Python Programming

(CS3107)

LABORATORY MANUAL & RECORD

(III YEAR- I SEM)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SANKETIKA VIDYA PARISHAD ENGINEERING COLLEGE

(APPROVED BY AICTE, AFFILIARED TO ANDHRA UNIVERSITY, ACCREDITED BY NAAC-A GRADE, ISO 9001:2015 CERFIFIED)
PM PALEM, VISAKHAPATNAM-41,
www.svpce.edu.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE VISION AND MISSION

VISION

To be a premier institute of knowledge of share quality research and development technologies towards national buildings

MISSION

1. Develop the state of the art environment of high quality of learning
2. Collaborate with industries and academic towards training research innovation and entrepreneurship
3. Create a platform of active participation co-circular and extra- circular activities

DEPARTMENT VISION AND MISSION

VISION

To impart quality education for producing highly talented globally recognizable techno carts and entrepreneurs with innovative ideas in computer science and engineering to meet industrial needs and societal expectations

MISSION

- To impart high standard value-based technical education in all aspects of Computer Science and Engineering through the state of the art infrastructure and innovative approach.
- To produce ethical, motivated, and skilled engineers through theoretical knowledge and practical applications.
- To impart the ability for tackling simple to complex problems individually as well as in a team.
- To develop globally competent engineers with strong foundations, capable of “out of the box” thinking so as to adapt to the rapidly changing scenarios requiring socially conscious green computing solutions.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

Graduates of B. Tech in Computer Science and Engineering Program shall be able to

PEO1: Strong foundation of knowledge and skills in the field of Computer Science and Engineering.

PEO2: Provide solutions to challenging problems in their profession by applying computer engineering theory and practices.

PEO3: Produce leadership and are effective in multidisciplinary environment.

PROGRAMME SPECIFIC OUTCOMES (PSOs)

PSO1: Ability to design and develop computer programs and understand the structure and development methodologies of software systems.

PSO2: Ability to apply their skills in the field of networking, web design, cloud computing and data analytics.

PSO3: Ability to understand the basic and advanced computing technologies towards getting employed or to become an entrepreneur

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM OUTCOMES

1. **Engineering Knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem Analysis:** Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/Development of Solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct Investigations of Complex Problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern Tool Usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The Engineer and Society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and Sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and Team Work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, give and receive clear instructions.
11. **Project Management and Finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-Long Learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS3107 PYTHON PROGRAMMING LAB

Course Objectives:

- familiarize students with key data structures in Python including lists and dictionaries and apply them in context of searching, sorting, text and file handling
- introduce students to calculation of statistical measures using Python such as measures of central tendency, correlation
- familiarize students with important Python data related libraries such as Numpy and Pandas and use them to manipulate arrays and data frames
- introduce students to data visualization in Python through creation of line plots, histograms, scatter plots, box plots and others
- Implementation of basic machine learning tasks in Python including pre-processing data, dimensionality reduction of data using PCA, clustering, classification and cross-validation.

Course Outcomes:

- After completion of the course the student should be able to:
- implement searching, sorting and handle text and files using Python data structures such as lists and dictionaries
- calculate statistical measures using Python such as measures of central tendency, correlation
- use Python data related libraries such as Numpy and Pandas and create data visualizations
- Implement basic machine learning tasks pre-processing data, compressing data, clustering, classification and cross-validation.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS3107 PYTHON PROGRAMMING LAB

Syllabus:

1. Python Programs on lists & Dictionaries
2. Python Programs on Searching and sorting
3. Python Programs on Text Handling
4. Python Programs on File Handling
5. Python Programs for calculating Mean, Mode, Median, Variance, Standard Deviation
6. Python Programs for Karl Pearson Coefficient of Correlation, Rank Correlation
7. Python Programs on NumPy Arrays, Linear algebra with NumPy
8. Python Programs for creation and manipulation of Data Frames using Pandas Library
9. Write a Python program for the following.
    - Simple Line Plots,
    - Adjusting the Plot: Line Colours and Styles, Axes Limits, Labelling Plots,
    - Simple Scatter Plots,
    - Histograms,
    - Customizing Plot Legends,
    - Choosing Elements for the Legend,
    - Boxplot
    - Multiple Legends,
    - Customizing Colorbars,
    - Multiple Subplots,
    - Text and Annotation,
    - Customizing Ticks

10. Python Programs for Data preprocessing: Handling missing values, handling categorical data, bringing features to same scale, selecting meaningful features
11. Python Program for Compressing data via dimensionality reduction: PCA
12. Python Programs for Data Clustering
13. Python Programs for Classification
14. Python Programs for Model Evaluation: K-fold cross validation

Reference Books:

1. Core Python Programming, Second Edition, Wesley J. Chun, Prentice Hall
2. Chris Albon,—Machine Learning with Python Cookbook-practical solutions from pre-processing to Deep learning‖, O'REILLY Publisher,2018
3. Mark Summerfield, Programming in Python 3--A Complete Introduction to the Python Language, Second Edition, Additson Wesley
4. Phuong Vo.T.H , Martin Czygan, Getting Started with Python Data Analysis, Packt Publishing Ltd
5. Armando Fandango, Python Data Analysis, Packt Publishing Ltd
6. Magnus VilhelmPersson and Luiz Felipe Martins, Mastering Python Data Analysis, Packt Publishing Ltd
7. Sebastian Raschka&VahidMirjalili, —Python Machine Learning‖, Packt Publisher, 201

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the lab with the synopsis / program / experiment details.
3. Student should enter into the laboratory with:
   a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
   b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
   c. Proper Dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during working hours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

TABLE OF CONTENTS

1. Python programs on lists and dictionaries

➢ Program to find largest number in a list

```
a=[]
n=int(input("Enter number of elements :"))
for i in range (1,n+1):
    b=int(input("Enter element :"))
a.append(b)
print("largest element is ",max(a))
```

Output

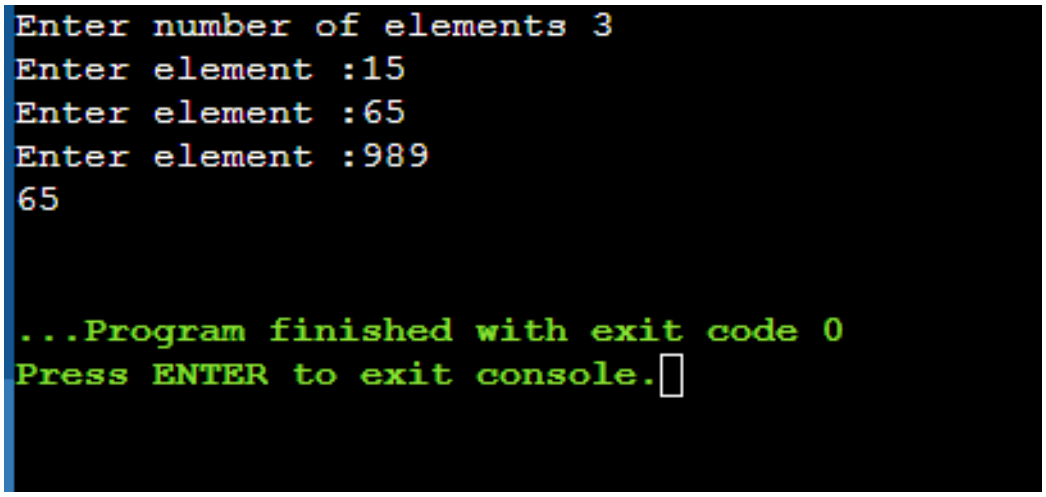
```
Enter number of elements :3
Enter element :23
Enter element :26
Enter element :59
largest element is  59


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Program to find the second largest number in a list

```
a=[]
n=int(input("Enter number of elements "))
for i in range(0,n):
    b=int(input("Enter element :"))
    a.append(b)
    a.sort()
    a.reverse()
i=0
while a[i]==a[i+1]:
```

```
    i+=1;
print(a[i+1])
```

Output



➢ Program to split even and odd elements into two lists

```
a=[]
n=int(input("Enter number of elements :"))
for i in range(n):
   b=int(input("Enter Element :"))
   a.append(b)
even=[]
odd=[]
for i in a:
    if(i%2==0):
        even.append(i)
    else:
        odd.append(i)
print("The even list is ",even)
print("The odd list is ",odd)
```

Output

```
Enter Element :23
Enter Element :65
Enter Element :22
Enter Element :62
Enter Element :35
Enter Element :98
The even list is  [22, 62, 98]
The odd list is  [23, 65, 35]


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Program to find if a key exists in a dictionary or not

```
d={'A':1, 'B':2, 'C':3, "D":33, "E":55}
key=input("Enter a key to check :")
if key in d.keys():
    print("key is present and the value of the key is :")
    print(d[key])
else:
    print("key is absent")
```

Output

```
Enter a key to check :E
key is present and the value of the key is :
55


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ program to add key value pair into dictionary

```
d={'A':1, 'B':2, 'C':3}
str=input("Enter the key and value you want to enter separated by a whitespace :")
d.update({str.split()[0] : int(str.split()[1])})
print("updated dictionary is :",d)
```

Output

```
Enter the key and value you want to enter separated by a whitespace :F 6
updated dictionary is : {'A': 1, 'B': 2, 'C': 3, 'F': 6}


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Program to find the sum of the values in a dictionary

```
d={'A':100, 'B':200, 'C':300,'D':600}
print("The total sum of the values in the dictionary is ",sum(d.values()))
```

Output

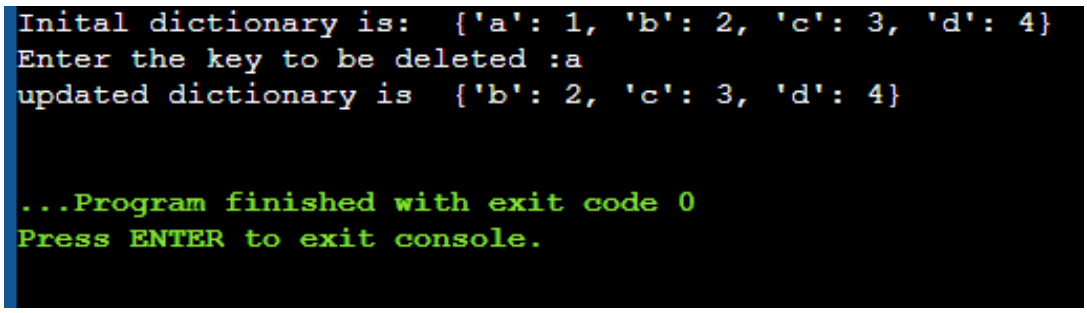
```
The total sum of the values in the dictionary is  1200


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Program to remove a key from a dictionary

```
d={'a':1, 'b':2, 'c':3, 'd':4}
print("Inital dictionary is: ",d)
key=input("Enter the key to be deleted :")
if key in d:
```

```
    del d[key]
    print("updated dictionary is ",d)
else:
    print("key not found")
```

Output

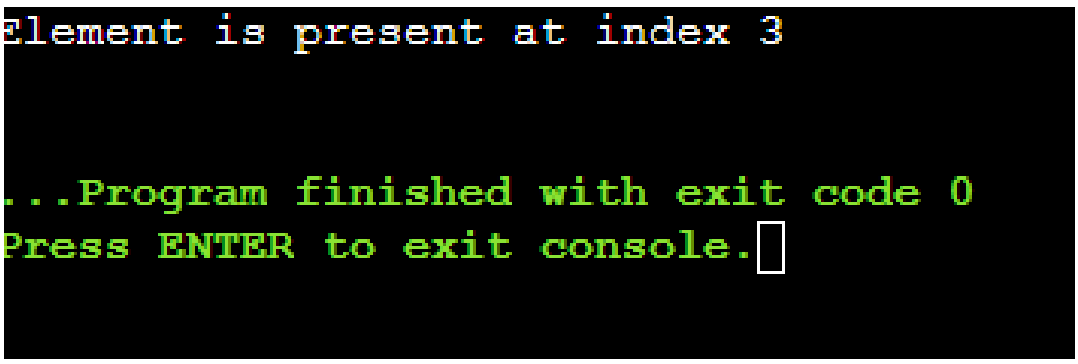


2. Python program on searching and sorting

➢ Linear search function

```
def search(arr, N, x):
    for i in range(0, N):
        if arr[i] == x:  # Fixed the comparison
            return i
    return -1  # Return -1 if element is not found
# Driver code
if __name___== '__main__':
    arr = [2, 3, 4, 10, 40]
    x = 10
    N = len(arr)
    result = search(arr, N, x)

    if result == -1:
        print("Element is not present")
    else:
        print("Element is present at index", result
```

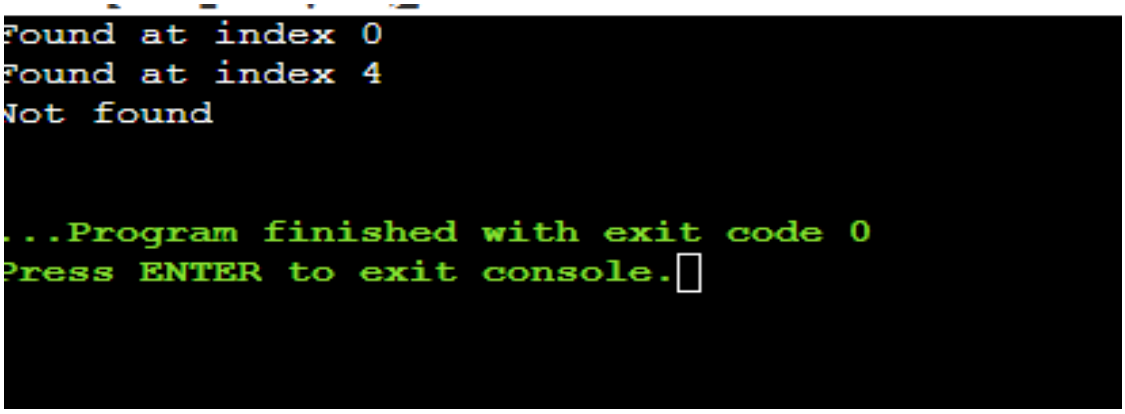
Output:



➢ Binary Search

```
def binary_search(V, To_Find):
    lo = 0
    hi = len(V) - 1
    while lo <= hi:  # We want to continue until the range is exhausted
        mid = (lo + hi) // 2
        if V[mid] == To_Find:  # If the mid value is the one we are looking for
            print("Found at index", mid)
            return  # Exit after finding the value
        elif V[mid] < To_Find:
            lo = mid + 1  # Move the low bound up
        else:
            hi = mid - 1  # Move the high bound down
    # If the loop finishes, it means the value was not found
    print("Not found")


if __name__ == '__main__':
    V = [1, 3, 4, 5, 6]
    To_Find = 1
    binary_search(V, To_Find)  # Expected: Found at index 0
    To_Find = 6
    binary_search(V, To_Find)  # Expected: Found at index 4
    To_Find = 10
    binary_search(V, To_Find)  # Expected: Not found
```

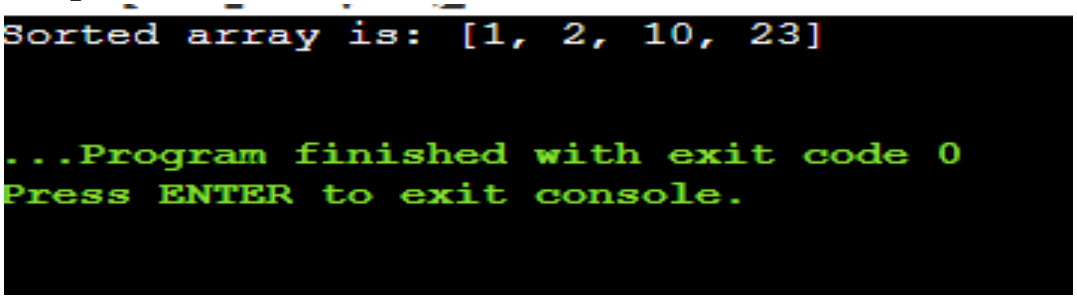
Output:



- **Bubble Sort**

```
def bubbleSort(arr):
    n = len(arr)
    for i in range(n):
        swapped = False  # Flag to detect if any swap was made during this pass
        for j in range(0, n - i - 1):  # Traverse the array from 0 to n-i-1
            if arr[j] > arr[j + 1]:  # Swap if the element is greater than the next element
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        # If no two elements were swapped by inner loop, then the array is sorted
        if not swapped:
            break

if __name___== '__main__':
    arr = [2, 1, 23, 10]
    bubbleSort(arr)
    print("Sorted array is:", arr)
```

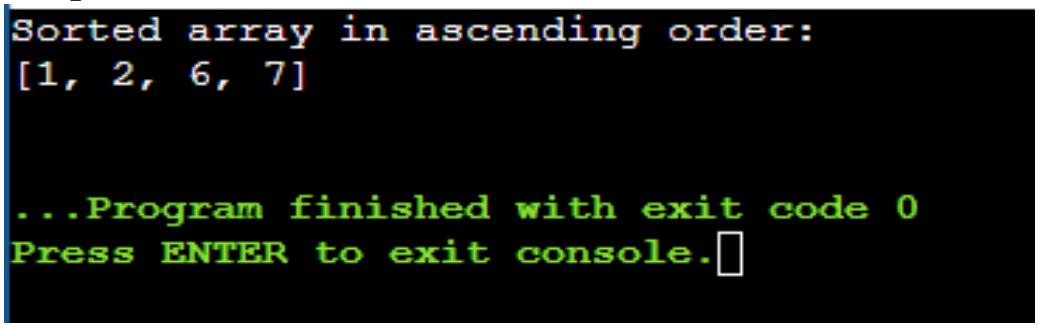
Output:



➢ Selection Sort

```
def selectionsort(array):
    size = len(array)  # Get the size of the array
    for s in range(size):
        min_idx = s
        for i in range(s + 1, size):
            if array[i] < array[min_idx]:  # Find the smallest element
                min_idx = i
        # Swap the found minimum element with the first element of the unsorted
part
        array[s], array[min_idx] = array[min_idx], array[s]

if __name___== '__main__':
    data = [7, 2, 1, 6]
    selectionsort(data)
    print('Sorted array in ascending order:')
    print(data)
```

Output:

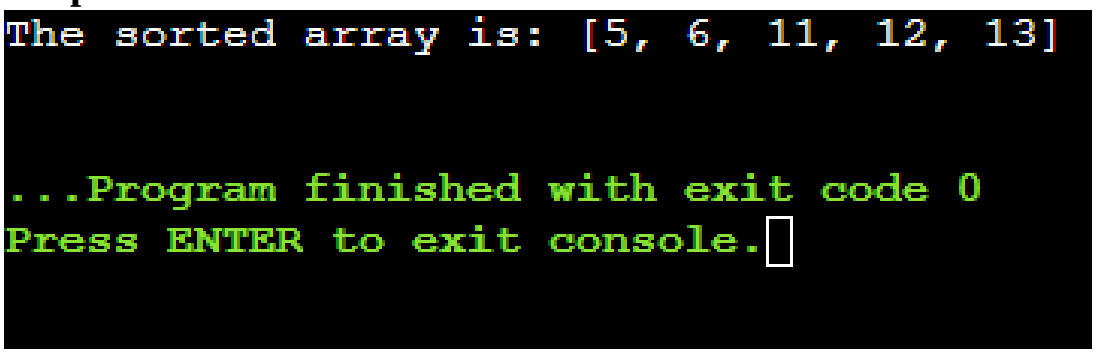


➢ Insertion Sort

```
def insertionSort(arr):

    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1

        arr[j + 1] = key
```

```
if __name___== '__main__':
    arr = [12, 11, 13, 5, 6]
    insertionSort(arr)
    print("The sorted array is:", arr)
```

Output:



- **Quick Sort:**

```
def partition(array, low, high):
    pivot = array[high]
    i = low - 1
    for j in range(low, high):
        if array[j] <= pivot:
            i = i + 1
            array[i], array[j] = array[j], array[i]

    array[i + 1], array[high] = array[high], array[i + 1]
    return i + 1

def quick_sort(array, low, high):
    if low < high:
        pi = partition(array, low, high)
        quick_sort(array, low, pi - 1)
        quick_sort(array, pi + 1, high)

if __name___== '__main__':
    array = [10, 7, 8, 9, 1, 5]
    quick_sort(array, 0, len(array) - 1)
    print(f"Sorted array: {array}")\
```

Output:

```
Sorted array: [1, 5, 7, 8, 9, 10]


...Program finished with exit code 0
Press ENTER to exit console.
```

- **Merge Sort**

```
def mergesort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        l = arr[:mid]
        r = arr[mid:]

        mergesort(l)
        mergesort(r)

        i = j = k = 0

        while i < len(l) and j < len(r):
            if l[i] <= r[j]:
                arr[k] = l[i]
                i += 1
            else:
                arr[k] = r[j]
                j += 1
            k += 1

        while i < len(l):
            arr[k] = l[i]
            i += 1
            k += 1

        while j < len(r):
            arr[k] = r[j]
            j += 1
            k += 1
```

```
def printlist(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()

if __name___== '__main__':
    arr = [12, 11, 10, 3, 5]
    print("Given array is:", end="\n")
    printlist(arr)
    mergesort(arr)
    print("Sorted array is:", end="\n")
    printlist(arr)
```

Output:

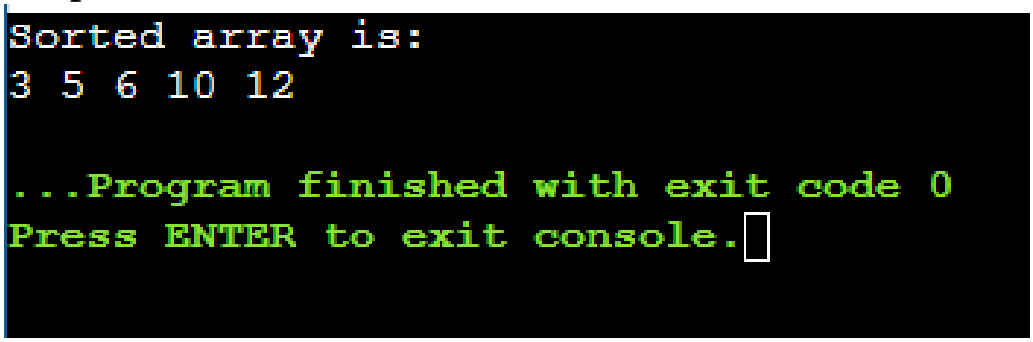
```
Given array is:
12 11 10 3 5
Sorted array is:
3 5 10 11 12


...Program finished with exit code 0
Press ENTER to exit console.
```

- **Heap Sort**

```
def heapify(arr, N, i):
    largest = i
    l = 2 * i + 1
    r = 2 * i + 2

    if l < N and arr[largest] < arr[l]:
        largest = l

    if r < N and arr[largest] < arr[r]:
        largest = r

    if largest != i:
        arr[i], arr[largest] = arr[largest], arr[i]
```

```
        heapify(arr, N, largest)

def heapsort(arr):
    N = len(arr)

    for i in range(N // 2 - 1, -1, -1):
        heapify(arr, N, i)

    for i in range(N - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        heapify(arr, i, 0)

if __name___== '__main__':
    arr = [10, 12, 3, 5, 6]
    heapsort(arr)
    N = len(arr)
    print('Sorted array is:')
    for i in range(N):
        print("%d" % arr[i], end=" ")
```

Output:



3. Python programs on text handling

➢ Text handling

```
a = "Hello world good morning"
print(len(a))
print(a.count("l"))
print(a.startswith("H"))
print(a.endswith("l"))
print(a.find("l"))
print(a.rfind("l"))
print(a.isalnum())
print(a.isdigit())
print(a.isalpha())
print(a.islower())
```

Output

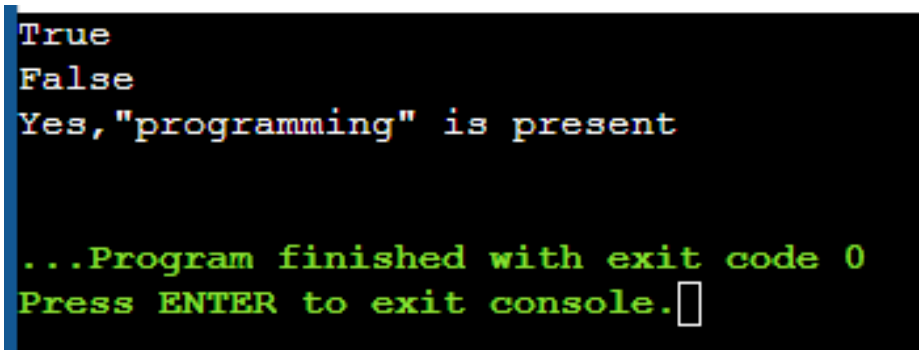
```
True
False
2
9
False
False
False
False

...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Checking in a string

```
text = "Python is a high-level, general-purpose programming language"
print("programming" in text)
```

```
print("programming" not in text)
if "programming" in text:
    print('Yes,"programming" is present')
```

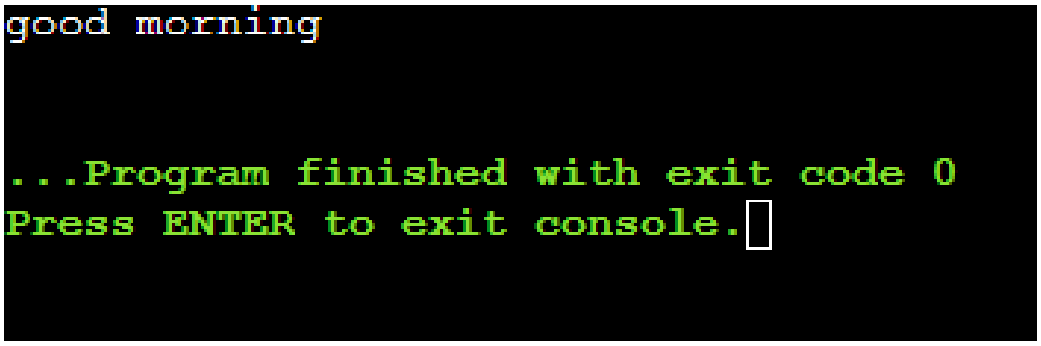
Output



➢ Concatenation in strings

```
a = "good"
b = "morning"
c = a + " " + b
print(c)
```

Output

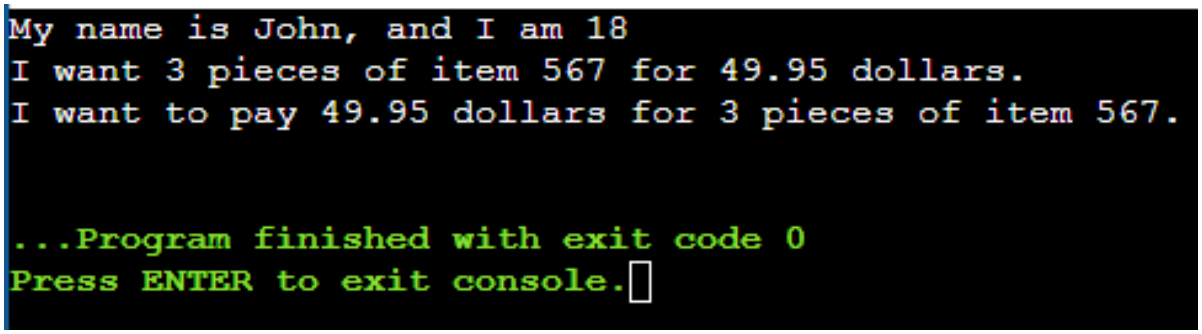


Format strings

```
age = 18
txt = "My name is John, and I am {}"
```

```
print(txt.format(age))
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

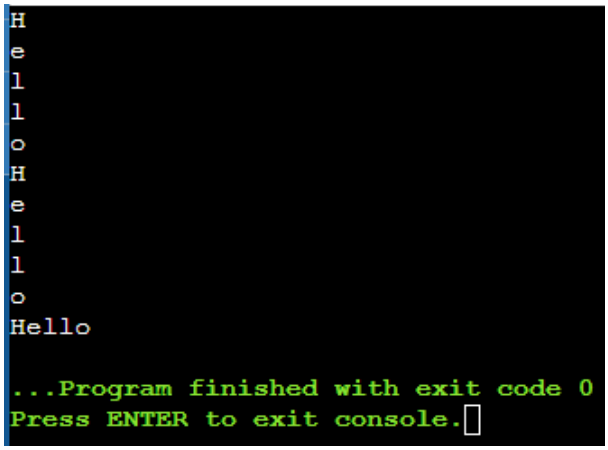
Output



➢ Looping in strings

```
a = "Hello"
for x in a:
    print(x)
for x in "Hello":
    print(x)
for x in a:
    print(x, end="")
```

Output



➢ Modify strings

```
a = "hello world"
print(a.upper())
b = "HelLO woRlD"
print(b.lower())
print(b.casefold())
c = " Hello world! "
print(c.strip())
print(c.lstrip())
print(c.rstrip())
d = "Hello World"
print(d.replace("He", "J"))
e = "hello, world"
print(e.split(","))
f = "hello world"
print(f.capitalize())
print(f.title())
g = "Hello My Name Is PETER"
print(g.swapcase())
```

Output

```
Hello world!
Hello world!
 Hello world!
Jllo World
['hello', ' world']
Hello world
Hello World
hELLO mY nAME iS peter


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Slicing a string

```
string = "Hello World"
print(string[2:5])
print(string[:5])
print(string[2:])
print(string[-5:-2])
```

Output

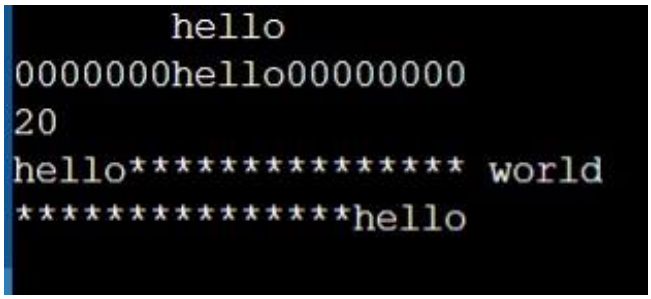
```
llo
Hello
llo World
Wor
```

➢ Align a string

```
txt = "hello"
a = txt.center(20)
print(a)
a = txt.center(20, "0")
```

```
print(a)
print(len(a))
a = txt.ljust(20, "*")
print(a, "world")
a = txt.rjust(20, "*")
print(a)
```

Output




```
# The partition() method searches for a specified string, and splits the string into a tuple containing three elements.
# The first element contains the part before the specified string.
# The second element contains the specified string.
# The third element contains the part after the string

txt = "I could eat bananas all day"
x = txt.partition("bananas")
print(x)
txt = "I could eat bananas all day"
x = txt.partition("apples")
print(x)
txt = "I could eat bananas all day, bananas are my favorite fruit"
x = txt.rpartition("bananas")
print(x)
```

Output

```
('I could eat ', 'bananas', ' all day')
('I could eat bananas all day', '', '')
('I could eat bananas all day, ', 'bananas', ' are my favorite fruit')


...Program finished with exit code 0
Press ENTER to exit console.
```

#translate and maketrans

The maketrans() method returns a mapping table that can be used with the translate() method to replace characters

```
txt = "Hello Sam!"

mytable = txt.maketrans("S", "P")

print(mytable)

print(txt.translate(mytable))

mydict = {83:  80}

print(txt.translate(mydict))

txt = "Good night Sam!"

x = "mSa"

y = "eJo"

z = "odnght"

mytable = txt.maketrans(x, y, z)

print(txt.translate(mytable))
```

Output

```
{83: 80}
Hello Pam!
Hello Pam!
G i Joe!


...Program finished with exit code 0
```

➢ Some other string functions

```
txt = "My name is Ståle"
x = txt.encode()
print(x)
txt = 'h\te\tl\tl\t0'
x = txt.expandtabs(1)
print(x)
txt = 'Demo'
print(txt.isidentifier())
print(txt.isprintable())
txt = "    "
print(txt.isspace())
myTuple = ("John", "Peter", "Vicky")
x = "#".join(myTuple)
print(x)
txt = "50"
print(txt.zfill(10))
txt = "Thank you for the music\nWelcome to the jungle"
print(txt.splitlines())
```

Output

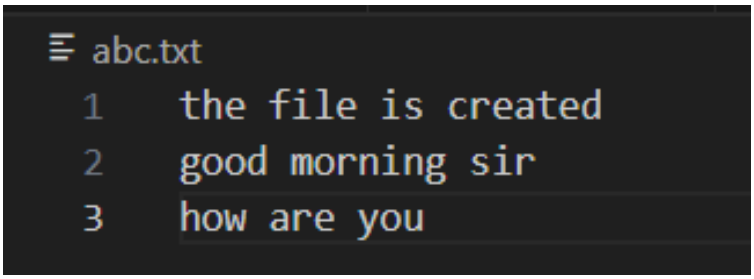
```
True
True
John#Peter#Vicky
0000000050
['Thank you for the music', 'Welcome to the jungle']
```

4. Python programs on file handling

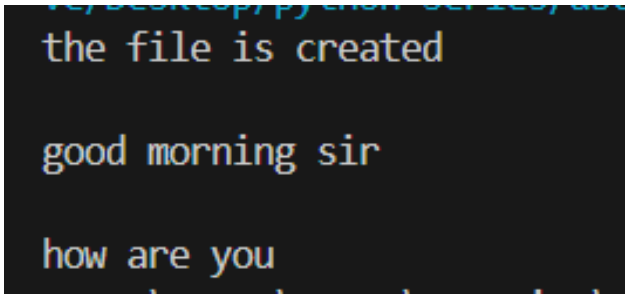
➢ Working of open() mode

```
file = open('abc.txt','r')
for each in file:
    print(each)
```

➢ The text in the file(abc.txt) is



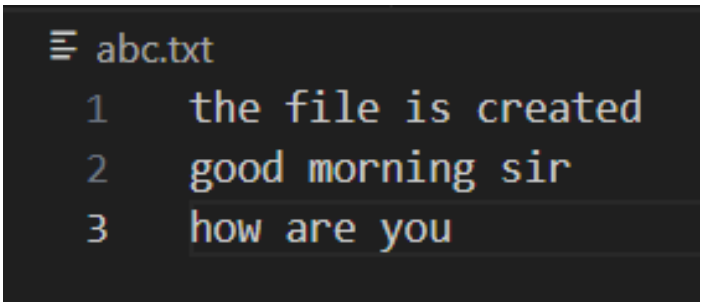
Output



➢ Working of read() mode

```
file=open('abc.txt','r')
print(file.read())
file.seek(0)
print(file.read(5))
```

➢ The text in file is



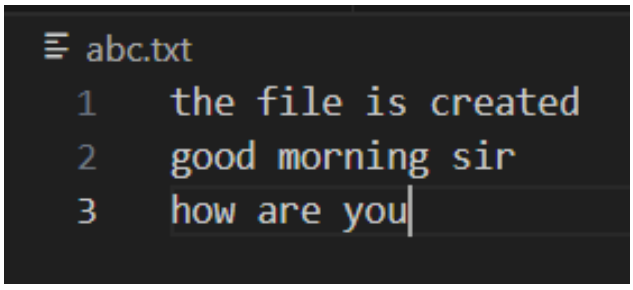
Output

```
the file is created
good morning sir
how are you
the f
```

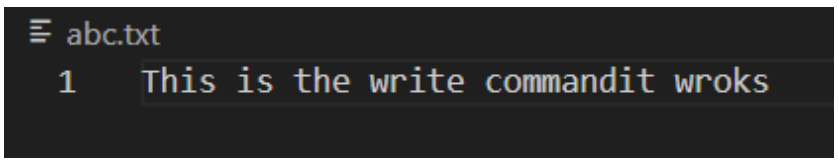
- **Creating a file using write() mode**

```
file = open('abc.txt','w')
file.write('This is the write command')
file.write('it wroks')
file.close()
```

- **The output before using write command**



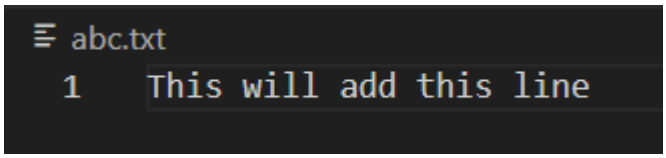
- **The output after using write command**



➢ Working of append() mode

```
file = open('abc.txt','a')
file.write('This will add this line')
file.close()
```

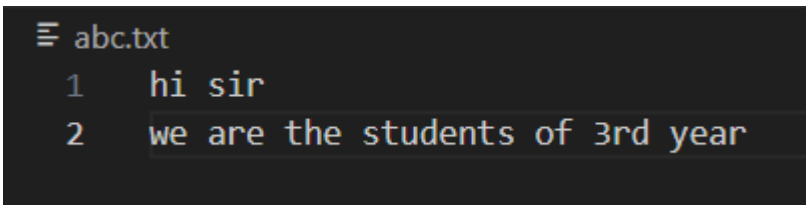
Output



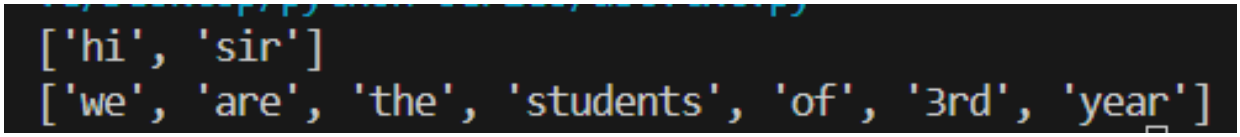
➢ Split() using file handling

```
with open('abc.txt','r') as file:
    data = file.readlines()
for line in data:
    word = line.split()
    print(word)
```

➢ The text in the file is



Output



➢ with block

```
with open('abc.txt') as file:
    data = file.read()
```

print(data)

- The text in the file is

abc.txt

```
1   hi sir
2   we are the students of 3rd year
```

Output

```
hi sir
we are the students of 3rd year
```

5. Python programs for calculating Mean, Mode, Median, Variance, Standard Deviation

➢ Calculating mean

```
lst = [19,22,34,26,32,30,24,24]
def mean(dataset):
    return sum(dataset)/len(dataset)
print(mean(lst))
```

Output

```
26.375


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Mean by using mean() from statistics

```
from statistics import mean
pythonis_machine_ages = [19,22,34,26,32,30,24,24]
print(mean(pythonis_machine_ages))
```

Output

```
26.375


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Calculating median

```
ls2 = [187,187,196,196,198,203,207,211,215]
ls2 = [187, 187, 196, 196, 198, 203, 207, 211, 215]
```

```
ls3 = [181, 187, 196, 198, 203, 207, 211, 215]
def median(dataset):
    data = sorted(dataset)
    index = len(data) // 2
    if len(dataset) % 2 != 0:
        return data[index]
    return (data[index - 1] + data[index]) / 2
print(median(ls2))
print(median(ls3))
```

Output

```
198
200.5


...Program finished with exit code 0
Press ENTER to exit console.
```

- **Median from statistic module**

```
ls2 = [181,187,196,196,198,203,207,211,215]
ls3 = [181,187,196,198,203,207,211,215]
print(median(ls2))
print(median(ls3))
```

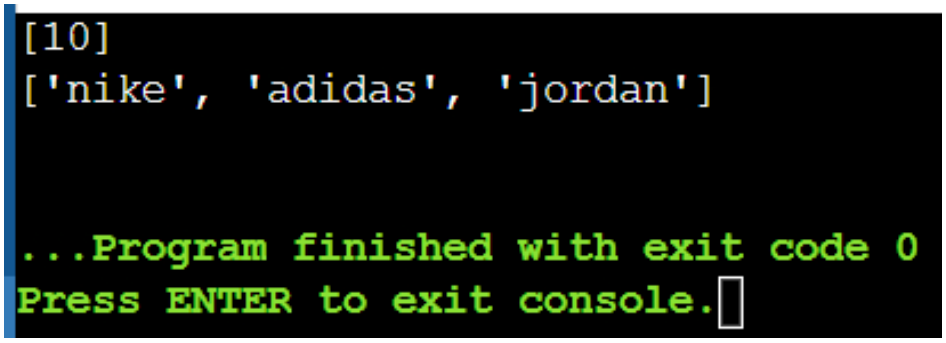
Output

```
198
200.5


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Calculating mode

```
ls2 = [3,15,23,42,30,10,10,12]
ls3 = ['nike','adidas','nike','jordan','jordan','rebook','under_amour','adidas']
def mode(dataset):
    frequency = {}
    for value in dataset:
        frequency[value] = frequency.get(value,0)+1
    most_frequent = max(frequency.values())
    modes = [key for key,value in frequency.items() if value == most_frequent]
    return modes
print(mode(ls2))
print(mode(ls3))
```

Output



➢ Using mode() & multimode from statistics module

```
from statistics import mode,multimode
ls2 = [3,15,23,42,30,10,10,12]
ls3 = ['nike','adidas','nike','jordan','jordan','reebok','under_amour','adidas']
print(mode(ls2))
print(mode(ls3))
print(multimode(ls2))
print(multimode(ls3))
```

Output

```
10
nike
[10]
['nike', 'adidas', 'jordan']


...Program finished with exit code 0
```

➢ Calculating variance

```
import statistics
print(statistics.variance([1, 3, 5, 7, 9, 11]))
print(statistics.variance([2, 2.5, 1.25, 3.1, 1.75, 2.8]))
print(statistics.variance([-11, 5.5, -3.4, 7.1]))
print(statistics.variance([1, 30, 50, 100]))
```

Output

```
14
0.47966666666666663
70.80333333333333
1736.9166666666667


...Program finished with exit code 0
```

➢ Calculating standard deviation

```
import statistics
print(statistics.stdev([1, 3, 5, 7, 9, 11]))
print(statistics.stdev([2, 2.5, 1.25, 3.1, 1.75, 2.8]))
print(statistics.stdev([-11, 5.5, -3.4, 7.1]))
print(statistics.stdev([1, 30, 50, 100]))
```

Output

```
3.7416573867739413
0.6925797186365383
8.414471660973927
41.67633221226008


...Program finished with exit code 0
```

6. Python programs for Karl Pearson Coefficient of Correlation and Rank Correlation

➢ Calculating Karl Pearson Coefficient of Correlation

```
import numpy as np
x_simple = np.array([-2, -1, 0, 1, 2])
y_simple = np.array([4, 1, 3, 2, 0])
my_rho = np.corrcoef(x_simple, y_simple)
print(my_rho)
```

Output:

```
[[ 1.  -0.7]
 [-0.7  1. ]]


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Calculating Rank Correlation

```
import pandas as pd
import scipy.stats
x = [15, 18, 21, 15, 21]
y = [25, 25, 27, 27, 27]
def spearmans_rank_correlation(x, y):
    xranks = pd.Series(x).rank()
    print("Rankings of X:")
    print(xranks)
```

```
yranks = pd.Series(y).rank()
print("Rankings of Y:")
print(yranks)
# Use spearmanr instead of pearsonr
correlation, _ = scipy.stats.spearmanr(xranks, yranks)
print("Spearman's Rank correlation:", correlation)
spearmans_rank_correlation(x, y)
```

Output:

```
Rankings of X:
0    1.5
1    3.0
2    4.5
3    1.5
4    4.5
dtype: float64
Rankings of Y:
0    1.5
1    1.5
2    4.0
3    4.0
4    4.0
dtype: float64
Spearman's Rank correlation: 0.45643546458763845
```

- **Calculating rank correlating using scipy.stats**

```
import scipy.stats
x = [15,18,21, 15, 21]
y = [25,25,27,27,27]
print(scipy.stats.spearmanr(x,  y)[0]
```

Output:

```
0.45643546458763845
```

7. Python programs on NumPy arrays and linear algebra with NumPy

➢ Python Numpy

Numpy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data.

NumPy is a general-purpose array-processing package.

It provides a high-performance multidimensional array object and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It is open-source software.

➢ Features of NumPy

NumPy has various features which make them popular over lists.

Some of these important features include:

A powerful N-dimensional array object

Sophisticated (broadcasting) functions

Tools for integrating C/C++ and Fortran code

Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy in Python can also be used as an efficient multi-dimensional container of generic data.

Arbitrary data types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

➢ Arrays in Numpy

Array in Numpy is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In Numpy, number of dimensions of the array is called rank of the array.A tuple of integers giving the size of the array along each dimension is known as shape of the array. An array class in Numpy is called as ndarray. Elements in Numpy arrays are accessed by using square brackets and can be initialized by using nested Python Lists.

➢ Program on numpy array

```
import numpy as np
arr = np.array( [[ 1, 2, 3],
                 [ 4, 2, 5]] )
print("Array is of type: ", type(arr))
print("No. of dimensions: ", arr.ndim)
print("Shape of array: ", arr.shape)
print("Size of array: ", arr.size)
print("Array stores elements of type: ", arr.dtype)
```

Output

```
Array is of type:  <class 'numpy.ndarray'>
No. of dimensions:  2
Shape of array:  (2, 3)
Size of array:  6
Array stores elements of type:  int64


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Creating a Numpy Array

Arrays in Numpy can be created by multiple ways, with various number of Ranks, defining the size of the Array. Arrays can also be created with the use of various data types such as lists, tuples, etc. The type of the resultant array is deduced from the type of the elements in the sequences.

➢ Program on creating a numpy array

```
import numpy as np
arr = np.array([1, 2, 3])
print("Array with Rank 1: \n",arr)
arr = np.array([[1, 2, 3],  [4, 5, 6]])
```

```
print("Array with Rank 2: \n", arr)
arr = np.array((1, 3, 2))
print("\nArray created using "
      "passed tuple:\n", arr)
```

Output

```
Array with Rank 1:
 [1 2 3]
Array with Rank 2:
 [[1 2 3]
 [4 5 6]]

Array created using passed tuple:
 [1 3 2]


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Accessing the array Index

In a numpy array, indexing or accessing the array index can be done in multiple ways. To print a range of an array, slicing is done. Slicing of an array is defining a range in a new array which is used to print a range of elements from the original array. Since, sliced array holds a range of elements of the original array, modifying content with the help of sliced array modifies the original array content.

➢ Program on accessing the array Index

```
import numpy as np
arr = np.array([[-1, 2, 0, 4],
                [4, -0.5, 6, 0],
                [2.6, 0, 7, 8],
                [3, -7, 4, 2.0]])
print("Initial Array: ")
print(arr)
sliced_arr = arr[:2, ::2]
```

```
print ("Array with first 2 rows and"
    " alternate columns(0 and 2):\n", sliced_arr)
Index_arr = arr[[1, 1, 0, 3],
                [3, 2, 1, 0]]
print ("\nElements at indices (1, 3), "
    "(1, 2), (0, 1), (3, 0):\n", Index_arr)
```

Output

```
Initial Array:
[[-1.   2.   0.   4. ]
 [ 4.  -0.5  6.   0. ]
 [ 2.6  0.   7.   8. ]
 [ 3.  -7.   4.   2. ]]
Array with first 2 rows and alternate columns(0 and 2):
 [[-1.  0.]
 [ 4.  6.]]

Elements at indices (1, 3), (1, 2), (0, 1), (3, 0):
 [0. 6. 2. 3.]


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Basic Array Operations

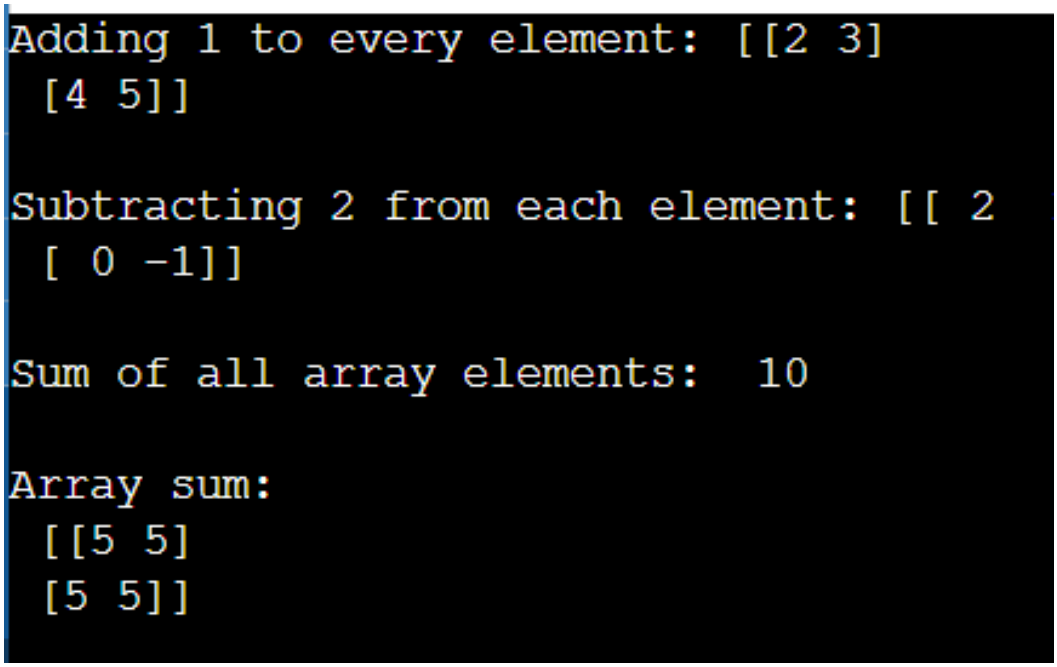
In numpy, arrays allow a wide range of operations which can be performed on a particular array or a combination of Arrays. These operation include some basic Mathematical operation as well as Unary and Binary operations.

➢ Program on basic Array Operations

```
import numpy as np
a = np.array([[1, 2],
              [3, 4]])
b = np.array([[4, 3],
              [2, 1]])
print ("Adding 1 to every element:", a + 1)
```

```
print ("\nSubtracting 2 from each element:", b - 2)
print ("\nSum of all array "
      "elements: ", a.sum())
print ("\nArray sum:\n", a + b)
```

Output



➢ Create Array of Fixed Size

Often, the element is of an array is originally unknown, but its size is known. Hence, NumPy offers several functions to create arrays with initial placeholder content.

This minimize the necessity of growing arrays, an expensive operation. For example: np.zeros, np.ones, np.full, np.empty, etc.

To create sequences of numbers, NumPy provides a function analogous to the range that returns arrays instead of lists.

➢ Program on Create Array of Fixed Size

```
import numpy as np
# Creating a 3X4 array with all zeros
c = np.zeros((3, 4))
```

```
print ("An array initialized with all zeros:\n", c)


# Create a constant value array of complex type
d = np.full((3, 3), 6, dtype = 'complex')
print ("An array initialized with all 6s."
       "Array type is complex:\n", d)


# Create an array with random values
e = np.random.random((2, 2))
print ("A random array:\n", e)
```

Output

```
An array initialized with all zeros:
 [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
An array initialized with all 6s.Array type is complex:
 [[6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]]
A random array:
 [[0.11358822 0.34802013]
 [0.59754595 0.04105465]]


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Create Using arange() Function

arange(): This function returns evenly spaced values within a given interval. Step size is specified**.**

➢ Program on Create Using arange() Function

```
import numpy as np
# Create a sequence of integers
# from 0 to 30 with steps of 5
```

```
f = np.arange(0, 30, 5)
print ("A sequential array with steps of 5:\n", f)
```

Output

```
A sequential array with steps of 5:
 [ 0  5 10 15 20 25]


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Create Using linspace() Function

linspace(): It returns evenly spaced values within a given interval.

➢ Program on Create Using arange() Function

```
import numpy as np
g = np.linspace(0, 5, 10)
print ("A sequential array with 10 values between"
                    "0 and 5:\n", g)
```

Output

```
A sequential array with 10 values between0 and 5:
 [0.         0.55555556 1.11111111 1.66666667 2.22222222 2.77777778
 3.33333333 3.88888889 4.44444444 5.        ]
```

➢ Reshaping Array using Reshape Method

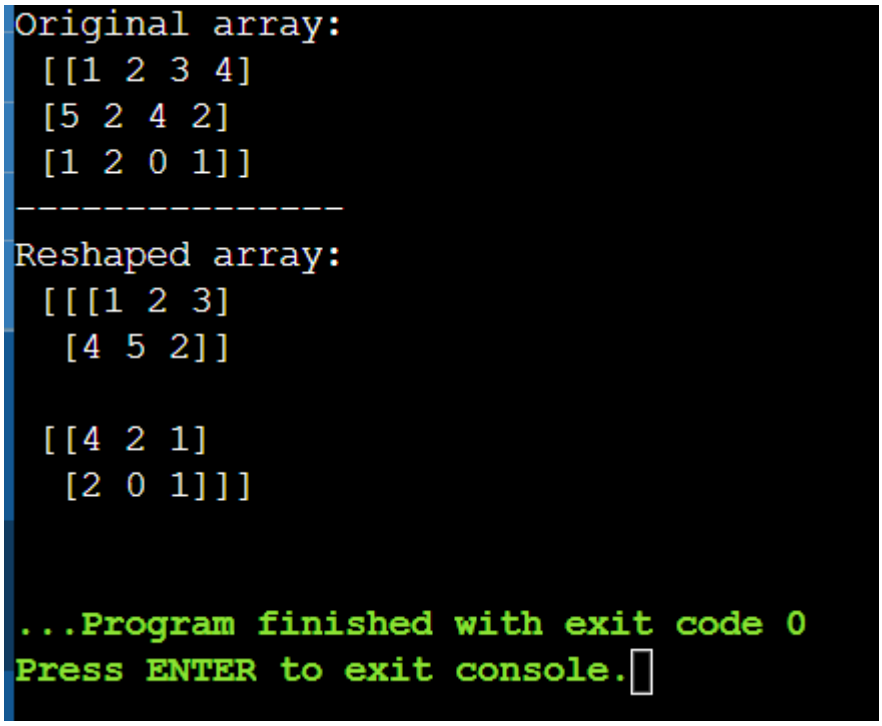
Reshaping array: We can use reshape method to reshape an array.

Consider an array with shape (a1, a2, a3, …, aN). We can reshape and convert it into another array with shape (b1, b2, b3, …, bM). The only required condition is a1 x a2 x a3 … x aN = b1 x b2 x b3 … x bM. (i.e. the original size of the array remains unchanged.)

➢ Program on Reshaping Array using Reshape Method

```
import numpy as np
arr = np.array([[1, 2, 3, 4],
                [5, 2, 4, 2],
                [1, 2, 0, 1]])
newarr = arr.reshape(2, 2, 3)
print ("Original array:\n", arr)
print("----------")
print ("Reshaped array:\n", newarr)
```

Output



➢ NumPy Array Indexing

Knowing the basics of NumPy array indexing is important for analyzing and manipulating the array object. NumPy in Python offers many ways to do array indexing.

- **Slicing:** Just like lists in Python, NumPy arrays can be sliced. As arrays can be multidimensional, you need to specify a slice for each dimension of the array.

- **Integer array indexing:** In this method, lists are passed for indexing for each dimension. One-to-one mapping of corresponding elements is done to construct a new arbitrary array.
- **Boolean array indexing:** This method is used when we want to pick elements from the array which satisfy some condition.

➢ Python program to demonstrate indexing in numpy

```
import numpy as np

arr = np.array([[-1, 2, 0, 4],
                [4, -0.5, 6, 0],
                [2.6, 0, 7, 8],
                [3, -7, 4, 2.0]])

temp = arr[:2, ::2]

print ("Array with first 2 rows and alternate"
                "columns(0 and 2):\n", temp)

temp = arr[[0, 1, 2, 3], [3, 2, 1, 0]]

print ("\nElements at indices (0, 3), (1, 2), (2, 1),"
                                "(3, 0):\n", temp)

cond = arr > 0 # cond is a boolean array

temp = arr[cond]

print ("\nElements greater than 0:\n", temp)
```

Output

```
Array with first 2 rows and alternatecolumns(0 and 2):
 [[-1.  0.]
 [ 4.  6.]]

Elements at indices (0, 3), (1, 2), (2, 1),(3, 0):
 [4. 6. 0. 3.]

Elements greater than 0:
 [2.  4.  4.  6.  2.6 7.  8.  3.  4.  2. ]
```

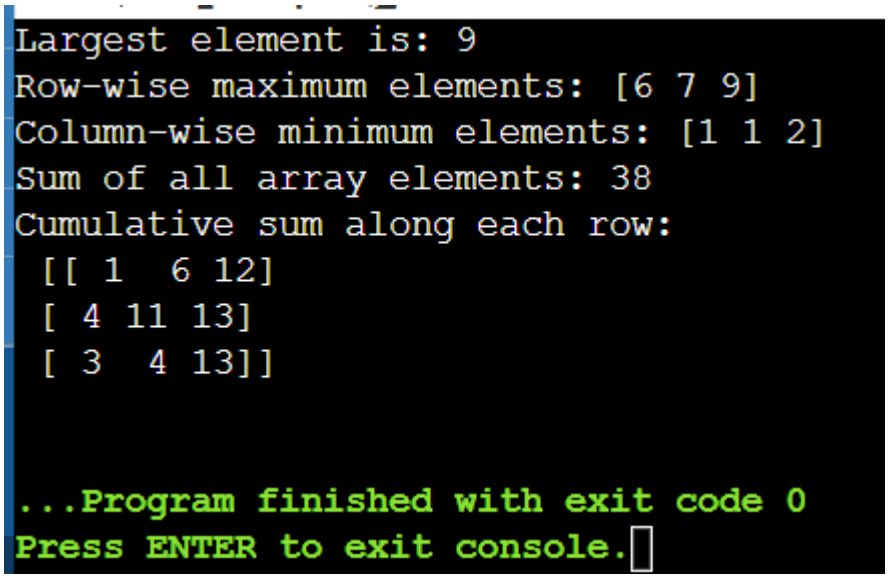
➢ NumPy – Unary Operators

Many unary operations are provided as a method of ndarray class. This includes sum, min, max, etc. These functions can also be applied row-wise or column-wise by setting an axis parameter.

➢ Python program to demonstrate unary operators in numpy

```
import numpy as np

arr = np.array([[1, 5, 6],
                [4, 7, 2],
                [3, 1, 9]])

print ("Largest element is:", arr.max())

print ("Row-wise maximum elements:",
                    arr.max(axis = 1))

print ("Column-wise minimum elements:",
                       arr.min(axis = 0))

print ("Sum of all array elements:", arr.sum())

print ("Cumulative sum along each row:\n",
                       arr.cumsum(axis = 1))
```

Output



➢ NumPy – Binary Operators

These operations apply to the array elementwise and a new array is created. You can use all basic arithmetic operators like +, -, /, etc. In the case of +=, -=, = operators, the existing array is modified.

➢ Python program to demonstrate binary operators in Numpy

```
import numpy as np
a = np.array([[1, 2],
              [3, 4]])
b = np.array([[4, 3],
              [2, 1]])
print ("Array sum:\n", a + b)
print ("Array multiplication:\n", a*b)
print ("Matrix multiplication:\n", a.dot(b))
```

Output

```
Array sum:
 [[5 5]
 [5 5]]
Array multiplication:
 [[4 6]
 [6 4]]
Matrix multiplication:
 [[ 8  5]
 [20 13]]


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Data Types in Numpy

Every Numpy array is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. Every ndarray has an associated data type (dtype) object. This data type object (dtype) provides information about the layout of

the array. The values of an ndarray are stored in a buffer which can be thought of as a contiguous block of memory bytes which can be interpreted by the dtype object. Numpy provides a large set of numeric datatypes that can be used to construct arrays. At the time of Array creation, Numpy tries to guess a datatype, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype.

➢ Constructing a Datatype Object

In Numpy, datatypes of Arrays need not to be defined unless a specific datatype is required. Numpy tries to guess the datatype for Arrays which are not predefined in the constructor function.

➢ Python Program to create a data type object

```
import numpy as np

x = np.array([1, 2])

print("Integer Datatype: ")

print(x.dtype)

x = np.array([1.0, 2.0])

print("\nFloat Datatype: ")

print(x.dtype)

x = np.array([1, 2], dtype = np.int64)

print("\nForcing a Datatype: ")

print(x.dtype)
```

Output

```
Integer Datatype:
int64

Float Datatype:
float64

Forcing a Datatype:
int64
```

➢ Math Operations on DataType array

In Numpy arrays, basic mathematical operations are performed element-wise on the array. These operations are applied both as operator overloads and as functions. Many useful functions are provided in Numpy for performing computations on Arrays such as sum: for addition of Array elements, T: for Transpose of elements, etc.

➢ Python Program to create a data type object

```
import numpy as np
arr1 = np.array([[4, 7], [2, 6]],
                dtype = np.float64)
# Second Array
arr2 = np.array([[3, 6], [2, 8]],
                dtype = np.float64)
Sum = np.add(arr1, arr2)
print("Addition of Two Arrays: ")
print(Sum)
Sum1 = np.sum(arr1)
print("\nAddition of Array elements: ")
print(Sum1)
Sqrt = np.sqrt(arr1)
print("\nSquare root of Array1 elements: ")
print(Sqrt)
Trans_arr = arr1.T
print("\nTranspose of Array: ")
print(Trans_arr)
```

Output

```
Addition of Two Arrays:
[[ 7. 13.]
 [ 4. 14.]]

Addition of Array elements:
19.0

Square root of Array1 elements:
[[2.         2.64575131]
 [1.41421356 2.44948974]]

Transpose of Array:
[[4. 2.]
 [7. 6.]]


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Numpy Linear Algebra

The Linear Algebra module of NumPy offers various methods to apply linear algebra on any numpy array.
One can find:

- rank, determinant, trace, etc. of an array.
- eigen values of matrices
- matrix and vector products (dot, inner, outer,etc. product), matrix exponentiation
- solve linear or tensor equations and much more!

Importing numpy as np

```
import numpy as np

A = np.array([[6, 1, 1],
              [4, -2, 5],
              [2, 8, 7]])
```

```
print("Rank of A:", np.linalg.matrix_rank(A))
print("\nTrace of A:", np.trace(A))
print("\nDeterminant of A:", np.linalg.det(A))
print("\nInverse of A:\n", np.linalg.inv(A))
print("\nMatrix A raised to power 3:\n",
      np.linalg.matrix_power(A, 3))
```

Output

```
Rank of A: 3

Trace of A: 11

Determinant of A: -306.0

Inverse of A:
 [[ 0.17647059 -0.00326797 -0.02287582]
 [ 0.05882353 -0.13071895  0.08496732]
 [-0.11764706  0.1503268   0.05228758]]

Matrix A raised to power 3:
 [[336 162 228]
 [406 162 469]
 [698 702 905]]


...Program finished with exit code 0
Press ENTER to exit console.
```

➢ Matrix eigenvalues Functions

numpy.linalg.eigh(a, UPLO='L') : This function is used to return the eigenvalues and eigenvectors of a complex Hermitian (conjugate symmetric) or a real symmetric matrix.Returns two objects, a 1-D array containing the eigenvalues of a, and a 2-D square array or matrix (depending on the input type) of the corresponding eigenvectors (in columns).

➢ Python Program to Matrix eigenvalues Functions

```
import numpy as np
from numpy import linalg as geek
a = np.array([[1, -2j], [2j, 5]])
print("Array is:", a)
c, d = geek.eigh(a)
print("Eigenvalues are:", c)
print("Eigenvectors are:", d)
```

Output

```
Array is: [[ 1.+0.j -0.-2.j]
 [ 0.+2.j  5.+0.j]]
Eigenvalues are: [0.17157288 5.82842712]
Eigenvectors are: [[-0.92387953+0.j         -0.38268343+0.j
 [ 0.        +0.38268343j  0.        -0.92387953j]]


...Program finished with exit code 0
Press ENTER to exit console.
```

numpy.linalg.eig(a) : This function is used to compute the eigenvalues and right eigenvectors of a square array.

➢ Python Program using numpy.linalg.eig

```
import numpy as np
from numpy import linalg as geek
a = np.diag((1, 2, 3))
print("Array is:", a)
c, d = geek.eig(a)
print("Eigenvalues are:", c)
print("Eigenvectors are:", d)
```

Output

```
Array is: [[1 0 0]
 [0 2 0]
 [0 0 3]]
Eigenvalues are: [1. 2. 3.]
Eigenvectors are: [[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

8. Python Programs for creation and manipulation of DataFrames using PandasLibrary

➢ Data Manipulation In Python Using Pandas

In Machine Learning, the model requires a dataset to operate, i.e. to train and test. But data doesn't come fully prepared and ready to use. There are discrepancies like Nan/ Null / NA values in many rows and columns. Sometimes the data set also contains some of the rows and columns which are not even required in the operation of our model. In such conditions, it requires proper cleaning and modification of the data set to make it an efficient input for our model. We achieve that by practicing Data Wrangling before giving data input to the model.

➢ Creating DataFrame

DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc.

- **Python program for creating dataframe**

```
# Importing the pandas library

import pandas as pd

student_register = pd.DataFrame()

student_register['Name'] = ['Abhijit','Smriti',
                            'Akash', 'Roshni']

student_register['Age'] = [20, 19, 20, 14]

student_register['Student'] = [False, True,
                               True, False]

print(student_register)
```

Output:

```
      Name  Age  Student
0  Abhijit   20    False
1   Smriti   19     True
2    Akash   20     True
3   Roshni   14    False
```

➢ Adding data in DataFrame using Append Function

append function is used to add rows of other dataframes to end of existing dataframe, returning a new dataframe object. Columns not in the original data frames are added as new columns and the new cells are populated with NaN value.

- **Program**

```
import pandas as pd

student_register = pd.DataFrame({
                'Name': ['John', 'Alice'],
                'Age': [20, 22],
                'Student': [True, False]
            })

new_person = pd.Series(['Mansi', 19, True], index=['Name', 'Age', 'Student'])

student_register = pd.concat([student_register, new_person.to_frame().T],
ignore_index=True)

print(student_register)
```

Output:

```
    Name Age Student
0   John  20    True
1  Alice  22   False
2  Mansi  19    True

=== Code Execution Successful ===
```

➢ Data Manipulation on Dataset

Data manipulation is a core task when working with datasets in Python, especially using libraries like pandas. These libraries provide a wide range of functionalities to load, clean, transform, and analyze data.

- **Getting Shape and information of the data**

```
# dimension of the dataframe

print('Shape: ')

print(student_register.shape)
```

```
print('-------------------------')
# showing info about the data
print('Info: ')
print(student_register.info())
print('-------------------------')
# correlation between columns
print('Correlation: ')
print(student_register.corr())
```

Output:

```
Shape:
(4, 3)
-------------------------------------
Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Name     4 non-null      object
 1   Age      4 non-null      int64
 2   Student  4 non-null      bool
dtypes: bool(1), int64(1), object(1)
memory usage: 196.0+ bytes
None
-------------------------------------
Correlation:
              Age   Student
Age      1.000000  0.502519
Student  0.502519  1.000000


...Program finished with exit code 0
Press ENTER to exit console.
```

- **Getting Statistical Analysis of Data**

```
# for showing the statistical
# info of the dataframe
print('Describe')
print(student_register.describe()
```

Output:

```
Describe
             Age
count   4.000000
mean   18.250000
std     2.872281
min    14.000000
25%    17.750000
50%    19.500000
75%    20.000000
max    20.000000


...Program finished with exit code 0
Press ENTER to exit console.
```

- **Dropping Columns from Data**

```
students = student_register.drop('Age', axis=1)
print(students.head())
```

Output:

```
      Name  Student
0  Abhijit    False
1   Smriti     True
2    Akash     True
3   Roshni    False


...Program finished with exit code 0
Press ENTER to exit console.
```

- **Dropping Rows from Data**

```
students = students.drop(2, axis=0)
print(students.head())
```

Output:

```
      Name  Student
0  Abhijit    False
1   Smriti     True
3   Roshni    False
```

9. Write a Python program for the following.

➢ Topics to be covered

- Simple Line Plots
- Adjusting the Plot: Line Colours and Styles, Axes Limits, Labelling Plots
- Simple Scatter Plots
- Histograms
- Customizing Plot Legends
- Choosing Elements for the Legend
- Boxplot
- Multiple Legends
- Customizing Colorbars
- Multiple Subplots
- Text and Annotation
- Customizing Ticks

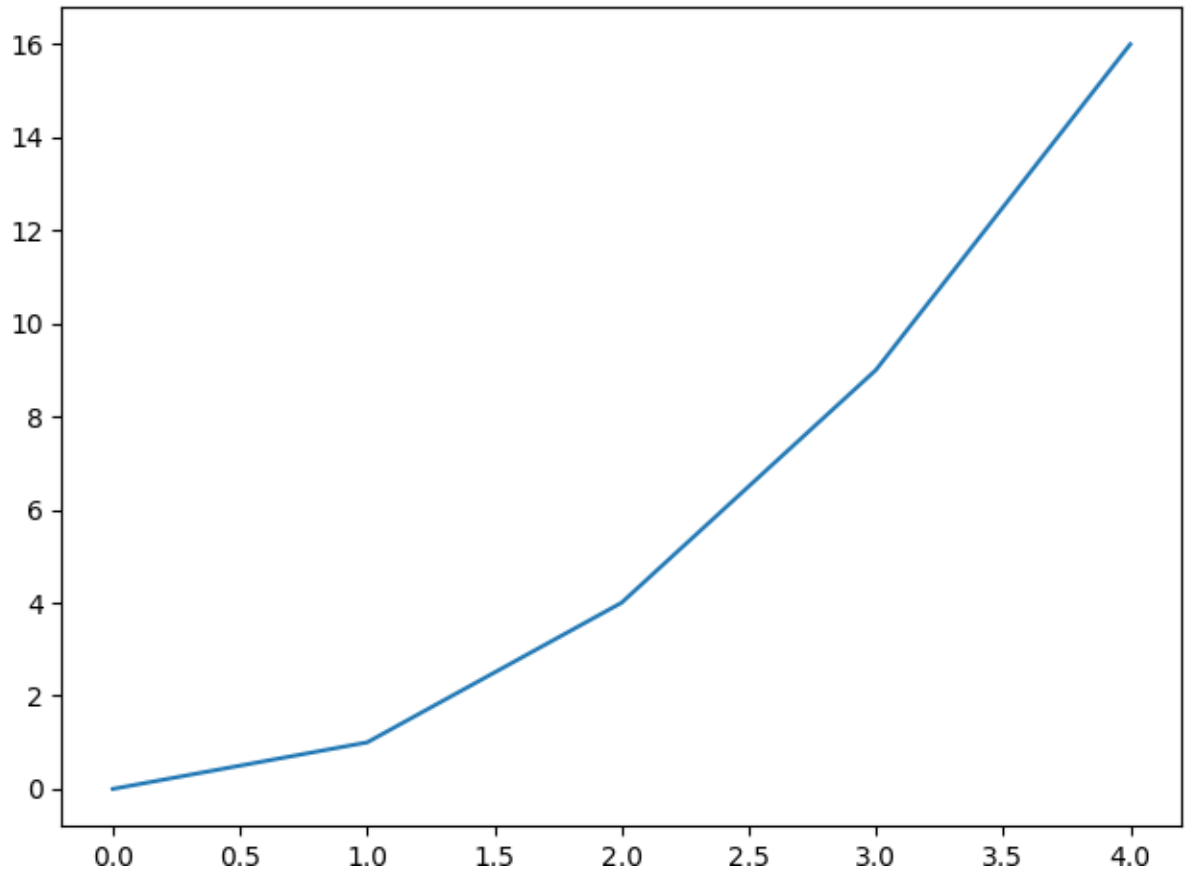
➢ Simple Line Plots

- A **simple line plot** in Python is a graphical representation of data where individual data points are connected by straight lines. It's commonly used to show trends over time or relationships between variables.
- To create a simple line plot in Python, you can use the matplotlib library. The line plot displays data on two axes (X and Y) with a line connecting each consecutive data point.
- Here's an example of creating a simple line plot:

➢ A program on simple line plot

```
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4]

y = [0, 1, 4, 9, 16]

plt.plot(x, y)

plt.show()
```

Output



Perhaps the simplest of all plots is the visualization of a single function y=f(x)y=f(x). Here we will take a first look at creating a simple plot of this type. As with all the following sections, we'll start by setting up the notebook for plotting and importing the packages we will use:

In[1]

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

For all Matplotlib plots, we start by creating a figure and an axes. In their simplest form, a figure and axes can be created as follows:

In[2]

```
fig = plt.figure()
ax = plt.axes()
```

In Matplotlib, the *figure* (an instance of the class plt.Figure) can be thought of as a single container that contains all the objects representing axes, graphics, text, and labels. The *axes* (an instance of the class plt.Axes) is what we see above: a bounding box with ticks and labels, which will eventually contain the plot elements that make up our visualization. Throughout this book, we'll commonly use the variable

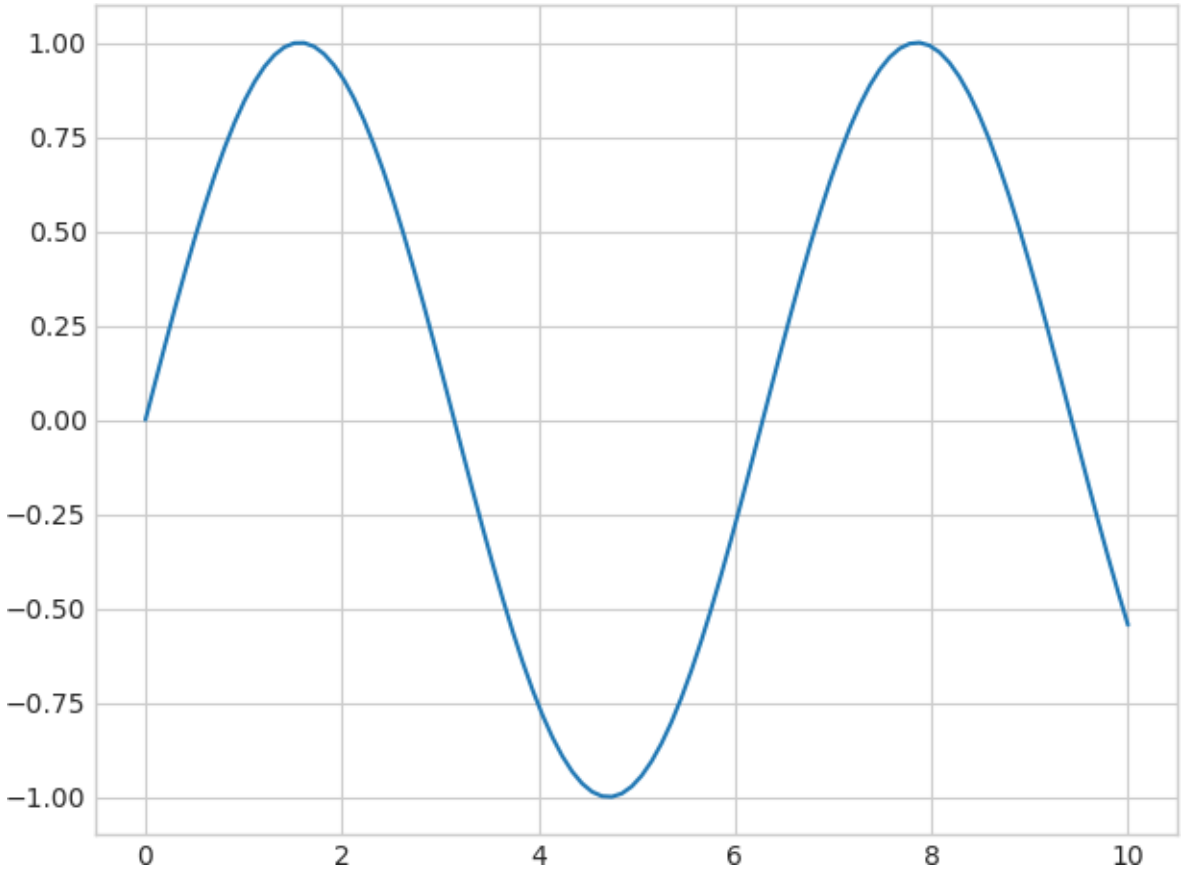
name `fig` to refer to a figure instance, and `ax` to refer to an axes instance or group of axes instances.

Once we have created an axes, we can use the `ax.plot` function to plot some data. Let's start with a simple sinusoid:

In[3]

```
x = np.linspace(0, 10, 100)
ax.plot(x, np.sin(x)
plt.show()
```

Output

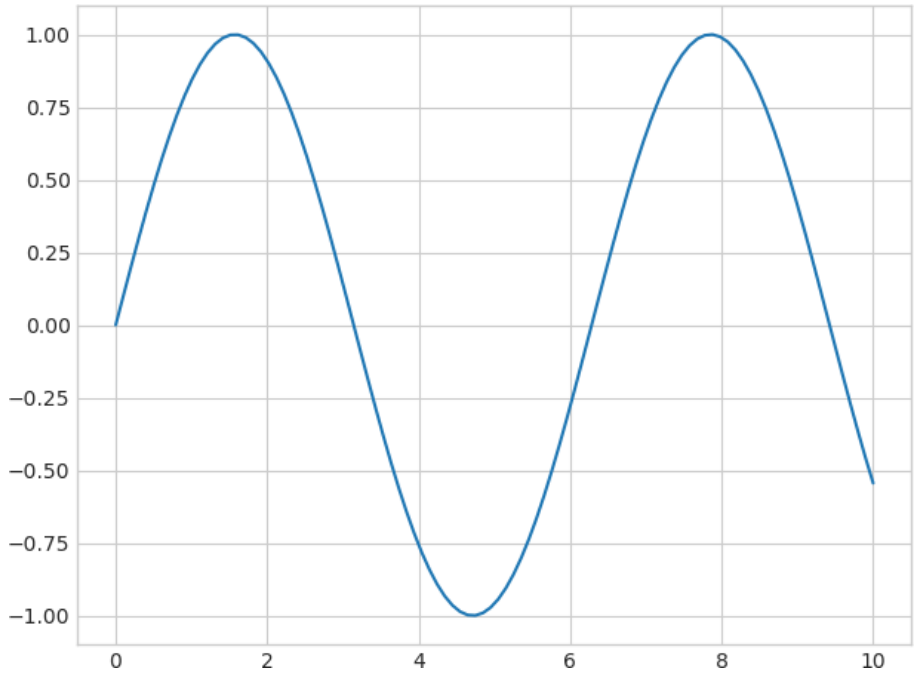


Alternatively, we can use the pylab interface and let the figure and axes be created for us in the background (see Two Interfaces for the Price of One for a discussion of these two interfaces):

In[4]

```
ax.plot(x, np.sin(x))
```

Output

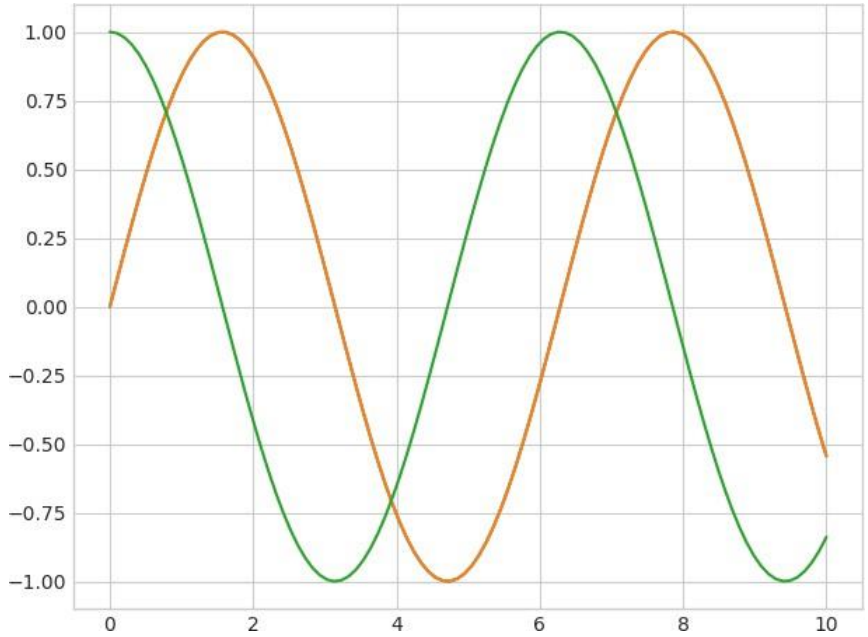


If we want to create a single figure with multiple lines, we can simply call the plot function multiple times:

In[5]

```
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
```

Output



That's all there is to plotting simple functions in Matplotlib! We'll now dive into some more details about how to control the appearance of the axes and lines.

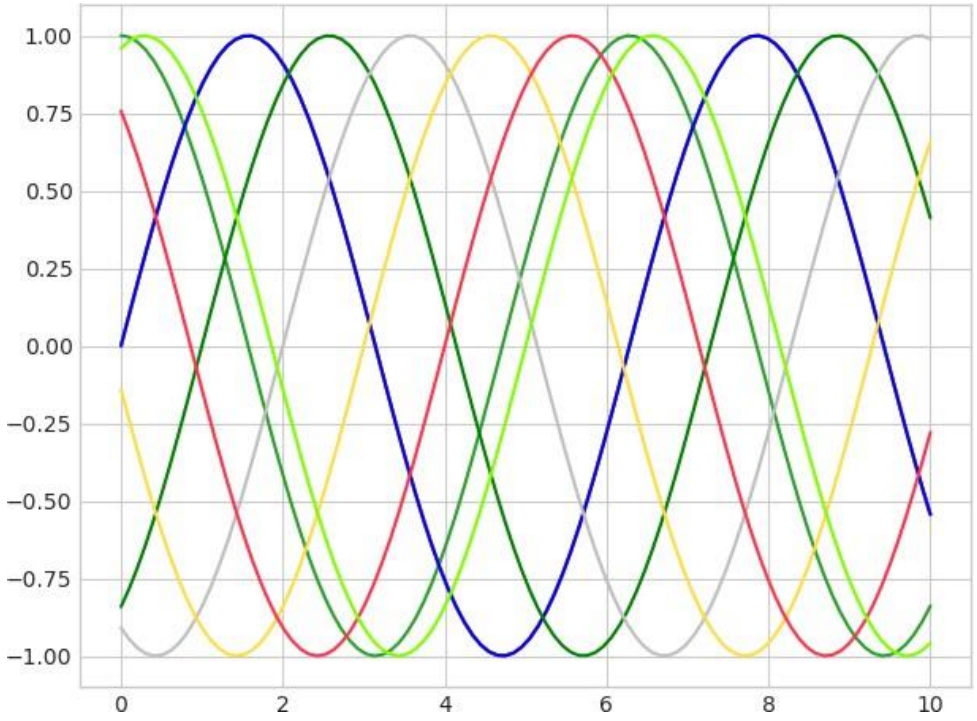
➢ Adjusting the Plot: Line Colors and Styles

The first adjustment you might wish to make to a plot is to control the line colors and styles. The `plt.plot()` function takes additional arguments that can be used to specify these. To adjust the color, you can use the `color` keyword, which accepts a string argument representing virtually any imaginable color. The color can be specified in a variety of ways:

In[6]

```
plt.plot(x, np.sin(x - 0), color='blue')
plt.plot(x, np.sin(x - 1), color='g')
plt.plot(x, np.sin(x - 2), color='0.75')
plt.plot(x, np.sin(x - 3), color='#FFDD44')
plt.plot(x, np.sin(x - 4), color=(1.0,0.2,0.3))
plt.plot(x, np.sin(x - 5), color='chartreuse')
```

Output

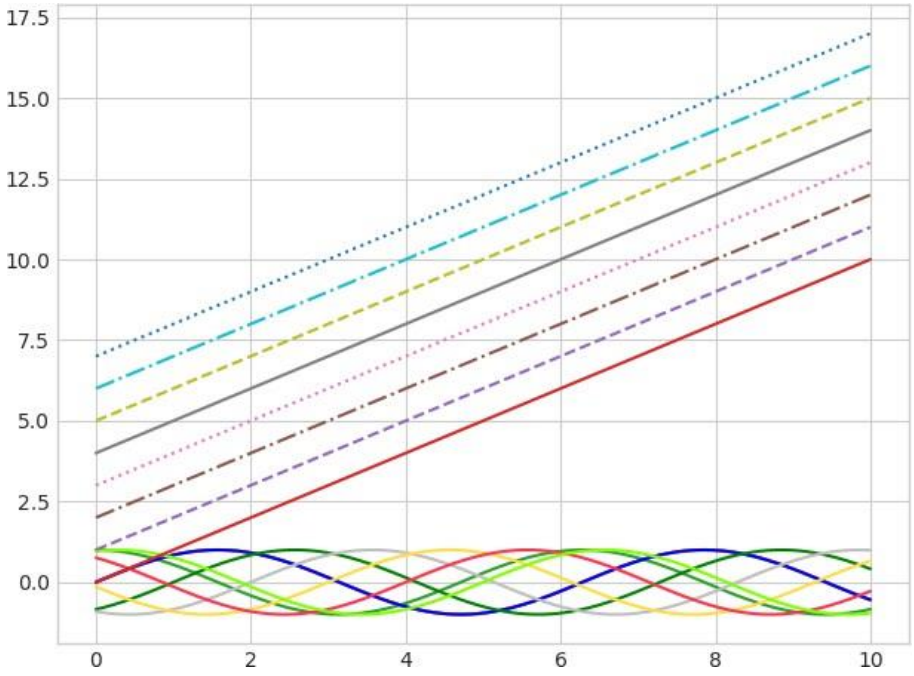


Similarly, the line style can be adjusted using the `linestyle` keyword:

In[7]

```
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted')
plt.plot(x, x + 4, linestyle='-')
plt.plot(x, x + 5, linestyle='--')
plt.plot(x, x + 6, linestyle='-.')
plt.plot(x, x + 7, linestyle=':')
```

Output

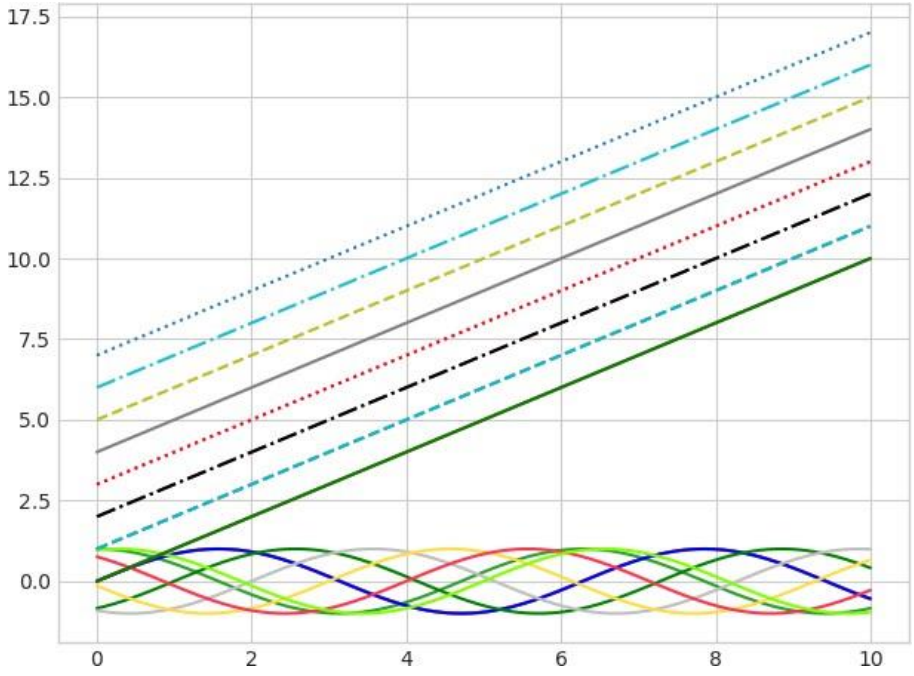


If you would like to be extremely terse, these linestyle and color codes can be combined into a single non-keyword argument to the plt.plot() function:

In[8]

```
plt.plot(x, x + 0, '-g')
plt.plot(x, x + 1, '--c')
plt.plot(x, x + 2, '-.k')
plt.plot(x, x + 3, ':r')
```

Output



➢ Adjusting the Plot: Axes Limits

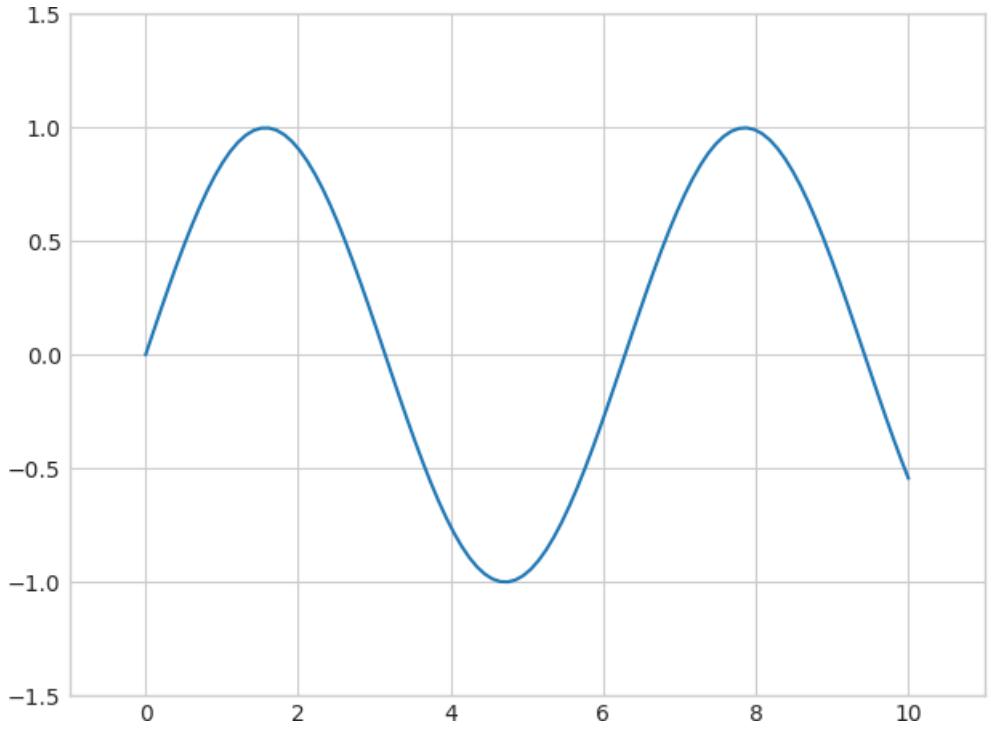
Matplotlib does a decent job of choosing default axes limits for your plot, but sometimes it's nice to have finer control. The most basic way to adjust axis limits is to use the plt.xlim() and plt.ylim() methods:

Along with **In[1]**, **In[2]**, **In[3]** and **In[4].**

In[9]

```
plt.xlim(-1, 11)
plt.ylim(-1.5, 1.5)
plt.show()
```

Output



If for some reason you'd like either axis to be displayed in reverse, you can simply reverse the order of the arguments:

In[10]

```
plt.plot(x, np.sin(x))
plt.xlim(10, 0)
plt.ylim(1.2, -1.2);
```

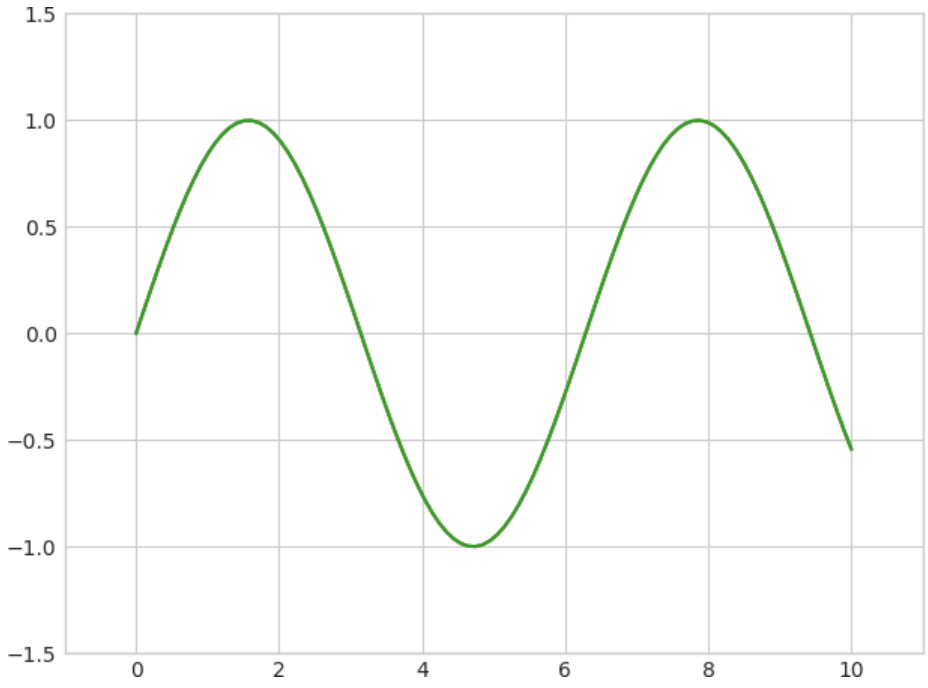
Output

A useful related method is `plt.axis()` (note here the potential confusion between *axes* with an *e*, and *axis* with an *i*). The `plt.axis()` method allows you to set the `x` and `y` limits with a single call, by passing a list which specifies `[xmin, xmax, ymin, ymax]`:

In[11]

```
plt.plot(x, np.sin(x))
plt.axis([-1, 11, -1.5, 1.5]);
```

Output

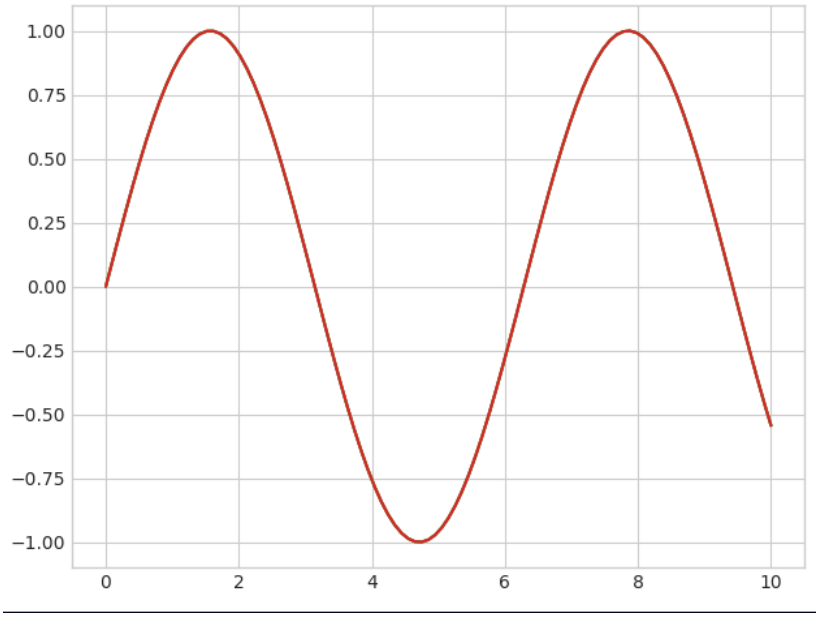


The plt.axis() method goes even beyond this, allowing you to do things like automatically tighten the bounds around the current plot:

In[12]

```
plt.plot(x, np.sin(x))
plt.axis('tight');
```

Output

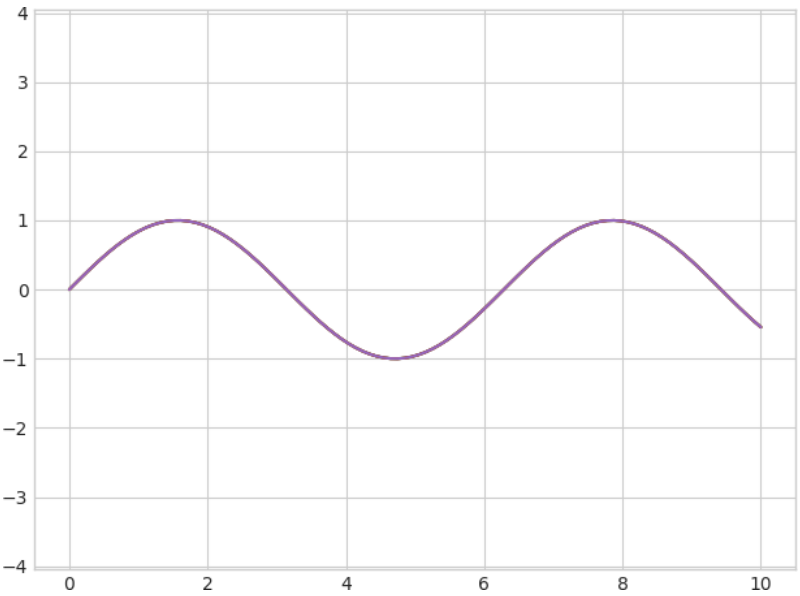


It allows even higher-level specifications, such as ensuring an equal aspect ratio so that on your screen, one unit in x is equal to one unit in y:

In[12]

```
plt.plot(x, np.sin(x))
plt.axis('equal');
```

Output



➢ Labeling Plots

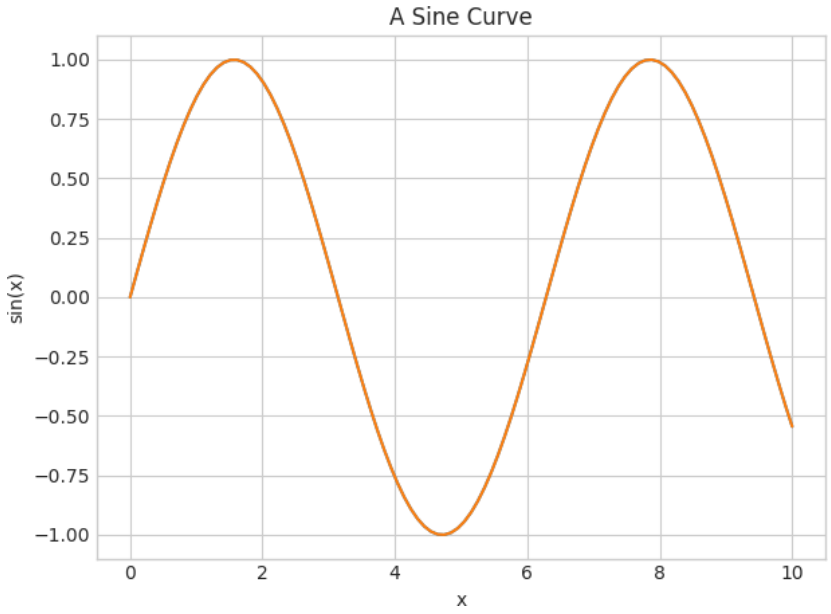
As the last piece of this section, we'll briefly look at the labeling of plots: titles, axis labels, and simple legends.

Titles and axis labels are the simplest such labels—there are methods that can be used to quickly set them:
Along with **In[1]**, **In[2]**, **In[3]** and **In[4].**

In[13]

```
plt.plot(x, np.sin(x))
plt.title("A Sine Curve")
plt.xlabel("x")
plt.ylabel("sin(x)")
```

Output



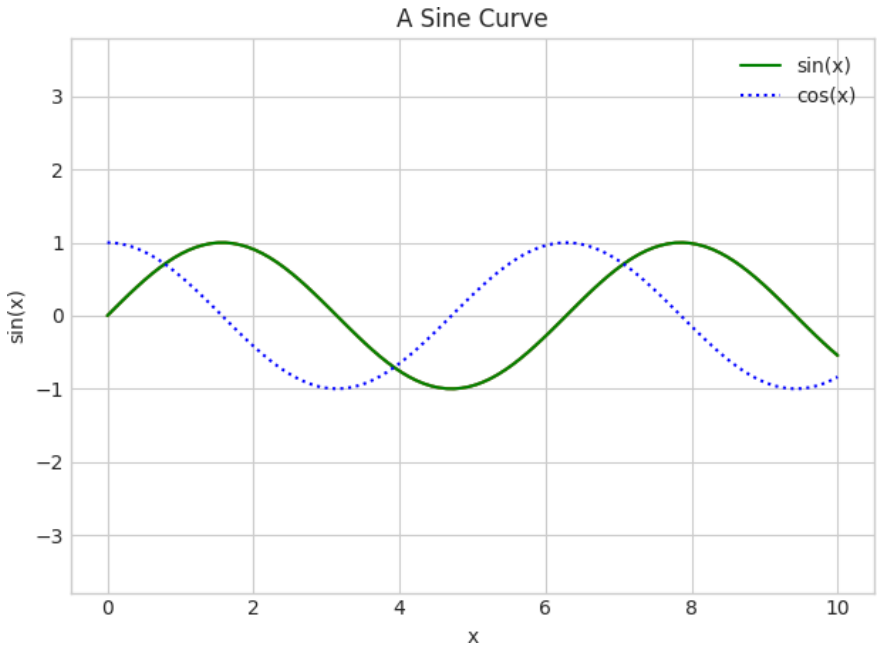
The position, size, and style of these labels can be adjusted using optional arguments to the function. For more information, see the Matplotlib documentation and the docstrings of each of these functions.

When multiple lines are being shown within a single axes, it can be useful to create a plot legend that labels each line type. Again, Matplotlib has a built-in way of quickly creating such a legend. It is done via the (you guessed it) plt.legend() method. Though there are several valid ways of using this, I find it easiest to specify the label of each line using the label keyword of the plot function:

In[14]

```
plt.plot(x, np.sin(x), '-g', label='sin(x)')
plt.plot(x, np.cos(x), ':b', label='cos(x)')
plt.axis('equal')
plt.legend();
```

Output



As you can see, the plt.legend() function keeps track of the line style and color, and matches these with the correct label. More information on specifying and formatting plot legends can be found in the plt.legend docstring; additionally, we will cover some more advanced legend options in Customizing Plot Legends.

➢ Aside: Matplotlib Gotchas

While most plt functions translate directly to ax methods (such as plt.plot() → ax.plot(), plt.legend() → ax.legend(), etc.), this is not the case for all commands. In particular, functions to set limits, labels, and titles are slightly modified. For transitioning between MATLAB-style functions and object-oriented methods, make the following changes:

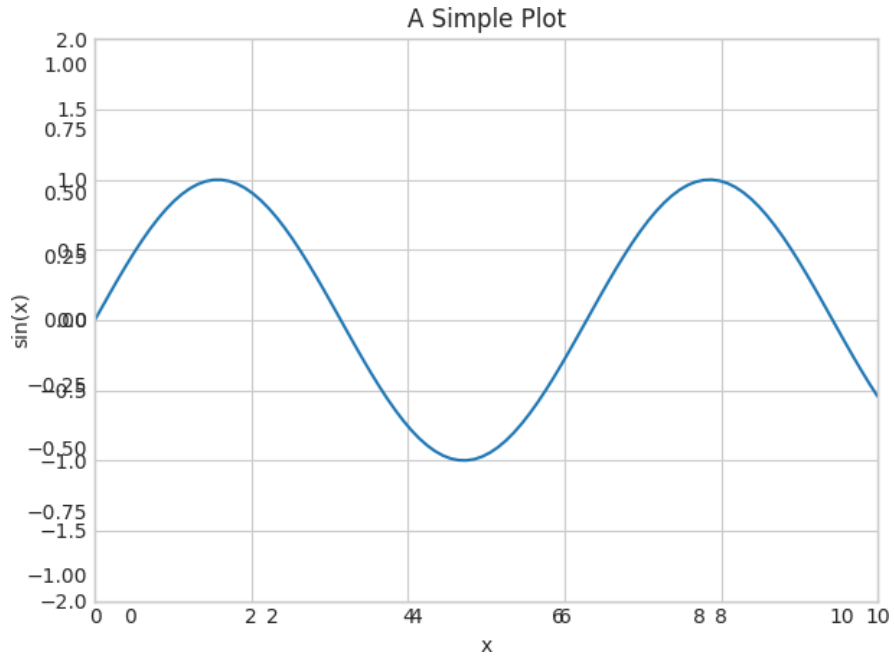
- plt.xlabel() → ax.set_xlabel()
- plt.ylabel() → ax.set_ylabel()
- plt.xlim() → ax.set_xlim()
- plt.ylim() → ax.set_ylim()
- plt.title() → ax.set_title()

In the object-oriented interface to plotting, rather than calling these functions individually, it is often more convenient to use the ax.set() method to set all these properties at once:

In[16]

```
ax = plt.axes()
ax.plot(x, np.sin(x))
ax.set(xlim=(0, 10), ylim=(-2, 2),
       xlabel='x', ylabel='sin(x)',
       title='A Simple Plot');
```

Output



➢ Simple Scatter Plots

There are various ways of creating plots using matplotlib.pyplot.scatter() in Python, There are some examples that illustrate the matplotlib.

pyplot.scatter() function in matplotlib.plot:

- Basic Scatter Plot
- Scatter Plot With Multiple Datasets
- Bubble Chart Plot
- Customized Scatter Plot

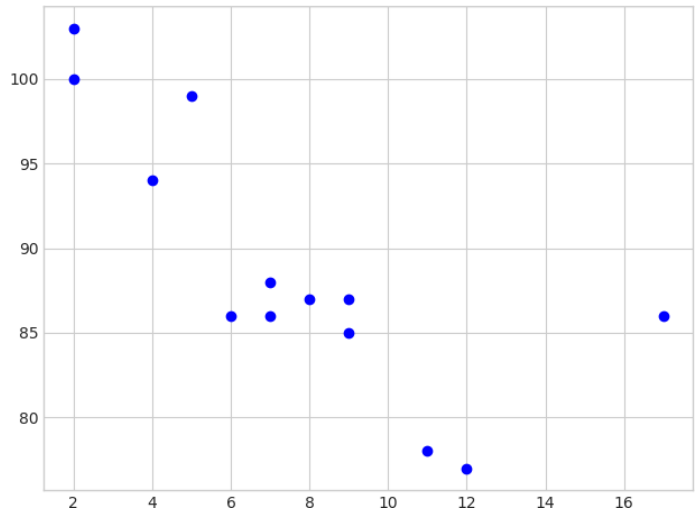
Scatter Plot in Matplotlib

By importing matpltlib. plot () we created a scatter plot. It defines x and y coordinates, then plots the points in blue and displays the plot.

➢ A python program on Scatter Plot

```
import matplotlib.pyplot as plt
x =[5, 7, 8, 7, 2, 17, 2, 9,
        4, 11, 12, 9, 6]
y =[99, 86, 87, 88, 100, 86,
        103, 87, 94, 78, 77, 85, 86]
plt.scatter(x, y, c ="blue")
# To show the plot
plt.show()
```

Output



➢ Plot Multiple Datasets on a Scatterplot

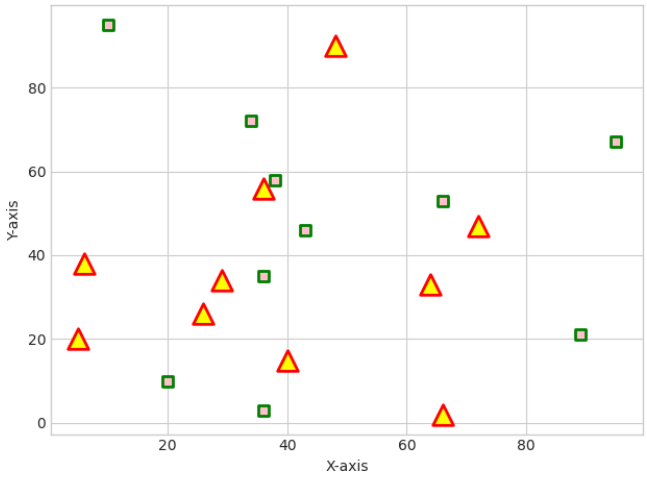
The below code generates a scatter plot showcasing two distinct datasets, each with its set of x and y coordinates. The code employs different markers, colors, and styling options for enhanced visualization.

➢ A python program on Scatter Plot

```
import matplotlib.pyplot as plt
x1 = [89, 43, 36, 36, 95, 10,
        66, 34, 38, 20]
```

```
y1 = [21, 46, 3, 35, 67, 95,
      53, 72, 58, 10]
x2 = [26, 29, 48, 64, 6, 5,
      36, 66, 72, 40]
y2 = [26, 34, 90, 33, 38,
      20, 56, 2, 47, 15]
plt.scatter(x1, y1, c ="pink",
                    linewidths = 2,
                    marker ="s",
                    edgecolor ="green",
                    s = 50)
plt.scatter(x2, y2, c ="yellow",
                    linewidths = 2,
                    marker ="^",
                    edgecolor ="red",
                    s = 200)
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

Output



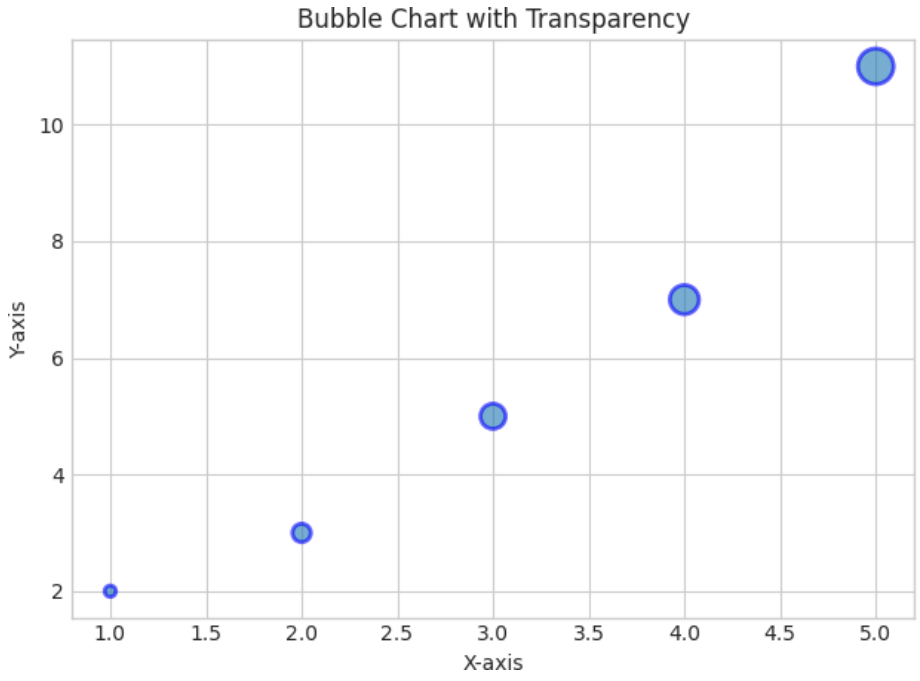
➢ Bubble Plots in Matplotlib

This code generates a bubble chart using Matplotlib. It plots points with specified x and y coordinates, each represented by a bubble with a size determined by the bubble_sizes list. The chart has customization for transparency, edge color, and linewidth. Finally, it displays the plot with a title and axis labels.

➢ A python program on Bubble Plots

```
import matplotlib.pyplot as plt

x_values = [1, 2, 3, 4, 5]

y_values = [2, 3, 5, 7, 11]

bubble_sizes = [30, 80, 150, 200, 300]

plt.scatter(x_values, y_values, s=bubble_sizes, alpha=0.6, edgecolors='b',
linewidths=2)

plt.title("Bubble Chart with Transparency")

plt.xlabel("X-axis")

plt.ylabel("Y-axis")

plt.show()
```

Output



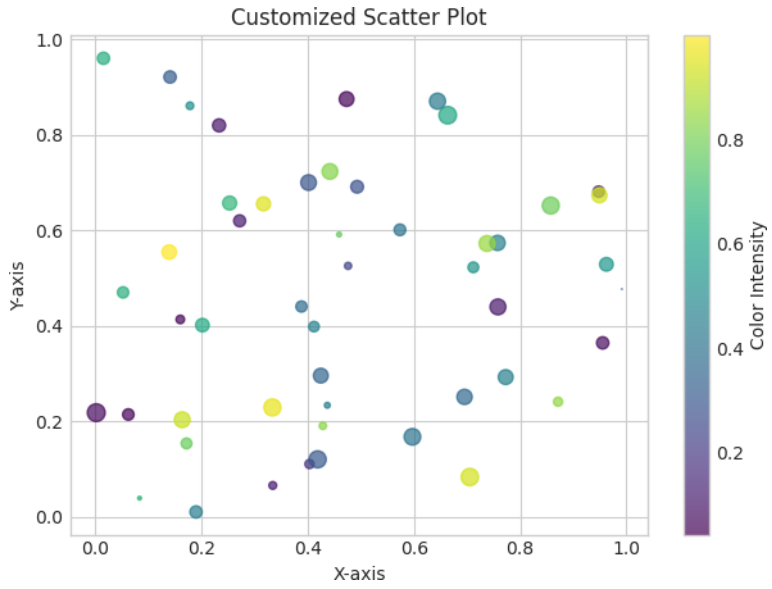
➢ Custom a Matplotlib Scatterplot

By importing Matplotlib we create a customized scatter plot using Matplotlib and NumPy. It generates random data for x and y coordinates, colors, and sizes. The scatter plot is then created with customized properties such as color, size, transparency, and colormap. The plot includes a title, axis labels, and a color intensity scale. Finally, the plot is displayed

➢ A python program on Bubble Plots

```
import matplotlib.pyplot as plt

import numpy as np

x = np.random.rand(50)

y = np.random.rand(50)

colors = np.random.rand(50)

sizes = 100 * np.random.rand(50)

plt.scatter(x, y, c=colors, s=sizes, alpha=0.7, cmap='viridis')

plt.title("Customized Scatter Plot")

plt.xlabel("X-axis")

plt.ylabel("Y-axis")

plt.colorbar(label='Color Intensity')

plt.show()
```

Output

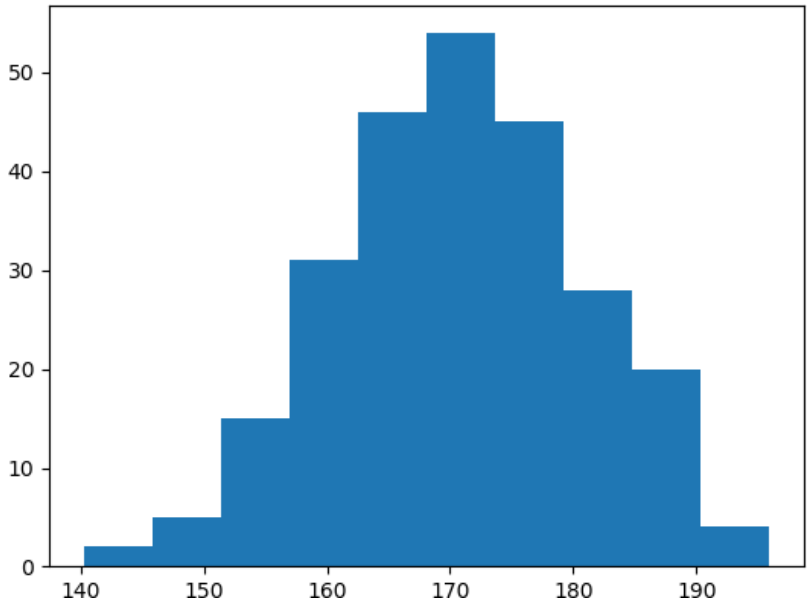


➢ Histogram

A histogram is a graph showing frequency distributions.

It is a graph showing the number of observations within each given interval.

Example: Say you ask for the height of 250 people, you might end up with a histogram like this:



You can read from the histogram that there are approximately:

2 people from 140 to 145cm

5 people from 145 to 150cm

15 people from 151 to 156cm

31 people from 157 to 162cm

46 people from 163 to 168cm

53 people from 168 to 173cm

45 people from 173 to 178cm

28 people from 179 to 184cm

21 people from 185 to 190cm

4 people from 190 to 195cm

A histogram in Python is a graphical representation of the distribution of a set of data. It divides the data into bins (intervals), and for each bin, the histogram shows how many data points fall into that interval. Histograms are useful for understanding the distribution, frequency, and spread of numerical data.

In Python, you can create histograms using libraries like Matplotlib and Seaborn.

➢ Key Components:

- Data: A dataset is required to generate a histogram. Here, np.random.randn(1000) generates 1000 random values following a normal distribution.
- plt.hist(): This function creates the histogram. The bins argument defines the number of intervals to divide the data into.
- plt.show(): This displays the histogram.

➢ How a Histogram Helps:

- Distribution: It shows how data points are spread across different intervals (e.g., if they are normally distributed).
- Identifying patterns: You can easily identify if the data follows any particular shape like normal distribution, skewed, etc.
- Outliers**:** Large spikes or gaps in the histogram can help detect outliers in your data.
- plt.hist(): This function creates the histogram. The bins argument defines the number of intervals to divide the data into.
- plt.show(): This displays the histogram.

➢ Create Histogram

In Matplotlib, we use the hist() function to create histograms.

The hist() function will use an array of numbers to create a histogram, the array is sent into the function as an argument.

For simplicity we use NumPy to randomly generate an array with 250 values, where the values will concentrate around 170, and the standard deviation is 10.

➢ A python program to create Histogram

```
import numpy as np

x = np.random.normal(170, 10, 250)

print(x)
```

Output

```
[170.71685041 176.13849698 170.80055826 174.91180044 178.15911012
 163.4956884  163.97759784 174.28376778 176.2297849  174.17900846
 166.43202186 177.4583412  168.87627787 179.08323399 174.15173669
 183.13182099 178.46294176 182.96682355 187.64961011 151.55832288
 157.71746389 177.56693718 171.83451512 183.53637757 162.78619641
 161.68154524 177.19424348 164.51360276 184.53268594 162.89061534
 162.53981975 149.17233204 169.27862775 166.37307796 139.45625201
 172.47136667 181.06849913 179.63363661 171.70405345 176.95995671
 175.81068428 172.76671301 197.51231317 182.7883077  176.35036948
 171.45712731 176.16331221 175.72965604 183.65477177 175.24552908
 171.86294316 172.03337168 193.55067995 162.81725377 173.56638448
 172.89664489 165.60851565 181.20265626 173.60617354 178.8510636
 159.52303924 176.37112018 180.40251243 156.23933166 168.03443263
 152.84044036 195.33224943 179.44497726 177.50921974 166.30537427
 171.53800817 184.1685211  176.58260813 183.10689874 180.75822594
 172.16542385 170.01560698 168.23730101 172.79981683 181.92275649
 170.32048548 177.00985925 175.77094217 160.20175372 171.411375
 172.77700163 181.44259315 156.14429471 173.58770484 182.8714354
 168.34527707 172.47659601 179.77336363 175.8951556  180.09917569
 167.34533749 170.76478006 173.24358773 183.89979932 174.0267926
 180.2330755  160.47939163 171.64749458 178.86596187 168.61769161
 146.67593052 192.12365805 171.31064937 182.86326447 177.71028394
 171.80588075 173.78232243 172.99920032 176.61652515 176.64077186
 169.79479658 146.91182514 185.57802425 178.72420874 187.75241236
 156.97544131 171.91357958 167.41925371 170.86598145 173.99003676
 153.72814415 173.18123189 171.49946251 167.45891358 170.81914941
 180.06834303 188.32845321 167.08982334 188.56033459 168.52036714
 168.08653336 159.33499077 171.59294396 189.07233609 161.83409254
 159.15603143 171.78262038 167.19349969 162.29274041 158.23120829
 154.54082398 149.26316835 164.56892226 156.88030257 164.20958319
 162.13745668 165.57605585 169.34560631 173.05114944 179.12995457
 166.17116536 161.94982207 172.40652952 173.57048911 168.97736679
 152.27823859 151.89118589 170.68840125 159.66589978 164.60130078
 161.64381141 186.43377836 161.08777567 196.01921018 174.85425455
 170.43506405 150.1739587  161.59438887 177.59667454 186.06677078
 172.09541529 181.29232117 162.00976952 158.12887682 178.8643164
 176.21034923 170.49466605 183.68492378 176.219061   168.80678951
 178.39044877 175.88103067 160.41097078 169.33426683 178.19346143
 166.67874702 174.38401146 163.7202042  165.16300879 152.45581477
 151.07915951 171.24511447 185.50929321 159.14639555 179.95795033
 171.87070915 170.87841081 158.04069217 192.27533649 178.53465054
 161.79105153 168.80817952 156.50262495 168.10002478 167.95580074
 164.55471101 167.88124544 170.42202145 177.95413427 171.05389033
 169.66339047 157.65083109 165.2983389  151.68154141 159.44290411
 175.69061854 160.8137713  167.97253157 178.3439643  173.6075216
 166.50795605 156.59458343 175.75902984 162.56690919 178.70716387
 188.21570507 192.63886118 164.86979912 182.98311663 177.59490579
 162.39197409 167.37471843 184.6508065  165.20573965 161.29188751
 176.01548927 154.07117636 168.01394949 185.03094057 174.18049186
 160.82632053 190.98005695 172.15473733 171.27480051 169.90424941]
```

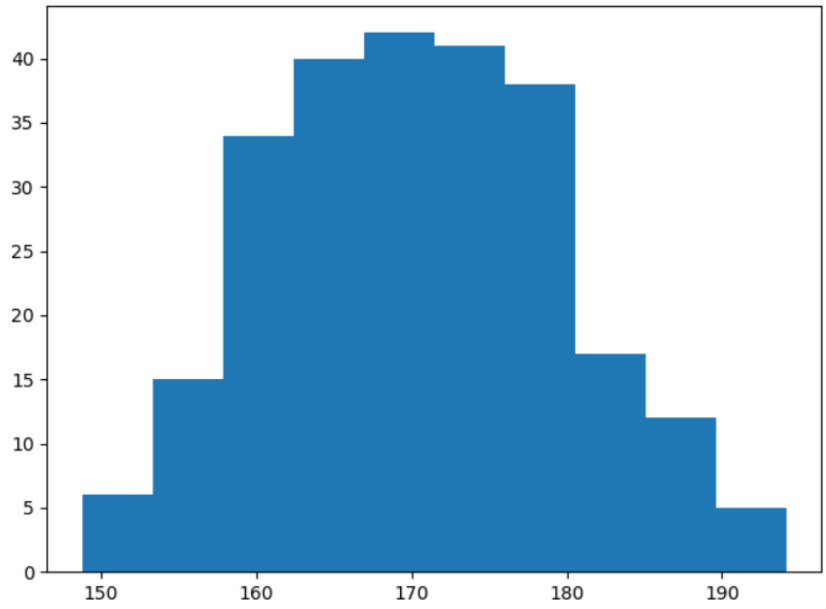
The hist() function will read the array and produce a histogram:

Example

A simple histogram:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.normal(170, 10, 250)
plt.hist(x)
plt.show()
```

Output



➢ Customizing Plot Legends

Creating custom legends in Python is a common requirement when using plotting libraries like Matplotlib. Legends provide important context to interpret a plot effectively. Here's a guide to composing custom legends:

1. Using plt.legend() with Custom Labels

When you have plotted multiple datasets, you can provide custom labels for the legend using the label parameter in plot commands and call plt.legend().

➢ A python program on Using plt.legend() with Custom Labels

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]

y1 = [1, 4, 9, 16, 25]

y2 = [1, 2, 3, 4, 5]

plt.plot(x, y1, label='Squared Values')

plt.plot(x, y2, label='Linear Values')

plt.legend()

plt.title('Custom Legend Example')

plt.show()
```

1. Composing Custom Legends with Line2D

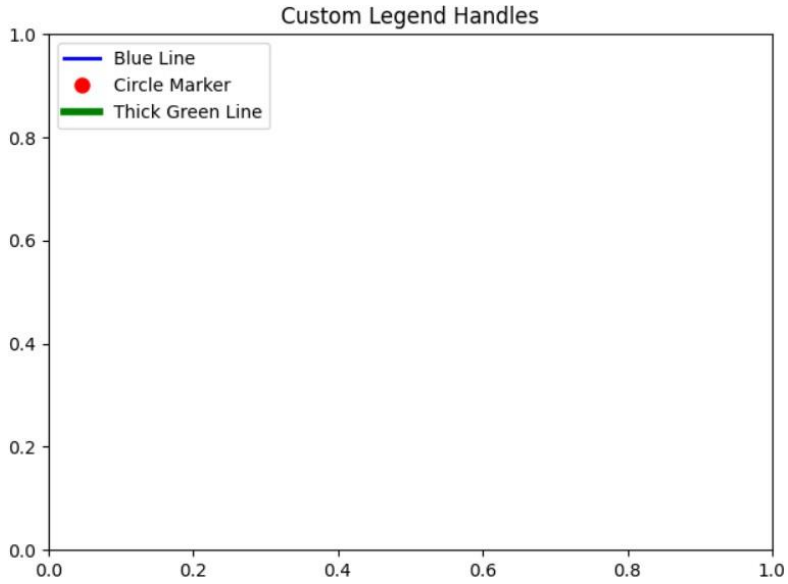
For more control, you can create custom legend handles using matplotlib.lines.Line2D.

➢ A python program on Composing Custom Legends with Line2D

```
from matplotlib.lines import Line2D

import matplotlib.pyplot as plt

legend_elements = [

    Line2D([0], [0], color='b', lw=2, label='Blue Line'),

    Line2D([0], [0], marker='o', color='w', label='Circle Marker', markerfacecolor='r',
markersize=10),

    Line2D([0], [0], color='g', lw=4, label='Thick Green Line')

]

plt.figure()
```

```
plt.legend(handles=legend_elements, loc='upper left')
plt.title('Custom Legend Handles')
plt.show()
```

Output



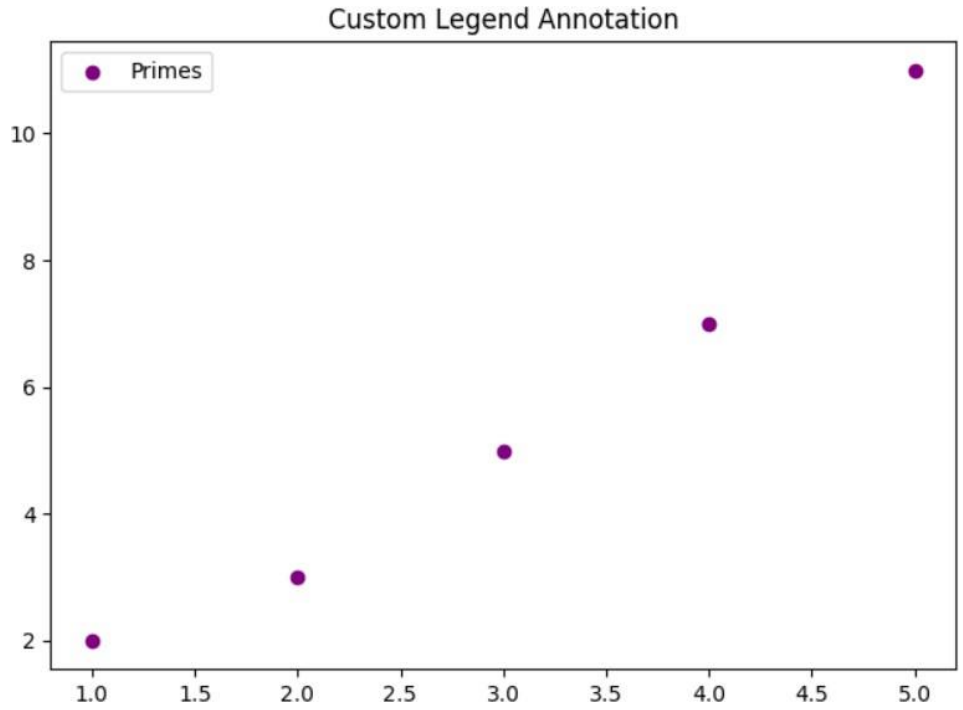
2. Adding Custom Annotations in the Legend

You can customize legend entries by combining various markers, lines, or text.

➢ A python program on Adding Custom Annotations in the Legend

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]
plt.scatter(x, y, color='purple', label='Prime Numbers')
plt.legend(['Primes'])
plt.title('Custom Legend Annotation')
plt.show()
```

Output



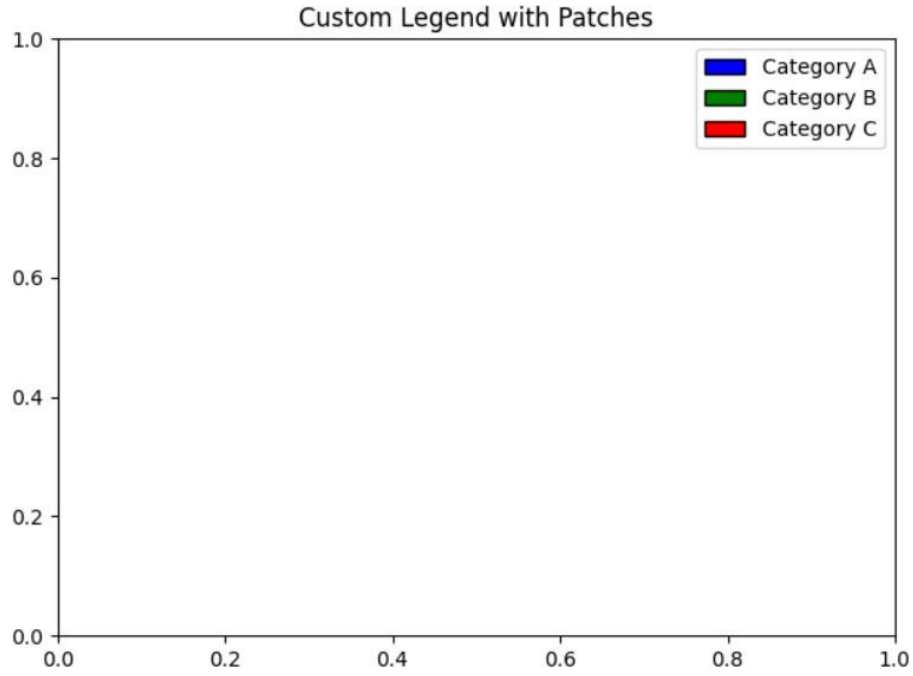
3. Using Patch Handles for Custom Shapes

If you're using filled shapes like bars or regions, matplotlib.patches.Patch can create custom legend entries.

➢ A python program on Using Patch Handles for Custom Shapes

```
from matplotlib.patches import Patch
import matplotlib.pyplot as plt
legend_elements = [
    Patch(facecolor='blue', edgecolor='black', label='Category A'),
    Patch(facecolor='green', edgecolor='black', label='Category B'),
    Patch(facecolor='red', edgecolor='black', label='Category C')
]
plt.figure()
plt.legend(handles=legend_elements, loc='upper right')
plt.title('Custom Legend with Patches')
plt.show()
```

Output



➢ Controlling Legend Location and Appearance

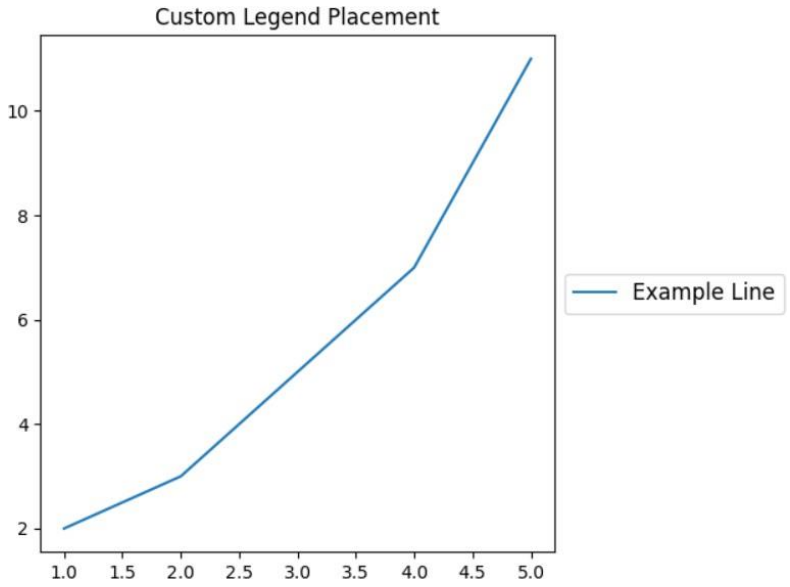
You can control the placement and appearance of legends using parameters such as:

- loc: Specifies the location (e.g., 'upper right', 'lower left', or (x, y) coordinates).
- bbox_to_anchor: Adjusts the legend position relative to the axes.
- fontsize: Changes the font size of the legend.
- frameon: Toggles the legend box.

➢ A python program on Controlling Legend Location and Appearance

```
import matplotlib.pyplot as plt
plt.plot(x, y, label='Example Line')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5), fontsize='large', frameon=True)
plt.title('Custom Legend Placement')
plt.show()
```

Output


Custom Legend Placement
10
8
6
4
2
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Example Line

➢ Choosing Elements for the Legend

When choosing elements for the legend in Python using Matplotlib, you can selectively include or exclude specific elements, customize their appearance, or define their order in the legend. Below are various approaches to control and refine the elements of the legend.

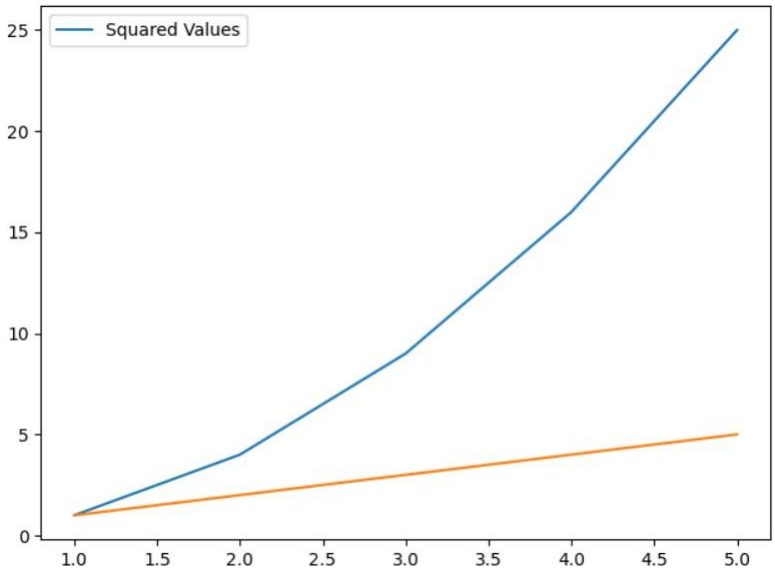
1. Use label and plt.legend()

- Use the label parameter in plotting commands to name the items you want in the legend.
- To exclude elements from the legend, simply omit the label or set it as _nolegend_.

➢ A python program on Use label and plt.legend()

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]

y1 = [1, 4, 9, 16, 25]

y2 = [1, 2, 3, 4, 5]

plt.plot(x, y1, label='Squared Values')

plt.plot(x, y2)  # This line won't appear in the legend

plt.legend()  # Includes only labeled elements

plt.show()
```

Output



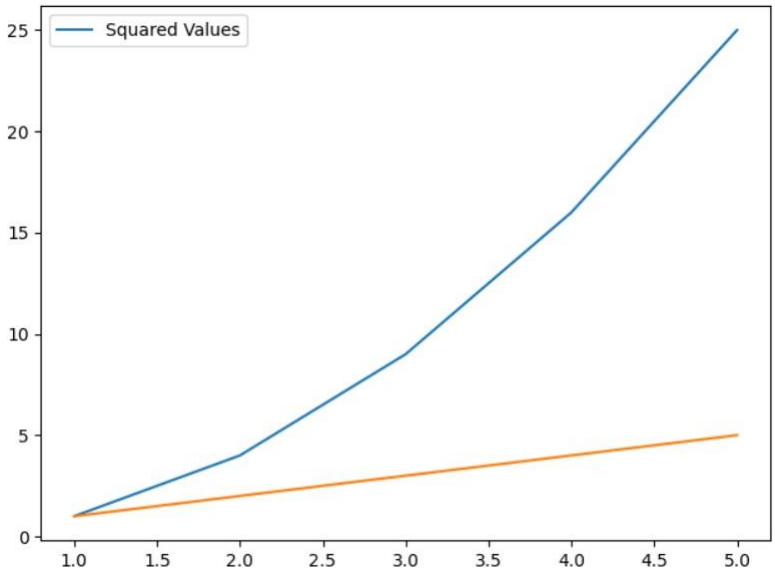
2. Include Only Specific Elements Using handles

You can explicitly define which elements should appear in the legend by using handles and labels.

➢ A python program on Include Only Specific Elements Using handles

```
import matplotlib.pyplot as plt

line1, = plt.plot(x, y1, label='Squared Values')

line2, = plt.plot(x, y2, label='Linear Values')

plt.legend(handles=[line1])  # Include only line1 in the legend

plt.show()
```

Output



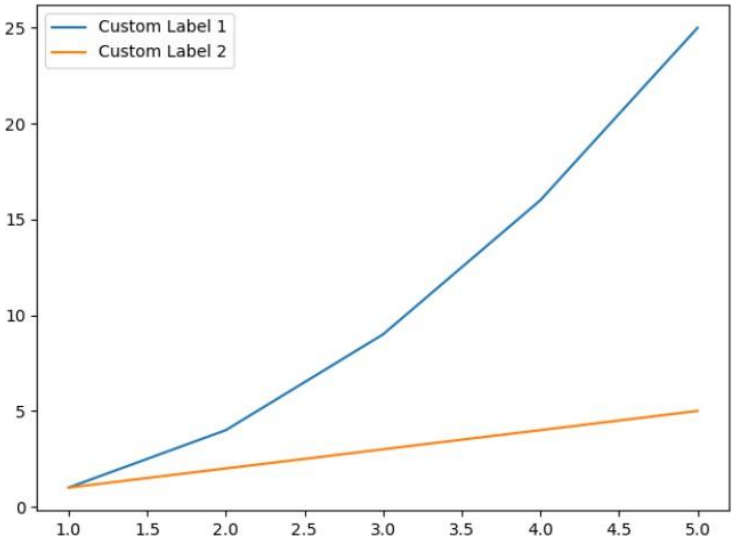
3. Customizing Legend Labels

You can use plt.legend() with custom labels for clarity or to rename legend items.

➢ A python program on Customizing Legend Labels

```
plt.plot(x, y1, label='Original')

plt.plot(x, y2, label='Another Line')

plt.legend(labels=['Custom Label 1', 'Custom Label 2'])

plt.show()
```

Output



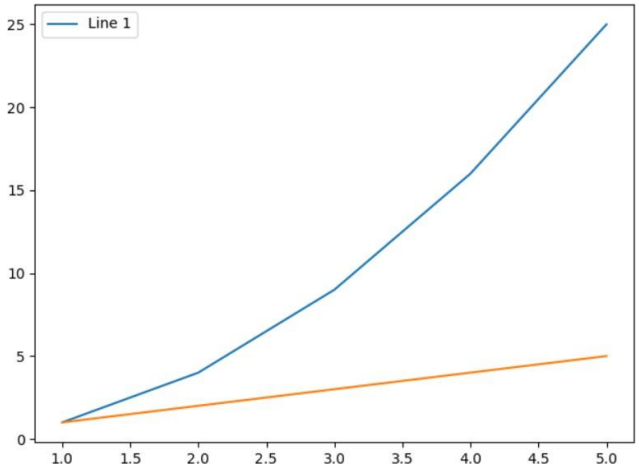
4. Using get_legend_handles_labels()

Retrieve all plot elements and their labels and customize the legend.

➢ A python program on Using get_legend_handles_labels()

```
fig, ax = plt.subplots()
line1, = ax.plot(x, y1, label='Line 1')
line2, = ax.plot(x, y2, label='Line 2')
handles, labels = ax.get_legend_handles_labels()
plt.legend(handles=[handles[0]], labels=[labels[0]])
plt.show()
```

Output



➢ Boxplot

Creating a boxplot in Python is straightforward using libraries like Matplotlib and Seaborn. A boxplot (or box-and-whisker plot) is used to display the distribution of data based on a five-number summary: minimum, first quartile (Q1), median, third quartile (Q3), and maximum.

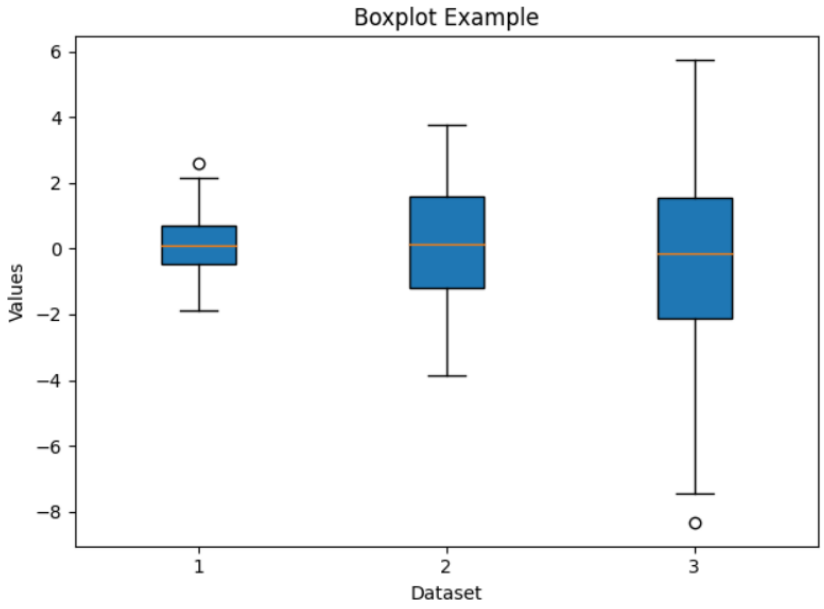
Here's a detailed guide on how to create and customize boxplots:

1. Creating a Basic Boxplot with Matplotlib

Program

```
import matplotlib.pyplot as plt

import numpy as np

data = [np.random.normal(0, std, 100) for std in range(1, 4)]plt.boxplot(data, patch_artist=True) # Use patch_artist=True for colored boxes

plt.title("Boxplot Example")

plt.xlabel("Dataset")

plt.ylabel("Values")

plt.show()
```

Output



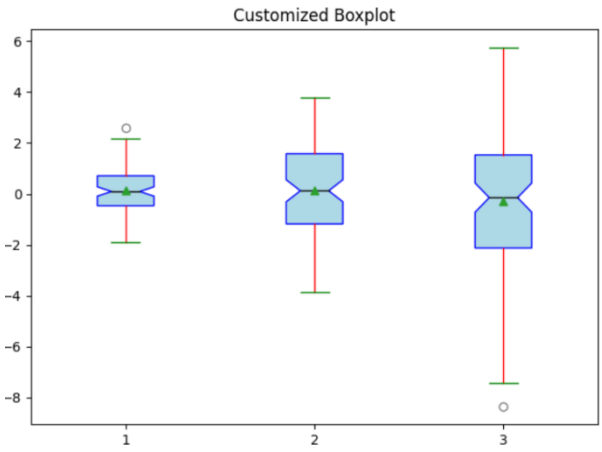
2. Adding Customizations to a Matplotlib Boxplot

You can customize the appearance of a boxplot using the parameters of plt.boxplot():

Program

```
plt.boxplot(
    data,
    patch_artist=True,
    notch=True,
    vert=True,
    showmeans=True,
    boxprops=dict(facecolor="lightblue", color="blue"),
    whiskerprops=dict(color="red"),
    capprops=dict(color="green"),
    flierprops=dict(marker="o", color="orange", alpha=0.5),
    medianprops=dict(color="black")
)
plt.title("Customized Boxplot")
plt.show()
```

Output



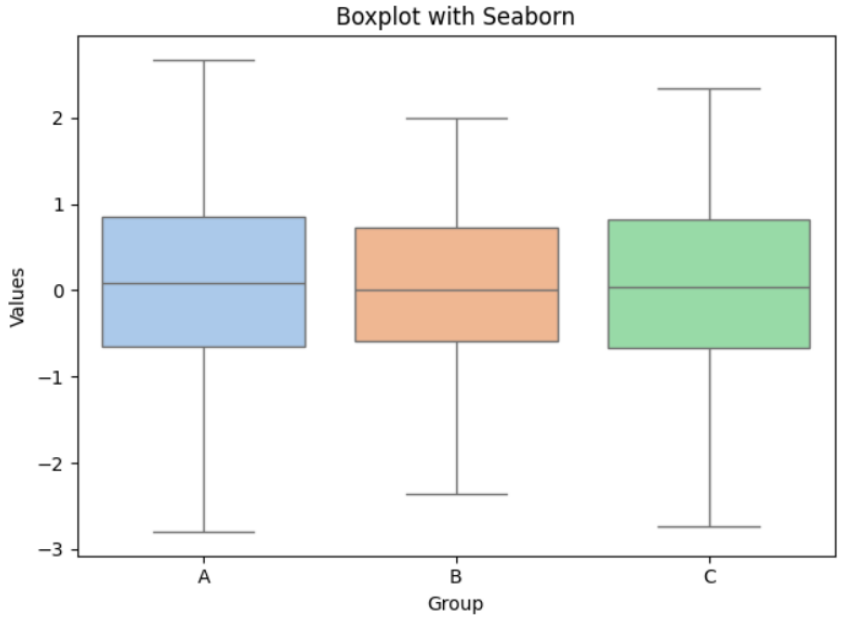
3. Creating a Boxplot with Seaborn

Seaborn provides a simpler and more aesthetic way to create boxplots:

Program

```
import seaborn as sns
import pandas as pd
df = pd.DataFrame({
    "Group": np.repeat(["A", "B", "C"], 100),
    "Values": np.random.randn(300)
})
sns.boxplot(x="Group", y="Values", data=df, palette="pastel")
plt.title("Boxplot with Seaborn")
plt.show()
```

Output



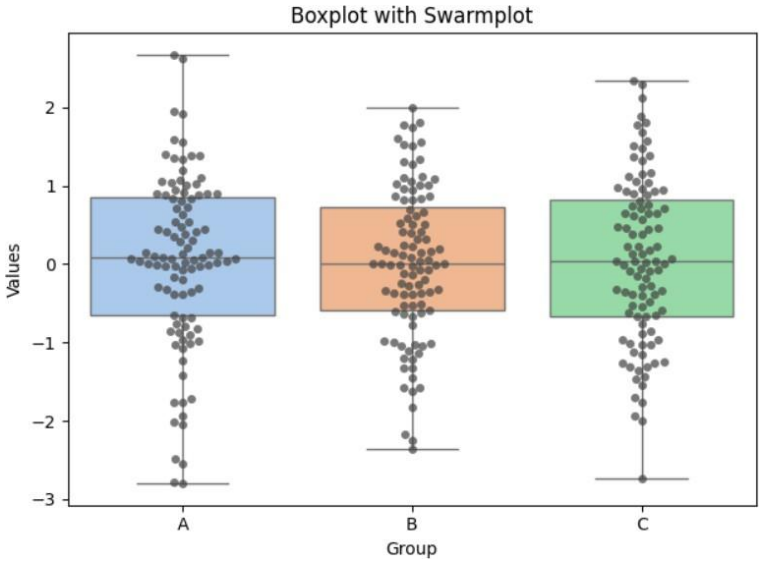
4. Adding Swarmplot or Stripplot for Individual Points

To visualize individual data points along with the boxplot:

Program

```
sns.boxplot(x="Group", y="Values", data=df, palette="pastel")
sns.swarmplot(x="Group", y="Values", data=df, color=".25", alpha=0.7)
plt.title("Boxplot with Swarmplot")
plt.show()
```

Output



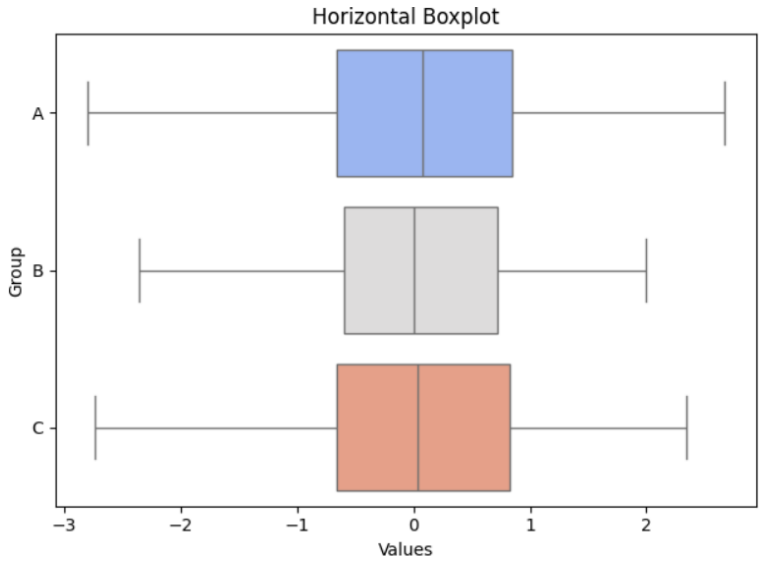
5. Horizontal Boxplot

To display a horizontal boxplot, use the orient parameter in Seaborn or set vert=False in Matplotlib.

Program

```
sns.boxplot(x="Values", y="Group", data=df, palette="coolwarm", orient="h")
plt.title("Horizontal Boxplot")
plt.show()
```

Output



➢ Multiple Legends

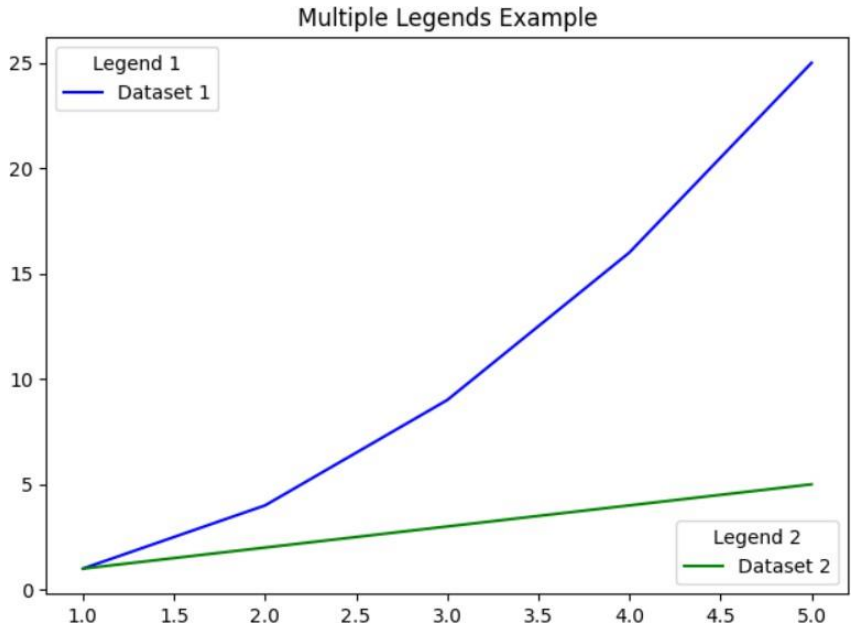
Creating multiple legends in a single plot in Python can be achieved using **Matplotlib**. By default, Matplotlib allows only one legend, but you can use ax.add_artist() to add additional legends. Here's how to create multiple legends in different scenarios:

1. Adding Multiple Legends Manually

Program

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y1 = [1, 4, 9, 16, 25]
y2 = [1, 2, 3, 4, 5]
fig, ax = plt.subplots()
line1, = ax.plot(x, y1, label='Dataset 1', color='blue')
line2, = ax.plot(x, y2, label='Dataset 2', color='green')
legend1 = ax.legend(handles=[line1], loc='upper left', title="Legend 1")
ax.add_artist(legend1)  # Add the first legend manually
ax.legend(handles=[line2], loc='lower right', title="Legend 2")
plt.title("Multiple Legends Example")
plt.show()
```

Output



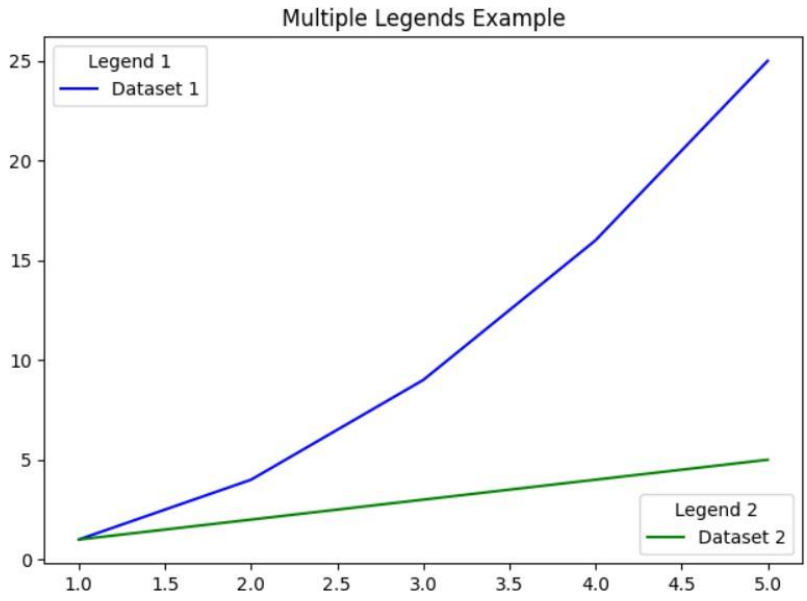
2. Multiple Legends for Different Data Groups

You can group your data visually and use different legends for each group:

Program

```
y3 = [25, 20, 15, 10, 5]
line3, = ax.plot(x, y3, label='Dataset 3', color='red')
legend1 = ax.legend(handles=[line1, line2], loc='upper center', title="Group 1")
ax.add_artist(legend1)
legend2 = ax.legend(handles=[line3], loc='lower center', title="Group 2")
ax.add_artist(legend2)
plt.show()
```

Output



3. Multiple Legends for Different Axes (Using twinx)

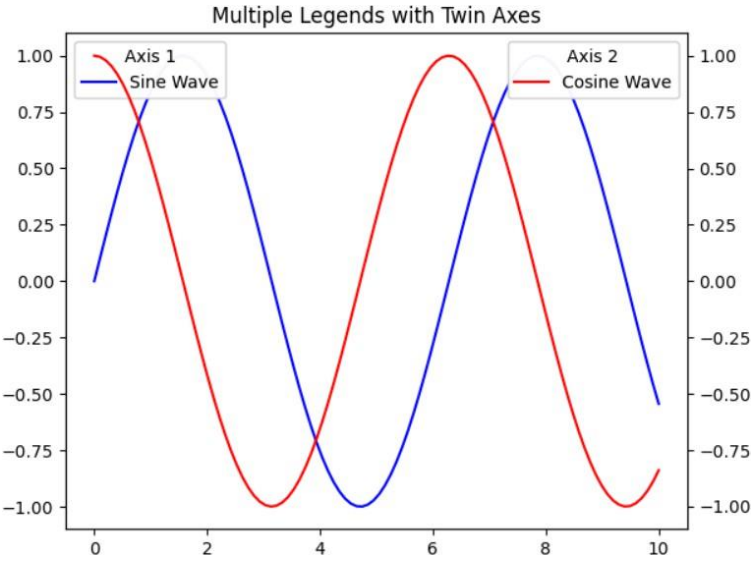
For plots with dual y-axes (twinx), you can add separate legends for each axis:

Program

```
import numpy as np
fig, ax1 = plt.subplots()
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
line1, = ax1.plot(x, y1, label="Sine Wave", color='blue')
ax2 = ax1.twinx()
y2 = np.cos(x)
line2, = ax2.plot(x, y2, label="Cosine Wave", color='red')
legend1 = ax1.legend(handles=[line1], loc='upper left', title="Axis 1")
```

```
ax1.add_artist(legend1)
legend2 = ax2.legend(handles=[line2], loc='upper right', title="Axis 2")
ax2.add_artist(legend2)
plt.title("Multiple Legends with Twin Axes")
plt.show()
```

Output

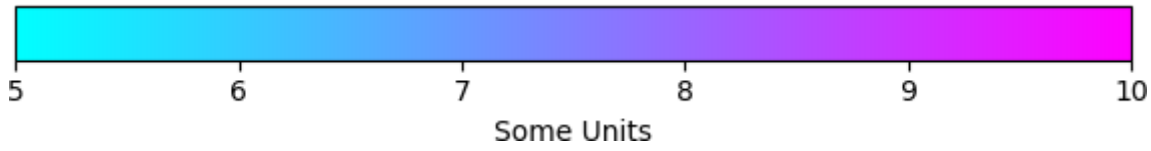


➢ Customized Colorbars

➢ Basic continuous colorbar

```
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np  # Importing numpy to create some dummy data
fig, ax = plt.subplots(figsize=(6, 1), layout='constrained')
cmap = mpl.cm.cool
norm = mpl.colors.Normalize(vmin=5, vmax=10)
dummy_data = np.array([[5, 10]])
sm = mpl.cm.ScalarMappable(cmap=cmap, norm=norm)
sm.set_array(dummy_data)
fig.colorbar(sm, cax=ax, orientation='horizontal', label='Some Units')
plt.show()
```

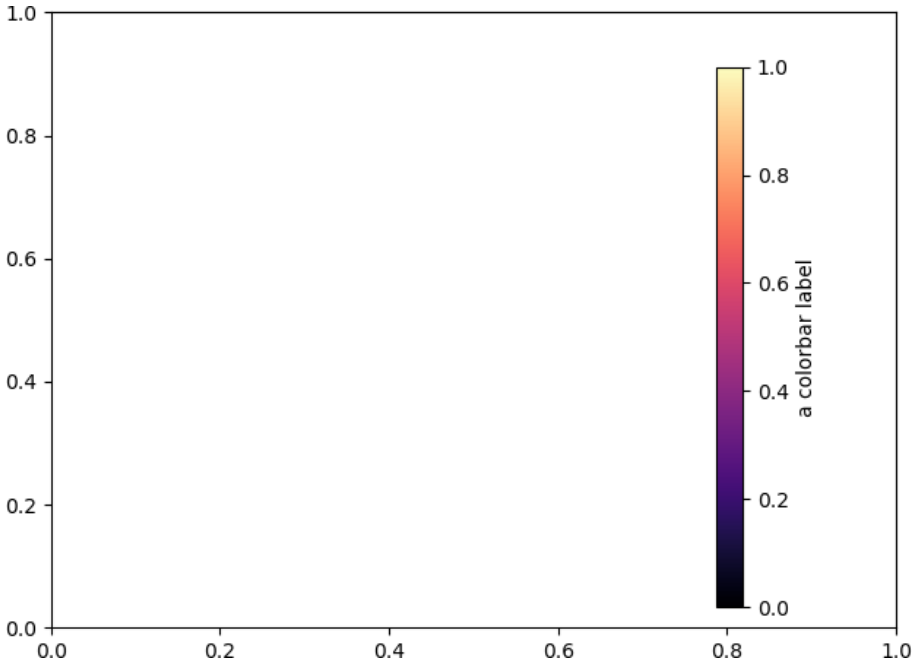
Output:



➢ Colorbar attached next to a pre-existing axes

```
import matplotlib.pyplot as plt
import matplotlib as mpl
fig, ax = plt.subplots(constrained_layout=True)
sm = mpl.cm.ScalarMappable(norm=mpl.colors.Normalize(0, 1), cmap='magma')
fig.colorbar(sm, ax=ax, orientation='vertical', label='a colorbar label')
plt.show()
```

Output:

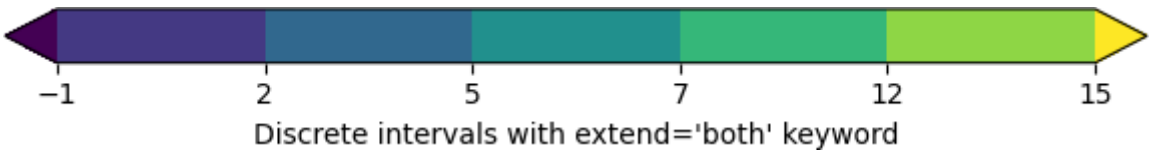


➢ Discrete and extended colorbar with continuous colorscale

```
import matplotlib.pyplot as plt
import matplotlib as mpl
fig, ax = plt.subplots(figsize=(6, 1), constrained_layout=True)
```

```
cmap = mpl.cm.viridis
bounds = [-1, 2, 5, 7, 12, 15]
norm = mpl.colors.BoundaryNorm(bounds, cmap.N, extend='both')
fig.colorbar(mpl.cm.ScalarMappable(norm=norm, cmap=cmap),
cax=ax, orientation='horizontal',
label="Discrete intervals with extend='both' keyword")
plt.show()
```

Output:



➢ Colorbar with arbitrary colors

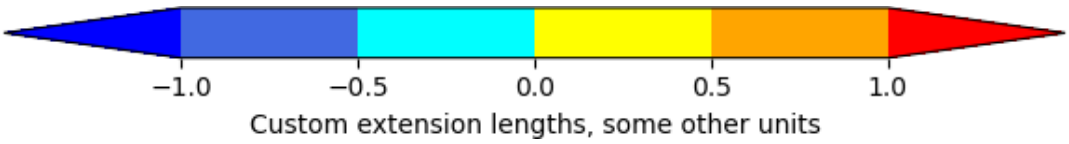
```
import matplotlib.pyplot as plt
import matplotlib as mpl
fig, ax = plt.subplots(figsize=(6, 1), constrained_layout=True)
cmap = (mpl.colors.ListedColormap(['red', 'green', 'blue', 'cyan'])
            .with_extremes(under='yellow', over='magenta'))
bounds = [1, 2, 4, 7, 8]
norm = mpl.colors.BoundaryNorm(bounds, cmap.N)
fig.colorbar(mpl.cm.ScalarMappable(cmap=cmap, norm=norm),
        cax=ax, orientation='horizontal',
        extend='both',
        spacing='proportional',
        label='Discrete intervals, some other units',
    )
plt.show()
```

Output:

➢ Colorbar with custom extension lengths

```
import matplotlib.pyplot as plt
import matplotlib as mpl
fig, ax = plt.subplots(figsize=(6, 1), constrained_layout=True)
cmap = (mpl.colors.ListedColormap(['royalblue', 'cyan', 'yellow', 'orange'])
            .with_extremes(over='red', under='blue'))
bounds = [-1.0, -0.5, 0.0, 0.5, 1.0]
norm = mpl.colors.BoundaryNorm(bounds, cmap.N)
fig.colorbar(
        mpl.cm.ScalarMappable(cmap=cmap, norm=norm),
        cax=ax, orientation='horizontal',
        extend='both',
        extendfrac='auto',
        spacing='uniform',
        label='Custom extension lengths, some other units',
    )
    plt.show()
```

Output:

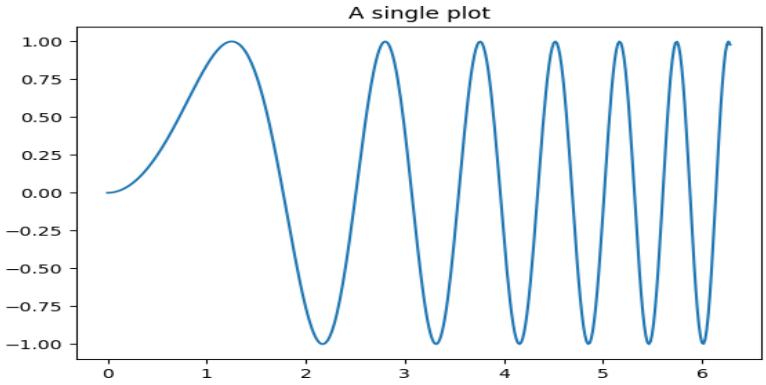


➢ Multiple Subplots

➢ Creating multiple subplots using plt.subplots

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 2 * np.pi, 400)
y = np.sin(x ** 2)
fig, ax = plt.subplots()
x.plot(x, y)
ax.set_title('A single plot')
```

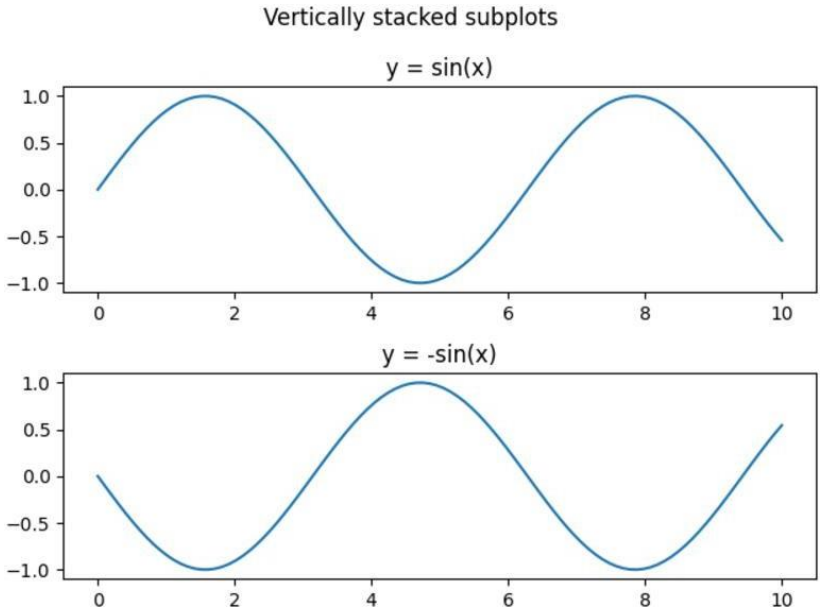
Output:



➢ Stacking subplots in one direction

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
y = np.sin(x)
fig, axs = plt.subplots(2)
fig.suptitle('Vertically stacked subplots')
axs[0].plot(x, y)
axs[0].set_title('y = sin(x)')
axs[1].plot(x, -y)
axs[1].set_title('y = -sin(x)')
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

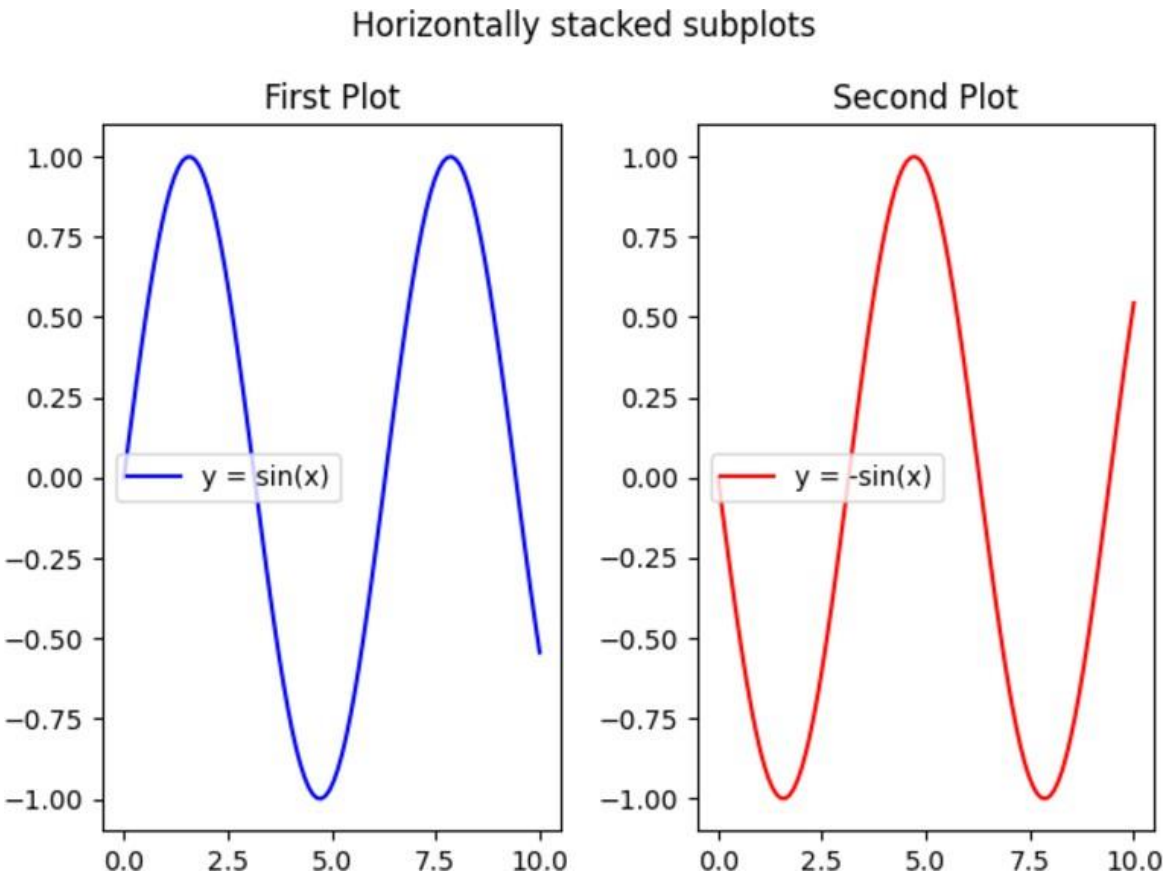
Output:



➢ Horizontally Stacked Subplots

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
y = np.sin(x)
fig, (ax1, ax2) = plt.subplots(1, 2)
fig.suptitle('Horizontally stacked subplots')
ax1.plot(x, y, color='blue', label='y = sin(x)')
ax1.set_title('First Plot')
ax1.legend()
ax2.plot(x, -y, color='red', label='y = -sin(x)')
ax2.set_title('Second Plot')
ax2.legend()
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

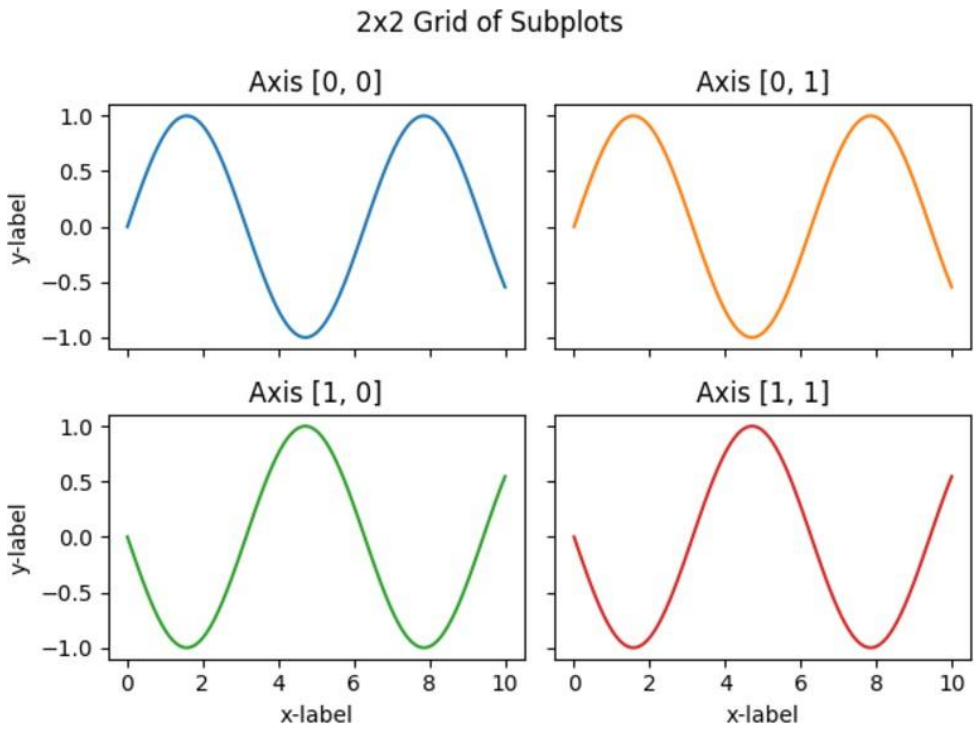
Output:




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

```
x = np.linspace(0, 10, 100)
y = np.sin(x)
fig, axs = plt.subplots(2, 2)
fig.suptitle('2x2 Grid of Subplots')
axs[0, 0].plot(x, y)
axs[0, 0].set_title('Axis [0, 0]')
axs[0, 1].plot(x, y, 'tab:orange')
axs[0, 1].set_title('Axis [0, 1]')
axs[1, 0].plot(x, -y, 'tab:green')
axs[1, 0].set_title('Axis [1, 0]')
axs[1, 1].plot(x, -y, 'tab:red')
axs[1, 1].set_title('Axis [1, 1]')
for ax in axs.flat:
 ax.set(xlabel='x-label', ylabel='y-label')
for ax in axs.flat:
ax.label_outer()
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

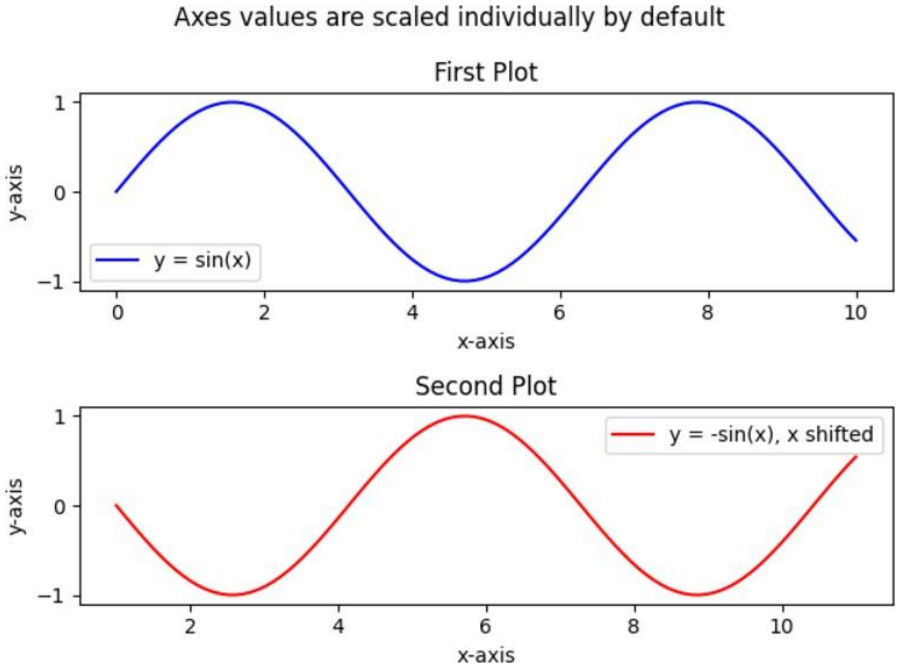
Output:



➢ Sharing axes

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
y = np.sin(x)
fig, (ax1, ax2) = plt.subplots(2)
fig.suptitle('Axes values are scaled individually by default')
ax1.plot(x, y, label='y = sin(x)', color='blue')
ax1.legend()
ax1.set_title('First Plot')
ax2.plot(x + 1, -y, label='y = -sin(x), x shifted', color='red')
ax2.legend()
ax2.set_title('Second Plot')
ax1.set_xlabel('x-axis')
ax1.set_ylabel('y-axis')
ax2.set_xlabel('x-axis')
ax2.set_ylabel('y-axis')
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

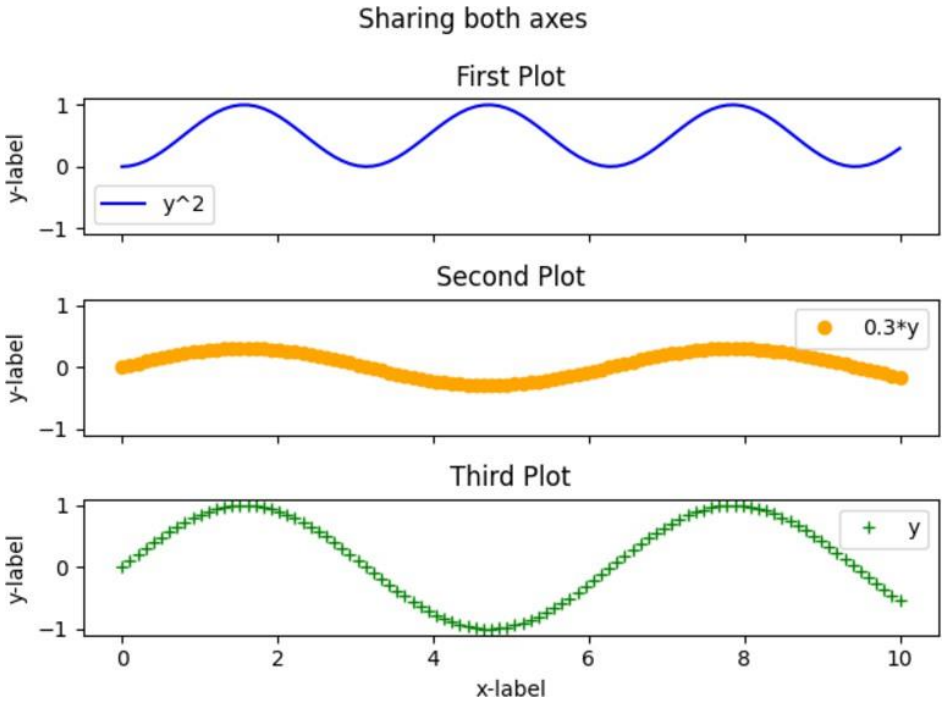
Output:



➢ Sharing both axes

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
y = np.sin(x)
fig, axs = plt.subplots(3, sharex=True, sharey=True)
fig.suptitle('Sharing both axes')
axs[0].plot(x, y ** 2, label='y^2', color='blue')
axs[0].legend()
axs[0].set_title('First Plot')
axs[1].plot(x, 0.3 * y, 'o', label='0.3*y', color='orange')
axs[1].legend()
axs[1].set_title('Second Plot')
axs[2].plot(x, y, '+', label='y', color='green')
axs[2].legend()
axs[2].set_title('Third Plot')
for ax in axs:
ax.set_ylabel('y-label')
axs[-1].set_xlabel('x-label')
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()
```

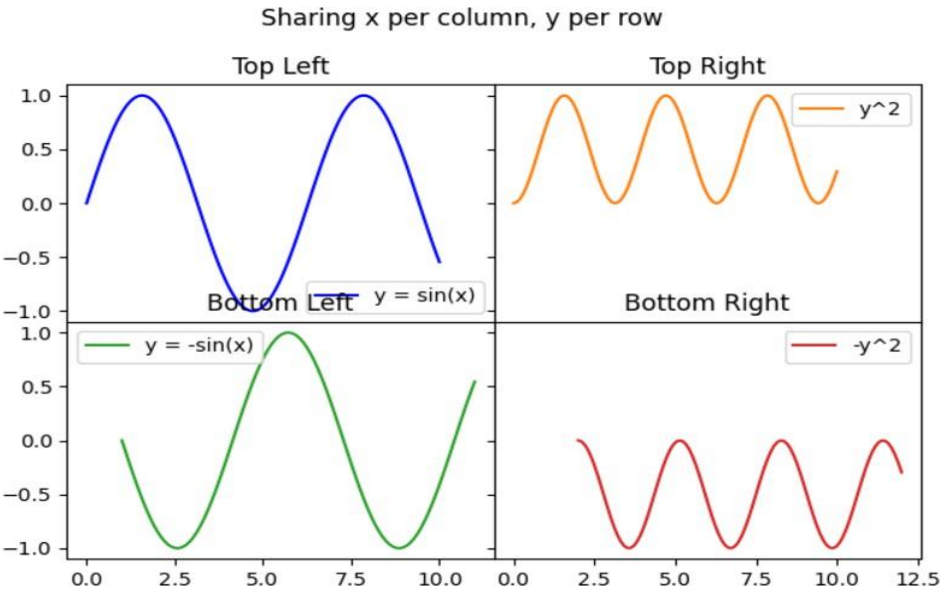
Output:



➢ Sharing x per column, y per row

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 10, 100)
y = np.sin(x)
fig = plt.figure()
gs = fig.add_gridspec(2, 2, hspace=0, wspace=0)
(ax1, ax2), (ax3, ax4) = gs.subplots(sharex='col', sharey='row')
fig.suptitle('Sharing x per column, y per row')
ax1.plot(x, y, label='y = sin(x)', color='blue')
ax1.legend()
ax1.set_title('Top Left')
ax2.plot(x, y**2, 'tab:orange', label='y^2')
ax2.legend()
ax2.set_title('Top Right')
ax3.plot(x + 1, -y, 'tab:green', label='y = -sin(x)')
ax3.legend()
ax3.set_title('Bottom Left')
ax4.plot(x + 2, -y**2, 'tab:red', label='-y^2')
ax4.legend()
ax4.set_title('Bottom Right')
for ax in fig.get_axes():
ax.label_outer()
plt.show()
```

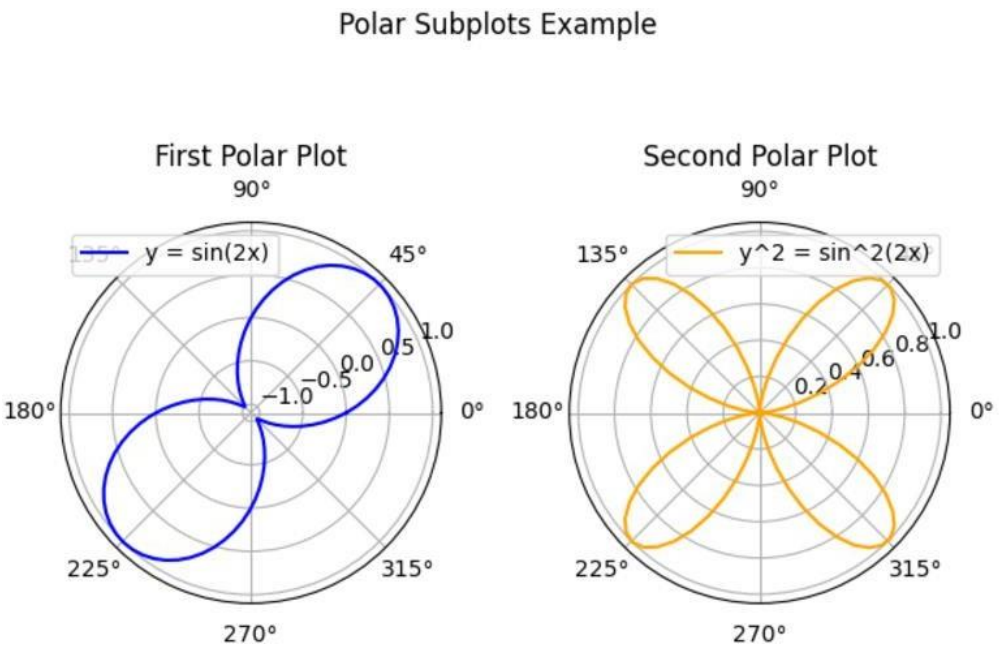
Output:



➢ Polar Axes

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(2 * x)
fig, (ax1, ax2) = plt.subplots(1, 2, subplot_kw=dict(projection='polar'))
fig.suptitle('Polar Subplots Example')
ax1.plot(x, y, label='y = sin(2x)', color='blue')
ax1.set_title('First Polar Plot')
ax1.legend()
ax2.plot(x, y ** 2, label='y^2 = sin^2(2x)', color='orange')
ax2.set_title('Second Polar Plot')
ax2.legend().
```

Output:



➢ Customizing Ticks

➢ Axis ticks:

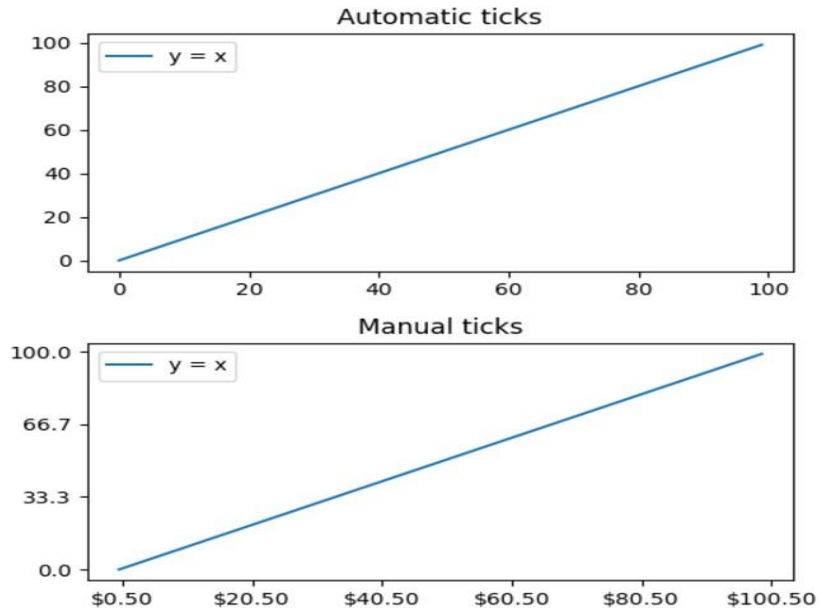
Axis ticks refer to the markers on the axes that indicate specific data values. These ticks, along with their labels, provide context for the plot's scale.

➢ Manual location and formats

```
import numpy as np
```

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
fig, axs = plt.subplots(2, 1, figsize=(5.4, 5.4), layout='constrained')
x = np.arange(100)
for nn, ax in enumerate(axs):
ax.plot(x, x, label='y = x')
ax.legend()
if nn == 1:
ax.set_title('Manual ticks')
ax.set_yticks(np.arange(0, 100.1, 100/3))
xticks = np.arange(0.50, 101, 20)
xlabels = [f'\\${x:1.2f}' for x in xticks]  ax.set_xticks(xticks, labels=xlabels)
else:
ax.set_title('Automatic ticks')
plt.show()
```

Output:

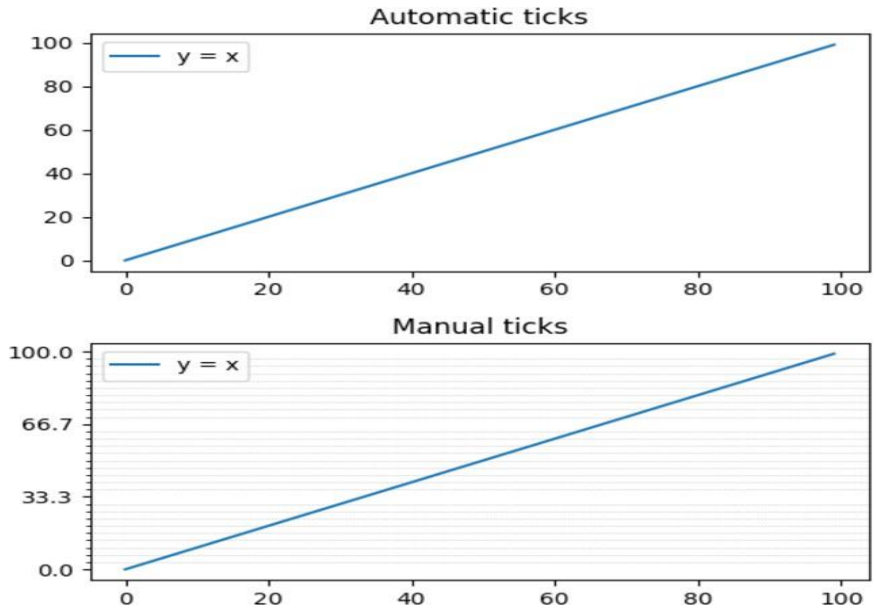


➢ Customizing Axis Ticks in Subplots

```
import numpy as np
import matplotlib.pyplot as plt
fig, axs = plt.subplots(2, 1, figsize=(5.4, 5.4), layout='constrained')
x = np.arange(100)
for nn, ax in enumerate(axs):
    ax.plot(x, x, label='y = x')
    ax.legend()
    if nn == 1:
```

```
        ax.set_title('Manual ticks')
        ax.set_yticks(np.arange(0, 100.1, 100/3))
        ax.set_yticks(np.arange(0, 100.1, 100/30), minor=True)
        ax.grid(True, which='minor', linestyle=':', linewidth=0.5)
    else:
        ax.set_title('Automatic ticks')
plt.show()
```

Output:



➢ Locators and Formatters

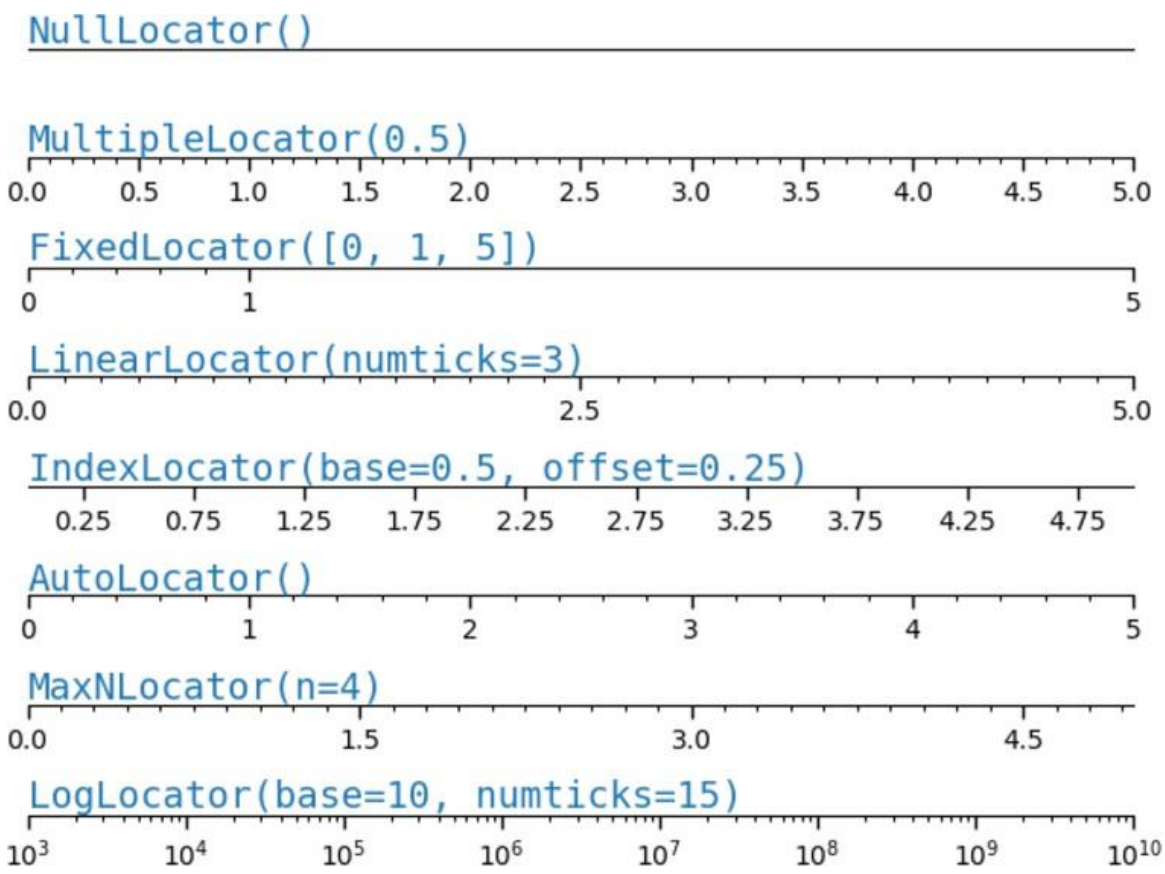
Locators and **formatters** are used to control the placement and formatting of ticks on the axes. They allow for fine control over how ticks and their labels appear on the plot.

➢ Locators Demonstration

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
def setup(ax, title):
        """Set up common parameters for the Axes in the example."""
         ax.yaxis.set_major_locator(ticker.NullLocator())
         ax.spines[['left', 'right', 'top']].set_visible(False)
         ax.xaxis.set_ticks_position('bottom')
         ax.tick_params(which='major', width=1.00, length=5)
         ax.tick_params(which='minor', width=0.75, length=2.5)
```

```
        ax.set_xlim(0, 5)
        ax.set_ylim(0, 1)
        ax.text(0.0, 0.2, title, transform=ax.transAxes,
                fontsize=14, fontname='Monospace', color='tab:blue')
fig, axs = plt.subplots(8, 1, layout='constrained')
# Null Locator
setup(axs[0], title="NullLocator()")
axs[0].xaxis.set_major_locator(ticker.NullLocator())
axs[0].xaxis.set_minor_locator(ticker.NullLocator())
# Multiple Locator
setup(axs[1], title="MultipleLocator(0.5)")
axs[1].xaxis.set_major_locator(ticker.MultipleLocator(0.5))
axs[1].xaxis.set_minor_locator(ticker.MultipleLocator(0.1))
# Fixed Locator
setup(axs[2], title="FixedLocator([0, 1, 5])")
axs[2].xaxis.set_major_locator(ticker.FixedLocator([0, 1, 5]))
axs[2].xaxis.set_minor_locator(ticker.FixedLocator(np.linspace(0.2, 0.8, 4)))
# Linear Locator
setup(axs[3], title="LinearLocator(numticks=3)")
axs[3].xaxis.set_major_locator(ticker.LinearLocator(3))
axs[3].xaxis.set_minor_locator(ticker.LinearLocator(31))
# Index Locator
setup(axs[4], title="IndexLocator(base=0.5, offset=0.25)")
axs[4].plot(range(0, 5), [0]*5, color='white')
axs[4].xaxis.set_major_locator(ticker.IndexLocator(base=0.5, offset=0.25))
# Auto Locator
setup(axs[5], title="AutoLocator()")
axs[5].xaxis.set_major_locator(ticker.AutoLocator())
axs[5].xaxis.set_minor_locator(ticker.AutoMinorLocator())
# MaxN Locator
setup(axs[6], title="MaxNLocator(n=4)")
axs[6].xaxis.set_major_locator(ticker.MaxNLocator(4))
axs[6].xaxis.set_minor_locator(ticker.MaxNLocator(40))
# Log Locator
setup(axs[7], title="LogLocator(base=10, numticks=15)")
axs[7].set_xlim(10**3, 10**10)
axs[7].set_xscale('log')
axs[7].xaxis.set_major_locator(ticker.LogLocator(base=10, numticks=15))
plt.show()
```

Output:

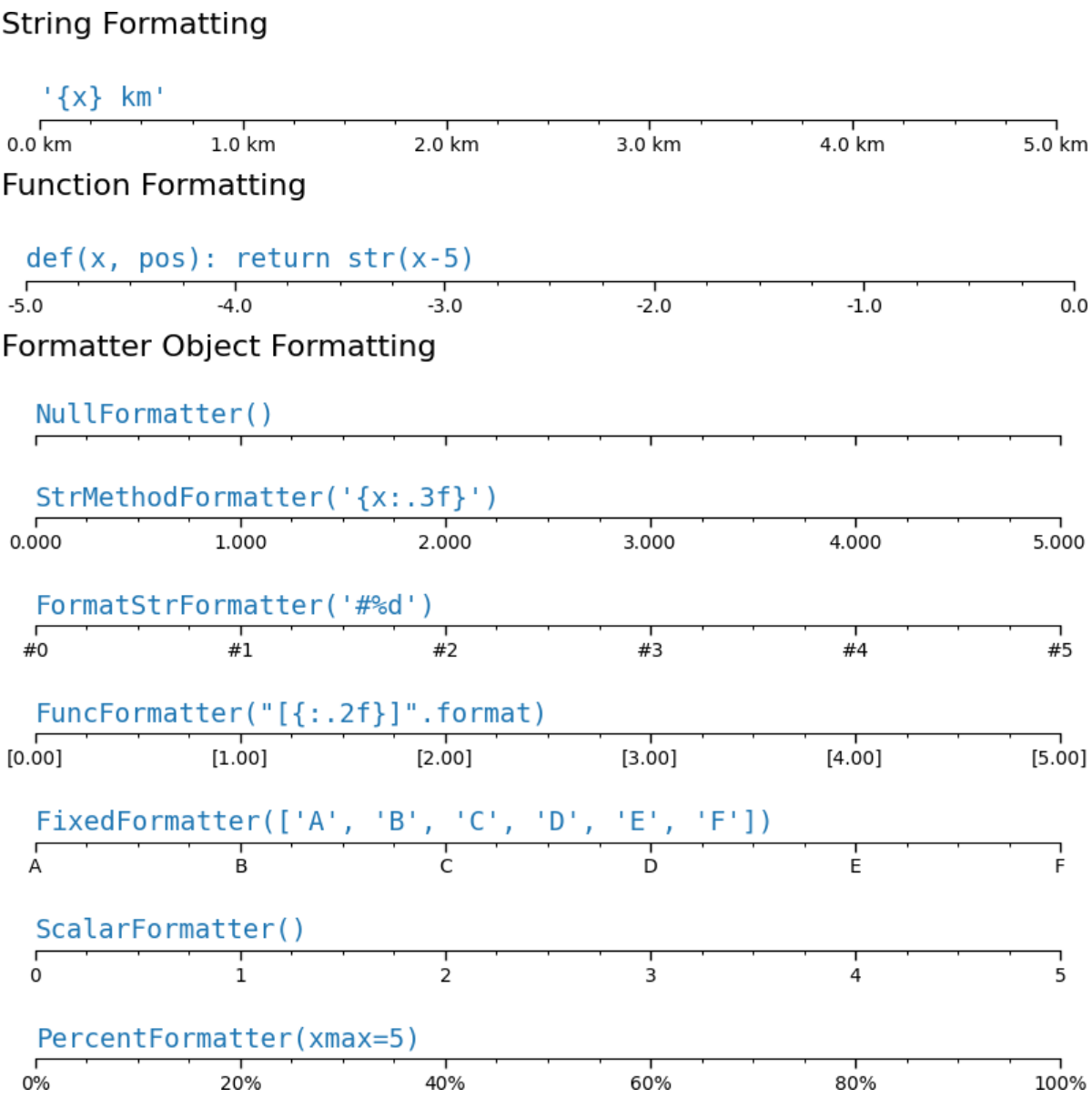


- **Tick Formatting**

```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
def setup(ax, title):
    """Set up common parameters for the Axes in the example."""
    ax.yaxis.set_major_locator(ticker.NullLocator())
    ax.spines[['left', 'right', 'top']].set_visible(False)
    ax.xaxis.set_major_locator(ticker.MultipleLocator(1.00))
    ax.xaxis.set_minor_locator(ticker.MultipleLocator(0.25))
    ax.xaxis.set_ticks_position('bottom')
    ax.tick_params(which='major', width=1.00, length=5)
    ax.tick_params(which='minor', width=0.75, length=2.5, labelsize=10)
    ax.set_xlim(0, 5)
    ax.set_ylim(0, 1)
    ax.text(0.0, 0.2, title, transform=ax.transAxes,
            fontsize=14, fontname='Monospace', color='tab:blue')
fig = plt.figure(figsize=(8, 8), constrained_layout=True)
fig0, fig1, fig2 = fig.subfigures(3, height_ratios=[1.5, 1.5, 7.5])
```

```
fig0.suptitle('String Formatting', fontsize=16, x=0, ha='left')
ax0 = fig0.subplots()
setup(ax0, title="'{x} km'")
ax0.xaxis.set_major_formatter(ticker.StrMethodFormatter("{x} km"))
fig1.suptitle('Function Formatting', fontsize=16, x=0, ha='left')
ax1 = fig1.subplots()
setup(ax1, title="def(x, pos): return str(x-5)")
ax1.xaxis.set_major_formatter(lambda x, pos: str(x-5))
fig2.suptitle('Formatter Object Formatting', fontsize=16, x=0, ha='left')
axs2 = fig2.subplots(7, 1)
setup(axs2[0], title="NullFormatter()")
axs2[0].xaxis.set_major_formatter(ticker.NullFormatter())
setup(axs2[1], title="StrMethodFormatter('{x:.3f}')")
axs2[1].xaxis.set_major_formatter(ticker.StrMethodFormatter("{x:.3f}"))
setup(axs2[2], title="FormatStrFormatter('#%d')")
axs2[2].xaxis.set_major_formatter(ticker.FormatStrFormatter("#%d"))
def fmt_two_digits(x, pos):
        return f'[{x:.2f}]'
setup(axs2[3], title='FuncFormatter("[{:.2f}]".format)')
axs2[3].xaxis.set_major_formatter(ticker.FuncFormatter(fmt_two_digits))
setup(axs2[4], title="FixedFormatter(['A', 'B', 'C', 'D', 'E', 'F'])")
positions = [0, 1, 2, 3, 4, 5]
labels = ['A', 'B', 'C', 'D', 'E', 'F']
axs2[4].xaxis.set_major_locator(ticker.FixedLocator(positions))
axs2[4].xaxis.set_major_formatter(ticker.FixedFormatter(labels))
setup(axs2[5], title="ScalarFormatter()")
axs2[5].xaxis.set_major_formatter(ticker.ScalarFormatter(useMathText=True))
setup(axs2[6], title="PercentFormatter(xmax=5)")
axs2[6].xaxis.set_major_formatter(ticker.PercentFormatter(xmax=5))
plt.show()
```

Output:



➢ Styling ticks (tick parameters)

The appearance of ticks can be controlled at a low level by finding the individual Tick on the axis. However, usually it is simplest to use tick_params to change all the objects at once.

The tick_params method can change the properties of ticks:

- length
- direction (in or out of the frame)
- colors
- width and length

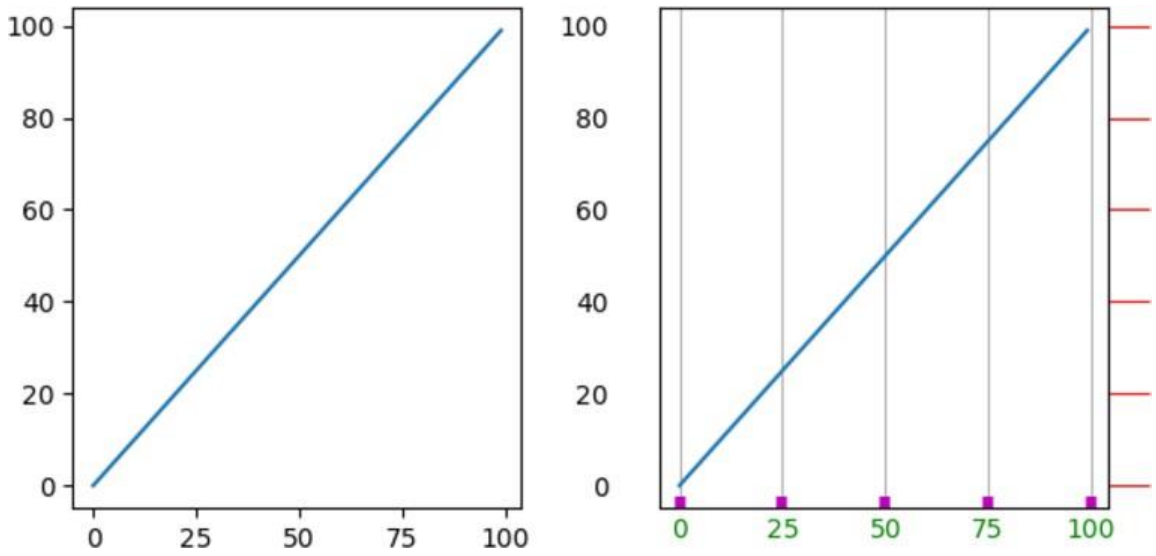
- whether the ticks are drawn at the bottom, top, left, or right of the Axes.

It also can control the tick labels:

- labelsize (fontsize)
- labelcolor (color of the label)
- labelrotation
- labelbottom, labeltop, labelleft, labelright

```
import matplotlib.pyplot as plt
import numpy as np
fig, axs = plt.subplots(1, 2, figsize=(6.4, 3.2))
for nn, ax in enumerate(axs):
        ax.plot(np.arange(100))
        if nn == 1:
           ax.grid(True)
           ax.tick_params(right=True, left=False, axis='y', color='r', length=16,
                    grid_color='none')
           ax.tick_params(axis='x', color='m', length=4, direction='in', width=4,
                    labelcolor='g')
plt.tight_layout()
plt.show()
```

Output:



➢ Text And Annotations

➢ Text in Matplotlib

Matplotlib has extensive text support, including support for mathematical expressions, truetype support for raster and vector outputs, newline separated text with arbitrary rotations, and Unicode support.

Basic text commands

implicit API	explicit API	description
`text`	`text`	Add text at an arbitrary location of the `Axes`.
`annotate`	`annotate`	Add an annotation, with an optional arrow, at an arbitrary location of the `Axes`.
`xlabel`	`set_xlabel`	Add a label to the `Axes`'s x-axis.
`ylabel`	`set_ylabel`	Add a label to the `Axes`'s y-axis.
`title`	`set_title`	Add a title to the `Axes`.
`figtext`	`text`	Add text at an arbitrary location of the `Figure`.
`suptitle`	`suptitle`	Add a title to the `Figure`.

10. Python Programs for Data preprocessing: Handling missing values, handling categorical data, bringing features to same scale, selecting meaningful features.

Importing the libraries:

libraries

import numpy as np # used for handling numbers

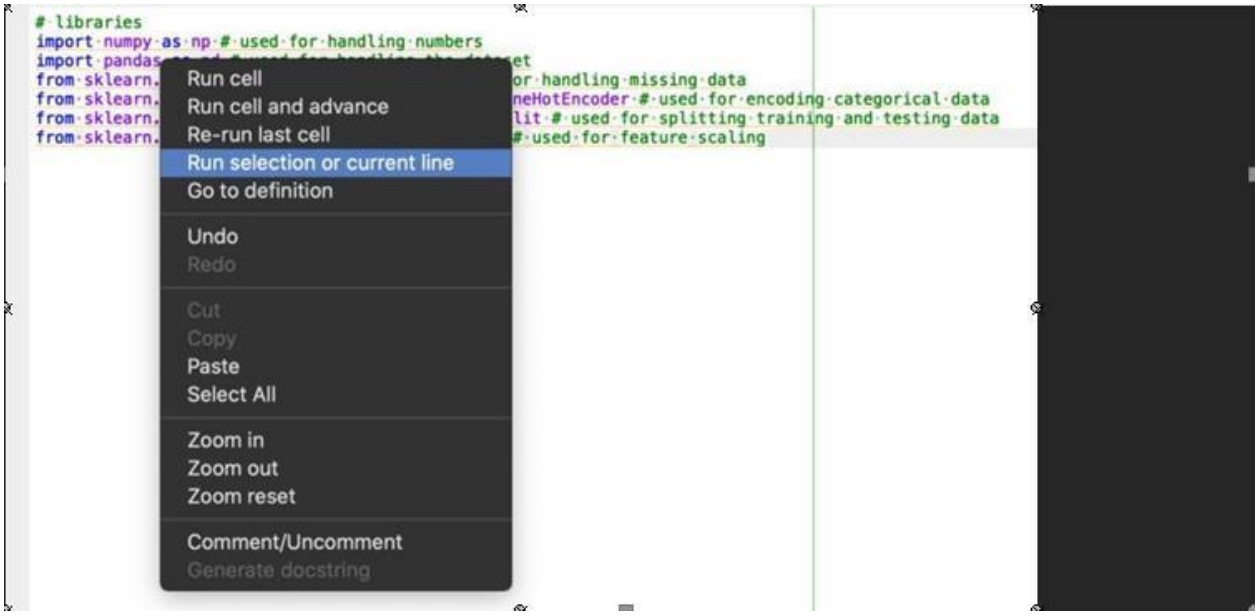
import pandas as pd # used for handling the dataset

from sklearn.impute import SimpleImputer # used for handling missing datafrom sklearn.preprocessing

import LabelEncoder, OneHotEncoder # used for encoding categorical datafrom sklearn.model_selection

import train_test_split # used for splitting training and testing datafrom sklearn.preprocessing

import StandardScaler # used for feature scaling



If you select and run the above code in Spyder, you should see a similar output in your IPython console.

import numpy as np # used for handling numbers

...: import pandas as pd # used for handling the dataset

...: from sklearn.impute import SimpleImputer # used for handling missing data

...: from sklearn.preprocessing import LabelEncoder, OneHotEncoder # used for encoding categorical data

...: from sklearn.model_selection import train_test_split # used for splitting training and testing data

...: from sklearn.preprocessing import StandardScaler # used for feature scaling

If you see any import errors, try to install those packages explicitly using pip command as follows.

pip install <package-name>

➢ Importing the Dataset

First of all, let us have a look at the dataset we are going to use for this particular example. You can find the https://github.com/tarunlnmiit/machine_learning/blob/master/DataPreprocessing.csv

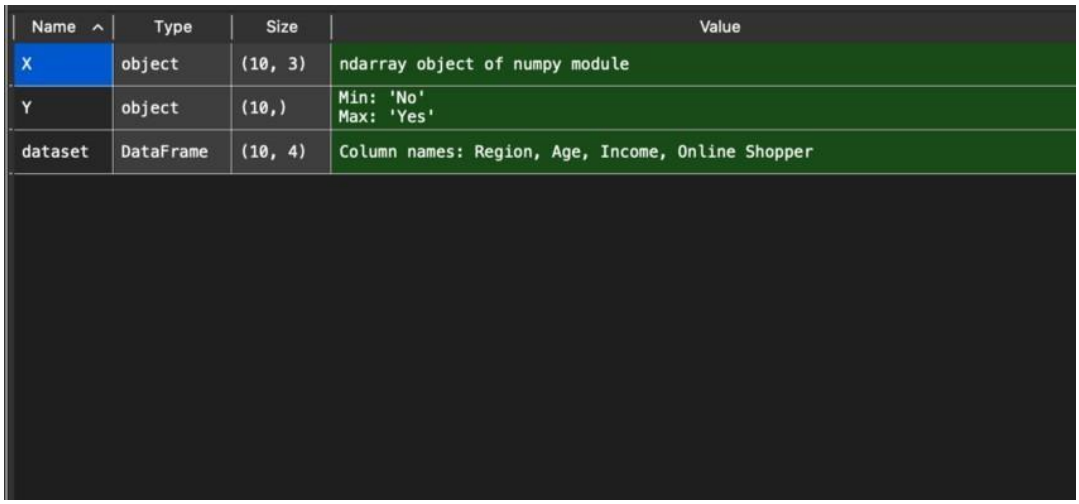
1	Region	Age	Income	Online Shopper
2	India	49	86400	No
3	Brazil	32	57600	Yes
4	USA	35	64800	No
5	Brazil	43	73200	No
6	USA	45		Yes
7	India	40	69600	Yes
8	Brazil		62400	No
9	India	53	94800	Yes
10	USA	55	99600	No
11	India	42	80400	Yes

In order to import this dataset into our script, we are apparently going to use pandas as follows.

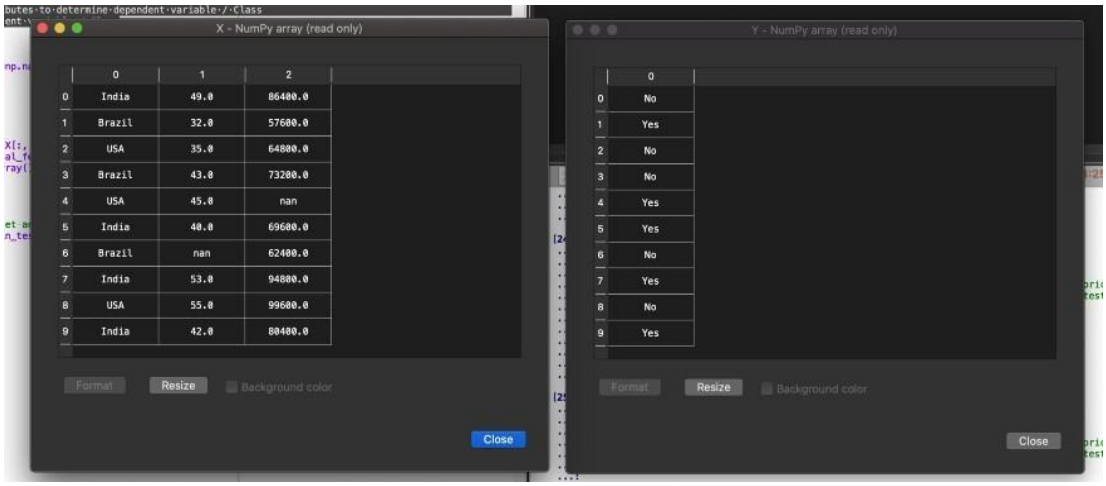
dataset = pd.read_csv('Data.csv') # to import the dataset into a variable# Splitting the attributes into independent and dependent attributes

X = dataset.iloc[:, :-1].values # attributes to determine dependent variable / Class Y = dataset.iloc[:, -1].values # dependent variable / Class

When you run this code section, you should not see any errors, if you do make sure the script and the *Data.csv* are in the same folder. When successfully executed, you can move to variable explorer in the Spyder UI and you will see the following three variables.



When you double click on each of these variables, you should see something similar.



If you face any errors in order to see these data variables, try to upgrade Spyder to Spyder version 4.

➢ Handling of Missing Data

Well the first idea is to remove the lines in the observations where there is some missing data. But that can be quite dangerous because imagine this data set contains crucial information. It would be quite dangerous to remove an observation. So we need to figure out a better idea to handle this problem. And another idea that's actually the most common idea to handle missing data is to take the mean of the columns.

If you noticed in our dataset, we have two values missing, one for age column in 7th data row and for Income column in 5th data row. Missing values should be handled during the data analysis. So, we do that as follows.

handling the missing data and replace missing values with nan from numpy and replace with mean of all the other values

imputer = SimpleImputer(missing_values=np.nan, strategy='mean') imputer = imputer.fit(X[:, 1:]) X[:, 1:] = imputer.transform(X[:, 1:])

After execution of this code, the independent variable *X* will transform into the following.

Here you can see, that the missing values have been replaced by the average values of the respective columns.

➢ Handling of Categorical Data

In this dataset we can see that we have two categorical variables. We have the Region variable and the Online Shopper variable. These two variables are categorical variables because simply they contain categories. The Region contains three categories. It's *India, USA & Brazil* and the online shopper variable contains two categories. *Yes* and *No* that's why they're called categorical variables.

You can guess that since machine learning models are based on mathematical equations you can intuitively understand that it would cause some problem if we keep the text here in the categorical variables in the equations because we would only want numbers in the equations. So that's why we need to encode the categorical variables. That is to encode the text that we have here into numbers. To do this we use the following code snippet.

```
# encode categorical data

from sklearn.preprocessing import LabelEncoder, OneHotEncoder labelencoder_X =
LabelEncoder()

X[:, 0] = labelencoder_X.fit_transform(X[:, 0]) onehotencoder =
OneHotEncoder(categorical_features=[0])
```

X = onehotencoder.fit_transform(X).toarray()labelencoder_Y = LabelEncoder() Y = labelencoder_Y.fit_transform(Y)

After execution of this code, the independent variable *X* and dependent variable *Y* will transform into the following.

Here, you can see that the Region variable is now made up of a 3 bit binary variable. The left most bit represents *India*, 2nd bit represents *Brazil* and the last bit represents *USA.* If the bit is 1 then it represents data for that country otherwise not. For *Online Shopper* variable, 1 represents Yes and 0 represents No.

➢ Splitting the dataset into training and testing datasets

Any machine learning algorithm needs to be tested for accuracy. In order to do that, we divide our data set into two parts: training set and testing set. As the name itself suggests, we use the training set to make the algorithm learn the behaviours present in the data and check the correctness of the algorithm by testing on testing set. In Python, we do that as follows:

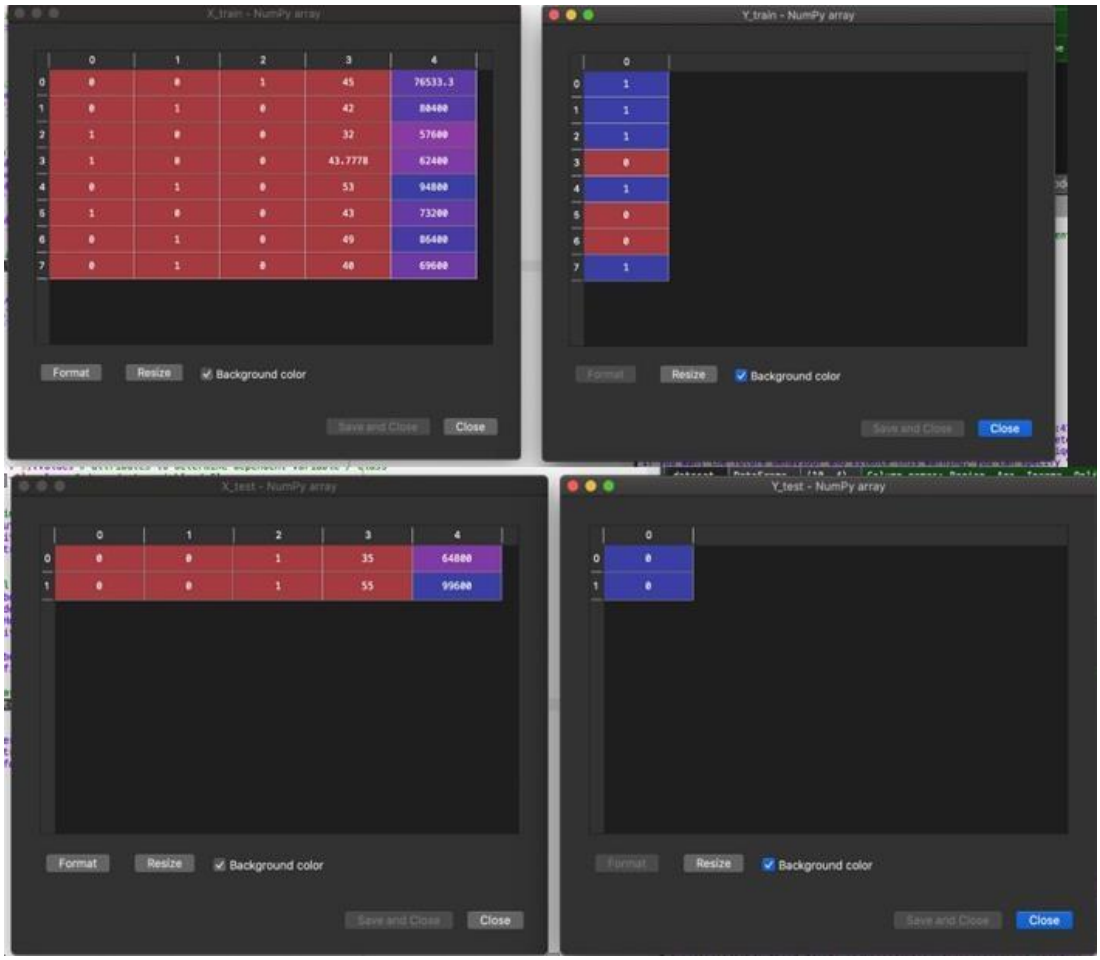
splitting the dataset into training set and test set

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)

Here, we are taking training set to be 80% of the original data set and testing set to be 20% of the original data set. This is usually the ratio in which they are split. But, you can come across sometimes to a 70–30% or 75–25% ratio split. But, you don't want to split it 50–50%. This can lead to *Model Overfitting.* This topic is too huge to be

covered in the same post. I will cover it in some future post. For now, we are going to split it in 80–20% ratio.

After split, our training set and testing set look like this.



➢ Feature Scaling

As you can see we have these two columns age and income that contains numerical numbers. You notice that the variables are not on the same scale because the age are going from 32 to 55 and the salaries going from 57.6 K to like 99.6 K.

So because this age variable in the salary variable don't have the same scale. This will cause some issues in your machinery models. And why is that. It's because your machine models a lot of machinery models are based on what is called the Euclidean distance.

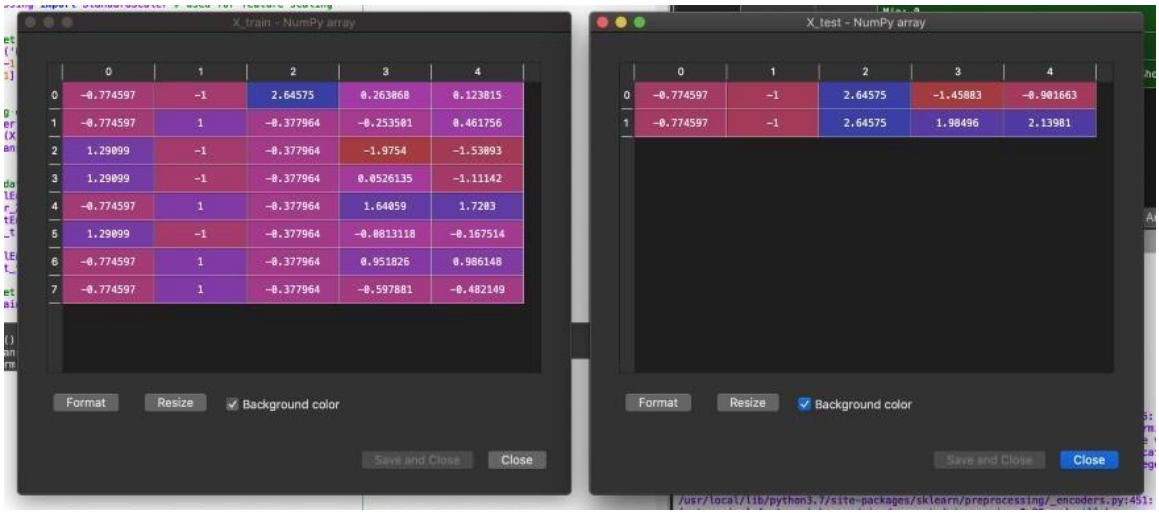
We use feature scaling to convert different scales to a standard scale to make it easier for Machine Learning algorithms. We do this in Python as follows:

feature scaling

sc_X = StandardScaler()

X_train = sc_X.fit_transform(X_train) X_test = sc_X.transform(X_test)

After the execution of this code, our training independent variable *X* and our testing independent variable *X* and look like this.



This data is now ready to be fed to a Machine Learning Algorithm.

This concludes this post on Data Preprocessing in Python.

11. Python program for compressing data via dimensionality reduction: PCA

Dimensionality reduction is a process used to reduce the number of features (or dimensions) in a dataset while retaining most of the important information. This can be achieved through techniques like Principal Component Analysis (PCA), t-SNE, or UMAP. Here's an example of compressing data using PCA in Python.

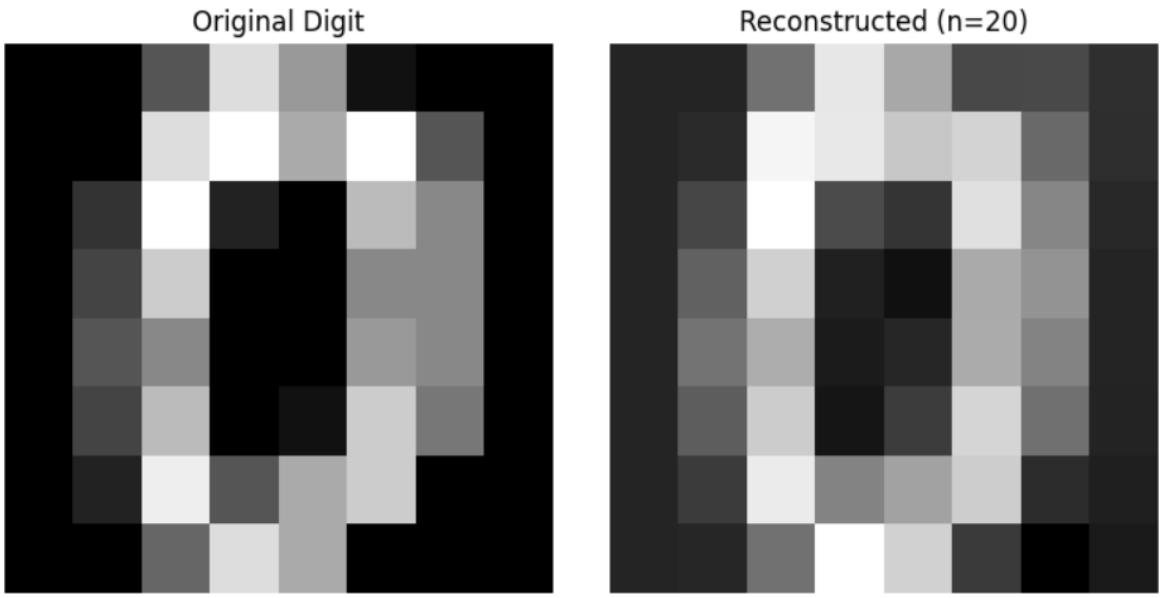
Python program specifically for compressing data using **Principal Component Analysis (PCA)**. This example includes both compression and reconstruction of the data, showing how PCA reduces dimensionality and retains most of the important information.

➢ Program

```
import numpy as np

import pandas as pd

from sklearn.decomposition import PCA

from sklearn.datasets import load_digits

import matplotlib.pyplot as plt

# Load sample data (Digits dataset)

digits = load_digits()

X = digits.data  # Feature matrix

y = digits.target  # Labels

# Check original data shape

print("Original data shape:", X.shape)

# Apply PCA for dimensionality reduction

n_components = 20  # Compress to 20 dimensions

pca = PCA(n_components=n_components)

X_compressed = pca.fit_transform(X)

# Check compressed data shape

print("Compressed data shape:", X_compressed.shape)

# Reconstruct the data from the compressed form

X_reconstructed = pca.inverse_transform(X_compressed)

# Compare original and reconstructed data
```

```
print("Reconstructed data shape:", X_reconstructed.shape)
# Visualize one original and one reconstructed digit
plt.figure(figsize=(8, 4))
# Original
plt.subplot(1, 2, 1)
plt.imshow(X[0].reshape(8, 8), cmap='gray')
plt.title("Original Digit")
plt.axis('off')
# Reconstructed
plt.subplot(1, 2, 2)
plt.imshow(X_reconstructed[0].reshape(8, 8), cmap='gray')
plt.title(f"Reconstructed (n={n_components})")
plt.axis('off')
plt.tight_layout()
plt.show()
```

Output



12. Python program for data clustering

Data clustering is an unsupervised machine learning technique used to group data points into clusters based on their similarities. Clustering is widely used for exploratory data analysis, pattern recognition, and feature engineering.

Data clustering is the process of grouping data points into clusters such that points within the same cluster are more similar to each other than to points in other clusters. Here's a comprehensive example of data clustering in Python using **K-Means**, **DBSCAN**, and **Agglomerative Clustering**.

Data clustering program using K-Means, DBSCAN, and Agglomerative Clustering.

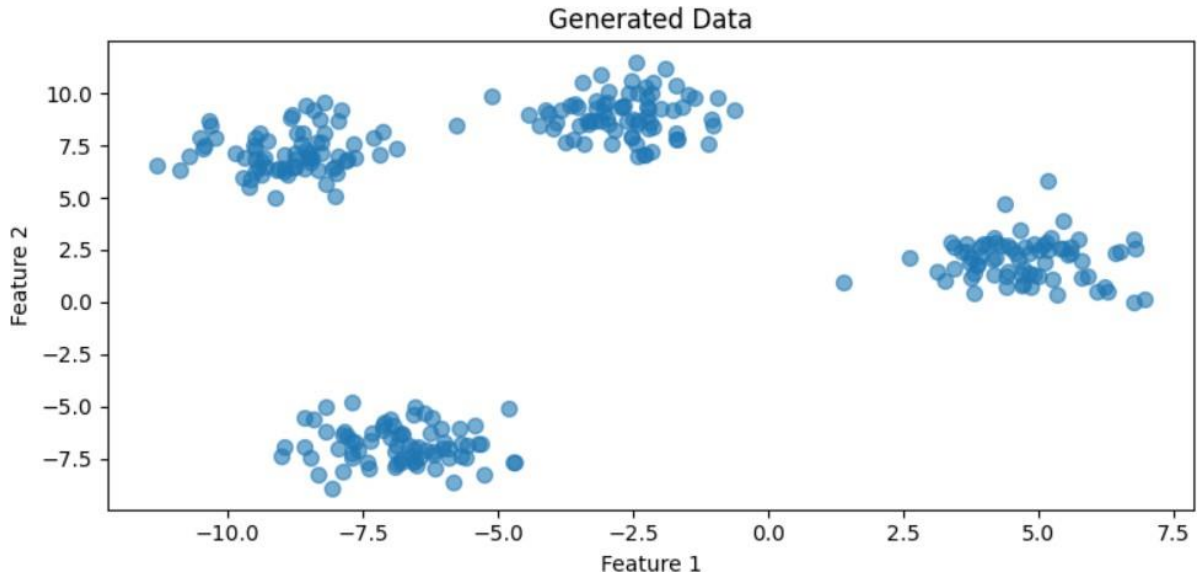
```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make_blobs

from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering

from sklearn.metrics import silhouette_score

# Generate synthetic dataset

n_samples = 300

n_features = 2

n_clusters = 4

random_state = 42

X, y_true = make_blobs(n_samples=n_samples, n_features=n_features, centers=n_clusters, random_state=random_state)

# Visualize the dataset

plt.scatter(X[:, 0], X[:, 1], s=50, alpha=0.6)

plt.title("Generated Data")

plt.xlabel("Feature 1")

plt.ylabel("Feature 2")

plt.show()
```

```
# 1. K-Means Clustering
kmeans = KMeans(n_clusters=n_clusters, random_state=random_state)
y_kmeans = kmeans.fit_predict(X)
kmeans_centroids = kmeans.cluster_centers_
# Visualize K-Means Clustering
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis', s=50, alpha=0.6)
plt.scatter(kmeans_centroids[:, 0], kmeans_centroids[:, 1], c='red', s=200, marker='X',
label='Centroids')
plt.title("K-Means Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
# 2. DBSCAN Clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)
y_dbscan = dbscan.fit_predict(X)
# Visualize DBSCAN Clustering
plt.scatter(X[:, 0], X[:, 1], c=y_dbscan, cmap='viridis', s=50, alpha=0.6)
plt.title("DBSCAN Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
# 3. Agglomerative Clustering
agglo = AgglomerativeClustering(n_clusters=n_clusters)
y_agglo = agglo.fit_predict(X)
# Visualize Agglomerative Clustering
plt.scatter(X[:, 0], X[:, 1], c=y_agglo, cmap='viridis', s=50, alpha=0.6)
plt.title("Agglomerative Clustering")
plt.xlabel("Feature 1")
```

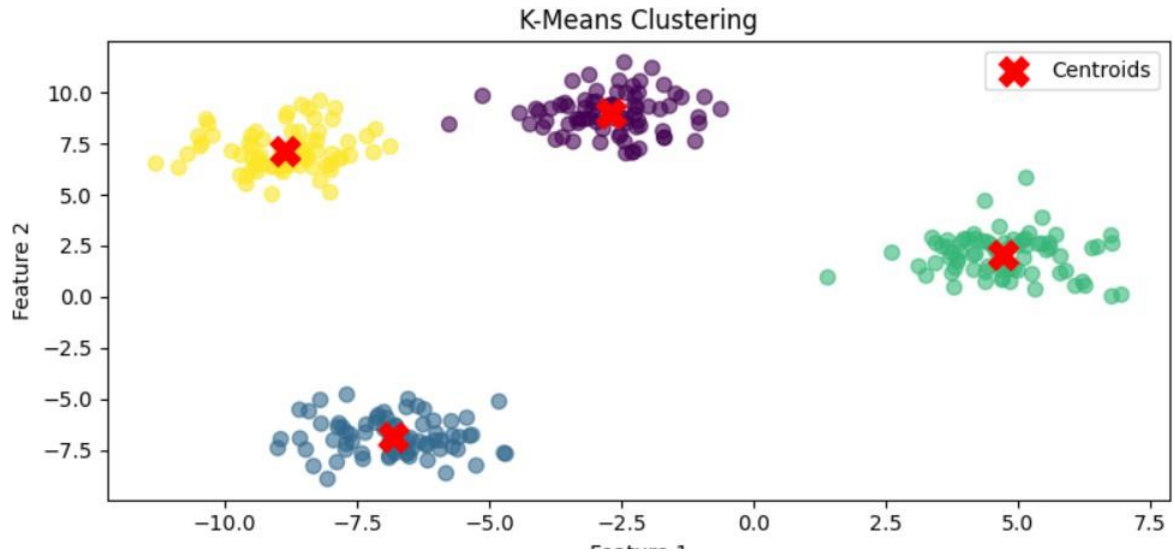
```
plt.ylabel("Feature 2")
plt.show()
# 4. Evaluate Clustering Results with Silhouette Score
print("Silhouette Score (K-Means):", silhouette_score(X, y_kmeans))
print("Silhouette Score (DBSCAN):", silhouette_score(X[y_dbscan != -1], y_dbscan[y_dbscan != -1]))
print("Silhouette Score (Agglomerative):", silhouette_score(X, y_agglo))
```

Outout

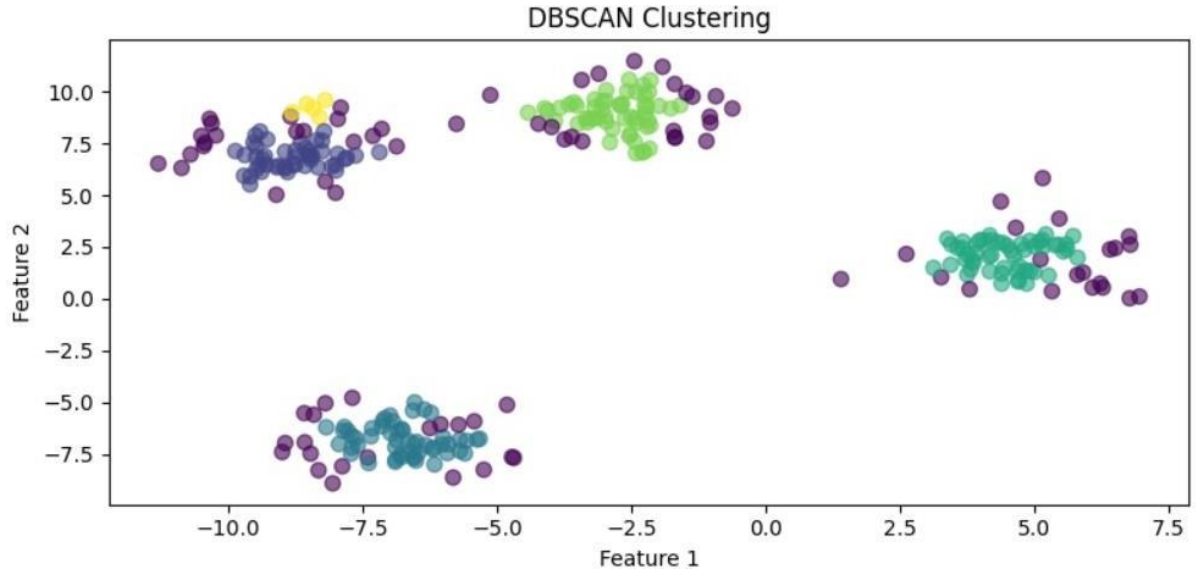
1.



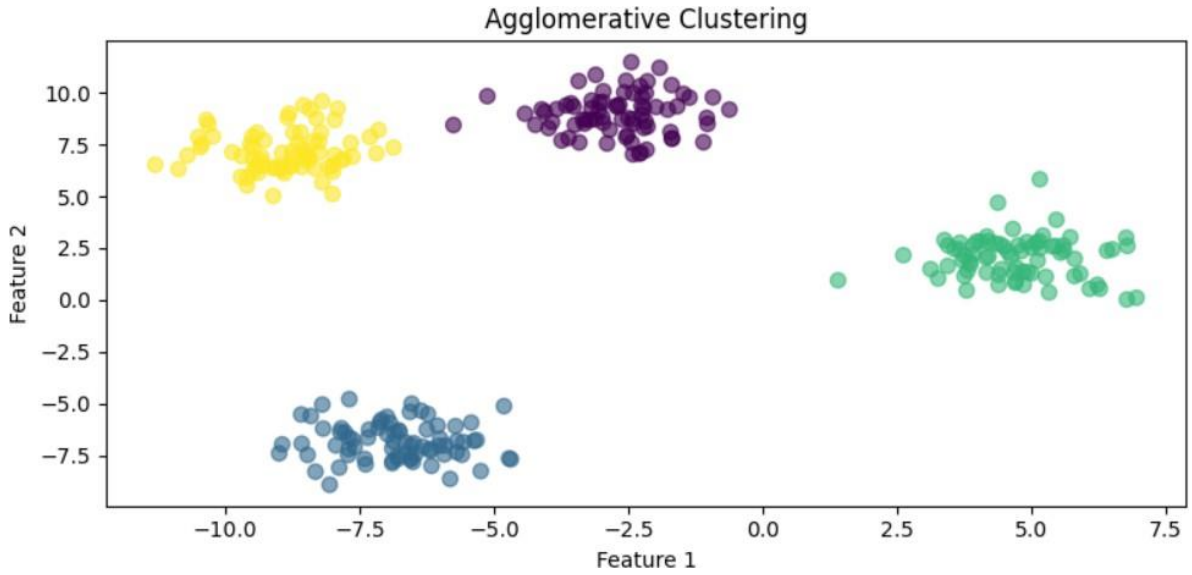
2.



3.


DBSCAN Clustering
Feature 2
Feature 1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5

4.


Agglomerative Clustering
Feature 2
Feature 1
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5

13. Python programs of classification

Classification is a large domain in the field of statistics and machine learning. Generally, classification can be broken down into two areas:

1. **Binary classification**, where we wish to group an outcome into one of two groups.
2. **Multi-class classification**, where we wish to group an outcome into one of multiple (more than two) groups.

In this post, the main focus will be on using a variety of classification algorithms across both of these domains, less emphasis will be placed on the theory behind them.

We can use libraries in Python such as Scikit-Learn for machine learning models, and Pandas to import data as data frames.

These can easily be installed and imported into Python with pip:

```
$ python3 -m pip install sklearn
$ python3 -m pip install pandas
```

```
import sklearn as sk
import pandas as pd
```

➢ Logistic Regression

Logistic regression is a widely used statistical technique for binary classification problems in machine learning. It predicts the probability of an event occurring (1) or not occurring (0) based on a set of input features. In Python, you can implement logistic regression using popular libraries such as Scikit-learn, Statsmodels, and TensorFlow.

➢ A python program on Logistic Regression

import numpy as np

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import classification_report, accuracy_score

Load the dataset (Iris dataset)

data = load_iris()

X = data.data # Features

```
y = (data.target == 0).astype(int)  # Binary classification: Class 0 vs. others

# Split data into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Train Logistic Regression model

model = LogisticRegression()

model.fit(X_train, y_train)

# Make predictions

y_pred = model.predict(X_test)

# Evaluate the model

print("Accuracy:", accuracy_score(y_test, y_pred))

print("Classification Report:\n", classification_report(y_test, y_pred))
```

Output

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        26
           1       1.00      1.00      1.00        19

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

➢ Decision Tree Classifier

A Decision Tree Classifier is a supervised learning algorithm used for classification tasks. It creates a tree-like model of decisions, where each internal node represents a

feature or attribute, each branch represents a decision rule, and each leaf node represents a class label or predicted outcome.

- **Key Characteristics:**

1. Hierarchical structure: The decision tree is composed of nodes and branches, forming a hierarchical structure.
2. Feature selection: The algorithm selects the most informative features at each node to split the data.
3. Splitting criteria: The algorithm uses a splitting criterion, such as Gini impurity or information gain, to determine the best feature and split point.
4. Pruning: Techniques like pre-pruning or post-pruning can be used to reduce the tree's complexity and prevent overfitting.

➢ A python program on Decision Tree Classifier

```
import numpy as np

from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import classification_report, accuracy_score

# Load the dataset (Iris dataset)

data = load_iris()

X = data.data  # Features

y = (data.target == 0).astype(int)  # Binary classification: Class 0 vs. others

# Split data into train and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Train Logistic Regression model

model = LogisticRegression()

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Evaluate the model

print("Accuracy:", accuracy_score(y_test, y_pred))

print("Classification Report:\n", classification_report(y_test, y_pred))
```

Output

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        26
           1       1.00      1.00      1.00        19

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

➢ Random Forest Classifier

The Random Forest Classifier is **a popular ensemble learning method in machine learning, particularly useful for classification tasks**. It combines multiple decision trees to improve the accuracy and robustness of predictions. Here's a breakdown of the key aspects:

➢ How it works:

1. **Bootstrap Sampling**: Each decision tree is trained on a random subset of the training data, with replacement (bagging).
2. **Feature Randomness**: At each node, a random subset of features is selected for splitting, reducing correlation among trees.
3. **Multiple Trees**: A forest consists of multiple decision trees, each trained independently.
4. **Voting**: The final prediction is determined by aggregating the outputs of individual trees, typically through majority voting or averaging.

➢ A python program on Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split
```

```
from sklearn.datasets import load_iris
from sklearn.metrics import classification_report, accuracy_score
# Load dataset
data = load_iris()
X = data.data
y = data.target
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
# Train Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
# Make predictions
y_pred = rf_model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

Output

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

➢ Support Vector Machine

A Support Vector Machine (SVM) is a powerful machine learning algorithm widely used for both linear and nonlinear classification, as well as regression and outlier detection tasks. SVMs are highly adaptable, making them suitable for various applications such as text classification, image classification, spam detection, handwriting identification, gene expression analysis, face detection, and anomaly detection.

➢ Key Concepts

1. Max-Margin Classifier: SVMs focus on finding the maximum separating hyperplane between different classes in the target feature space, making them robust for both binary and multiclass classification.
2. Kernel Trick: SVMs use kernel functions to transform the data into a higher-dimensional feature space, enabling linear separation of nonlinearly separable data.
3. Support Vectors: Data points that are closest to the optimal hyperplane are called support vectors, and they determine the position and orientation of the hyperplane.

➢ A python program on support vector machine

```
from sklearn.datasets import load_iris

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

from sklearn.metrics import classification_report, accuracy_score

# Load dataset

data = load_iris()

X = data.data

y = data.target

# Split the data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Train SVM Classifier

svm_model = SVC(kernel='linear', random_state=42)

svm_model.fit(X_train, y_train)

# Make predictions
```

```
y_pred = svm_model.predict(X_test)

# Evaluate the model

print("Accuracy:", accuracy_score(y_test, y_pred))

print("Classification Report:\n", classification_report(y_test, y_pred))
```

Output

```
Accuracy: 1.0
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

➢ K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a non-parametric, supervised machine learning algorithm used for both classification and regression tasks. It was first developed by Evelyn Fix and Joseph Hodges in 1951 and later expanded by Thomas Cover.

➢ Key Concepts

1. K: The number of nearest neighbors to consider for prediction. A small value of K (e.g., 1 or 3) can lead to overfitting, while a larger value (e.g., 5 or 10) can result in smoother predictions.
2. Distance metric: The algorithm calculates the distance between the new data point and existing data points in the training set. Common distance metrics include Euclidean distance, Manhattan distance, and Minkowski distance.
3. Majority voting: In classification, the algorithm assigns the class label of the majority of the K nearest neighbors to the new data point.

➢ A python program on K-Nearest Neighbors (KNN)

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
# Load dataset
data = load_iris()
X = data.data
y = data.target
# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
# Train KNN Classifier
knn_model = KNeighborsClassifier(n_neighbors=3)
knn_model.fit(X_train, y_train)
# Make predictions
y_pred = knn_model.predict(X_test)
# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
```

Output

```
Accuracy: 1.0
```

14. python program on model evaluation k-fold cross validation

Python program that demonstrates how to evaluate a model using K-Fold Cross-Validation with scikit-learn. This method helps assess the model's performance by splitting the dataset into k folds and using each fold as a test set while the others are used for training.

➢ python program on model evaluation k-fold cross validation

```
from sklearn.datasets import load_iris

from sklearn.model_selection import KFold, cross_val_score

from sklearn.ensemble import RandomForestClassifier

import numpy as np

# Load the Iris dataset

data = load_iris()

X = data.data  # Features

y = data.target  # Target labels

# Define the model

model = RandomForestClassifier(random_state=42)

# Set up K-Fold Cross-Validation

k = 5  # Number of folds

kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Perform Cross-Validation

scores = cross_val_score(model, X, y, cv=kf, scoring='accuracy')

# Print the results

print(f"Cross-Validation Scores: {scores}")

print(f"Mean Accuracy: {np.mean(scores):.4f}")

print(f"Standard Deviation: {np.std(scores):.4f}")
```

Output

```
Cross-Validation Scores: [1.         0.96666667 0.93333333 0.93333333 0.96666667]

Mean Accuracy: 0.9600

Standard Deviation: 0.0249
```

Viva - Voice Questions

1. What is Python?

Python is a high-level, interpreted programming language known for its easy-to-read syntax and dynamic typing. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

2. What are Python's key features?

Python has several key features:

- Easy-to-read syntax.
- Interpreted language.
- Dynamically typed.
- Supports multiple programming paradigms.
- Large standard library.
- Automatic memory management (garbage collection).
- Extensible and embeddable.

3. What are Python's data types?

Python's built-in data types include:

- Numeric types: int, float, complex.
- Sequence types: list, tuple, range.
- Text type: str.
- Mapping type: dict.
- Set types: set, frozen set.
- Boolean type: bool.
- Binary types: bytes, bytearray, memoryview.
- None type: NoneType.

4. What is the difference between a list and a tuple in Python?

A list is mutable, meaning its elements can be changed, whereas a tuple is immutable, meaning its elements cannot be modified after creation.

5. What are Python decorators?

A decorator is a function that modifies the behavior of another function or method. It is typically used to add functionality to an existing code without modifying the code itself.

6. What is the difference between `deepcopy()` and `copy()` in Python?

The copy() function creates a shallow copy of an object, meaning changes to nested objects within the copy will affect the original. deepcopy(), on the other hand, creates a deep copy, recursively copying all objects, meaning changes to nested objects will not affect the original.

7. Explain the concept of lambda functions in Python.

A lambda function is a small, anonymous function defined with the lambda keyword. It can have any number of arguments but only one expression. It is often used when a simple function is needed for a short period of time.

Example:

```
square = lambda x: x ** 2
print(square(5))  # Output: 25
```

8. What is the difference between `is` and `==` in Python?

is checks for object identity, i.e., whether two variables refer to the same object in memory. == checks for equality, i.e., whether two variables have the same value.

9. What are Python's control flow statements?

Python's control flow statements include:

- Conditional statements: if, elif, else.
- Looping statements: for, while.
- Control flow modifiers: break, continue, pass.

10. Explain Python's exception handling mechanism.

Python uses try, except, else, and finally to handle exceptions. The try block contains code that may raise an exception. The except block handles the exception. The else block runs if no exception is raised, and finally runs no matter what.

Example:

```
try:
    x = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
finally:
    print("This will always execute")
```

11. What are Python generators?

A generator is a special type of iterator in Python that uses the yield keyword to produce values one at a time. Generators are memory efficient because they generate values on-the-fly instead of storing them in memory.

12. What is the purpose of the `self` keyword in Python classes?

self refers to the instance of the class and allows you to access instance variables and methods within the class. It must be the first parameter in instance methods.

13. How is Python interpreted?

Python language is an interpreted language. Python program runs directly from the source code. It converts the source code that is written by the programmer into an intermediate language, which is again translated into machine language that has to be executed

14. How is memory managed in Python?

Python memory is managed by Python private heap space. All Python objects and data structures are located in a private heap. The programmer does not have an access to this private heap, and the interpreter takes care of this Python private heap.

The allocation of Python heap space for Python objects is done by the Python memory manager. The core API gives access to some tools for the programmer to code.

Python also has an inbuilt garbage collector, which recycles all the unused memory and frees the memory and makes it available to the heap space

15. How are arguments passed by value or by reference?

Everything in Python is an object, and all variables hold references to the objects. The reference values are according to the functions. Therefore, you cannot change the value of the references. However, you can change the objects if it is mutable

16. Explain namespace in Python

In Python, every name introduced has a place where it lives and can be hooked for. This is known as a namespace. It is like a box where a variable name is mapped to the object placed. Whenever the variable is searched out, this box will be searched to get the corresponding object.

17. Why lambda forms in python do not have statements?

A lambda form in python does not have statements as it is used to make new function object and then return them at runtime.

18. Explain pass in Python

Pass means no-operation Python statement, or in other words, it is a place holder in a compound statement, where there should be a blank left, and nothing has to be written there.

19. What is module and package in Python?

In Python, module is the way to structure a program. Each Python program file is a module, which imports other modules like objects and attributes.

The folder of Python program is a package of modules. A package can have modules or subfolders.

20. What are the rules for local and global variables in Python?

Here are the rules for local and global variables in Python:

Local variables: If a variable is assigned a new value anywhere within the function's body, it's assumed to be local.

Global variables: Those variables that are only referenced inside a function are implicitly global.

SAMPLE PROGRAMS

1. Write a python program to find factorial of a given number using functions
2. Write a Python function that takes two lists and returns True if they are equal otherwise false
3. Write a python program to check whether the given string is palindrome or not.
4. Write a python program to find largest number among three numbers.
5. Write a python program which accepts the radius of a circle from user and computes the area (use math module)
6. Write a python Program to call data member and function using classes and objects