New syllabus 2021-22

Chapter 3

Data Visualization

Informatics Practices

Class XII (As per CBSE Board)

Visit : python.mykvs.in for regular updates

Data visualization

"A picture is worth a thousand words". Most of us are familiar with this expression. Data visualization plays an essential role in the representation of both small and large-scale data. It especially applies when trying to explain the analysis of increasingly large datasets.

Data visualization is the discipline of trying to expose the data to understand it by placing it in a visual context. Its main goal is to distill large datasets into visual graphics to allow for easy understanding of complex relationships within the data.

Several data visualization libraries are available in Python, namely Matplotlib, Seaborn, and Folium etc.

Purpose of Data visualization

- Better analysis
- Quick action
- Identifying patterns
- Finding errors
- Understanding the story
- Exploring business insights
- Grasping the Latest Trends

Plotting library

Matplotlib is the whole python package/ library used to create 2D graphs and plots by using python scripts. pyplot is a module in matplotlib, which supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc. It is used along with NumPy to provide an environment for MatLab.

Pyplot provides the state-machine interface to the plotting library in matplotlib.It means that figures and axes are implicitly and automatically created to achieve the desired plot.For example, calling plot from pyplot will automatically create the necessary figure and axes to achieve the desired plot. Setting a title will then automatically set that title to the current axes object.The pyplot interface is generally preferred for non-interactive plotting (i.e., scripting).

Matplotlib – pyplot features

Following features are provided in matplotlib library for data visualization.

- Drawing – plots can be drawn based on passed data through specific functions.
- Customization – plots can be customized as per requirement after specifying it in the arguments of the functions.Like color, style (dashed, dotted), width; adding label, title, and legend in plots can be customized.
- Saving – After drawing and customization plots can be saved for future use.

How to plot in matplotlib

Steps to plot in matplotlib

- Install matplotlib by pip command - pip install matplotlib in command prompt
- Create a .py & import matplotlib library in it using - import matplotlib.pyplot as plt statement
- Set data points in plot() method of plt object
- Customize plot through changing different parameters
- Call the show() method to display plot
- Save the plot/graph if required

Types of plot using matplotlib

- LINE PLOT
- BAR GRAPH
- HISTOGRAM
- PIE CHART
- FREQUENCY POLYGON
- BOX PLOT
- SCATTER PLOT

Matplotlib –line plot

Line Plot

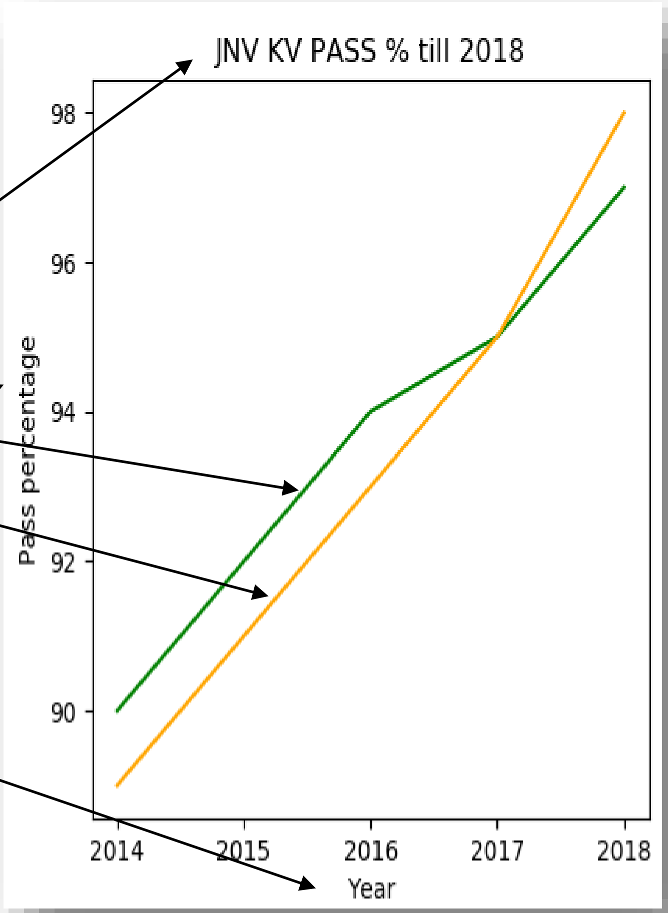
A line plot/chart is a graph that shows the frequency of data occurring along a number line.

The line plot is represented by a series of datapoints connected with a straight line. Generally line plots are used to display trends over time. A line plot or line graph can be created using the plot() function available in pyplot library. We can, not only just plot a line but we can explicitly define the grid, the x and y axis scale and labels, title and display options etc.

Matplotlib –line plot

E.G.PROGRAM

```
import numpy as np
import matplotlib.pyplot as plt
year = [2014,2015,2016,2017,2018]
jnvpasspercentage = [90,92,94,95,97]
kvpasspercentage = [89,91,93,95,98]
plt.plot(year, jnvpasspercentage, color='g')
plt.plot(year, kvpasspercentage, color='orange')
plt.xlabel('Year')
plt.ylabel('Pass percentage')
plt.title('JNV KV PASS % till 2018')
plt.show()
```



Note:- As many lines required call plot() function multiple times with suitable arguments.

Matplotlib –line plot

Line Plot customization

- **Custom line color**
  plt.plot(year, kvpasspercentage, color='orange')
  Change the value in color argument.like 'b' for blue,'r','c',.....
- **Custom line style**
  plt.plot([1,1.1,1,1.1,1], linestyle='-' , linewidth=4).
  set linestyle to any of '-' for solid line style, '--' for dashed, '-.' , ':' for dotted line
- **Custom line width**
  plt.plot('x', 'y', data=df, linewidth=22)
  set linewidth as required
- **Title**
  plt.title('JNV KV PASS % till 2018') – Change it as per requirement
- **Lable** - plt.xlabel('Year') - change x or y label as per requirement
- **Legend -** plt.legend(('jnv','kv'),loc='upper right',frameon=False)
  Change (),loc,frameon property as per requirement

Matplotlib –Bar Graph

Bar Graph

A graph drawn using rectangular bars to show how large each value is. The bars can be horizontal or vertical.

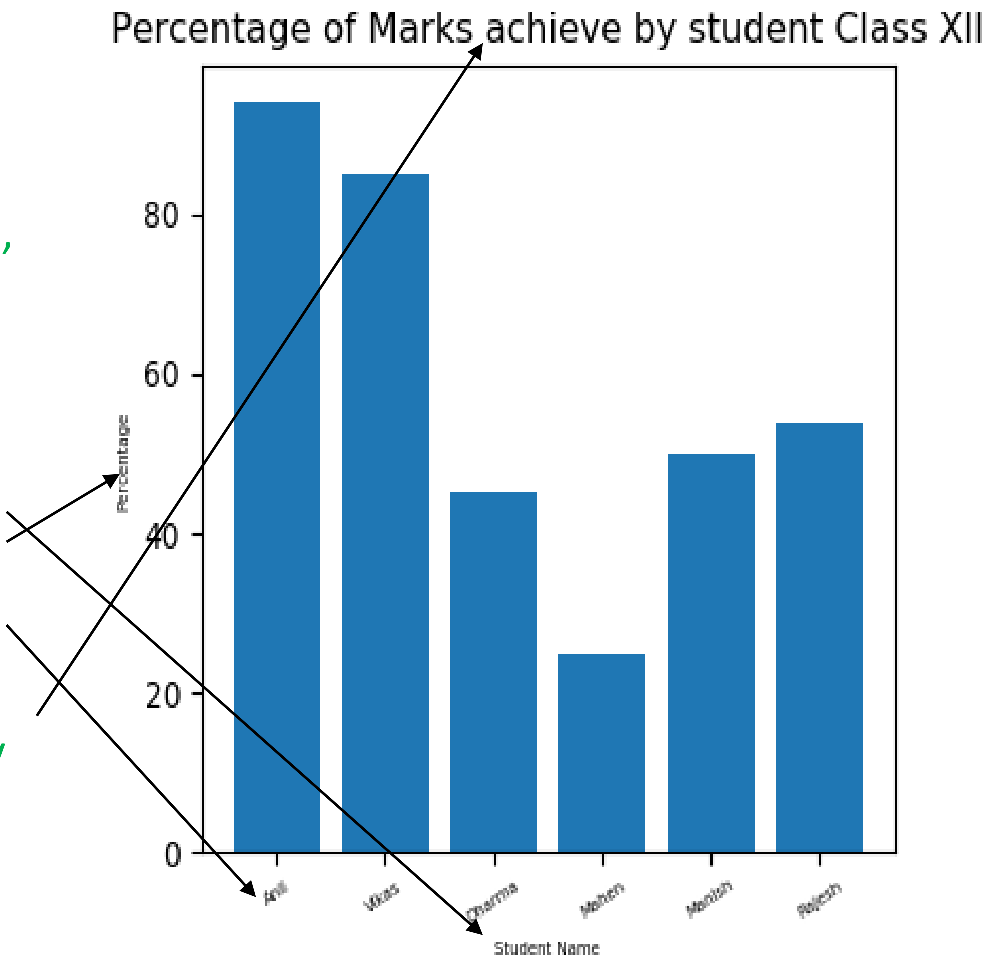
A bar graph makes it easy to compare data between different groups at a glance. Bar graph represents categories on one axis and a discrete value in the other. The goal bar graph is to show the relationship between the two axes. Bar graph can also show big changes in data over time.

Plotting with Pyplot

Plot bar graphs

e.g program

```
import matplotlib.pyplot as plt
import numpy as np
label = ['Anil', 'Vikas', 'Dharma', 'Mahen',
'Manish', 'Rajesh']
per = [94,85,45,25,50,54]
index = np.arange(len(label))
plt.bar(index, per)
plt.xlabel('Student Name', fontsize=5)
plt.ylabel('Percentage', fontsize=5)
plt.xticks(index, label, fontsize=5,
rotation=30)
plt.title('Percentage of Marks achieve by
student Class XII')
plt.show()
#Note – use barh () for horizontal bars
```



Matplotlib –Bar graph

Bar graph customization

- **Custom bar color**

  plt.bar(index, per,color="green",edgecolor="blue")

  Change the value in color,edgecolor argument.like ‘b’ for blue,’r’,’c’,…..
- **Custom line style**

  plt.bar(index, per,color="green",edgecolor="blue",linewidth=4,linestyle='--')

  set linestyle to any of '-‘ for solid line style, '--‘ for dashed, '-.‘ , ':‘ for dotted line
- **Custom line width**

  plt.bar(index, per,color="green",edgecolor="blue",linewidth=4)

  set linewidth as required
- **Title**

  plt.title('Percentage of Marks achieve by student Class XII')

  Change it as per requirement
- **Lable** - plt.xlabel('Student Name', fontsize=5)- change x or y label as per requirement
- **Legend -** plt.legend(('jnv','kv'),loc='upper right‘, frameon=False)

  Change (),loc,frameon property as per requirement

Matplotlib –Histogram

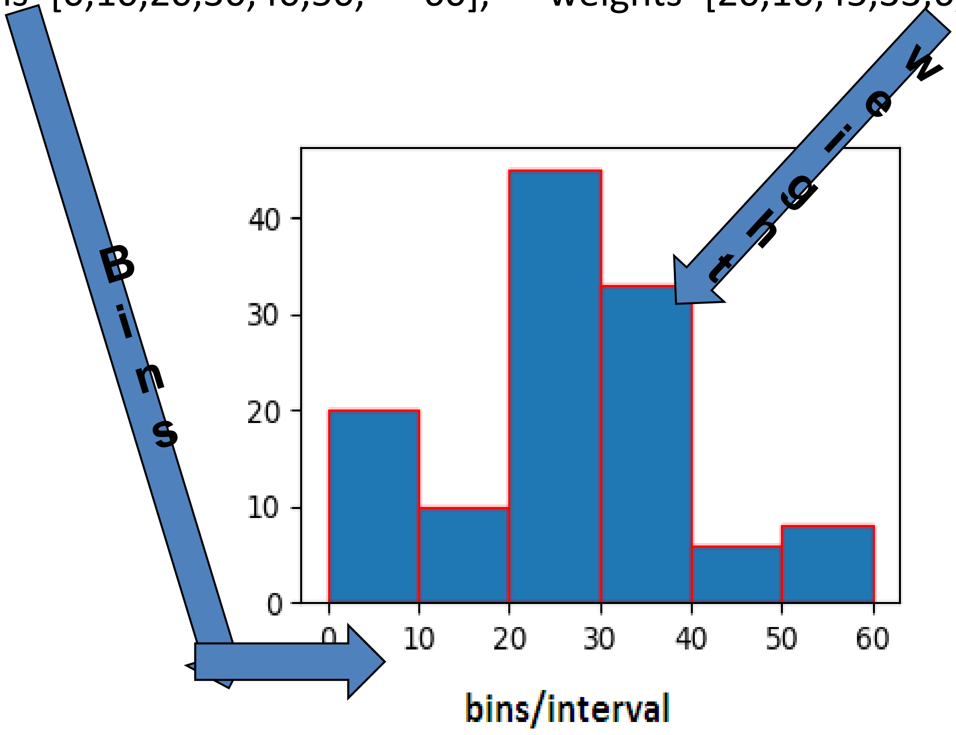
A histogram is a graphical representation which organizes a group of data points into user-specified ranges.
Histogram provides a visual interpretation of numerical data by showing the number of data points that fall within a specified range of values (“bins”). It is similar to a vertical bar graph but without gaps between the bars.

Matplotlib –Histogram

Histogram in Python –

```
import numpy as np
import matplotlib.pyplot as plt
data = [1,11,21,31,41]
plt.hist([5,15,25,35,45,     55],     bins=[0,10,20,30,40,50,     60],     weights=[20,10,45,33,6,8],
edgecolor="red")
plt.show()
```

#first argument of hist() method is position (x,y Coordinate) of weight, where weight is to be displayed. No of coordinates must match with No of weight otherwise error will generate
#Second argument is interval
#Third argument is weight for bars



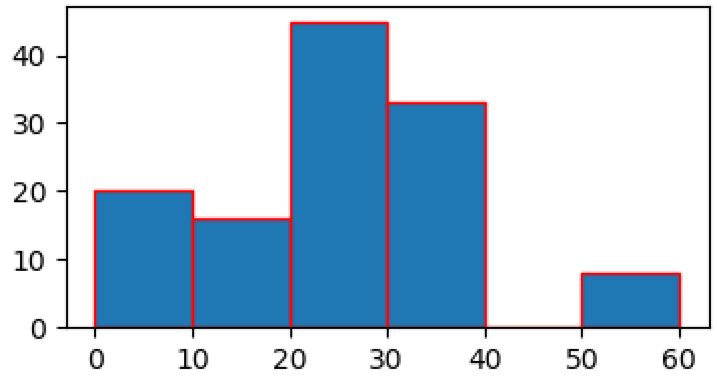
Matplotlib –Histogram

Histogram in Python –

For better understading we develop the same program with minor change .

```
import numpy as np
import matplotlib.pyplot as plt
data = [1,11,21,31,41]
plt.hist([5,15,25,35,15,   55],   bins=[0,10,20,30,40,50,   60],   weights=[20,10,45,33,6,8],
edgecolor="red")
plt.show()
```

# at interval(bin)40 to 50 no bar because we have not mentioned position from 40 to 50 in first argument(list) of hist method. Where as in interval 10 to 20 width is being Displayed as 16 (10+6 both weights are added) because 15 is twice In first argument.



Matplotlib –Histogram

Customization of Histogram –

By default bars of histogram is displayed in blue color but we can change it to other color with following code .

plt.hist([1,11,21,31,41, 51], bins=[0,10,20,30,40,50, 60], weights=[10,1,0,33,6,8], facecolor='y', edgecolor="red")

In above code we are passing 'y' as facecolor means yellow color to be displayed in bars.

To give a name to the histogram write below code before calling show()

plt.title("Histogram Heading")

Edge color and bar color can be set using following parameter in hist() method

edgecolor='#E6E6E6',color='#EE6666 .color value can be rgb in hexadecimal form

For x and y label below code can be written

plt.xlabel('Value')

plt.ylabel('Frequency')

Matplotlib –pie chart

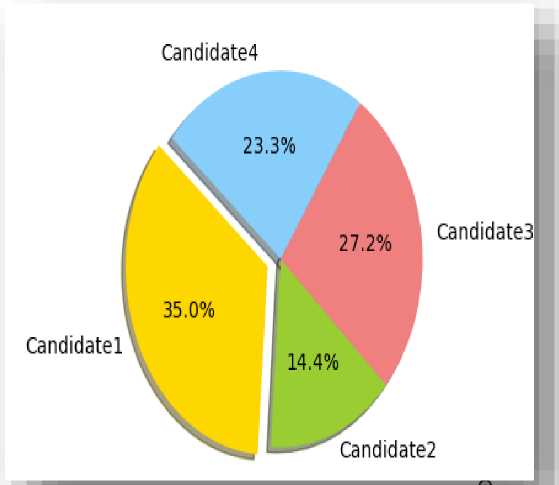
A pie graph/pie chart is a specialized graph used in statistics. The independent variable is plotted around a circle.

Pie Charts shows proportions and percentages between categories, by dividing a circle into proportional segments/parts. Each arc length represents a proportion of each category, while the full circle represents the total sum of all the data, equal to 100%

Matplotlib –pie chart

e.g.program

```
import matplotlib.pyplot as plt
# Data to plot
labels = 'Candidate1', 'Candidate2', 'Candidate3', 'Candidate4'
votes = [315, 130, 245, 210]
sizes=votes
colors = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue']
explode = (0.1, 0, 0, 0)  # explode 1st slice
# Plot
plt.pie(sizes, explode=explode, labels=labels, colors=colors,
      autopct='%1.1f%%', shadow=True, startangle=140)
plt.axis('equal')
plt.show()
```



Matplotlib –Pie chart

Pie chart customization

The pie chart drawn using the Matplotlib.pyplot can be customized of its several aspects.

- The startangle parameter rotates the pie chart by the specified number of degrees. The rotation is counter clock wise and performed on X Axis of the pie chart.
- Shadow effect can be provided using the shadow parameter of the pie() function. Passing True will make a shadow appear below the rim of the pie chart. By default value of shadow is False and there will be no shadow of the pie chart.
- The wedges of the pie chart can be further customized using the wedgeprop parameter. A python dictionary with the name value pairs describing the wedge properties like edgecolor, linewidth can be passed as the wedgeprop argument.
- By setting the frame argument to True, the axes frame is drawn around the pie chart.
- Autopct parameter of the arc() function controls how the percentages are displayed in the wedges. Either format string starting with a % can be specified or a function can be specified.

  e.g., %.1f will display percentage values in the format 25.0, 35.2 and so on.

  %.2f%% will display percentage values in the format 50.25, 75.5 and so on.

Matplotlib – Frequency Polygon

A frequency polygon is a graph constructed by using lines to join the midpoints of each interval, or bin. The heights of the points represent the frequencies. A frequency polygon can be created from the histogram or by calculating the midpoints of the bins from the frequency distribution table.
There is no separate method for creating a frequency polygon in matplotlib.First we have to create a step type histogram.
plt.hist(x,bins=20,histtype='ste[')
Then join midpoint of each set of adjacent bins to create a frequency polygon.

Matplotlib – Frequency Polygon

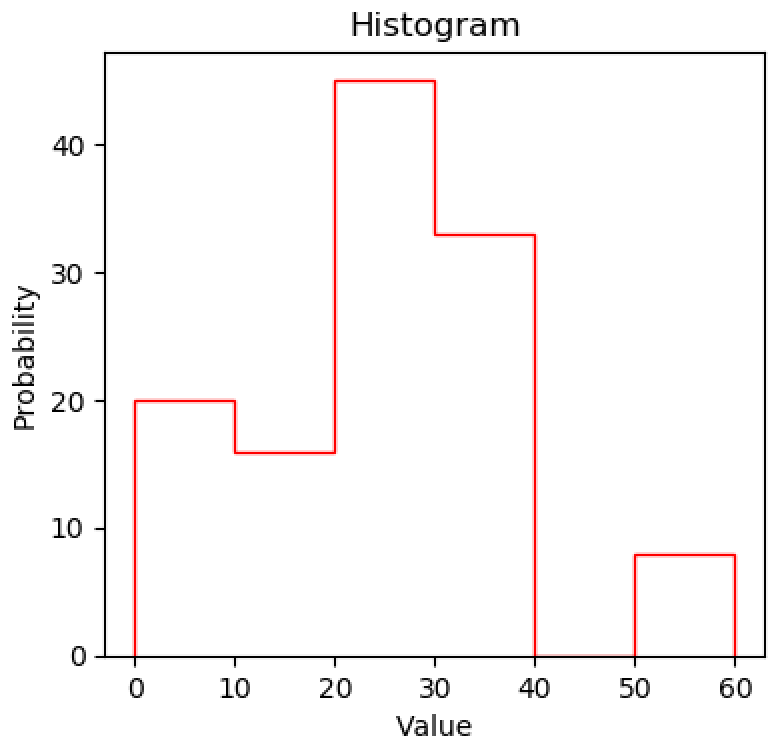
Frequency polygons

If we just connect the top center points of each bins then we obtain relative frequency polygon.

e.g.program

```
import numpy as np
import matplotlib.pyplot as plt
data = [1,11,21,31,41]
plt.hist([5,15,25,35,15, 55],
bins=[0,10,20,30,40,50, 60],
weights=[20,10,45,33,6,8],
edgecolor="red",histtype='step')
#plt.hist(data, bins=20, histtype='step')
plt.xlabel('Value')
plt.ylabel('Probability')
plt.title('Histogram')
plt.show()
```

#Note – it's customization is similar to histogram customization



Matplotlib – box plot

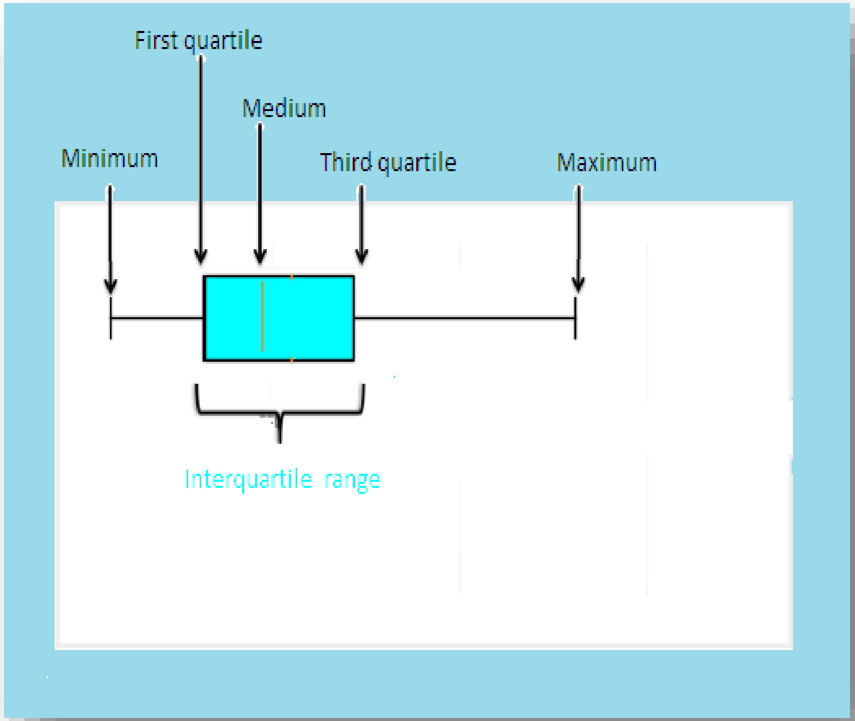
Box Plots

A Box Plot is the visual representation of the statistical five number summary of a given data set.

A Five Number Summary includes:

- Minimum
- First Quartile
- Median (Second Quartile)
- Third Quartile
- Maximum

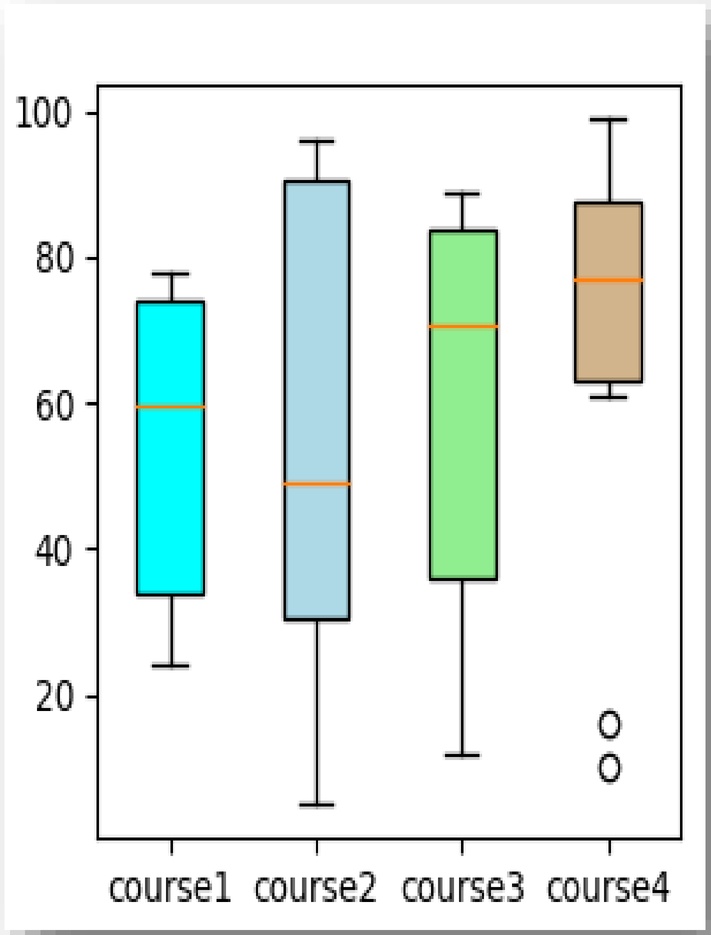
Box plots are useful as they provide a visual summary of the data enabling researchers to quickly identify mean values, the dispersion of the data set, and signs of skewness.



Matplotlib – box plot

e.g.program

```
import matplotlib.pyplot as plt
value1 = [72,76,24,40,57,62,75,78,31,32]
value2=[62,5,91,25,36,32,96,95,30,90]
value3=[23,89,12,78,72,89,25,69,68,86]
value4=[99,73,70,16,81,61,88,98,10,87]
box_plot_data=[value1,value2,value3,value4]
box=plt.boxplot(box_plot_data,vert=1,patch_artist=
True,labels=['course1','course2','course3','course4'],
)
 colors = ['cyan', 'lightblue', 'lightgreen', 'tan']
for patch, color in zip(box['boxes'], colors):
    patch.set_facecolor(color)
plt.show()
```



Note:- if vert=0 in boxplot() is set then horizontal box plots will be drawn

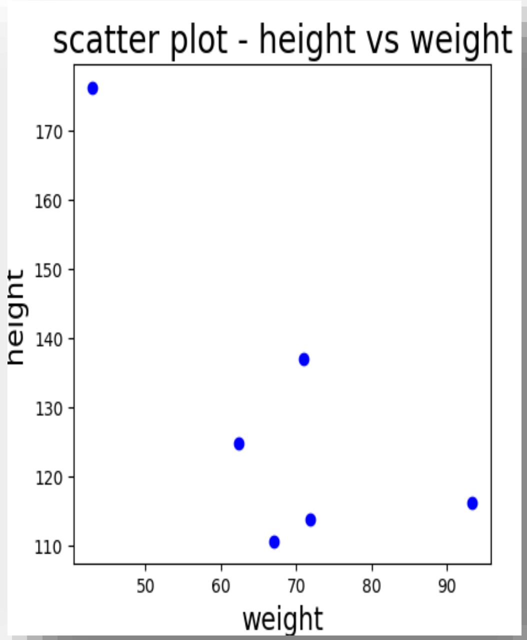
Matplotlib – scatter plot

Scatter plots

A scatter plot is a two-dimensional data visualization that uses dots to represent the values obtained for two different variables - one plotted along the x-axis and the other plotted along the y-axis.

e.g.program

```
import matplotlib.pyplot as plt
 weight1=[93.3,67,62.3,43,71,71.8]
height1=[116.3,110.7,124.8,176.3,137.1,113.9]
 plt.scatter(weight1,height1,c='b',marker='o')
plt.xlabel('weight', fontsize=16)
plt.ylabel('height', fontsize=16)
plt.title('scatter plot - height vs weight',fontsize=20)
plt.show()
```



Matplotlib – scatter plot

Customize Scatter plots

Adjust Color of Scatter Points -Utilize the c argument for the scatter method and set it to green to make the scatter points green.

plt.scatter(weight1,height1,c=‘green',marker='o')

Adjust the Size of Scatter Points -Utilize the s argument in our scatter method and pass in the value 75 to make larger scatter points that are easier to see.

plt.scatter(weight1,height1,c=‘green',marker='o‘,s=75)

Adjust the Transparency of Scatter Points- Utilize the alpha argument in our scatter method and pass in a numeric value between 0 and 1.

plt.scatter(weight1,height1,c=‘green',marker='o‘,s=75,alpha=0.5)

Change marker type- pass value as per requirement like marker=‘-’,marker=‘v’ etc.

Matplotlib – How to save plot

For future use we have to save the plot.To save any plot savefig() method is used.plots can be saved like pdf,svg,png,jpg file formats.

```
plt.savefig('line_plot.pdf')
plt.savefig('line_plot.svg')
plt.savefig('line_plot.png')
```

Parameter for saving plots .e.g.

```
plt.savefig('line_plot.jpg', dpi=300, quality=80, optimize=True,
progressive=True)
```

Which Export Format to Use?

The export as vector-based SVG or PDF files is generally preferred over bitmap-based PNG or JPG files as they are richer formats, usually providing higher quality plots along with smaller file sizes.