DATA VISUALIZATION IN PYTHON

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.

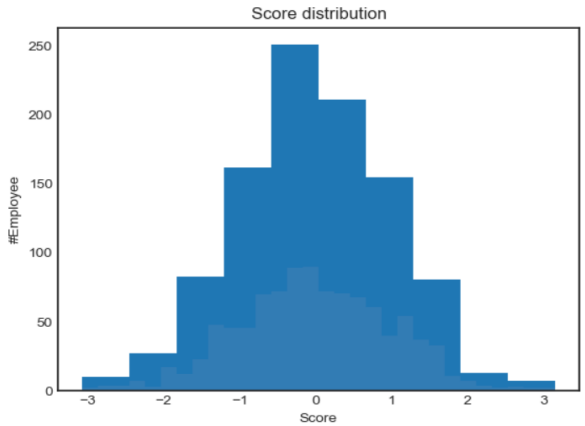
Note: This topic is to introduce you to the topic of visualization. You can refer our book **"Learn and Practice Data Visualization using Python by Swapnil Saurav"** for complete course on visualization. The book contains lot of programs and also different case studies to show how visualization is being used in an effective manner.

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

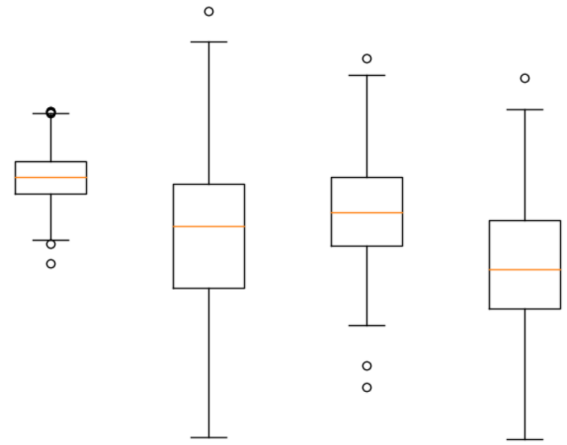


Creating a histogram provides a visual representation of data distribution. Histograms can display a large amount of data and the frequency. The function will calculate and return a frequency distribution. Histograms are useful exploratory data visualizations for spotting outliers, skew, bimodality,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

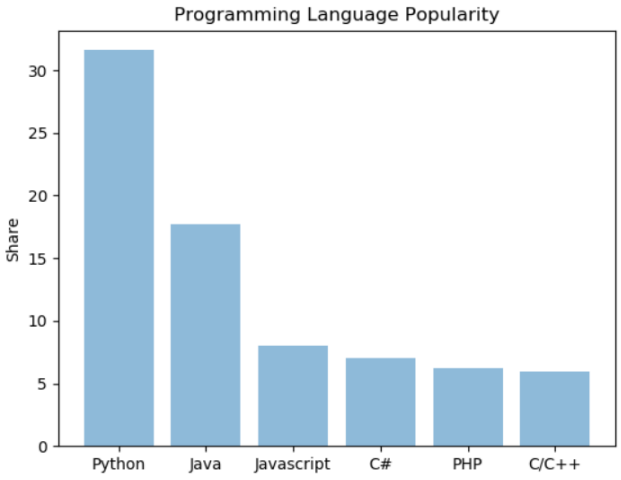


A boxplot is a graph that gives you a good indication of how the values in the data are spread out. Although boxplots may seem primitive in comparison to a histogram or density plot, they have the advantage of taking up less space, which is useful when comparing distributions between many groups or datasets. 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.

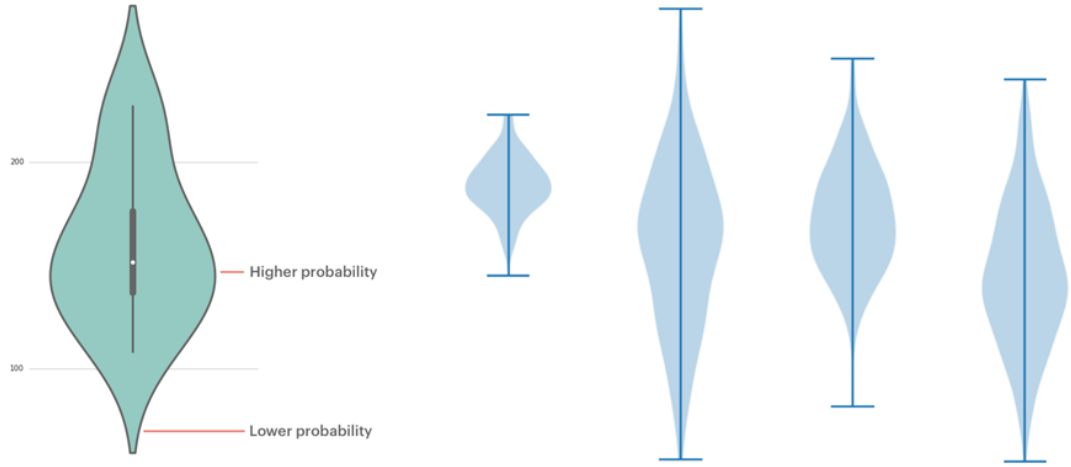
```
import numpy as np
import matplotlib.pyplot as plt

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:

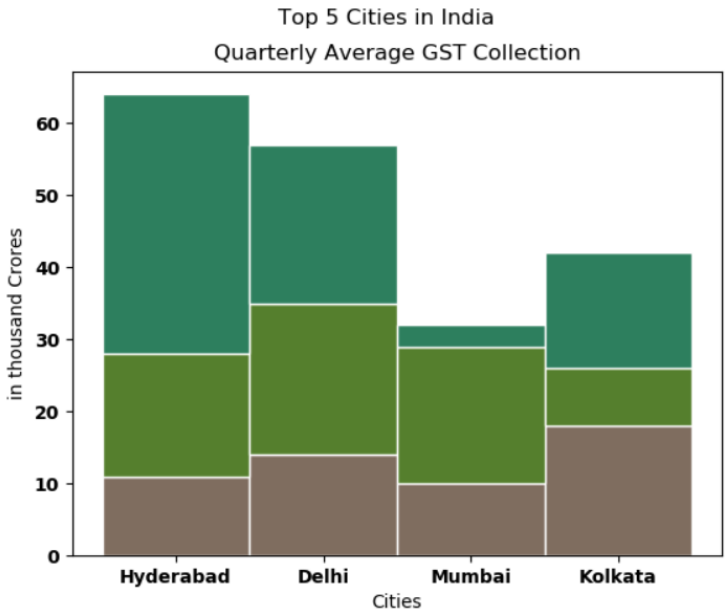
- the white dot represents the median
- the thick gray bar in the center represents the interquartile range
- the thin gray line represents the rest of the distribution, except for points that are determined to be “outliers” using a method that is a function of the interquartile range.

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.




```
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

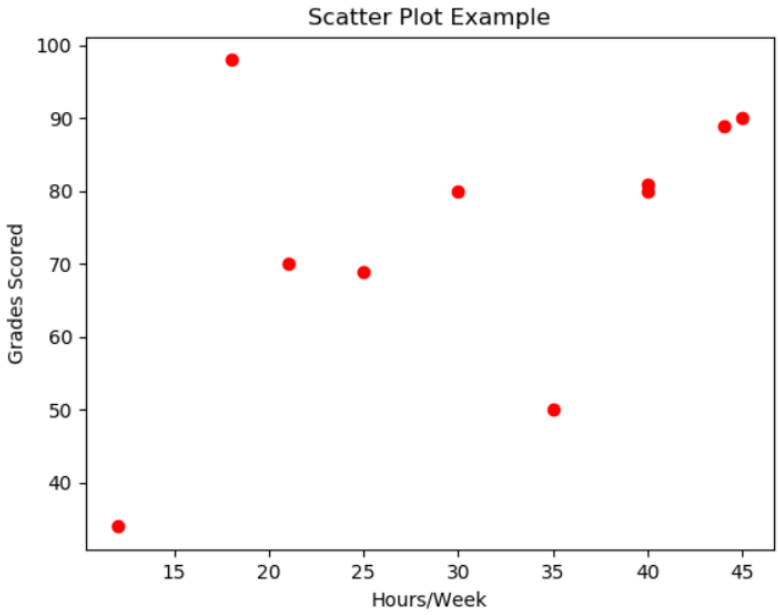


If you have groups and subgroups, you probably want to display the subgroups values in a grouped barplot or a stacked barplot. In the first case, subgroups are displayed one beside each other, in the second case subgroups are displayed on top of each other. Here is a code showing how to do a stacked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

```
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

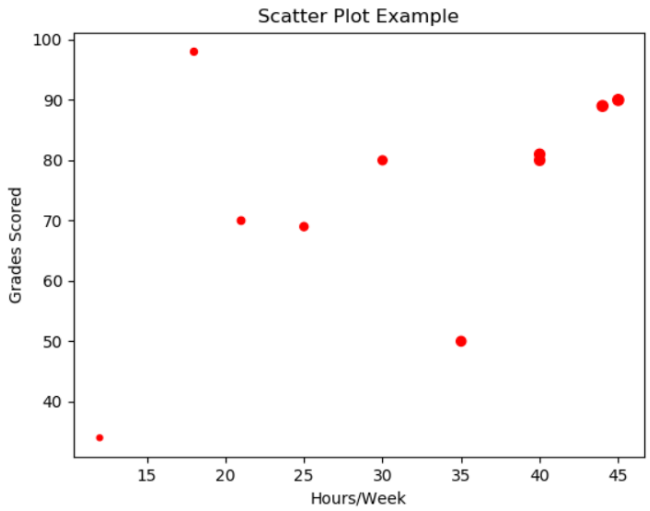


Scatter plots have the ability to show trends, clusters, patterns, and relationships in a cloud of data points—especially a very large one. Note: It's important to remember that correlation does not always equal causation, and other unnoticed variables could be influencing the data in a chart.

```
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()
```

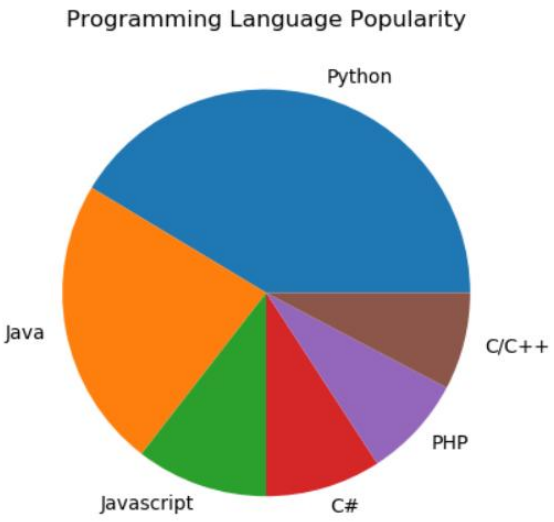
BUBBLE PLOT



While scatterplot chart is ideal for showing the correlation between two values, a bubble chart adds a third dimension to visualize data, either in small multiples or across a few variables. Bubble charts are optimal for showing the relationship between data using the size of the data plot as the third visual element.

```
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, s=(x_hours_study
+ y_Student_grades)*200, color='r')
#Note: same code as Scatter plot,
just that it has an extra parameter
S
# S is used to indicate the size of
the bubble
plt.title('Scatter Plot Example')
plt.xlabel('Hours/Week')
plt.ylabel('Grades Scored')
plt.show()
```

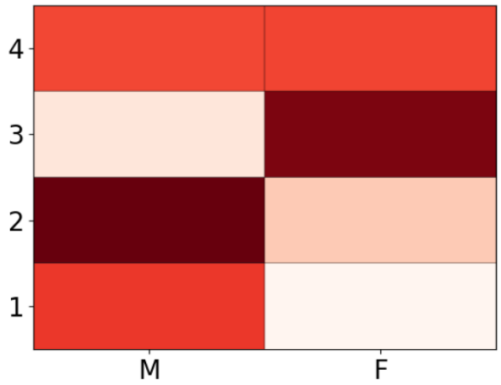
PIE CHART



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()
```