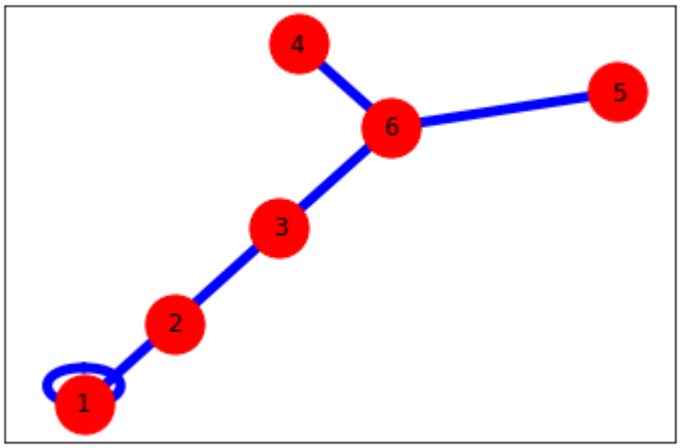
#Visualizing Graphs with NetworkX

Graphs in data structures are non-linear data structures made up of a finite number of nodes or vertices and the edges that connect them. Graphs in data structures are used to address real-world problems in which it represents the problem area as a network like telephone networks, circuit networks, and social networks.

##Undirected Graphs

An undirected graph comprises a set of nodes and links connecting them. The order of the two connected vertices is irrelevant and has no direction.

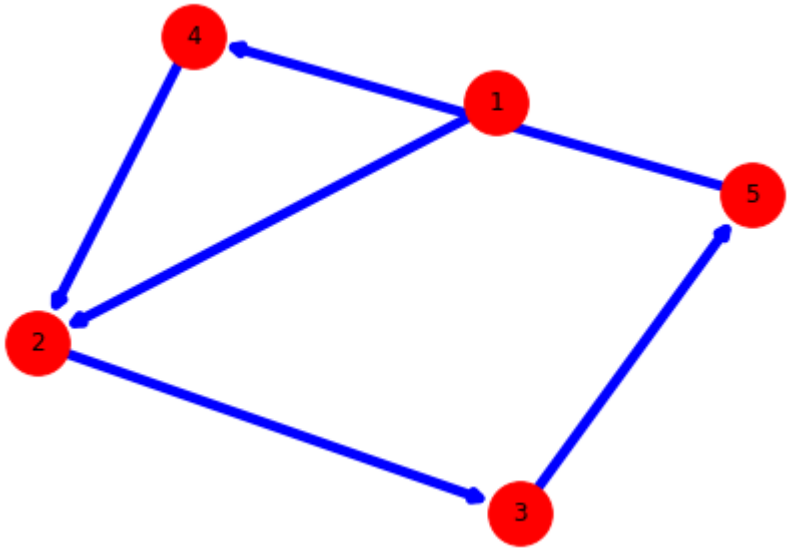
In [ ]:
```
import networkx as nx
import matplotlib.pyplot as plt
G = nx.Graph()
H = nx.Graph()
G.add_node(1)
G.add_nodes_from([2, 3])
G.add_nodes_from(range(4, 7))
G.add_edge(1, 2)
G.add_edge(1, 1)
G.add_edges_from([(2,3), (3,6), (4,6), (5,6)])
nx.draw_networkx(G, node_size=850, node_color='red', width=5, edge_color='bl
plt.show()
```



##Directed Graphs

A directed graph is a data structure that stores data in vertices or nodes and these nodes or vertices are connected and also directed by edges (one vertex is directed towards another vertex through an edge). Directed graph is also called a digraph or directed network.

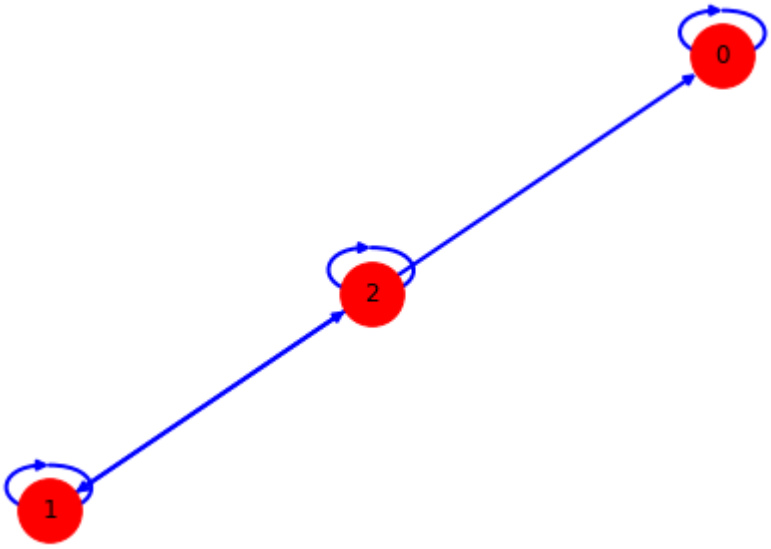
In [ ]:
```
import networkx as nx
import matplotlib.pyplot as plt

g = nx.DiGraph()
g.add_nodes_from([1,2,3,4,5])
g.add_edge(1,2)
g.add_edge(4,2)
g.add_edge(3,5)
g.add_edge(2,3)
g.add_edge(5,4)

nx.draw(g,with_labels=True, node_size=1000, node_color='red', width=5, edge_
plt.draw()
plt.show()
```



##Creating a Directed graph from an Adjacency Matrix

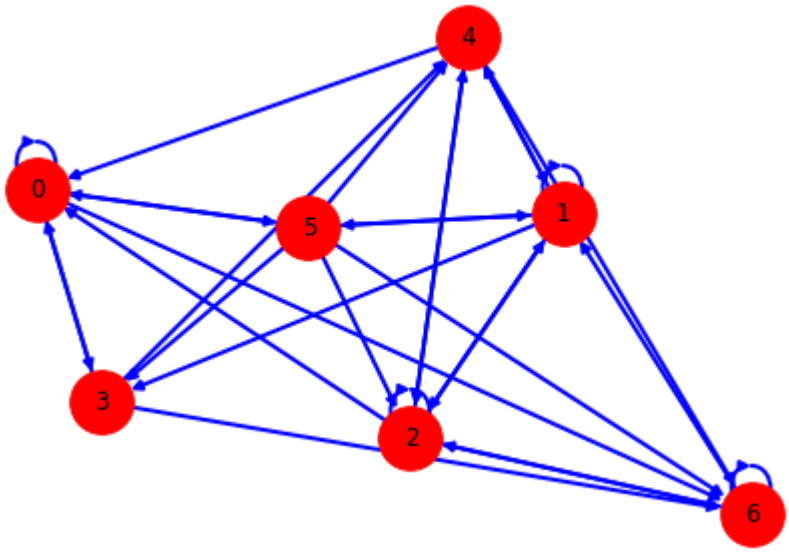
###Directed Graph from Adjacency Matrix with 3 nodes

In [ ]:
```
import networkx as nx
import matplotlib.pyplot as plt
g = nx.DiGraph()
L = [[1,0,0,1,0,1,1],[0,1,1,1,1,1,0],[1,1,1,0,1,0,1]]
g = nx.DiGraph()
g.add_nodes_from([0,1,2])
for i in range(0,3):
  for j in range(0,3):
    if L[i][j] == 1:
      g.add_edge(i,j)
nx.draw(g, with_labels=True, node_size=1000, node_color='red', width=2, edge
plt.show()
```



###Directed Graph from Adjacency Matrix with 7 nodes

In [ ]:
```
import networkx as nx
import matplotlib.pyplot as plt
g = nx.DiGraph()
L = [[1,0,0,1,0,1,1],[0,1,1,1,1,1,0],[1,1,1,0,1,0,1],[1,0,0,0,1,0,1],[1,1,1,
g = nx.DiGraph()
g.add_nodes_from([0,1,2,3,4,5,6])
for i in range(0,7):
  for j in range(0,7):
    if L[i][j] == 1:
      g.add_edge(i,j)
nx.draw(g, with_labels=True, node_size=1000, node_color='red', width=2, edge
plt.show()
```

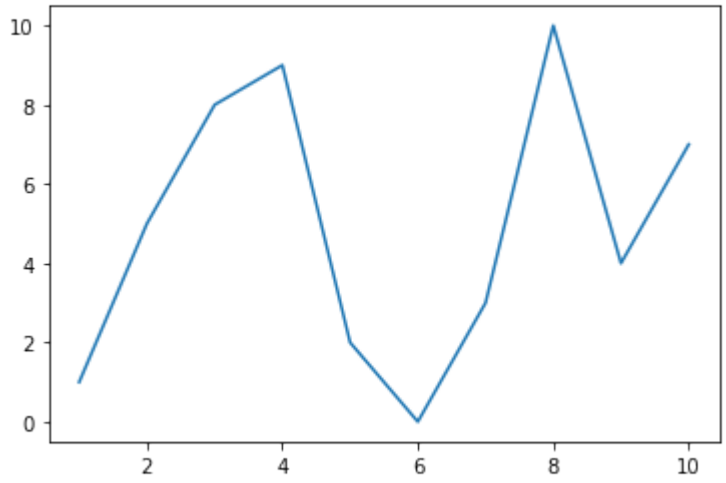


##Starting with Plots

###Defining the Plot

Plots show graphically what you've defined numerically. To define a plot, you need some values, the matplotlib.pyplot module, and an idea of what you want to display, as shown in the following code.

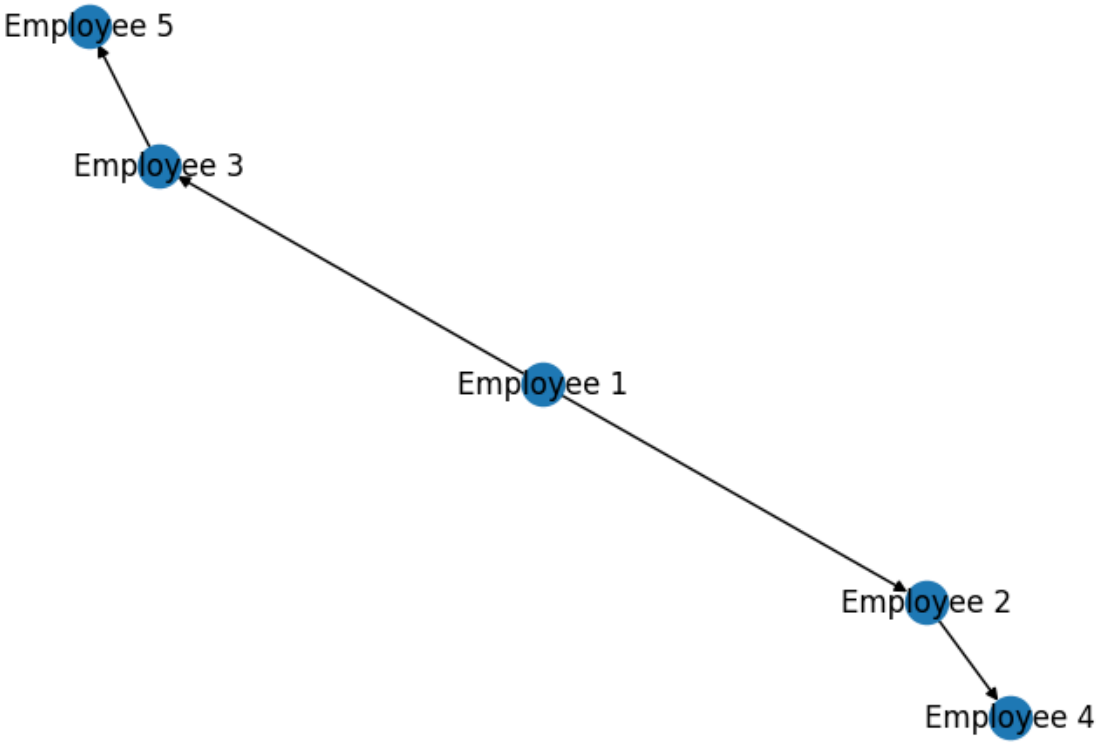
In [ ]:
```
values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
import matplotlib.pyplot as plt
plt.plot(range(1,11), values)
plt.show()
```



You have been hired as a network analyst by a company to analyze the social network of their employees. The company has provided you with the following data: There are 5 employees in the company, each identified by a unique ID from 1 to 5. The following relationships exist between the employees:

1. Employee 1 is friends with Employee 2 and Employee 3.
2. Employee 2 is friends with Employee 4.
3. Employee 3 is friends with Employee 5. Your task is to create a NetworkX graph representing this social network and display it.

In [1]:
```
import networkx as nx
import matplotlib.pyplot as plt
# Create an empty undirected graph
G = nx.DiGraph()
# Add nodes to the graph
G.add_nodes_from([1, 2, 3, 4, 5])
# Add edges to the graph
G.add_edge(1, 2)
G.add_edge(1, 3)
G.add_edge(2, 4)
G.add_edge(3, 5)
# Set the node labels
labels = {1: 'Employee 1', 2: 'Employee 2',
3: 'Employee 3',
4: 'Employee 4', 5: 'Employee 5'}
# Draw the graph with node labels
nx.draw(G, labels=labels, with_labels=True)
plt.show()
```

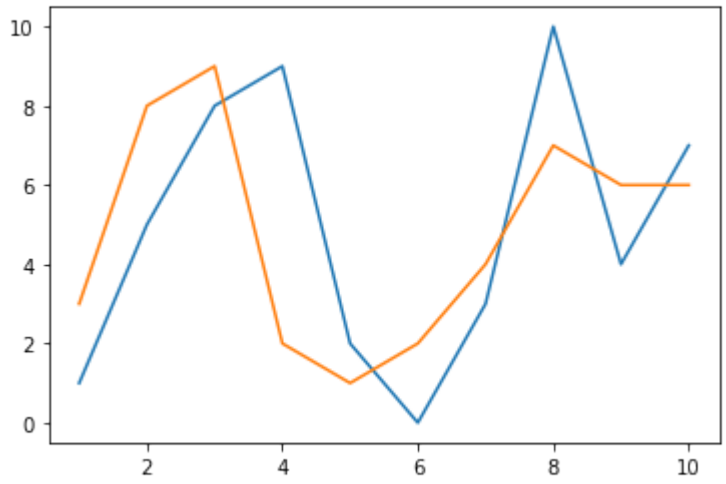


In this case, the code tells the plt.plot() function to create a plot using x-axis values between 1 and 11 and y-axis values as they appear in values. Calling plot.show() displays the plot in a separate dialog box

###Drawing multiple lines and plots

You encounter many situations in which you must use multiple plot lines, such as when comparing two sets of values. To create such plots using MatPlotLib, you simply call plt.plot() multiple times — once for each plot line, as shown in the following example.

In [ ]:
```
values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]
import matplotlib.pyplot as plt
plt.plot(range(1,11), values)
plt.plot(range(1,11), values2)
plt.show()
```

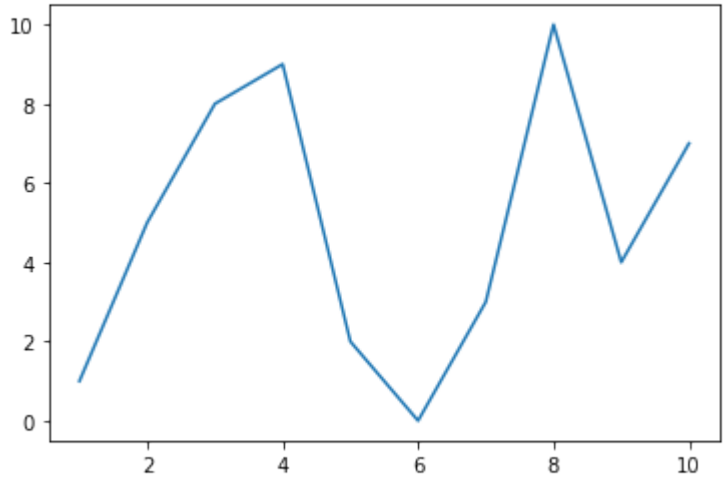


When you run this example, you see two plot lines. The line graphs are different colors so that you can tell them apart.

###Saving your work to disk

Sometimes you need to save the graphic automatically. In this case, you can save it programmatically using the plt.savefig() function, as shown in the following code:

In [ ]:
```
values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
import matplotlib.pyplot as plt
plt.plot(range(1,11), values)
plt.savefig('MySamplePlot.png', format='png')
```



In this case, you must provide a minimum of two inputs. The first input is the filename. You may optionally include a path for saving the file. The second input is the file format. In this case, the example saves the file in Portable Network Graphic (PNG) format, but you have other options: Portable Document Format (PDF), Postscript (PS), Encapsulated Postscript (EPS), and Scalable Vector Graphics (SVG).

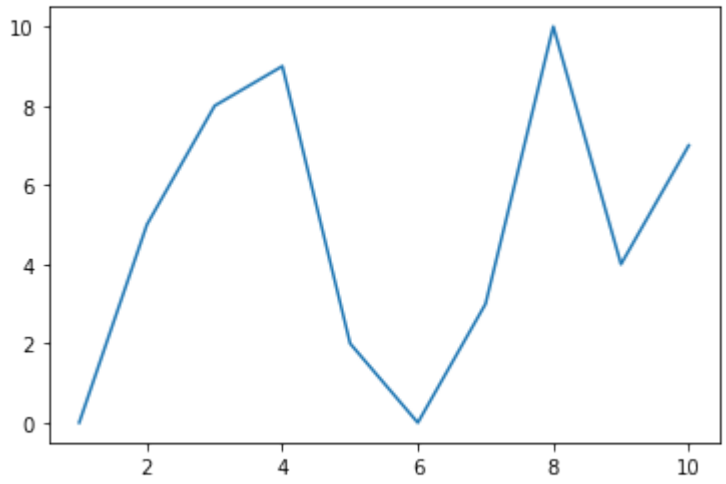
##Setting the Axis

The use of axes, ticks, and grids make it possible to illustrate graphically the relative size of data elements so that the viewer gains an appreciation of comparative measure.

###Getting the axes

The axes define the x and y plane of the graphic. The x axis runs horizontally, and the y axis runs vertically. In many cases, you can allow MatPlotLib to perform any required formatting for you. However, sometimes you need to obtain access to the axes and format them manually. The following code shows how to obtain access to the axes for a plot:

In [ ]:
```
values = [0, 5, 8, 9, 2, 0, 3, 10, 4, 7]
import matplotlib.pyplot as plt
ax = plt.axes()
plt.plot(range(1,11), values)
plt.show()
```

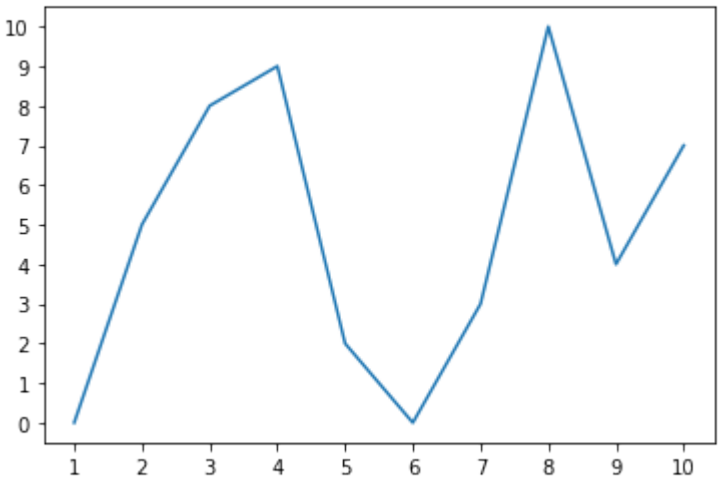


The reason you place the axes in a variable, ax, instead of manipulating them directly is to make writing the code simpler and more efficient. In this case, you simply turn on the default axes by calling plt.axes(); then you place a handle to the axes in ax. A handle is a sort of pointer to the axes. Think of it as you would a frying pan. You wouldn't lift the frying pan directly but would instead use its handle when picking it up.

###Formatting the axes

Simply displaying the axes won't be enough in many cases. You want to change the way MatPlotLib displays them. For example, you may not want the highest value t to reach to the top of the graph. The following example shows just a small number of tasks you can perform after you have access to the axes:

In [ ]:
```
values = [0, 5, 8, 9, 2, 0, 3, 10, 4, 7]
import matplotlib.pyplot as plt
ax = plt.axes()
ax.set_xticks([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
ax.set_yticks([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
plt.plot(range(1,11), values)
plt.show()
```



The set_xticks() and set_yticks() calls change the ticks used to display data.

##Defining the Line Appearance

Just drawing lines on a page won't do much for you if you need to help the viewer understand the importance of your data. In most cases, you need to use different line styles to ensure that the viewer can tell one data grouping from another. However, to emphasize the importance or value of a particular data grouping, you need to employ color. The use of color communicates all sorts of ideas to the viewer. For example, green often denotes that something is safe, while red communicates danger.

###Woking with Line Styles

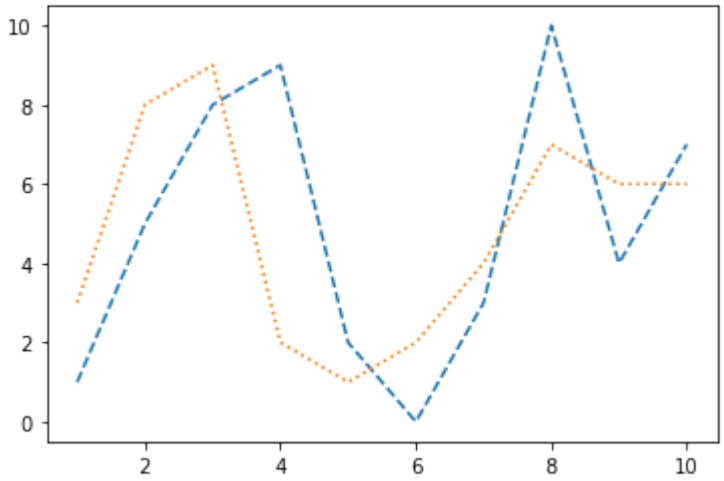
Line styles help differentiate graphs by drawing the lines in various ways. Using a unique presentation for each line helps you distinguish each line so that you can call it out (even when the printout is in shades of gray). You could also call out a particular line graph by using a different line style for it (and using the same style for the other lines).

MatPlotLib Line Styles

Character	*Line Style*
'-'	Solid line
'--'	Dashed line
'-.'	Dash-dot line
':'	Dotted line

The line style appears as a third argument to the plot() function call. You simply provide the desired string for the line type, as shown in the following example.

In [ ]:
```
values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]
import matplotlib.pyplot as plt
plt.plot(range(1,11), values, '--')
plt.plot(range(1,11), values2, ':')
plt.show()
```



In this case, the first line graph uses a dashed line style, while the second line graph uses a dotted line style.

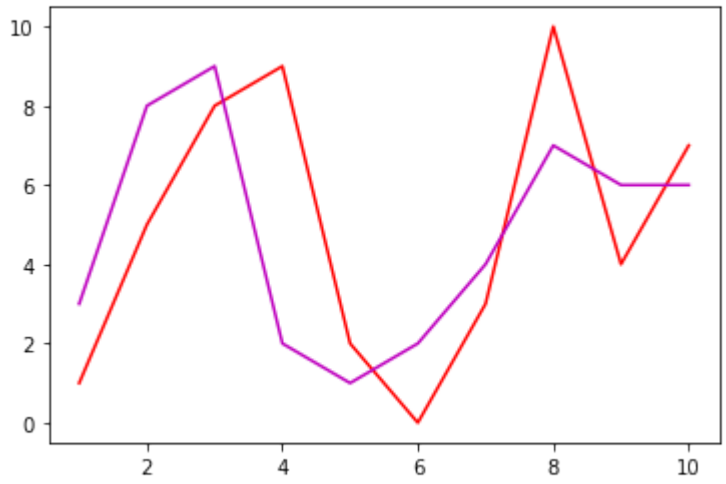
##Using Colour

Colours Supported by MatPlotLib

Character	*Color*
'b'	Blue
'g'	Green
'r'	Red
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

As with line styles, the color appears in a string as the third argument to the plot() function call. In this case, the viewer sees two lines — one in red and the other in magenta.

In [ ]:
```
values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]
import matplotlib.pyplot as plt
plt.plot(range(1,11), values, 'r')
plt.plot(range(1,11), values2, 'm')
plt.show()
```



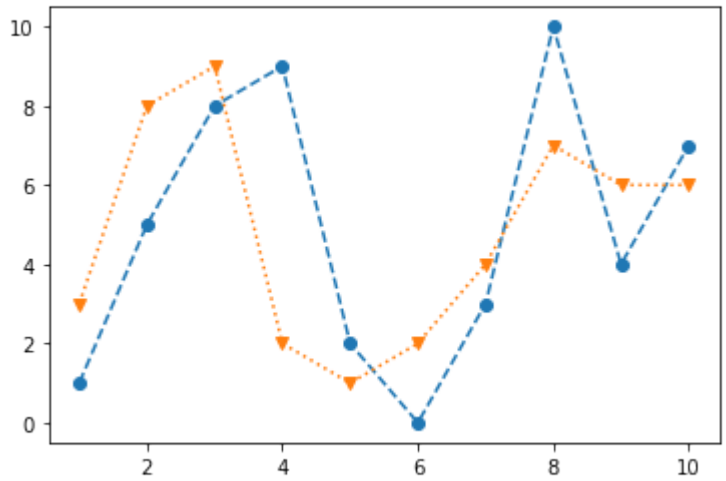
##Adding Markers

Markers add a special symbol to each data point in a line graph.

Character	*Marker Type*
'.'	Point
','	Pixel
'o'	Circle
'v'	Triangle 1 down
'^'	Triangle 1 up
'<'	Triangle 1 left
'>'	Triangle 1 right
'1'	Triangle 2 down
'2'	Triangle 2 up
'3'	Triangle 2 left
'4'	Triangle 2 right
's'	Square
'p'	Pentagon
'*'	Star
'h'	Hexagon style 1
'H'	Hexagon style 2
'+'	Plus
'x'	X
'D'	Diamond
'd'	Thin diamond
'|'	Vertical line
'_'	Horizontal line

As with line style and color, you add markers as the third argument to a plot() call. In the following example, you see the effects of combining line style with a marker to provide a unique line graph presentation.

In [ ]:
```
values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]
import matplotlib.pyplot as plt
plt.plot(range(1,11), values, 'o--')
plt.plot(range(1,11), values2, 'v:')
plt.show()
```



##Using Labels, Annotations, and Legends

To fully document your graph, you usually have to resort to labels, annotations, and legends. Each of these elements has a different purpose, as follows:

✓✓Label: Provides positive identification of a particular data element or grouping. The purpose is to make it easy for the viewer to know the name or kind of data illustrated.

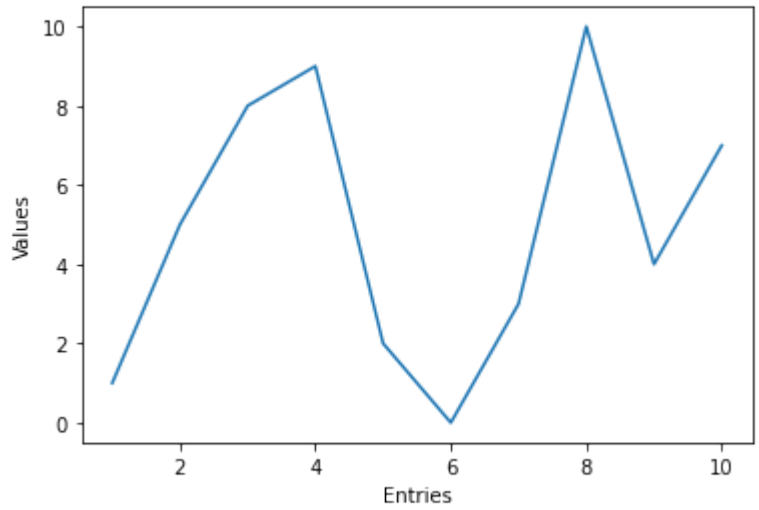
✓✓Annotation: Augments the information the viewer can immediately see about the data with notes, sources, or other useful information. In contrast to a label, the purpose of annotation is to help extend the viewer's knowledge of the data rather than simply identify it.

✓✓Legend: Presents a listing of the data groups within the graph and often provides cues (such as line type or color) to make identification of the data group easier. For example, all the red points may belong to group A, while all the blue points may belong to group B.

###Adding labels

Labels help people understand the significance of each axis of any graph you create. Without labels, the values portrayed don't have any significance. In addition to a moniker, such as rainfall, you can also add units of measure, such as inches or centimeters, so that your audience knows how to interpret the data shown. The following example shows how to add labels to your graph:

In [ ]:
```
values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
import matplotlib.pyplot as plt
plt.xlabel('Entries')
plt.ylabel('Values')
plt.plot(range(1,11), values)
plt.show()
```

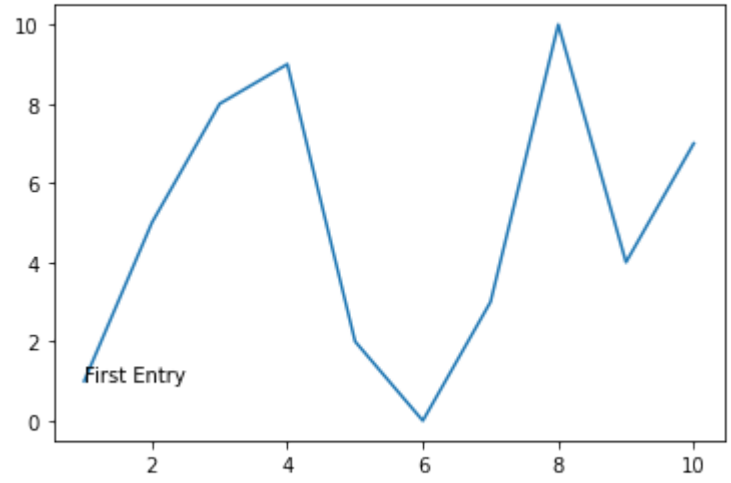


The call to xlabel() documents the x axis of your graph, while the call the ylabel() documents the y axis of your graph.

###Annotating the chart

You use annotation to draw special attention to points of interest on a graph. For example, you may want to point out that a specific data point is outside the usual range expected for a particular dataset. The following example shows how to add annotation to a graph.

In [ ]:
```
values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
import matplotlib.pyplot as plt
plt.annotate(xy=[1,1], s='First Entry')
plt.plot(range(1,11), values)
plt.show()
```



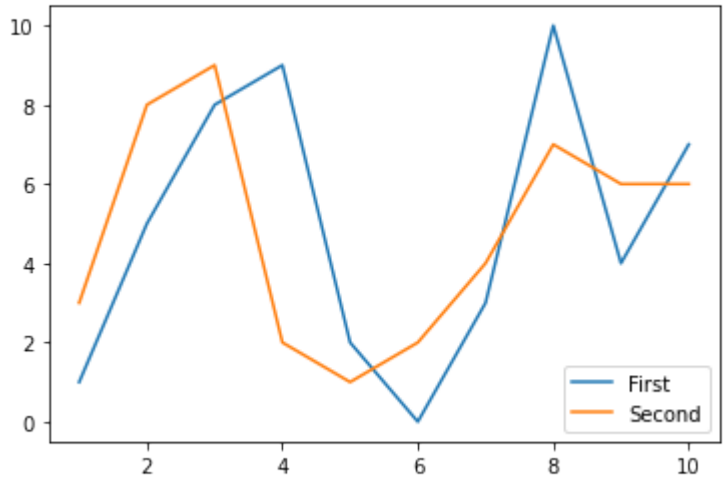
The call to annotate() provides the labeling you need. You must provide a location for the annotation by using the xy parameter, as well as provide text to place at the location by using the s parameter. The annotate() function also provides other parameters that you can

use to create special formatting or placement onscreen.

###Creating a legend

A legend documents the individual elements of a plot. Each line is presented in a table that contains a label for it so that people can differentiate between each line. For example, one line may represent sales in 2014 and another line may represent sales in 2015, so you include an entry in the legend for each line that is labeled 2014 and 2015. The following example shows how to add a legend to your plot.

In [ ]:
```
values = [1, 5, 8, 9, 2, 0, 3, 10, 4, 7]
values2 = [3, 8, 9, 2, 1, 2, 4, 7, 6, 6]
import matplotlib.pyplot as plt
line1 = plt.plot(range(1,11), values)
line2 = plt.plot(range(1,11), values2)
plt.legend(['First', 'Second'], loc=0)
plt.show()
```



The call to legend() occurs after you create the plots, not before, as with some of the other functions described in this chapter. You must provide a handle to each of the plots. Notice how line1 is set equal to the first plot() call and line2 is set equal to the second plot() call. The default location for the legend is the upper-right corner of the plot, which proved inconvenient for this particular example. Adding the loc parameter lets you place the legend in a different location.

Location Code The string 'best' places the legend at the location, among the nine locations defined so far, with the minimum overlap with other drawn artists. This option can be quite slow for plots with large amounts of data; your plotting speed may benefit from providing a specific location.

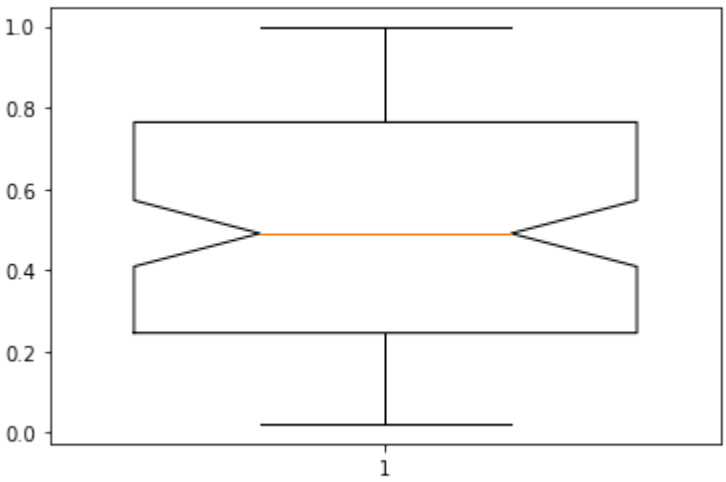
Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

##Depicting groups using box plots

Box plots provide a means of depicting groups of numbers through their quartiles (three points dividing a group into four equal parts). A box plot may also have lines, called whiskers, indicating data outside the upper and lower quartiles. The spacing shown within a box plot helps indicate the skew and dispersion of the data. The following example shows how to create a box plot with randomized data.

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data = np.random.rand(100)
print(data)
plt.boxplot(data, widths=0.75, notch=True)
plt.show()
df = pd.DataFrame(data)
df.describe()
```

```
[0.48364155 0.74443544 0.90957398 0.56670075 0.6035898  0.19273642
 0.34158054 0.44701421 0.80004038 0.89611798 0.59747664 0.60322175
 0.5050701  0.68994933 0.38188729 0.78208702 0.24962534 0.10507605
 0.82767139 0.39552329 0.97008222 0.66432551 0.23045654 0.15828333
 0.99673945 0.13292209 0.35236237 0.25275013 0.78218925 0.85126987
 0.24331957 0.38829111 0.4191286  0.18107402 0.43340731 0.49313244
 0.91919692 0.94380602 0.17653672 0.61981375 0.8644114  0.45596857
 0.49806737 0.42077926 0.09987478 0.86096489 0.95076565 0.62211225
 0.7578719  0.27882884 0.74539356 0.82357236 0.24067143 0.94774861
 0.74095655 0.24576883 0.15290277 0.72767504 0.40920639 0.15301477
 0.48706619 0.04212436 0.06546059 0.44942077 0.82095955 0.40287768
 0.91302126 0.49398362 0.02195523 0.06915274 0.70610377 0.23370683
 0.46793316 0.05962644 0.11747195 0.13066853 0.83517172 0.71672314
 0.18961878 0.56849417 0.92541072 0.02991776 0.32510416 0.45993223
 0.98503383 0.28308742 0.41471164 0.09993233 0.51329696 0.59101684
 0.6328024  0.14392462 0.44374082 0.91557311 0.89370896 0.75350164
 0.80617077 0.93091021 0.74187662 0.13864116]
```



Out[25]:

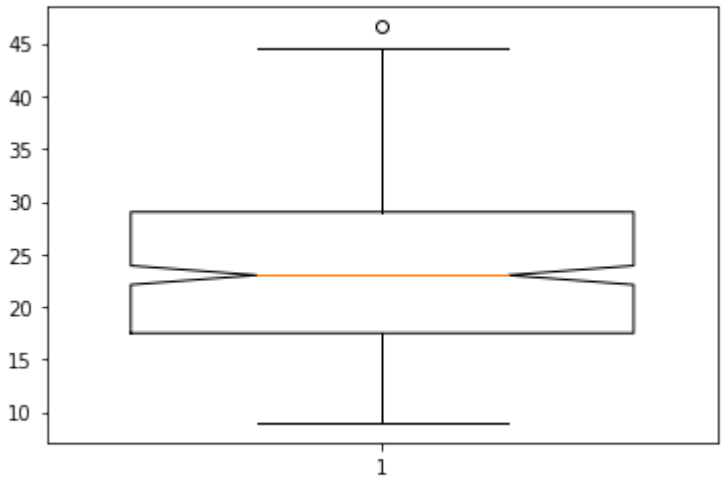
	0
count	100.000000
mean	0.511485
std	0.290328
min	0.021955
25%	0.245157
50%	0.490099
75%	0.763926
max	0.996739

The call to boxplot()requires only data as input. All other parameters have default settings. In this case, the code sets the presentation of outliers to green Xs by setting the sym parameter. You use widths to modify the size of the box (made extra large in this case to make the box easier to see). Finally, you can create a square box or a box with a notch using the notch parameter (which normally defaults to False).

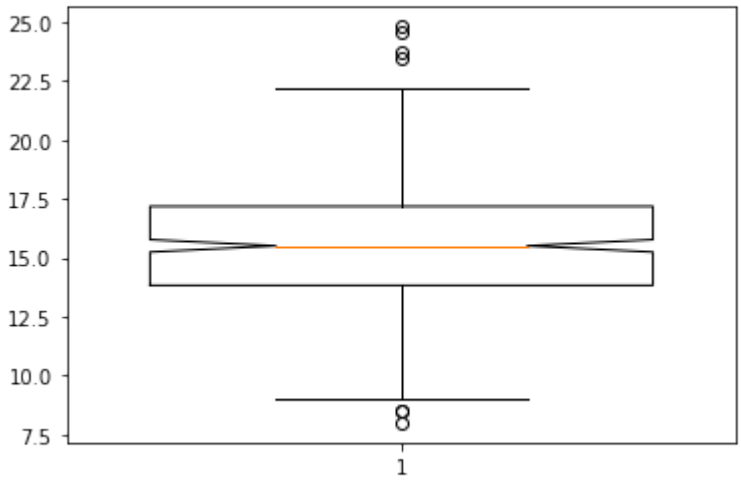
The box shows the three data points as the box, with the red line in the middle being the median. The two black horizontal lines connected to the box by whiskers show the upper and lower limits (for four quartiles). The outliers appear above and below the upper and lower limit lines as green Xs.

Boxplots can be useful for detecting outliers too. As seen in the previous chapter in the auto-mpg dataset, the mpg and acceleration columns had outliers. They can be clearly seen by creating boxplots.

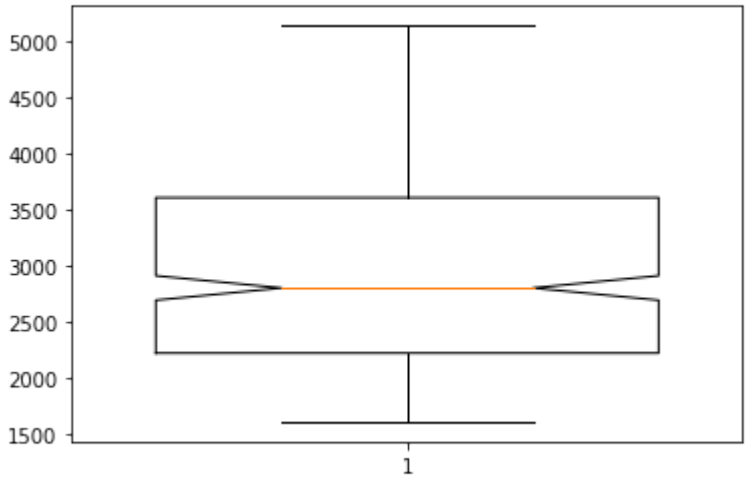
In [ ]:
```
import pandas as pd
dataset = pd.read_csv('https://raw.githubusercontent.com/Jovita7/Data-Analys
import matplotlib.pyplot as plt
plt.boxplot(dataset['mpg'], widths=0.75, notch=True)
plt.show()
```



In [ ]:
```
plt.boxplot(dataset['acceleration'], widths=0.75, notch=True)
plt.show()
```



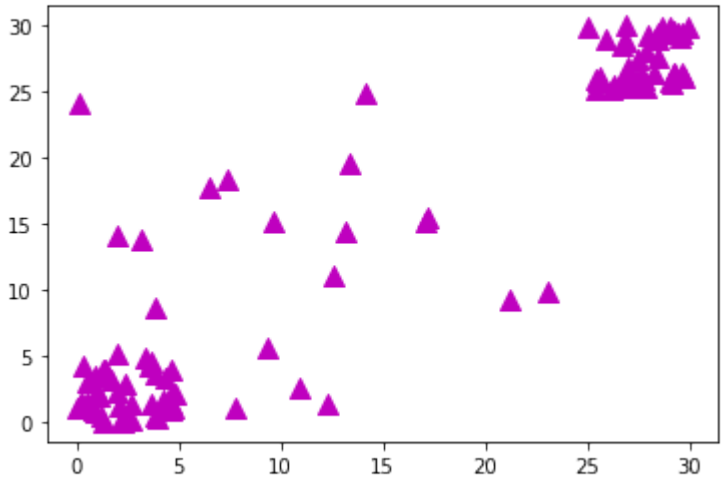
In [ ]:
```
plt.boxplot(dataset['weight'], widths=0.75, notch=True)
plt.show()
```



##Seeing data patterns using scatterplots

Scatterplots show clusters of data rather than trends (as with line graphs) or discrete values (as with bar charts). The purpose of a scatterplot is to help you see data patterns. The following example shows how to create a scatterplot using randomized data:

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
x1 = 5 * np.random.rand(40)
x2 = 5 * np.random.rand(40) + 25
x3 = 25 * np.random.rand(20)
x = np.concatenate((x1, x2, x3))
y1 = 5 * np.random.rand(40)
y2 = 5 * np.random.rand(40) + 25
y3 = 25 * np.random.rand(20)
y = np.concatenate((y1, y2, y3))
plt.scatter(x, y, s=[100], marker='^', c='m')
plt.show()
```



The example begins by generating random x and y coordinates. For each x coordinate, you must have a corresponding y coordinate. It's possible to create a scatterplot using just the x and y coordinates. It's possible to dress up a scatterplot in a number of ways. In this case,

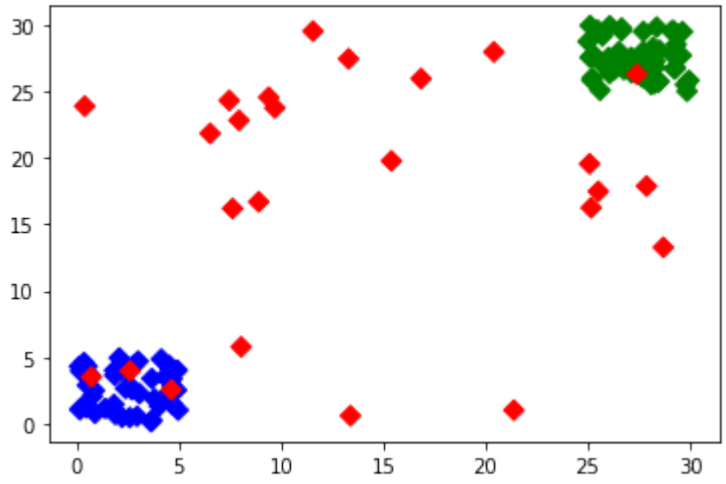
the s parameter determines the size of each data point. The marker parameter determines the data point shape. You use the c parameter to define the colors for all the data points, or

##Creating Advanced Scatterplots

Scatterplots are especially important for data science because they can show data patterns that aren't obvious when viewed in other ways. You can see data groupings with relative ease and help the viewer understand when data belongs to a particular group. You can also show overlaps between groups and even demonstrate when certain data is outside the expected range. Showing these various kinds of relationships in the data is an advanced technique that you need to know in order to make the best use of MatPlotLib.

Color is the third axis when working with a scatterplot. Using color lets you highlight groups so that others can see them with greater ease. The following example shows how you can use color to show groups within a scatterplot:

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
x1 = 5 * np.random.rand(50)
x2 = 5 * np.random.rand(50) + 25
x3 = 30 * np.random.rand(25)
x = np.concatenate((x1, x2, x3))
y1 = 5 * np.random.rand(50)
y2 = 5 * np.random.rand(50) + 25
y3 = 30 * np.random.rand(25)
y = np.concatenate((y1, y2, y3))
color_array = ['b'] * 50 + ['g'] * 50 + ['r'] * 25
plt.scatter(x, y, s=[50], marker='D', c=color_array)
plt.show()
```



The example works essentially the same as the scatterplot example in the previous section, except that this example uses an array for the colors. The first group is blue, followed by green for the second group. Any outliers appear in red.

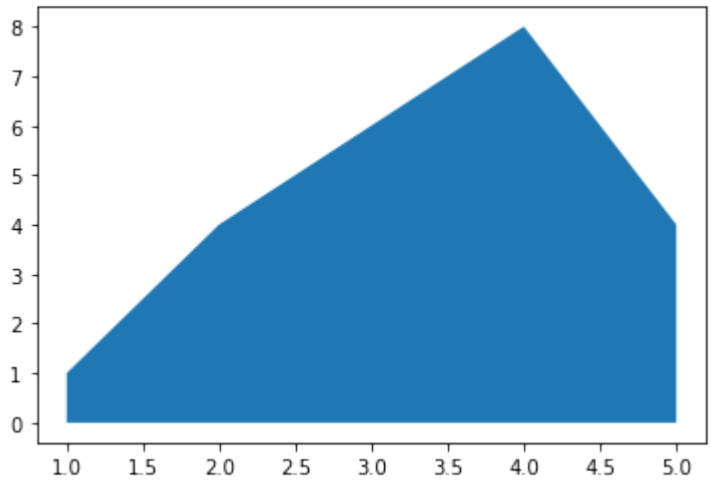
##Area plots/charts

An area chart combines the line chart and bar chart to show how one or more groups' numeric values change over the progression of a second variable, typically that of time. An area chart is distinguished from a line chart by the addition of shading between lines and a

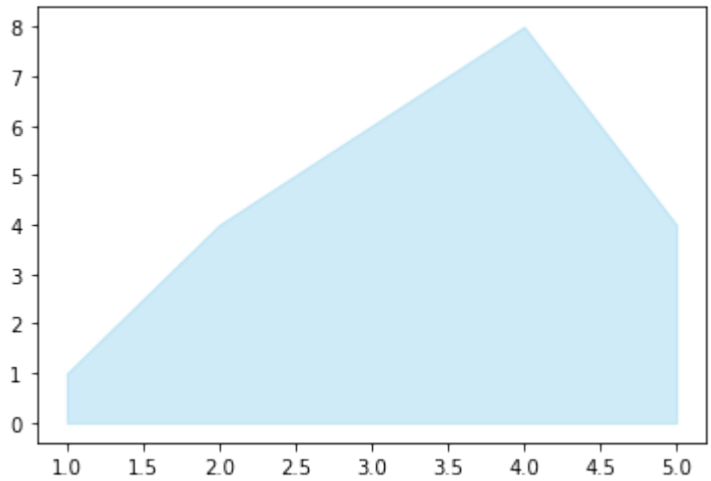
An area chart is typically used with multiple lines to make a comparison between groups (aka series) or to show how a whole is divided into component parts.

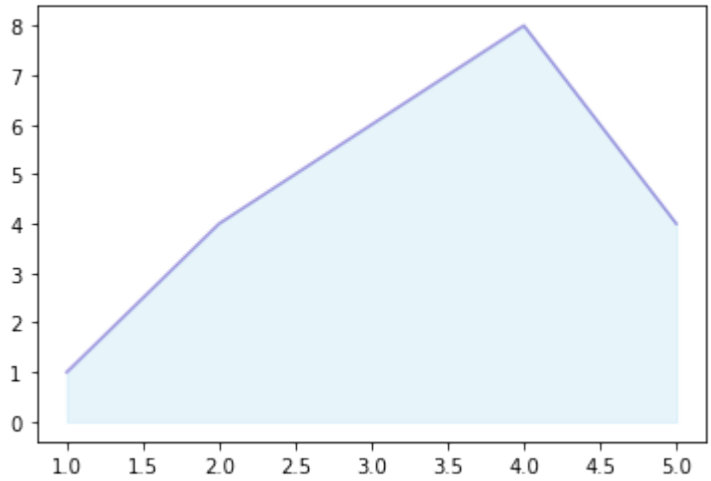
###Simple Area Chart

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

# Create data
x=range(1,6)
y=[1,4,6,8,4]

# Area plot
plt.fill_between(x, y)
plt.show()
```



In [ ]:
```
# Change the color and its transparency
plt.fill_between( x, y, color="skyblue", alpha=0.4)

# Show the graph
plt.show()

# Same, but add a stronger line on top (edge)
plt.fill_between( x, y, color="skyblue", alpha=0.2)
plt.plot(x, y, color="Slateblue", alpha=0.6)
# See the line plot function to learn how to customize the plt.plot functior

# Show the graph
plt.show()
```



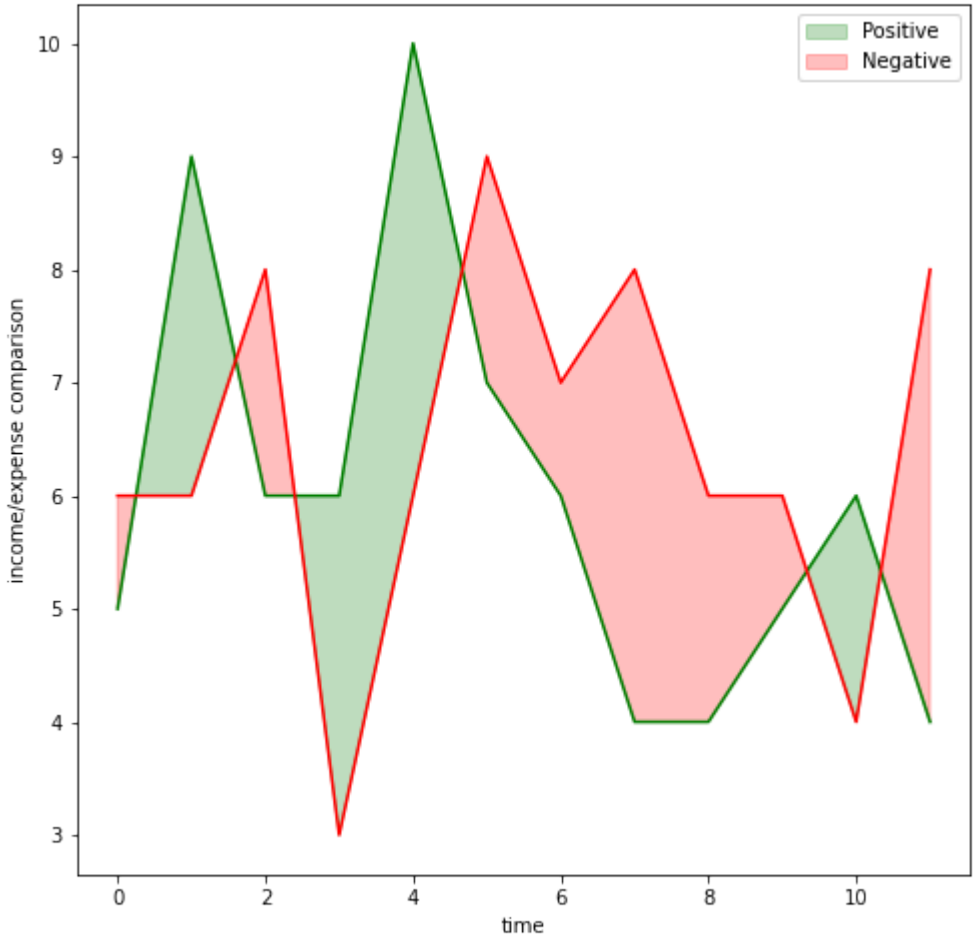


###Area fill between two lines in Matplotlib

NumPy arange() is one of the array creation routines based on numerical ranges. It creates an instance of ndarray with evenly spaced values and returns the reference to it.

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
time = np.arange(12)
income = np.array([5, 9, 6, 6, 10, 7, 6, 4, 4, 5, 6, 4])
expenses = np.array([6, 6, 8, 3, 6, 9, 7, 8, 6, 6, 4, 8])

# Initialize figure and axis
fig, ax = plt.subplots(figsize=(8, 8))

# Plot lines
ax.plot(time, income, color="green")
ax.plot(time, expenses, color="red")

# Fill area when income > expenses with green
ax.fill_between(
    time, income, expenses, where=(income > expenses),
    color="green", alpha=0.25,
    label="Positive", interpolate = True
)

# Fill area when income <= expenses with red
ax.fill_between(
    time, income, expenses, where=[i<=j for i,j in zip(income,expenses)],
    color="red", alpha=0.25,
    label="Negative", interpolate = True
)

plt.xlabel('time')
plt.ylabel('income/expense comparison')

ax.legend()

plt.show()
```

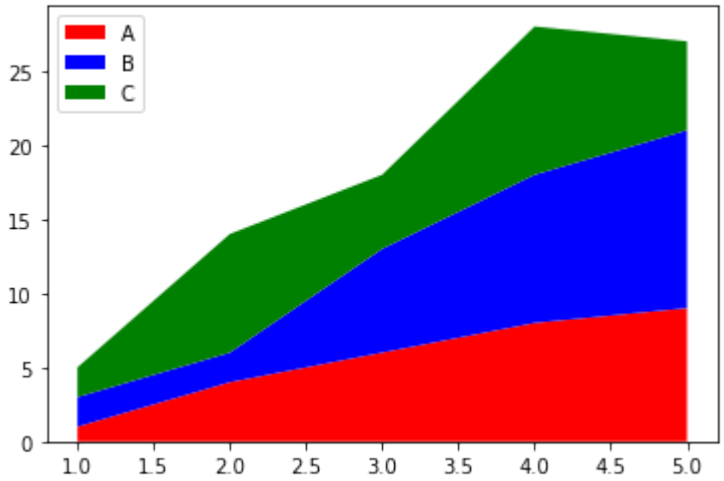


###Stacked area Chart

A stacked area chart displays the evolution of a numeric variable for several groups of a dataset. Each group is displayed on top of each other, making it easy to read the evolution of the total, but hard to read each group value accurately. In python, stacked area charts are mainly done thanks to the stackplot() function

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

# Create data
x=range(1,6)
y1=[1,4,6,8,9]
y2=[2,2,7,10,12]
y3=[2,8,5,10,6]

# Basic stacked area chart.
plt.stackplot(x,y1, y2, y3, labels=['A','B','C'], colors = ['red', 'blue', '
plt.legend(loc='upper left')
```

Out[16]: <matplotlib.legend.Legend at 0x7f737fbff5e0>



##Waffle Charts

A waffle chart is an interesting visualization that is normally created to display progress toward goals. It is commonly an effective option when you are trying to add interesting visualization features to a visual that consists mainly of cells, such as an Excel dashboard.

In [ ]:
```
!pip install pywaffle
```

```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) ht
tps://us-python.pkg.dev/colab-wheels/public/simple/ (https://us-python.pk
g.dev/colab-wheels/public/simple/)
Collecting pywaffle
  Downloading pywaffle-1.1.0-py2.py3-none-any.whl (30 kB)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.8/dist
-packages (from pywaffle) (3.2.2)
Collecting fontawesomefree
  Downloading fontawesomefree-6.2.1-py3-none-any.whl (25.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 25.1/25.1 MB 38.7 MB/s eta
0:00:00
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.8/di
st-packages (from matplotlib->pywaffle) (0.11.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
/usr/local/lib/python3.8/dist-packages (from matplotlib->pywaffle) (3.0.9)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python
3.8/dist-packages (from matplotlib->pywaffle) (1.4.4)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.8/dis
t-packages (from matplotlib->pywaffle) (1.21.6)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/pyth
on3.8/dist-packages (from matplotlib->pywaffle) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-p
ackages (from python-dateutil>=2.1->matplotlib->pywaffle) (1.15.0)
Installing collected packages: fontawesomefree, pywaffle
Successfully installed fontawesomefree-6.2.1 pywaffle-1.1.0
```

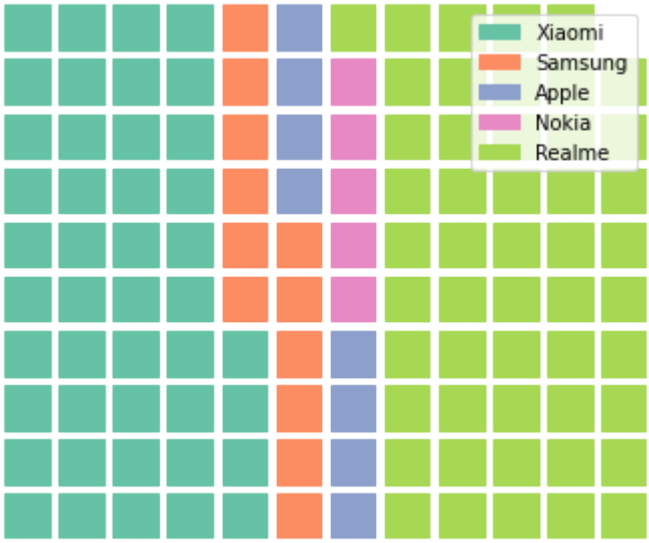
In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
from pywaffle import Waffle

# creation of a dataframe
data ={'phone': ['Xiaomi', 'Samsung',
                 'Apple', 'Nokia', 'Realme'],
       'stock': [44, 12, 8, 5, 50]
     }

df = pd.DataFrame(data)
```

In [ ]:
```
df
```

Out[8]:

	phone	stock
0	Xiaomi	44
1	Samsung	12
2	Apple	8
3	Nokia	5
4	Realme	50

In [ ]:
```
# To plot the waffle Chart
fig = plt.figure(
    FigureClass = Waffle,
    rows = 10,
    values = df.stock,
    labels = list(df.phone)
)
```



In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
from pywaffle import Waffle

# creation of a dataframe
Data={'USA': 113, 'China': 88, 'Japan': 58, 'Great Britain': 65, 'ROC': 71,
d={"s":list(Data.keys()),"m":list(Data.values())}
print(d)

df = pd.DataFrame(d)
df
```

```
{'s': ['USA', 'China', 'Japan', 'Great Britain', 'ROC', 'Australia', 'Neth
erlands', 'France', 'Germany', 'Italy'], 'm': [113, 88, 58, 65, 71, 46, 3
6, 33, 37, 40]}
```

Out[14]:

	s	m
0	USA	113
1	China	88
2	Japan	58
3	Great Britain	65
4	ROC	71
5	Australia	46
6	Netherlands	36
7	France	33
8	Germany	37
9	Italy	40

In [ ]:

```
# To plot the waffle Chart
fig = plt.figure(figsize=(10,10),
    FigureClass = Waffle,
    rows = 30,
    values = df.m,
    labels = list(df.s)
)
```

##Word Clouds

`Word` clouds (also known as text clouds or tag clouds) work in a simple way: the more a specific word appears in a source of textual data (such as a speech, blog post, or database), the bigger and bolder it appears in the word cloud.

Luckily, a Python package already exists in Python for generating `word` clouds. The package, called `word_cloud` was developed by **Andreas Mueller**. You can learn more about the package by following this link (https://github.com/amueller/word_cloud/).

Let's use this package to learn how to generate a word cloud for a given text document.

First, let's install the package.

In [ ]:
```
# install wordcloud
!pip3 install wordcloud==1.8.1

# import package and its set of stopwords
from wordcloud import WordCloud, STOPWORDS

print ('Wordcloud is installed and imported!')
```

```
Requirement already satisfied: wordcloud==1.8.1 in c:\programdata\anaconda
3\lib\site-packages (1.8.1)
Requirement already satisfied: numpy>=1.6.1 in c:\programdata\anaconda3\li
b\site-packages (from wordcloud==1.8.1) (1.19.2)
Requirement already satisfied: matplotlib in c:\programdata\anaconda3\lib
\site-packages (from wordcloud==1.8.1) (3.3.2)
Requirement already satisfied: pillow in c:\programdata\anaconda3\lib\site
-packages (from wordcloud==1.8.1) (8.0.1)
Requirement already satisfied: python-dateutil>=2.1 in c:\programdata\anac
onda3\lib\site-packages (from matplotlib->wordcloud==1.8.1) (2.8.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
c:\programdata\anaconda3\lib\site-packages (from matplotlib->wordcloud==1.
8.1) (2.4.7)
Requirement already satisfied: certifi>=2020.06.20 in c:\programdata\anaco
nda3\lib\site-packages (from matplotlib->wordcloud==1.8.1) (2020.6.20)
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\li
b\site-packages (from matplotlib->wordcloud==1.8.1) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anacond
a3\lib\site-packages (from matplotlib->wordcloud==1.8.1) (1.3.0)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\si
te-packages (from python-dateutil>=2.1->matplotlib->wordcloud==1.8.1) (1.1
5.0)
Wordcloud is installed and imported!
```

`Word` clouds are commonly used to perform high-level analysis and visualization of text data. Let's try to analyze a short novel written by **Lewis Carroll** titled *Alice's Adventures in Wonderland*. Let's go ahead and download a *.txt* file of the novel.

Due to Google, Colab, I'm using urllib to open the file

In [ ]:
```
# open the file and read it into a variable alice_novel
#Due to Google, Colab, I'm using urllib to open the file
import urllib

# open the file and read it into a variable alice_novel
alice_novel = urllib.request.urlopen('https://cf-courses-data.s3.us.cloud-ob
```

Next, let's use the stopwords that we imported from `word_cloud` . We use the function *set*

In [ ]:
```
stopwords = set(STOPWORDS)
stopwords
```

Out[4]:
```
{'a',
 'about',
 'above',
 'after',
 'again',
 'against',
 'all',
 'also',
 'am',
 'an',
 'and',
 'any',
 'are',
 "aren't",
 'as',
 'at',
 'be',
 'because',
 'been',
```

Create a word cloud object and generate a word cloud. For simplicity, let's generate a word cloud using only the first 2000 words in the novel.

In [ ]:
```
# instantiate a word cloud object
alice_wc = WordCloud(
    background_color='white',
    max_words=2000,
    stopwords=stopwords
)

# generate the word cloud
alice_wc.generate(alice_novel)
```

Out[47]:
```
<wordcloud.wordcloud.WordCloud at 0x7fd84f6e7580>
```

Awesome! Now that the `word` cloud is created, let's visualize it.

In [ ]:
```
# display the word cloud
import matplotlib.pyplot as plt
plt.imshow(alice_wc)
plt.axis('off')
plt.show()
```

The matplotlib function imshow() creates an image from a 2-dimensional numpy array.

Interesting! So in the first 2000 words in the novel, the most common words are **Alice**, **said**, **little**, **Queen**, and so on. Let's resize the cloud so that we can see the less frequent words a little better.

In [ ]:
```
fig = plt.figure(figsize=(14, 18))

# display the cloud
plt.imshow(alice_wc)
plt.axis('off')
plt.show()
```

Much better! However, **said** isn't really an informative word. So let's add it to our stopwords and re-generate the cloud.

In [ ]:
```
stopwords.add('said') # add the words said to stopwords

# re-generate the word cloud
alice_wc.generate(alice_novel)

# display the cloud
fig = plt.figure(figsize=(14, 18))

plt.imshow(alice_wc)
plt.axis('off')
plt.show()
```

In [ ]:
```python
import requests
import matplotlib.pyplot as plt
url="https://www.gutenberg.org/files/11/11-0.txt"
response=requests.get(url)
text=response.text
alice_wc = WordCloud(
    background_color='white',
    max_words=2000,
    stopwords=stopwords
)

# generate the word cloud
alice_wc.generate(text)
fig = plt.figure(figsize=(14, 18))

plt.imshow(alice_wc)
plt.axis('off')
plt.show()
```

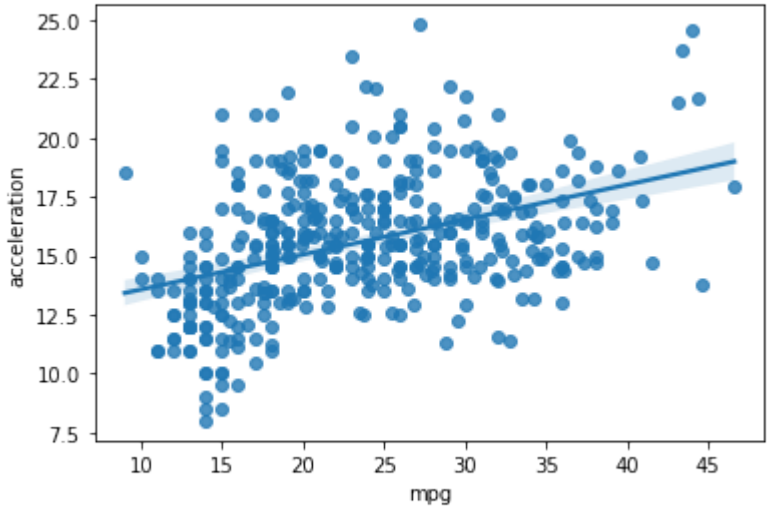
Regression Plots

> Seaborn is a Python visualization library based on matplotlib. It provides a high-level interface for drawing attractive statistical graphics. You can learn more about *seaborn* by following this link (https://seaborn.pydata.org/?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork970-2022-01-01) and more about *seaborn* regression plots by following this link (http://seaborn.pydata.org/generated/seaborn.regplot.html?utm_medium=Exinfluencer&utm_source=Exinfluencer&utm_content=000026UJ&utm_SkillsNetwork-Channel-SkillsNetworkCoursesIBMDeveloperSkillsNetworkDV0101ENSkillsNetwork970-2022-01-01).

With *seaborn*, generating a regression plot is as simple as calling the **regplot** function.

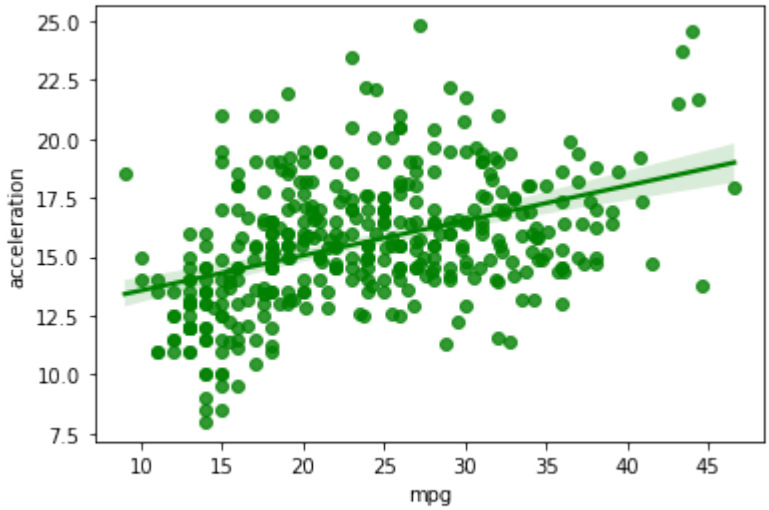
In [ ]:
```
import seaborn as sns
import pandas as pd
dataset = pd.read_csv('https://raw.githubusercontent.com/Jovita7/Data-Analys
sns.regplot(x='mpg', y='acceleration', data=dataset)
```

Out[2]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f02177095b0>
```



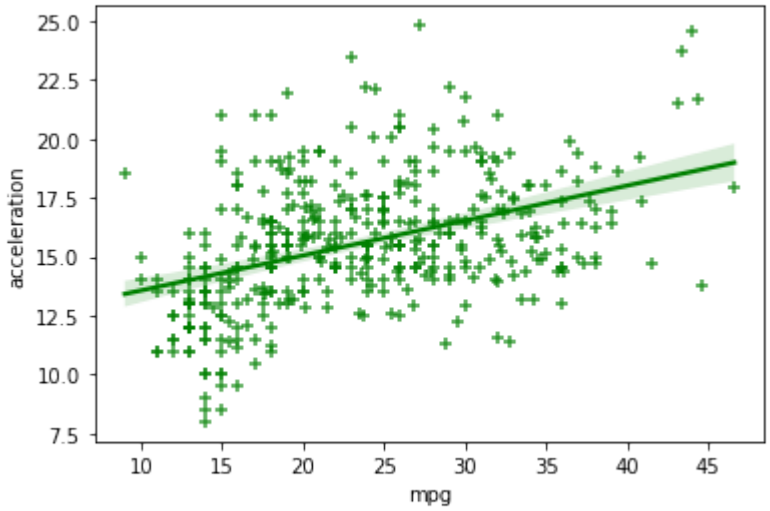
This is not magic; it is *seaborn*! You can also customize the color of the scatter plot and regression line. Let's change the color to green.

In [ ]:
```
sns.regplot(x='mpg', y='acceleration', data = dataset, color='green')
plt.show()
```



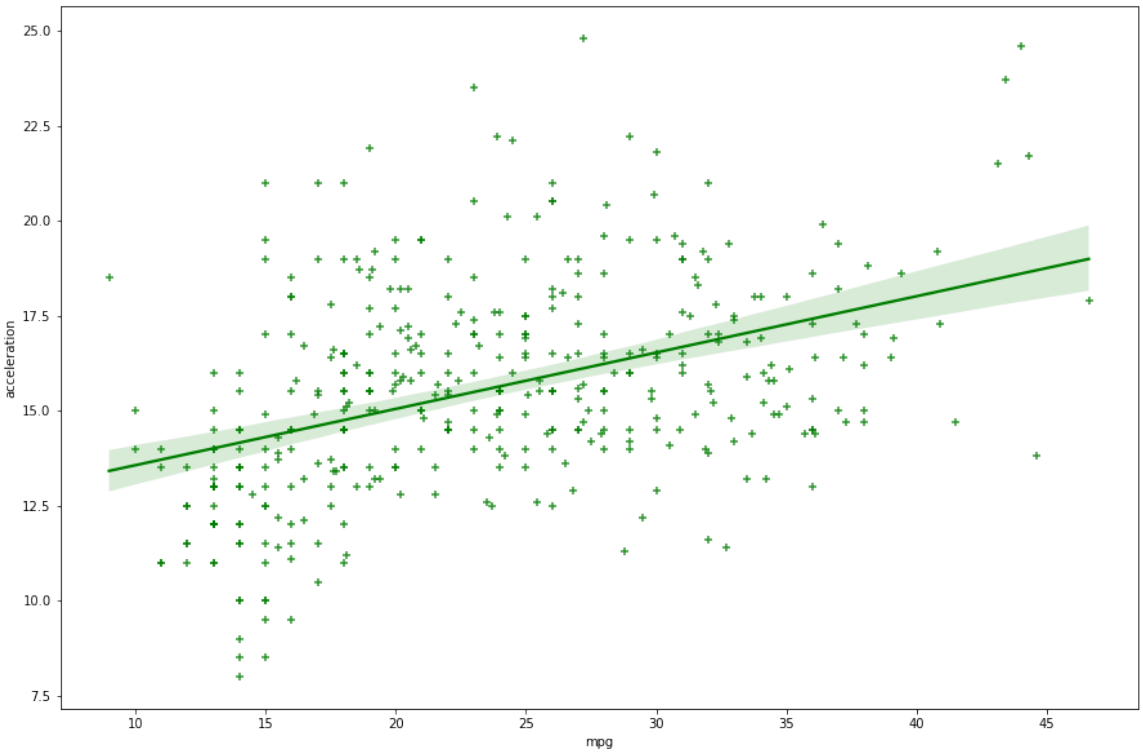
You can always customize the marker shape, so instead of circular markers, let's use `+`.

In [ ]:
```
sns.regplot(x='mpg', y='acceleration', data = dataset, color='green', marker
plt.show()
```



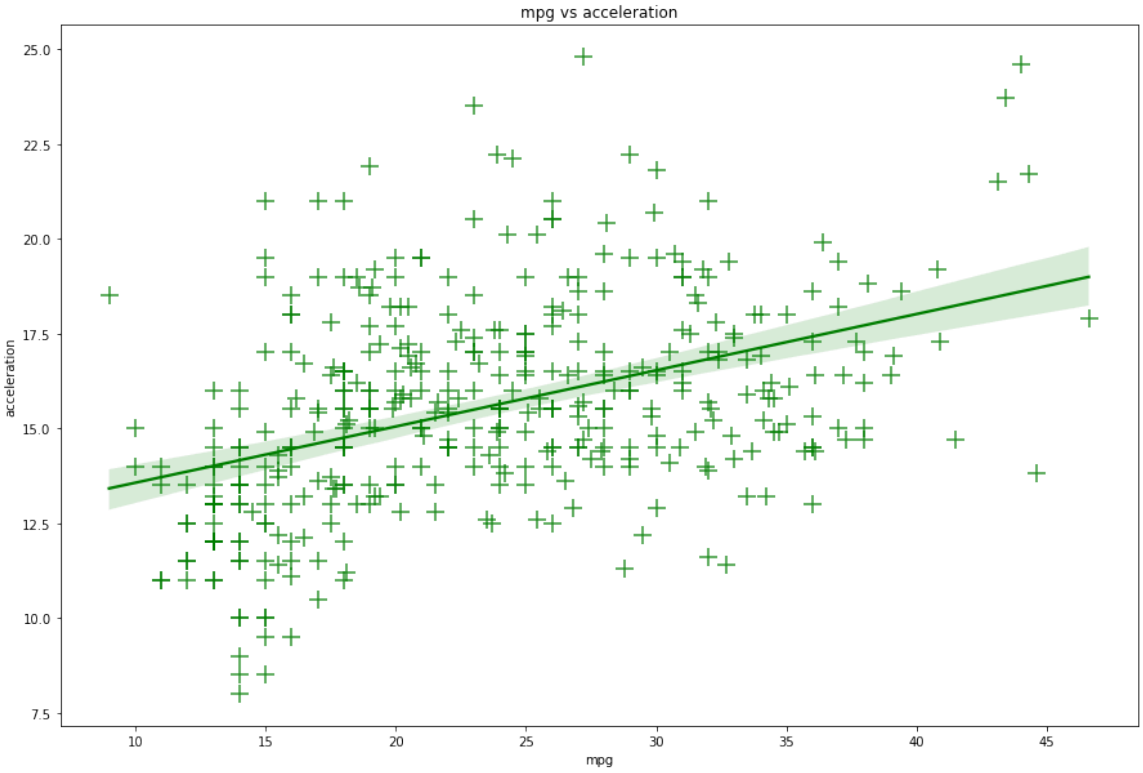
Let's blow up the plot a little so that it is more appealing to the sight.

In [ ]:
```
plt.figure(figsize=(15, 10))
sns.regplot(x='mpg', y='acceleration', data = dataset, color='green', marker
plt.show()
```



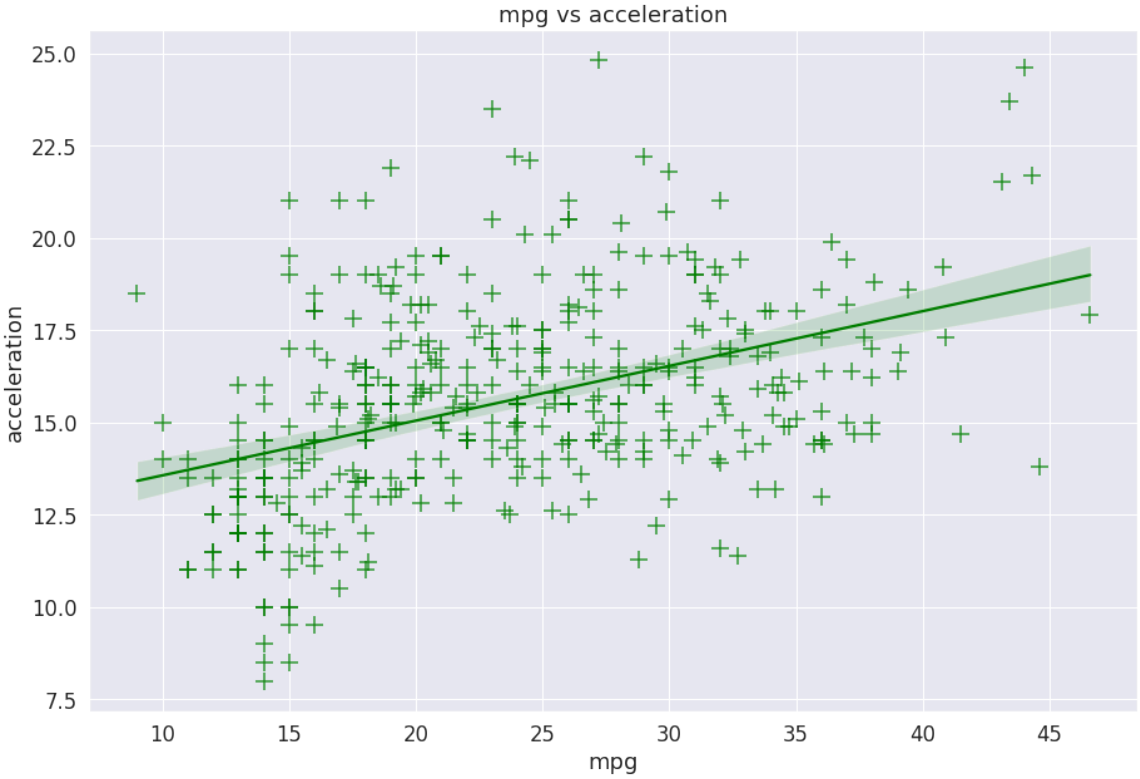
And let's increase the size of markers so they match the new size of the figure, and add a title and x- and y-labels.

In [ ]:
```
plt.figure(figsize=(15, 10))
ax = sns.regplot(x='mpg', y='acceleration', data = dataset, color='green', m
ax.set(xlabel='mpg', ylabel='acceleration') # add x- and y-labels
ax.set_title('mpg vs acceleration') # add title
plt.show()
```



And finally increase the font size of the tickmark labels, the title, and the x- and y-labels so they don't feel left out!

In [ ]:
```
plt.figure(figsize=(15, 10))
sns.set(font_scale=1.5)

ax = sns.regplot(x='mpg', y='acceleration', data = dataset, color='green', m
ax.set(xlabel='mpg', ylabel='acceleration') # add x- and y-labels
ax.set_title('mpg vs acceleration') # add title
plt.show()
```



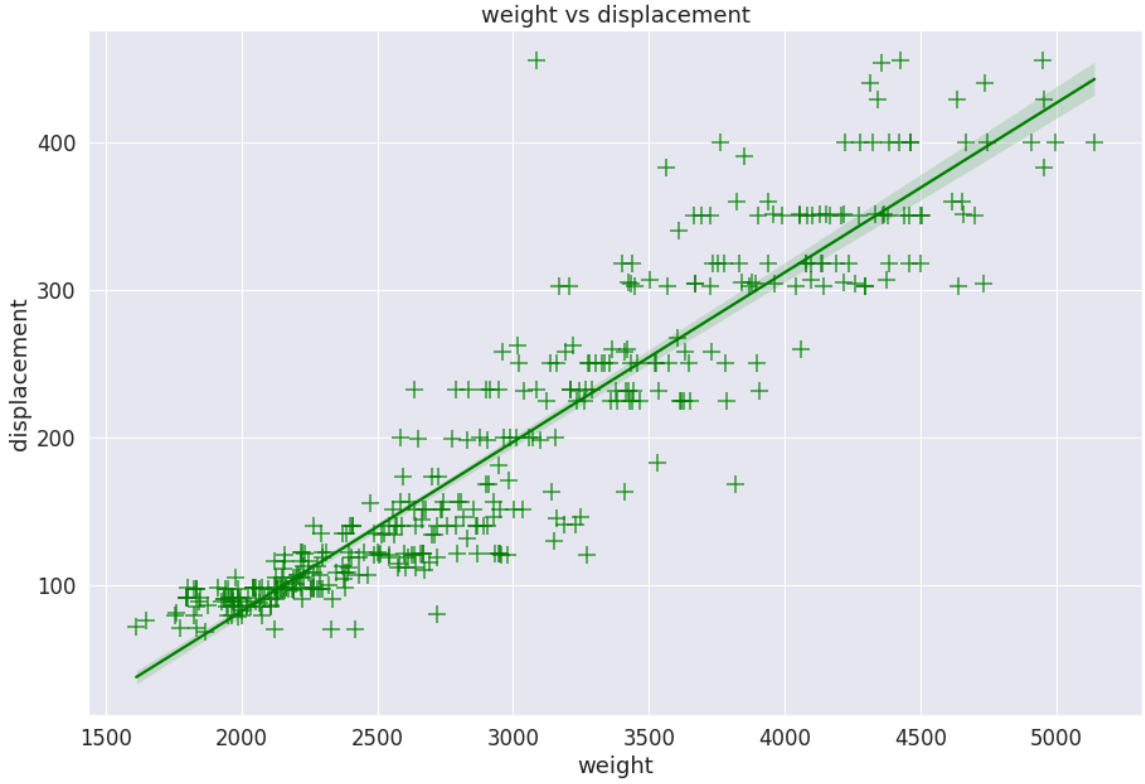
In [ ]:
```
dataset.corr()
```

Out[9]:

	mpg	cylinders	displacement	weight	acceleration	model year	origin
mpg	1.000000	-0.775396	-0.804203	-0.831741	0.420289	0.579267	0.563450
cylinders	-0.775396	1.000000	0.950721	0.896017	-0.505419	-0.348746	-0.562543
displacement	-0.804203	0.950721	1.000000	0.932824	-0.543684	-0.370164	-0.609409
weight	-0.831741	0.896017	0.932824	1.000000	-0.417457	-0.306564	-0.581024
acceleration	0.420289	-0.505419	-0.543684	-0.417457	1.000000	0.288137	0.205873
model year	0.579267	-0.348746	-0.370164	-0.306564	0.288137	1.000000	0.180662
origin	0.563450	-0.562543	-0.609409	-0.581024	0.205873	0.180662	1.000000

Let's create a regression plot for attributes with a good correlation

In [ ]:
```
plt.figure(figsize=(15, 10))
sns.set(font_scale=1.5)

ax = sns.regplot(x='weight', y='displacement', data = dataset, color='green'
ax.set(xlabel='weight', ylabel='displacement') # add x- and y-labels
ax.set_title('weight vs displacement') # add title
plt.show()
```

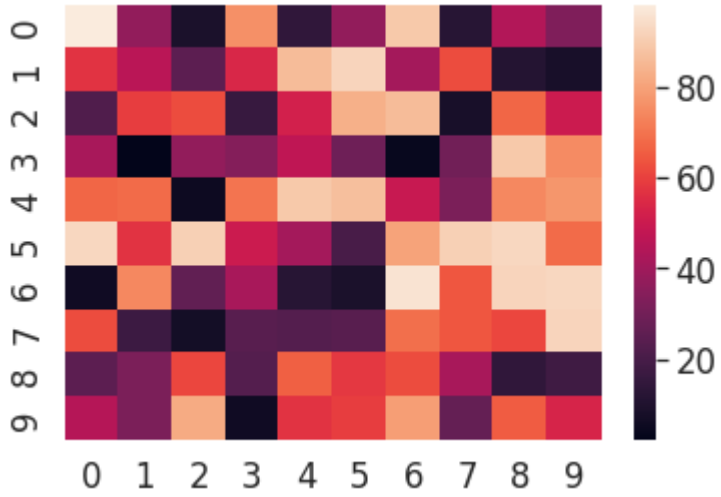


##Heatmaps

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. In this, to represent more common values or higher activities brighter colors basically reddish colors are used and to represent less common or activity values, darker colors are preferred. Heatmap is also defined by the name of the shading matrix. Heatmaps in Seaborn can be plotted by using the seaborn.heatmap() function.

In [ ]:
```
# importing the modules
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# generating 2-D 10x10 matrix of random numbers
# from 1 to 100
data = np.random.randint(low = 1,
                         high = 100,
                         size = (10, 10))
print("The data to be plotted:\n")
print(data)

# plotting the heatmap
hm = sns.heatmap(data = data)

# displaying the plotted heatmap
plt.show()
```

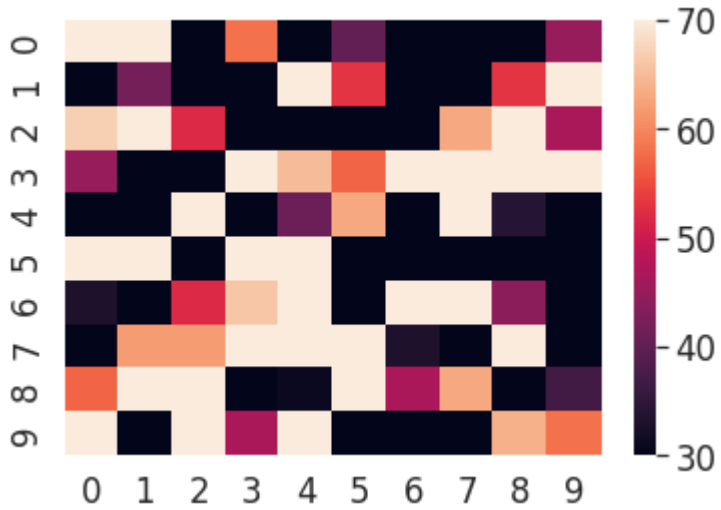
```
The data to be plotted:

[[98 37  9 76 14 37 89 12 44 33]
 [57 46 25 54 86 92 41 62 11  8]
 [22 59 62 16 52 83 86  8 67 50]
 [42  2 37 34 47 29  4 30 89 75]
 [67 68  5 70 89 87 49 32 74 77]
 [93 57 91 50 41 20 80 91 93 68]
 [ 6 74 26 42 12  9 96 64 92 93]
 [62 17  7 24 23 24 69 64 61 92]
 [25 32 61 23 66 58 62 42 14 18]
 [45 32 82  6 57 59 79 27 65 53]]
```



###Anchoring the colormap

If we set the vmin value to 30 and the vmax value to 70, then only the cells with values between 30 and 70 will be displayed. This is called anchoring the colormap.

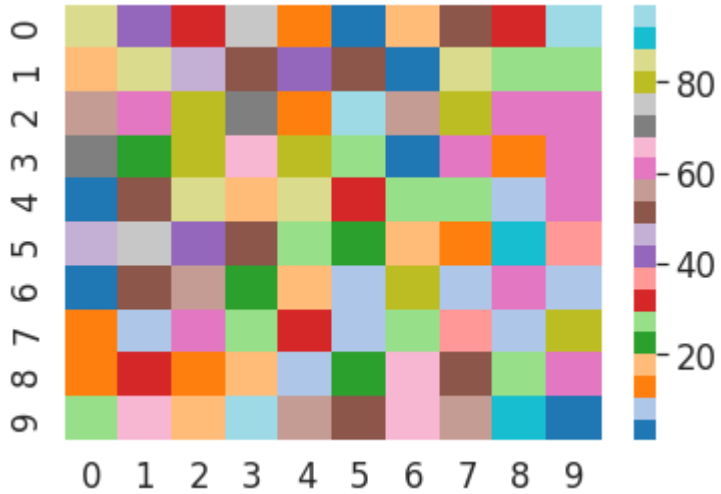
In [ ]:
```
# importing the modules
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt

# generating 2-D 10x10 matrix of random numbers
# from 1 to 100
data = np.random.randint(low=1,
                         high=100,
                         size=(10, 10))

# setting the parameter values
vmin = 30
vmax = 70

# plotting the heatmap
hm = sn.heatmap(data=data,
                vmin=vmin,
                vmax=vmax)

# displaying the plotted heatmap
plt.show()
```



###Choosing the colormap

In this, we will be looking at the cmap parameter. Matplotlib provides us with multiple colormaps, you can look at all of them here https://matplotlib.org/stable/tutorials/colors/colormaps.html (https://matplotlib.org/stable/tutorials/colors/colormaps.html). In our example, we'll be using tab20.

In [ ]:
```
# importing the modules
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt

# generating 2-D 10x10 matrix of random numbers
# from 1 to 100
data = np.random.randint(low=1,
                         high=100,
                         size=(10, 10))

# setting the parameter values
cmap = "tab20"

# plotting the heatmap
hm = sn.heatmap(data=data,
                cmap=cmap)

# displaying the plotted heatmap
plt.show()
```



###Displaying the cell values

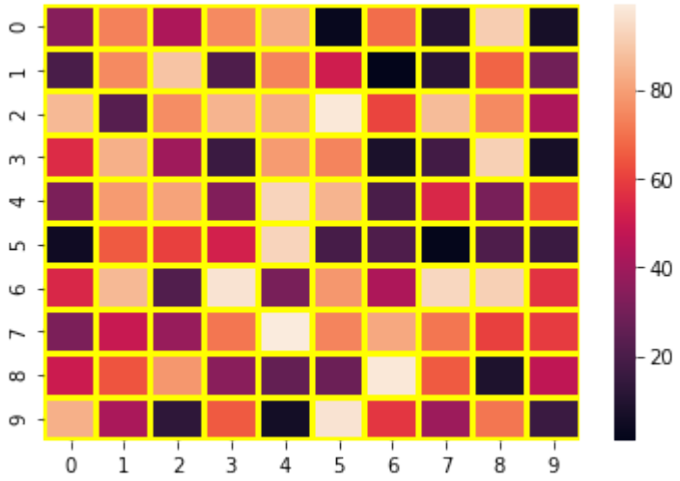
If we want to display the value of the cells, then we pass the parameter annot as True.

In [ ]:
```
# importing the modules
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt

# generating 2-D 10x10 matrix of random numbers
# from 1 to 100
data = np.random.randint(low=1,
                         high=100,
                         size=(10, 10))

# setting the parameter values
annot = True

# plotting the heatmap
hm = sn.heatmap(data=data,
                annot=annot)

# displaying the plotted heatmap
plt.show()
```

###Customizing the separating line

We can change the thickness and the color of the lines separating the cells using the linewidths and linecolor parameters respectively.

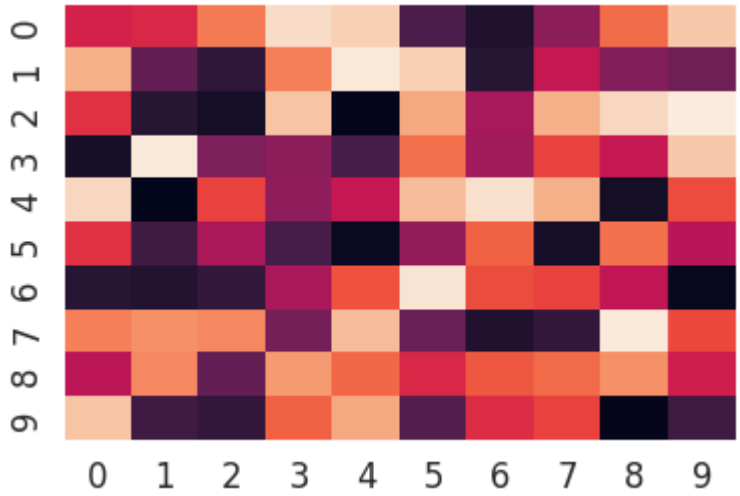
In [ ]:
```
# importing the modules
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt

# generating 2-D 10x10 matrix of random numbers
# from 1 to 100
data = np.random.randint(low=1,
                         high=100,
                         size=(10, 10))

# setting the parameter values
linewidths = 2
linecolor = "yellow"

# plotting the heatmap
hm = sn.heatmap(data=data,
                linewidths=linewidths,
                linecolor=linecolor)

# displaying the plotted heatmap
plt.show()
```



###Hiding the colorbar

We can disable the colorbar by setting the cbar parameter to False.

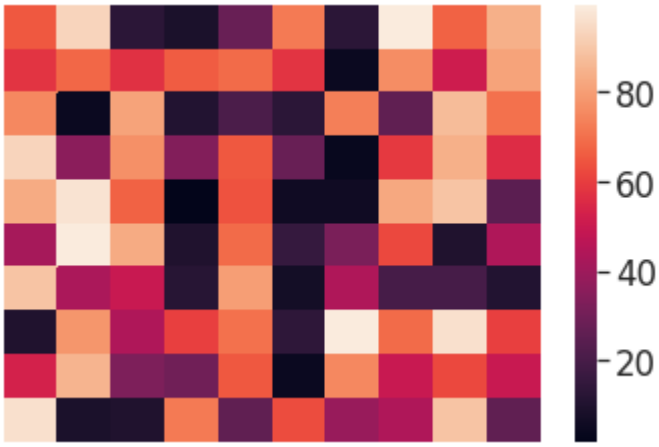
In [ ]:
```
# importing the modules
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt

# generating 2-D 10x10 matrix of random numbers
# from 1 to 100
data = np.random.randint(low=1,
                         high=100,
                         size=(10, 10))

# setting the parameter values
cbar = False

# plotting the heatmap
hm = sn.heatmap(data=data,
                cbar=cbar)

# displaying the plotted heatmap
plt.show()
```



###Removing the labels

We can disable the x-label and the y-label by passing False in the xticklabels and yticklabels parameters respectively.

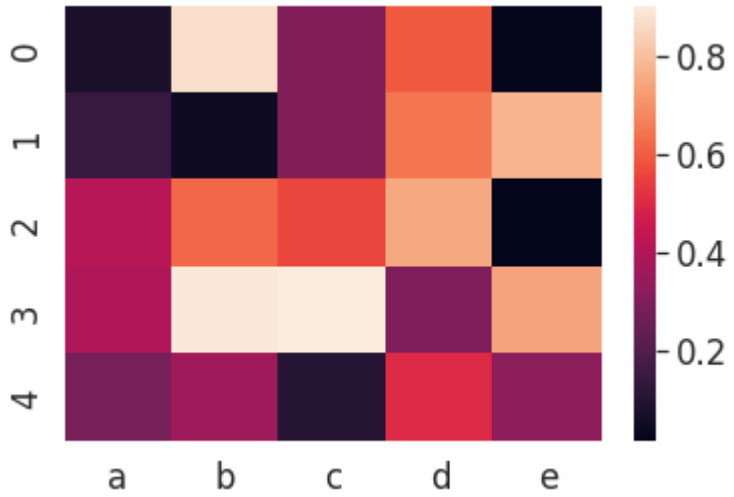
In [ ]:
```
# importing the modules
import numpy as np
import seaborn as sn
import matplotlib.pyplot as plt

# generating 2-D 10x10 matrix of random numbers
# from 1 to 100
data = np.random.randint(low=1,
                         high=100,
                         size=(10, 10))

# setting the parameter values
xticklabels = False
yticklabels = False

# plotting the heatmap
hm = sn.heatmap(data=data,
                xticklabels=xticklabels,
                yticklabels=yticklabels)

# displaying the plotted heatmap
plt.show()
```



###Heatmap from a DataFrame

In [ ]:
```
# Library
import seaborn as sns
import pandas as pd
import numpy as np

# Create a dataset
df = pd.DataFrame(np.random.random((5,5)), columns=["a","b","c","d","e"])
print(df.head())
# Default heatmap
p1 = sns.heatmap(df)
```

```
          a         b         c         d         e
0  0.072480  0.871889  0.308052  0.596718  0.017439
1  0.144703  0.046921  0.308569  0.647216  0.776646
2  0.415223  0.618931  0.562276  0.751926  0.013390
3  0.398378  0.898447  0.902951  0.300957  0.739753
4  0.282393  0.365788  0.101875  0.500152  0.327363
```



#Geospatial Data with Folium

Geospatial data is information that describes objects, events or other features with a location on or near the surface of the earth.

In [ ]:
```
!pip install folium
```

```
Collecting folium
  Downloading folium-0.14.0-py2.py3-none-any.whl (102 kB)
Collecting branca>=0.6.0
  Downloading branca-0.6.0-py3-none-any.whl (24 kB)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-
packages (from folium) (1.19.2)
Requirement already satisfied: jinja2>=2.9 in c:\programdata\anaconda3\lib
\site-packages (from folium) (2.11.2)
Requirement already satisfied: requests in c:\programdata\anaconda3\lib\si
te-packages (from folium) (2.24.0)
Requirement already satisfied: MarkupSafe>=0.23 in c:\programdata\anaconda
3\lib\site-packages (from jinja2>=2.9->folium) (1.1.1)
Requirement already satisfied: chardet<4,>=3.0.2 in c:\programdata\anacond
a3\lib\site-packages (from requests->folium) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
c:\programdata\anaconda3\lib\site-packages (from requests->folium) (1.25.1
1)
Requirement already satisfied: idna<3,>=2.5 in c:\programdata\anaconda3\li
b\site-packages (from requests->folium) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in c:\programdata\anacon
da3\lib\site-packages (from requests->folium) (2020.6.20)
Installing collected packages: branca, folium
Successfully installed branca-0.6.0 folium-0.14.0
```

Generating the world map is straightforward in **Folium**. You simply create a **Folium** *Map* object, and then you display it. What is attractive about **Folium** maps is that they are interactive, so you can zoom into any region of interest despite the initial zoom level.

In [ ]:
```
# define the world map
import folium
world_map = folium.Map()


# display world map
world_map
```

Out[1]:
Make this Notebook Trusted to load map: File -> Trust Notebook

You can customize this default definition of the world map by specifying the centre of your map, and the initial zoom level.

All locations on a map are defined by their respective *Latitude* and *Longitude* values. So you can create a map and pass in a center of *Latitude* and *Longitude* values of **[0, 0]**.

For a defined center, you can also define the initial zoom level into that location when the map is rendered. **The higher the zoom level the more the map is zoomed into the center**.

Let's create a map centered around Canada and play with the zoom level to see how it affects the rendered map.

In [ ]:
```
# define the world map centered around Canada with a low zoom level
world_map = folium.Map(location=[56.130, -106.35], zoom_start=4)

# display world map
world_map
```

Out[4]: Make this Notebook Trusted to load map: File -> Trust Notebook

Let's create the map again with a higher zoom level.

In [ ]:
```
# define the world map centered around Canada with a higher zoom level
world_map = folium.Map(location=[56.130, -106.35], zoom_start=8)

# display world map
world_map
```

Out[5]: Make this Notebook Trusted to load map: File -> Trust Notebook

As you can see, the higher the zoom level the more the map is zoomed into the given center.

Another cool feature of **Folium** is that you can generate different map styles.

Other types of styles of maps can be found in the list below:

The following tilesets are built-in to Folium. Pass any of the following to the "tiles" keyword:

- "OpenStreetMap"
- "Mapbox Bright" (Limited levels of zoom for free tiles)
- "Mapbox Control Room" (Limited levels of zoom for free tiles)
- "Stamen" (Terrain, Toner, and Watercolor)
- "Cloudmade" (Must pass API key)
- "Mapbox" (Must pass API key)
- "CartoDB" (positron and dark_matter)

  Cloudmade and Mapbox cannot be used without an API key to Folium

##Map with Markers

Let's download and import the data on police department incidents using *pandas* `read_csv()` method.

Download the dataset and read it into a *pandas* dataframe:

In [ ]:
```
import pandas as pd
df_incidents = pd.read_csv('https://cf-courses-data.s3.us.cloud-object-stora
```

Let's take a look at the first five items in our dataset.

In [ ]:
```
df_incidents.head()
```

Out[3]:

	IncidntNum	Category	Descript	DayOfWeek	Date	Time	PdDistrict	Resc
0	120058272	WEAPON LAWS	POSS OF PROHIBITED WEAPON	Friday	01/29/2016 12:00:00 AM	11:00	SOUTHERN	AR BC
1	120058272	WEAPON LAWS	FIREARM, LOADED, IN VEHICLE, POSSESSION OR USE	Friday	01/29/2016 12:00:00 AM	11:00	SOUTHERN	AR BC
2	141059263	WARRANTS	WARRANT ARREST	Monday	04/25/2016 12:00:00 AM	14:59	BAYVIEW	AR BC
3	160013662	NON-CRIMINAL	LOST PROPERTY	Tuesday	01/05/2016 12:00:00 AM	23:50	TENDERLOIN	
4	160002740	NON-CRIMINAL	LOST PROPERTY	Friday	01/01/2016 12:00:00 AM	00:30	MISSION	

So each row consists of 13 features:

1. **IncidntNum**: Incident Number
2. **Category**: Category of crime or incident
3. **Descript**: Description of the crime or incident
4. **DayOfWeek**: The day of week on which the incident occurred
5. **Date**: The Date on which the incident occurred
6. **Time**: The time of day on which the incident occurred
7. **PdDistrict**: The police department district
8. **Resolution**: The resolution of the crime in terms whether the perpetrator was arrested or not
9. **Address**: The closest address to where the incident took place
10. **X**: The longitude value of the crime location
11. **Y**: The latitude value of the crime location
12. **Location**: A tuple of the latitude and the longitude values
13. **PdId**: The police department ID

Let's find out how many entries there are in our dataset.

In [ ]:
```
df_incidents.shape
```

Out[4]:
```
(150500, 13)
```

So the dataframe consists of 150,500 crimes, which took place in the year 2016. In order to reduce computational cost, let's just work with the first 100 incidents in this dataset.

In [ ]:
```
# get the first 100 crimes in the df_incidents dataframe
limit = 100
df_incidents = df_incidents.iloc[0:limit, :]
```

Let's confirm that our dataframe now consists only of 100 crimes.

In [ ]:
```
df_incidents.shape
```

Out[6]:
```
(100, 13)
```

Now that we reduced the data a little, let's visualize where these crimes took place in the city of San Francisco. We will use the default style, and we will initialize the zoom level to 12.

In [ ]:
```
# San Francisco latitude and longitude values
latitude = 37.77
longitude = -122.42
```

In [ ]:
```
# create map and display it
sanfran_map = folium.Map(location=[latitude, longitude], zoom_start=12)

# display the map of San Francisco
sanfran_map
```

Out[8]:
Make this Notebook Trusted to load map: File -> Trust Notebook

Now let's superimpose the locations of the crimes onto the map. The way to do that in **Folium** is to create a *feature group* with its own features and style and then add it to the `sanfran_map`.

In [ ]:
```
# instantiate a feature group for the incidents in the dataframe
incidents = folium.map.FeatureGroup()

# loop through the 100 crimes and add each to the incidents feature group
for lat, lng, in zip(df_incidents.Y, df_incidents.X):
    incidents.add_child(
        folium.features.CircleMarker(
            [lat, lng],
            radius=5, # define how big you want the circle markers to be
            color='yellow',
            fill=True,
            fill_color='blue',
            fill_opacity=0.6
        )
    )

# add incidents to map
sanfran_map.add_child(incidents)
```

Out[9]: Make this Notebook Trusted to load map: File -> Trust Notebook

You can also add some pop-up text that would get displayed when you hover over a marker. Let's make each marker display the category of the crime when hovered over.

In [ ]:
```
# instantiate a feature group for the incidents in the dataframe
incidents = folium.map.FeatureGroup()

# loop through the 100 crimes and add each to the incidents feature group
for lat, lng, in zip(df_incidents.Y, df_incidents.X):
    incidents.add_child(
        folium.features.CircleMarker(
            [lat, lng],
            radius=5, # define how big you want the circle markers to be
            color='yellow',
            fill=True,
            fill_color='blue',
            fill_opacity=0.6
        )
    )

# add pop-up text to each marker on the map
latitudes = list(df_incidents.Y)
longitudes = list(df_incidents.X)
labels = list(df_incidents.Category)

for lat,log,label in zip(latitudes,longitudes,labels):
    folium.Marker([lat,log],popup=label).add_to(sanfran_map)

sanfran_map.add_child(incidents)
```

Out[11]: Make this Notebook Trusted to load map: File -> Trust Notebook

##Choropleth Maps

In [ ]:
```
import folium
state_unemp = pd.read_csv("https://raw.githubusercontent.com/Jovita7/Data-Ar
state_geo = "https://raw.githubusercontent.com/Jovita7/Data-Analysis-and-Vis
```

In [ ]:
```
usa_state = folium.Map(location=[48, -102], zoom_start=3)
folium.Choropleth(
    geo_data = state_geo,                  #json
    name ='choropleth',
    data = state_unemp,
    columns = ['State', 'Unemployment'], #columns to work on
    key_on ='feature.id',
    fill_color ='YlGnBu',     #I passed colors Yellow,Green,Blue
    fill_opacity = 0.7,
    line_opacity = 0.2,
   legend_name = "Unemployment scale"
).add_to(usa_state)
usa_state
```

Out[13]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [ ]:
```
#Create a boxplot of the distribuon of
#temperatures in different cies. Take data from
#'temperatures.csv'
import pandas as pd
import matplotlib.pyplot as plt
# Load data from the CSV file
data = pd.read_csv('temperatures.csv')
# Create a boxplot
plt.boxplot(data.drop('Date', axis=1).values)
# Set the labels for the x-axis and y-axis
plt.xticks(range(1, len(data.columns)),
 data.columns[:-1])
plt.ylabel('Temperature (°C)')
# Show the plot
plt.show()
```

In [ ]:
```
import networkx as nx
import matplotlib.pyplot as plt
# Define the Instagram follow data as a dictionary
instagram = {'person1': [0,1,1,0,1], 'person2': [0,0,1,0,1],
 'person3': [1,1,0,1,1],
 'person4': [1,1,1,0,0], 'person5': [1,1,0,0,0]}
# Create a directed graph using the from_dict_of_lists()
#function from the networkx library
G = nx.DiGraph(instagram)
# Draw the graph using the draw()
#function from the matplotlib library
nx.draw(G, with_labels=True)
plt.show()
```

In [ ]:
```
"""You have been given a dataset of car prices and
their respective horsepower, mileage, and
weight. You have been tasked to analyze the
rela□onship between thesevariables and create
a scatter plot to visualize the patterns.
Dataset: The dataset, named "car_data.csv" :
"""

import pandas as pd
import matplotlib.pyplot as plt
# Read in the dataset as a pandas dataframe
df = pd.read_csv("car_data.csv")
# Create a scatter plot of price vs horsepower
plt.scatter(df['Horsepower'], df['Price'])
plt.xlabel('Horsepower')
plt.ylabel('Price')
plt.title('Price vs Horsepower')
# Show the plot
plt.show()
# Create a scatter plot of price vs mileage
plt.scatter(df['Mileage'], df['Price'])
plt.xlabel('Mileage')
plt.ylabel('Price')
plt.title('Price vs Mileage')
# Show the plot
plt.show()
# Create a scatter plot of price vs weight
plt.scatter(df['Weight'], df['Price'])
plt.xlabel('Weight')
plt.ylabel('Price')
plt.title('Price vs Weight')
# Show the plot
plt.show()
```

In [ ]:

```
"""Use the file heights_weights.csv which contains
10000 non-null values for heights and weights.
The Male column shows 1 if the person is a
Male and 0 if the person is a
Female.
1. Convert this file into a pandas Data Frame.
2. Display basic information like memory and data types for this data frame.
3. Display basic statistics like mean, std, quar□les, etc. for this data fra
4. Create a correlation table for the data frame and comment about what kinc
there between Height and Weight.
5. Do Height and Weight contain any outliers? Answer by crea□ng boxplots for
6. Finally, create a sca□er plot of Weight v/s Height with the following spe
(i) use + sign, colour green and size 50 for markers.
(ii) Label X Axis as Weight and Y Axis as Height.
(iii) Display title on top as Weight vs Height
"""

df = pd.read_csv('heights_weights.csv')
# 2. Display basic information about the data frame
print("ans")
print(df.info())
# 3. Display basic statistics for the data frame
print("ans")
print(df.describe())
# 4. Create a correlation table and comment about the
#correlation between Height and Weight
print("ans")
correlation_table = df.corr()
print(correlation_table)
# There is a strong positive correlation between Height and Weight,
#as indicated by the correlation coefficient
#of 0.924756.
# 5. Check for outliers by creating boxplots for Height and Weight
print("ans")
df.boxplot(column=['Height'])
plt.show()
df.boxplot(column=['Weight'])
plt.show()
# 6. Create a scatter plot of Weight vs Height
#with specific markers and labels
print("ans")
plt.scatter(df['Weight'], df['Height'], marker='+',
 color='green', s=50)
plt.xlabel('Weight')
plt.ylabel('Height')
plt.title('Weight vs Height')
plt.show()
```

In [ ]:
```
"""Use the California_Houses.csv file to create a
map with the first 200 rows using the latitudes
and longitudes given in the file with the
following customizations:
1. Colour of circle markers should be green with red fill and the type of ma
terrain
2. Add pop up labels using the popula on from the file."""
import folium
import pandas as pd
# Load the data from the CSV file into a pandas DataFrame
data = pd.read_csv('California_Houses.csv')
# Create a map centered at the mean latitude
# and longitude of the first 200 rows
m = folium.Map(location=[data.iloc[:200]['latitude'].mean(),
 data.iloc[:200]['longitude'].mean()],
 zoom_start=10,
 tiles='Stamen Terrain')
# Add a circle marker for each row in the DataFrame
for index, row in data.iloc[:200].iterrows():
 # Define the location of the circle marker
 location = [row['latitude'], row['longitude']]
 # Define the pop-up label for the circle marker
 popup_label = "Population: " + str(row['population'])
 # Add the circle marker to the map with customizations
 folium.CircleMarker(location=location, radius=5, color='green',
 fill_color='red', popup=popup_label).add_to(m)
# Display the map
m
```

In [ ]:
```
"""The file "student_scores.csv" contains the
marks scored by a group of students in three
subjects: Maths, Science, and English. Each
row contains the name of the student, their
score in Maths, Science, and English. Create a
pandas DataFrame from this data and create a
heatmap to visualize the correlations between
the scores in these three subjects."""

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# load the data into a pandas DataFrame
df = pd.read_csv('student_scores.csv')
# create a correlation matrix
corr = df.corr()
# create a heatmap using seaborn
sns.heatmap(corr, annot=True, cmap='coolwarm')
# add a title
plt.title('Correlation Matrix for Student Scores')
# show the plot
plt.show()
```

In [ ]:
```
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
from pywaffle import Waffle
# Create a DataFrame from the given data
data = pd.DataFrame.from_dict({'USA': 113, 'China': 88, 'Japan': 58,
 'Great Britain': 65, 'ROC': 71,
 'Australia': 46, 'Netherlands': 36,
 'France': 33, 'Germany': 37, 'Italy': 40},
 orient='index', columns=['medal_count'])
# Set up waffle chart parameters
fig = plt.figure(
 FigureClass=Waffle,
 rows=10,
 values=data['medal_count'],
 labels=list(data.index),
 colors=['#3F7FBF', '#DB3236', '#F5A623', '#1EB849', '#AA66CC',
 '#FFD100', '#00A3E0', '#E54028', '#00A651', '#6CABDD'],
 legend={'loc': 'upper left', 'bbox_to_anchor': (1.1, 1)}
)
# Add title
plt.title('2020 Tokyo Olympics Medal Count')
# Show the chart
plt.show()
```

In [ ]:
```
"""You are given a dataset containing customer
reviews of a restaurant. Your task is to create a
wordcloud of the most frequent words used in
the reviews after removing the stopwords"""
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
# Load the dataset
df = pd.read_csv("restaurant_reviews.csv")
# Join all the reviews into a single string
text = " ".join(review for review in df.restaurant_name)
# Create a set of stopwords
stopwords = set(STOPWORDS)
# Add some additional stopwords specific
#to the restaurant domain
stopwords.update(["restaurant", "food", "service", "menu", "meal"])
# Generate a wordcloud image
wordcloud = WordCloud(stopwords=stopwords, max_words=50,
 background_color="white").generate(text)
# Display the generated image
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [ ]:

```
"""You have been hired by a transporta□on
company to analyze their delivery routes. The company has provided you with
represen□ng the adjacency of the ci□es
they deliver to. Each row and column of the matrix represents a city, and th
represents the number of deliveries from city i to city j. Your task is to c
directed graph represen□ng the delivery routes and display it. The graph sho
for each city, and edges should represent delivery routes. The edges should
number of deliveries along that route. Matrix = [ [0, 2, 3, 0, 1], [0, 0, 1,
0, 2, 3], [0, 0, 0, 0, 0] ]
"""

import networkx as nx
import matplotlib.pyplot as plt
Matrix = [
 [0, 2, 3, 0, 1],
 [0, 0, 1, 4, 0],
 [0, 0, 0, 1, 2],
 [0, 0, 0, 2, 3],
 [0, 0, 0, 0, 0]
]
# Create a directed graph
G = nx.DiGraph()
# Add nodes to the graph
for i in range(len(Matrix)):
 G.add_node(i)
# Add edges to the graph with labels
for i in range(len(Matrix)):
 for j in range(len(Matrix)):
 if Matrix[i][j] > 0:
 G.add_edge(i, j, weight=Matrix[i][j])
# Draw the graph with edge labels
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True)
labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
plt.show()
```