CHOOSE A CHART TYPE

An effective chart is one that:

- Conveys the right information without distorting facts.
- Is simple but elegant. It should not force you to think much in order to get it.
- Aesthetics supports information rather that overshadow it.
- Is not overloaded with information.

The list in the next section, sorts the visualizations based on its primary purpose. Primarily, there are 8 types of objectives you may construct plots. So, before you actually make the plot, try and figure what findings and relationships you would like to convey or examine through the visualization. Chances are it will fall under one (or sometimes more) of these 8 categories. Then there are other 8 other types of visualization technique that are also helpful. We will cover them in this chapter.

CHART TYPES AND APPLICATIONS

Here you can find a list of charts categorised by their data visualization functions or by what you want a chart to communicate to an audience. While the allocation of each chart into specific functions isn't a perfect system, it still works as a useful guide for selecting chart based on your analysis or communication needs.

Comparisons	Proportions	Relationships	Distribution
Visualisation methods that help show the differences or similarities between values.	Visualization methods that use size or area to show differences or similarities between values or for parts to a whole.	Visualization methods that show relationships and connections between the data or show correlations between two or more variables.	Visualization methods that display frequency, how data spread out over an interval or is grouped.
With an axis:	Bubble Chart	Heatmap	Box & Whisker Plot
	Bubble Map	Marimekko Chart	Bubble Chart
Bar Chart	Circle Packing	Parallel Coordinates Plot	Density Plot
Box & Whisker Plot	Dot Matrix Chart	Radar Chart	Dot Matrix Chart
Bubble Chart	Nightingale Rose Chart	Venn Diagram	Histogram
Bullet Graph	Proportional Area Chart		Multi-set Bar Chart
Histogram	Stacked Bar Graph		Parallel Sets
Line Graph	Word Cloud		Pictogram Chart
Marimekko Chart			Stem & Leaf Plot
Multi-set Bar Chart			Tally Chart
Nightingale Rose Chart			Timeline
Parallel Coordinates Plot			Violin Plot
Population Pyramid			

Comparisons	Proportions	Relationships	Distribution
Radar Chart Radial Bar Chart Radial Column Chart Span Chart Stacked Area Graph Stacked Bar Graph			
Without an axis Chord Diagram Choropleth Map Donut Chart Dot Matrix Chart Heatmap Parallel Sets	**Proportions in parts-to-a-whole relationships:** Donut Chart Marimekko Chart Parallel Sets Pie Chart Sankey Diagram Stacked Bar Graph Treemap	**For showing connections:** Arc Diagram Brainstorm Chord Diagram Connection Map Network Diagram Non-ribbon Chord Diagram Tree Diagram	**Distribution shown over age and sex in a population** Population Pyramid **Distribution in a body of text** data visualisation Word Cloud
Pictogram Chart Pie Chart Proportional Area Chart Tally Chart Treemap Venn Diagram		**For finding correlations:** Bubble Chart Heatmap Scatterplot	**Distribution shown geographically** Dot Map Connection Map Flow Map

Concepts	Location	Hierarchy	Processes & methods
Help explain and show ideas or concepts.	Show data over geographical regions.	Show how data or objects are ranked and ordered together in an organisation or system.	help explain processes or methods.
Brainstorm Flow Chart Illustration Diagram Venn Diagram	Bubble Map Choropleth Map Connection Map Dot Map Flow Map	Circle Packing Sunburst Diagram Tree Diagram Treemap	Flow Chart Gantt Chart Illustration Diagram Parallel Sets Sankey Diagram

Patterns	Patterns - Continous	Data over time	Range

Patterns	Patterns - Continous	Data over time	Range
Visualization methods that can reveal forms or patterns in the data to give it meaning.		Display the variations between upper and lower limits on a scale.	show data over a time period to find trends or changes over time
Arc Diagram Area Graph Bar Chart Box & Whisker Plot Bubble Chart Candlestick Chart Choropleth Map Connection Map Density Plot Dot Map Dot Matrix Chart Heatmap Histogram Kagi Chart	Line Graph Multi-set Bar Chart Open-high-low-close Chart Parallel Coordinates Plot Point & Figure Chart Population Pyramid Radar Chart Scatterplot Spiral Plot Stacked Area Graph Stream Graph Timeline Violin Plot	Box & Whisker Plot Bullet Graph Candlestick Chart Error Bars Histogram Gantt Chart Kagi Chart Open-high-low-close Chart Span Chart Violin Plot	Area Graph Bubble Chart Candlestick Chart Gantt Chart Heatmap Histogram Line Graph Nightingale Rose Chart Open-high-low-close Chart Spiral Plot Stacked Area Graph Stream Graph
How things work	Movement or flow	Analysing text	Reference tool
Illustrate how an object or system functions.	Visualization methods that are useful for showing movement data or the flow of data.	Visualisation methods that reveal patterns and insights from a body of text.	Visualization methods that can be used as a referencing tool to easily look-up individual data points.
Flow Chart Illustration Diagram Sankey Diagram	Connection Map Flow Map Parallel Sets Sankey Diagram	Word Cloud	**Dates and time:** Calendar Gantt Chart Time Table Timeline **Individual Data Values:**

Patterns	Patterns - Continous	Data over time	Range
			Stem & Leaf Plot

Above table should make our decision of selecting the plots easy. Some plots are part of more than one category indicating we can use those plots for multi-purpose. In the next section, we cover how to plot some of these using MatplotLib and Seaborn. In the next chapter too, we will look at the some of the advanced tools available in Python and learn to draw few more plots.

DATA VISUALIZATION USING MATPLOTLIB

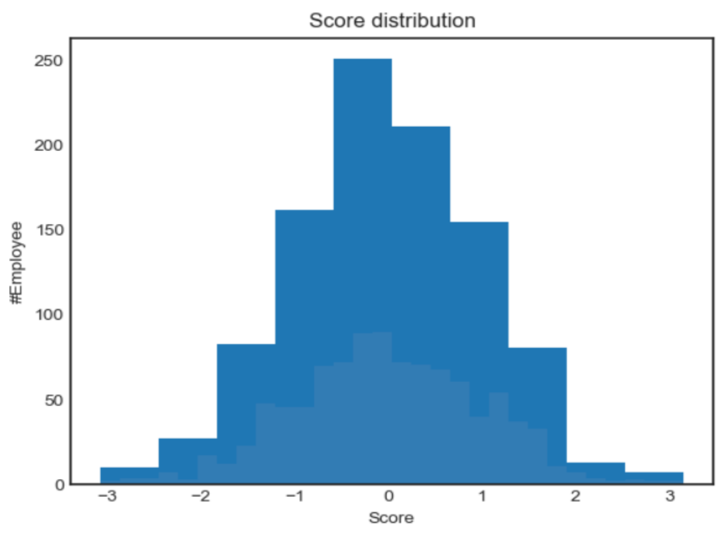
Data scientists are no less than artists. They make paintings in form of digital visualization (of data) with a motive of manifesting the hidden patterns / insights in it. It is even more interesting to know that, the tendency of human perception, cognition and communication increases when he / she gets exposed to visualized form of any content/data. Because of the way the human brain processes information, using charts or graphs to visualize large amounts of complex data is easier than poring over spreadsheets or reports. Data visualization is a quick, easy way to convey concepts in a universal manner – and you can experiment with different scenarios by making slight adjustments.
Data visualization can also:

- Identify areas that need attention or improvement.
- Clarify which factors influence customer behavior.
- Help you understand which products to place where.
- Predict sales volumes.

Package
Matplotlib: Python based plotting library offers matplotlib with a complete 2D support along with limited 3D graphic support. It is useful in producing publication quality figures in interactive environment across platforms. It can also be used for animations as well.

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

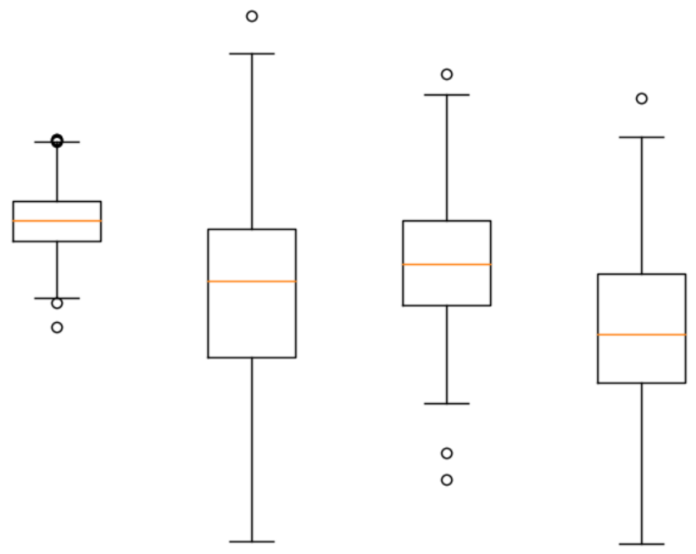
HISTOGRAM



Creating a histogram provides a visual representation of data distribution. Histograms can display a large amount of data and the frequency. The function will calculate and return a frequency distribution. Histograms are useful exploratory data visualizations for spotting outliers, skew, bimodality, and other shape features in the distribution as well as for comparing subgroups in the data.

```
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')

data = np.random.randn(1000)  #Take randon number to generate large
dataset
#Instead you can provide your own data from a file as well
plt.hist(data)
#The hist() function has many options to tune both the calculation
and the display
plt.hist(data, bins=30, alpha=0.5,histtype='stepfilled',
color='steelblue',
         edgecolor='none');
#Labels
plt.title('Score distribution')
plt.xlabel('Score')
plt.ylabel('#Employee')
#Matplotlib plots can be saved as image files
# using the plt.savefig() function.
#supported file formats: eps, pdf, pgf, png, ps, raw, rgba, svg,
svgz
plt. savefig("SavedPlot1.png")
plt.show()
```

BOXPLOT

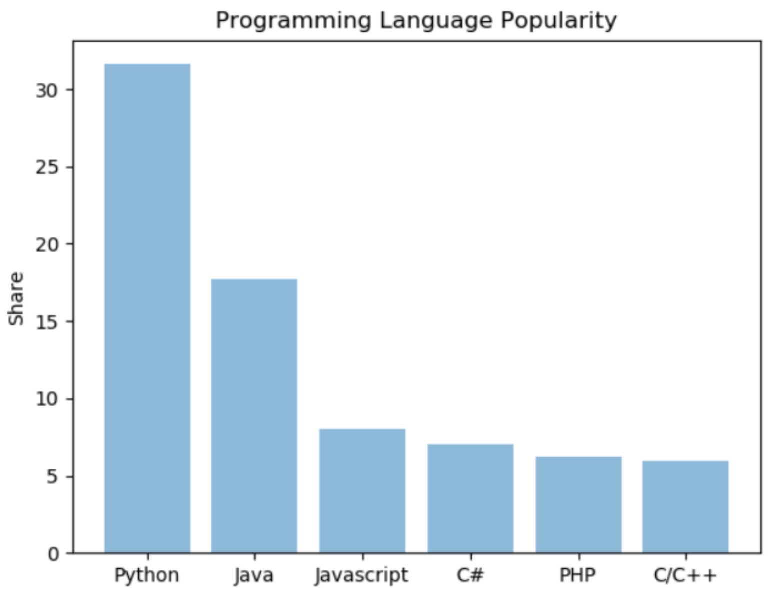


A boxplot is a graph that gives you a good indication of how the values in the data are spread out. Although boxplots may seem primitive in comparison to a histogram or density plot, they have the advantage of taking up less space, which is useful when comparing distributions between many groups or datasets. A box plot which is also known as a whisker plot displays a summary of a set of data containing the minimum, first quartile, median, third quartile, and maximum. In a box plot, we draw a box from the first quartile to the third quartile. A vertical line goes through the box at the median. The whiskers go from each quartile to the minimum or maximum.

```
import numpy as np
import matplotlib.pyplot as plt
#Create 4 sample dataset using random numbers
np.random.seed(10)
collectn_1 = np.random.normal(100, 10, 200)
collectn_2 = np.random.normal(80, 30, 200)
collectn_3 = np.random.normal(90, 20, 200)
```

```
collectn_4 = np.random.normal(70, 25, 200)
fig = plt.figure()
# Create an axes instance
ax = fig.add_axes([0,0,1,1])
# Group the data together as a List
plot_data = [collectn_1, collectn_2, collectn_3, collectn_4]
# Create the boxplot
bp = ax.boxplot(plot_data)

plt. savefig("SavedPlot1.png")
plt.show()
```

BAR CHART



A bar chart illustrates changes over time. But if there is more than one variable, a bar chart can make it easier to compare the data for each variable at each moment in time or multiple variables at a given point of time. For example, a bar chart could compare the popularity of the programming languages based on the usage.

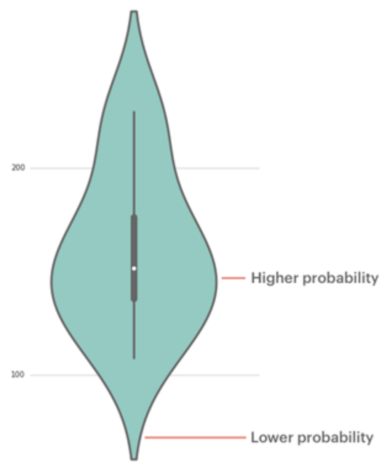
```
import numpy as np
import matplotlib.pyplot as plt

objects = ('Python', 'Java', 'Javascript', 'C#', 'PHP', 'C/C++')
y_pos = np.arange(len(objects))
performance = [31.6,17.7,8,7,6.2,5.9]

plt.bar(y_pos, performance, align='center', alpha=0.5)
plt.xticks(y_pos, objects)
plt.ylabel('Share')
plt.title('Programming Language Popularity')

plt.show()
```

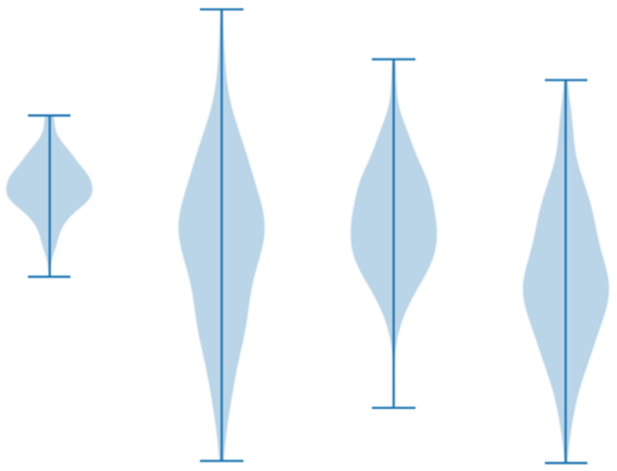
VIOLIN CHART

Violin plots have many of the same summary statistics as box plots:

- the white dot represents the median
- the thick gray bar in the center represents the interquartile range
- the thin gray line represents the rest of the distribution, except for points that are determined to be “outliers” using a method that is a function of the interquartile range.

On each side of the gray line is a kernel density estimation to show the distribution shape of the data. Wider sections of the violin plot represent a higher probability that members of the population will take on the given value; the skinnier sections represent a lower probability.

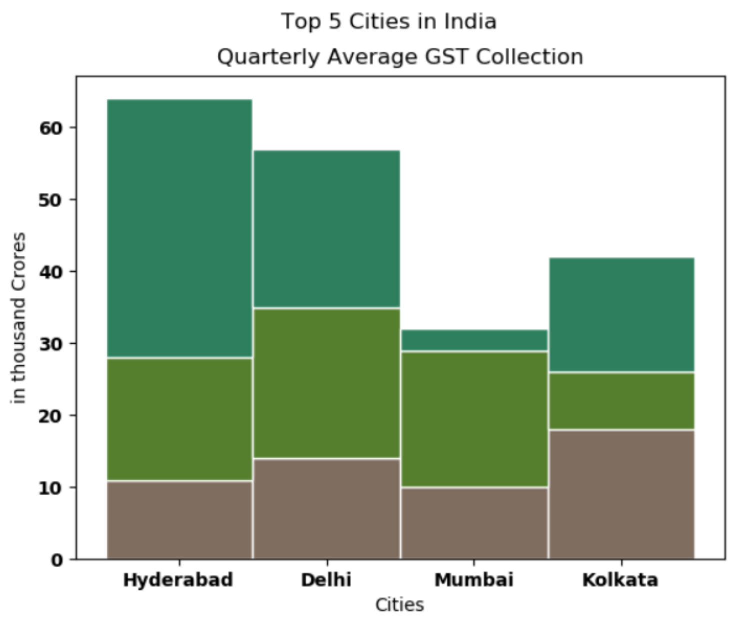




```python
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(10)
data_1 = np.random.normal(100, 10, 200)
data_2 = np.random.normal(80, 30, 200)
data_3 = np.random.normal(90, 20, 200)
data_4 = np.random.normal(70, 25, 200)

## combine these different collections into a list
data_to_plot = [data_1, data_2, data_3, data_4]

# Create a figure instance
fig = plt.figure()

# Create an axes instance
ax = fig.add_axes([0,0,1,1])

# Create the boxplot
bp = ax.violinplot(data_to_plot)
plt.show()
```

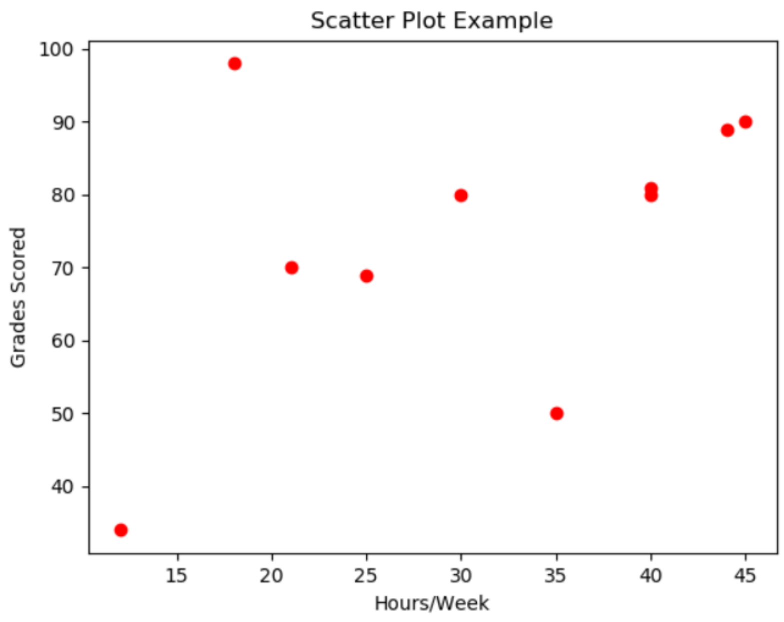
STACKED COLUMN CHART

If you have groups and subgroups, you probably want to display the subgroups values in a grouped barplot or a stacked barplot. In the first case, subgroups are displayed one beside each other, in the second case subgroups are displayed on top of each other. Here is a code showing how to do a stacked barplot. Note that it can easily be turned as a stacked percent barplot.



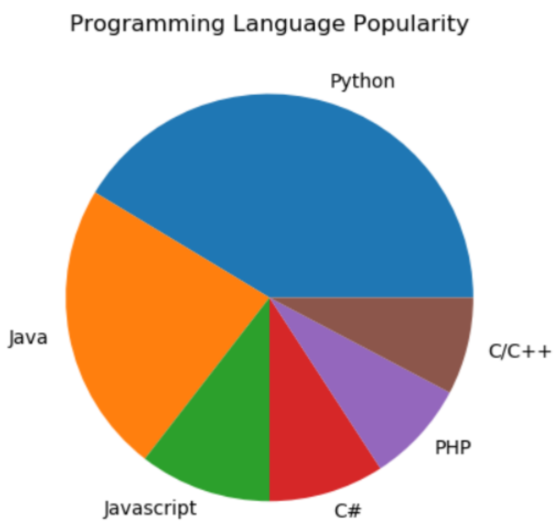
```python
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rc
import pandas as pd

# y-axis in bold
rc('font', weight='bold')

# Values of each group
bars1 = [11, 14, 10, 18]
bars2 = [17, 21, 19, 8]
bars3 = [36, 22, 3, 16]

# Heights of bars1 + bars2
bars = np.add(bars1, bars2).tolist()

# The position of the bars on the x-axis
r = [0, 1, 2, 3]

# Names of group and bar width
names = ['Hyderabad', 'Delhi', 'Mumbai', 'Kolkata', 'Chennai']
barWidth = 1

# Create brown bars
plt.bar(r, bars1, color='#7f6d5f', edgecolor='white',
width=barWidth)
# Create green bars (middle), on top of the firs ones
plt.bar(r, bars2, bottom=bars1, color='#557f2d', edgecolor='white',
width=barWidth)
# Create green bars (top)
plt.bar(r, bars3, bottom=bars, color='#2d7f5e', edgecolor='white',
width=barWidth)

# Custom X axis
plt.xticks(r, names, fontweight='bold')
plt.xlabel("Cities")
plt.ylabel("in thousand Crores")
plt.title("Quarterly Average GST Collection")
plt.suptitle("Top 5 Cities in India")

# Show graphic
plt.show()
```

SCATTER PLOT



Scatter plots have the ability to show trends, clusters, patterns, and relationships in a cloud of data points—especially a very large one. Note: It's important to remember that correlation does not always equal causation, and other unnoticed variables could be influencing the data in a chart.

```
import matplotlib.pyplot as plt
x_hours_study = [44, 25, 35, 40, 18, 40, 45, 21, 30, 12]
y_Student_grades = [89, 69, 50, 81, 98, 80, 90, 70, 80, 34]

plt.scatter(x_hours_study, y_Student_grades, color='r')
plt.title('Scatter Plot Example')
plt.xlabel('Hours/Week')
plt.ylabel('Grades Scored')
plt.show()
```

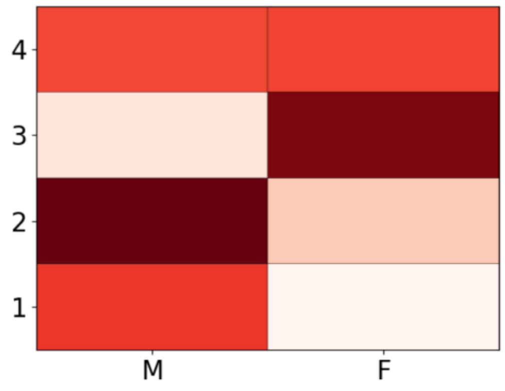


PIE CHART

Pie charts are typically used to tell a story about the parts-to-whole aspect of a set of data. Pie charts are one of the most overused graphs in the world and in most cases is not the best way to present data. They often distort the information and make it more difficult for decision-makers to understand the messages they contain.

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

objects = ('Python', 'Java', 'Javascript', 'C#', 'PHP', 'C/C++')
y_pos = np.arange(len(objects))
performance = [31.6,17.7,8,7,6.2,5.9]
plt.pie(performance, labels = objects)
plt.title('Programming Language Popularity')

plt.show()
```

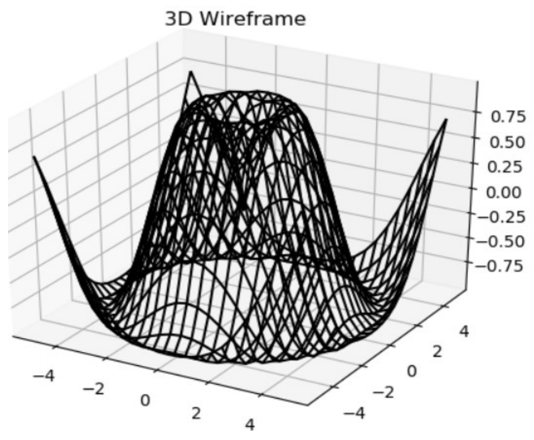
HEAT MAP



A heat map (or heatmap) is a graphical representation of data where the individual values contained in a matrix are represented as colors. It is a bit like looking a data table from above. It is really useful to display a general view of numerical data, not to extract specific data point but to give rough estimate of how data looks.

```
import matplotlib.pyplot as plt
import numpy as np
#Generate a random number, you can refer your data values also
data = np.random.rand(4,2)
rows = list('1234') #rows categories
columns = list('MF') #column categories
fig,ax=plt.subplots()
#Advance color controls
ax.pcolor(data,cmap=plt.cm.Reds,edgecolors='k')
ax.set_xticks(np.arange(0,2)+0.5)
ax.set_yticks(np.arange(0,4)+0.5)
# Here we position the tick labels for x and y axis
ax.xaxis.tick_bottom()
ax.yaxis.tick_left()
#Values against each labels
ax.set_xticklabels(columns,minor=False,fontsize=20)
ax.set_yticklabels(rows,minor=False,fontsize=20)
plt.show()
```

3D WIREFRAME PLOT



Wireframe plot takes a grid of values and projects it onto the specified three-dimensional surface, and can make the resulting three-dimensional forms quite easy to visualize. The *plot_wireframe*() function is used for the purpose

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt

def f(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))

x = np.linspace(-5, 5, 25)
y = np.linspace(-5, 5, 25)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_wireframe(X, Y, Z, color='black')
ax.set_title('3D Wireframe')
plt. savefig("SavedPlot1.png")
plt.show()
```

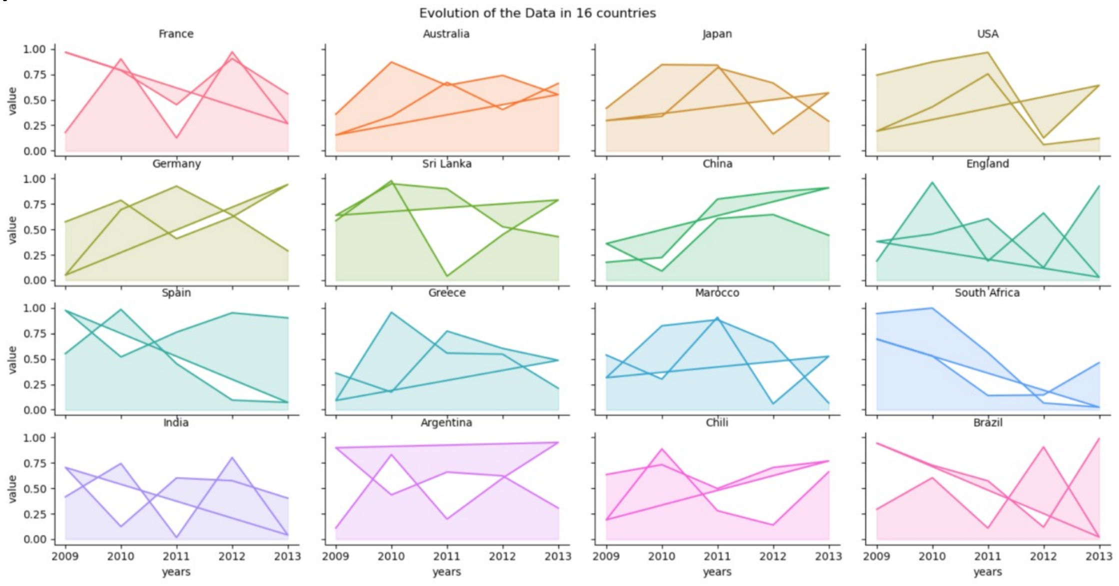
ADVANCED PLOTS TO PRACTICE

AREA GRAPH

We have 2 numerical variables (year and some variable like number of deaths from accident), and a categorical variable (the country). The area charts are created using the fill_between function of matplotlib. The faceting is made using the FacetGrid utility of seaborn.

```
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

# Create a dataset
my_count = ["France", "Australia", "Japan", "USA", "Germany", "Sri
Lanka", "China", "England", "Spain", "Greece", "Marocco",
             "South Africa", "India", "Argentina", "Chili",
"Brazil"]
df = pd.DataFrame({
    "country": np.repeat(my_count, 10),
    "years": [2009, 2010, 2011, 2012,2013]*32 ,
    "value": np.random.rand(160)
})

# Create a grid : initialize it
g = sns.FacetGrid(df, col='country', hue='country', col_wrap=4, )
# Add the line over the area with the plot function
g = g.map(plt.plot, 'years', 'value')

# Fill the area with fill_between
g = g.map(plt.fill_between, 'years', 'value',
alpha=0.2).set_titles("{col_name} country")

# Control the title of each facet
g = g.set_titles("{col_name}")
# Add a title for the whole plo
plt.subplots_adjust(top=0.92)
g = g.fig.suptitle('Evolution of the Data in 16 countries')

plt.show()
```

Output:



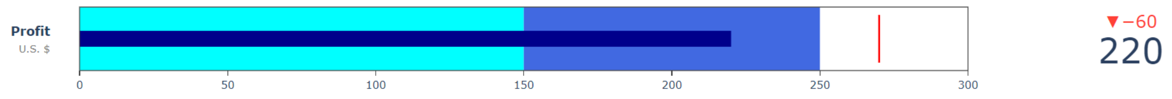
BULLET GRAPH

Stephen Few's Bullet Chart was invented to replace dashboard gauges and meters, combining both types of charts into simple bar charts with qualitative bars (steps), quantitative bar (bar) and performance line (threshold); all into one simple layout. Steps typically are broken into several values, which are defined with an array. The bar represent the actual value that a particular variable reached, and the threshold usually indicate a goal point relative to the value achieved by the bar.

Single Value:

```
import plotly.graph_objects as go

fig = go.Figure(go.Indicator(
    mode = "number+gauge+delta", value = 220,
    domain = {'x': [0, 1], 'y': [0, 1]},
    delta = {'reference': 280, 'position': "top"},
    title = {'text':"<b>Profit</b><br><span style='color: gray; font-
size:0.8em'>U.S. $</span>", 'font': {"size": 14}},
    gauge = {
        'shape': "bullet",
        'axis': {'range': [None, 300]},
        'threshold': {
            'line': {'color': "red", 'width': 2},
            'thickness': 0.75, 'value': 270},
        'bgcolor': "white",
        'steps': [
            {'range': [0, 150], 'color': "cyan"},
            {'range': [150, 250], 'color': "royalblue"}],
        'bar': {'color': "darkblue"}}))
fig.update_layout(height = 250)
fig.show()
```

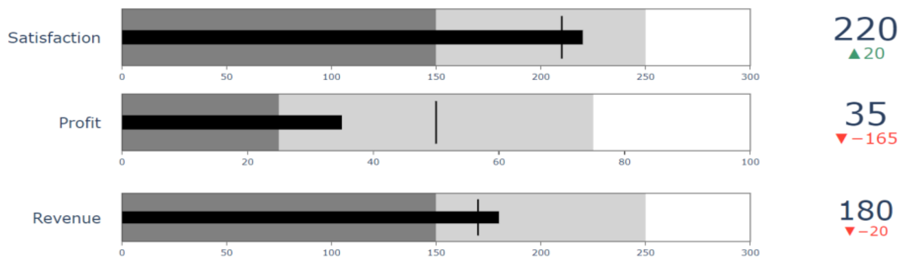
Output:



MultiValue on same graph

```
import plotly.graph_objects as go

fig = go.Figure()
fig.add_trace(go.Indicator(
    mode = "number+gauge+delta", value = 180,
    delta = {'reference': 200},
    domain = {'x': [0.25, 1], 'y': [0.08, 0.25]},
    title = {'text': "Revenue"},
    gauge = {
        'shape': "bullet",
        'axis': {'range': [None, 300]},
        'threshold': {
            'line': {'color': "black", 'width': 2},
            'thickness': 0.75,
            'value': 170},
        'steps': [
            {'range': [0, 150], 'color': "gray"},
            {'range': [150, 250], 'color': "lightgray"}],
        'bar': {'color': "black"}}))

fig.add_trace(go.Indicator(
    mode = "number+gauge+delta", value = 35,
    delta = {'reference': 200},
    domain = {'x': [0.25, 1], 'y': [0.4, 0.6]},
    title = {'text': "Profit"},
    gauge = {
        'shape': "bullet",
        'axis': {'range': [None, 100]},
        'threshold': {
            'line': {'color': "black", 'width': 2},
            'thickness': 0.75,
            'value': 50},
        'steps': [
            {'range': [0, 25], 'color': "gray"},
            {'range': [25, 75], 'color': "lightgray"}],
        'bar': {'color': "black"}}))

fig.add_trace(go.Indicator(
    mode = "number+gauge+delta", value = 220,
    delta = {'reference': 200},
    domain = {'x': [0.25, 1], 'y': [0.7, 0.9]},
    title = {'text' :"Satisfaction"},
    gauge = {
        'shape': "bullet",
        'axis': {'range': [None, 300]},
        'threshold': {
            'line': {'color': "black", 'width': 2},
            'thickness': 0.75,
            'value': 210},
        'steps': [
            {'range': [0, 150], 'color': "gray"},
            {'range': [150, 250], 'color': "lightgray"}],
        'bar': {'color': "black"}}))
fig.update_layout(height = 400 , margin = {'t':0, 'b':0, 'l':0})

fig.show()
```

Output:

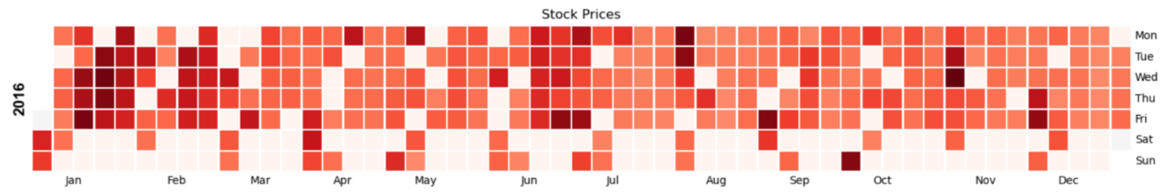


CALENDAR

Using calendar, we can highlight the data on a calendar for a year. We will use stock_price dataset to highlight the reservation_status_date on the plot.

```
import pandas as pd
import calmap
import matplotlib.pyplot as plt

#Import the dataset
df =
pd.read_csv("https://raw.githubusercontent.com/swapnilsaurav/Dataset
/master/stock_price.csv")
# prepare the data for plotting
df["date"] = pd.to_datetime(df["date"])
print(df["date"])
# the data must be a series with a datetime index
df.set_index("date", inplace = True)
x = df[df["year"] == 2016]["Close"]

# plot the data using calmap
calmap.calendarplot(x, fig_kws={'figsize': (16,10)},
yearlabel_kws={'color':'black', 'fontsize':14},
subplot_kws={'title':'Stock Prices'})
plt.show()
```

Output:



MATPLOTLIB.AXES.AXES.TWINX() IN PYTHON

The Axes Class of MatPlotLib contains most of the figure elements: Axis, Tick, Line2D, Text, Polygon, etc., and sets the coordinate system. And the instances of Axes supports callbacks through callbacks attribute.

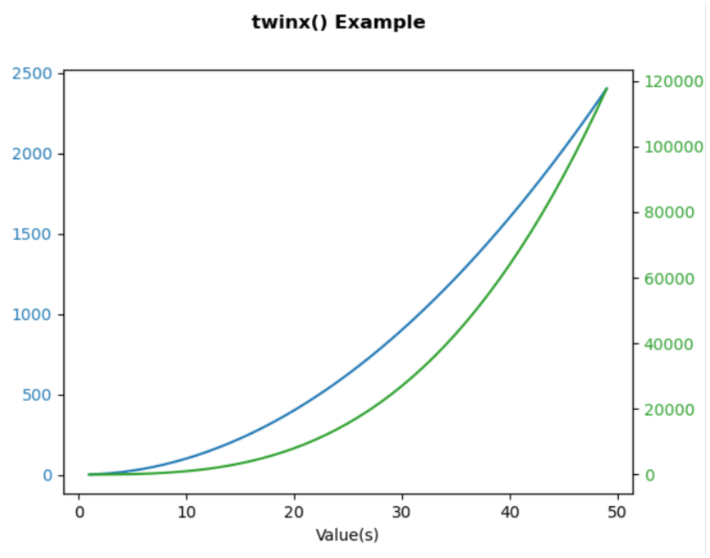
matplotlib.axes.Axes.twinx()

The Axes.twinx() function in axes module of matplotlib library is used to create a twin Axes sharing the xaxis.

Example:

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

# Create some mock data
t = np.arange(1, 50)
data1 = t**2
data2 = t**3

fig, ax1 = plt.subplots()
color = 'tab:blue'
ax1.set_xlabel('Value(s)')
ax1.set_ylabel('Square Value', color = color)
ax1.plot(t, data1, color = color)
ax1.tick_params(axis ='y', labelcolor = color)

ax2 = ax1.twinx()
color = 'tab:green'
ax2.set_ylabel('Cube', color = color)
ax2.plot(t, data2, color = color)
ax2.tick_params(axis ='y', labelcolor = color)

fig.suptitle('twinx() Example\n\n', fontweight ="bold")
plt.show()
```

Output:



WORD CLOUD

Let’s create a word cloud by reading a website.

```
import nltk
from urllib.request import urlopen
from bs4 import BeautifulSoup
#Step 1 - Read in the data
url = "https://storymirror.com/read/story/english/0hlplnpj/ojass-helps-
little-people"
html = urlopen(url).read()
print(html)
#Step 2 - Extract just the Text from the webpage
soup = BeautifulSoup(html)
print(soup)
# kill all script and style elements
for script in soup(["script", "style"]):
    script.extract()    # rip it out

print(soup)
#Step 3 - Extract plain text and remove whitespacing
text = soup.get_text()
print(text)
# break into lines and remove leading and trailing space on each
lines = (line.strip() for line in text.splitlines())
# break multi-headlines into a line each
chunks = (phrase.strip() for line in lines for phrase in line.split("
"))
# drop blank lines
text = 'n'.join(chunk for chunk in chunks if chunk)

print(text)

#Step 4 - Remove stop words, tokenise and convert to lower case
#download and print the stop words for the English language
from nltk.corpus import stopwords
#nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
print(stop_words)
```

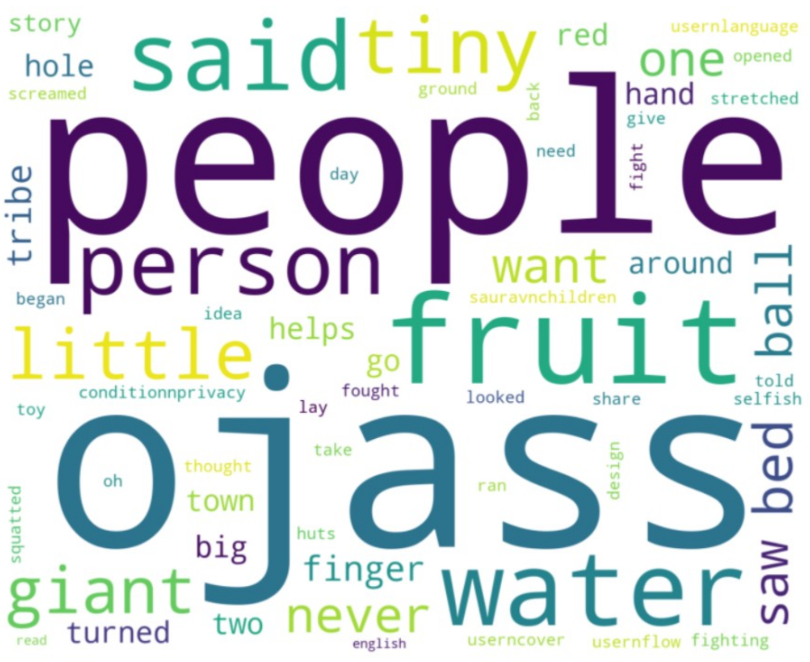
```
#tokenise the data set
from nltk.tokenize import sent_tokenize, word_tokenize
words = word_tokenize(text)
print(words)
# removes punctuation and numbers
wordsFiltered = [word.lower() for word in words if word.isalpha()]
print(wordsFiltered)

# remove stop words from tokenised data set
filtered_words = [word for word in wordsFiltered if word not in
stopwords.words('english')]
print(filtered_words)

#Step 5 - Create the Word Cloud
from wordcloud import WordCloud
import matplotlib.pyplot as plt
wc = WordCloud(max_words=1000, margin=10, background_color='white',
scale=3, relative_scaling = 0.5, width=500, height=400,
random_state=1).generate(' '.join(filtered_words))
plt.figure(figsize=(20,10))
plt.imshow(wc)
plt.axis("off")
plt.show()
```

Note: If you are using Python version higher than 3.7, worldcloud is not available. You will have to install it using manual process. Look for the version compatible with your system from here: https://www.lfd.uci.edu/~gohlke/pythonlibs/#wordcloud

Open your command prompt and change directory to the folder you have saved the downloaded file. Now install using following command:

```
cmd> python -m pip install <filename>
```

Output:



Best Online Resource on this topic:
There is a very good post on Kaggle by user Python10PM, he has created 80 plots using Python, must read for anyone who wants to expand their knowledge on the subject:

https://www.kaggle.com/python10pm/plotting-with-python-learn-80-plots-step-by-step