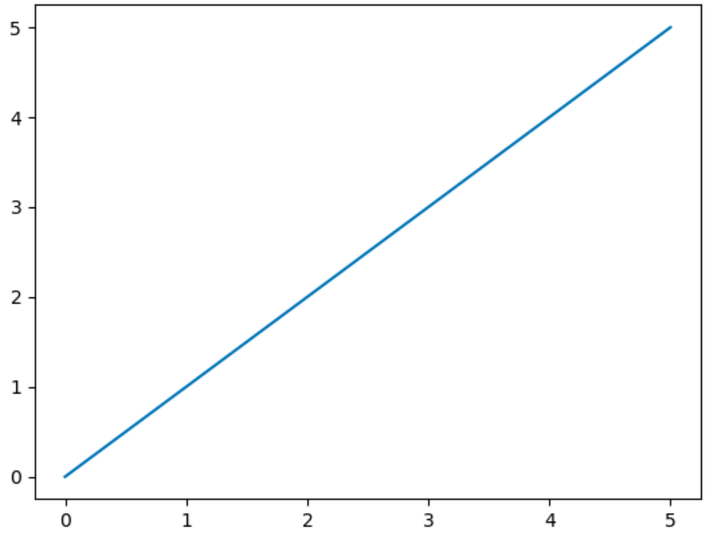
Business Intelligence-relaterade programspråk

Matplotlib

Matplotlib

- Matplotlib is a low level graph plotting library in python that serves as a visualization utility.
- https://github.com/matplotlib/matplotlib



Matplotlib

Installation

- Matplotlib is probably already installed with your version (try first!), but if it's not, install with pip.

```
pip install matplotlib
```

Matplotlib

Pyplot

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

```
import matplotlib.pyplot as plt
```

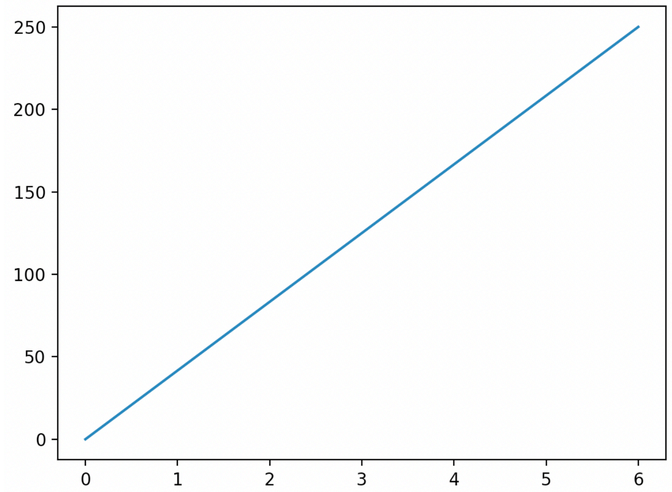
- Now the Pyplot package can be referred to as `plt`.

Matplotlib

Pyplot

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

```
import matplotlib.pyplot as plt

xpoints = [0, 6]
ypoints = [0, 250]

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



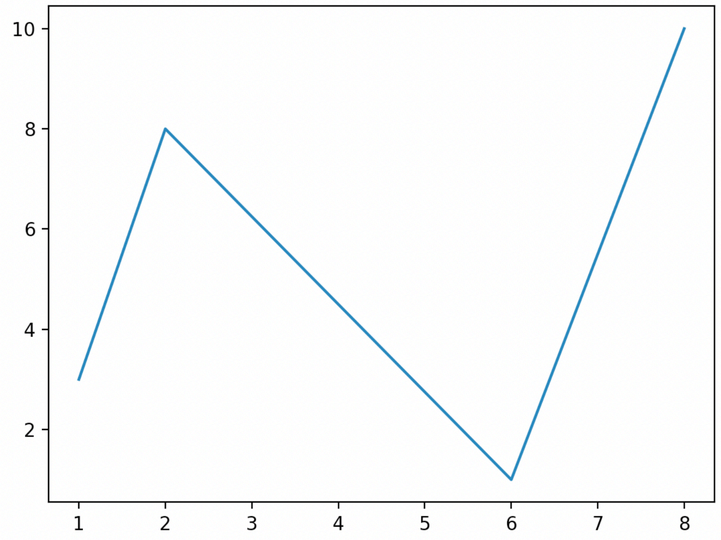
- The `plot()` function is used to draw points (markers) in a diagram.
- By default, the `plot()` function draws a line from point to point.

Matplotlib

Multiple Points

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

```
import matplotlib.pyplot as plt

xpoints = [1, 2, 6, 8]
ypoints = [3, 8, 1, 10]

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

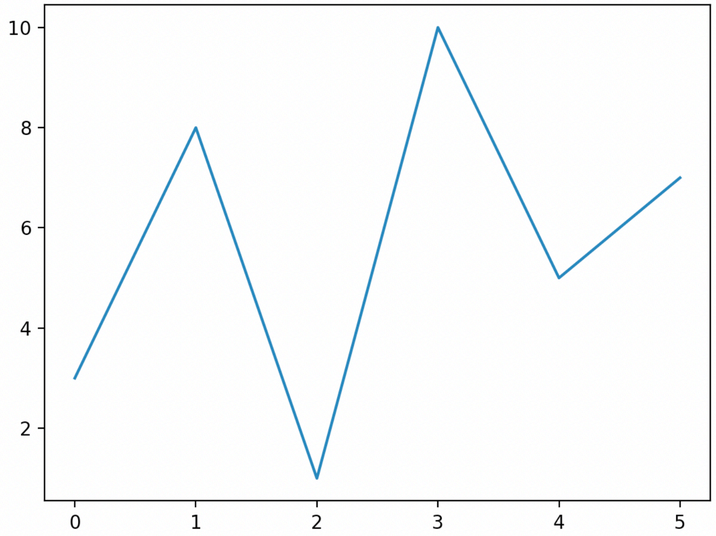


Matplotlib

Default X-Points

- If we do not specify the points in the x-axis, they will get the default values 0, 1, 2, 3, (etc. depending on the length of the y-points.
- So, if we take the same example as above, and leave out the x-points, the diagram will look like this.

```
import matplotlib.pyplot as plt

ypoints = [3, 8, 1, 10, 5, 7]

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

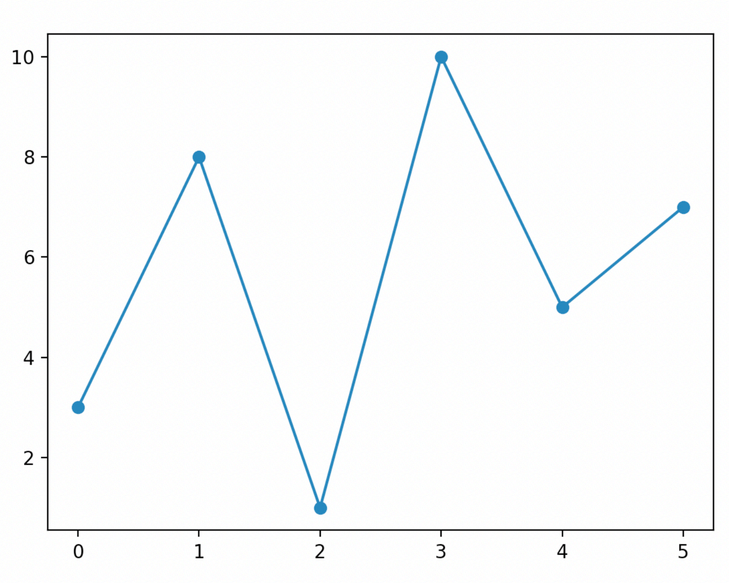


Matplotlib

Markers

- You can use the keyword argument `marker` to emphasize each point with a specified marker.

```
import matplotlib.pyplot as plt

ypoints = [3, 8, 1, 10, 5, 7]

plt.plot(ypoints, marker='o')
plt.show()
```



Matplotlib

Markers

- You can choose any of these markers.

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

Matplotlib

Format Strings `fmt`

- You can use 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
```

- The following two lines accomplishes the same thing.

```
>>> plot(x, y, 'go--', linewidth=2, markersize=12)

>>> plot(x, y, color='green', marker='o', linestyle='dashed',
linewidth=2, markersize=12)
```

Line Syntax	Description
'-'	Solid line
':'	Dotted line
'--'	Dashed line
'-.'	Dashed/dotted line

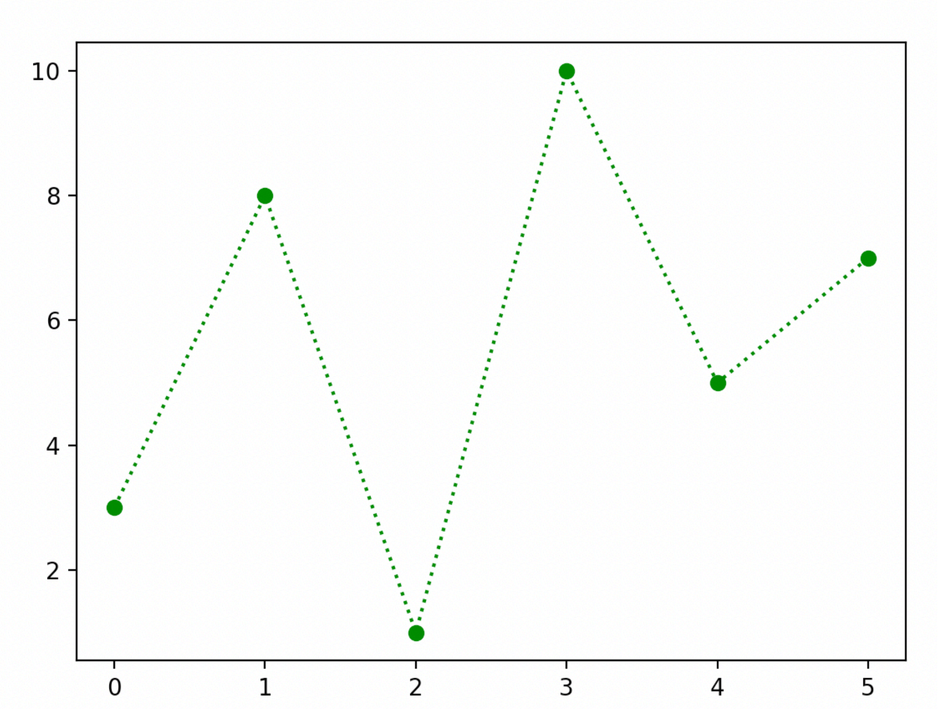
character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

Matplotlib

Format Strings `fmt`

- Mark each point with a circle

```
import matplotlib.pyplot as plt

ypoints = [3, 8, 1, 10, 5, 7]

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

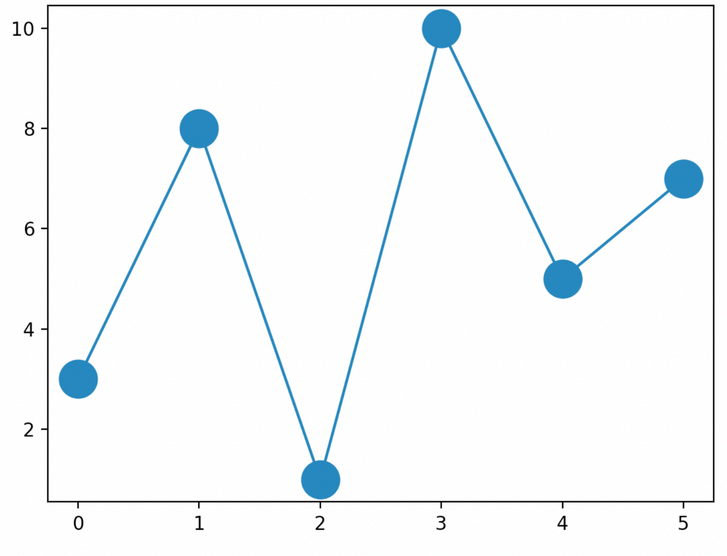


Matplotlib

Marker Size

- You can use the keyword argument `markersize` or the shorter version, `ms` to set the size of the markers.

```
import matplotlib.pyplot as plt

ypoints = [3, 8, 1, 10, 5, 7]

plt.plot(ypoints, marker='o', ms=20)
plt.show()
```

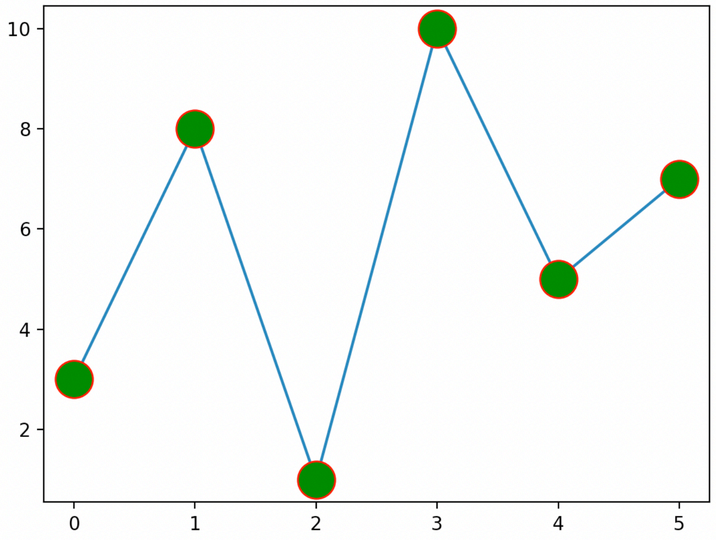


Matplotlib

Marker Color

- You can use the keyword argument `markeredgecolor` or the shorter `mec` to set the color of the edge of the markers.
- You can use the keyword argument `markerfacecolor` or the shorter `mfc` to set the color inside the edge of the markers.

```
import matplotlib.pyplot as plt

ypoints = [3, 8, 1, 10, 5, 7]

plt.plot(ypoints, marker='o', ms=20, mec='r', mfc='g')
plt.show()
```



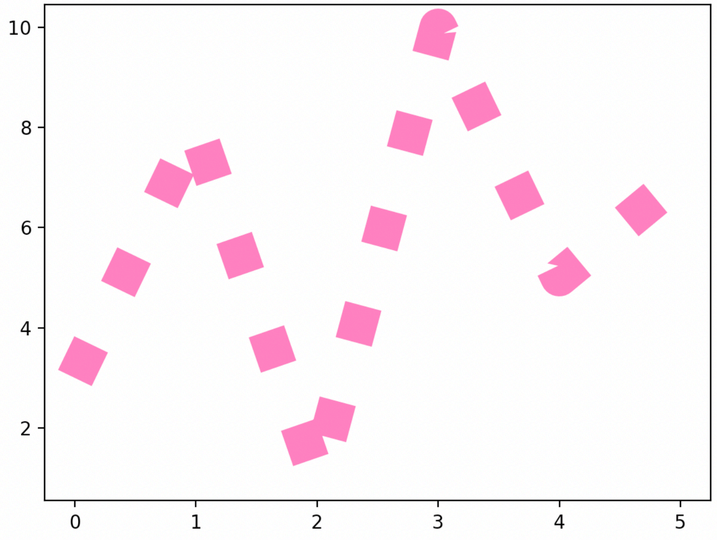
Matplotlib

Line

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

- You can use the keyword argument `linestyle`, or shorter `ls`, to change the style of the plotted line.
- You can use the keyword argument `color` or the shorter `c` to set the color of the line.
- You can use the keyword argument `linewidth` or the shorter `lw` to change the width of the line.

```
import matplotlib.pyplot as plt

ypoints = [3, 8, 1, 10, 5, 7]

plt.plot(ypoints, ls=':', c='hotpink', lw='20')
plt.show()
```

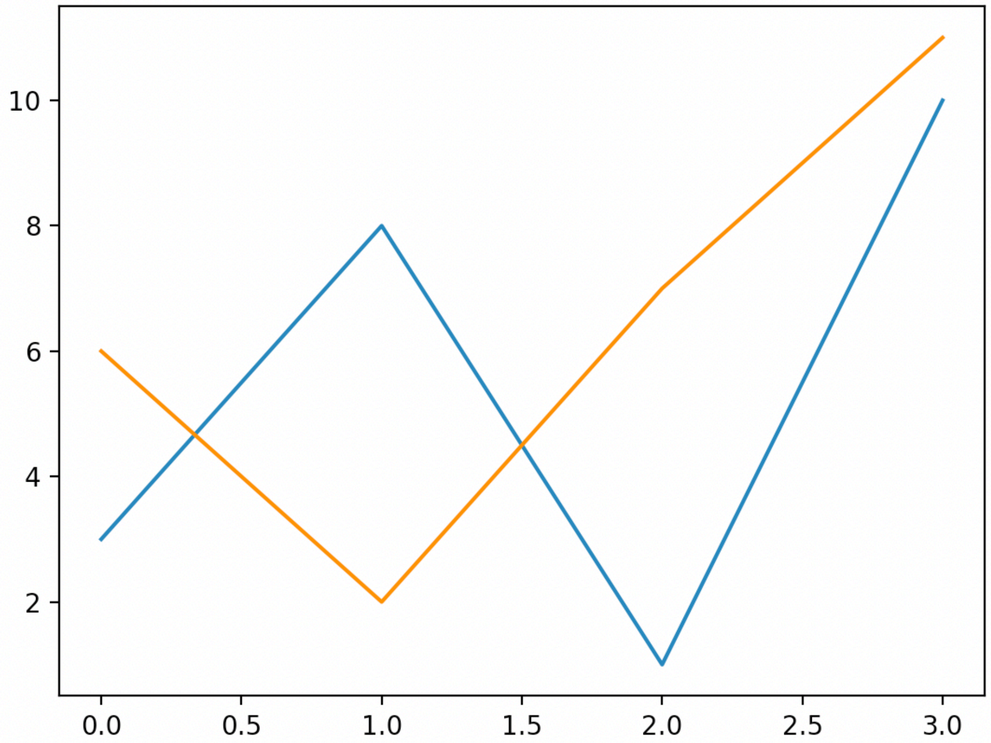


Matplotlib

Multiple Lines

- You can plot as many lines as you like by simply adding more `plt.plot()` functions:

```
import matplotlib.pyplot as plt

y1 = [3, 8, 1, 10]
y2 = [6, 2, 7, 11]

plt.plot(y1)
plt.plot(y2)

plt.show()
```

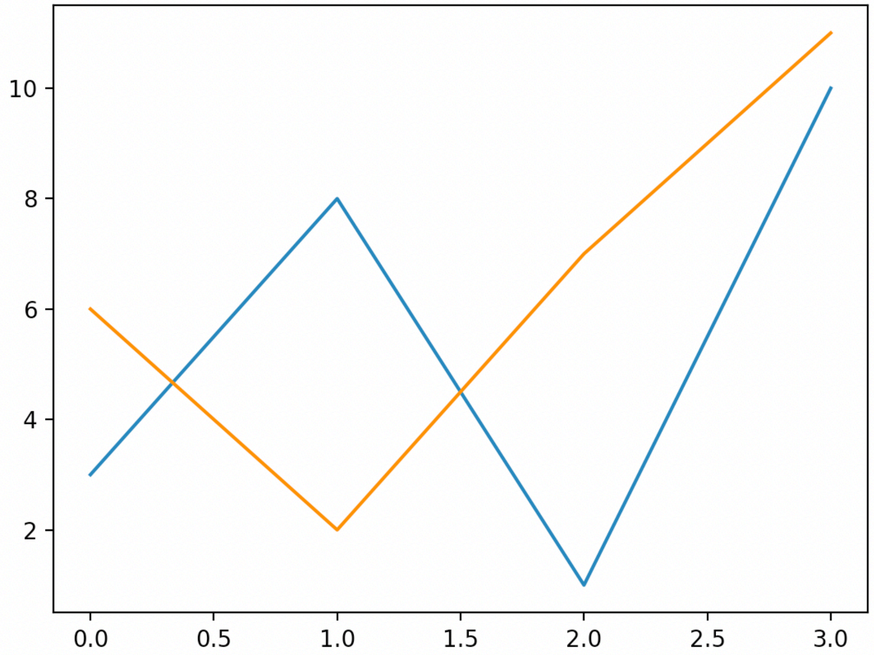


Matplotlib

Multiple Lines

- You can also plot many lines by adding the points for the x- and y-axis for each line in the same `plt.plot()` function.
- The x- and y- values come in pairs.

```
import matplotlib.pyplot as plt

x1 = [0, 1, 2, 3]
y1 = [3, 8, 1, 10]
x2 = [0, 1, 2, 3]
y2 = [6, 2, 7, 11]

plt.plot(x1, y1, x2, y2)

plt.show()
```

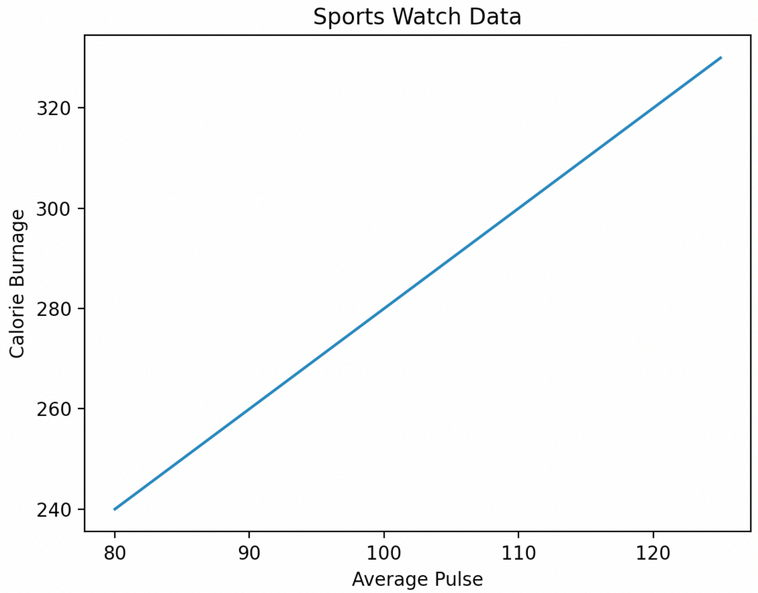


Matplotlib

Labels and Title

- With Pyplot, you can use the `xlabel()` and `ylabel()` functions to set a label for the x- and y-axis.
- You can use the `title()` function to set a title for the plot

```
import matplotlib.pyplot as plt

x = [80, 85, 90, 95, 100, 105, 110, 115, 120, 125]
y = [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()
```

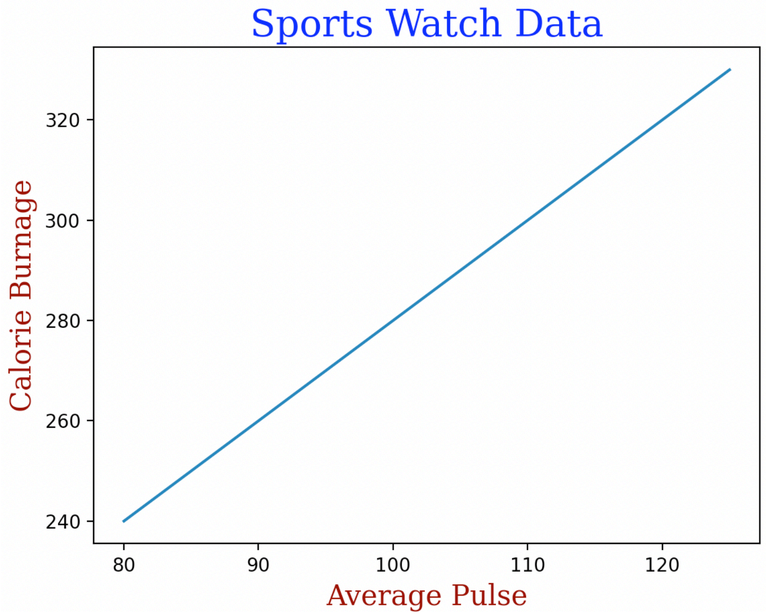


Matplotlib

Set Font Properties for Title and Labels

- You can use the `fontdict` parameter in `xlabel()`, `ylabel()`, and `title()` to set font properties for the title and labels.

```
import matplotlib.pyplot as plt

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

plt.plot(x, y)

font1 = {'family': 'serif', 'color': 'blue', 'size': 20}
font2 = {'family': 'serif', 'color': 'darkred', 'size': 15}

plt.title("Sports Watch Data", fontdict=font1)
plt.xlabel("Average Pulse", fontdict=font2)
plt.ylabel("Calorie Burnage", fontdict=font2)

plt.show()
```

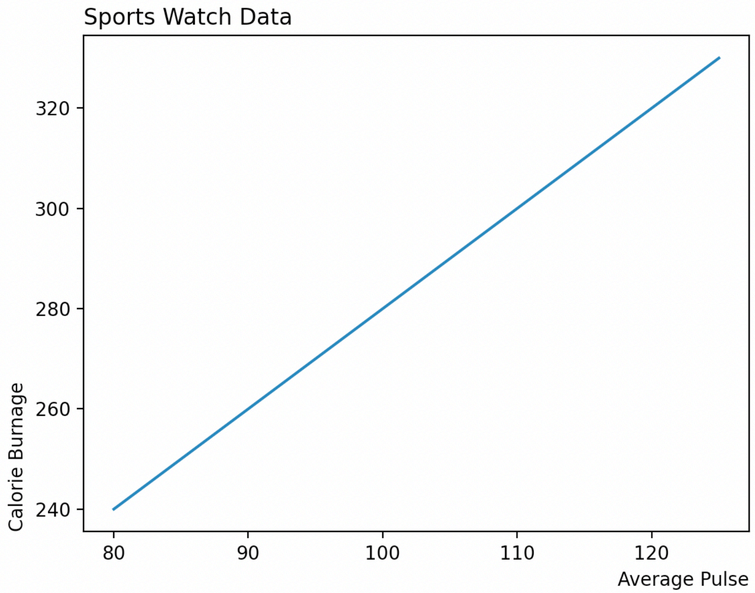


Matplotlib

Position the Title

- You can use the `loc` parameter in `title()` to position the title.
- Legal values are: 'left', 'right', and 'center'. Default value is ’center'.
- For `ylabel()`, the following values are legal: 'bottom', 'center', 'top'

```
import matplotlib.pyplot as plt

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

plt.plot(x, y)

plt.title("Sports Watch Data", loc='left')
plt.xlabel("Average Pulse", loc='right')
plt.ylabel("Calorie Burnage", loc=’bottom')

plt.show()
```

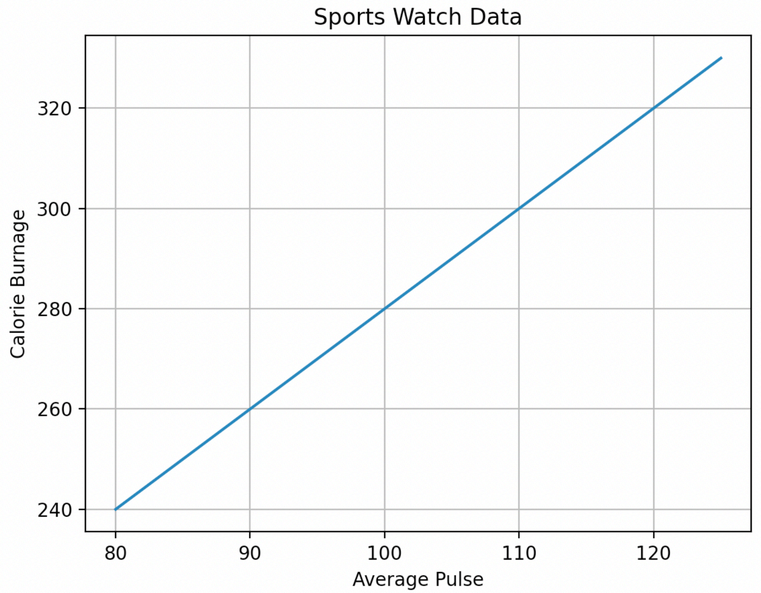


Matplotlib

Add Grid Lines to a Plot

- With Pyplot, you can use the `grid()` function to add grid lines to the plot.

```
import matplotlib.pyplot as plt

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

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid()

plt.show()
```

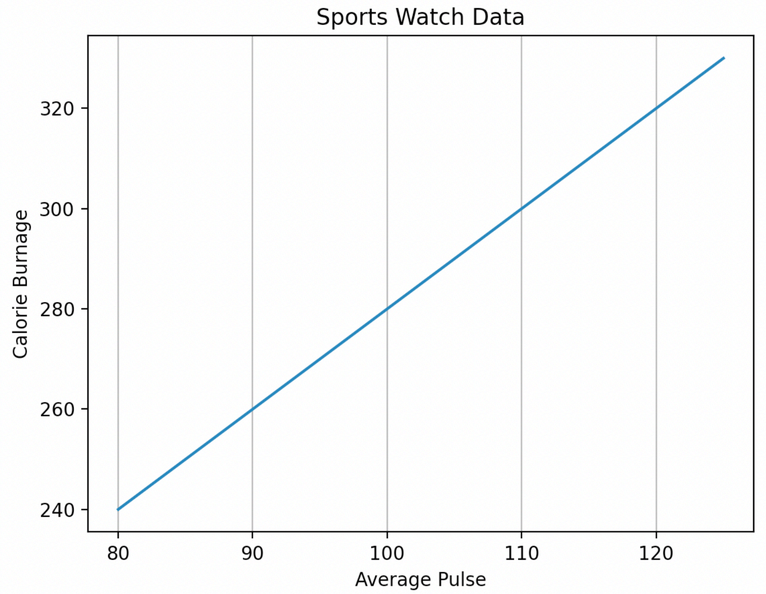


Matplotlib

Specify Which Grid Lines to Display

- You can use the `axis` parameter in the `grid()` function to specify which grid lines to display.
- Legal values are: 'x', 'y', and 'both'. Default value is 'both'.

```
import matplotlib.pyplot as plt

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

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid(axis='x')

plt.show()
```

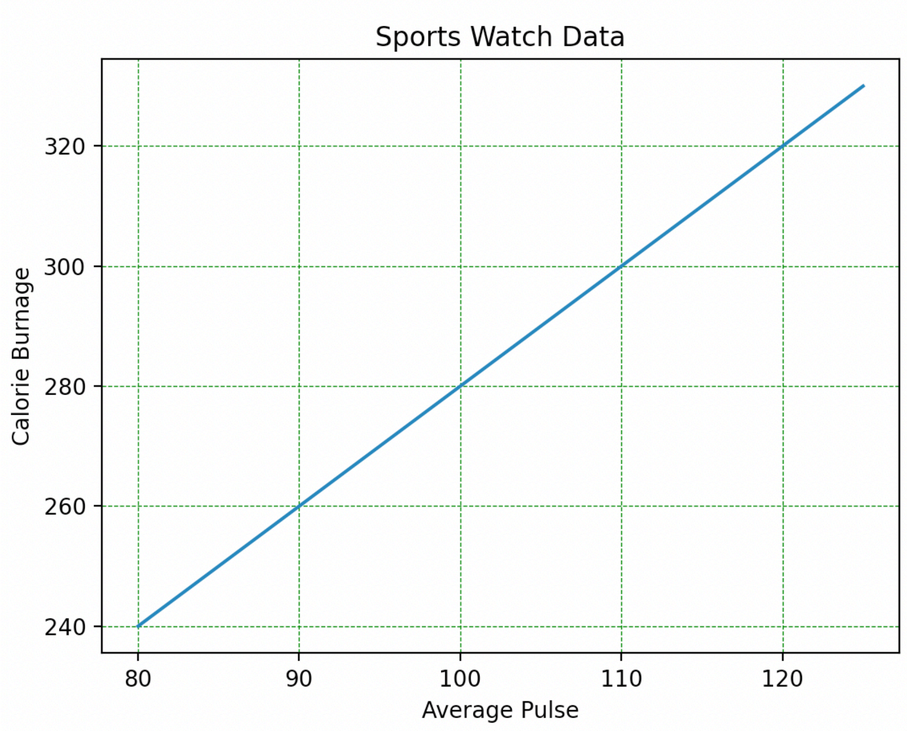


Matplotlib

Set Line Properties for the Grid

- You can also set the line properties of the grid, like this: grid(color = 'color', linestyle = 'linestyle', linewidth = number).

```
import matplotlib.pyplot as plt

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

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.plot(x, y)

plt.grid(color='green', linestyle='--', linewidth=0.5)

plt.show()
```

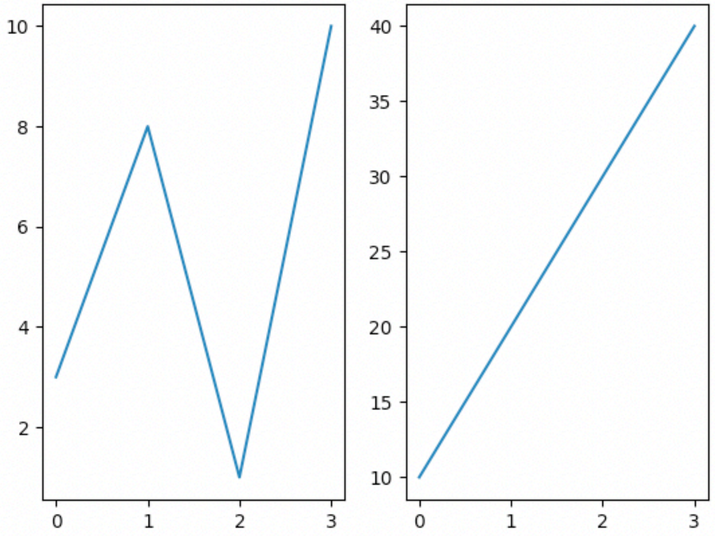


Matplotlib

Display Multiple Plots

- With the `subplots()` function you can draw multiple plots in one figure.

```
import matplotlib.pyplot as plt

# plot 1:
x = [0, 1, 2, 3]
y = [3, 8, 1, 10]

plt.subplot(1, 2, 1)
plt.plot(x, y)

# plot 2:
x = [0, 1, 2, 3]
y = [10, 20, 30, 40]

plt.subplot(1, 2, 2)
plt.plot(x, y)

plt.show()
```

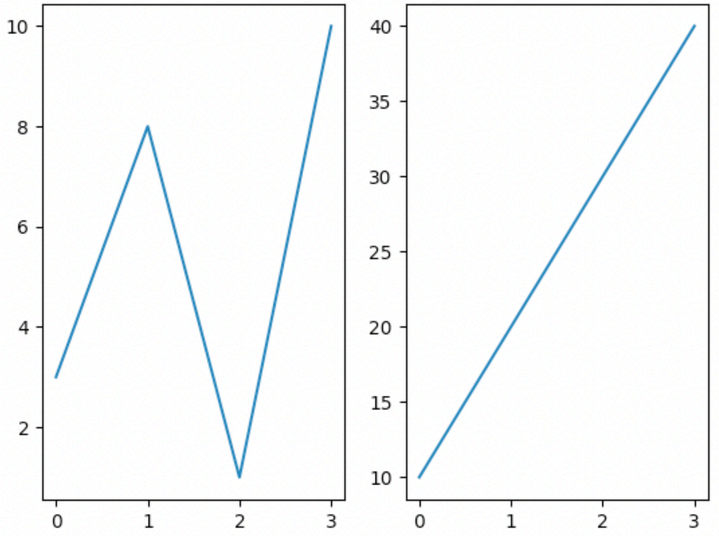


Matplotlib

The `subplots()` Function

- The `subplots()` function takes three arguments that describes the layout of the figure.
- The layout is organized in rows and columns, which are represented by the first and second argument.
- The third argument represents the index of the current plot.

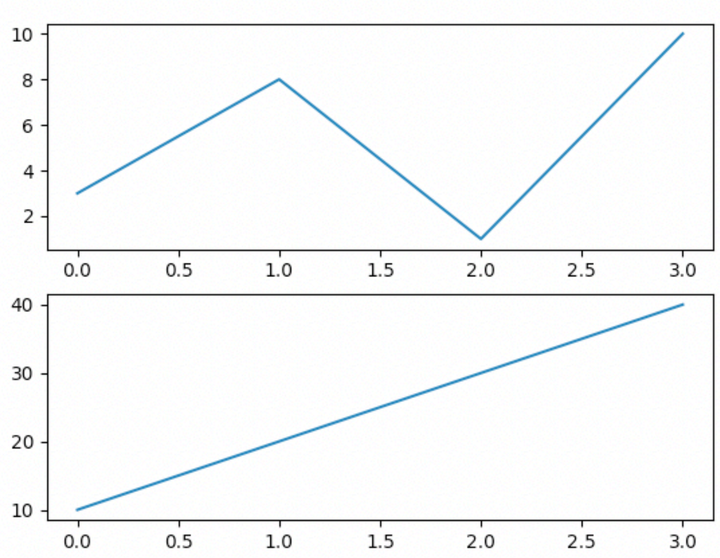
```
#plt.subplot(1, 2, 1)
#the figure has 1 row, 2 columns, and this plot is
the first plot.

plt.subplot(1, 2, 2)
#the figure has 1 row, 2 columns, and this plot is
the second plot.
```



Matplotlib

The `subplots()` Function

- So, if we want a figure with 2 rows an 1 column (meaning that the two plots will be displayed on top of each other instead of side-by-side), we can write the syntax like this.

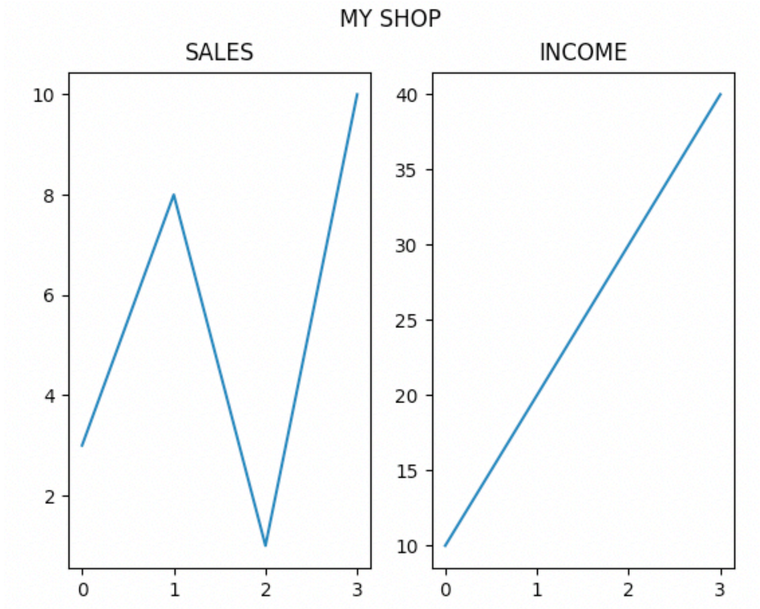



```
import matplotlib.pyplot as plt

# plot 1:
x = [0, 1, 2, 3]
y = [3, 8, 1, 10]

plt.subplot(2, 1, 1)
plt.plot(x, y)

# plot 2:
x = [0, 1, 2, 3]
y = [10, 20, 30, 40]

plt.subplot(2, 1, 2)
plt.plot(x, y)

plt.show()
```

Matplotlib

Super Title

- You can add a title to the entire figure with the `suptitle()` function.

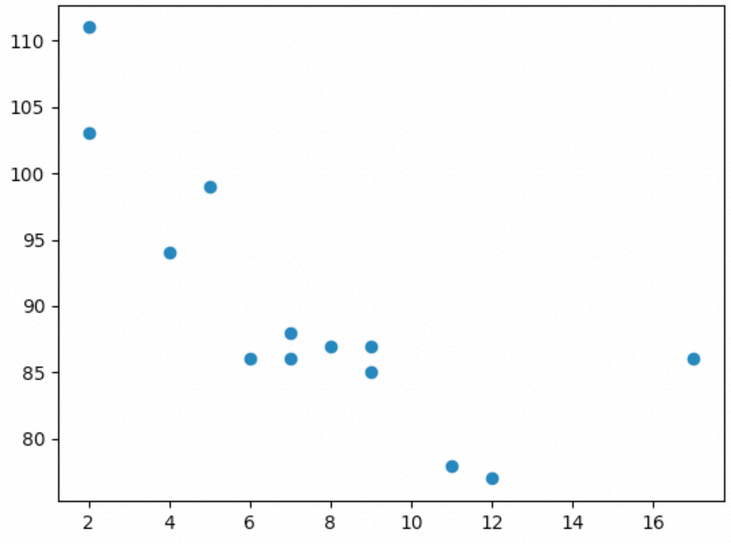



```
import matplotlib.pyplot as plt

# plot 1:
x = [0, 1, 2, 3]
y = [3, 8, 1, 10]

plt.subplot(1, 2, 1)
plt.plot(x, y)
plt.title("SALES")

# plot 2:
x = [0, 1, 2, 3]
y = [10, 20, 30, 40]

plt.subplot(1, 2, 2)
plt.plot(x, y)
plt.title("INCOME")

plt.suptitle("MY SHOP")
plt.show()
```

Matplotlib

Creating Scatter Plots

- 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:

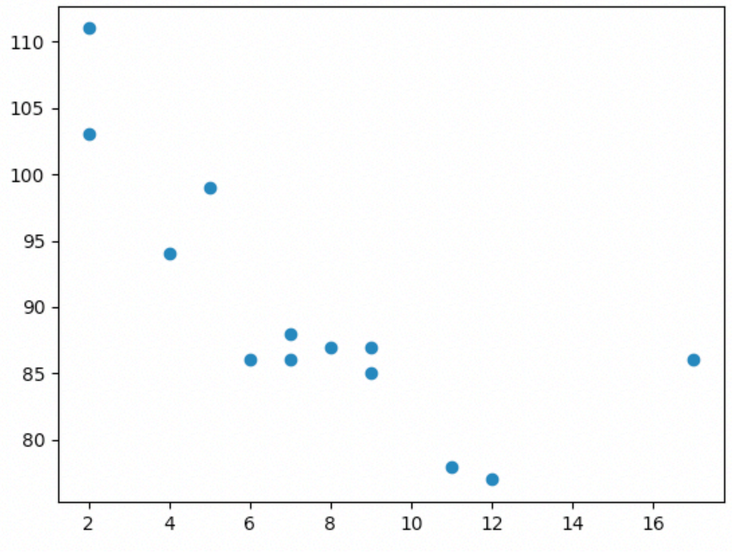



```
import matplotlib.pyplot as plt


x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9,
6]
y = [99, 86, 87, 88, 111, 86, 103, 87, 94,
78, 77, 85, 86]

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

Matplotlib

Creating Scatter Plots

- The observation in the example is the result of 13 cars passing by.
- The X-axis shows how old the car is.

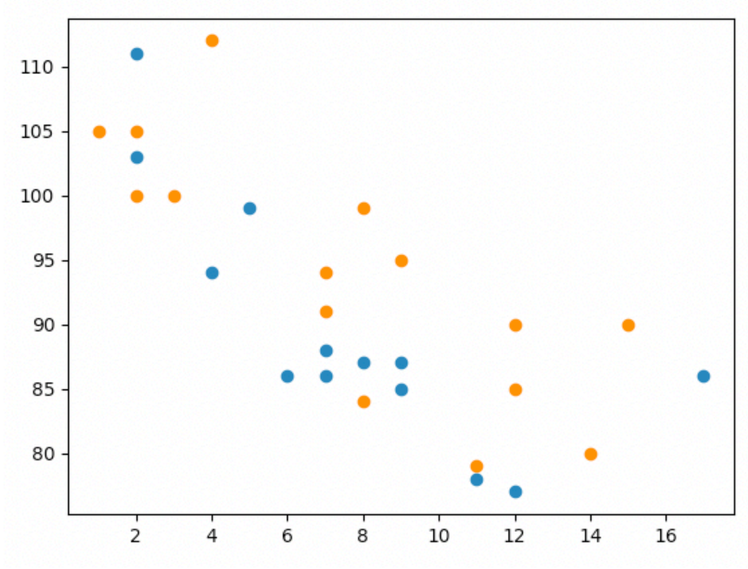


- The Y-axis shows the speed of the car when it passes.
- Are there any relationships between the observations?
- It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

Matplotlib

Compare Plots

- In the example, there seems to be a relationship between speed and age, but what if we plot the observations from another day as well? Will the scatter plot tell us something else?

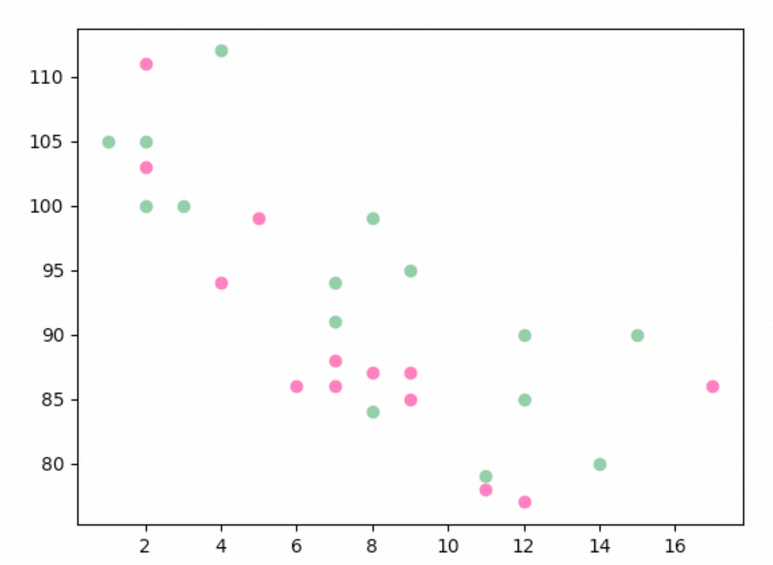



```
import matplotlib.pyplot as plt

# day one, the age and speed of 13 cars:
x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6]
y = [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 = [2, 2, 8, 1, 15, 8, 12, 9, 7, 3, 11, 4, 7,
14, 12]
y = [100, 105, 84, 105, 90, 99, 90, 95, 94, 100,
79, 112, 91, 80, 85]
plt.scatter(x, y)

plt.show()
```

Matplotlib

Colors

- You can set your own color for each scatter plot with the `color` or the `c` argument.

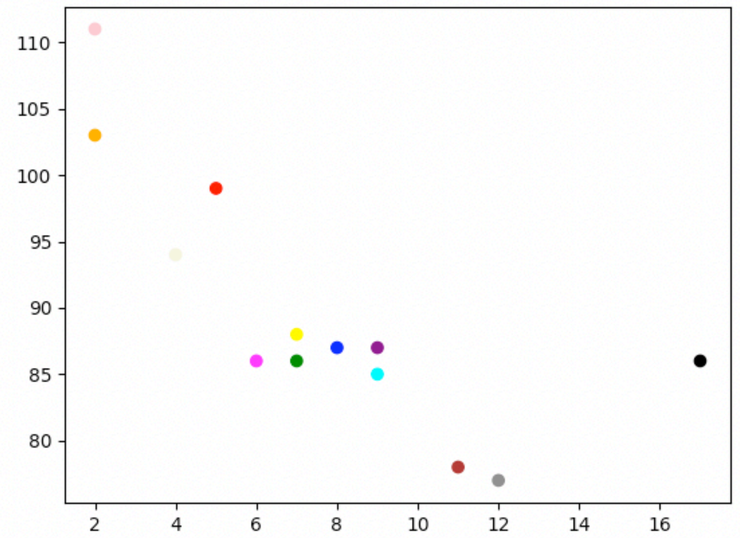



```
import matplotlib.pyplot as plt

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

# day two, the age and speed of 15 cars:
x = [2, 2, 8, 1, 15, 8, 12, 9, 7, 3, 11, 4, 7,
14, 12]
y = [100, 105, 84, 105, 90, 99, 90, 95, 94, 100,
79, 112, 91, 80, 85]
plt.scatter(x, y, color='#88c999')

plt.show()
```

Matplotlib

Color Each Dot

- You can even set a specific color for each dot by using an array of colors as value for the `c` argument:
- Note: You cannot use the `color` argument for this, only the `c` argument.




```
import matplotlib.pyplot as plt

x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9,
6]
y = [99, 86, 87, 88, 111, 86, 103, 87, 94,
78, 77, 85, 86]
colors = ["red", "green", "blue", "yellow",
"pink", ”black", "orange", "purple",
"beige", "brown", "gray", "cyan", "magenta"]

plt.scatter(x, y, c=colors)

plt.show()
```

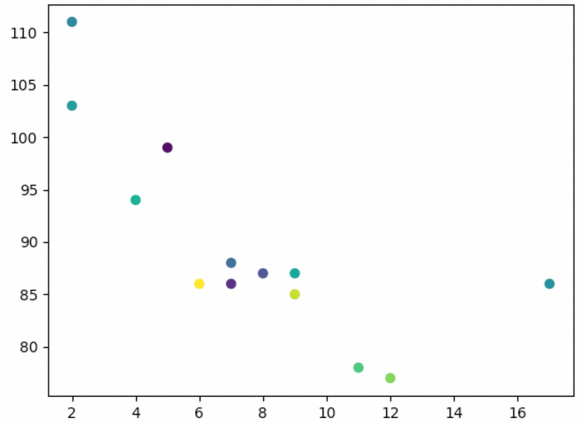
Matplotlib

ColorMap

- The Matplotlib module has a number of available colormaps.
- A colormap is like a list of colors, where each color has a value that ranges from 0 to 100.
- This colormap is called 'viridis' and as you can see it ranges from 0, which is a purple color, and up to 100, which is a yellow color.

Matplotlib

How to Use the ColorMap

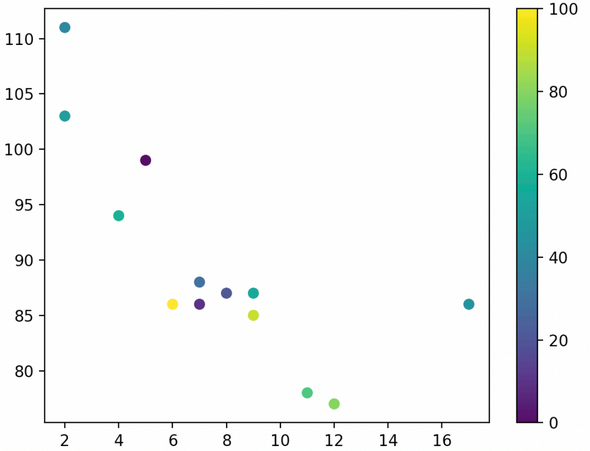


- You can specify the colormap with the keyword argument cmap with the value of the colormap, in this case 'viridis' which is one of the built-in colormaps available in Matplotlib.
- In addition you have to create an array with values (from 0 to 100), one value for each of the point in the scatter plot:

```
import matplotlib.pyplot as plt

x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11,
12, 9, 6]
y = [99, 86, 87, 88, 111, 86, 103, 87,
94, 78, 77, 85, 86]
colors = [0, 10, 20, 30, 40, 45, 50,
55, 60, 70, 80, 90, 100]

plt.scatter(x, y, c=colors,
cmap='viridis')

plt.show()
```

Matplotlib

How to Use the ColorMap



- You can include the colormap in the drawing by including the `plt.colorbar()` statement.
- https://matplotlib.org/stable/tutorials/colors/colormaps.html

```
import matplotlib.pyplot as plt

x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9,
6]
y = [99, 86, 87, 88, 111, 86, 103, 87, 94,
78, 77, 85, 86]
colors = [0, 10, 20, 30, 40, 45, 50, 55, 60,
70, 80, 90, 100]
```

```
plt.scatter(x, y, c=colors, cmap='viridis')

x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9,
6]
y = [99, 86, 87, 88, 111, 86, 103, 87, 94,
78, 77, 85, 86]
colors = [0, 10, 20, 30, 40, 45, 50, 55, 60,
70, 80, 90, 100]

plt.scatter(x, y, c=colors, cmap='viridis')

plt.colorbar()

plt.show()
```

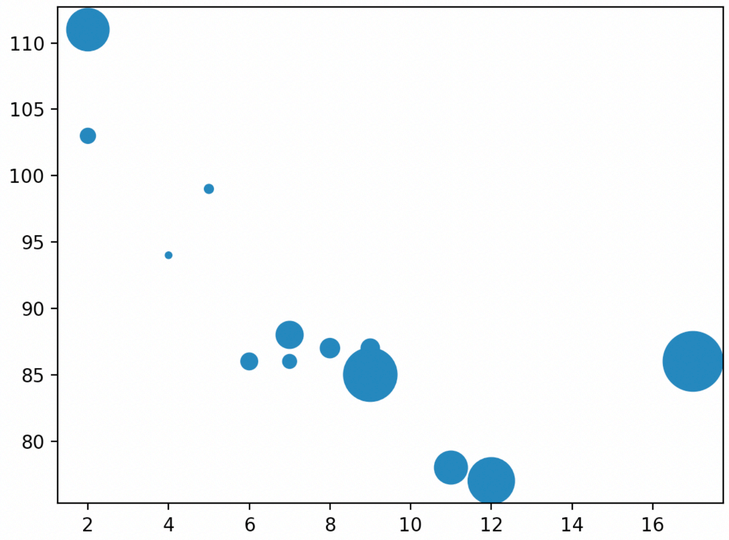
Matplotlib

Size

- You can change the size of the dots with the s argument.
- Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis

```
import matplotlib.pyplot as plt

x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11,
12, 9, 6]
y = [99, 86, 87, 88, 111, 86, 103, 87,
94, 78, 77, 85, 86]
```

```
sizes = [20, 50, 100, 200, 500, 1000,
60, 90, 10, 300, 600, 800, 75]

plt.scatter(x, y, s=sizes)

plt.show()
```



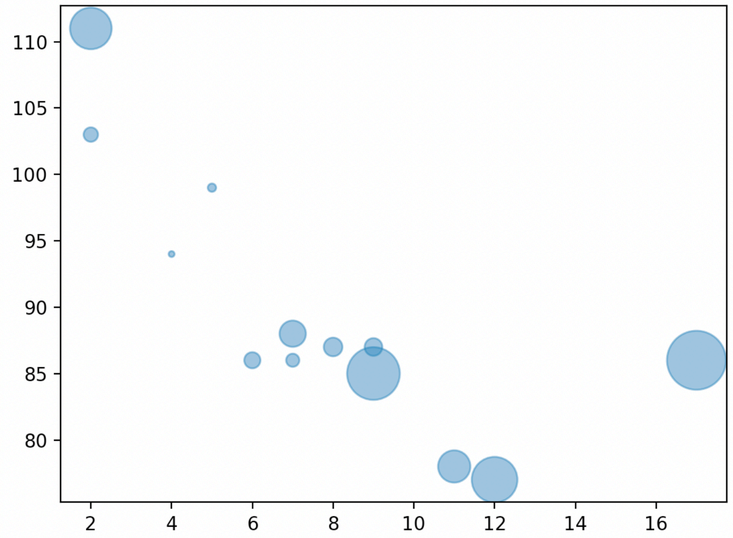
Matplotlib

Alpha

- You can adjust the transparency of the dots with the alpha argument.
- Just like colors, make sure the array for sizes has the same length as the arrays for the x- and y-axis.

```
import matplotlib.pyplot as plt

x = [5, 7, 8, 7, 2, 17, 2, 9, 4, 11,
12, 9, 6]
y = [99, 86, 87, 88, 111, 86, 103, 87,
94, 78, 77, 85, 86]
```

```
sizes = [20, 50, 100, 200, 500, 1000,
60, 90, 10, 300, 600, 800, 75]

plt.scatter(x, y, s=sizes, alpha=0.5)

plt.show()
```



Exercise

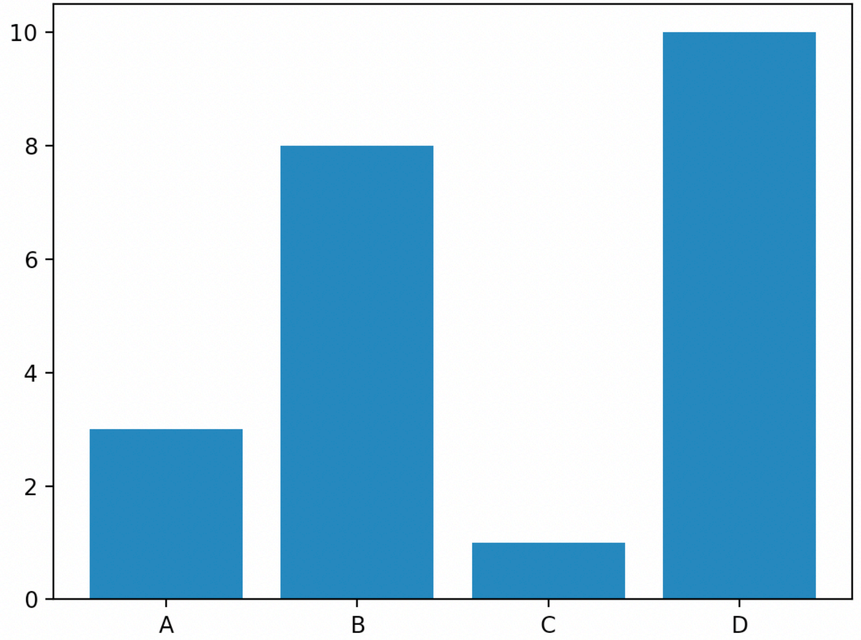
- Make a scatter graph with data from classic models products.
- On the x-axxis, use the buy price.
- On the y-axxis, use the MSRP.

Matplotlib

Creating Bars

- You can use the `bar()` function to draw bar graphs.

```
import matplotlib.pyplot as plt

x = ["A", "B", "C", "D"]
y = [3, 8, 1, 10]

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

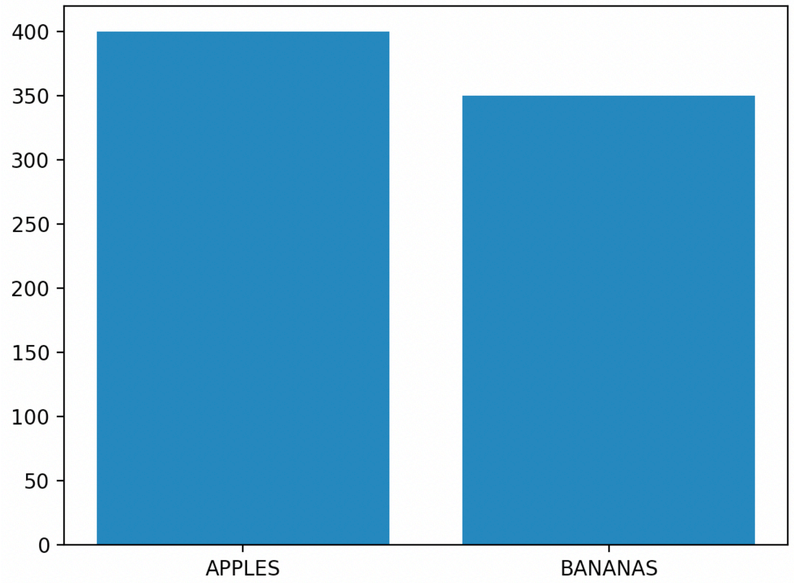


Matplotlib

Creating Bars

- The `bar()` function takes arguments that describes the layout of the bars.
- The categories and their values represented by the first and second argument as arrays.

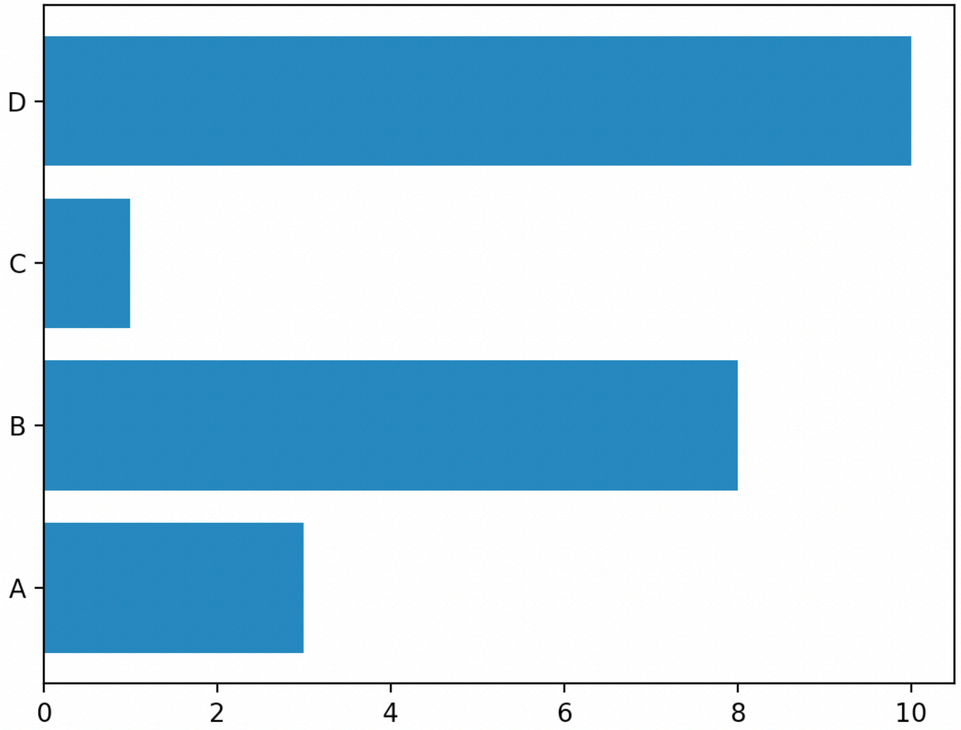
```
import matplotlib.pyplot as plt

x = ["APPLES", "BANANAS"]
y = [400, 350]
plt.bar(x, y)
plt.show()
```



Matplotlib

Horizontal Bars

- If you want the bars to be displayed horizontally instead of vertically, use the `barh()` function.

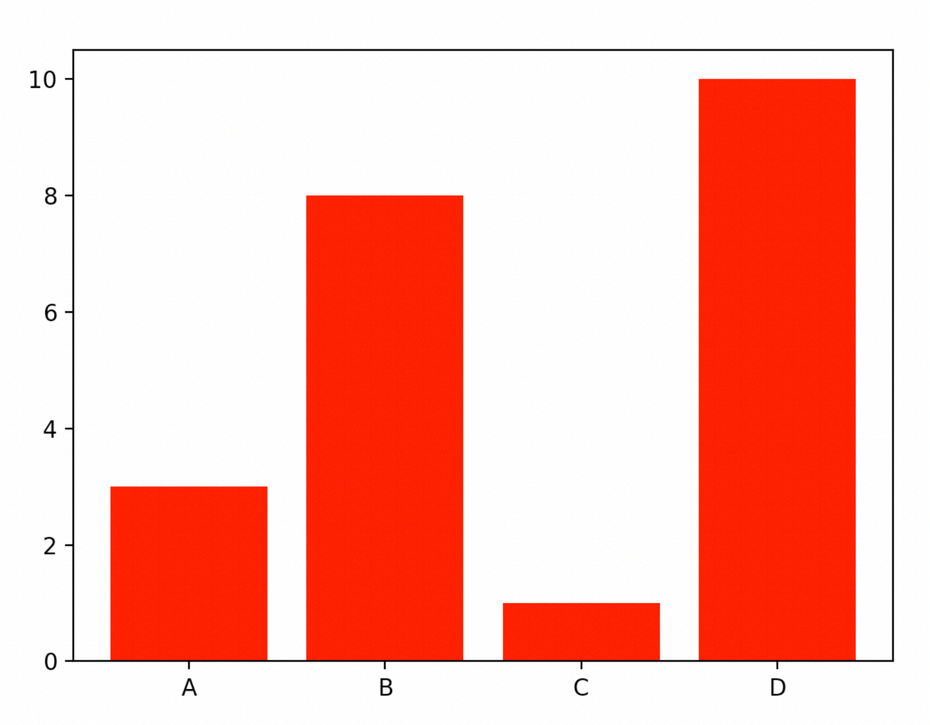



```
import matplotlib.pyplot as plt

x = ["A", "B", "C", "D"]
y = [3, 8, 1, 10]

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

Matplotlib

Bar Color

- The `bar()` and `barh()` takes the keyword argument `color` to set the color of the bars.



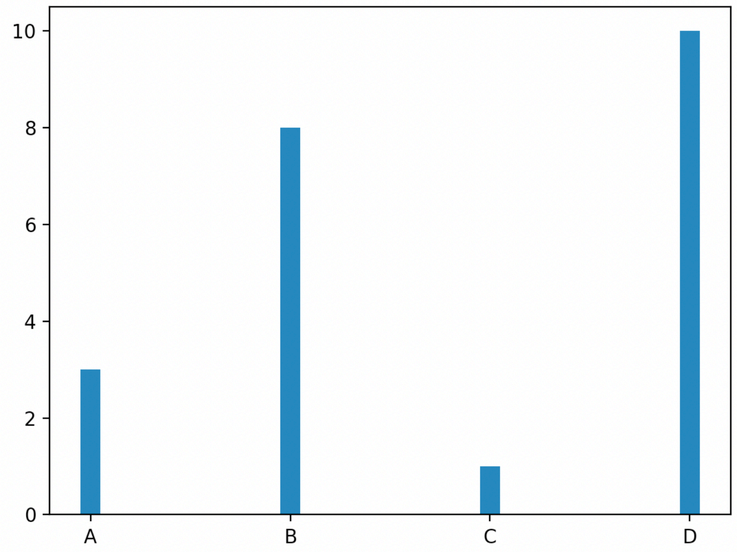

```
import matplotlib.pyplot as plt

x = ["A", "B", "C", "D"]
y = [3, 8, 1, 10]

plt.bar(x, y, color = "red")
plt.show()
```

Matplotlib

Bar Width

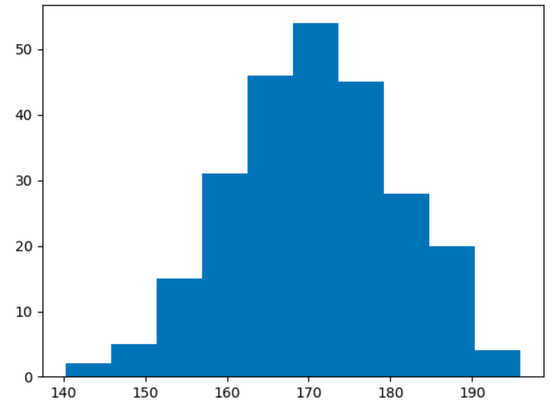
- The `bar()` takes the keyword argument width to set the width of the bars.
- The default width value is 0.8
- Note: For horizontal bars, use height instead of width.

```
import matplotlib.pyplot as plt

x = ["A", "B", "C", "D"]
y = [3, 8, 1, 10]

plt.bar(x, y, width = 0.1)
plt.show()
```



Matplotlib

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.
- You can read from the histogram that there are approximately:

- 2 people from 140 to 145cm
  5 people from 145 to 150cm
  15 people from 151 to 156cm
  31 people from 157 to 162cm
  46 people from 163 to 168cm
  53 people from 168 to 173cm
  45 people from 173 to 178cm
  28 people from 179 to 184cm
  21 people from 185 to 190cm
  4 people from 190 to 195cm

Matplotlib

Create Histogram

```
[167.62255766 175.32495609 152.84661337 165.50264047 163.17457988
 162.29867872 172.83638413 168.67303667 164.57361342 180.81120541
 170.57782187 167.53075749 176.15356275 176.95378312 158.4125473
 187.8842668  159.03730075 166.69284332 160.73882029 152.22378865
 164.01255164 163.95288674 176.58146832 173.19849526 169.40206527
 166.88861903 149.90348576 148.39039643 177.90349066 166.72462233
 177.44776004 170.93335636 173.26312881 174.76534435 162.28791953
 166.77301551 160.53785202 170.67972019 159.11594186 165.36992993
 178.38979253 171.52158489 173.32636678 159.63894401 151.95735707
```

- In Matplotlib, we use the `hist()` function to create histograms.
- The `hist()` function will use an array of numbers to create a histogram, the array is sent into the function as an argument.
- For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10.

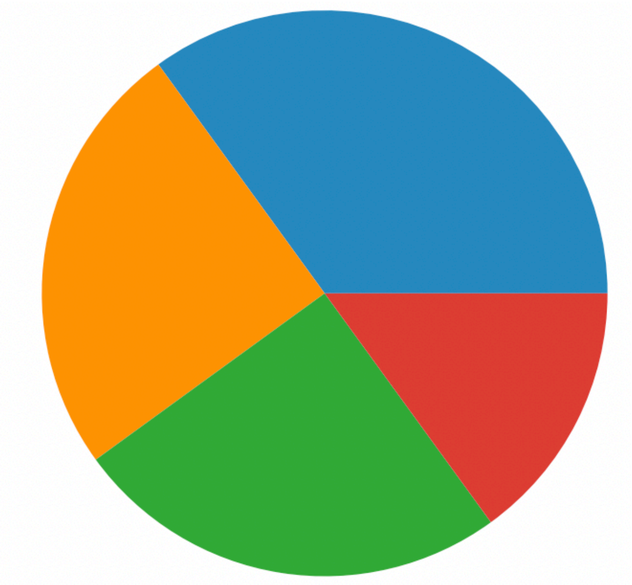
```
import numpy as np

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

print(x)
```

- Learn more about NumPy here:
  - https://www.w3schools.com/python/numpy/default.asp

Matplotlib

Creating Pie Charts

- You can use the `pie()` function to draw pie charts.

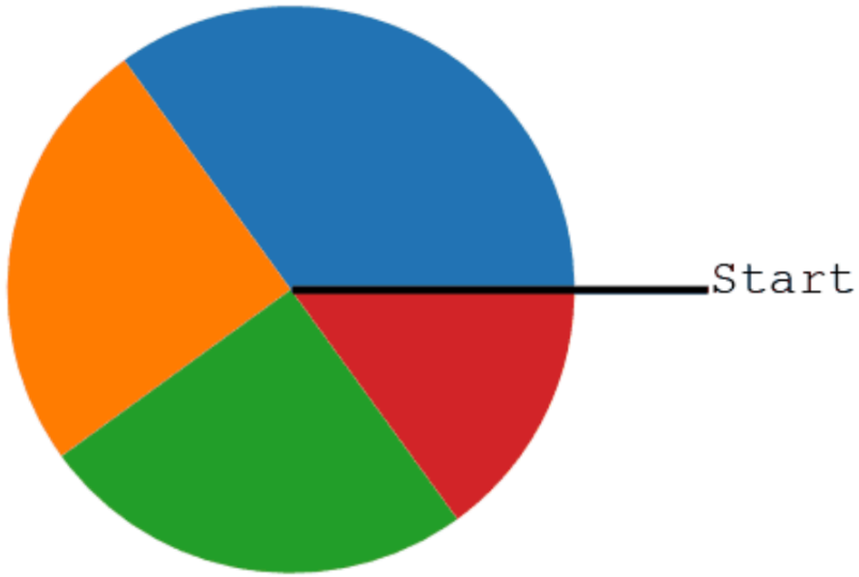


```
import matplotlib.pyplot as plt

y = [35, 25, 25, 15]

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

Matplotlib

Creating Pie Charts

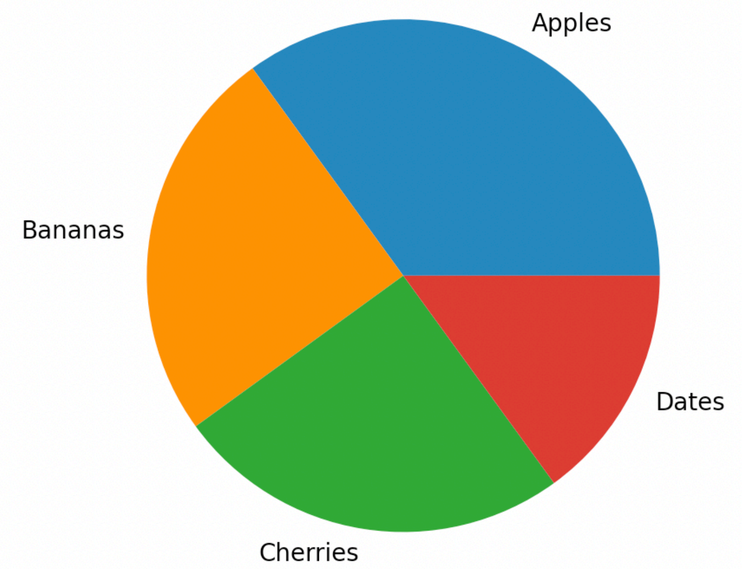
- As you can see the pie chart draws one piece (called a wedge) for each value in the array (in this case `[35, 25, 25, 15]`).
- By default the plotting of the first wedge starts from the x-axis and move counterclockwise.
- *Note:* The size of each wedge is determined by comparing the value with all the other values.



Matplotlib

Labels

- Add labels to the pie chart with the `label` parameter.
- The `label` parameter must be an array with one label for each w

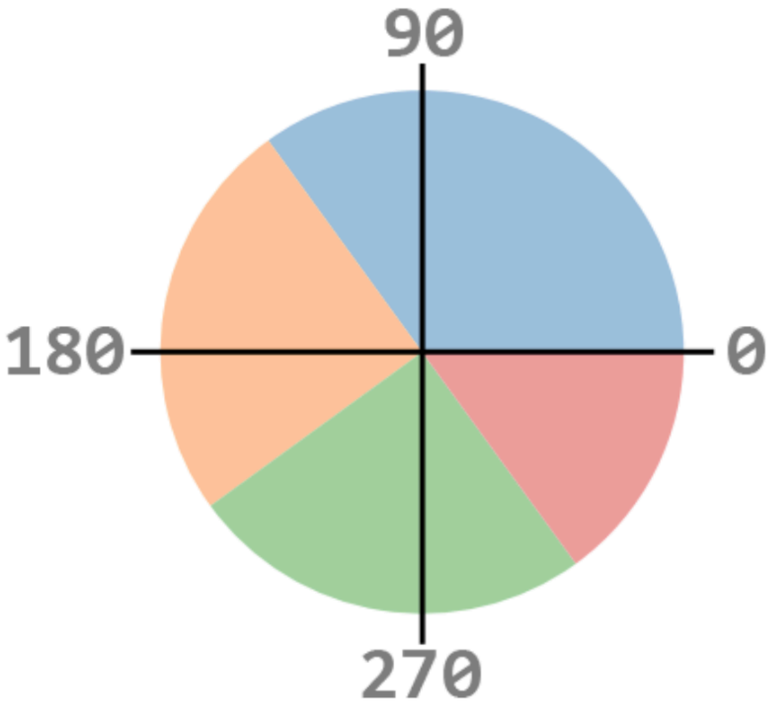



```
import matplotlib.pyplot as plt

y = [35, 25, 25, 15]

mylabels = ["Apples", "Bananas",
"Cherries", "Dates"]

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

Matplotlib

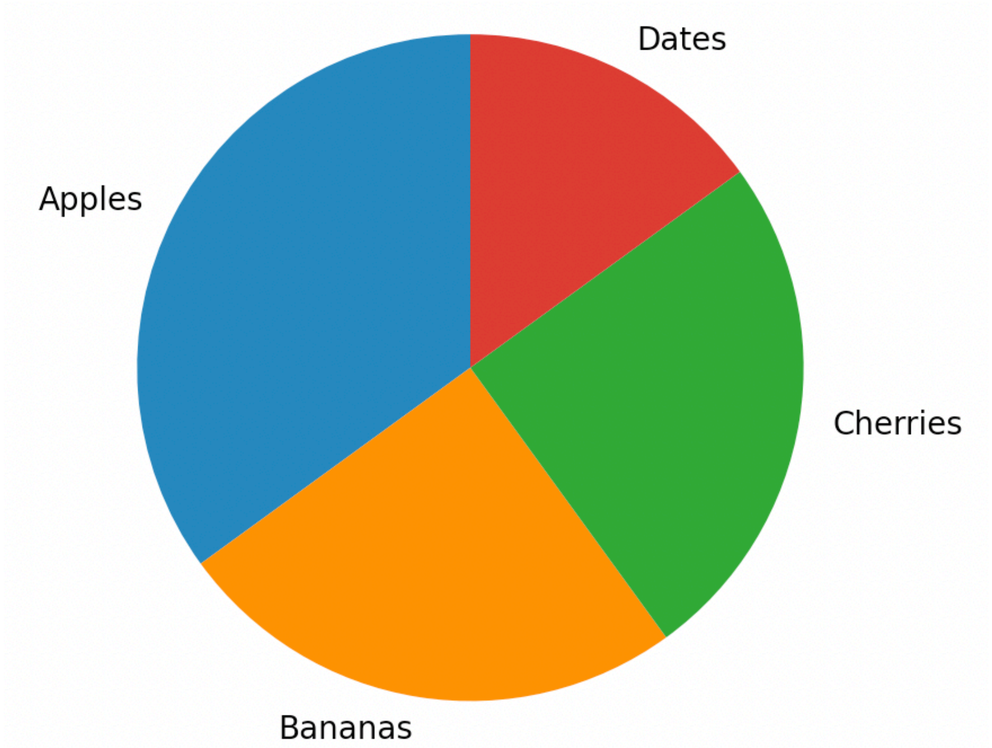
Start Angle

- As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a `startangle` parameter.
- The `startangle` parameter is defined with an angle in degrees, default angle is 0.



Matplotlib

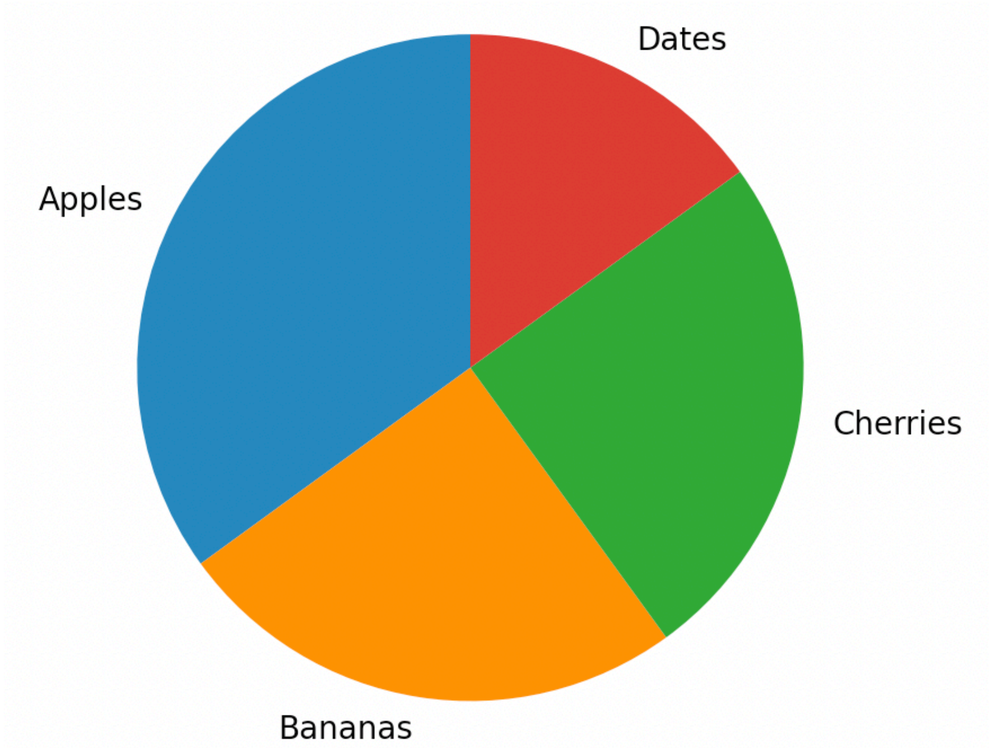
Start Angle




```
import matplotlib.pyplot as plt

y = [35, 25, 25, 15]

mylabels = ["Apples", "Bananas",
"Cherries", "Dates"]

plt.pie(y, labels=mylabels,
startangle=90)
plt.show()
```

Matplotlib

Start Angle




```
import matplotlib.pyplot as plt

y = [35, 25, 25, 15]

mylabels = ["Apples", "Bananas",
"Cherries", "Dates"]

plt.pie(y, labels=mylabels,
startangle=90)
plt.show()
```