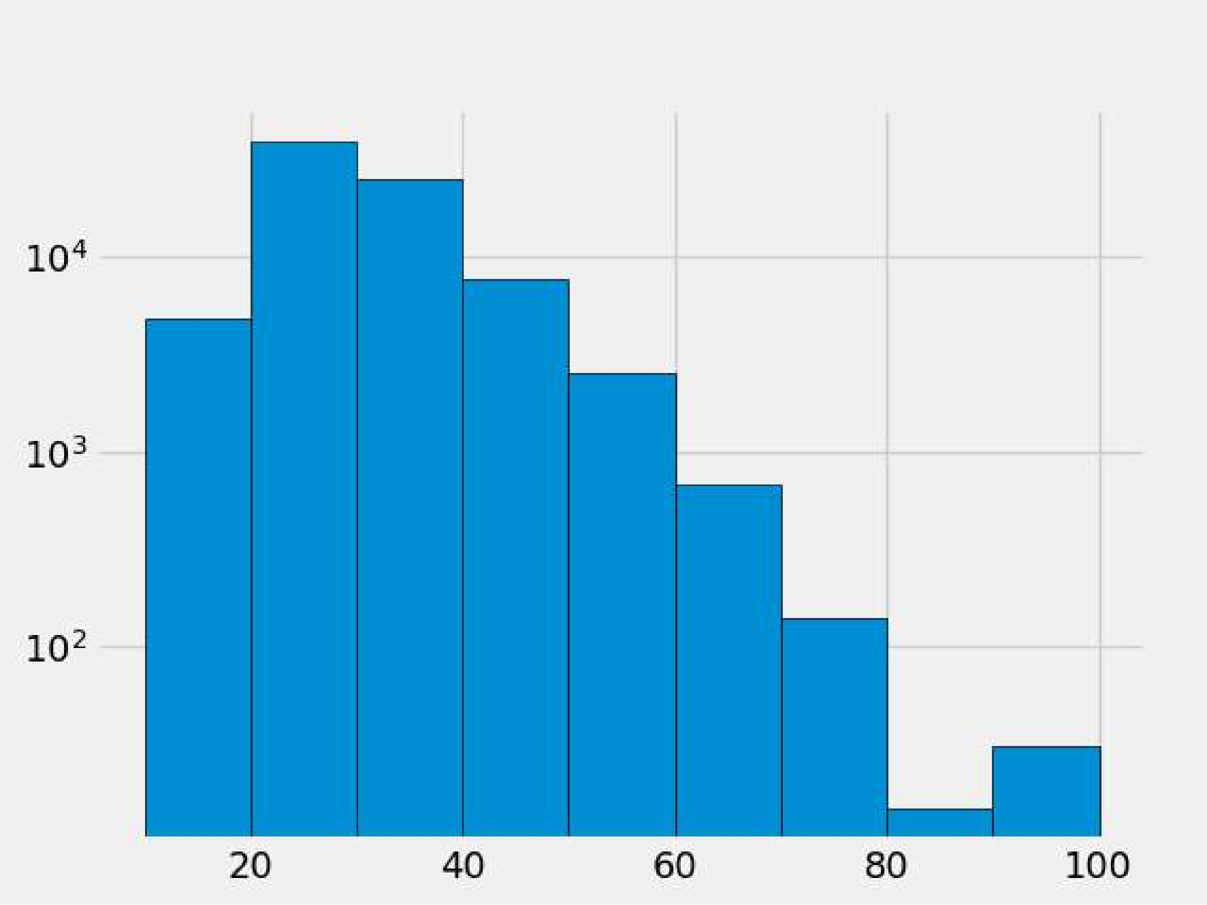
3 people from 30-40

1 people from 40-50

1 people from 50-60

```
data = pd.read_csv('data.csv')
ids = data['Responder_id']
ages = data['Age']

bins = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

plt.hist(ages, bins=bins, edgecolor='black', log=True)
```

`log = True` → To plot on a logarithmic scale



#To plot a vertical line

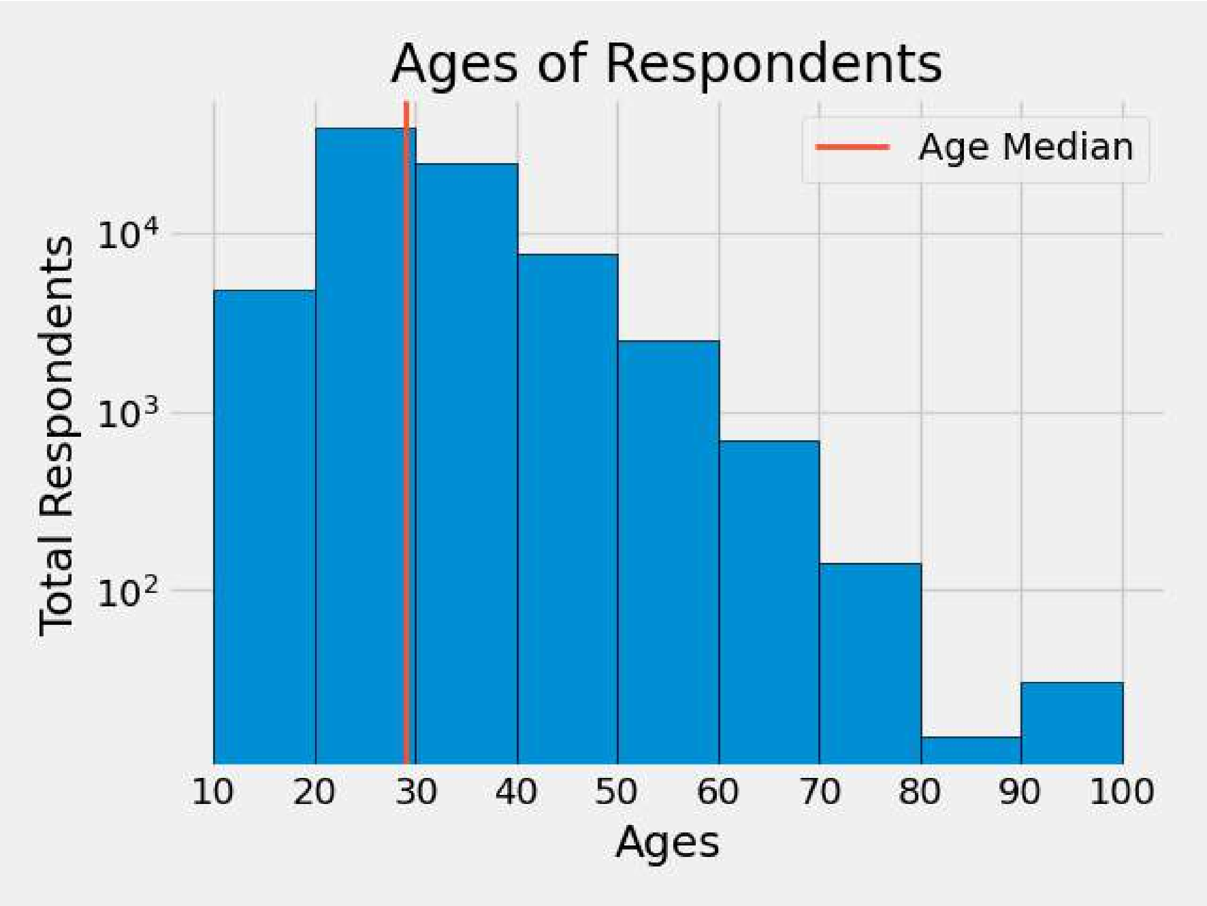
```
plt.hist(ages, bins=bins, edgecolor='black', log=True)

median_age = 29
```

```
color = '#fc4f30'
plt.xticks(bins)

plt.axvline(median_age, color=color, label='Age Median', line
```

`plt.axvline(int,….)` → To access a vertical line

`linewidth` `=` `int` → line thickness

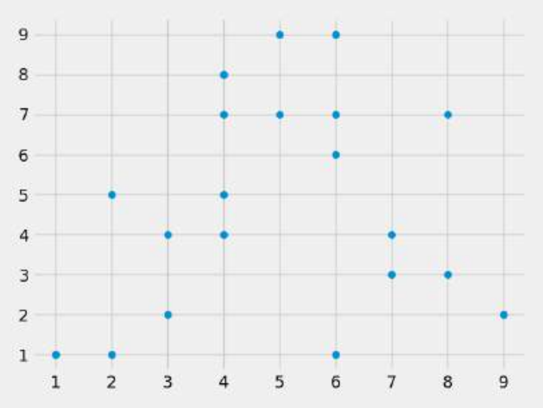
`plt.xticks(bins)` → to set the positions and labels of the ticks on the x-axis to match the bins, ensuring they are evenly spaced and clearly labeled



Scatter Plots

To show a relationship between to sets of values

```
x = [5, 7, 8, 5, 6, 7, 9, 2, 3, 4, 4, 4, 2, 6, 3, 6, 8, 6, 4,
y = [7, 4, 3, 9, 1, 3, 2, 5, 2, 4, 8, 7, 1, 6, 4, 9, 7, 7, 5,
plt.scatter(x, y)
```



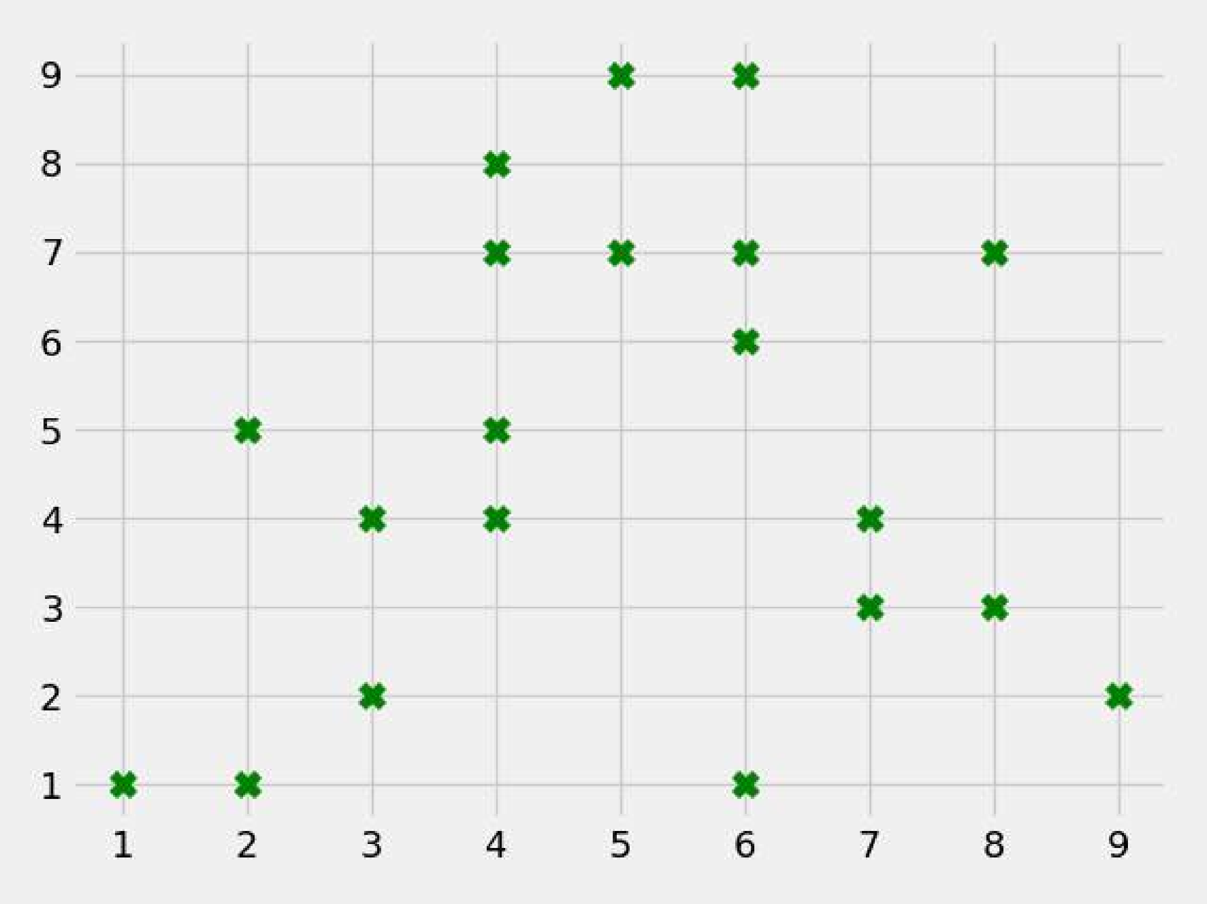
#There is no correlation in this dataset

```
plt.scatter(x, y, s=100, c='green', marker='X')
```

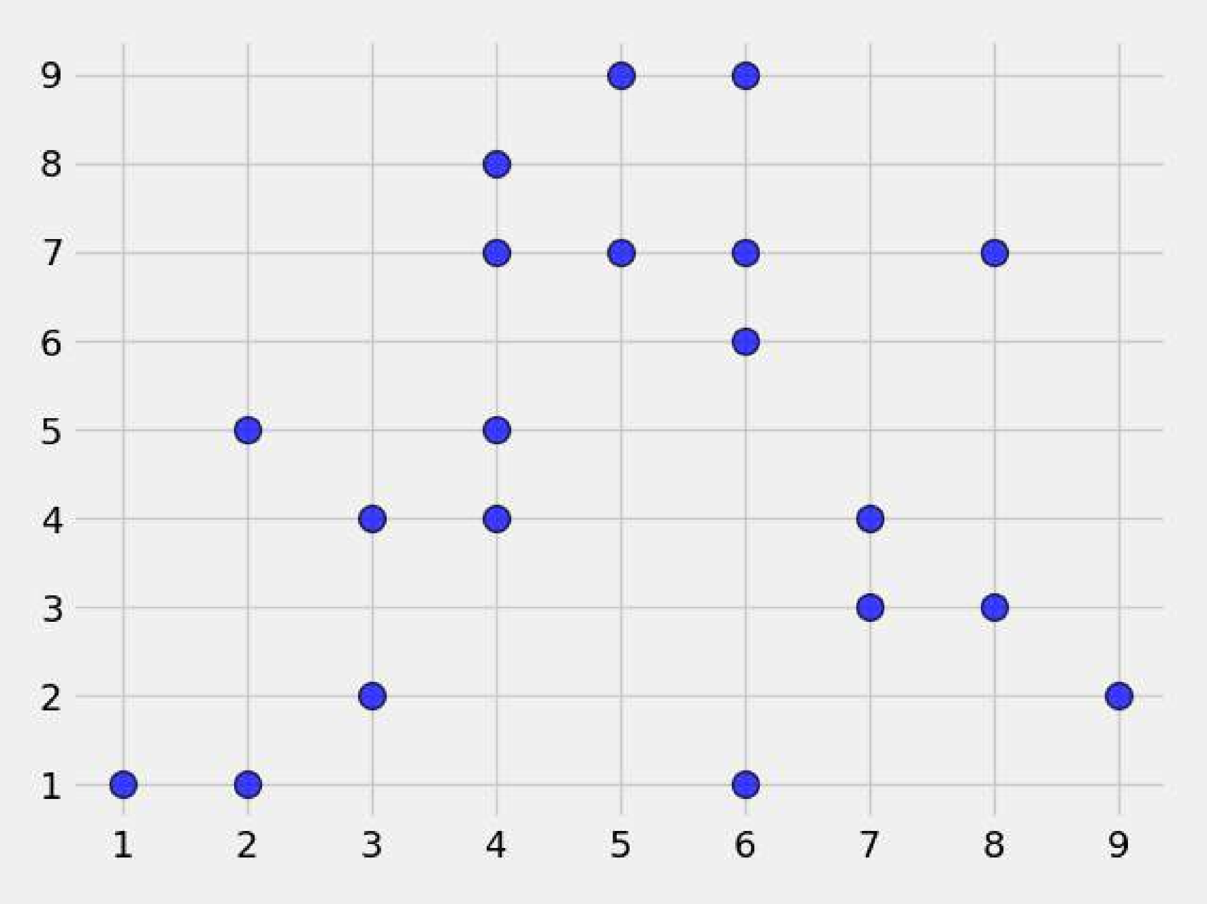
`s=int` → control the size of the circles

`c=’ex. green’` → set a color

`marker=’ex.X’` → set a style



```
plt.scatter(x, y, s=100, c='blue', edgecolors='black', linewi
```



#Having multiple colors and sizes gives us the ability to add more datasets

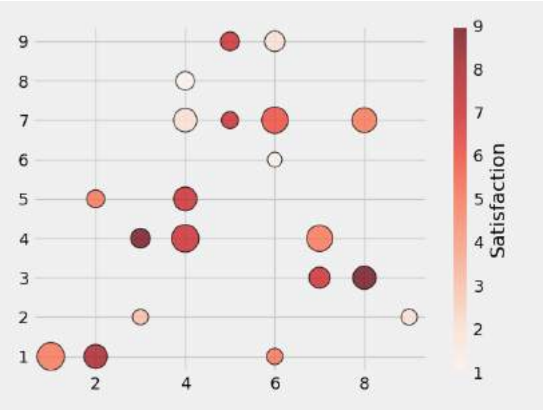
```
x = [5, 7, 8, 5, 6, 7, 9, 2, 3, 4, 4, 4, 2, 6, 3, 6, 8, 6, 4,
y = [7, 4, 3, 9, 1, 3, 2, 5, 2, 4, 8, 7, 1, 6, 4, 9, 7, 7, 5,
colors = [7, 5, 9, 7, 5, 7, 2, 5, 3, 7, 1, 2, 8, 1, 9, 2, 5,
sizes = [209, 486, 381, 255, 191, 315, 185, 228, 174,
         538, 239, 394, 399, 153, 273, 293, 436, 501, 397, 53

plt.scatter(x, y, s=sizes, c=colors, cmap='Reds', edgecolors=

cbar = plt.colorbar()
cbar.set_label('Satisfaction')
```

`c=colors` #colors is a list → it gives the dots a different shade of color depending on their value.

`cmap=’ex.Reds’` → the data points will have varying shades of red based on their values.

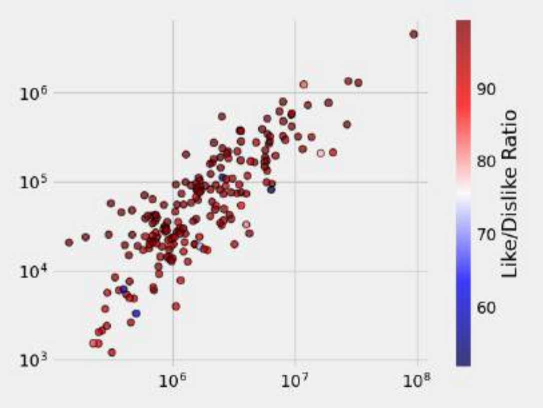
`plt.colorbar()` → To add a color bar legend

`s = sizes` → to customize and vary the size of the points based on another dimension of your data.



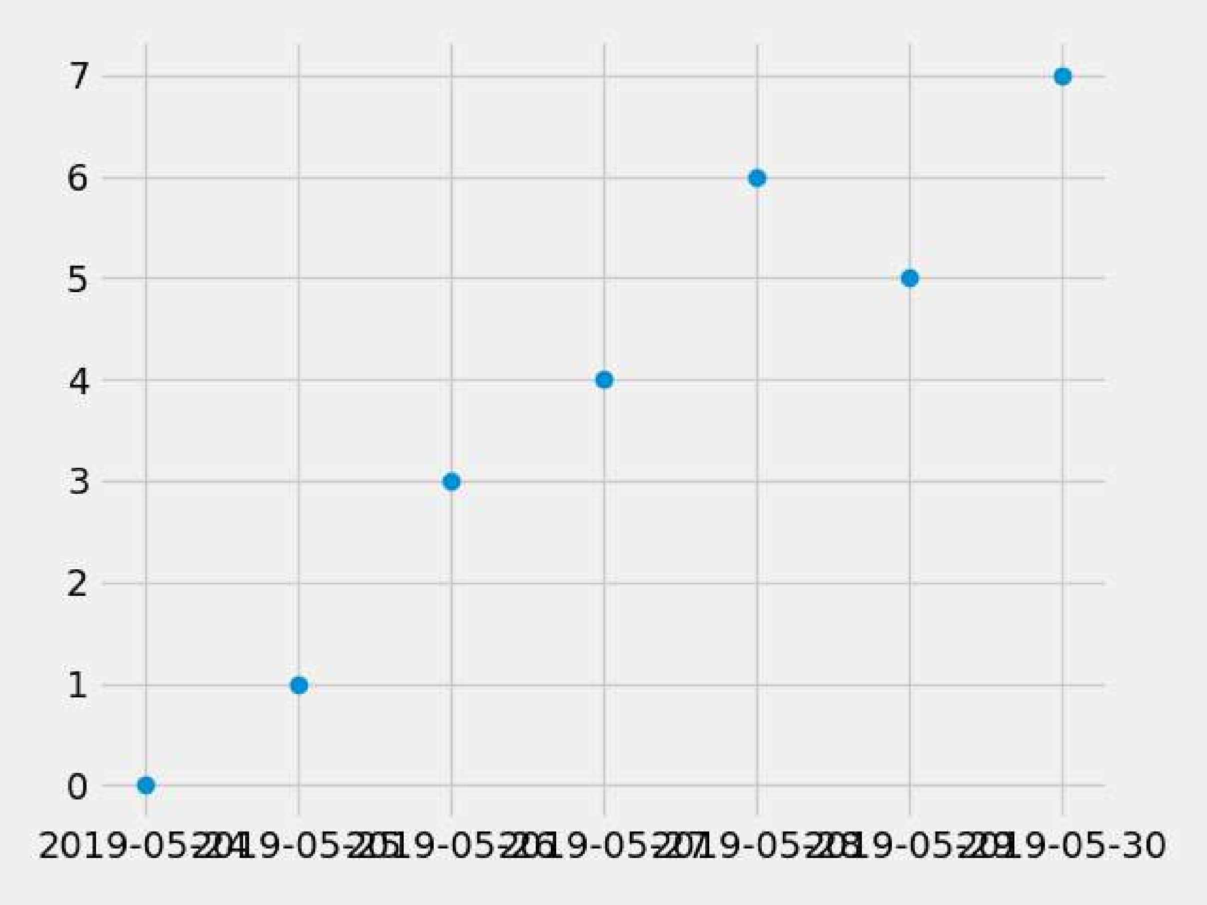
#Real World Data

```
data = pd.read_csv('data3.csv')
view_count = data['view_count']
likes = data['likes']
ratio = data['ratio']

plt.scatter(view_count, likes, c=ratio, cmap="seismic",
            edgecolors='black', linewidths=1, alpha=0.75)
cbar = plt.colorbar()
cbar.set_label('Like/Dislike Ratio')
plt.xscale('log')
plt.yscale('log')
```



Plotting Time Series Data

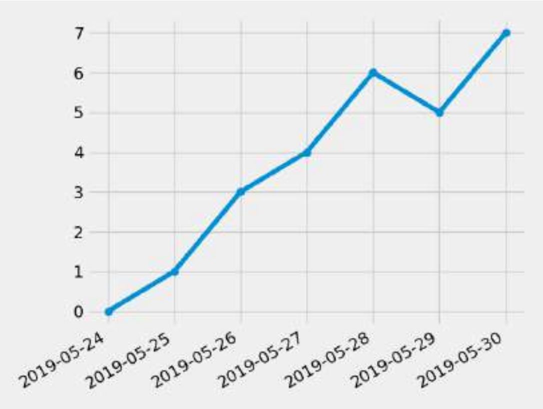
```
import pandas as pd
from datetime import datetime, timedelta
from matplotlib import pyplot as plt
from matplotlib import dates as mpl_dates

dates = [
    datetime(2019, 5, 24),
    datetime(2019, 5, 25),
    datetime(2019, 5, 26),
    datetime(2019, 5, 27),
    datetime(2019, 5, 28),
    datetime(2019, 5, 29),
    datetime(2019, 5, 30)
]

y = [0, 1, 3, 4, 6, 5, 7]

plt.plot_date(dates, y)
```



```
plt.plot_date(dates, y, linestyle='solid')

plt.gcf().autofmt_xdate()
```

`gcf` → get current figure

`plt.gcf().autofmt_xdate()` → Automatically format the x-axis dates



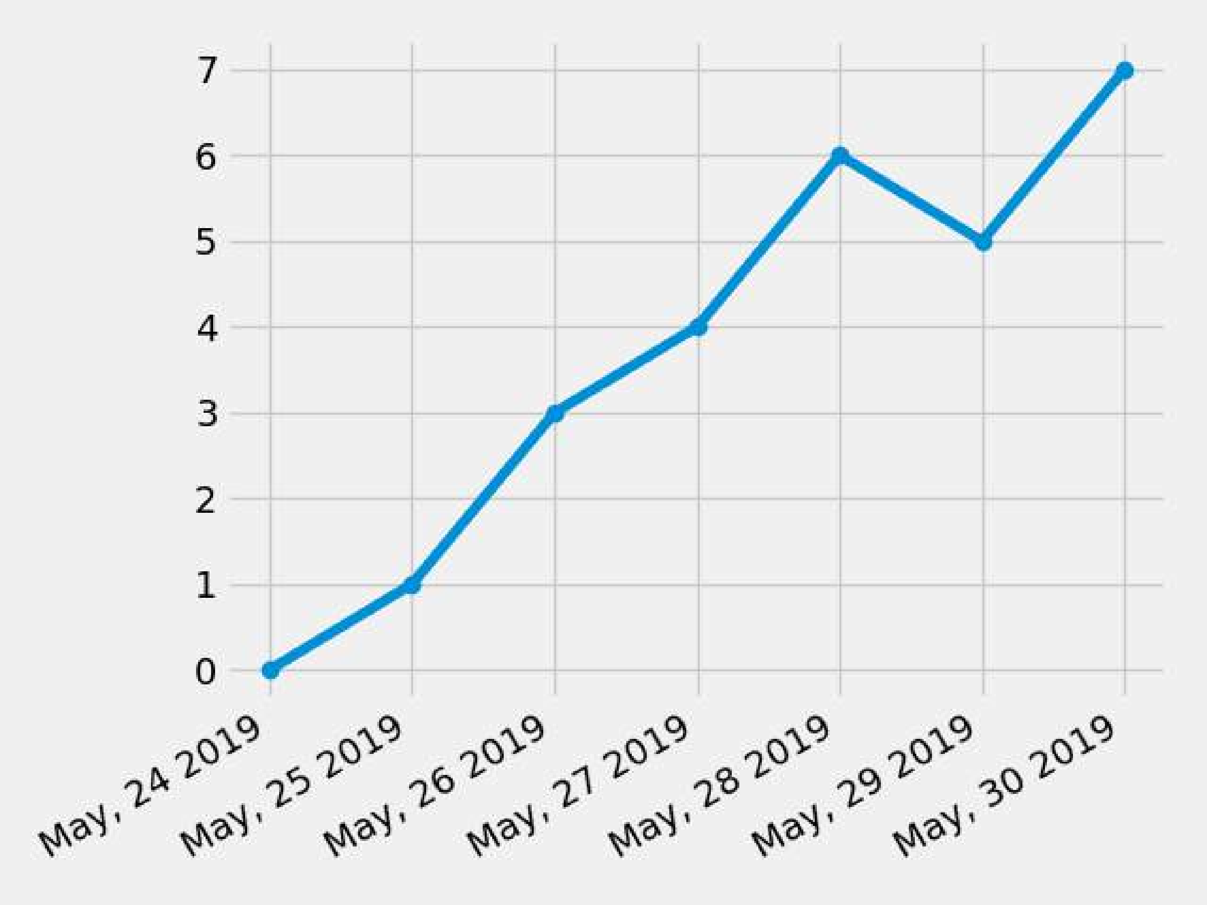
```
date_format = mpl_dates.DateFormatter('%b, %d %Y')

plt.gca().xaxis.set_major_formatter(date_format)
```

`gca` → Get Current Axis

→ This code formats the date on the x-axis

`mpl_dates.DateFormatter( '%b, %d %Y' )` → **This creates a date formatter object that formats dates in the "Month, Day Year" format** (e.g., "Jul, 10 2024").

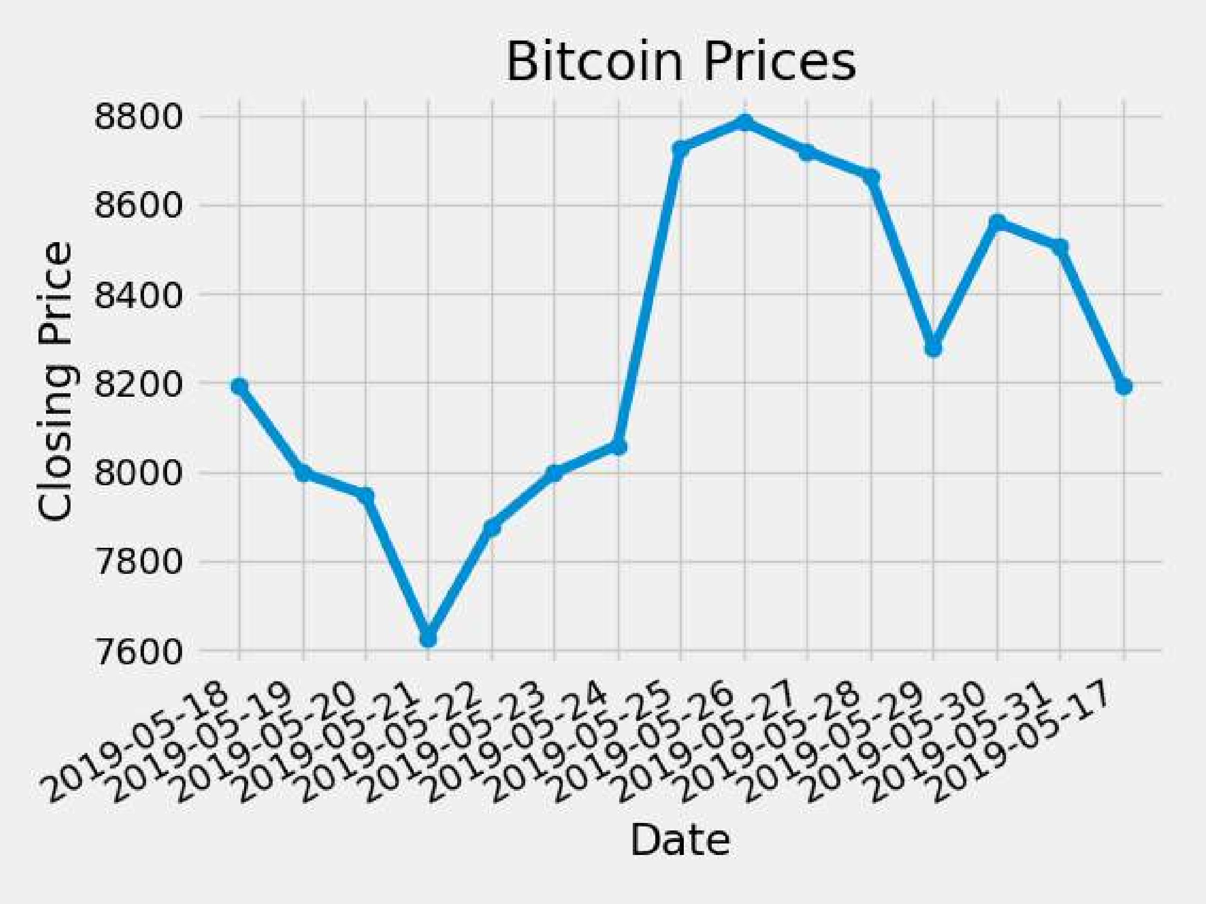
`plt.gca().xaxis.set_major_formatter( date_format )` → **This sets the major formatter of the x-axis to the date format specified by date_format.**



#Real World Data

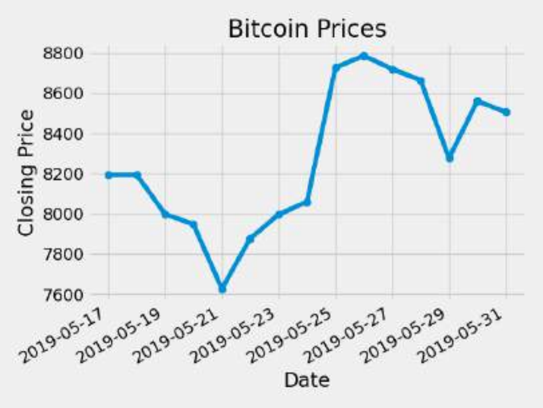
```
data = pd.read_csv('data4.csv')
price_date = data['Date']
price_close = data['Close']
```

```
plt.plot_date(price_date, price_close, linestyle='solid')

plt.gcf().autofmt_xdate()

plt.title('Bitcoin Prices')
plt.xlabel('Date')
plt.ylabel('Closing Price')
```



#Right now the date is String so to convert it to a datetime using Pandas:

> data = pd.read_csv('data4.csv')
>
> data['Date'] = pd.to_datetime(data['Date'])
> data.sort_values('Date', inplace=True)



Plotting Live Data in Real-Time

```
import random
from itertools import count
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# Initialize the values
x_vals = []
y_vals = []

# Create an index generator
index = count()

# Define the animation function
def animate(i):
    x_vals.append(next(index))
    y_vals.append(random.randint(0, 5))

    plt.cla()  # Clear the current axes
    plt.plot(x_vals, y_vals)
    plt.tight_layout()

# Create the animation and assign it to a variable to prevent
ani = FuncAnimation(plt.gcf(), animate, interval=1000)
```

The `animate(i)` function is part of the code used to plot live data in real-time using Matplotlib. Here's a breakdown of what it does:

Append new x and y values: Each time the function is called, it appends a new value to the `x_vals` list (using `next(index)` to get the next index value) and a random integer between 0 and 5 to the `y_vals` list (using `random.randint(0, 5)`).

Clear the current axes: The function clears the current axes with `plt.cla()` to ensure that the plot is refreshed with each new frame, rather than being drawn over the previous frame.

Plot the updated values: After clearing the axes, the function plots the updated lists `x_vals` and `y_vals` using `plt.plot(x_vals, y_vals)`.

Adjust layout: Finally, `plt.tight_layout()` is called to automatically adjust the plot's parameters to give it a cleaner look.

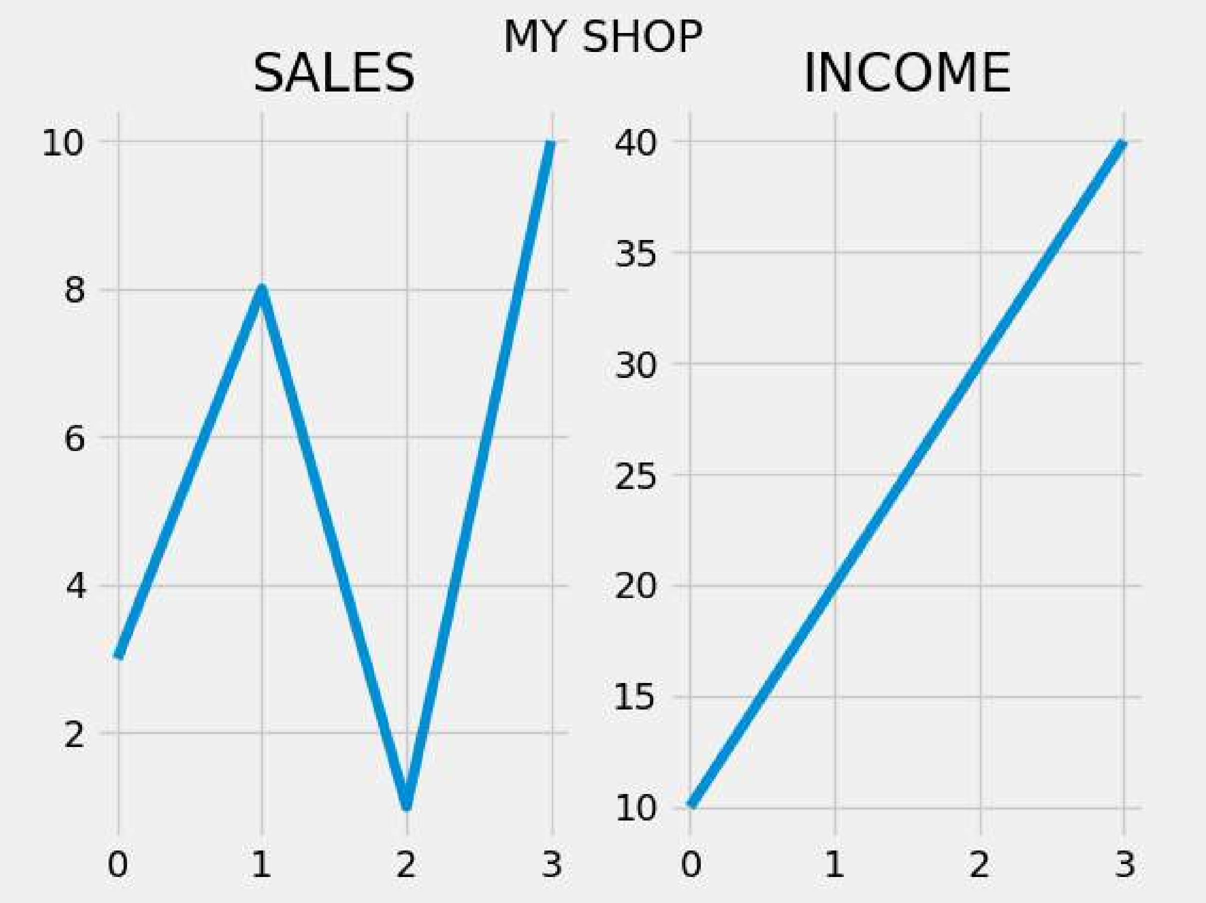
This function is used in conjunction with `FuncAnimation` to create an animation that updates the plot in real-time

`interval=1000` → 1sec

Subplots

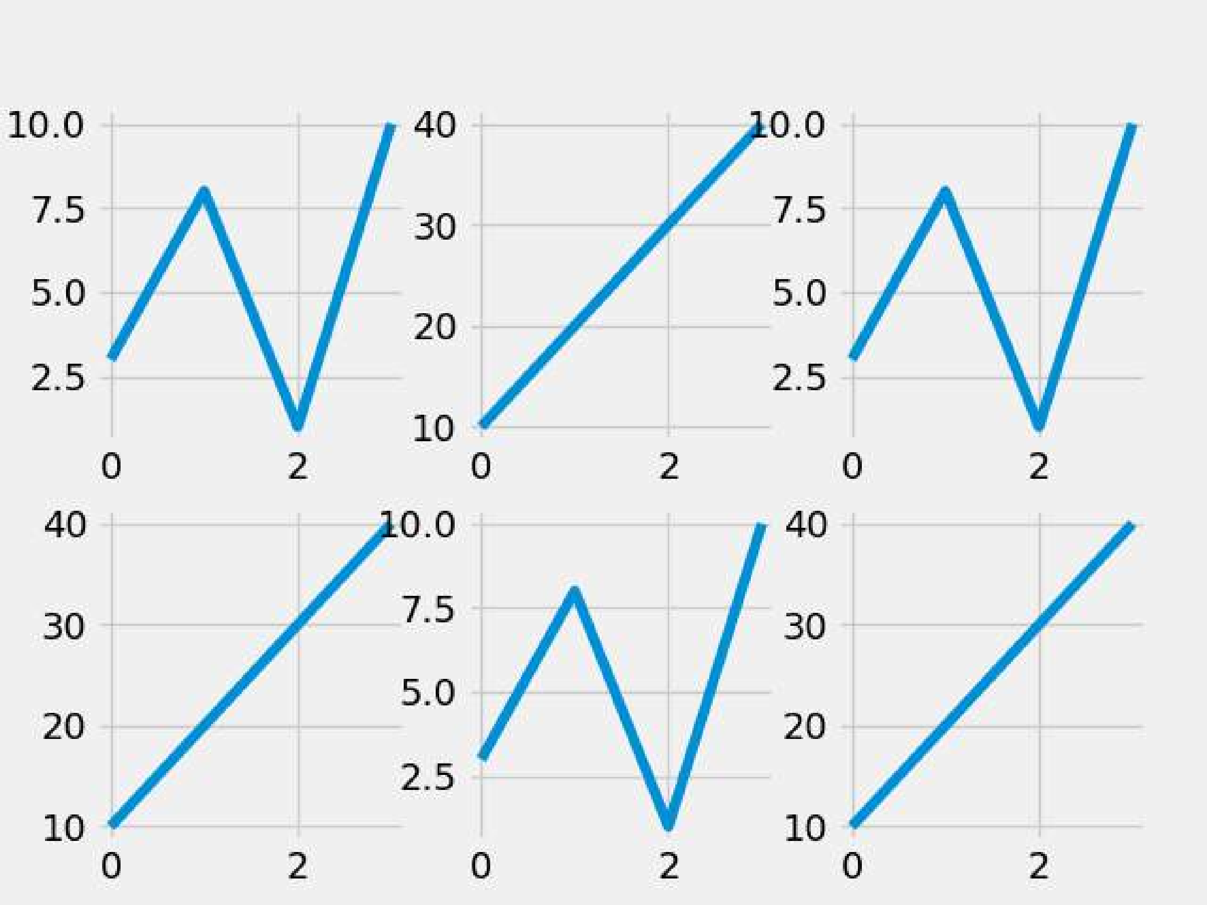
```
import matplotlib.pyplot as plt
import numpy as np

#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)
plt.title("SALES")

#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.title("INCOME")
plt.suptitle("MY SHOP")
```

`plt.subplot(rows, columns, panel number)`



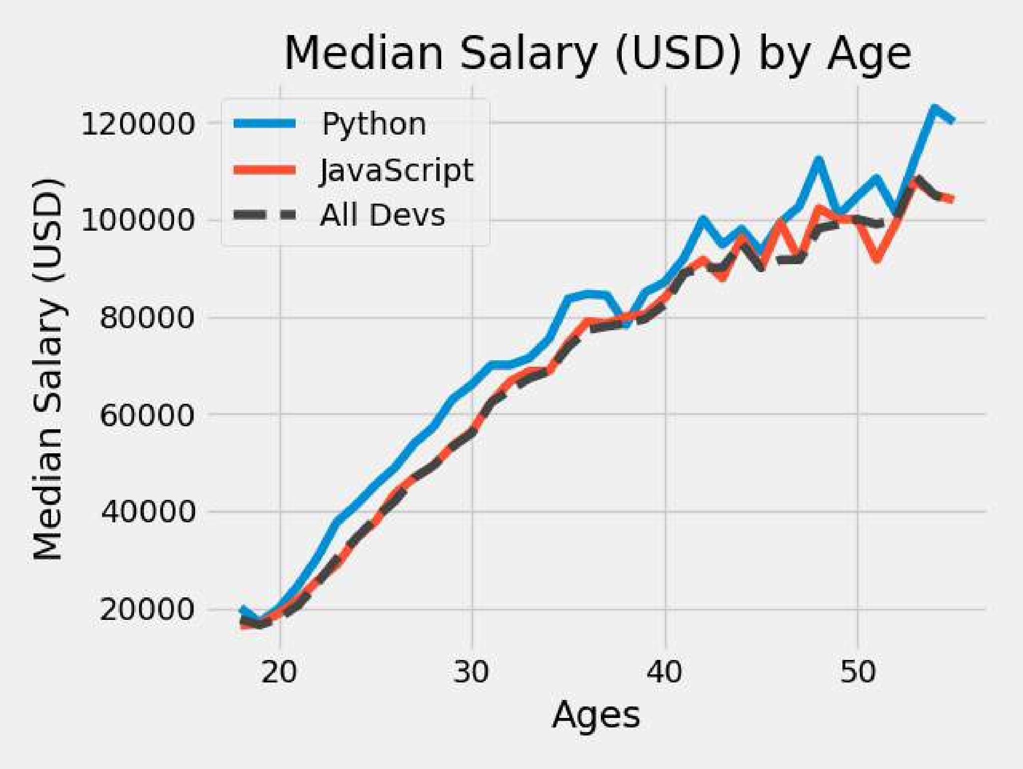
```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

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)
```

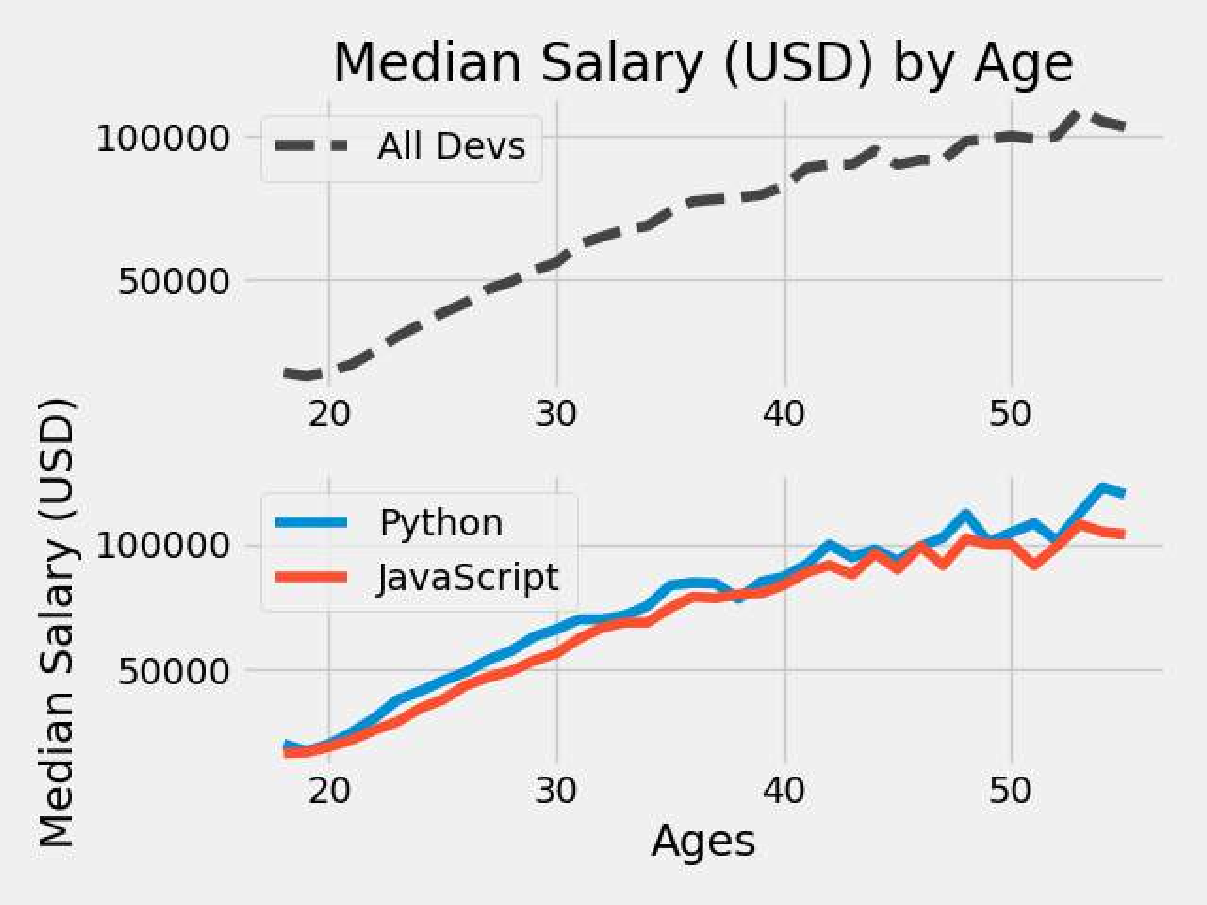


#Another Example:

```
data = pd.read_csv('data.csv')
ages = data['Age']
dev_salaries = data['All_Devs']
py_salaries = data['Python']
js_salaries = data['JavaScript']

plt.plot(ages, py_salaries, label='Python')
plt.plot(ages, js_salaries, label='JavaScript')
plt.plot(ages, dev_salaries, color='#444444',
         linestyle='--', label='All Devs')

plt.title('Median Salary (USD) by Age')
plt.xlabel('Ages')
plt.ylabel('Median Salary (USD)')
plt.legend()
```



```
plt.subplot(2, 1, 1)
plt.plot(ages, dev_salaries, color='#444444',
         linestyle='--', label='All Devs')
plt.title('Median Salary (USD) by Age')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(ages, py_salaries, label='Python')
plt.plot(ages, js_salaries, label='JavaScript')
plt.legend()

plt.xlabel('Ages')
plt.ylabel('Median Salary (USD)')

plt.tight_layout()
```



#Let's explore more advanced topics:

```
fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, sharex=True)

ax1.plot(ages, dev_salaries, color='#444444',
         linestyle='--', label='All Devs')

ax2.plot(ages, py_salaries, label='Python')
ax2.plot(ages, js_salaries, label='JavaScript')

ax1.legend()
ax1.set_title('Median Salary (USD) by Age')
ax1.set_ylabel('Median Salary (USD)')

ax2.legend()
ax2.set_xlabel('Ages')
ax2.set_ylabel('Median Salary (USD)')
```

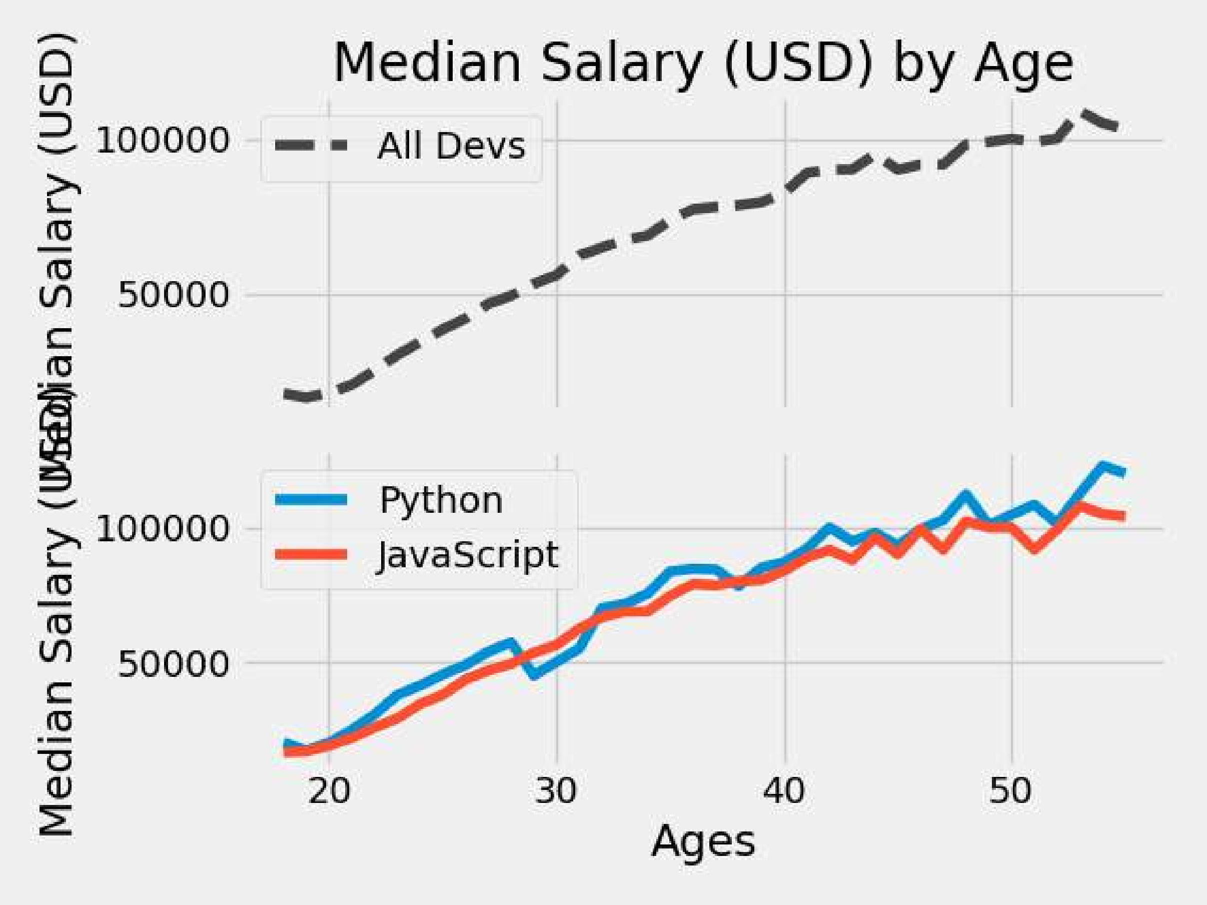
`fig` → the Figure object contains all the subplots.

`(ax1, ax2)`: These are the Axes objects representing the individual subplots.

`nrows=2`: The number of rows.

`ncols=1`: The number of columns in the subplot grid.

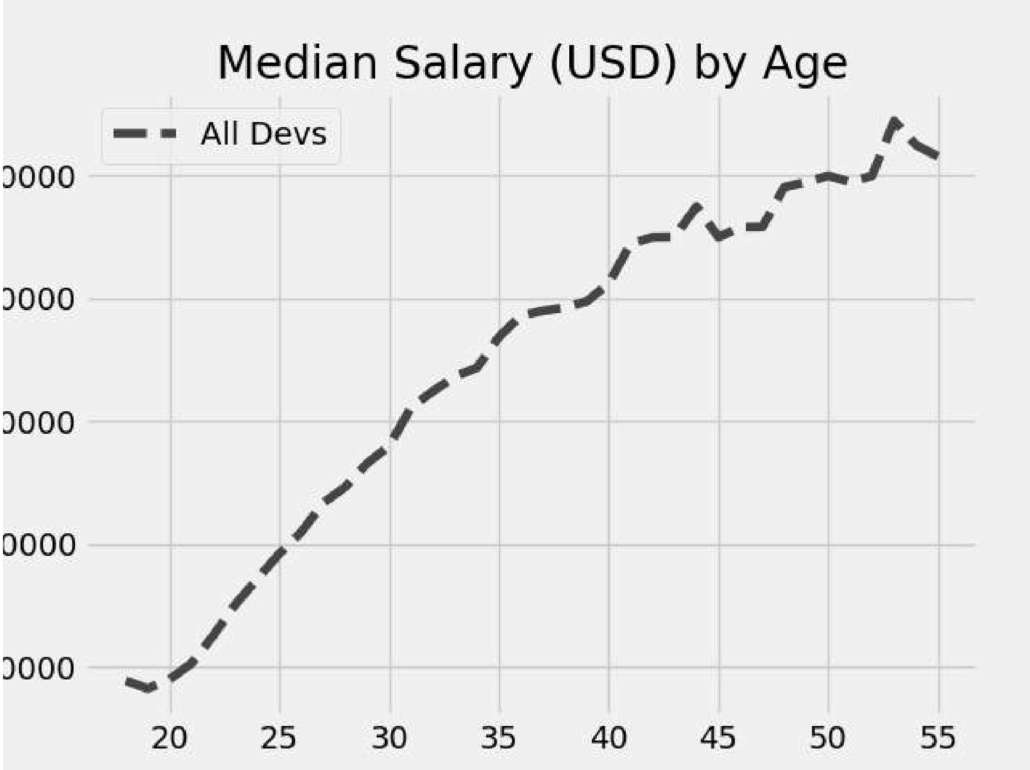
`sharex=True` → Both `ax1` and `ax2` will have the same x-axis limits and ticks to compare data with the same x-axis in multiple subplots.

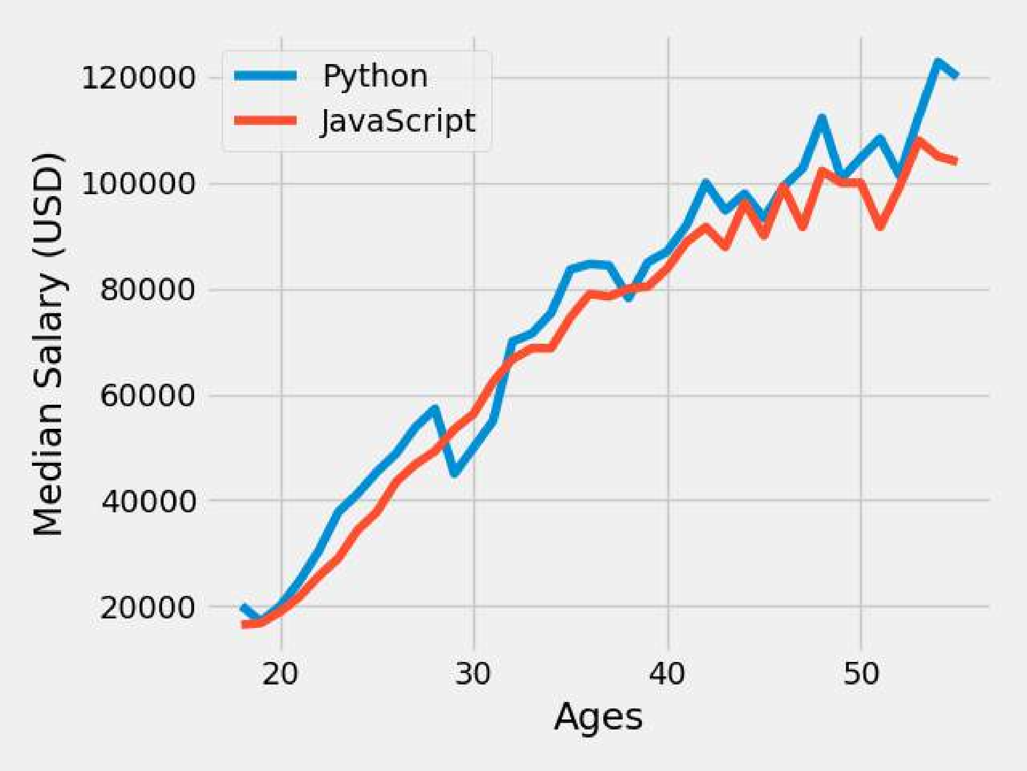


#To have to separate figures and save them:

```
fig1, ax1 = plt.subplots()
fig2, ax2 = plt.subplots()

ax1.plot(ages, dev_salaries, color='#444444',
         linestyle='--', label='All Devs')

ax2.plot(ages, py_salaries, label='Python')
```

```
ax2.plot(ages, js_salaries, label='JavaScript')

ax1.legend()
ax1.set_title('Median Salary (USD) by Age')
ax1.set_ylabel('Median Salary (USD)')

ax2.legend()
ax2.set_xlabel('Ages')
ax2.set_ylabel('Median Salary (USD)')

fig1.savefig('fig1.png')
fig2.savefig('fig2.png')
```

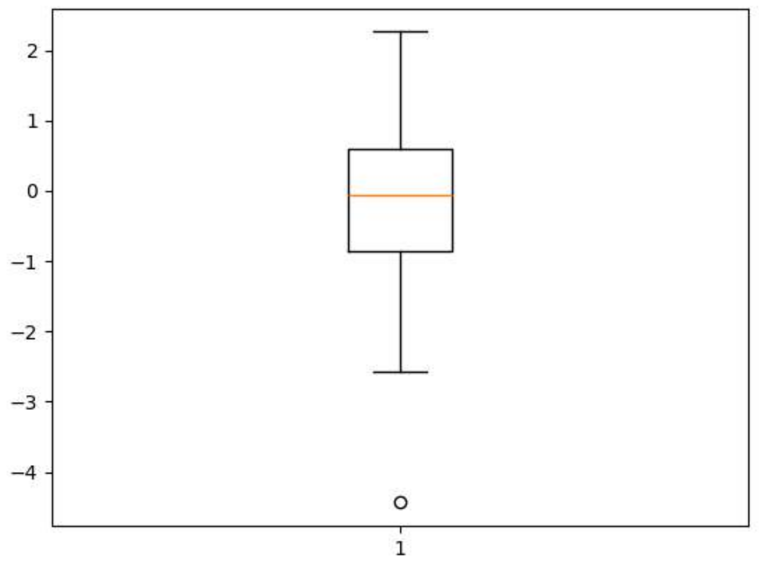




Box plots

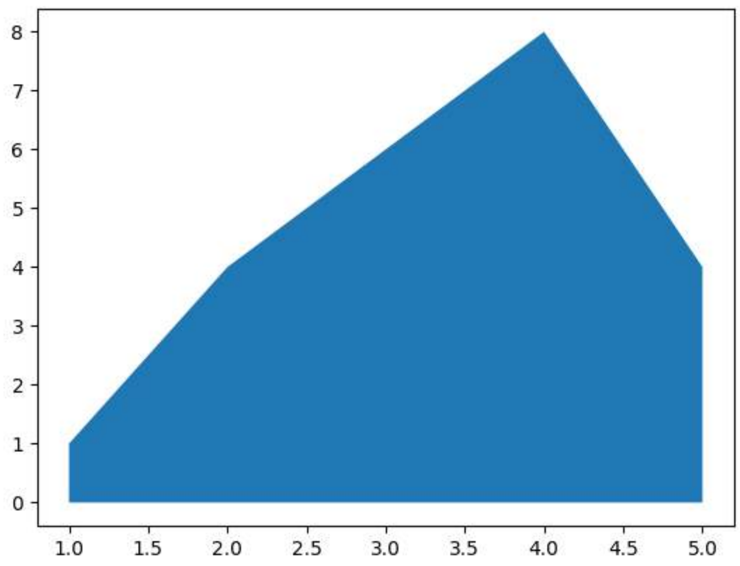
```
data3 = np.random.randn(100)

plt.boxplot(data3)
```

`boxplot()` → takes a set of values and computes the mean, median and other statistical quantities.



Area Chart

```
# Create some data
x12 = range(1, 6)
y12 = [1, 4, 6, 8, 4]

# Area plot
plt.fill_between(x12, y12)
```


8
7
6
5
4
3
2
1
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0