

**2020
EDITION**

Textbook for CBSE Class XII

INFORMATICS PRACTICES

With

Python

PREETI ARORA

- ❖ NumPy
- ❖ Python Pandas
- ❖ Data Visualization using Pyplot
- ❖ Software Engineering
- ❖ Django
- ❖ SQL (RDBMS)
- ❖ Python-MySQL Connectivity
- ❖ Society, Law and Ethics

Web Support
at

sultan-chand.com/ws/ipp12

- Presentation on Python Pandas
- Sample Papers • Model Test Papers
- Program Codes • Projects • Practical File
- Django Installation Guide • Viva Voce

SCS

sultan chand

SULTAN CHAND & SONS (P) LTD

Educational Publishers

4859/24, Darya Ganj, New Delhi-110 002

Phones : 4354 6000 (100 Lines), 2324 3939

Fax : (011) 4354 6004, 2325 4295

E-mail : scs@sultanchandebooks.com

Buy books online at: www.sultan-chand.com

ISBN: 978-93-89174-54-0

First Edition 2019

Second Thoroughly Revised Edition 2020

©All rights reserved.

No part of this book may be reproduced or copied in any form or by any means (graphic, electronic or mechanical, including photocopying, recording, taping, or information retrieval system) or reproduced on any disc, tape, perforated media or any other information storage device, etc., without the prior written permission of the publishers. Breach of this condition is liable for legal action. **Anyone who brings information regarding any such reproduction will be handsomely rewarded.**

Publication of **Key** to this book is strictly prohibited.

Every effort has been made to avoid errors or omissions in this publication. In spite of this, some errors might have crept in. Any mistake, error or discrepancy noted may be brought to our notice which shall be taken care of in the next edition. It is notified that neither the publishers nor the author or seller will be responsible for any damage or loss of action to anyone, of any kind, in any manner, therefrom.

For faulty binding, misprints or for missing pages, etc., the publishers' liability is limited to replacement within one month of the purchase by a similar edition. All expenses in this connection are to be borne by the purchaser.

All disputes are subject to Delhi jurisdiction only.

PREFACE

Programming is important for learning to innovate and create eco-friendly solutions to global problems. Programming is also important in our day-to-day life to enhance the power of computers and internet. A significant step towards learning innovative programming solutions is through Python programming which this book has at its core.

This thoroughly revised **Informatics Practices with Python** for Class XII provides an in-depth understanding of the Informatics Practices (065) curriculum and strictly adheres to the guidelines laid down by the CBSE. The book deals with detailed concepts of Python Pandas, NumPy, Data Visualization, Django, Relational Database Management System (SQL), Python-MySQL Connectivity, Basic Software Engineering and Cyber Ethics.

Python is a popular object-oriented language used both for stand-alone programs and scripting applications in a variety of domains. This book adopts a contemporary approach to the most popular Python library, Pandas, with stress on principles of good programming, such as clarity, legibility and efficiency in program design. Thus, an interactive programming style has been emphasized/expressed throughout the book.

The hallmark of this book is that it teaches Python Pandas concepts in detail and usage of several other Python libraries, such as plotting graphs and charts using Python Pyplot, and establishing Python-MySQL Connectivity. Ample case studies to understand the basic concepts of Software Engineering with a student-centred approach have also been provided in the book. With easy-to-understand examples, practical implementations and other tools, the student will learn how to create and implement Python series and dataframes, and develop GUI applications based on it.

The text of the book has been presented in a friendly and easy-to-comprehend language. The book contains example programs that are concise and practical besides diagrams and examples from real-life applications. Each chapter provides tested, debugged and error-free codes with screenshots. Based on the CBSE curriculum, the book has been divided into four units:

Unit I: Data Handling (DH-2) – Chapters 1 to 3

This unit contains three chapters covering fundamentals of Pandas including Series and Dataframes. Advanced concepts of Pandas such as Pivoting, Sorting, Aggregation, Function Applications, Reindexing and Quantiles have been explained in detail with ample examples and associated codes.

Apart from the above, detailed concepts of NumPy (ndarrays) and their implementation, Data Visualization using Pyplot in terms of Line chart, Bar chart, Scatter plot, Histograms, Frequency Polygons and Boxplot have also been thoroughly discussed.

Unit II: Basic Software Engineering (BSE) – Chapters 4 and 5

This unit has been divided into two chapters—4 and 5—and covers Software Engineering concepts such as Software process models, Delivery models, Agile methods, business-use diagrams and some practical aspects and their implementation.

Unit III: Data Management (DM-2) – Chapters 6 to 8

This unit covers development of Django web application, Interface Python with an SQL database and SQL commands, Aggregation functions along with important SQL clauses such as group by, having and order by.

Unit IV: Society, Law and Ethics (SLE-2) – Chapter 9

This unit deals with intellectual property rights, plagiarism, digital rights management, licensing, open source and standards, privacy laws, frauds and cybercrimes such as phishing, illegal downloading, child pornography, cyber scams and frauds, cyber forensics, IT Act, 2000, and Unique ID and Biometrics. It also explains related concepts of technology and society, e-waste management, and gender and disability issues while teaching and using computers and the role of new media in society with case studies.

The book has two appendices containing a Sample Question Paper (Solved) and a Model Test Paper (Unsolved).

As part of our Web Support, Presentation on detailed concepts of Python Pandas, Chapter-wise Program Codes, Projects based on Python-MySQL Connectivity using Pandas, Practical File, Sample Papers, Model Test Papers for practice, Installation Guide to Django and Viva Voce questions are available online and can be accessed at sultan-chand.com/ws/ipp12. Guide to Django Installation is also available in QR Code. Besides, exam-related updates, if any, will be made available online in due course.

I am confident that students and teachers will benefit immensely by making the best use of this book.

Your feedback is important to me. Any suggestions for the improvement of this book will be highly appreciated and duly acknowledged.

My special thanks are due to Ms Rinku Kumari and Ms Payal Bhattacharjee for their valuable suggestions during the course of my writing this book.

Last but not the least, I express my deep gratitude to my esteemed publishers, **Sultan Chand & Sons (P) Ltd**, for their patience, guidance and support.

AUTHOR

CONTENTS

1. NumPy	1.1–1.42
1.1 Introduction	1.1
1.2 What is NumPy	1.1
1.3 Working with NumPy	1.3
1.4 How to Create a NumPy Array	1.4
1.5 Operations on NumPy Array	1.8
1.5.1 Array Slicing	1.8
1.5.2 Joins in Arrays	1.10
1.5.3 Array Subsets	1.12
1.6 Arithmetic Operations on Arrays	1.13
1.7 Applications of NumPy Arrays	1.16
1.7.1 Covariance	1.16
1.7.2 Correlation	1.21
1.7.3 Linear Regression	1.23
2. Data Visualization Using Pyplot	2.1–2.60
2.1 Introduction	2.1
2.2 Matplotlib	2.2
2.3 NumPy	2.3
2.4 Installing Matplotlib	2.3
2.5 Types of Visualization	2.4
2.6 Basic Visualization Rules	2.5
2.7 Basic Nomenclature of a Plot	2.5
2.8 Line Plot/Chart	2.6
2.8.1 Multiple Plots	2.9
2.8.2 Multiple Views	2.10
2.9 Scatter Chart	2.18
2.10 Bar Plot/Chart	2.22
2.11 Histograms	2.25
2.12 Saving Plots to File	2.31
2.13 Frequency Polygons	2.32
2.14 Box Plot	2.34
3. Python Pandas	3.1–3.95
3.1 Introduction	3.1
3.2 Pandas	3.2
3.2.1 Features of Pandas	3.2
3.3 Installing Pandas	3.3
3.4 Data Structures in Pandas	3.5
3.4.1 Series	3.5
3.4.2 Creation of Series	3.6
3.4.3 Creating an Empty Series using Series() Method	3.6
3.4.4 Creating a Series using Series() with Arguments	3.6
3.4.5 Creating a Series from Dictionary	3.13
3.4.6 Creating a Series using a Mathematical Expression/Function	3.15
3.5 Series Object Attributes	3.16
3.5.1 Retrieving Values from a Series using head() and tail() functions	3.16
3.6 Mathematical Operations on Series	3.17
3.7 Vector Operations on Series	3.18
3.8 Retrieving Values Using Conditions	3.18
3.9 Deleting Elements from a Series	3.19
3.10 Dataframes	3.19
3.11 Binary Operations	3.30
3.12 Matching and Broadcasting Operation	3.32
3.13 Missing Data and Filling Values	3.34
3.14 Comparing the Series	3.35
3.15 Combining Dataframes	3.37
3.16 Boolean Reduction	3.38
3.17 Descriptive Statistics with Pandas	3.39
3.17.1 max()	3.40
3.17.2 min()	3.41
3.17.3 sum()	3.42
3.17.4 count()	3.42

3.17.5	mode(), mean(), median()	. . .	3.43
3.17.6	quantile	. . .	3.47
3.17.7	var()	. . .	3.52
3.18	Advanced Operations on Dataframes	. . .	3.53
3.19	Sorting	. . .	3.62
3.20	Creating Histogram	. . .	3.67
3.21	Function Application	. . .	3.69
3.21.1	pipe()	. . .	3.69
3.21.2	apply()	. . .	3.71
3.21.3	applymap()	. . .	3.72
3.21.4	groupby() in Pandas	. . .	3.73
3.21.5	transform()	. . .	3.75
3.22	Reindexing and Altering Labels	. . .	3.77
4.	Introduction to Software Engineering		4.1–4.22
4.1	Introduction	. . .	4.1
4.2	What is Software Engineering	. . .	4.1
4.2.1	Need for Software Engineering	. . .	4.2
4.3	Software Process	. . .	4.3
4.4	Software Process Activities	. . .	4.3
4.4.1	Software Specification	. . .	4.4
4.4.2	Software Design and Development	. . .	4.4
4.4.3	Software Validation (Testing)	. . .	4.5
4.4.4	Software Evolution/Evaluation	. . .	4.6
4.5	Software Process Models	. . .	4.6
4.5.1	Waterfall Model	. . .	4.7
4.5.2	Evolutionary Model	. . .	4.9
4.5.3	Component-based Model	. . .	4.14
4.6	Delivery Models	. . .	4.14
4.6.1	Incremental Delivery Model	. . .	4.15
4.6.2	Spiral Delivery Model	. . .	4.16
5.	Agile Methods and Practical Aspects of Software Engineering		5.1–5.30
5.1	Introduction	. . .	5.1
5.2	What is Agile Software Development	. . .	5.2
5.3	Pair Programming	. . .	5.4
5.4	Scrum	. . .	5.6
5.4.1	Scrum Team—Roles and Responsibilities	. . .	5.6
5.4.2	Scrum Events	. . .	5.7
5.5	Version Control System	. . .	5.9
5.5.1	Significance of Using Version Control System	. . .	5.9
5.5.2	Types of Version Control System	. . .	5.10
5.5.3	Why Use a Version Control System	. . .	5.11
5.6	GIT—A Distributed Version Control System	. . .	5.11
5.7	Business Use-Case Diagram	. . .	5.12
5.7.1	What is a Use-Case Diagram	. . .	5.13
5.7.2	Case Study and Use-Case Diagram for Result Management System	. . .	5.15
5.7.3	Use-Case Diagram of a Software System—“Shopping App” Use-Case Diagram	. . .	5.18
5.7.4	Use Case Diagram of a Software System—“Banking App” Use-Case Diagram	. . .	5.19
6.	Web Development with Django		6.1–6.32
6.1	Introduction	. . .	6.1
6.2	What is a Framework	. . .	6.1
6.3	What is Django	. . .	6.2
6.4	Django Web Framework	. . .	6.3
6.5	How Django Works	. . .	6.4
6.6	Django Installation	. . .	6.5
6.7	Web Server	. . .	6.5
6.8	Creating Projects	. . .	6.6
6.9	Creating Apps of a Django Project	. . .	6.15
6.10	GET and POST Methods	. . .	6.17
6.10.1	Difference Between GET and POST Methods	. . .	6.17
6.10.2	Minimal Django-Based Web Application that Parses a GET	. . .	6.20
6.10.3	Minimal Django-Based Web Application that Parses a POST	. . .	6.23
6.11	Working with Flat Files and CSV Files	. . .	6.24
6.11.1	Write the Fields to a Flat File	. . .	6.25
6.11.2	Write the Fields to a CSV File	. . .	6.25
6.11.3	Read the Fields from a CSV File	. . .	6.26

7. Interface Python with SQL	7.1–7.28
7.1 Introduction	7.1
7.2 Python-MySQL Connectivity	7.2
7.3 Why Python	7.2
7.4 Installing MySQL Connector	7.3
7.4.1 MySQLdb	7.4
7.4.2 Steps for Creating Database Connectivity Applications	7.6
7.5 Establishing Connection	7.7
7.6 Creating Cursor Object	7.8
7.7 Creating a Database	7.9
7.8 Closing Cursor and Connection	7.21
7.9 Operations on a Table in a Nutshell	7.21
8. More on SQL	8.1–8.32
8.1 Introduction	8.1
8.2 Functions in MySQL	8.1
8.3 Aggregate Functions in SQL	8.3
8.4 Sorting in SQL—Order By	8.9
8.5 Group By	8.10
8.5.1 Having Clause	8.10
8.6 Aggregate Functions & Conditions on Groups (Having Clause)	8.12
9. Society, Law and Ethics	9.1–9.36
9.1 Introduction	9.1
9.2 Intellectual Property Rights	9.2
9.3 Plagiarism	9.2
9.4 Digital Rights Management	9.3
9.5 Licensing	9.5
9.6 Open Source and Open Data	9.8
9.7 Privacy Laws	9.9
9.8 Cybercrime	9.10
9.8.1 Phishing	9.11
9.8.2 Illegal Downloading	9.11
9.8.3 Child Pornography	9.12
9.8.4 Cyber Scams and Frauds	9.12
9.8.5 Cyber Forensics	9.13
9.9 Information Technology Act, 2000	9.14
9.10 Unique IDs and Biometrics	9.15
9.11 Impact of Technology Change on Society	9.15
9.12 E-Waste Management	9.16
9.13 Gender and Disability Issues while Teaching and Using Computers	9.19
9.14 Role of New Media in Society	9.20
9.14.1 Online Campaigns	9.20
9.14.2 Crowdsourcing	9.21
9.15 Issues with the Internet	9.23
9.15.1 Net Neutrality	9.24
9.15.2 Internet as an Echo Chamber	9.24
9.15.3 Internet Addiction	9.25
9.16 Role of New Media—Case Studies	9.26
9.16.1 Case Study 1: Arab Spring	9.26
9.16.2 Case Study 2: WikiLeaks	9.26
9.16.3 Case Study 3: Bitcoins	9.27
APPENDICES	A.1–A.10
Appendix A: Sample Question Paper (Solved)	A.1–A.7
Appendix B: Model Test Paper (Unsolved)	A.8–A.10

To My Parents

Shri Gulshan Kumar Arora

and

Smt. Kamlesh Arora





Data Visualization Using Pyplot

2.1 INTRODUCTION

“A picture is worth a thousand words.”

We all know that images or visuals are a powerful form of communication. We often use them to understand a situation better or to condense pieces of information into a graphical representation. Visualization is the easiest way to analyze and absorb information. It is the first step for any kind of data analysis work. Visuals or better called *Data Visualization* help us to easily understand a complex problem and see certain patterns. They also help in identifying patterns, relationships and outliers in data and in understanding business problems better and quickly. Insights gathered from the visuals help in building strategies for businesses. For this reason, data visualization techniques have gained popularity. **Data visualization** basically refers to the graphical or visual representation of information and data using visual elements like charts, graphs, maps, etc.

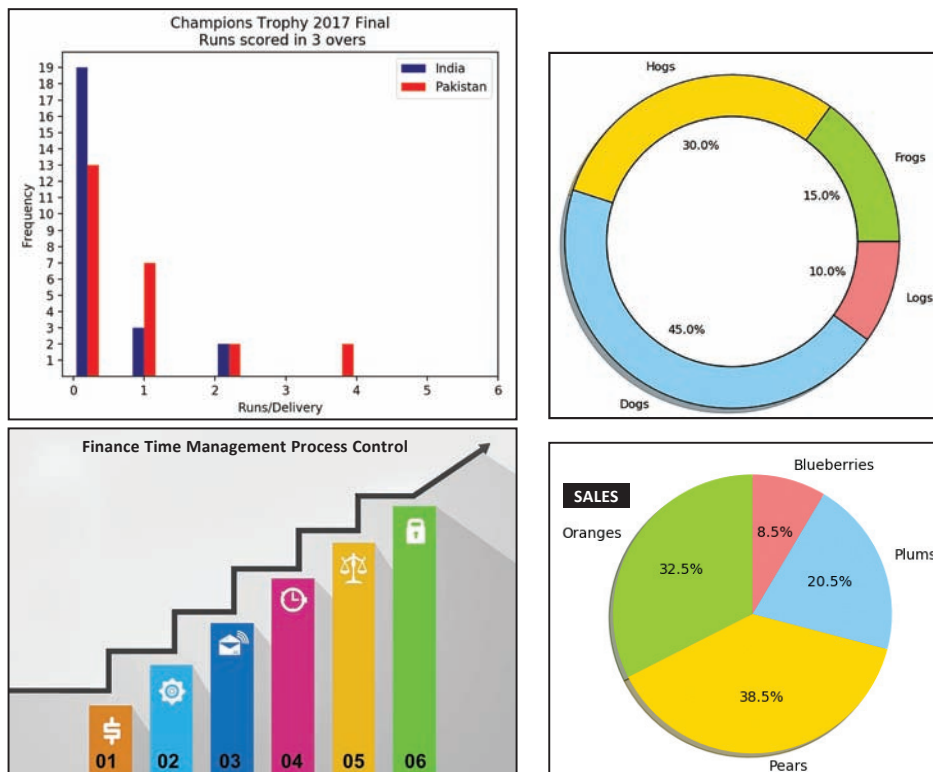


Fig. 2.1: Significance of Graphs in Various Applications

Python is an interpreted language with a strong core functions basis and a powerful modular aspect which allows us to expand the language with external modules that offer new functionalities. Thus, we have an extensible language with tools to accomplish a single task in the best possible way. Modules are often organized in packages. A package is a structured collection of modules that has the same purpose. Data visualization in Python can be done via many packages. One example of a package is **Matplotlib**.

Matplotlib package can be used in Python scripts, Jupyter notebook and web application servers. In Python, we can use two exclusive libraries for visualization, commonly known as *matplotlib* and *seaborn*. However, we shall be restricting ourselves to the usage of matplotlib only.

CTM: Data Visualization refers to the graphical or visual representation of information and data using visual elements like charts, graphs, maps, etc.

2.2 MATPLOTLIB

Matplotlib is a 2D plotting library that helps in visualizing figures. Matplotlib is used in Python as it is a robust, free and easy library for data visualization. It is easy to learn and understand.



Data visualization is an important part of business activities as organizations nowadays collect a huge amount of data. Sensors all over the world are collecting climate data, user data through clicks, car data for prediction of steering wheels, etc. All this data collected holds key insights for businesses and visualizations make these insights easy to interpret. **Data is only as good as it is presented.**

Matplotlib is the most popular plotting library for Python. It gives us control over every aspect of a figure. It supports interactive and non-interactive plotting and can save images in several output formats (PNG, PS and others). It was originally written by J.D.Hunter and is actively being developed. It is distributed under a BSD-Style Licence.

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, powerspectra, error charts, etc. It is used along with NumPy to provide an environment for MatLab. It supports interactive and non-interactive plotting and can save images in several output formats (PNG, PS and others).

CTM: Matplotlib is a Python 2D plotting library which produces publication-quality figures. Pyplot is a module of matplotlib library (of Python) containing collection of methods which allows a user to create 2D plots and graphs easily and interactively.

It is a programming platform, designed specifically for engineers and scientists, which allows the most natural expression of computational mathematics.

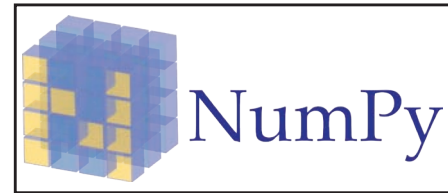
Pyplot provides the state-machine interface to the plotting library in matplotlib. It means that figures and axes are implicitly and automatically created to achieve the desired plot. *For example*, calling `plot()` from pyplot will automatically create the necessary figure and axes to achieve the desired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

Each pyplot function makes some changes to a figure, like creating a figure, creating a plotting area in a figure, plotting some lines in a plotting area, decorating the plot with labels, etc.

CTM: A plot is a graphical representation technique for representing a dataset, usually as a graph, showing the relationship between two or more variables.

2.3 NumPy

Another library which helps in the process of plotting graphs/charts using pyplot is NumPy. NumPy stands for numerical Python. NumPy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object and tools for working with these arrays.



Using NumPy, a developer can perform the following operations:

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

We can install NumPy using the popular Python package installer, pip. Type the following command at the command prompt—

```
>C:\pip install numpy
```

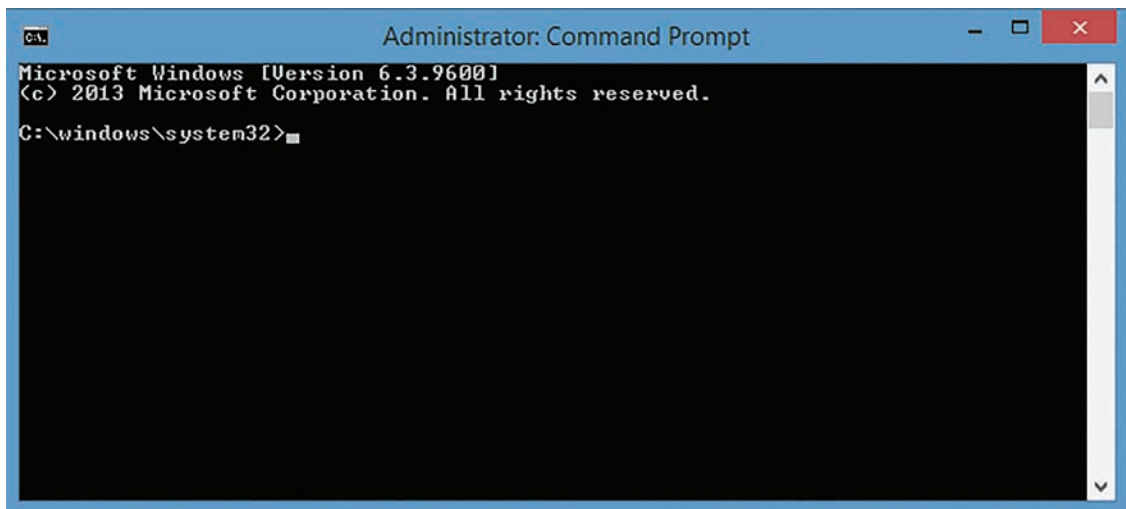
NumPy will get installed onto your system and will be ready to be used.

2.4 INSTALLING MATPLOTLIB

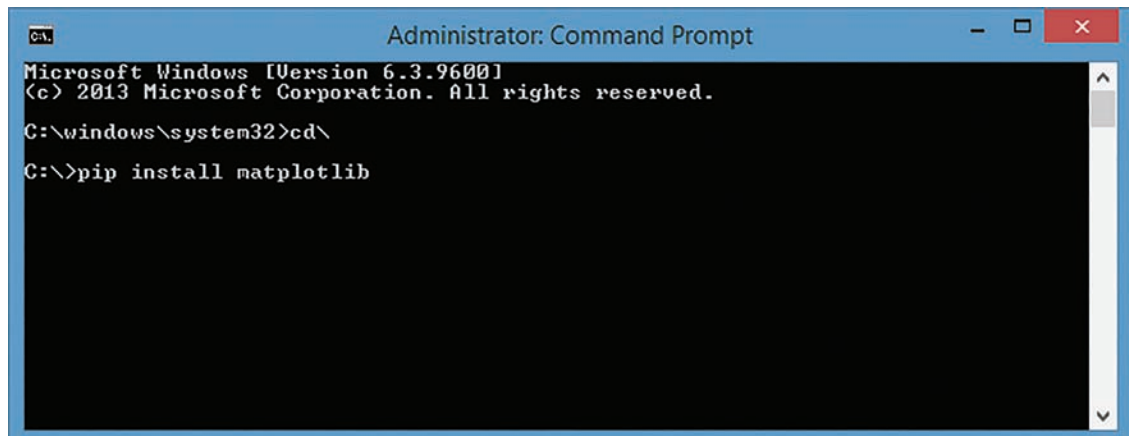
Before we start plotting graphs in matplotlib, it needs to be installed first. For the installation of Matplotlib, follow the steps listed below:

Step 1: Open cmd (command prompt) and run command prompt as an Administrator.

The following window gets displayed.



Step 2: Type `cd\` to move to the root directory.



```
Administrator: Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

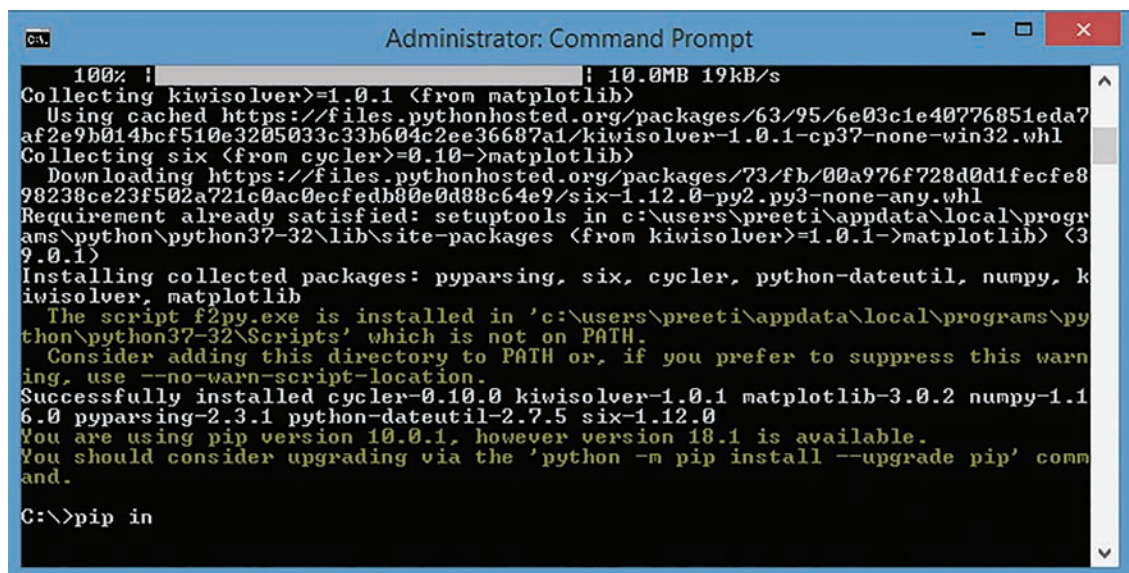
C:\windows\system32>cd\
C:\>pip install matplotlib
```

Step 3: Type: `pip install matplotlib` (with internet connection) as shown in the above window.

Note: Please ensure that the system has Python Shell installed before installing matplotlib Library.

Step 4: Installation of Matplotlib will start.

After the installation is successfully done on the system, an appropriate message shall be displayed as shown in the window given below:



```
Administrator: Command Prompt
100% | | 10.0MB 19kB/s
Collecting kiwisolver>=1.0.1 (from matplotlib)
Using cached https://files.pythonhosted.org/packages/63/95/6e03c1e40776851eda7af2e9b014bcf510e3205033c33b604c2ee36687a1/kiwisolver-1.0.1-cp37-none-win32.whl
Collecting six (from cyclor)=0.10->matplotlib)
Downloading https://files.pythonhosted.org/packages/73/fb/00a976f728d0d1fecfe898238ce23f502a721c0ac0ecfedb80e0d88c64e9/six-1.12.0-py2.py3-none-any.whl
Requirement already satisfied: setuptools in c:\users\preeti\appdata\local\programs\python\python37-32\lib\site-packages (from kiwisolver)=1.0.1->matplotlib) (39.0.1)
Installing collected packages: pyparsing, six, cyclor, python-dateutil, numpy, kiwisolver, matplotlib
The script f2py.exe is installed in 'c:\users\preeti\appdata\local\programs\python\python37-32\Scripts' which is not on PATH.
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
Successfully installed cyclor-0.10.0 kiwisolver-1.0.1 matplotlib-3.0.2 numpy-1.16.0 pyparsing-2.3.1 python-dateutil-2.7.5 six-1.12.0
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\>pip in
```

➤ After the successful installation of the above Library package (Matplotlib), we can plot various types of graphs in Python shell using pyplot methods.

POINT TO REMEMBER
Internet connection is required only at the time of installation of Matplotlib. After installation gets completed, there is no such requirement.

2.5 TYPES OF VISUALIZATION

Matplotlib can be used to explore the basic plotting capabilities for single or multiple lines. We can add information to the plots such as legends, axis labels and titles. It also provides the capability to save a plot to a file.

There are many types of visualizations available with Matplotlib. Some of the most famous are: **line plot, scatter plot, histogram, box plot, bar chart and pie chart.**

In this chapter, we shall discuss line chart, bar chart, histogram, frequency polygons, box plots and scatter plots as per the CBSE curriculum.

2.6 BASIC VISUALIZATION RULES

Before we look at some of the plots, let us introduce some basic rules. These rules help us to make nice and informative plots instead of confusing ones.

Before you plot/create any chart or graph type, make sure to import the matplotlib.pyplot library by giving the command:

```
import matplotlib.pyplot
```

OR `import matplotlib.pyplot as plt`

Here, plt is an alias name for pyplot, so now you can use plt for typing pyplot commands as plt. <command>

Along with pyplot, if you are using NumPy functionality, make sure to import it also using the command as:

```
import numpy as np
```

The next step is to choose an **appropriate** plot type. If there are various options, we should compare them and choose the one that fits our model the best.

Third, when we choose the type of plot, one of the most important things is to **label the axis**. If we don't do this, the plot is not informative enough. When there are no axis labels, we can try to look at the code to see what data is used and, if we're lucky, we'll understand the plot.

Fourth, we can add a title to make our plot more informative.

Fifth, add labels for different categories when needed.

Sixth, optionally we can add a text or an arrow at relevant data points.

Seventh, in some cases, we can use some sizes and colours of the data to make the plot more informative.

For drawing simple charts, the line charts and scatter charts are almost similar. The only difference is the presence/absence of the lines connecting the points. Also, using **plot()** function of pyplot, you can create both of these basic charts. However, the scatter charts can also be created using **scatter()** function, which we shall be learning in the subsequent topics.

Let us begin with line chart first.

2.7 BASIC NOMENCLATURE OF A PLOT

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).

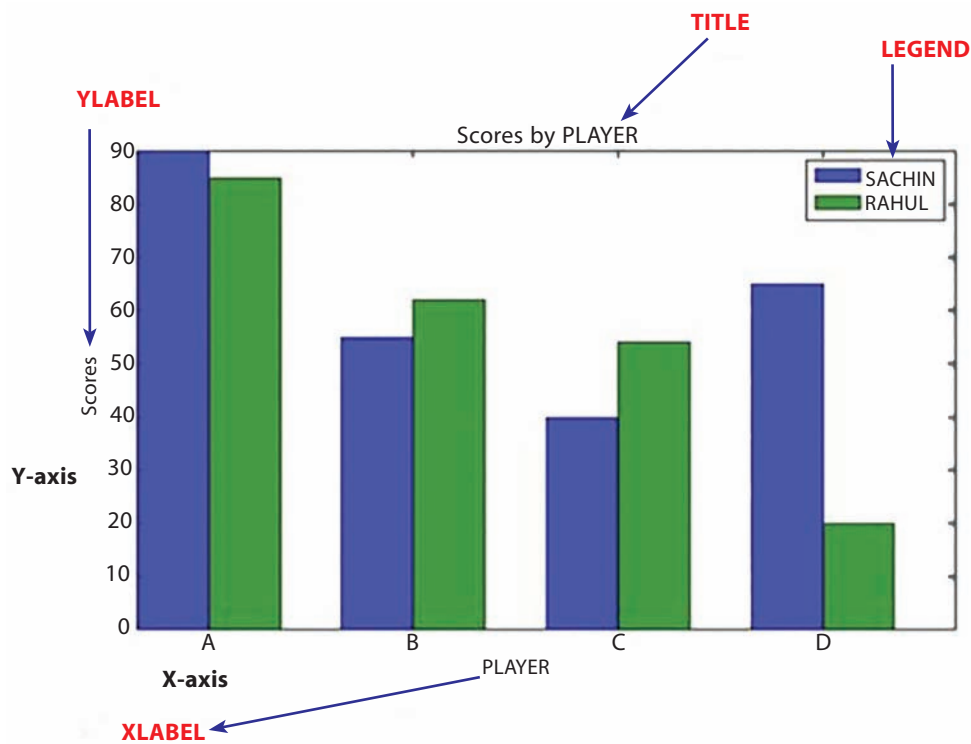


Fig. 2.2: Basic Components of a Graph/Chart

A Matplotlib figure can be categorized into several parts as below:

- **Figure:** It is a whole figure which may contain one or more than one axes (plots). You can think of a **Figure** as a canvas which contains plots.
- **Axes:** It is what we generally think of as a plot. A **Figure** can contain many Axes. It contains two or three (in the case of 3D) **Axis** objects. Each Axes has a title, an x-label and a y-label.
- **Axis:** They are the number line like objects and take care of generating the graph limits.
- **Artist:** Everything which one can see on the figure is an artist like Text objects, Line2D objects, collection objects. Most Artists are tied to Axes.
- **Labels:** To manage the axes dimensions of a plot, another important piece of information to add to a plot is the axes labels, since they usually specify what kind of data we are plotting.
- **Title:** Just like in a book or a paper, the title of a graph describes what it is. Matplotlib provides a simple function, **plt.title()**, to add a title to an image.
- **Legend:** Legends are used to explain what each line means in the current figure.

2.8 LINE PLOT/CHART

Line plot/chart is a type of plot which displays information as a series of **data points** called “markers” **connected by straight lines**. In this type of plot, we need the measurement points to be **ordered** (typically by their X-axis values). This type of plot is often used to visualize a trend in data over intervals of time—a **time series**.

The line chart is represented by a series of data points connected by a straight line. Generally, line charts are used to display trends over time. A line chart or line graph can be created using the `plot()` function available in pyplot library. We can not only just plot a line but also explicitly define the grid, the X-axis and Y-axis scale and labels, title and display options.

To make a line plot with matplotlib, we call `plt.plot()`. The first argument is used for the data on the horizontal axis, and the second is used for the data on the vertical axis. This function generates your plot but it doesn't display it. To display the plot, we need to call the `plt.show()` function.

Markers and Line Styles

In the Practical Implementation examples that we are going to implement in successive subtopics, all the plots are made of points with lines joining them. The points are the pairs (x,y) from the X and Y input lists we pass to `plot()`; lines are the straight segments connecting any two adjacent points.

Points are almost invisible, if not for the edges in the graph. However, they are the real generators of the plot because points mark positions. As a result, they are called **markers** in matplotlib terminology.

By default, Matplotlib draws markers as a single dot and lines as straight thin segments; there are situations where we would like to change either the marker style (to clearly identify them in the plot) or the line style.

CTM: A line chart or line graph is a type of chart which displays information as a series of data points called 'markers' connected by straight line segments.

In order to draw a line plot, the steps to be followed are as under:

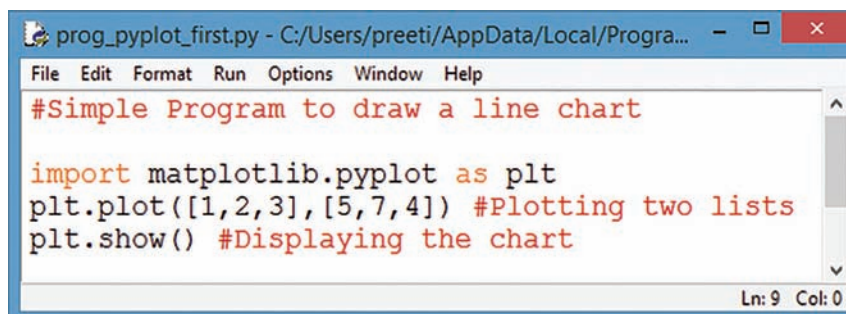
Steps:

1. Importing matplotlib.
2. `plt.plot(x, y, color, others)` Plot y versus x as lines and/or markers.
3. `plt.xlabel("Your Text")` Set the X-axis label of the current axes.
4. `plt.ylabel("Your Text")` Set the Y-axis label of the current axes.
5. `plt.set_title("Your Title")` Set a title of the current axes.
6. `plt.show()` Display a figure.

This can be better understood through the Practical Implementation that follows.

Practical Implementation-1

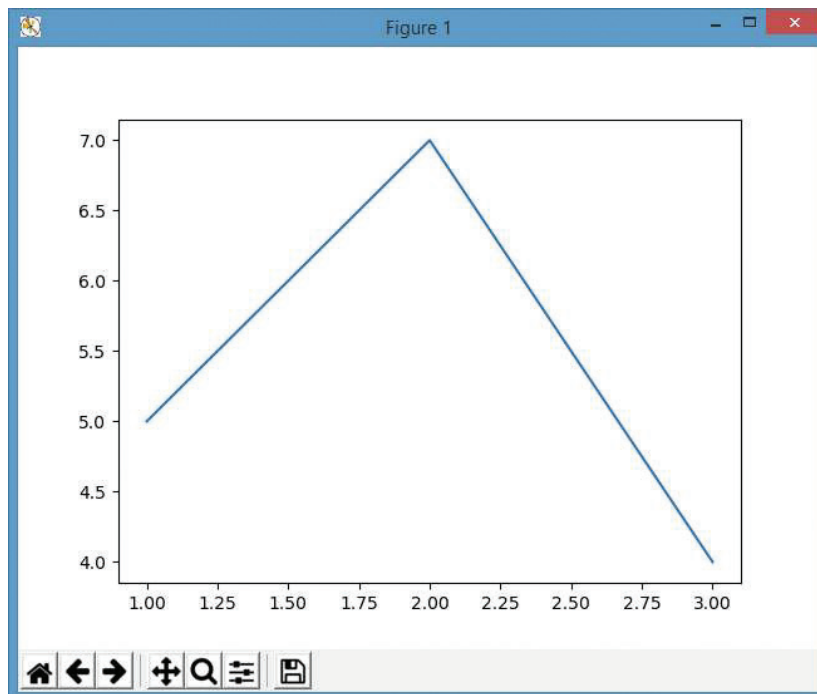
To plot a simple line chart using two lists.



```
prog_pyplot_first.py - C:/Users/preeti/AppData/Local/Progra... - □ ×
File Edit Format Run Options Window Help
#Simple Program to draw a line chart

import matplotlib.pyplot as plt
plt.plot([1,2,3],[5,7,4]) #Plotting two lists
plt.show() #Displaying the chart

Ln: 9 Col: 0
```



Explanation:

For plotting a line plot, the first thing to be done is importing matplotlib using the statement:

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

Here, plt is a short name or an alias name and a standard. Similarly, NumPy is imported as np, which we will be using for successive charts. These short names make the code more clear and easy to understand. In the next statement, plt is used to call plot() method, which plots the graph between the given set of values.

```
plt.plot([1,2,3],[5,7,4])
```

This statement describes two lists as the arguments to be plotted in the graph. Since everything is drawn in the background first, it is brought and displayed in the front using the command plt.show().





plt.show() is used to display the graph.

Interactive Navigation Toolbar



When using matplotlib.pyplot, the toolbar is enabled by default for every figure. It provides basic elaboration and manipulation functions for interactive plotting. At the bottom of the window, we can find the navigation toolbar. A description of each of its buttons (from left to right) follows:

Apart from the plot area, we have a few options displayed at the bottom-left corner of the window.

-  **Reset Original Window:** We can resize the plot window as per our requirement. It can be reset to the original window size using this option.
-  **Back to Previous View:** It takes us to the previous view from the current view.
-  **Forward to Next View:** It takes us to the next view from the current view.
-  **Pan Axis with Left Mouse, Zoom with Right:** Used to zoom any section of the plotted figure.

- **Pan:** Click on the left mouse button and hold it to pan the figure, dragging it to a new position. Once you are happy with the position, release the mouse button. While panning, if we press (or hold) the x or y key, then the panning is limited to the selected axis.
 - **Zoom:** Click on the right mouse button and hold it to zoom the figure, dragging it to a new position. Movement to the right or to the left generates a proportional zoom in or out of the X-axis of the figure. The same holds true for the up or down movement of the Y-axis. The point where we click the mouse remains still so that we are able to zoom around a given point in the figure. The x and y keys work in the same way as mentioned earlier, but now we can press the **Ctrl** key to preserve the aspect ratio.
- Q Zoom to Rectangle:** Enabling this mode, we can draw a rectangle on the figure (hold the left mouse button while drawing it) and the view will be zoomed to that rectangle.
- Configure Subplots:** When we click on this button, a window pops up that allows us to configure the various spaces that surround the figure (left, right, up, bottom, between).
- Save the Figure:** This option is used to save the figure drawn to the hard disk by giving a proper name to it. Click on this button and a save file dialog box will pop up that allows us to save the current figure.

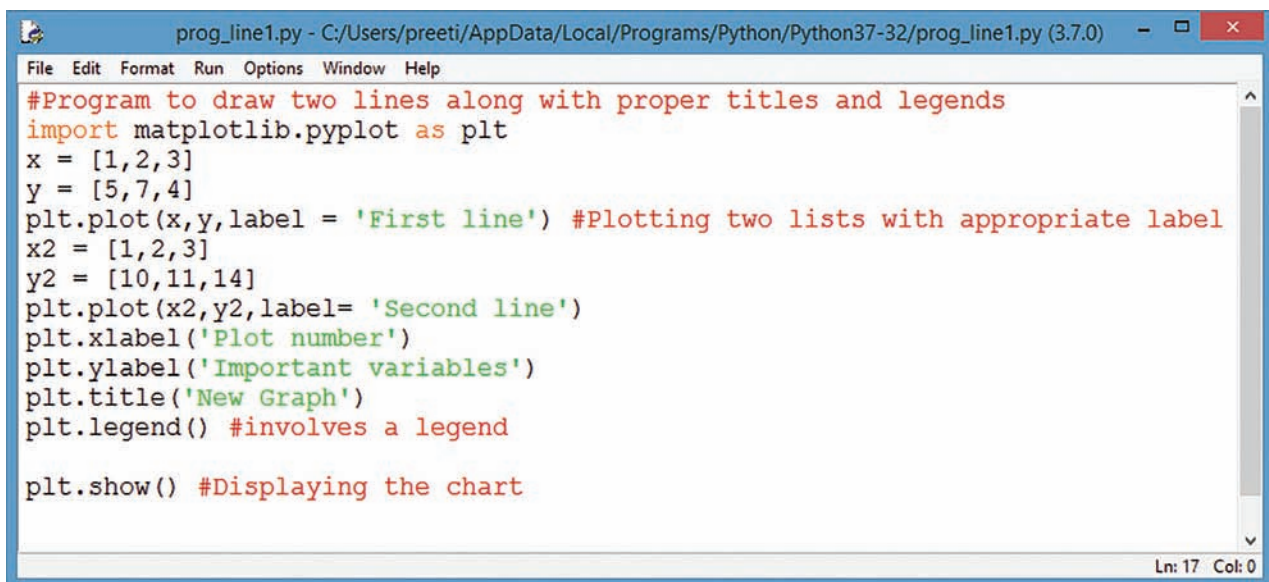
This is useful because we might want to add some additional customizations to our plot before we display it. *For example*, we might want to add labels to the axis and title for the plot.

2.8.1 Multiple Plots

If we want to plot multiple lines in one chart, we can simply call the plot() function multiple times.

Practical Implementation-2

To add legends, titles and labels to a line plot with multiple lines.

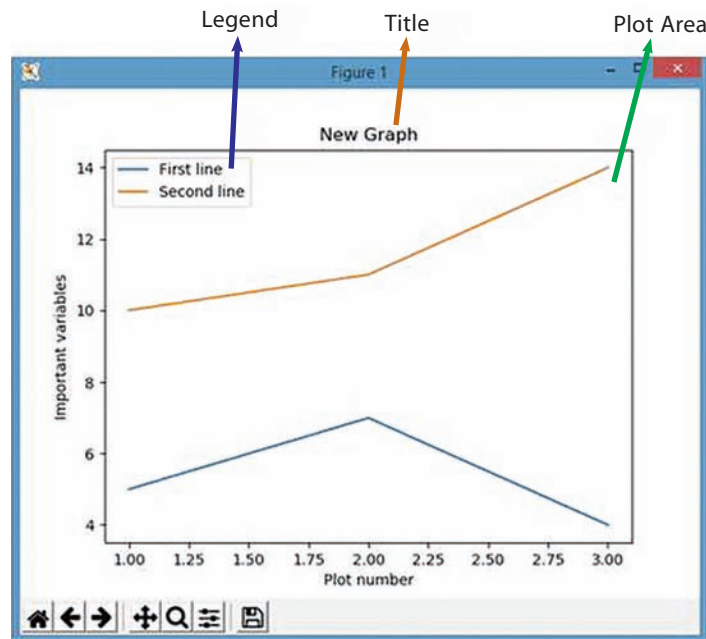


```

prog_line1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_line1.py (3.7.0)
File Edit Format Run Options Window Help
#Program to draw two lines along with proper titles and legends
import matplotlib.pyplot as plt
x = [1,2,3]
y = [5,7,4]
plt.plot(x,y,label = 'First line') #Plotting two lists with appropriate label
x2 = [1,2,3]
y2 = [10,11,14]
plt.plot(x2,y2,label= 'Second line')
plt.xlabel('Plot number')
plt.ylabel('Important variables')
plt.title('New Graph')
plt.legend() #involves a legend

plt.show() #Displaying the chart
Ln: 17 Col: 0
  
```

CTM: Legends can be dynamically changed.



Explanation:

In the above program, we have drawn two lines using line chart with proper titles given along X-axis and Y-axis. Also, a new term has been used, *i.e.*, legend. If we look at the line graphs of our previous examples, we realize that we have to look into our code to understand what kind of function is depicted. This information should be available in the diagram for the sake of convenience. Legends are used for this purpose. So, legend is the text or string that “has to be read” to understand the graph. Legends are used in line graphs to explain the function or the values underlying the different lines of the graph.

Learning Tip: We can draw as many lines as required by calling plot() function multiple times with suitable arguments.

2.8.2 Multiple Views

In case we want to plot legends in different views in the same window, we can use the subplot() function as shown in Practical Implementation–3:

Practical Implementation–3

To plot two lines in two different views of the same window.

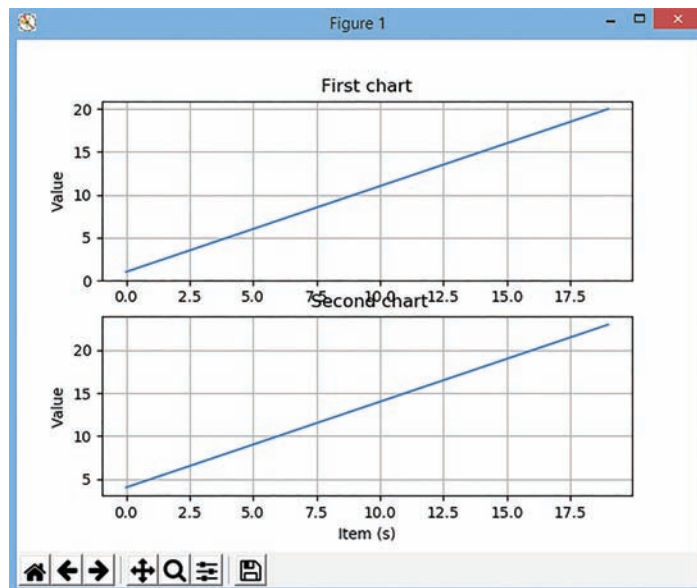
```

prog_multiple_views.py - C:\Users\preeti\AppData\Local\Programs\Python\Python37-...
File Edit Format Run Options Window Help
#Plotting line chart in different views
import matplotlib.pyplot as plt
import numpy as np #Importing numpy

t = np.arange(0.0, 20.0, 1)
s = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
s2 = [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]

plt.subplot(2, 1, 1)
plt.plot(t, s)
plt.ylabel('Value')
plt.title('First chart')
plt.grid(True)

plt.subplot(2, 1, 2)
plt.plot(t, s2)
plt.xlabel('Item (s)')
plt.ylabel('Value')
plt.title('\n\n Second chart')
plt.grid(True)
plt.show()
    
```



In the above program, the `plt.subplot()` statement is used. The `subplot()` command specifies `numrows`, `numcols` and `fignum`. In the above program, we have imported NumPy library through the statement—`import numpy as np`.

Practical Implementation–3A

To plot two lines in two different views of the same window by adjusting the space between the subplots. (Modification of Practical Implementation–3)

```

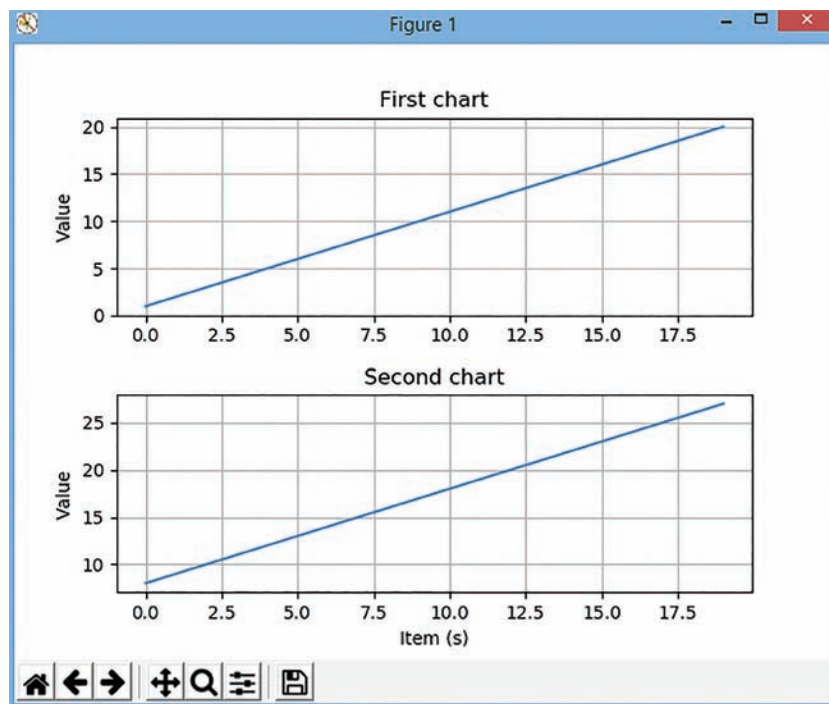
prog_multiple_views.py - C:\Users\preeti\AppData\Local\Programs\Python\Python37-32\prog_multiple_views.py (... - □ ×
File Edit Format Run Options Window Help
#Plotting line chart in different views
import matplotlib.pyplot as plt
import numpy as np #Importing numpy

t = np.arange(0.0, 20.0, 1)
s = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
s2 = [8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27]

plt.subplot(2,1,1)
plt.ylabel('Value')
plt.title('First chart')
plt.grid(True)
plt.plot(t,s)
plt.subplots_adjust(hspace=0.4, wspace=0.4) #Adjust the spacing between
#two subplots
plt.subplot(2,1,2)
plt.xlabel('Item (s)')
plt.ylabel('Value')
plt.title('Second chart')
plt.plot(t,s2)
plt.grid(True)
plt.show()
Ln: 29 Col: 0

```

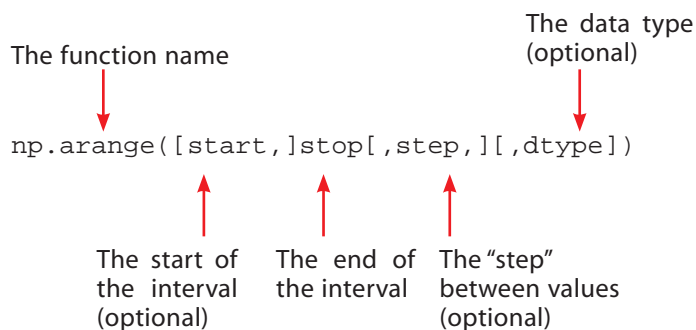
In the above program, we have modified the code for Practical Implementation-3. `subplots_adjust()` method is used for providing horizontal spaces (`hspace`) and width-wise spaces (`wspace`) between two subplots so that their respective titles or any other subcomponents don't overlap or collide with each other and, hence, the output is so obtained.



arange()

One more function that we have used in the above code is `arange()`. The NumPy `arange` function (sometimes called `np.arange`) is a tool for creating numeric sequences in Python. It returns evenly spaced numeric values within an interval, stored as a NumPy array or we can say a list (*i.e.*, an ndarray object).

The syntax for `arange()` is:



Here,

- **start** (optional)

The start parameter indicates the beginning value of the range.

This parameter is *optional*, so if you omit it, it will automatically default to 0.

- **stop** (required)

The stop parameter indicates the end of the range. Keep in mind that like all Python indexing, this value will not be included in the resulting range.

- **step** (optional)

The step parameter specifies the spacing between values in the sequence.

This parameter is optional. If you don't specify a step value, by default, the step value will be 1.

➤ **dtype** (optional)

The dtype parameter specifies the data type.

For example, to create a range of values from 0 to 8, in increments of 2, we will use the start position of 0 and a stop position of 8. To increment in steps of 2, we'll set the step parameter to 2. Hence, the statement shall be:

```
np.arange(start = 0, stop = 8, step = 2)
```

This command shall create a sequence of numbers as shown:

0	2	4	6
---	---	---	---

The last value should be 8 but as per the syntax it is to be excluded and, hence, 6 will be taken as the last number to be displayed for the given range.

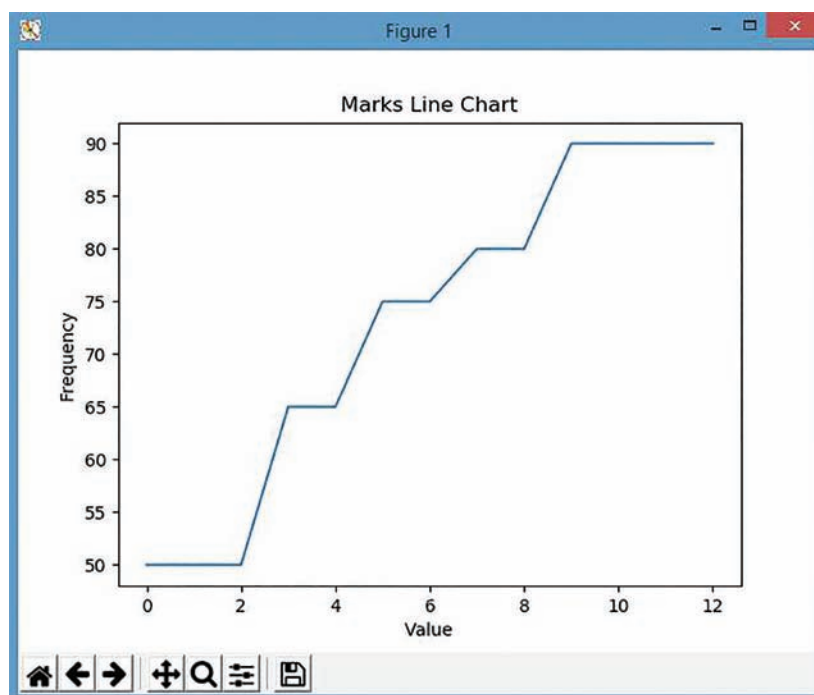
Practical Implementation-4

Program to plot frequency of marks using line chart.

```
prog_pyplot4.py - C:/Users/preeti/AppData/Local/Programs/Python/Pyt... - □ ×
File Edit Format Run Options Window Help
#Program to plot frequency of marks using Line Chart
import matplotlib.pyplot as plt

def fnplot(list1):
    plt.plot(list1)
    plt.title("Marks Line Chart")
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    plt.show()

list1=[50,50,50,65,65,75,75,80,80,90,90,90,90]
fnplot(list1)
Ln: 16 Col: 0
```



Practical Implementation-5

Program to plot a sine wave using line chart.

```

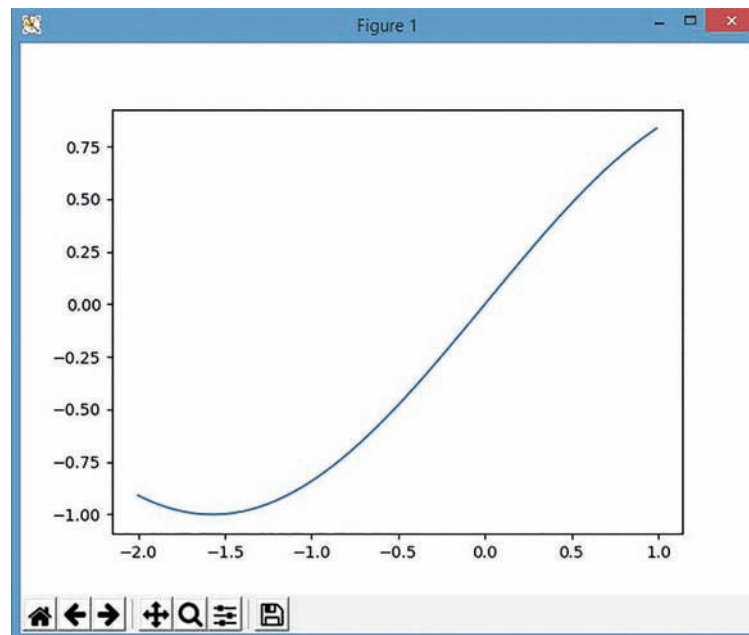
prog_sinewve.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_sinewv...
File Edit Format Run Options Window Help
#Program for plotting sine wave using Line chart

import matplotlib.pyplot as plt
import numpy as np
xvals = np.arange(-2, 1, 0.01) # Grid of 0.01 spacing from -2 to 1
yvals = np.sin(xvals) # Evaluate function on xvals
plt.plot(xvals, yvals) # Create line plot with yvals against xvals
plt.show() # Show the figure
Ln: 12 Col: 0

```

In the program given on the previous page, we have used another method `arange()`. It is used to provide a range of points to be displayed as a grid, with first argument as the starting point followed by end point/value, and the third argument constituting the increment/step value.

Sine wave is formed by using `sin()` method and passing the values on the X-axis as the parameter.



Practical Implementation-6

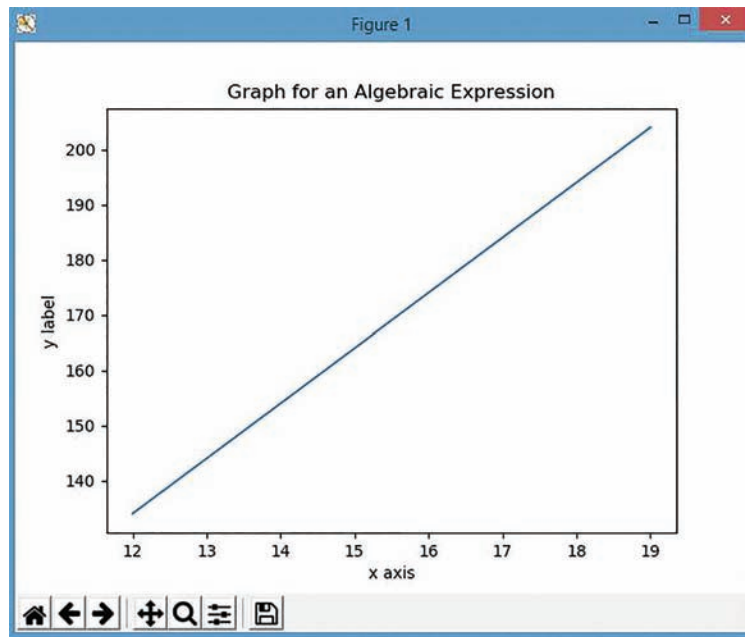
Program to plot an algebraic expression: $10x + 14$ using line chart.

```

prog_algebrpyplot.py - C:/Users/preeti/AppData/Local/Progr...
File Edit Format Run Options Window Help
#Program to evaluate an algebraic expression
# 10x + 14 using Line Chart

import numpy as np
from matplotlib import pyplot as plt
x=np.arange(12,20)
y=10*x+14
plt.title("Graph for an Algebraic Expression")
plt.xlabel("x axis")
plt.ylabel("y label")
plt.plot(x,y)
plt.show()
Ln: 15 Col: 0

```



Practical Implementation-7

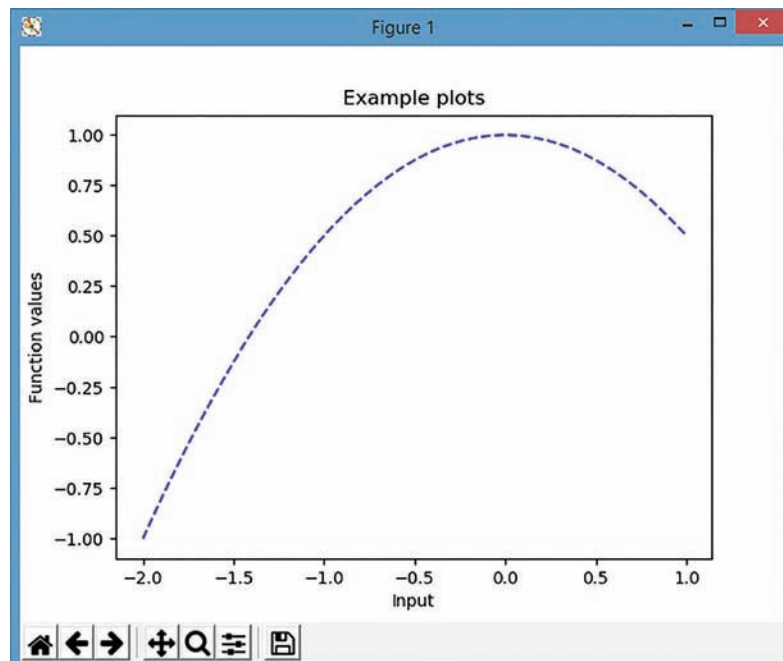
Program to plot a quadratic equation using dashed line chart.

```

prog_quad_pyplot.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_quad_pyplot.py (... - □ ×
File Edit Format Run Options Window Help
#Program to plot a quadratic equation using dashed Line Chart
# 1 - 0.5*x**2

import matplotlib.pyplot as plt
import numpy as np
xvals = np.arange(-2, 1, 0.01) #Grid of 0.01 spacing from -2 to 1
newyvals = 1 - 0.5 * xvals**2 #Evaluate quadratic approximation on xvals
plt.plot(xvals, newyvals, 'b--') #Create line plot with blue dashed line
plt.title('Example plots') #Creates heading Text
plt.xlabel('Input') #Creates Text along x-axis
plt.ylabel('Function values') #Creates text along y-axis
plt.show() # Show the figure (remove the previous instance)
Ln: 15 Col: 0

```



For plotting any equation, NumPy library is required to be imported along with matplotlib. With the plot(), we have given an argument as 'b--' which denotes that the line which will be displayed shall be of dashed type with blue colour as given with the starting letter 'b'.

Changing Line Colours and Styles

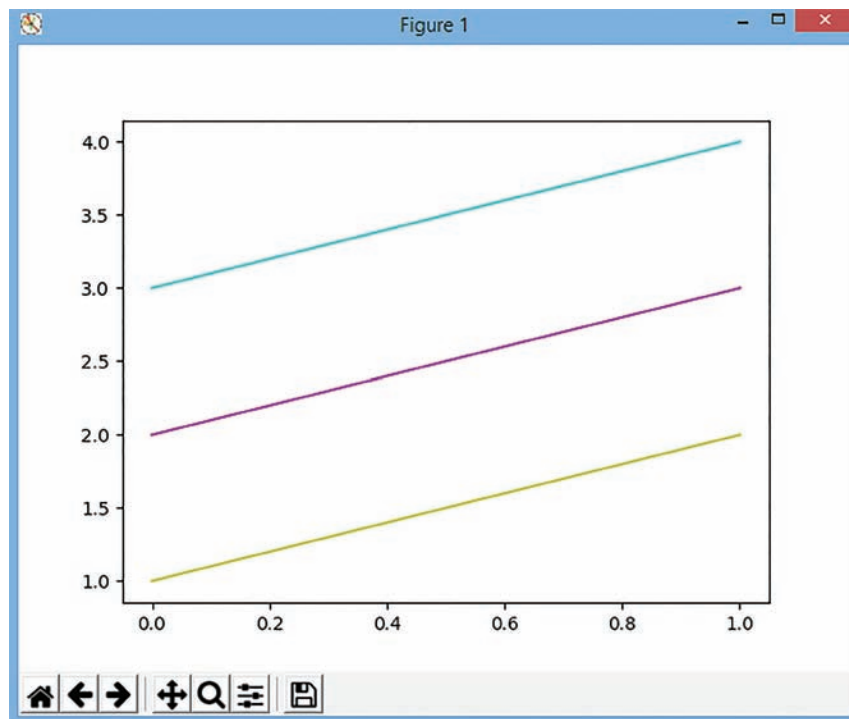
We have already seen that in a multiline plot, Matplotlib automatically chooses different colours for different lines. We are also free to choose them by ourselves. Matplotlib provides different styles and colours for line(s) we are plotting. To understand this better, it has been implemented in Practical Implementation–8.

Practical Implementation–8

To plot multiple lines with different colours defined explicitly.

```
prog_line_color.py - C:/Users/preeti/AppData/Local/Programs/Python/Pyt...
File Edit Format Run Options Window Help
#Multiple lines with multiple colors given explicitly

import matplotlib.pyplot as plt
import numpy as np
y = np.arange(1, 3)
plt.plot(y, 'y')
plt.plot(y+1, 'm')
plt.plot(y+2, 'c')
plt.show()
Ln: 12 Col: 0
```



In the preceding code, we specify colour as the last argument (in this case, with an implicit Y-axis)—to draw yellow, magenta and cyan lines (from bottom to top).

Here is a table of the abbreviations used to select colours:

Colour abbreviation	Colour name
b	blue
c	cyan
g	green
k	black
m	magenta
r	red
w	white
y	yellow

Apart from using colour abbreviations, we can also give complete names for colours like yellow, red, blue, etc., and hence, the output shown below is obtained.

Learning Tip: Even if you skip the colour information in `plot()`, Python will plot multiple lines in the same plot with different colours but these colours are decided internally by matplotlib (python).

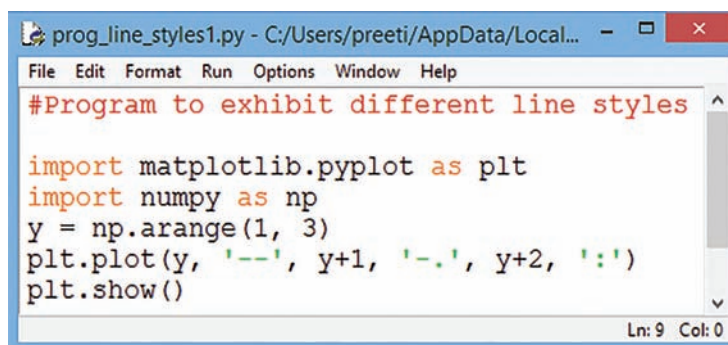
Also, the line styles can also be changed as per the needs of the user. Matplotlib allows us to use different line styles. All the available styles are listed in the following table:

Style abbreviation	Style
-	solid line
--	dashed line
-.	dash-dot line
:	dotted line

All the lines seen until now were proper ones without any dots or dashes. Matplotlib allows us to use different line styles which are implemented in Practical Implementation-9.

Practical Implementation-9

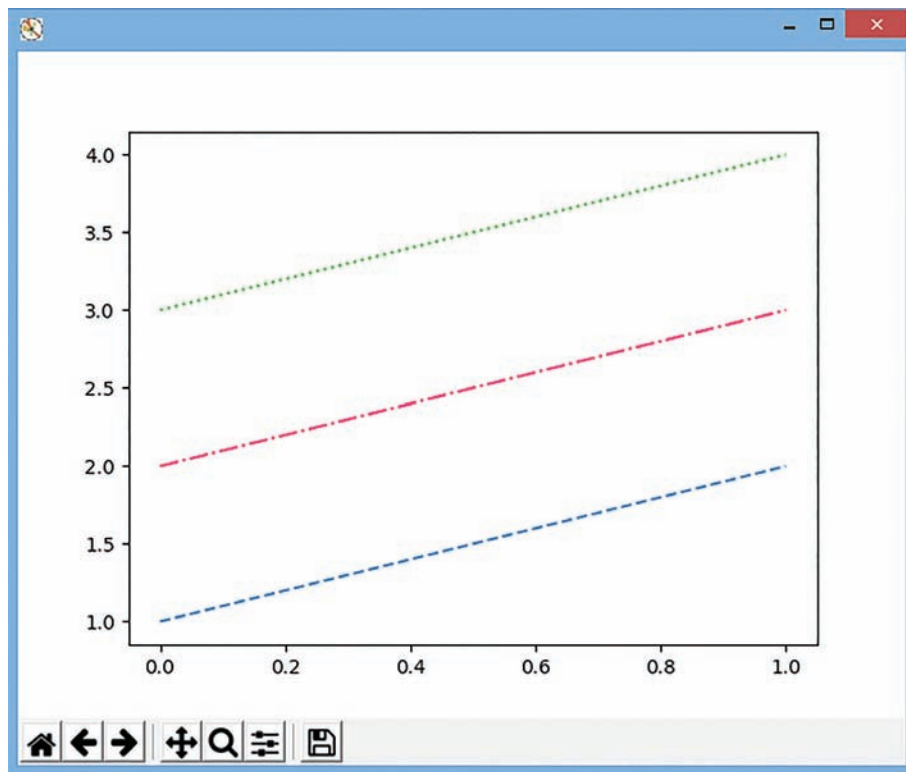
To plot lines with different styles using `plot()` function.



```
prog_line_styles1.py - C:/Users/preeti/AppData/Local...
File Edit Format Run Options Window Help
#Program to exhibit different line styles

import matplotlib.pyplot as plt
import numpy as np
y = np.arange(1, 3)
plt.plot(y, '--', y+1, '-.', y+2, ':')
plt.show()
Ln: 9 Col: 0
```

This code snippet generates a blue dashed line, a green dash-dotted line, and a red dotted line.



2.9 SCATTER CHART

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. The data visualization is done as a collection of points not connected by lines. Each of them has its coordinates determined by the value of the variables (one variable determines the X position, the other the Y position). A scatter plot is often used to identify potential association between two variables.

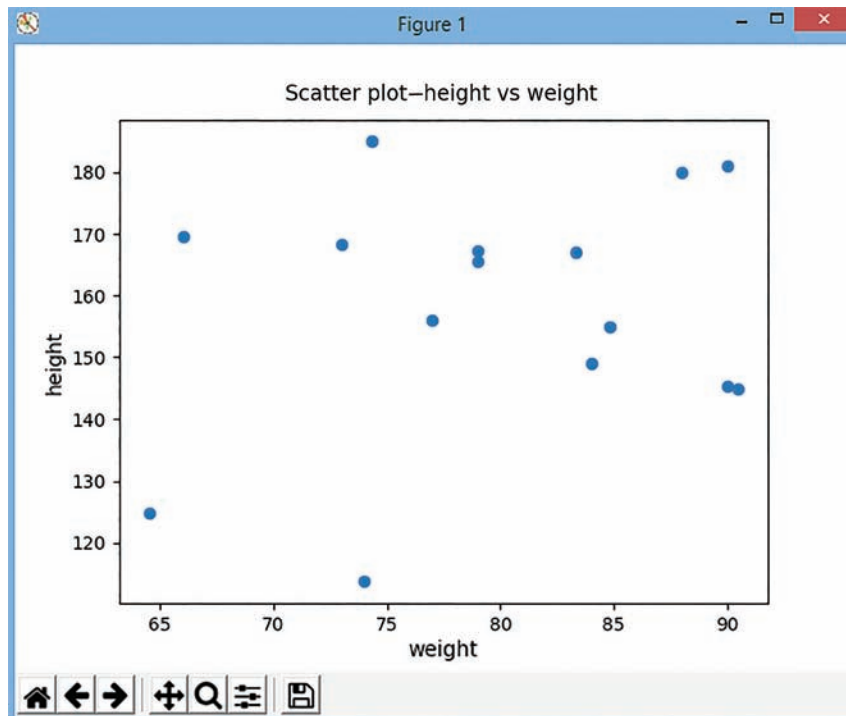
Practical Implementation-10

To plot a Scatter chart for given heights and weights of 15 students.

```

prog_scatter1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_scatter1.py (3.7.0)
File Edit Format Run Options Window Help
#To plot Scatter chart for height vs weight of the students
import matplotlib.pyplot as plt
weight =[83.3,74,64.5,79,90,84.8,84,88,66,77,79,90.5,74.3,90,73]
height =[167,113.7,124.8,165.5,145.4,155,149,180,169.5,156,167.3,145,185,181,168.3]
plt.scatter(weight,height)
plt.xlabel('weight')
plt.ylabel('height')
plt.title('Scatter plot -height vs weight',fontsize=20)
plt.show()
Ln: 12 Col: 0

```



In the above plot, the scatter chart is simple and displayed as per the default settings. We can decorate the chart by using some of the following keyword arguments:

- **s**: This stands for the size of the markers in pixel*pixel. It can be a single value (to be used for all the points) or an array of the same size of X and Y (so that each point will have its own size).
- **c**: This is the points colour. It can be a single value or a list of colours (that will be cycled on the points plotted) eventually of the same size of X and Y.
- **marker**: This specifies the marker to be used to plot the points; the available values are:

Marker value	Description
s	Square
o	Circle
^	Triangle up
v	Triangle down
>	Triangle right
<	Triangle left
d	Diamond
p	Pentagon
h	Hexagon
8	Octagon
+	Plus
x	Cross

Practical Implementation–11

Modification of Practical Implementation-10.

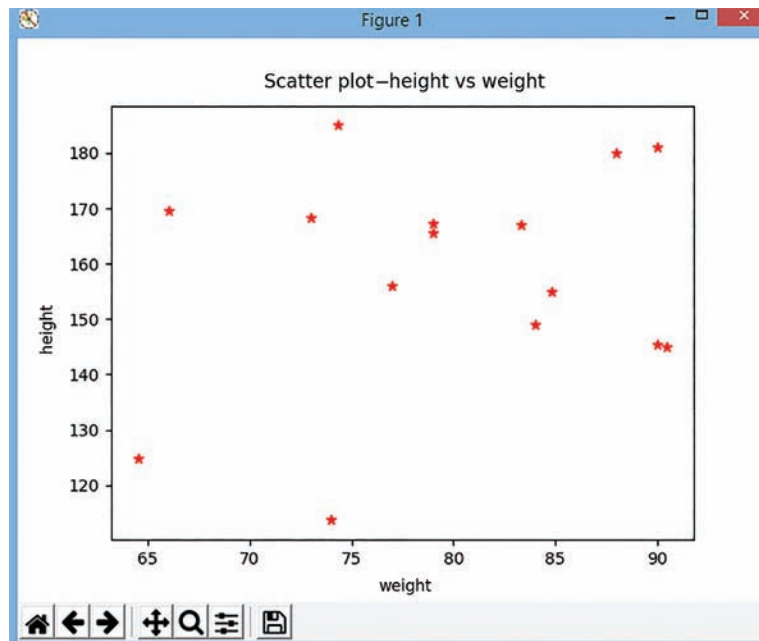
In the previous implementation, the plot exhibited no special attributes and the attributes like colour and shape were decided by matplotlib. You can also select the marker value and its associated colour displayed as per your choice by explicitly defining 'c' and 'marker' with scatter() function.

```

prog_scatter1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_scatter1.py (3.7.0)
File Edit Format Run Options Window Help
#To plot Scatter chart for height v/s weight of the students with
#well defined color and marker value

import matplotlib.pyplot as plt
weight =[83.3,74,64.5,79,90,84.8,84,88,66,77,79,90.5,74.3,90,73]
height =[167,113.7,124.8,165.5,145.4,155,149,180,169.5,156,167.3,145,185,181,168.3]
plt.scatter(weight,height,c='r',marker='*') #color as red and star as marker sign
plt.xlabel('weight',fontsize=16)
plt.ylabel('height',fontsize=16)
plt.title('Scatter plot -height vs weight',fontsize=20)
plt.show()
Ln: 13 Col: 0

```



As it is observed from the output window, the font size for title, x-label and y-label has been displayed as given in the command. The colour and marker style was taken as “red” and star “*” symbol as the marker value and, hence, the scatter plot is obtained accordingly.

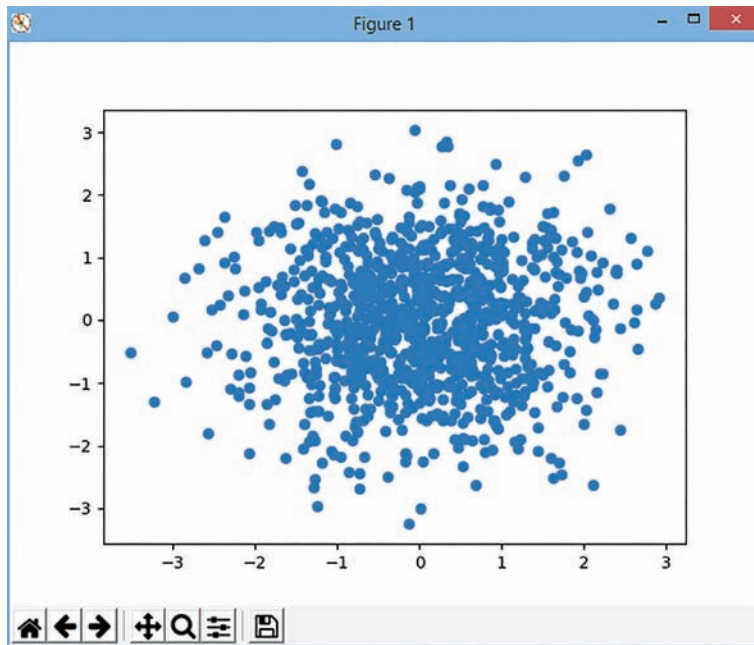
Practical Implementation–12

To generate a Scatter plot on the basis of dataset generated randomly (use of randn() function from Python random library).

In this question, we shall be importing numpy library for randn() function. Using the randn() NumPy function to generate the datasets.

```
prog_scatter2.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/pr... - [ ] [x]
File Edit Format Run Options Window Help
#To plot a scatter chart using randomly generated datasets.

import matplotlib.pyplot as plt
import numpy as np
x = np.random.randn(1000)
y = np.random.randn(1000)
plt.scatter(x, y)
plt.show()
Ln: 11 Col: 0
```



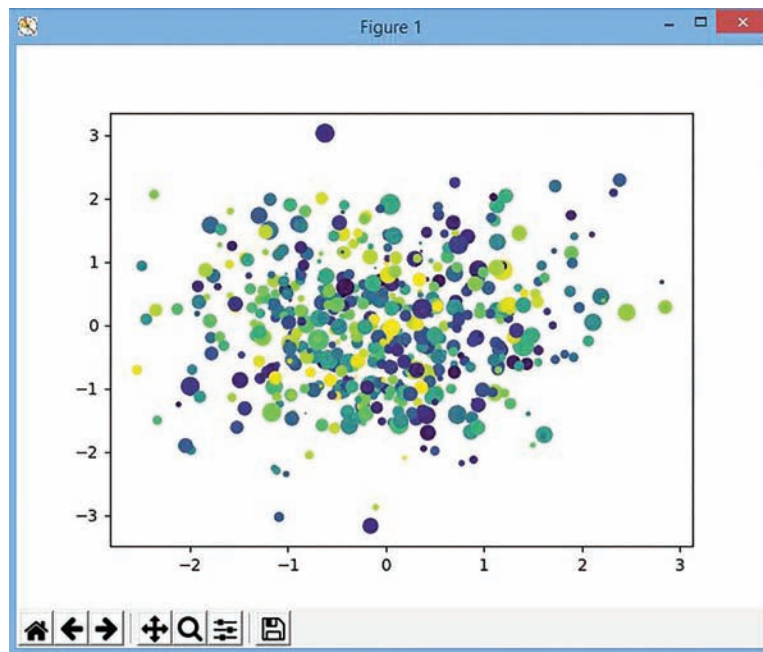
As it is observed from the output window, there are 'n' number of datasets generated using the `randn()` from NumPy library. Since we have not mentioned any colour for marker points, it will display the points in blue colour by default.

Practical Implementation-13

To generate a Scatter plot on the basis of dataset generated randomly and marker colours and size are different and explicitly defined.

```
prog_scatter2.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/p... - [ ] [x]
File Edit Format Run Options Window Help
#To plot a scatter chart using randomly generated data sets
#with different color and sizes for each point

import matplotlib.pyplot as plt
import numpy as np
x = np.random.randn(1000)
y = np.random.randn(1000)
size = 50*np.random.randn(1000)
colors = np.random.rand(1000)
plt.scatter(x, y, s=size, c=colors)
plt.show()
Ln: 15 Col: 0
```



2.10 BAR PLOT/CHART

A bar chart represents categorical data with rectangular bars. Each bar has a height which corresponds to the value it represents. It is useful when we want to compare a given numeric value on different categories. It can also be used with two data series. The bars can be plotted vertically or horizontally.

A bar chart/bar graph is a very commonly-used two-dimensional data visualization made up of rectangular bars, each for a specific category, with its length representing the value of that category.

Additionally, we can also configure other characteristics for the chart, like width of the bars, colour, etc., among others. The X-axis will be a range with the same quantity of items as the Y-axis. Let us take a simple example where we will store the configurations we want in variables and then will pass them to the `bar()` function:

To make a bar chart with matplotlib, we need to use the `plt.bar()` function.

Practical Implementation-14

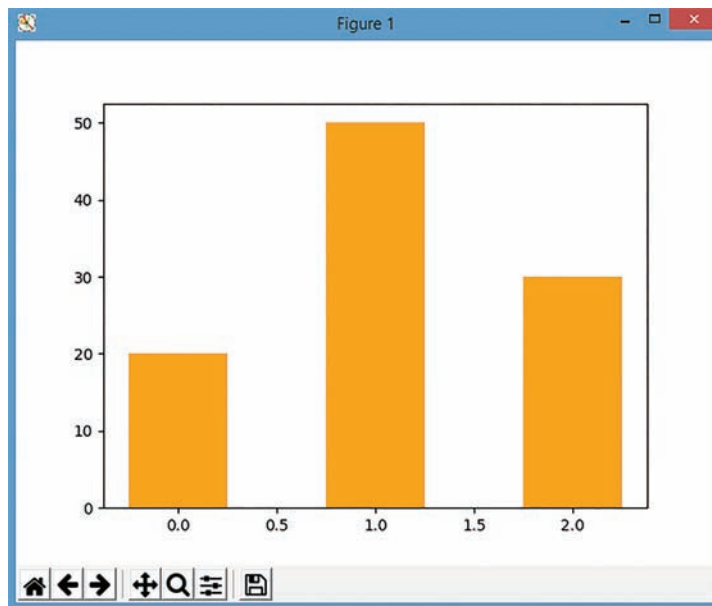
To plot a simple bar chart with orange in colour.

```

prog_bar2.py - C:/Users/preeti/AppData/Local/Programs/Python/Pyt...
File Edit Format Run Options Window Help
#To plot a simple bar chart
import matplotlib.pyplot as plt

# Variables for the bar chart
y_axis = [20,50,30]
x_axis = range(len(y_axis))
plt.bar(x_axis, y_axis, width=.5, color='orange')
plt.show()
Ln: 13 Col: 0

```



Matplotlib charts can be horizontal. To create a horizontal bar chart, type the following code as shown in Practical Implementation-15.

Practical Implementation-15

To plot a bar chart horizontally.

```

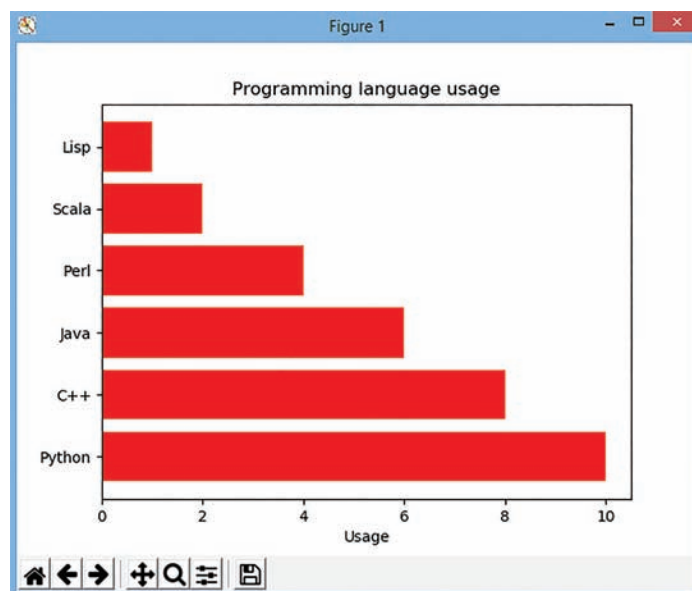
prog_bar_horizon.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-3...
File Edit Format Run Options Window Help
import numpy as np
import matplotlib.pyplot as plt

objects = ('Python', 'C++', 'Java', 'Perl', 'Scala', 'Lisp')
y_pos = np.arange(len(objects))
performance = [10,8,6,4,2,1]

plt.barh(y_pos, performance, align='center', color='r')
plt.yticks(y_pos, objects)
plt.xlabel('Usage')
plt.title('Programming language usage')

plt.show()
Ln: 17 Col: 0

```



Practical Implementation–16

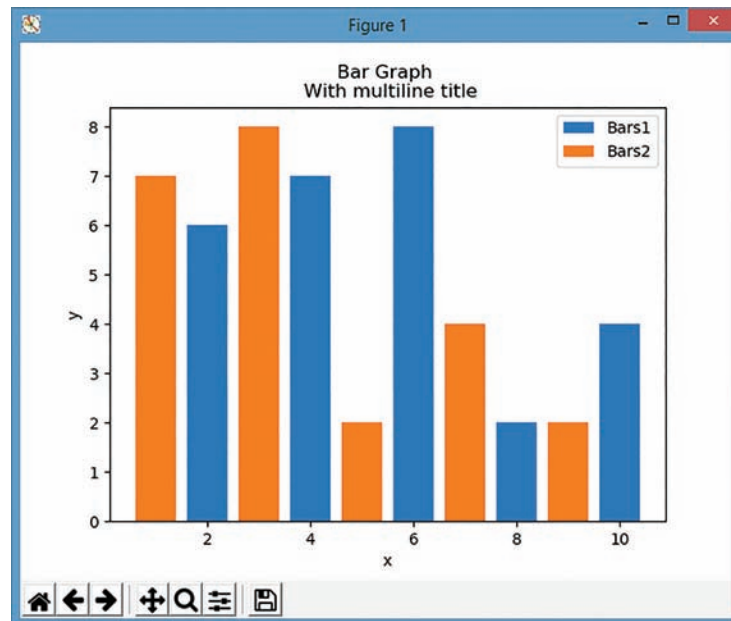
To plot the elements of two lists using a bar chart.

```

prog_bar3.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/pro...
File Edit Format Run Options Window Help
#To plot two list elements using bar chart
import matplotlib.pyplot as plt

x = [2,4,6,8,10]
y = [6,7,8,2,4]

x2 = [1,3,5,7,9]
y2 = [7,8,2,4,2]
plt.bar(x,y,label="Bars1")
plt.bar(x2,y2,label="Bars2")
plt.xlabel('x')
plt.ylabel('y')
plt.title('Bar Graph \n With multiline title')
plt.legend()
plt.show()
Ln: 18 Col: 0
    
```



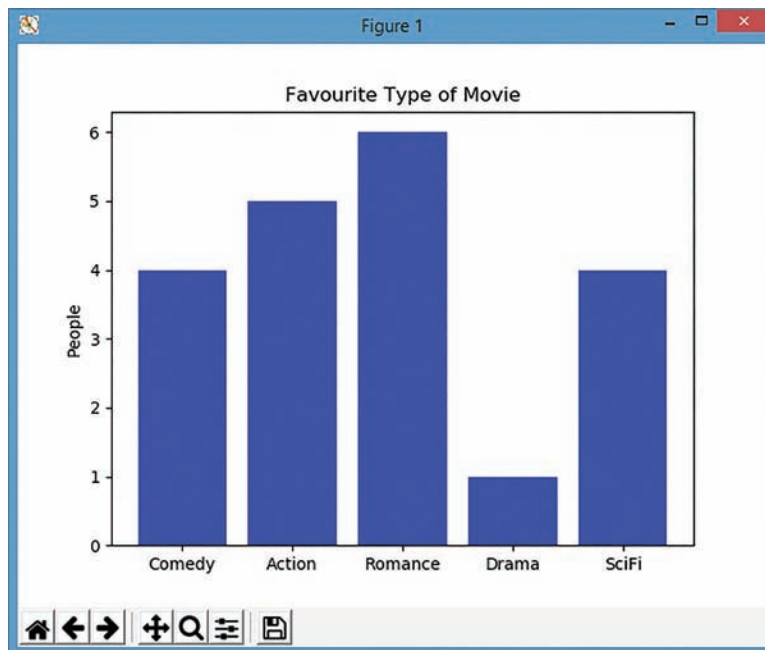
Practical Implementation–17

To plot a bar chart showing the favourite movie of different sets of people.

```

prog_movie_pyplot.py - C:/Users/preeti/AppData/Local/Programs/Python/Python3...
File Edit Format Run Options Window Help
#Program to plot a bar chart showing the choice of
#favourite movie among the people

import numpy as np
import matplotlib.pyplot as plt
objects = ('Comedy', 'Action', 'Romance', 'Drama', 'SciFi')
y_pos = np.arange(len(objects))
Types = (4,5,6,1,4)
plt.bar(y_pos, Types, align='center', color='blue')
plt.xticks(y_pos, objects) #set location and label
plt.ylabel('People')
plt.title('Favourite Type of Movie')
plt.show()
Ln: 16 Col: 0
    
```

2.11 HISTOGRAMS

A histogram is a powerful technique in data visualization. It is an accurate graphical representation of the distribution of numerical data. It was first introduced by Karl Pearson. It is an estimate of the distribution of a continuous variable (quantitative variable). It is similar to a bar graph.

In other words, we can say that histogram charts are a graphical display of frequencies, represented as bars. They show what portion of the dataset falls into each category, usually specified as non-overlapping intervals called *bins*.

To construct a histogram, the first step is to “bin” the range of values, *i.e.*, divide the entire range of values into a series of intervals and then count how many values fall into each interval. The bins are usually specified as consecutive, non-overlapping intervals of a variable. The bins (intervals) must be adjacent, and are often (but are not required to be) of equal size.

➤ Difference between a bar chart/graph and a histogram

A bar chart majorly represents categorical data (data that has some labels associated with it); they are usually represented using rectangular bars with lengths proportional to the values that they represent. Histograms, on the other hand, are used to describe distributions. Given a set of data, what are their distributions.

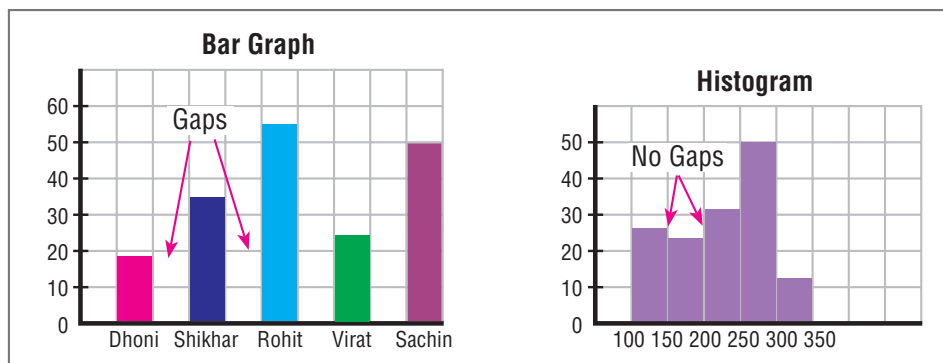


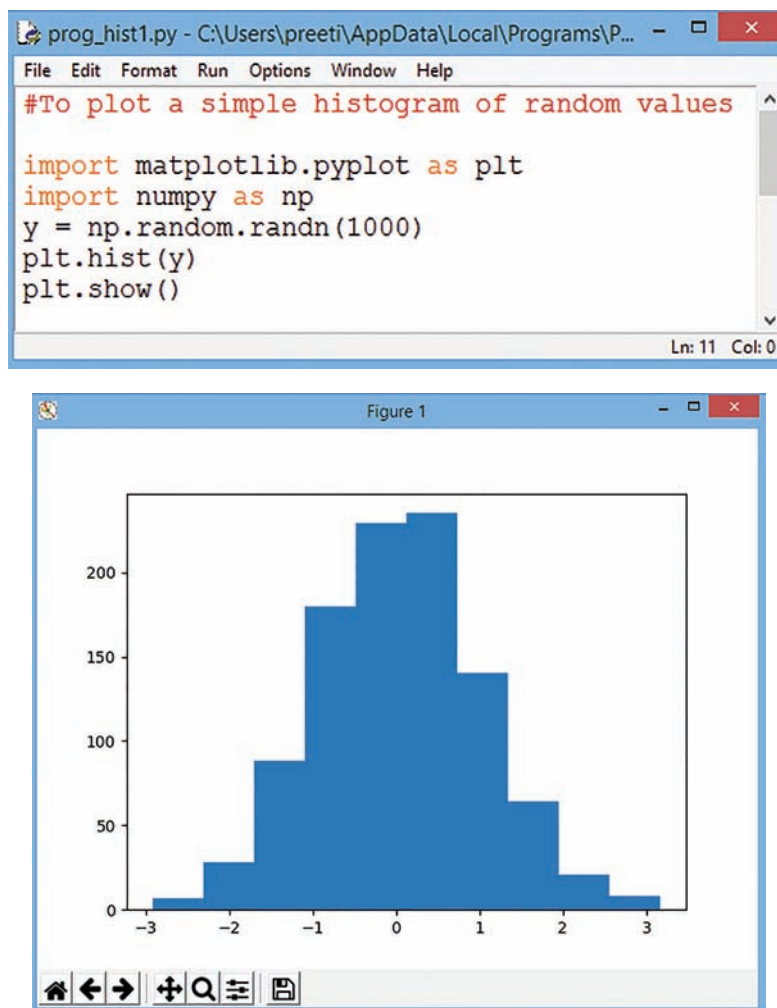
Fig. 2.3: Bar Chart vs Histogram

Drawing a histogram in Python is quite simple. **hist()** function is called for creating a histogram. Before we start drawing a histogram, a few concepts should be clear in your mind. The components of a histogram plot constitute:

- **Title:** To display heading of the histogram.
- **Colour:** To show the colour of the bar.
- **Axis:** Y-axis and X-axis.
- **Data:** The data can be represented as an array.
- **Height and width of bars:** This is determined based on the analysis. The width of the bar is called bin or intervals.
- **Border colour:** To display border colour of the bar.

Practical Implementation–18

To plot a histogram using randomly generated datasets.



Histogram plots group up values into bins of values. By default, `hist()` uses a bin value of 10 (so only ten categories, or bars, are computed), but we can customize it, either by passing an additional parameter, *for example*, in `hist(y, <bins>)`, or using the `bin` keyword argument as `hist(y, bin=<bins>)`.

The normal or Gaussian distribution is a continuous probability distribution characterized by a symmetric bell-shaped curve. A normal distribution is defined by its centre (mean) and spread (standard deviation.). The bulk of the observations generated from a normal distribution lie near the mean, which lies at the exact centre of the distribution. As a rule of thumb, about 68% of the data lies within 1 standard deviation of the mean, 95% lies within 2 standard deviations and 99.7% lies within 3 standard deviations.

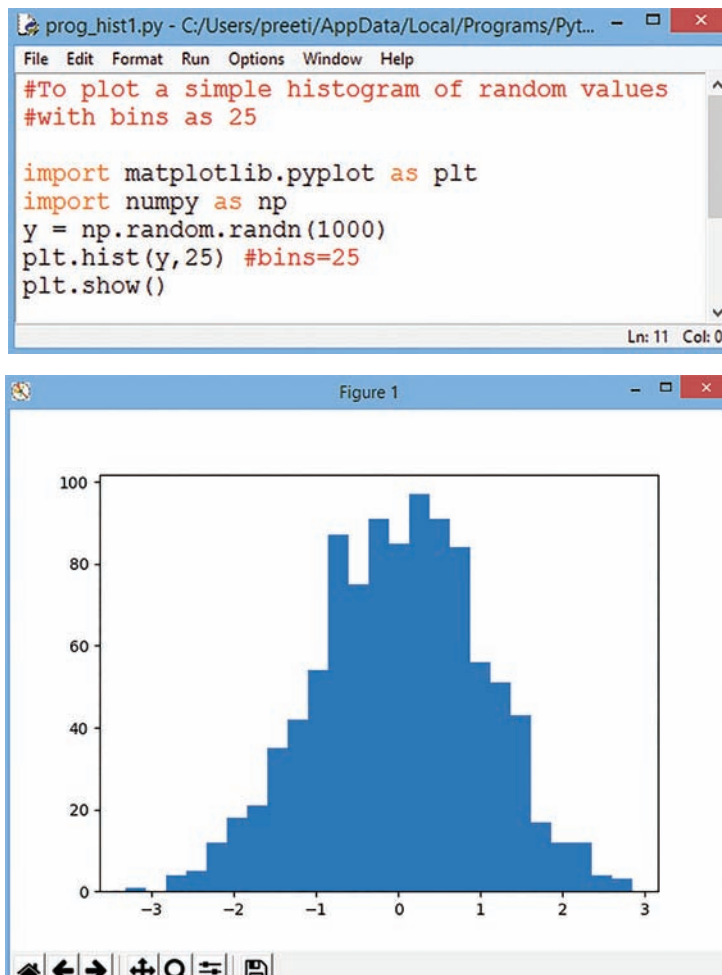
The normal distribution is perhaps the most important distribution in all of statistics. It turns out that many real world phenomena, like IQ test scores and human heights, roughly follow a normal distribution, so it is often used to model random variables. Many common statistical tests assume distributions are normal.

The above-plotted histogram is known as Normal Distribution curve, which is bell shaped. Normal distribution has a bell-shaped density curve described by its mean μ and standard deviation σ . The density curve is symmetrical, centered about its mean, with its spread determined by its standard deviation showing that data near the mean are more frequent in occurrence than data far from the mean.

The normal distribution is a form of presenting data by arranging the probability distribution of each value in the data. Most values remain around the mean value making the arrangement symmetric.

Practical Implementation-19

Replotting the previous dataset, but with bin=25



This results in a set of finer grained bars.

As evident from the above output, the edges of bars are not clear. This can be done by using attribute “edgecolor” in hist() function, as shown in the next implementation.

Histograms are a very common type of plots when we are looking at data like height and weight, stock prices, waiting time for a customer, etc., which are continuous in nature. Histogram’s data is plotted within a range against its frequency.

Practical Implementation–20

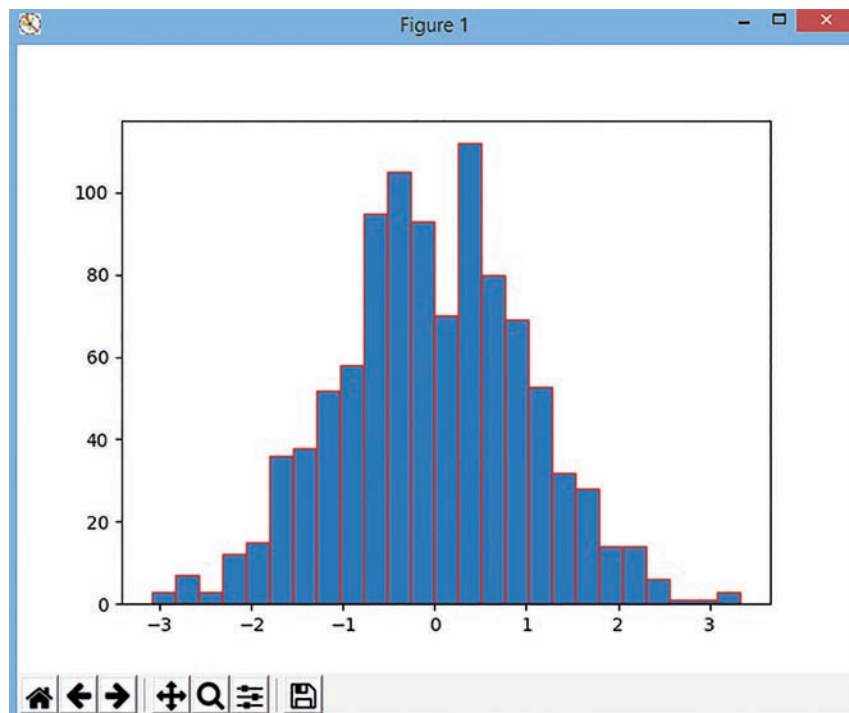
To display a histogram with well-defined edges. (Modification of Implementation-19)

```

prog_hist1.py - C:/Users/preeti/AppData/Local/Programs/P... - □ ×
File Edit Format Run Options Window Help
#To plot a simple histogram of random values
#with bins as 25

import matplotlib.pyplot as plt
import numpy as np
y = np.random.randn(1000)
plt.hist(y,25,edgecolor="red") #bins=25 and
#edges color as red
plt.show()
Ln: 12 Col: 0

```



Basically, histograms are used to represent data given in the form of some groups. X-axis is about bin ranges where Y-axis talks about frequency. So, if you want to represent age-wise population in the form of a graph, then histogram suits well as it tells you *how many* exist in certain group range or bin, if you talk in the context of histograms. This we will now implement in Practical Implementation–21 for plotting a histogram for displaying the number of students having the same weight.

Practical Implementation-21

To generate a histogram for displaying the number of students having the same weights.

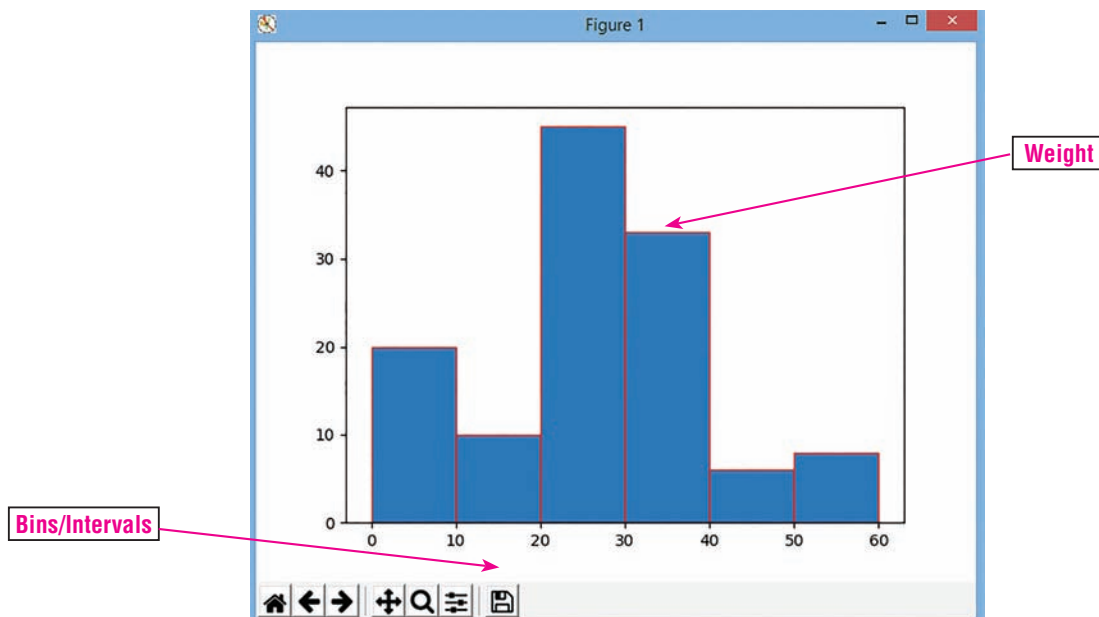
```
prog_edited_hist_studdata.py - C:\Users\preeti\AppData\Local\Programs\Python\Python37-32\prog_edited_hist... - □ ×
File Edit Format Run Options Window Help
#To plot a histogram for number of students with their respective weights
import matplotlib.pyplot as plt

data_students=[5,15,25,35,45,55]
plt.hist(data_students,bins=[0,10,20,30,40,50,60],weights=[20,10,45,33,6,8],
         edgecolor="red")
plt.show()
```

In the above program, the histogram shall display the number of students lying in the same weight range, by inputting the student data as data_students. To plot this dataset, hist() function is used. The first argument passed to hist() is the position of (x, y) coordinates where the value of weight of each student shall be displayed. The second argument calculates the bins or range of intervals on the basis of data_students. The third argument holds the respective weights.

One thing is to be kept in mind that the number of coordinates, *i.e.*, the first argument must match with the third argument, *i.e.*, number of weights to be displayed; else it will generate an error.

With show() function, the histogram generated gets displayed as shown in the output obtained.



Now, the question may arise—can we modify the above program in any way? The answer is, yes we can. In case we don't wish to plot a particular range in continuity, it can be skipped while plotting the histogram. This shall become more clear in Practical Implementation-22.

Practical Implementation–22

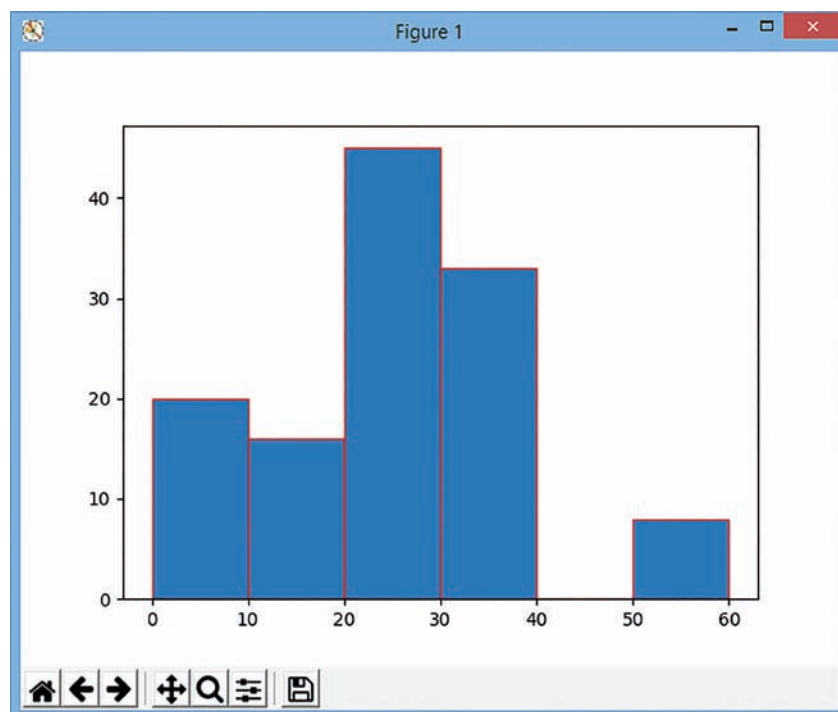
To generate a histogram for displaying the number of students having the same weights, assuming there is no student having weight in the range of 40 to 50 kgs. (Modification of Practical Implementation-21)

```

prog_edited_hist_studdata.py - C:\Users\preeti\AppData\Local\Programs\Python\Python37-32\prog_edited_hist...
File Edit Format Run Options Window Help
#To plot a histogram for number of students with no value for range 40-50

import matplotlib.pyplot as plt

data_students=[5,15,25,35,15,55]
plt.hist(data_students,bins=[0,10,20,30,40,50,60],weights=[20,10,45,33,6,8],
         edgecolor="red")
plt.show()
Ln:11 Col:0
    
```



As it is observed from the output window there is no value for the range 40 to 50. This is because in the source code, we have placed two values as 15 for the coordinates. At interval(bin) 40 to 50, no bar is displayed because we have not mentioned position from 40 to 50 in first argument (list) of hist() method, whereas in interval 10 to 20 width is being displayed as 16 (10+6 both weights are added) because 15 is twice in the first argument. As a result, one bar is skipped.

➤ Changing the Look of the Histogram

By default, bars of histogram are displayed in blue colour but you can change it as per your colour choice with the following code:

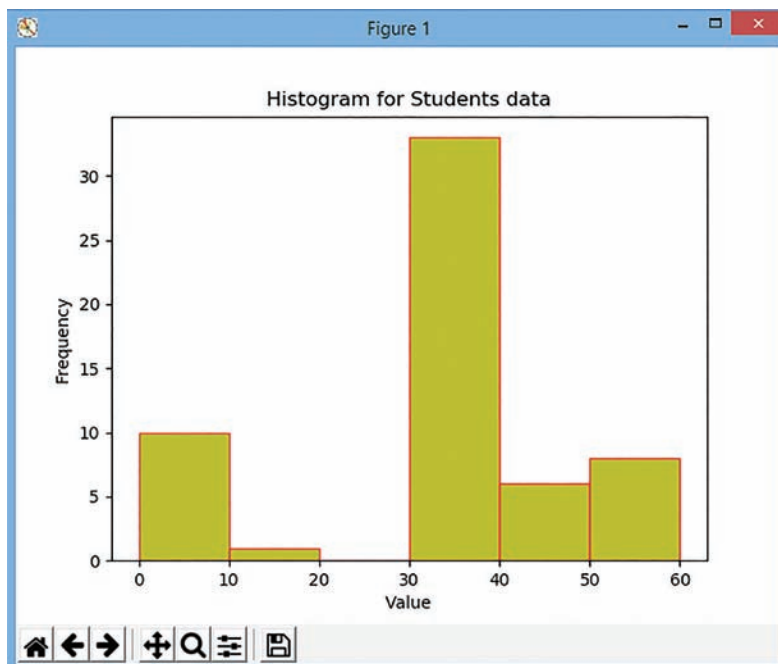
```

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")
    
```

```
prog_hist_datastud.py - C:\Users\preeti\AppData\Local\Programs\Python\Python37-32\prog_hist_datastud.py (3... - □ ×
File Edit Format Run Options Window Help
#To plot a histogram with different formatting and suitable title and labels

import numpy as np
import matplotlib.pyplot as plt

data_students=[1,11,21,31,41,51]
plt.hist(data_students,bins=[0,10,20,30,40,50,60],weights=[10,1,0,33,6,8],
         facecolor='y',edgecolor="red")
plt.title("Histogram for Students data")
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.savefig("student.png")
plt.show()
```



In the above code, we are passing 'y' as face colour means yellow colour to be displayed in bars with edges in red colour.

To give a name to the histogram, given statement is to be added before calling show()

```
plt.title("HistogramHeading")
```

For x and y labels, type the given code:

```
plt.xlabel('Value')
```

```
plt.ylabel('Frequency')
```

This will generate a histogram accordingly as shown above.

2.12 SAVING PLOTS TO FILE

The active figure can be saved to file using plt.savefig() method.

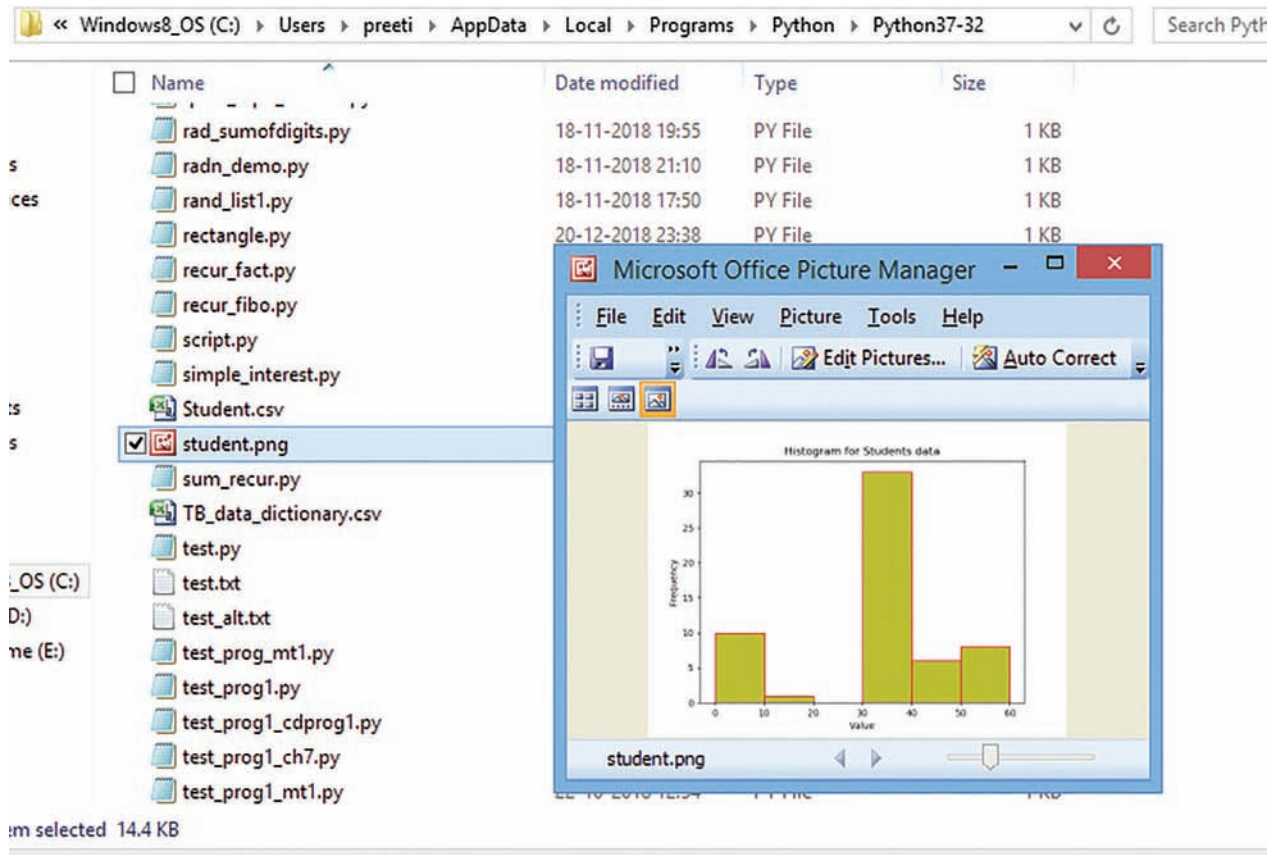
The histogram created in the above implementation can be saved by clicking on the Save button on the GUI panel. It can also be done using savefig() and typing the following code:

```
plt.savefig("student.png")
```

Also, the following code will save the histogram as a PNG image.

This statement shall save the histogram with the name as “student” and “png” extension.

On execution of the above-added code, the histogram or for that matter any plot shall be saved on the hard disk in the parent folder for Python (where Python has been installed) as shown in the screenshot given below:



2.13 FREQUENCY POLYGONS

Frequency polygons are a graphical device for understanding the shapes of distributions. They serve the same purpose as histograms, but are especially helpful for comparing sets of data. Frequency polygons are also a good choice for displaying *cumulative frequency distributions*.

In a frequency polygon, the number of observations is marked with a single point at the midpoint of an interval. A straight line then connects each set of points. Frequency polygons make it easy to compare two or more distributions on the same set of axes.

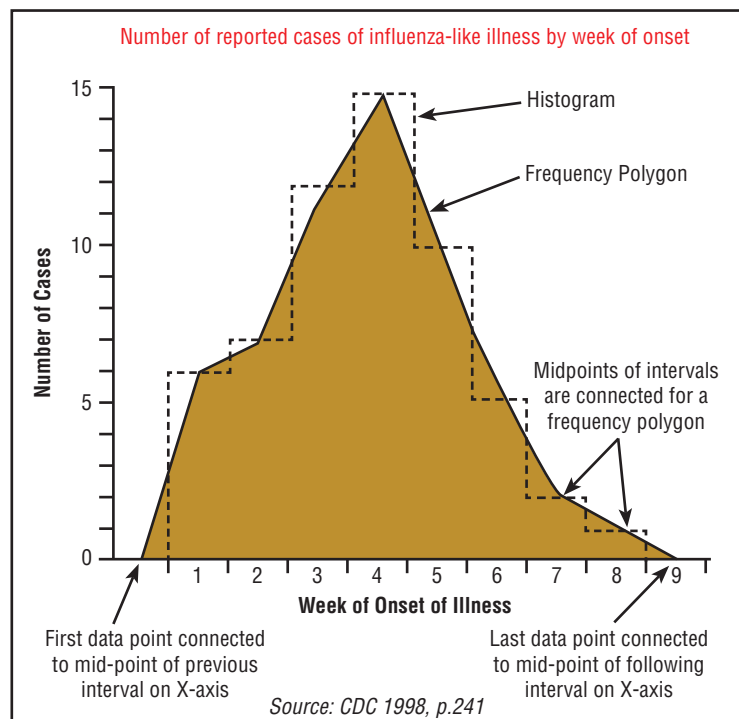


Fig. 2.4: Frequency Polygon

Let's look at an example of a frequency polygon. Creating a frequency polygon for the number of influenza cases per week.

Notice the dotted outline of a histogram for the same data. A frequency polygon smooths out the abrupt changes that may appear in a histogram, and is useful for demonstrating continuity in the variables being studied. In this example, the number of reported cases of influenza-like illness peaked during week 4 after the onset of illness.

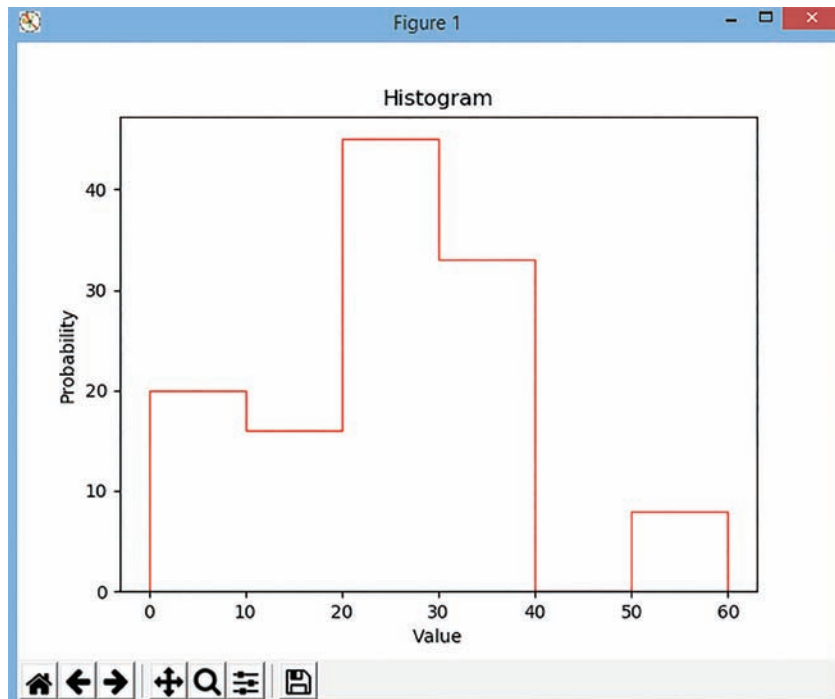
Like a histogram, frequency polygons are used to display the entire frequency distribution (counts) of a continuous variable. They must be closed at both ends because the area under the curve represents all of the data. By contrast, an arithmetic-scale line graph represents a series of observed data points (counts or rates), usually over time—it simply plots data points.

Practical Implementation-23

To generate a frequency polygon with reference to the histogram created in Practical Implementation-22.

```
prog_freq_poly.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_freq_poly.py (3.7.0)
File Edit Format Run Options Window Help
#To construct a Frequency polygon on the basis
#of student_data v/s weight histogram

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



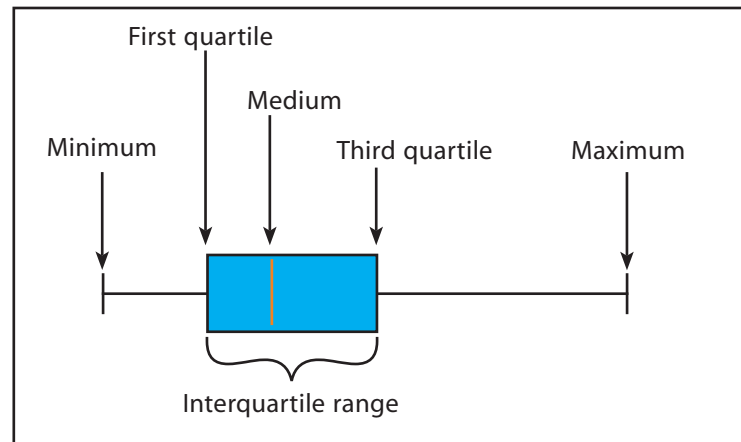
2.14 BOX PLOT

Box plots are descriptive diagrams that help compare the distribution of different series of data. They are *descriptive* because they show measures (e.g., the *median*) which do not assume an underlying probability distribution.

A Box plot is the visual representation of the statistical five-number summary of a given dataset.

A Five-Number Summary includes:

- Minimum
- First Quartile (Q1)
- Median (Second Quartile-Q2)
- Third Quartile (Q3)
- Maximum



➤ What is a Box plot

A box plot, also called a box and whisker plot, is a way to show the **spread** and **centres** of a dataset. **Measures of spread** include the **interquartile range** and the **mean** of the dataset. Measures of centre include the mean or **average** and **median** (the middle of a dataset).

The box and whisker chart shows you how your data is spread out.

Five pieces of information (the “five-number summary”) are generally included in the chart:

- The minimum (the smallest number in the dataset). The minimum is shown at the far left of the chart, at the end of the left “whisker.”
- First quartile, Q1, is the far left of the box (or the far right of the left whisker).
- The median is shown as a line in the centre of the box.
- Third quartile, Q3, shown at the far right of the box (at the far left of the right whisker).
- The maximum (the largest number in the dataset), shown at the far right of the box.

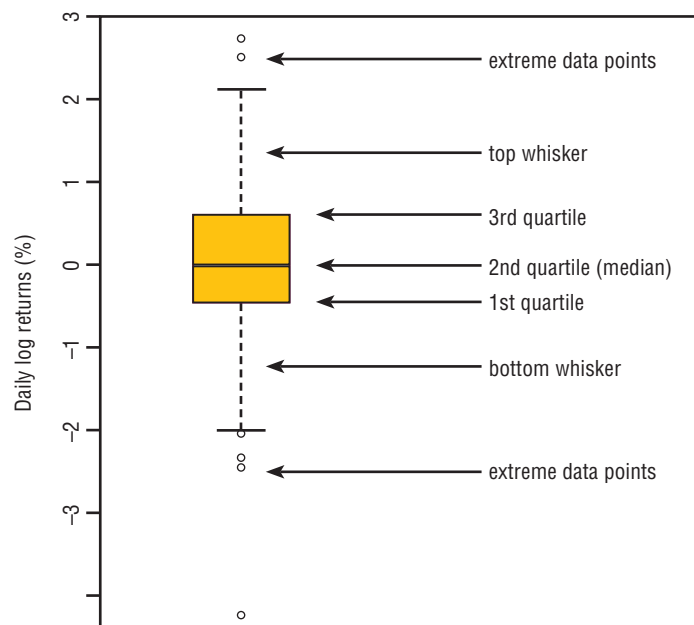
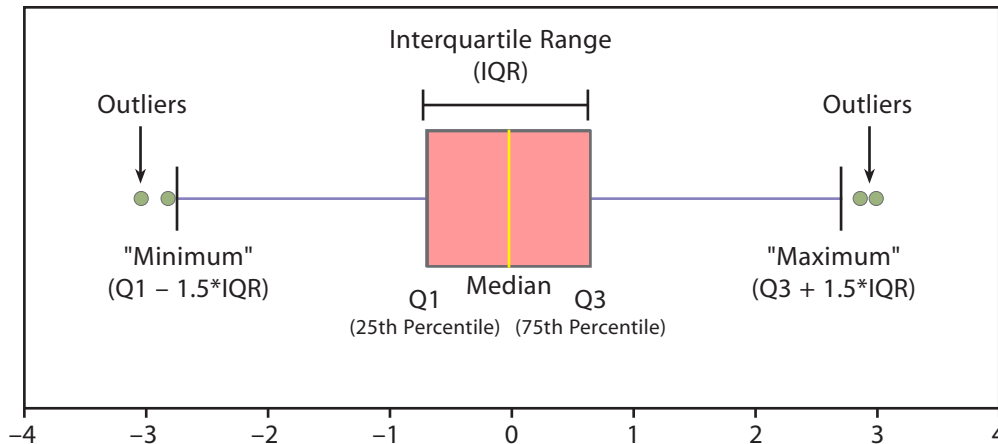


Fig. 2.5: Representation of a Quartile

Thus, in a nutshell, the various components can be explained as:



- **median (Q2/50th Percentile):** the middle value of the dataset.
- **first quartile (Q1/25th Percentile):** the middle number between the smallest number (not the “minimum”) and the median of the dataset.
- **third quartile (Q3/75th Percentile):** the middle value between the median and the highest value (not the “maximum”) of the dataset.
- **interquartile range (IQR):** 25th to the 75th percentile.
- **whiskers (shown in blue)**
- **outliers (shown as green circles)**
- **“maximum”:** $Q3 + 1.5 * IQR$
- **“minimum”:** $Q1 - 1.5 * IQR$

➤ What is Quartile

In the above description, an important term which is to be discussed most importantly is “quartile” which has been derived or taken from the word “quantile”.

The word “quantile” finds its origin in the word quantity which means a quantile is where a sample is divided into equal-sized subgroups (that’s why it’s sometimes called a “fractile”). It can also refer to dividing a probability distribution into areas of equal probability.

The median is a kind of quantile; the median is placed in a probability distribution at the centre so that exactly half of the data is lower than the median and half of the data is above the median. The median cuts a distribution into two equal parts and that is why sometimes it is called 2-quantile.

So we can say that quartiles are the quantiles which divide the distribution of data into four equal parts. Deciles are quantiles that divide a distribution into 10 equal parts while percentiles are quantiles that divide a distribution into 100 equal parts.

➤ How to Find Quartiles

Q1: Find the number in the following set of data where 30 per cent of values fall below it and 70 per cent fall above it:

2 4 5 7 9 11 12 17 19 21 22 31 35 36 45 44 55 68 79 80 81 88 90 91 92 100 112 113 114 120 121 132 145 148 149 152 157 170 180 190

- Step 1:** Order the data from smallest to largest. The data in question is already in ascending order.
- Step 2:** Count how many observations you have in your dataset. This particular dataset has 40 items.
- Step 3:** Convert any percentage to a decimal for “q”. We are looking for a number where 30 per cent of the values fall below it, so convert that to .3.
- Step 4:** Insert your values into the formula:
 $i^{\text{th}} \text{ observation} = q (n + 1)$
 $i^{\text{th}} \text{ observation} = .3 (40 + 1) = 12.3$
- Answer:** The i^{th} observation is at 12.3, so we round off to 12 (remembering that this formula is an estimate). The 12th number in the set is 31, which is the number below which 30 per cent of the values fall.

In a nutshell, it can be interpreted as:

- **The minimum and the maximum** are just the min and max values from our data.
- **The median** is the value that separates the higher half of a data from the lower half. It is calculated by the following steps: **order your values** and find the **middle one**. In a case when our count of values is even, we actually have 2 middle numbers, so the median here is calculated by summing these 2 numbers and dividing the sum by 2. *For example*, if we have the numbers 1, 2, 5, 6, 8, 9, the **median** will be $(5 + 6) / 2 = 5.5$.
- **The first quartile** is the **median** of the **data values to the left** of the median in our ordered values. *For example*, if we have the numbers 1, 3, 4, 7, 8, 8, 9, the first quartile is the median from the 1, 3, 4 values, so it is 3.
- **The third quartile** is the **median** of the **data values to the right** of the median in our ordered values. *For example*, if we use the numbers 1, 3, 4, 7, 8, 8, 9 again, the third quartile is the median from the 8, 8, 9 values, so it is 8.
- The **IQR (Interquartile Range)** approximates the amount of spread in the middle 50% of the data. The **formula** is the **third quartile – the first quartile**.
- This type of plot can also show **outliers**. An outlier is a data value that lies **outside the overall pattern**. They are visualized as **circles**. When we have outliers, the **minimum** and the **maximum** are visualized as the min and the max values from the values which aren't outliers. There are many ways to identify what an outlier is. A commonly used rule says that a value is an outlier if it's **less than the first quartile – 1.5 * IQR** or **higher than the third quartile + 1.5 * IQR**.

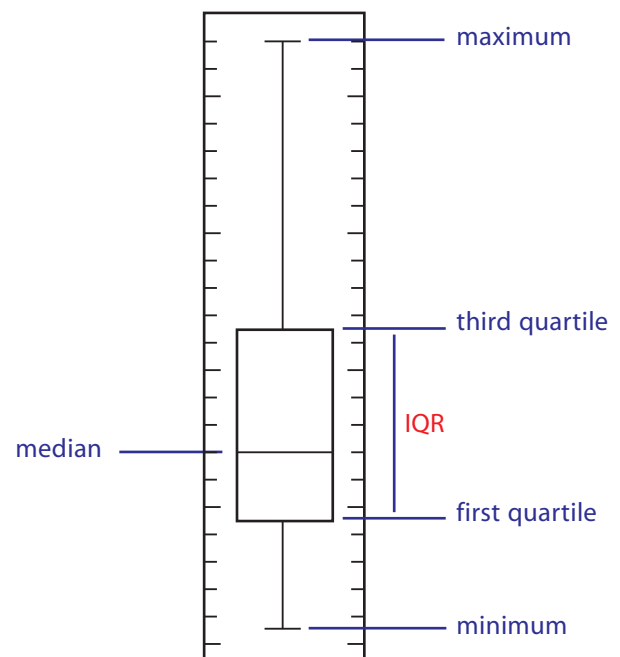
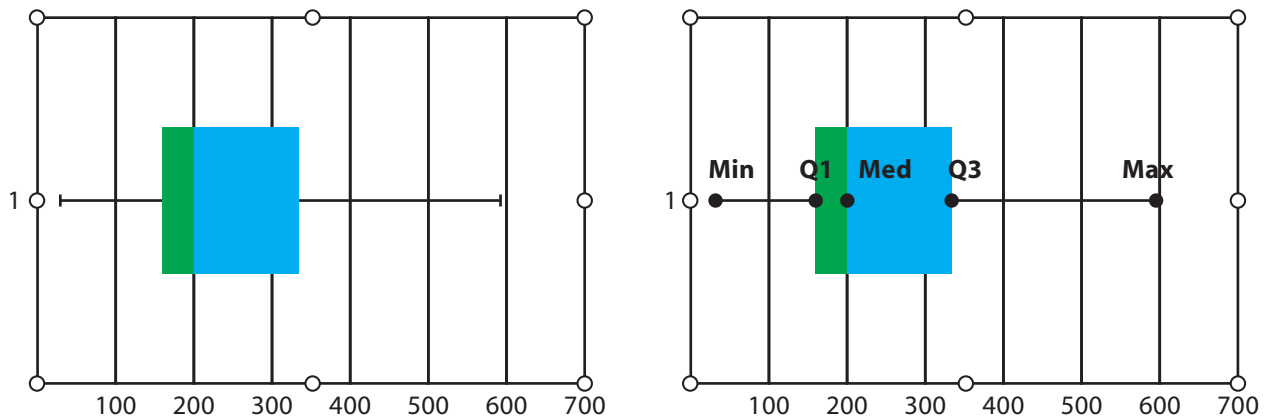


Fig. 2.6: Components of a Box Plot

➤ How to Read a Box Plot

A box plot is a way to show a five-number summary in a chart. The main part of the chart (the “box”) shows where the middle portion of the data is: the interquartile range. **The Interquartile Range or IQR is the distance between the Upper and Lower Quartile.** At the ends of the box, you find the first quartile (the 25% mark) and the third quartile (the 75% mark). The far left of the chart (at the end of the left “whisker”) is the minimum (the smallest number in the set) and the far right is the maximum (the largest number in the set). Finally, the median is represented by a vertical bar in the centre of the box. All these components to be read can be described as an “artist”.

For example, given below is a box plot. It can be read and summarized as:



The above image shows a box and whiskers chart with the following information:

- Minimum: 20
- Q_1 : 160
- Median: 200
- Q_3 : 330
- Maximum: 590

Box plots aren’t used that much in real life. However, they can be a useful tool for getting a quick summary of data.

The box plot is generated by using **boxplot()** method, with the data set values, to be passed as an argument to it.

Practical Implementation-24

To plot a simple box plot by passing list of lists.

```
list1 = [43,76,34,63,56,82,87,55,64,87,95,23,14,65,67,25,23,85]
```

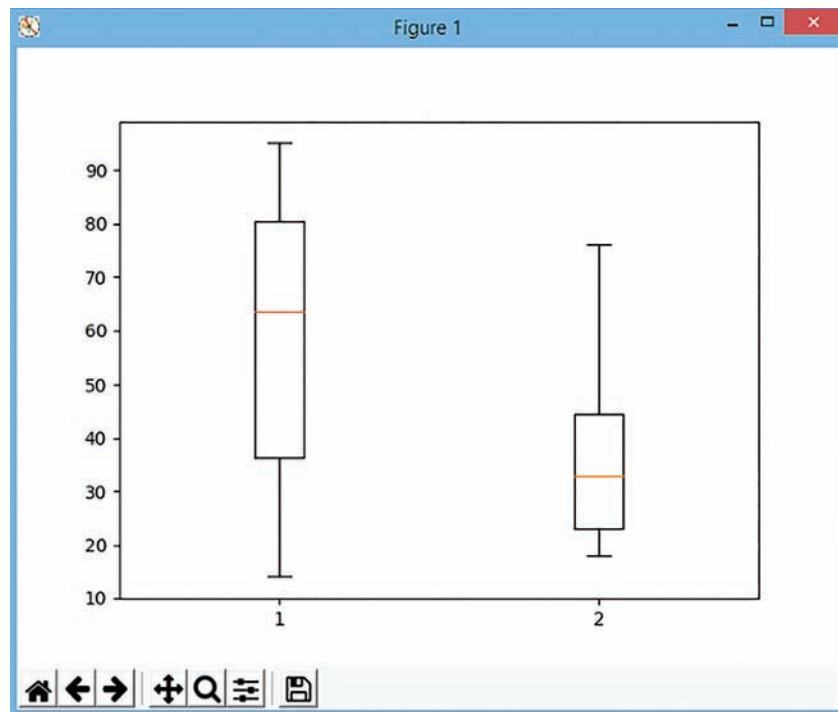
```
list2 = [34,45,34,23,43,76,26,18,24,74,23,56,23,23,34,56,32,23]
```

```
prog_boxplot.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_boxplo... - □ ×
File Edit Format Run Options Window Help
#To plot a simple boxplot

import matplotlib.pyplot as plt

list1 = [43,76,34,63,56,82,87,55,64,87,95,23,14,65,67,25,23,85]
list2 = [34,45,34,23,43,76,26,18,24,74,23,56,23,23,34,56,32,23]
data = [list1,list2] #Passing nested lists (lists inside a list)

plt.boxplot(data)
plt.show()
```



Practical Implementation-25

To plot a box plot for the frequency of marks obtained in each of the four courses offered by “Vidhyarthi University”.

```
value1=[72,76,24,40,57,62,75,78,31,32]
```

```
value2=[62,5,91,25,36,32,96,95,30,90]
```

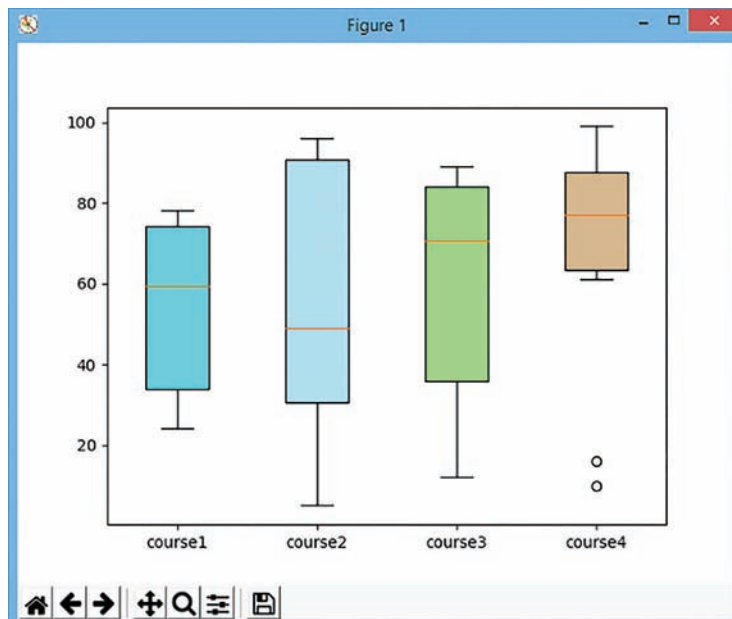
```
value3=[23,89,12,78,72,89,25,69,68,86]
```

```
value4=[99,73,70,16,81,61,88,98,10,87]
```

```

prog_boxplot1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_boxpl...
File Edit Format Run Options Window Help
#To plot a boxplot for frequency distribution of marks

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()
Ln: 17 Col: 0
    
```



- In the above code, four lists are passed to another list variable named `box_plot_data` and it is plotted with `boxplot` function. `boxplot()` function takes the data array to be plotted as input in the first argument, the second argument `vert=1` signifies that it is a vertical plot; the third argument `patch_artist=True` fills the box plot and the fourth argument takes the label to be plotted.

A patch is a 2D artist with a face color and an edge color. When we set the attribute `patch_artist` as `True`, it fills the box plot with colour, else if it is set to `False`, it will produce boxes with a `2DLine` without any colour fill.

The next statement defines a list of colours to be filled in the respective boxes. The number of colours in the list should match the number of boxes to be filled. The next statement (for loop) has been given for iterations to be carried for filling each box of the box plot.

- Colors list takes up four different colours and is passed to four different boxes of the box plot with `patch.set_facecolor()` function. This argument sets the patch face color.

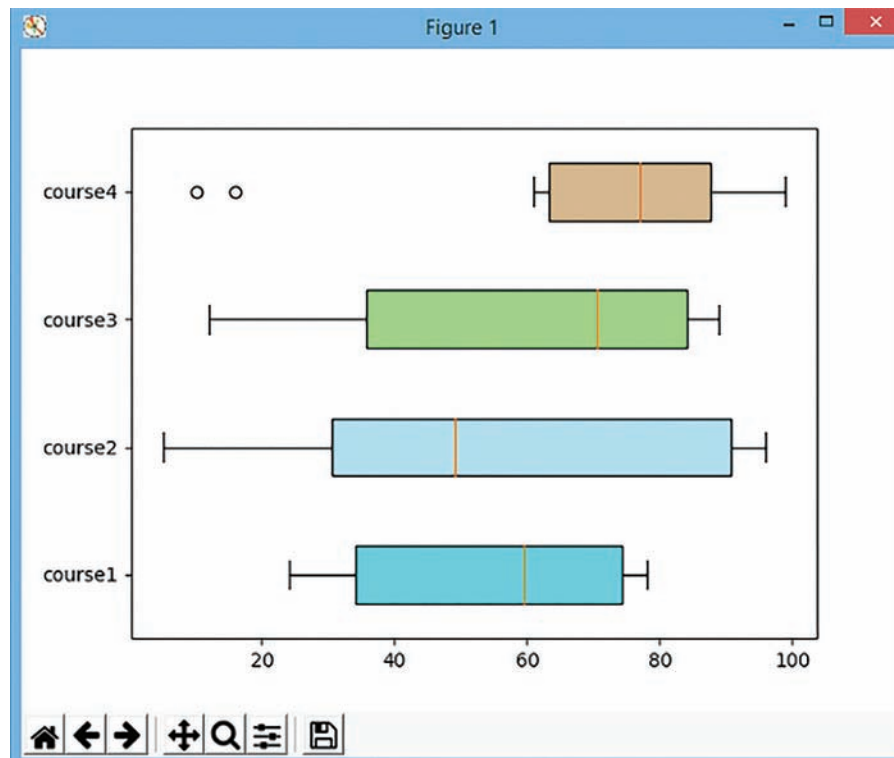
Note: if `vert=0` in `boxplot()` is set, then horizontal box plots will be drawn.

```

prog_boxplot1.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_boxpl...
File Edit Format Run Options Window Help
#To plot a boxplot for frequency distribution of marks

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=0,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()
Ln: 9 Col: 36

```



In the above code, if we add a new argument “notch”, then it will result in the creation of another type of box plot known as “Notch plot”.

```

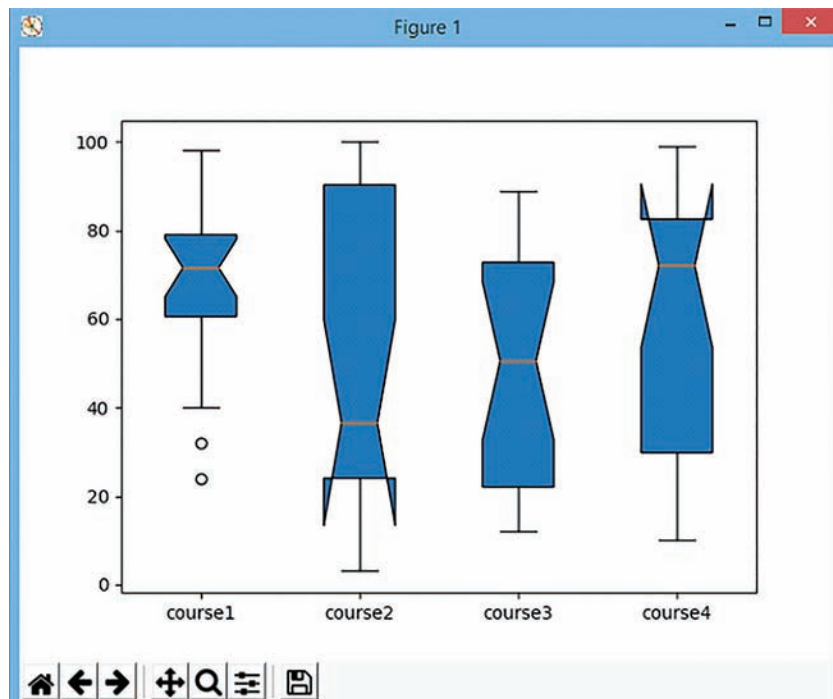
prog_notchboxplot.py - C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_notchbo...
File Edit Format Run Options Window Help
#To plot a Notch boxplot
import matplotlib.pyplot as plt

value1 = [82,76,24,40,67,62,75,78,71,32,98,89,78,67,72,82,87,66,56,52]
value2=[62,5,91,25,36,32,96,95,3,90,95,32,27,55,100,15,71,11,37,21]
value3=[23,89,12,78,72,89,25,69,68,86,19,49,15,16,16,75,65,31,25,52]
value4=[59,73,70,16,81,61,88,98,10,87,29,72,16,23,72,88,78,99,75,30]

box_plot_data=[value1,value2,value3,value4]
plt.boxplot(box_plot_data,notch='True',patch_artist=True,
            labels=['course1', 'course2', 'course3', 'course4'])
plt.show()
Ln: 15 Col: 0

```

As can be observed from the code, `boxplot()` function takes the data array to be plotted as input in the first argument and the second argument `notch='True'` creates the notch format of the box plot. The third argument `patch_artist=True` fills the box plot with colour while the fourth argument takes the label to be plotted and, as a result, the output in the form of a “NOTCH” type box plot is obtained as shown on the next page.



NOTCH is a logical attribute for `boxplot()` method. NOTCH means narrowing of the box around the median. If TRUE, it creates a notched box plot. The notch displays a confidence interval around the median which is normally based on the median ($\pm 1.58 * IQR / \sqrt{n}$). Notches are used to compare groups; if the notches of two boxes do not overlap, this signifies that the medians differ. The width of the notches is proportional to the Interquartile Range (IQR) of the sample and inversely proportional to the square root of the size of the sample.

In the above program code, by setting the argument for NOTCH attribute as True, it will narrow down the range of given values (inputted list) towards the median value and, hence, a notch-shaped box plot is created.



MEMORY BYTES

- In Python, we can use two exclusive libraries for visualization, commonly known as *matplotlib* and *seaborn*.
- Matplotlib is a Python package for 2D plotting that generates production-quality graphs.
- The aim of matplotlib is to generate graphs.
- We can add information to the plots such as legends, axis labels and titles.
- It is required to import the main matplotlib sub-module for plotting pyplot.
- A **plot** is a graphical technique for representing a data set, usually as a **graph**, showing the relationship between two or more variables.
- Matplotlib is a package with a collection of command style functions that makes it work. It is a programming platform designed specifically for engineers and scientists which allows the most natural expression of computational mathematics.
- Each pyplot function makes some change to a figure, *e.g.*, creating a figure, creating a plotting area in a figure, plotting some lines in a plotting area, decorating the plot with labels, etc.
- **plot()** is a versatile command and takes an arbitrary number of arguments.
- In **bar charts**, each column represents a group defined by a categorical variable.

- Seaborn is a library for creating informative and attractive statistical graphics in Python. This library is based on Matplotlib. Seaborn offers various features such as built-in themes, color palettes, functions and tools to visualize univariate, bivariate, linear regression, matrices of data, statistical time series, etc., which let us build complex visualizations.
- A line chart or line graph is a type of chart which displays information as a series of data points called ‘markers’ connected by straight line segments.
- For the bar chart, we use the bar() function, where we define the position of the bars on the X-axis and their height on the Y-axis.
- You can change line colour, width, line-style, marker-type, marker-colour, marker-size in plot() function.
- The scatter chart is a graph of plotted points on two axes that show the relationship between two sets of data.
- You can create scatter charts using either plot() function or scatter() function.
- The axes can be labelled using xlabel() and ylabel() functions.
- The title() function adds title to the plot.
- Using legend() function, one can add legends to a plot where multiple data ranges have been plotted, but before that the data ranges must have their label argument defined in plot() or bar() function.
- A histogram provides a visual interpretation of numerical data by showing the number of data points that fall within a specified range of values (called bins).
- Pyplot module’s hist() lets you create histograms.
- A frequency polygon is a type of frequency distribution graph.
- In a frequency polygon, the number of observations is marked with a single point at the midpoint of an interval.
- The box plot is used to show the range and the middle half of the ranked data.
- The boxplot() of pyplot lets you draw box plots.

OBJECTIVE TYPE QUESTIONS

1. Fill in the blanks.

- (a) refers to the graphical or visual representation of information and data using visual elements like charts, graphs and maps, etc.
- (b) is a collection of methods with matplotlib library which allows the user to construct 2D plots easily and interactively.
- (c) The chart is a graph of plotted points on two axes that shows the relationship between two sets of data.
- (d) is the text that appears on the top of the plot and defines what the chart is about.
- (e) The axes of a plot can be labelled using and functions.
- (f) A is a summarization tool for discrete or continuous data.
- (g) Pyplot module’s lets you create histograms.
- (h) In a, the number of observations is marked with a single point at the midpoint of an interval.
- (i) The is used to show the range and the middle half of the ranked data.
- (j) The module of Pyplot lets you draw box plots.
- (k) Two functions of Pyplot library used to create scatter charts are and
- (l) Barh() function is used to create bar chart.
- (m) The area on which actual plot will appear is defined by
- (n) describe the number of data points that fall within a specified range of values.
- (o) To change the orientation of the histogram, we can use argument with hist().

- Answers:** (a) Data Visualization (b) Pyplot (c) scatter
 (d) Title (e) xlabel(), ylabel() (f) histogram
 (g) hist() (h) frequency polygon (i) box plot
 (j) boxplot() (k) plot(), scatter() (l) horizontal
 (m) axes (n) Bins (o) orientation

2. State whether the following statements are True or False.

- (a) The matplotlib is a Python interface.
- (b) To save the plot, we have to use save_graph() function.
- (c) Frequency polygons are drawn with respect to the histogram created.
- (d) The box plot is also described as five-number summary plot.
- (e) In box plot, the highest and lowest scores are not joined to the box by straight lines.
- (f) Plot can be saved in a pdf format.
- (g) We can specify different colours for different bars of a bar chart.
- (h) To use Pyplot for data visualization, we have to import it by giving import command for matplotlib.pyplot.
- (i) Pyplot is a Python library.
- (j) Frequency polygon is a type of frequency distribution graph.
- (k) To specify a common width for all bars in a bar graph, we have to use thick argument.
- (l) To add a Title to the plot, we have to call function header().
- (m) Markers are data points in the graphs.
- (n) Line style argument of plot() function is not required in scatter chart.
- (o) When we don't specify X or Y limits for a plot, then Pyplot does not automatically decide limits as per values being plotted.

- Answers:** (a) False (b) False (c) True (d) True (e) False (f) True
 (g) True (h) True (i) False (j) True (k) False (l) False
 (m) True (n) True (o) False

3. Multiple Choice Questions (MCQs)

- (a) Which Python package is used for 2D graphics?
 - (i) matplotlib.pyplot (ii) matplotlib.pip
 - (iii) matplotlib.numpy (iv) matplotlib.plt
- (b) The most popular data visualization library in Python is:
 - (i) pip (ii) matinfolib (iii) matplotlib (iv) matpiplib
- (c) Matplotlib allows you to create:
 - (i) table (ii) charts (iii) maps (iv) infographics
- (d) Which of the following is not a visualization under matplotlib?
 - (i) Scatter plot (ii) Histogram (iii) Box plot (iv) Table plot
- (e) Which plot displays the distribution of data based on the five-number summary?
 - (i) Scatter plot (ii) Line plot (iii) Box plot (iv) Chart plot
- (f) Which of the following commands is used to install matplotlib for coding?
 - (i) import plt.matplotlib as plot (ii) import plot.matplotlib as pt
 - (iii) import matplotlib.plt as plot (iv) import matplotlib.pyplot as plt
- (g) Which of the following methods should be employed in the code to display a plot()?
 - (i) show() (ii) display() (iii) execute() (iv) plot()
- (h) Which of the following statements is used to create a histogram of 'step' type with 20 bins?
 - (i) plt.hist(x, bins=20, histype="barstacked") (ii) plt.hist(x, bins=20)
 - (iii) plt.hist(x, bins=20, histype="step") (iv) plt.hist(x, bins=20, histype=hist())
- (i) Which of the following plots makes it easy to compare two or more distributions on the same set of axes?
 - (i) Box plot (ii) Histogram (iii) Frequency Polygon (iv) Bar chart

- (j) The part of chart which identifies different sets of data plotted on plot by using different colours is called:
 (i) legends (ii) title (iii) axes (iv) figure
- (k) Which of the following is an incorrect example of savefig() function?
 (i) plt.savefig("bar1.pdf") (ii) plt.savefig("bar1.png")
 (iii) plt.savefig("bar1.eps") (iv) plt.savefig("bar1.ppt")
- (l) Which of the following plots makes it easy to compare two or more distributions on the same set of axes?
 (i) Box plot (ii) Histogram (iii) Frequency polygon (iv) Bar chart
- (m) Consider the snippet given below:

```
import matplotlib.pyplot as plt
#arr1,arr2 defined here
colors=['r','b','k','g','m']
sizes=[50,120,220,150,80]
plt.scatter(arr1,arr2, c=colors, s=sizes, marker="s")
```

 With reference to the above code, what will be the shape of marker?
 (i) square (ii) circle (iii) star (iv) diamond
- (n) With reference to the code in (m), what will be the colour of the last point?
 (i) Red (ii) Blue (iii) Black (iv) Magenta
- (o) The scatter() function:
 (i) is a powerful method of creating scatter plots than plot() function
 (ii) can create line graph
 (iii) can create bar graph
 (iv) None of the above

Answers: (a) (i) (b) (iii) (c) (ii) (d) (iv) (e) (iii) (f) (iv)
 (g) (i) (h) (iii) (i) (iii) (j) (i) (k) (iv) (l) (iii)
 (m) (i) (n) (iv) (o) (i)

SOLVED QUESTIONS

1. What is Python matplotlib?

Ans. **matplotlib.pyplot** is a plotting library used for 2D graphics in Python programming language. It can be used in Python scripts, shell, web application servers and other graphical user interface toolkits.

2. How can we install matplotlib?

Ans. It is easy to install Python matplotlib library with pip statement – pip install matplotlib.

3. What are the various types of plots offered by matplotlib?

Ans. Matplotlib offers several types of plots:

- Line Graph
- Bar Graph
- Histogram
- Scatter Plot
- Area Plot
- Pie Chart

4. What is data visualization? What is its significance?

Ans. Data visualization is a general term that describes any effort to help people understand the significance of data by placing it in a visual context. In simple words, Data visualization is the process of displaying data/information in graphical charts, figures and bars.

5. Name the functions you will use to create a (i) line chart, (ii) bar chart, (iii) scatter chart.

Ans. (i) matplotlib.pyplot.plot()
 (ii) matplotlib.pyplot.bar()
 (iii) matplotlib.pyplot.plot() and matplotlib.pyplot.scatter()

6. Mr. Sanjay wants to plot a bar graph for the given set of values of subject on x-axis and number of students who opted for that subject on y-axis. [CBSE Sample Paper 2020]

Complete the code to perform the following:

- (i) To plot the bar graph in statement 1
- (ii) To display the graph in statement 2

```
import matplotlib.pyplot as plt
x=['Hindi', 'English', 'Science', 'SST']
y=[10,20,30,40]
```

_____ Statement 1

_____ Statement 2

- Ans.** (i) `plt.bar(x, y)`
(ii) `plt.show()`

7. Mr. Harry wants to draw a line chart using a list of elements named LIST. Complete the code to perform the following operations:

- (i) To plot a line chart using the given LIST,
- (ii) To give a y-axis label to the line chart named "Sample Numbers".

```
import matplotlib.pyplot as PLINE
LIST=[10,20,30,40,50,60]
```

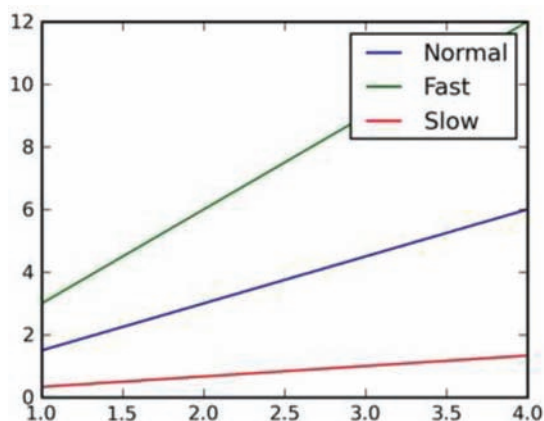
_____ Statement 1

_____ Statement 2

```
PLINE.show()
```

- Ans.** (i) `PLINE.plot(LIST)`
(ii) `PLINE.ylabel("Sample Numbers")`

8. Write a code to plot the speed of a passenger train as shown in the figure given below:

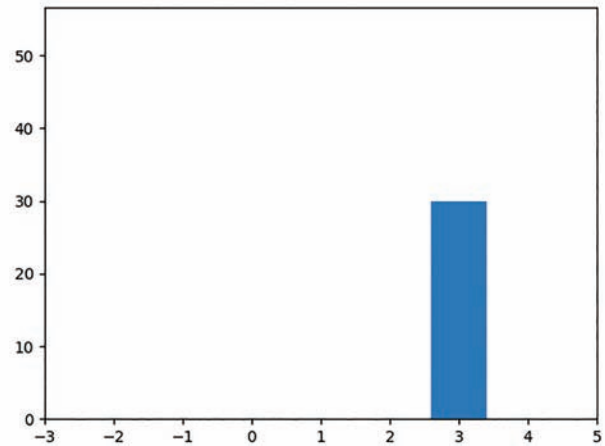


- Ans.**

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1, 5)
plt.plot(x, x*1.5, label='Normal')
plt.plot(x, x*3.0, label='Fast')
plt.plot(x, x/3.0, label='Slow')
plt.legend()
plt.show()
```

9. Consider the code given below (all required libraries are imported) and the output produced by it. Why is the chart showing one bar only while we are plotting four values on the chart?

```
a = [3, 6, 9, 12]
b = [30, 48, 54, 48]
plt.xlim(-3, 5)
plt.bar(a, b)
plt.show()
```



Ans. The given chart is showing a single bar as the limits of X-axis have been set as -3 to 5 . On this range, only one value from the data range being plotted falls, *i.e.*, only $a[0]$ and $b[0]$ fall on this range. Thus, only a single value $b[0]$, *i.e.*, 30 is plotted against $a[0]$, *i.e.*, 3 .

10. What changes will you make to the code of previous question so that the bars are visible for all four points? But do keep in mind that the X-axis must begin from the point -3 .

Ans. If we change the limits of X-axis so that all the points being plotted fall in the range of limits, all values will show. Thus, we have changed the limits from $=3$ to 15 , in place of -3 to 5 .

```
plt.xlim(-3, 15)
plt.bar(a, b)
plt.show()
```

11. Why is the following code not producing any result? Why is it giving error? (Note that all required libraries have been imported and are available)

```
a = range(10, 50, 12)
b = range(90, 200, 20)
matplotlib.pyplot.plot(a, b)
```

Ans. The above code is producing error because the two sequences being plotted, *i.e.*, a and b do not match in shape. While sequence ‘ a ’ contains 4 elements ($10, 22, 34$ and 46), sequence ‘ b ’ contains 6 elements ($90, 110, 130, 150, 170$ and 190). For plotting, it is necessary that the two sequences being plotted must match in their shape.

12. What modification is required to rectify the error in previous question code?

Ans. Since both the sequences being plotted must match in their shape, we can achieve this either by adding two elements to sequence a so that it has the same shape as sequence b (*i.e.*, 6 elements) or by removing two elements from sequence b so that it matches the shape of sequence a (*i.e.*, 4 elements)

For instance,

```
a = range(10, 50, 12)
b = range(90, 160, 20)
matplotlib.pyplot.plot(a, b)
```

13. What is a scatter chart? How is it different from a line chart?

Ans. The scatter chart is a graph of plotted points that show the relationship between two sets of data. With a scatter plot a mark or marker (usually a dot or small circle), represents a single data point. With one mark (point) for every data point a visual distribution of the data can be seen. Depending on how tightly the points cluster together, you may be able to discern a clear trend in the data.

The difference is that with a scatter plot, the decision is made from the data points such that the individual points should not be connected directly together with a line but, instead express a trend.

14. What is a histogram? How is it useful?

Ans. A histogram is a statistical tool used to summarize discrete or continuous data. It provides a visual interpretation of numerical data by showing the number of data points that fall within a specified range of values (called “bins”).

15. Given below is the data about several airlines and flight arrival delay.

	Flights_arr_delay	Name
0	11.0	Air India
1	20.0	SpiceJet Ltd.
2	33.0	Jet Airways
3	-18.0	Indigo
4	-25.0	Air India
5	12.0	GoAir
6	19.0	JetAirways
7	-14.0	SpiceJet Ltd.
8	-8.0	Air India
9	8.0	GoAir

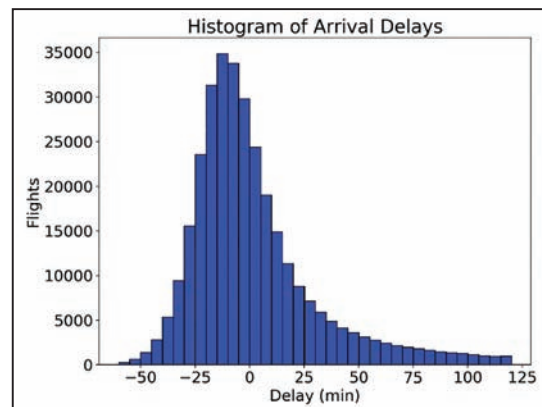
The flight arrival delays are in minutes and negative values mean the flights came early. There are over 300,000 flights with a minimum delay of -60 minutes and a maximum delay of 120 minutes. The other column is the name of the airline which we can use for comparisons.

Ans. For the plot calls, we specify the binwidth by the number of bins. For this plot, we will use bins that are 5 minutes in length, which means that the number of bins will be the range of the data (from -60 to 120 minutes) divided by the binwidth, *i.e.*, 5 minutes (bins = int(180/5)).

```
# Import the libraries
import matplotlib.pyplot as plt
flights_arr_delay=[-25.0,-18.0,-14.0,
                  -8.0,8.0,11.0,12.0,
                  19.0,20.0,33.0]

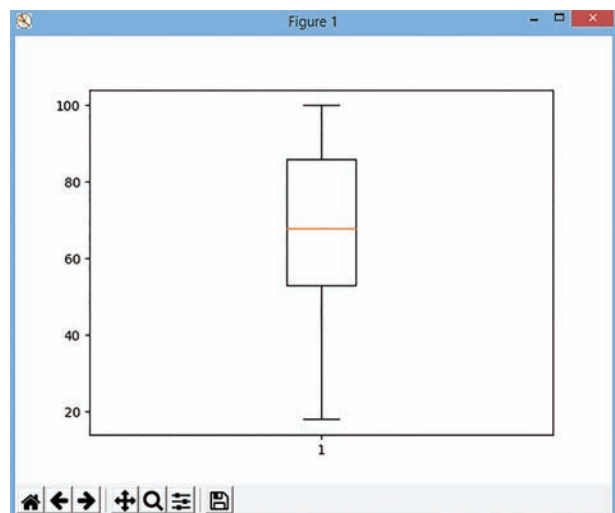
# matplotlib histogram
plt.hist(flights_arr_delay, color =
        'blue', edgecolor = 'black',
        bins = int(180/5))

# Add labels
plt.title('Histogram of Arrival Delays')
plt.xlabel('Delay (min)')
plt.ylabel('Flights')
plt.show()
```



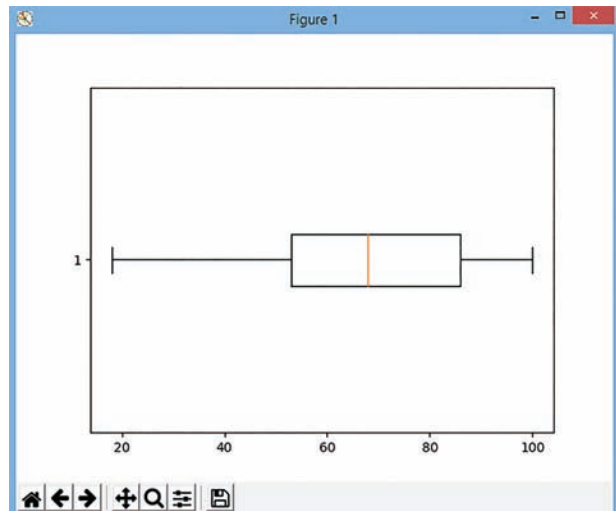
16. Create a box plot from the following set of data:
34, 18, 100, 27, 54, 93, 59, 61, 87, 68, 85, 78, 82, 91

```
Ans. import matplotlib.pyplot as plt
a = [34, 18, 100, 27, 54, 93, 59, 61,
     87, 68, 85, 78, 82, 91]
plt.boxplot(a)
plt.show()
```



17. Create the same box plot as previous question, but change the orientation to horizontal.

```
Ans. import matplotlib.pyplot as plt
a = [34, 18, 100, 27, 54, 52, 93, 59,
     61, 87, 68, 85, 78, 82, 91]
plt.boxplot(a, vert=False)
plt.show()
```



18. How are bar charts represented using matplotlib?

Ans. Bar charts display rectangular bars (either vertical or horizontal) with their length proportional to the values they represent. They are commonly used to visually compare two or more values. The bar() function is used to generate bar charts in Matplotlib. The function expects two lists of values: the X coordinates that are the positions of the bar's left margin and the height of the bars.

19. What is the purpose of plot function?

Ans. plot() is a versatile command and takes an arbitrary number of arguments. For example, plot(x, y) to plot x versus y.

20. Which function is required to plot a bar graph?

Ans. Bar in place of plot(), i.e., bar() function is used to plot a bar graph.

21. List the methods used with pyplot.

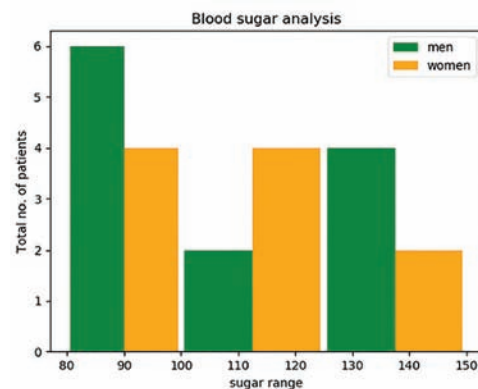
Ans. Various methods used with pyplot object are:

- plot()
- show()
- title()
- xlabel()
- ylabel()
- explode()
- bar()
- hist()
- scatter()
- box plot()

22. Given below are the sugar levels for men and women in a city. Compare the sugar levels amongst them through a histogram.

Men: [113,85,90,150,149,88,93,115,135,80,77,82,129]
 Women: [67,98,89,120,133,150,84,69,89,79,120,112,100]

```
Ans. import matplotlib.pyplot as plt
blood_sugar_men = [113,85,90,150,149,88,93,
                  115,135,80,77,82,129]
blood_sugar_women = [67,98,89,120,133,150,84,
                    69,89,79,120,112,100]
plt.xlabel('sugar range')
plt.ylabel('Total no. of patients')
plt.title('Blood sugar analysis')
plt.hist([blood_sugar_men, blood_sugar_women], bins=[80,100,125,150],
         rwidth=0.95, color=['green', 'orange'], label=['men', 'women'])
plt.legend()
plt.show()
```



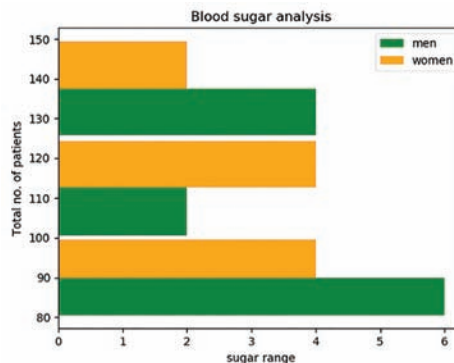
23. Modify the above code for displaying the histogram horizontally.

Ans. This can be done by using attribute "orientation" in hist() function and setting it to 'horizontal'.

```
import matplotlib.pyplot as plt
blood_sugar_men = [113,85,90,150,149,88,93,
                  115,135,80,77,82,129]
blood_sugar_women = [67,98,89,120,133,150,84,
                    69,89,79,120,112,100]

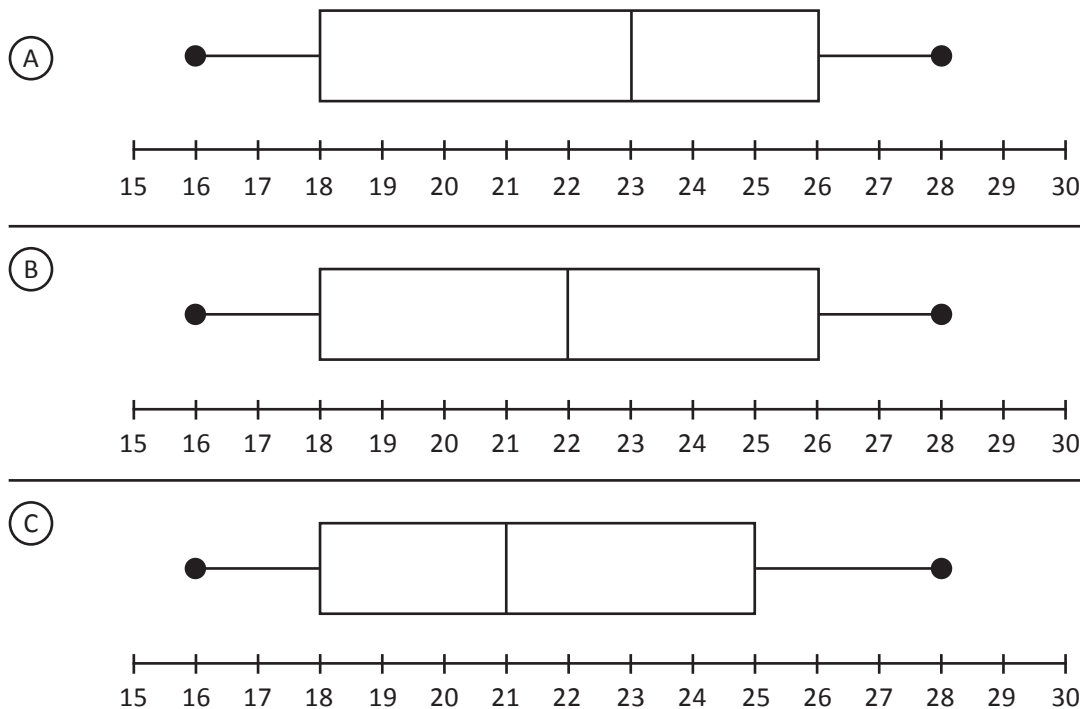
plt.xlabel('sugar range')
plt.ylabel('Total no. of patients')
plt.title('Blood sugar analysis')
plt.hist([blood_sugar_men, blood_sugar_women],
         bins=[80,100,125,150],
         rwidth=0.95,
         color=['green', 'orange'],
         label=['men', 'women'],
         orientation='horizontal')

plt.legend()
plt.show()
```



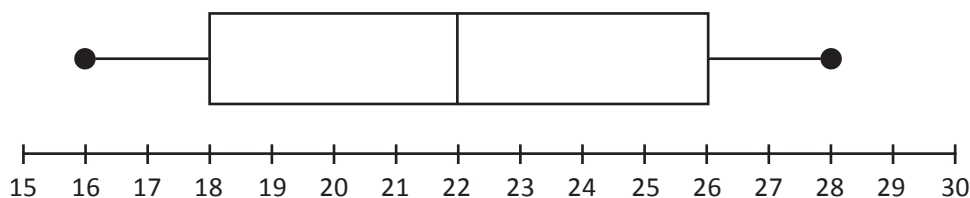
24. The data below represents the number of pages each student in class 12C read during reading time.
16,16,16,20,21,21,23,25,26,26,28,28

Which box plot given below correctly summarizes the data?



Ans. B is the correct answer.

The following box plot correctly summarizes the data:



16,16,16,20,21,21,23,25,26,26,28,28

Min = 16

$$\text{Median} = \frac{21+23}{2} = 22$$

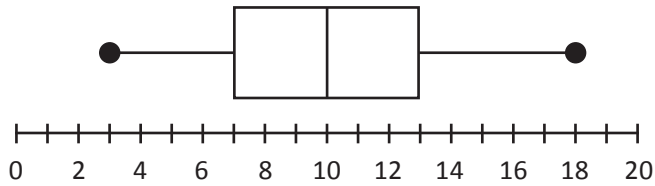
Max = 28

16,16,16,20,21,21,23,25,26,26,28,28

$$Q_1 = \frac{16+20}{2} = 18$$

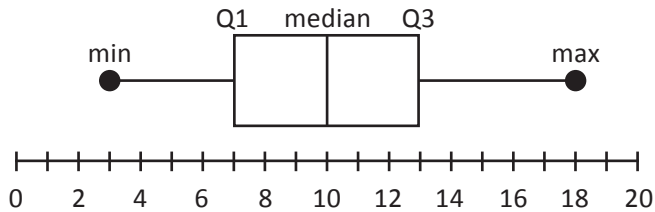
$$Q_3 = \frac{16+20}{2} = 26$$

25. Which dataset could be represented by the box plot shown below?



- (A) 3,4,8,9,9,10,12,13,13,16,18
- (B) 3,4,7,9,9,10,12,13,13,16,18
- (C) 3,4,8,9,9,12,12,13,13,16,18
- (D) 2,4,7,9,9,10,12,13,13,16,18

Ans.



The minimum of the box plot is 3.

We can eliminate one of the datasets because it does not have a minimum of 3.

2,4,7,9,9,10,12,13,13,16,18

The median of the box plot is 10.

We can eliminate one of the datasets because it does not have a median of 10.

3,4,8,9,9,12,12,13,13,16,18

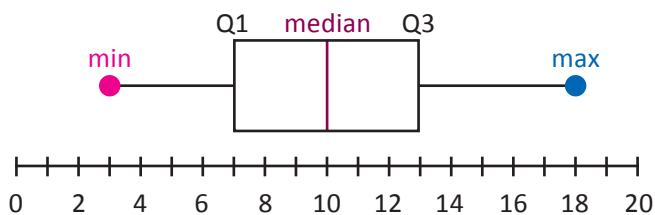
The first quartile of the box plot is 7.

We can eliminate one of the datasets because it does not have a first quartile of 7.

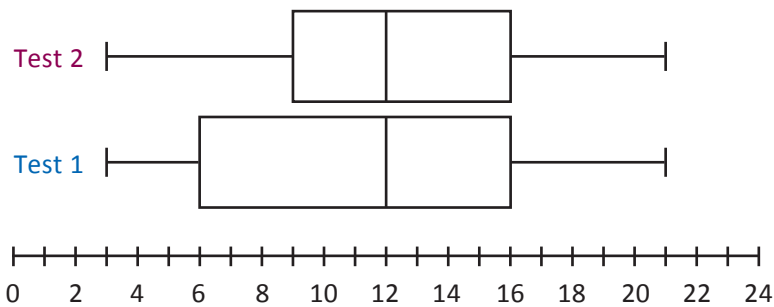
3,4,8,9,9,12,12,13,13,16,18

The following dataset could be represented by the box plot.

3,4,7,9,9,10,12,13,13,16,18



26. Suppose that the box-and-whisker plots below represent quiz scores out of 25 points for Quiz 1 and Quiz 2 for the same class. What do these box-and-whisker plots show about how the class did on test #2 compared to test #1?



Ans. These box-and-whisker plots show that the lowest score, the highest score and Q_3 are all the same for both the exams. So performance on the two exams was quite similar. However, the movement Q_1 up from a score of 6 to a score of 9, indicates that there was an overall improvement. On the first test, approximately 75% of the students scored at or above a score of 6. On the second test, the same number of students (75%) scored at or above a score of 9.

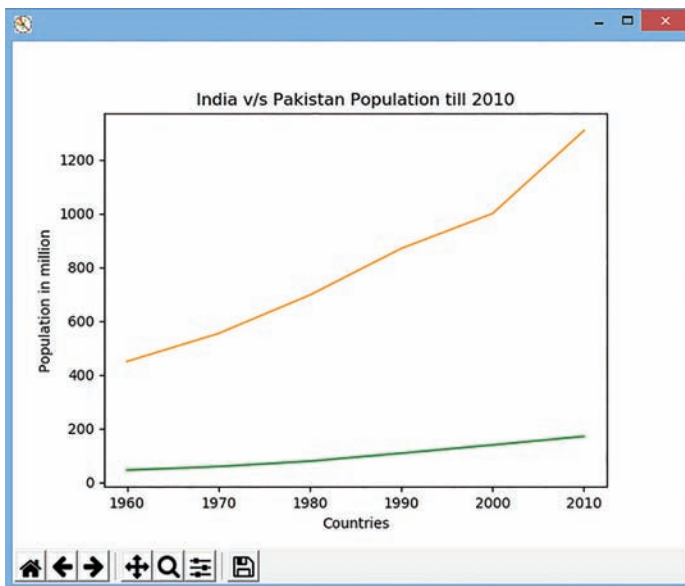
27. Which module needs to be imported for showing data in a chart form?

Ans. Matplotlib

28. Plot a line chart to depict a comparison of population between India and Pakistan.

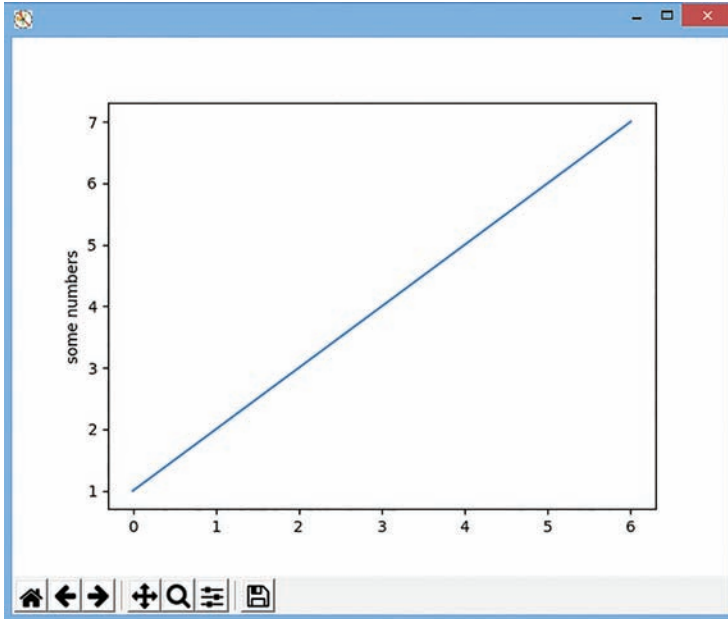
Ans. #Population comparison between India and Pakistan

```
from matplotlib import pyplot as plt
year = [1960, 1970, 1980, 1990, 2000, 2010]
popul_pakistan = [44.91, 58.09, 78.07, 107.7, 138.5, 170.6]
popul_india = [449.48, 553.57, 696.783, 870.133, 1000.4, 1309.1]
plt.plot(year, popul_pakistan, color='green')
plt.plot(year, popul_india, color='orange')
plt.xlabel('Countries')
plt.ylabel('Population in million')
plt.title('India v/s Pakistan Population till 2010')
plt.show()
```



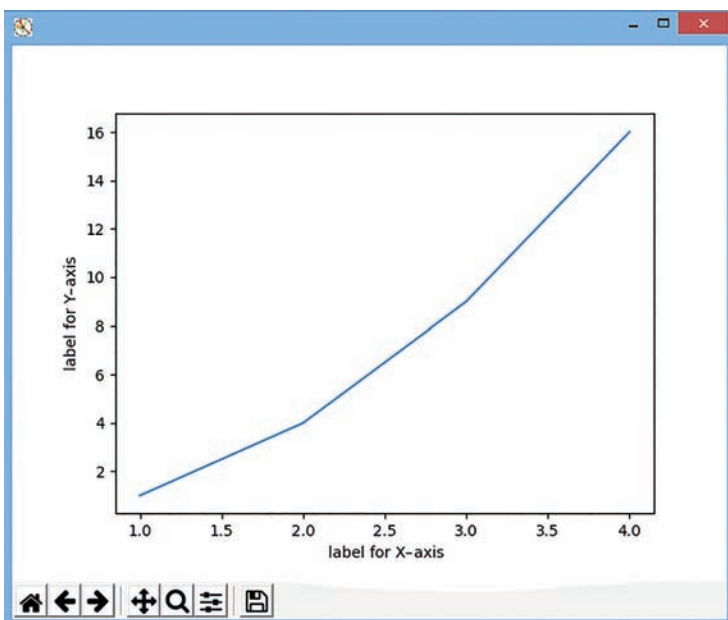
29. Plot list elements using line chart.

```
Ans. #To plot list elements using Line Chart
#in Python pyplot
import matplotlib.pyplot as plt
list2 = [1, 2, 3, 4, 5, 6, 7]
plt.plot(list2)
plt.ylabel('some numbers')
plt.show()
```



30. Write a program to plot list elements between X-axis and Y-axis.

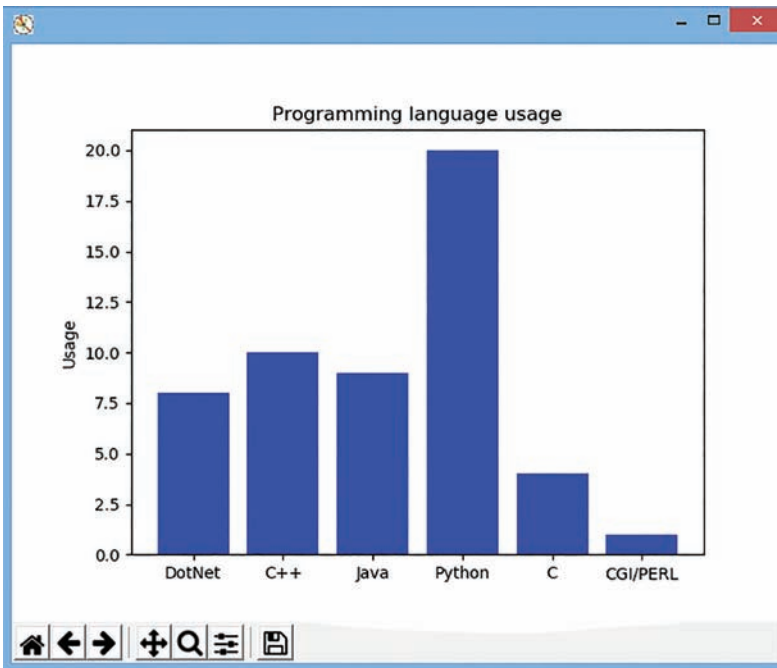
```
Ans. #Program to plot a simple Line chart holding list elements
#against x-axis and y-axis respectively
import matplotlib.pyplot as plt
plt.plot([1,2,3,4],[1,4,9,16])
plt.xlabel("label for X axis")
plt.ylabel("label for Y axis")
plt.show()
```



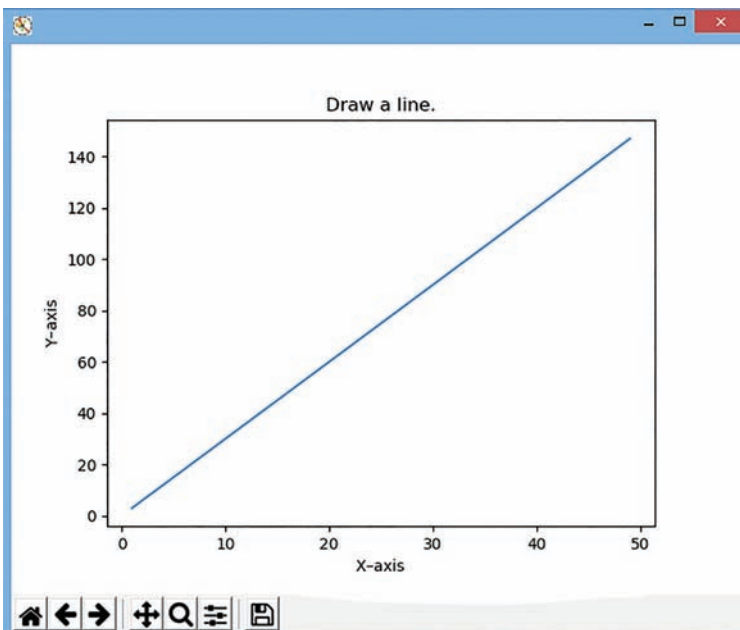
31. Plot a bar chart to depict the popularity of various programming languages.

Ans. #Program to plot a Bar chart on the basis of popularity of Programming Languages

```
import numpy as np
import matplotlib.pyplot as plt
objects = ('DotNet', 'C++', 'Java', 'Python', 'C', 'CGI/PERL')
y_pos = np.arange(len(objects))
performance = [8,10,9,20,4,1]
plt.bar(y_pos, performance, align='center', color='blue')
plt.xticks(y_pos, objects) #set location and label
plt.ylabel('Usage')
plt.title('Programming language usage')
plt.show()
```



32. Write a Python program to draw a line with a suitable label in the X-axis and Y-axis, and a title. The code snippet gives the output shown in the following screenshot:



```

Ans. import matplotlib.pyplot as plt
X = range(1, 50)
Y = [value * 3 for value in X]
print("Values of X:")
print(range(1,50))
print("Values of Y (thrice of X):")
print(Y)
# Plot lines and/or markers to the Axes.
plt.plot(X, Y)
# Set the x axis label of the current axis.
plt.xlabel('x - axis')
# Set the y axis label of the current axis.
plt.ylabel('y - axis')
# Set a title
plt.title('Draw a line. ')
# Display the figure.
plt.show()

```

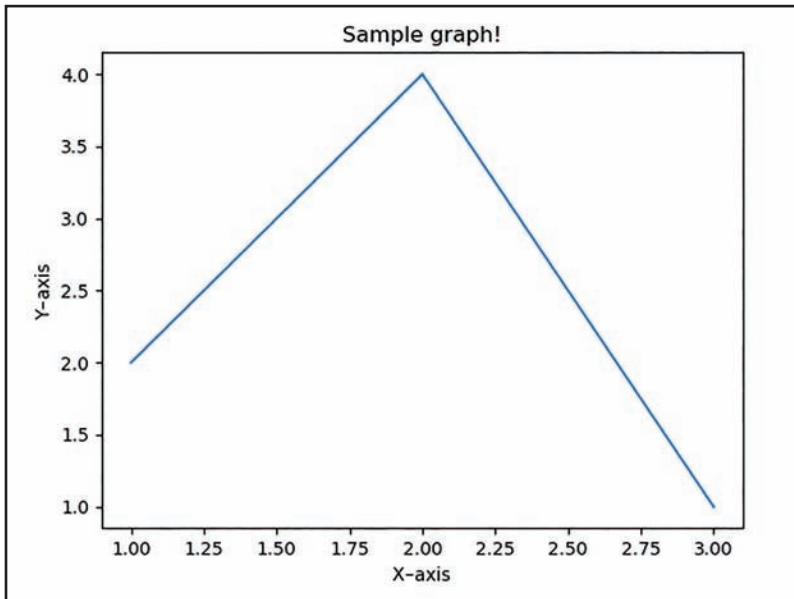
```

>>>
RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_s
olveques1_pyplot.py
Values of X:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
Values of Y (thrice of X):
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57,
60, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99, 102, 105, 108, 111
, 114, 117, 120, 123, 126, 129, 132, 135, 138, 141, 144, 147]
>>>

```

Ln: 9 Col: 4

33. Write a Python program to draw a line using given axis values with a suitable label in the X-axis and Y-axis, and a title. The code snippet gives the output as shown in the following screenshot:



```

Ans. import matplotlib.pyplot as plt
# x axis values
x = [1,2,3]
# y axis values

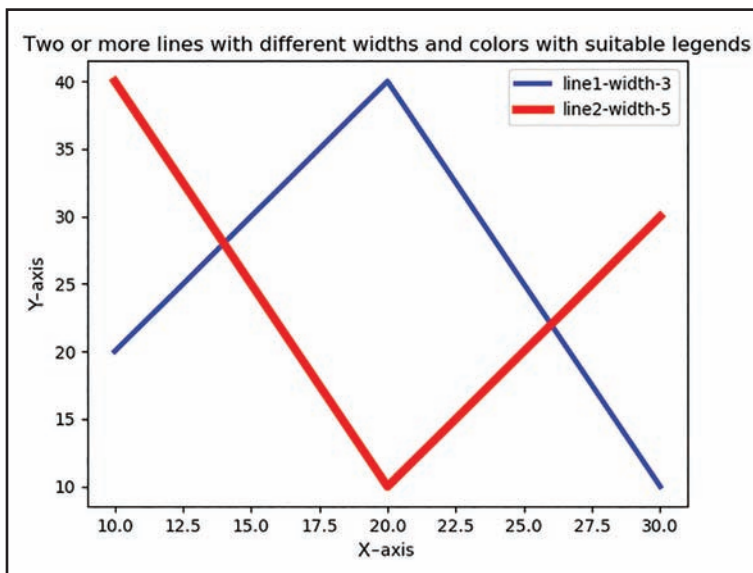
```

```

y = [2,4,1]
# Plot lines and/or markers to the Axes.
plt.plot(x, y)
# Set the x axis label of the current axis.
plt.xlabel('X - axis')
# Set the y axis label of the current axis.
plt.ylabel('Y - axis')
# Set a title
plt.title('Sample graph!')
# Display a figure.
plt.show()

```

34. Write a Python program to plot two or more lines with legends, different widths and colours.



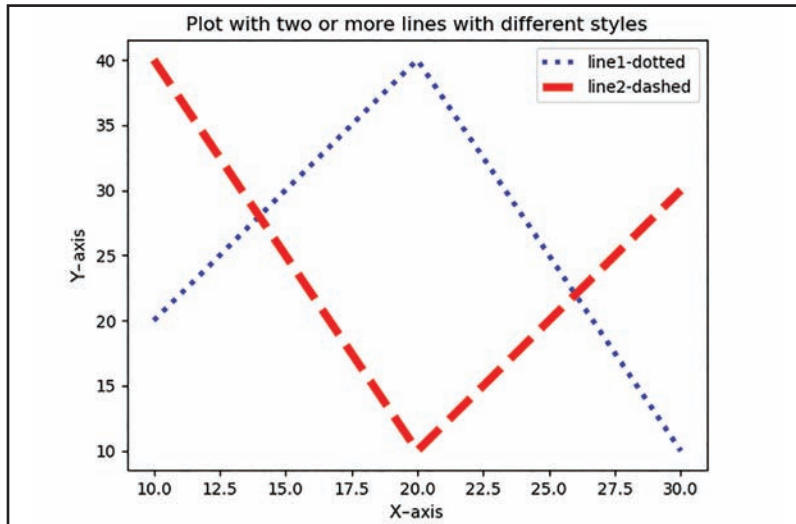
```

Ans. import matplotlib.pyplot as plt
# line 1 points
x1 = [10,20,30]
y1 = [20,40,10]
# line 2 points
x2 = [10,20,30]
y2 = [40,10,30]
# Set the x axis label of the current axis.
plt.xlabel('X - axis')
# Set the y axis label of the current axis.
plt.ylabel('Y - axis')
# Set a title
plt.title('Two or more lines with different widths and colors with suitable legends')
# Display the figure.
plt.plot(x1,y1, color='blue', linewidth = 3, label = 'line1-width-3')
plt.plot(x2,y2, color='red', linewidth = 5, label = 'line2-width-5')
# show a legend on the plot
plt.legend()
plt.show()

```

35. Write a Python program to plot two or more lines with different styles (dotted lines).

Ans.



```
Ans. import matplotlib.pyplot as plt
# line 1 points
x1 = [10,20,30]
y1 = [20,40,10]
# line 2 points
x2 = [10,20,30]
y2 = [40,10,30]
# Set the x axis label of the current axis.
plt.xlabel('X - axis')
# Set the y axis label of the current axis.
plt.ylabel('Y - axis')
# Plot lines and/or markers to the Axes.
plt.plot(x1,y1, color='blue', linewidth = 3, label = 'line1-dotted',
linestyle='dotted')
plt.plot(x2,y2, color='red', linewidth = 5, label = 'line2-dashed',
linestyle='dashed')
# Set a title
plt.title("Plot with two or more lines with different styles")
# show a legend on the plot
plt.legend()
# function to show the plot
plt.show()
```

36. Write a Python program to create a bar plot of scores by group and gender. Use multiple X values on the same chart for men and women.

Sample Data:

Means (men) = (22, 30, 35, 35, 26)

Means (women) = (25, 32, 30, 35, 29)

```
Ans. import numpy as np
import matplotlib.pyplot as plt
# data to plot
n_groups = 5
men_means = (22, 30, 35, 35, 26)
women_means = (25, 32, 30, 35, 29)
y_pos = np.arange(len(men_means))
plt.bar(2*y_pos,women_means,color='g',label="Women's score")
plt.bar(2*y_pos+1,men_means,color='b',label="Men's score")
plt.xlabel('Person')
plt.ylabel('Scores')
```



```
plt.title('Scores by person')
plt.legend()
plt.show()
```

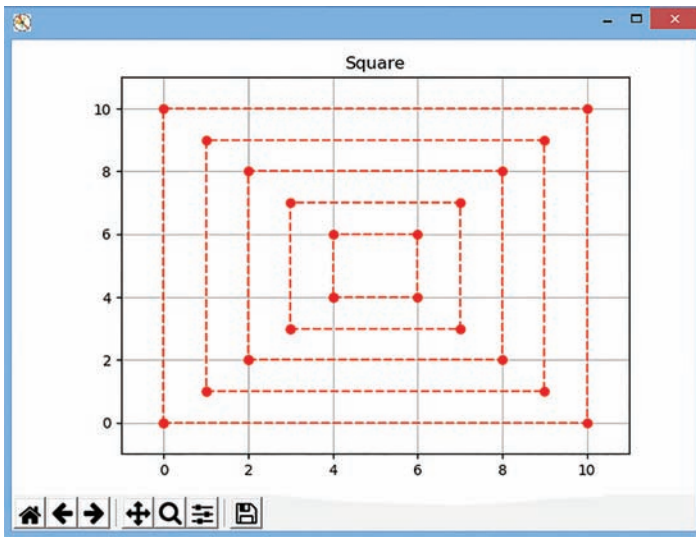


37. Plot a square using grid() in pyplot.

Ans. #To plot a square through function using pyplot

```
import matplotlib.pyplot as plt
def square(x, y):
    '''
    Objective: To plot a square
    Input Parameters: x, y - lists of x coordinates and y
    coordinates respectively
    Return Value: None
    '''
    if (x[1] - x[0]) <= 0:
        return
    plt.plot(x, y, 'ro--')
    return square([x[0]+1, x[1]-1, x[2]-1, x[3]+1, x[4]+1],[y[0]+1, y[1]+1,
        y[2]-1, y[3]-1, y[4]+1])
def main():
    '''
    Objective: To plot a square based on user input
    Input Parameter: None
    Return Value: None
    '''
    size = int(input('Enter size of the square: '))
    x = [0, size, size, 0, 0]
    y = [0, 0, size, size, 0]
    square(x, y)
    plt.title('Square')
    plt.axis([min(x)-1, max(x)+1, min(y)-1, max(y)+1])
    plt.grid()
    plt.show()
if __name__ == '__main__':
    main()
```

```
*Python 3.7.0 Shell*
File Edit Shell Debug Options Window Help
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
RESTART: C:/Users/preeti/AppData/Local/Programs/Python/Python37-32/prog_pyplot3.py
Enter size of the square: 10
```



38. Depict the relationship between unemployment Rate and Stock Index Price through a scatter plot.

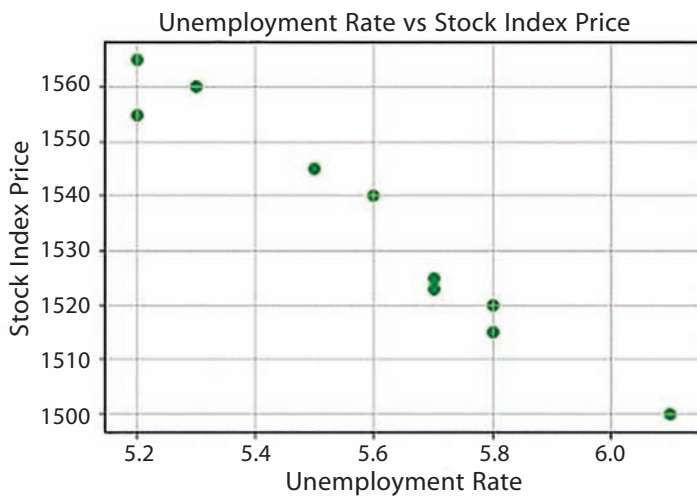
Unemployment_Rate	Stock_Index_Price
6.1	1500
5.8	1520
5.7	1525
5.7	1523
5.8	1515
5.6	1540
5.5	1545
5.3	1560
5.2	1555
5.2	1565

Ans. `import matplotlib.pyplot as plt`

```

Unemployment_Rate = [6.1,5.8,5.7,5.7,5.8,5.6,5.5,5.3,5.2,5.2]
Stock_Index_Price = [1500,1520,1525,1523,1515,1540,1545,1560,1555,1565]
plt.scatter(Unemployment_Rate, Stock_Index_Price, color='green')
plt.title('Unemployment Rate Vs Stock Index Price', fontsize=14)
plt.xlabel('Unemployment Rate', fontsize=14)
plt.ylabel('Stock Index Price', fontsize=14)
plt.grid(True)
plt.show()

```

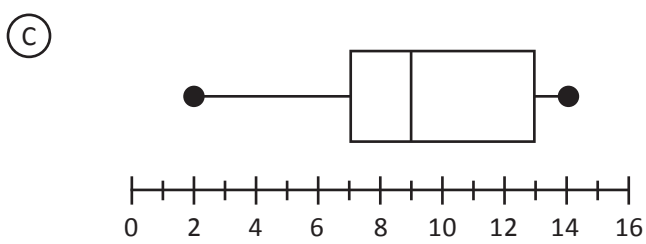
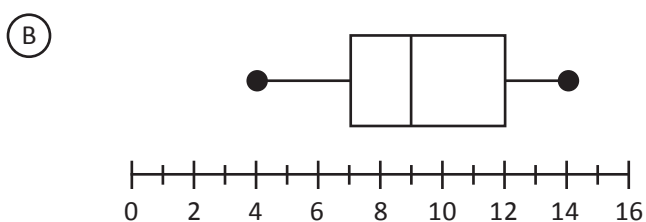
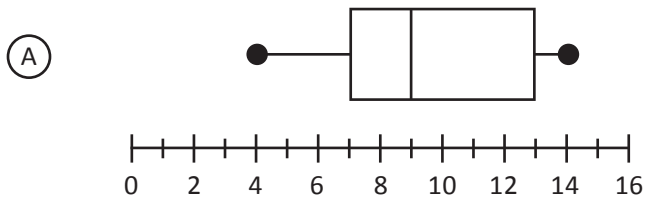


UNSOLVED QUESTIONS

- Plot a line chart for depicting the population for the last 5 years as per the specifications given below:
 - plt.title("My Title") will add a title "My Title" to your plot.
 - plt.xlabel("Year") will add a label "Year" to your X-axis.
 - plt.ylabel("Population") will add a label "Population" to your Y-axis.
 - plt.xticks([1, 2, 3, 4, 5]) set the numbers on the X-axis to be 1, 2, 3, 4, 5. Pass it and label as a second argument. *For example*, if we use this code plt.xticks([1, 2, 3, 4, 5], ["1M", "2M", "3M", "4M", "5M"]), it will set the labels 1M, 2M, 3M, 4M, 5M on the X-axis.
 - plt.yticks() — works the same as plt.xticks(), but for the Y-axis.
- What is matplotlib?
- What do you mean by pyplot?
- How do we update pip?
- Why do we update pip?
- How many types of graphs are plotted using pyplot?
- Explain the utility of explode().
- How is a pie chart different from a bar graph?
- Which function is used to show the graph?
- Write a Python program to draw a line with a suitable label in the X-axis and Y-axis, and a title.
- Write a Python program to plot two or more lines with legends, different widths and colours.
- Write a Python program to plot two or more lines and set the line markers.
- Write a Python program to create a pie chart with the title of the Stream and percentage of Students. Make multiple wedges of the pie.
Sample data:
Stream : Science, Commerce, Humanities, Vocational, FMM
Strengths: 29%, 30%, 21%, 13%, 7%
- Write a Python program to display a bar chart of the number of students in a class. Use different colours for each bar.
Sample data:
Class : I,II,III,IV,V,VI,VII,VIII,IX, X
Strengths: 40,43,45,47,49,38,50,37,43,39
- Write a Python program to display a horizontal bar chart of the number of students in a class.
Sample data:
Class : I,II,III,IV,V,VI,VII,VIII,IX, X
Strengths: 40,43,45,47,49,38,50,37,43,39
- Plot a pie chart of a class test of 40 students based on random sets of marks obtained by the students (MM=100).
- Plot a line graph for: $y^2=4*x$
- Write a Python program to plot the function $y = x^2$ using the pyplot or matplotlib libraries.
- What are the various types of graphs?
- Name the various methods used with pyplot object.
- Write the specific purpose of the following functions used in plotting:
 - shape()
 - legend()
- Write a Python program to plot the function $y=x^2$.
- Plot a histogram of a class test of 40 students based on random sets of marks obtained by the students (MM=100).
- A list namely temp contains average temperature for seven days of last week. You want to see how the temperature changes in last seven days. Which chart type will you plot for the same and why?

25. Write the code to practically produce a chart of previous question (i) using line chart, (ii) using scatter chart.
26. What is a histogram? How do you create histograms in Python?
27. What are the various types of histograms that can be created through hist() function?
28. When should you create histograms and when should you create bar charts to present data visually?
29. What is a frequency polygon? How do you create it?
30. What is Box plot? How do you create it in Pyplot?
31. Given the following set of data:
 Weight measurements for 14 values of muffins (in grams)
 78, 72, 69, 81, 63, 67, 65
 79, 74, 71, 83, 71, 79, 80
- (a) Create a simple histogram from the above data.
 (b) Create a horizontal histogram from the above data.
 (c) Create a step type of histogram from the above data.
 (d) Create a cumulative histogram from the above data.
32. Create/draw frequency polygon from the data used in the above question.
33. From the following ordered set of data:
 63, 65, 67, 69, 71, 71, 72, 74, 75, 78, 79, 79, 80, 81, 83
- (a) Create a horizontal box plot
 (b) Create a vertical box plot
34. Kritika was asked to write the names of a few libraries in Python used for data analysis and one method of each. Help her write at least 3 libraries and their methods.
35. The data below represents the number of workouts each member of ABC Gym attended last month.
 4, 5, 7, 7, 7, 8, 10, 11, 11, 13, 13, 14

Which box plot correctly summarizes the data?



36. The following amounts (in ₹) were the hourly collection from BIG MARKET at a local store in one day in December:
 19, 26, 25, 37, 32, 28, 22, 23, 29, 34, 39 and 31.
 Construct the box-and-whisker plot for the amount collected.