**Experiment 1,2&3**print('hello');

Output-

hello

a **=** int(input("Enter a number: "))

b **=** int(input("Enter another number: "))

print(a**+**b);

Output-

Enter a number: 10

Enter another number: 5

15

In [4]:

**for** i **in** range(0,4):

print("10", end**=**"\n")

Output-

10

10

10

10

In [5]:

i **=** 0

**while**(i**<**5):

print(i, end**=**" ")

i **=** i**+**1

Output-

0 1 2 3 4

In [7]:

​

**def** greet():

name **=** input("What's your name? ")

print(f"Hello, {name}!")

greet()

​

Output-

What's your name? HARSH

Hello, HARSH!

Worksheets 1 to 4

Worksheet 1

In [2]:

1.Create a code block that returns 'True' **if** a given date **is** even **and** returns ‘False**!** **if** it **is** **not**

2.Create a code block that can identify whether three given values can form the sides of a triangle (Hint

: The sum of two sides of a triangle **is** always larger than the third one)

3.Write a code that returns 'True' **if** the triangle **with** the three given sides **is** equilateral

4.Write a code that returns 'True' **if** the triangle **with** the three given sides **is** isosceles

5.Write a code that returns 'True' **if** the triangle **with** the three given sides **is** scalene

6.Write a code that returns 'True'’ **if** the triangle **with** the three given sides follows the Pythagoras theorm

​

In [6]:

**def** is\_even\_date(date):

**return** date **%** 2 **==** 0

​

date **=** int(input("Enter a date: "))

print(is\_even\_date(date))

​ Output-

Enter a date: 10

True

In [7]:

**def** is\_triangle(side1, side2, side3):

**return** (side1 **+** side2 **>** side3) **and** (side1 **+** side3 **>** side2) **and** (side2 **+** side3 **>** side1)

​

side1 **=** float(input("Enter the length of side 1: "))

side2 **=** float(input("Enter the length of side 2: "))

side3 **=** float(input("Enter the length of side 3: "))

print(is\_triangle(side1, side2, side3))

​ Output-

Enter the length of side 1: 5

Enter the length of side 2: 3

Enter the length of side 3: 5

True

In [9]:

**def** is\_equilateral(side1, side2, side3):

**return** side1 **==** side2 **==** side3

​

​

side1 **=** float(input("Enter the length of side 1: "))

side2 **=** float(input("Enter the length of side 2: "))

side3 **=** float(input("Enter the length of side 3: "))

print(is\_equilateral(side1, side2, side3))

​ Output-

Enter the length of side 1: 8

Enter the length of side 2: 9

Enter the length of side 3: 10

False

In [10]:

**def** is\_isosceles(side1, side2, side3):

**return** side1 **==** side2 **or** side1 **==** side3 **or** side2 **==** side3

​

side1 **=** float(input("Enter the length of side 1: "))

side2 **=** float(input("Enter the length of side 2: "))

side3 **=** float(input("Enter the length of side 3: "))

print(is\_isosceles(side1, side2, side3))

​ Output-

Enter the length of side 1: 30

Enter the length of side 2: 30

Enter the length of side 3: 60

True

In [11]:

**def** is\_scalene(side1, side2, side3):

**return** side1 **!=** side2 **!=** side3 **!=** side1

​

​

side1 **=** float(input("Enter the length of side 1: "))

side2 **=** float(input("Enter the length of side 2: "))

side3 **=** float(input("Enter the length of side 3: "))

print(is\_scalene(side1, side2, side3))

​ Output-

Enter the length of side 1: 24

Enter the length of side 2: 53

Enter the length of side 3: 6

True

In [12]:

**def** is\_pythagorean(side1, side2, side3):

sides **=** sorted([side1, side2, side3])

**return** sides[0]**\*\***2 **+** sides[1]**\*\***2 **==** sides[2]**\*\***2

side1 **=** float(input("Enter the length of side 1: "))

side2 **=** float(input("Enter the length of side 2: "))

side3 **=** float(input("Enter the length of side 3: "))

print(is\_pythagorean(side1, side2, side3))

​ Output-

Enter the length of side 1: 50

Enter the length of side 2: 50

Enter the length of side 3: 90

False

In [13]:

Worksheet 2

Questions

1.Write a code to calculate the length ofa given string

2.Write a Python code to get a single string **from** two given strings, separated by a space **and** swap

the first two characters of each string

3.Write a Python code to add ‘ily' at the end of any given string

4.Write a Python code to get all the even**-**positioned characters **in** a string

5.Write a Python code to get all the odd**-**positioned characters **in** a string

6.Write a Python code to reverse a given string

7.Write a Python code to reverse a given string **and** concatenate it at the end of the original string

In [14]:

**def** calculatestrlen(string):

**return** len(string)

​

input\_string **=** input("Enter a string: ")

print("Length of the string:", calculatestrlen(input\_string))

​ Output-

Enter a string: harsh

Length of the string: 5

In [16]:

**def** swapchar(string1, string2):

new\_string1 **=** string2[:2] **+** string1[2:]

new\_string2 **=** string1[:2] **+** string2[2:]

**return** new\_string1 **+** ' ' **+** new\_string2

​

string1 **=** input("Enter the first string: ")

string2 **=** input("Enter the second string: ")

print(swapchar(string1, string2))

​ Output-

Enter the first string: machine

Enter the second string: learning

lechine maarning

In [20]:

string**=**input("enter the string: ")

print(string**+**"ily")

Output-

enter the string: abc

abcily

In [22]:

string1**=** input("enter the firststring: ")

print(string1[1: :2])

enter the firststring: harsh

as

In [25]:

string **=** input("Enter a string: ")

print( string[::2])

​

Output-

Enter a string: harsh

hrh

In [26]:

stringl**=** input("enter the firststring:- ")

print(string1[: :**-**1])

Output-

enter the firststring:- harsh

hsrah

In [29]:

string1**=** input("enter the firststring:")

srtingrevarse**=**string1[: :**-**1]

print(string1**+**srtingrevarse)

Output-

enter the firststring:example

exampleelpmaxe

In [30]:

Worksheet 3

Questions

1.Write a Python code to sort a nested list

2.Write a Python code to replace the third last element **in** a list by 0

3.Write a code to get the smallest **and** largest numbers **from** a list

4.Write a Python code to print a given list after removing the elements at the even numbered

positions

5.Write a Python code to append a list Write a Python code to find the second largest number

**in** a list Write a Python code to insert 0 at all the odd indices **in** a lis

x **=** int(input("Enter the number of elements in the list: "))

list1 **=** []

**for** i **in** range(x):

a **=** input("Enter the element: ")

list1.append(a)

​

print("List 1:", list1)

​

list2 **=** []

**for** i **in** range(x):

a **=** input("Enter the element: ")

list2.append(a)

​

print("List 2:", list2)

​

list3 **=** []

list3.append(list1)

list3.append(list2)

​

print("Nested List before sorting:", list3)

sorted\_list **=** sorted(list3)

print("Nested List after sorting:", sorted\_list)

​ Output-

Enter the number of elements in the list: 3

Enter the element: 1

Enter the element: 2

Enter the element: 3

List 1: ['1', '2', '3']

Enter the element: 5

Enter the element: 1

Enter the element: 2

List 2: ['5', '1', '2']

Nested List before sorting: [['1', '2', '3'], ['5', '1', '2']]

Nested List after sorting: [['1', '2', '3'], ['5', '1', '2']]

In [4]:

a **=** int(input("Enter the number of elements in list: "))

list1 **=** []

**for** i **in** range(0, a):

b **=** input("Enter the element: ")

list1.append(b)

print("Original List:", list1)

​

**for** i **in** range(a **-** 1, **-**1, **-**1):

**if** i **==** (a **-** 3):

list1[i] **=** '0'

**break**

​

print("List after replacing third last element with 0:", list1)

​ Output-

Enter the number of elements in list: 3

Enter the element: 1

Enter the element: 2

Enter the element: 4

Original List: ['1', '2', '4']

List after replacing third last element with 0: ['0', '2', '4']

In [8]:

a **=** int(input("Enter the number of elements in list: "))

list1 **=** []

**for** i **in** range(0, a):

b **=** int(input("Enter the element: "))

list1.append(b)

print("List:", list1)

print("Maximum:", max(list1))

print("Minimum:", min(list1))

​ Output-

Enter the number of elements in list: 4

Enter the element: 10

Enter the element: 30

Enter the element: 20

Enter the element: 5

List: [10, 30, 20, 5]

Maximum: 30

Minimum: 5

In [9]:

a **=** int(input("Enter the number of elements in list: "))

list1 **=** []

**for** i **in** range(0, a):

b **=** input("Enter the element: ")

list1.append(b)

​

print("Original List:", list1)

​

**del** list1[1:]

print("List after removing elements starting from index 1:", list1)

​ Output-

Enter the number of elements in list: 4

Enter the element: 10

Enter the element: 20

Enter the element: 30

Enter the element: 40

Original List: ['10', '20', '30', '40']

List after removing elements starting from index 1: ['10']

In [15]:

​

mylist **=** []

**for** \_ **in** range(int(input("Enter the number of elements in the list: "))):

mylist.append(int(input("Enter an element: ")))

​

print("List after appending elements:", mylist)

​

sortedlist **=** sorted(mylist)

**if** len(sortedlist) **<** 2:

print("List does not contain enough elements.")

**else**:

print("Second largest number in the list:", sortedlist[**-**2])

​ Output-

Enter the number of elements in the list: 4

Enter an element: 10

Enter an element: 20

Enter an element: 30

Enter an element: 40

List after appending elements: [10, 20, 30, 40]

Second largest number in the list: 30

In [16]:

mylist **=** []

**for** \_ **in** range(int(input("Enter the number of elements in the list: "))):

mylist.append(int(input("Enter an element: ")))

​

print("Original List:", mylist)

​

​

**for** i **in** range(1, len(mylist), 2):

mylist.insert(i, 0)

​

print("List after inserting 0 at odd indices:", mylist)

​ Output-

Enter the number of elements in the list: 5

Enter an element: 1

Enter an element: 2

Enter an element: 3

Enter an element: 4

Enter an element: 5

Original List: [1, 2, 3, 4, 5]

List after inserting 0 at odd indices: [1, 0, 2, 0, 3, 4, 5]

Worksheet 4

Questions

1.Write a code to initialise a dictionary using the dict function

2.Write a code to get a sorted listed of values in a dictionary

3. Write a Python code to get a list of the keys in a dictionary and then uppercase all the keys ​

​

dictionary1 **=** dict(name**=**'harsh', age**=**'20', hobby**=**['fooball', 'philosophy'])

​

print(dictionary1)

​Output-

{'name': 'harsh', 'age': '20', 'hobby': ['fooball', 'philosophy']}

In [4]:

​

dictionary1 **=** {'name': 'harsh', 'age': '20', 'hobby': 'football'}

​

print(sorted(dictionary1.values()))

​ Output-

['20', 'football', 'harsh']

In [5]:

​

dictionary1 **=** {'name': 'harsh', 'age': '20', 'hobby': 'football'}

dictionary1 **=** {key.upper(): value **for** key, value **in** dictionary1.items()}

print(dictionary1)

​ Output-

{'NAME': 'harsh', 'AGE': '20', 'HOBBY': 'football'}

NumPy:

**import** numpy **as** np

np.ones(5)

Output-

outpOut[8]:

array([1., 1., 1., 1., 1.])

In [10]:

np.arange(1,10,2)

Out[10]:

Output-

array([1, 3, 5, 7, 9])

In [12]:

np.linspace(1,2,10)

Out[12]:

Output-

array([1. , 1.11111111, 1.22222222, 1.33333333, 1.44444444,

1.55555556, 1.66666667, 1.77777778, 1.88888889, 2. ])

In [17]:

array1 **=** np.random.randint(1, 10, size**=**10)

print(array1)

​

Output-

[9 7 7 4 5 4 5 5 4 9]

In [18]:

a **=** np.arange(6)

print("1D array:\n", a)

b **=** np.arange(12).reshape(4, 3)

print("2D array:\n", b)

c **=** np.arange(24).reshape(2, 3, 4)

print("3D array:\n", c)

​

Output-

1D array:

[0 1 2 3 4 5]

2D array:

[[ 0 1 2]

[ 3 4 5]

[ 6 7 8]

[ 9 10 11]]

3D array:

[[[ 0 1 2 3]

[ 4 5 6 7]

[ 8 9 10 11]]

[[12 13 14 15]

[16 17 18 19]

[20 21 22 23]]]

In [22]:

A **=** np.array([[1, 2], [3, 4]])

B **=** np.array([[5, 6], [7, 8]])

​

result **=** A.dot(B)

​

print("Matrix A:\n", A)

print("Matrix B:\n", B)

print("Matrix product:\n", result)

​

Output-

Matrix A:

[[1 2]

[3 4]]

Matrix B:

[[5 6]

[7 8]]

Matrix product:

[[19 22]

[43 50]]

In [24]:

arr **=** np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

subset **=** arr[:2]

print("Subset of the array (first two rows):\n", subset)

slice\_arr **=** arr[1:4]

print("Slice of the array (elements from index 1 to 4):\n", slice\_arr)

index **=** arr[2, 1]

print("Element at row 2, column 1:", index)

print("Elements in arr:")

**for** row **in** arr:

**for** element **in** row:

print(element, end**=**" ")

print()

Output-

Subset of the array (first two rows):

[[1 2 3]

[4 5 6]]

Slice of the array (elements from index 1 to 4):

[[4 5 6]

[7 8 9]]

Element at row 2, column 1: 8

Elements in arr:

1 2 3

4 5 6

7 8 9

In [26]:

**import** numpy **as** np

​

arr **=** np.array([10,11,2,4,63,22])

print("Given array:", arr)

​

sorted\_index\_array **=** np.argsort(arr)

sorted\_array **=** arr[sorted\_index\_array]

print("Sorted array:", sorted\_array)

​

n **=** 1

result **=** sorted\_array[**-**n:]

print("{} largest value:".format(n), result[0])

​

Output-

Given array: [10 11 2 4 63 22]

Sorted array: [ 2 4 10 11 22 63]

1 largest value: 63

import numpy as np

myarr1 = np.array([7,5,10,1,2,11], np.int32)

print(myarr1)

print(myarr1.tolist())

print(myarr1[2])

myarr1[2] = 70

print(myarr1)

print(myarr1.shape)

myarr1.sort()

print(myarr1)

Output: [ 7 5 10 1 2 11]

[7, 5, 10, 1, 2, 11]

10

[ 7 5 70 1 2 11]

(6,)

[ 1 2 5 7 11 70]

myarr2 = np.array([[10,20,30,40,50]], np.int8)

print(myarr2)

print(myarr2[0][2])

myarr2[0][2] = 70

print(myarr2)

print(myarr2.shape)

Output: [[10 20 30 40 50]]

30

[[10 20 70 40 50]]

(1, 5)

listarray = np.array([[10,20,30],[50,60,70],[80,90,100]])

print(listarray)

Output: [[ 10 20 30]

[ 50 60 70]

[ 80 90 100]]

print(listarray.shape)

print(listarray.size)

print(listarray.dtype)

Output: (3, 3)

9

Int32

np.array({10,20,30,40})

Output: array({40, 10, 20, 30}, dtype=object)

zeroArray = np.zeros((2,4))

Output: array([[0., 0., 0., 0.],

[0., 0., 0., 0.]])

rangeArray = np.arange(15)

Output: array([ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])

lnArray = np.linspace(1,10,7)

Output: array([ 1. , 2.5, 4. , 5.5, 7. , 8.5, 10. ])

emptyArray1 = np.empty(7)

print(emptyArray1)

emptyArray2 = np.empty((2,5))

Output: [ 1. 2.5 4. 5.5 7. 8.5 10. ]

array([[nan, 0., 0., 0., 0.],

[ 0., 0., 0., 0., 0.]])

identityArray = np.identity(10)

print(identityArray)

Output: [[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]

demoArray = np.arange(99)

demoArray = demoArray.reshape(3,33)

3\*31 will have 93 elements

demoArray

print(demoArray.shape)

demoArray = demoArray.ravel()

demoArray

print(demoArray.shape)

Output: (3, 33)

(99,)

x = [[7,2,1], [10,3,21], [5,9,0]]

ar = np.array(x)

ar

print(ar.size)

print(ar.sum(axis=0))

print(ar.sum(axis=1))

print(ar.T)

print(ar.ndim)

print(ar.nbytes)

for element in ar.flat:

print(element, end=" ")

Output: 9

[22 14 22]

[10 34 14]

[[ 7 10 5]

[ 2 3 9]

[ 1 21 0]]

2

36

7 2 1 10 3 21 5 9 0

myArray = np.array([7,5,10,1,2,11])

print(myArray.argmax())

print(myArray.argmin())

print(myArray.argsort())

ind = myArray.argsort()

ls = [i for i in ind]

ls.reverse()

Output: 5

3

[3 4 1 0 2 5]

[5, 2, 0, 1, 4, 3]

myArray = np.array([[7,2,1], [10,3,21], [5,9,0]])

print(myArray.argmax())

print(myArray.argmin())

0 1 2 3 4 5 5 7 8

[7,2,1], [10,3,21], [5,9,0]

print(myArray.argmax(axis=0))

print(myArray.argmin(axis=1))

print(myArray.argsort())

print(myArray.argsort(axis=0))

Output: 5

8

[1 2 1]

[2 1 2]

[[2 1 0]

[1 0 2]

[2 0 1]]

[[2 0 2]

[0 1 0]

[1 2 1]]

array1 = np.array([[7,2,1], [10,3,21], [5,9,0]])

array2 = np.array([[1,2,1], [4,0,6], [8,1,0]])

sum\_of\_array = array1 + array2

print(sum\_of\_array)

multiplication\_of\_array = array1 \* array2

print(multiplication\_of\_array)

print(np.sqrt(array1))

Output: [[ 8 4 2]

[14 3 27]

[13 10 0]]

[[ 7 4 1]

[ 40 0 126]

[ 40 9 0]]

[[2.64575131 1.41421356 1. ]

[3.16227766 1.73205081 4.58257569]

[2.23606798 3. 0. ]]

print(array1)

print(array1.sum())

print(array1.max())

print(array1.min())

print(np.where(array1>5))

print(type(np.where(array1>5)))

print(np.count\_nonzero(array1))

print(np.nonzero(array1))

Output: [[ 7 2 1]

[10 3 21]

[ 5 9 0]]

58

21

0

(array([0, 1, 1, 2], dtype=int64), array([0, 0, 2, 1], dtype=int64))

<class 'tuple'>

8

(array([0, 0, 0, 1, 1, 1, 2, 2], dtype=int64), array([0, 1, 2, 0, 1, 2, 0, 1], dtype=int64))

import sys

python\_array = [10,20,30,40,50]

numpy\_array = np.array(python\_array)

sizeOf\_pythonArray = len(python\_array) \* sys.getsizeof(1)

print(sizeOf\_pythonArray)

sizeOf\_numpyArray = numpy\_array.size \* numpy\_array.itemsize

print(sizeOf\_numpyArray)

Ouput: 140

20

PANDAS

**import** numpy **as** np

**import** pandas **as** pd

​

a **=** pd.Series([3, 4, 5, 6])

print(a)

​

0 3

1 4

2 5

3 6

dtype: int64

In [5]:

a[1:]

Out[5]:

1 4

2 5

3 6

dtype: int64

In [6]:

b\_charseries**=**pd.Series(['a','b','c'])

b\_charseries

Out[6]:

0 a

1 b

2 c

dtype: object

In [7]:

b\_charseries[[1,0]]

Out[7]:

1 b

0 a

dtype: object

In [12]:

b **=** pd.Series(np.array(range(1, 6)) **\*\*** 3, index**=**['a', 'b', 'c', 'd', 'e'])

print(b)

a 1

b 8

c 27

d 64

e 125

dtype: int32

In [11]:

b **=** pd.Series(np.array(range(1, 6)) **\*\*** 3, index**=**['a', 'b', 'c', 'd', 'e'])

print(b)

a 1

b 8

c 27

d 64

e 125

dtype: int32

In [15]:

market\_df**=**pd.read\_csv("./market\_fact.csv")

market\_df[5:15]

Out[15]:

|  | **Ord\_id** | **Prod\_id** | **Ship\_id** | **Cust\_id** | **Sales** | **Discount** | **Order\_Quantity** | **Profit** | **Shipping\_Cost** | **Product\_Base\_Margin** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **5** | Ord\_5446 | Prod\_6 | SHP\_7608 | Cust\_1818 | 164.0200 | 0.03 | 23 | -47.64 | 6.15 | 0.37 |
| **6** | Ord\_31 | Prod\_12 | SHP\_41 | Cust\_26 | 14.7600 | 0.01 | 5 | 1.32 | 0.50 | 0.36 |
| **7** | Ord\_4725 | Prod\_4 | SHP\_6593 | Cust\_1641 | 3410.1575 | 0.10 | 48 | 1137.91 | 0.99 | 0.55 |
| **8** | Ord\_4725 | Prod\_13 | SHP\_6593 | Cust\_1641 | 162.0000 | 0.01 | 33 | 45.84 | 0.71 | 0.52 |
| **9** | Ord\_4725 | Prod\_6 | SHP\_6593 | Cust\_1641 | 57.2200 | 0.07 | 8 | -27.72 | 6.60 | 0.37 |
| **10** | Ord\_4743 | Prod\_2 | SHP\_6615 | Cust\_1641 | 4072.0100 | 0.01 | 43 | 1675.98 | 0.99 | 0.56 |
| **11** | Ord\_1925 | Prod\_6 | SHP\_2637 | Cust\_708 | 465.9000 | 0.05 | 38 | 79.34 | 4.86 | 0.38 |
| **12** | Ord\_2978 | Prod\_16 | SHP\_4112 | Cust\_1088 | 305.0500 | 0.04 | 27 | 23.12 | 3.37 | 0.57 |
| **13** | Ord\_2207 | Prod\_11 | SHP\_3093 | Cust\_839 | 3364.2480 | 0.10 | 15 | -693.23 | 61.76 | 0.78 |
| **14** | Ord\_2207 | Prod\_10 | SHP\_3006 | Cust\_839 | 1410.9300 | 0.08 | 10 | -317.48 | 36.09 | 0.77 |

In [16]:

market\_df.tail(10)

Out[16]:

|  | **Ord\_id** | **Prod\_id** | **Ship\_id** | **Cust\_id** | **Sales** | **Discount** | **Order\_Quantity** | **Profit** | **Shipping\_Cost** | **Product\_Base\_Margin** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **8389** | Ord\_4444 | Prod\_13 | SHP\_6192 | Cust\_1519 | 159.4100 | 0.00 | 44 | 34.68 | 0.98 | 0.52 |
| **8390** | Ord\_5435 | Prod\_16 | SHP\_7594 | Cust\_1798 | 316.9900 | 0.04 | 47 | -276.54 | 8.37 | 0.58 |
| **8391** | Ord\_5435 | Prod\_4 | SHP\_7594 | Cust\_1798 | 1991.8985 | 0.07 | 20 | 88.36 | 7.69 | 0.58 |
| **8392** | Ord\_5384 | Prod\_9 | SHP\_7519 | Cust\_1798 | 181.5000 | 0.08 | 43 | -6.24 | 2.50 | 0.37 |
| **8393** | Ord\_5348 | Prod\_8 | SHP\_7470 | Cust\_1798 | 356.7200 | 0.07 | 9 | 12.61 | 1.99 | 0.44 |
| **8394** | Ord\_5353 | Prod\_4 | SHP\_7479 | Cust\_1798 | 2841.4395 | 0.08 | 28 | 374.63 | 7.69 | 0.59 |
| **8395** | Ord\_5411 | Prod\_6 | SHP\_7555 | Cust\_1798 | 127.1600 | 0.10 | 20 | -74.03 | 6.92 | 0.37 |
| **8396** | Ord\_5388 | Prod\_6 | SHP\_7524 | Cust\_1798 | 243.0500 | 0.02 | 39 | -70.85 | 5.35 | 0.40 |
| **8397** | Ord\_5348 | Prod\_15 | SHP\_7469 | Cust\_1798 | 3872.8700 | 0.03 | 23 | 565.34 | 30.00 | 0.62 |
| **8398** | Ord\_5459 | Prod\_6 | SHP\_7628 | Cust\_1798 | 603.6900 | 0.00 | 47 | 131.39 | 4.86 | 0.38 |

In [17]:

​

market\_df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 8399 entries, 0 to 8398

Data columns (total 10 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Ord\_id 8399 non-null object

1 Prod\_id 8399 non-null object

2 Ship\_id 8399 non-null object

3 Cust\_id 8399 non-null object

4 Sales 8399 non-null float64

5 Discount 8399 non-null float64

6 Order\_Quantity 8399 non-null int64

7 Profit 8399 non-null float64

8 Shipping\_Cost 8399 non-null float64

9 Product\_Base\_Margin 8336 non-null float64

dtypes: float64(5), int64(1), object(4)

memory usage: 656.3+ KB

In [18]:

market\_df.describe()

Out[18]:

|  | **Sales** | **Discount** | **Order\_Quantity** | **Profit** | **Shipping\_Cost** | **Product\_Base\_Margin** |
| --- | --- | --- | --- | --- | --- | --- |
| **count** | 8399.000000 | 8399.000000 | 8399.000000 | 8399.000000 | 8399.000000 | 8336.000000 |
| **mean** | 1775.878179 | 0.049671 | 25.571735 | 181.184424 | 12.838557 | 0.512513 |
| **std** | 3585.050525 | 0.031823 | 14.481071 | 1196.653371 | 17.264052 | 0.135589 |
| **min** | 2.240000 | 0.000000 | 1.000000 | -14140.700000 | 0.490000 | 0.350000 |
| **25%** | 143.195000 | 0.020000 | 13.000000 | -83.315000 | 3.300000 | 0.380000 |
| **50%** | 449.420000 | 0.050000 | 26.000000 | -1.500000 | 6.070000 | 0.520000 |
| **75%** | 1709.320000 | 0.080000 | 38.000000 | 162.750000 | 13.990000 | 0.590000 |
| **max** | 89061.050000 | 0.250000 | 50.000000 | 27220.690000 | 164.730000 | 0.850000 |

In [19]:

market\_df.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 8399 entries, 0 to 8398

Data columns (total 10 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Ord\_id 8399 non-null object

1 Prod\_id 8399 non-null object

2 Ship\_id 8399 non-null object

3 Cust\_id 8399 non-null object

4 Sales 8399 non-null float64

5 Discount 8399 non-null float64

6 Order\_Quantity 8399 non-null int64

7 Profit 8399 non-null float64

8 Shipping\_Cost 8399 non-null float64

9 Product\_Base\_Margin 8336 non-null float64

dtypes: float64(5), int64(1), object(4)

memory usage: 656.3+ KB

In [20]:

market\_df.sort\_values(by**=**['Prod\_id', 'Sales'], ascending **=** **True**)

Out[20]:

|  | **Ord\_id** | **Prod\_id** | **Ship\_id** | **Cust\_id** | **Sales** | **Discount** | **Order\_Quantity** | **Profit** | **Shipping\_Cost** | **Product\_Base\_Margin** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1761** | Ord\_286 | Prod\_1 | SHP\_387 | Cust\_90 | 18.15 | 0.04 | 1 | -7.26 | 6.13 | 0.57 |
| **3185** | Ord\_2314 | Prod\_1 | SHP\_3171 | Cust\_899 | 18.16 | 0.03 | 1 | -7.25 | 6.13 | 0.57 |
| **3819** | Ord\_439 | Prod\_1 | SHP\_587 | Cust\_136 | 18.73 | 0.05 | 1 | -6.68 | 6.13 | 0.57 |
| **6213** | Ord\_2746 | Prod\_1 | SHP\_3767 | Cust\_1030 | 22.61 | 0.03 | 1 | -8.40 | 7.51 | 0.57 |
| **387** | Ord\_3713 | Prod\_1 | SHP\_5145 | Cust\_1307 | 27.83 | 0.09 | 2 | -22.14 | 9.45 | 0.60 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **171** | Ord\_2973 | Prod\_9 | SHP\_6073 | Cust\_1480 | 5410.95 | 0.09 | 36 | 2077.91 | 19.99 | 0.39 |
| **1533** | Ord\_4059 | Prod\_9 | SHP\_5660 | Cust\_1378 | 5587.20 | 0.05 | 36 | 2254.16 | 19.99 | 0.39 |
| **2506** | Ord\_262 | Prod\_9 | SHP\_358 | Cust\_66 | 6553.45 | 0.03 | 39 | 2969.81 | 19.99 | 0.39 |
| **1485** | Ord\_4356 | Prod\_9 | SHP\_6074 | Cust\_1481 | 6831.72 | 0.01 | 41 | 3081.02 | 19.99 | 0.39 |
| **2519** | Ord\_2197 | Prod\_9 | SHP\_2994 | Cust\_827 | 7522.80 | 0.04 | 48 | 3187.37 | 19.99 | 0.39 |

8399 rows × 10 columns

In [21]:

market\_df.reset\_index()

Out[21]:

|  | **index** | **Ord\_id** | **Prod\_id** | **Ship\_id** | **Cust\_id** | **Sales** | **Discount** | **Order\_Quantity** | **Profit** | **Shipping\_Cost** | **Product\_Base\_Margin** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 0 | Ord\_5446 | Prod\_16 | SHP\_7609 | Cust\_1818 | 136.8100 | 0.01 | 23 | -30.51 | 3.60 | 0.56 |
| **1** | 1 | Ord\_5406 | Prod\_13 | SHP\_7549 | Cust\_1818 | 42.2700 | 0.01 | 13 | 4.56 | 0.93 | 0.54 |
| **2** | 2 | Ord\_5446 | Prod\_4 | SHP\_7610 | Cust\_1818 | 4701.6900 | 0.00 | 26 | 1148.90 | 2.50 | 0.59 |
| **3** | 3 | Ord\_5456 | Prod\_6 | SHP\_7625 | Cust\_1818 | 2337.8900 | 0.09 | 43 | 729.34 | 14.30 | 0.37 |
| **4** | 4 | Ord\_5485 | Prod\_17 | SHP\_7664 | Cust\_1818 | 4233.1500 | 0.08 | 35 | 1219.87 | 26.30 | 0.38 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **8394** | 8394 | Ord\_5353 | Prod\_4 | SHP\_7479 | Cust\_1798 | 2841.4395 | 0.08 | 28 | 374.63 | 7.69 | 0.59 |
| **8395** | 8395 | Ord\_5411 | Prod\_6 | SHP\_7555 | Cust\_1798 | 127.1600 | 0.10 | 20 | -74.03 | 6.92 | 0.37 |
| **8396** | 8396 | Ord\_5388 | Prod\_6 | SHP\_7524 | Cust\_1798 | 243.0500 | 0.02 | 39 | -70.85 | 5.35 | 0.40 |
| **8397** | 8397 | Ord\_5348 | Prod\_15 | SHP\_7469 | Cust\_1798 | 3872.8700 | 0.03 | 23 | 565.34 | 30.00 | 0.62 |
| **8398** | 8398 | Ord\_5459 | Prod\_6 | SHP\_7628 | Cust\_1798 | 603.6900 | 0.00 | 47 | 131.39 | 4.86 | 0.38 |

8399 rows × 11 columns

In [30]:

market\_df.reset\_index(drop**=True**)

Out[30]:

|  | **Ord\_id** | **Prod\_id** | **Ship\_id** | **Cust\_id** | **Sales** | **Discount** | **Order\_Quantity** | **Profit** | **Shipping\_Cost** | **Product\_Base\_Margin** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Ord\_5446 | Prod\_16 | SHP\_7609 | Cust\_1818 | 136.8100 | 0.01 | 23 | -30.51 | 3.60 | 0.56 |
| **1** | Ord\_5406 | Prod\_13 | SHP\_7549 | Cust\_1818 | 42.2700 | 0.01 | 13 | 4.56 | 0.93 | 0.54 |
| **2** | Ord\_5446 | Prod\_4 | SHP\_7610 | Cust\_1818 | 4701.6900 | 0.00 | 26 | 1148.90 | 2.50 | 0.59 |
| **3** | Ord\_5456 | Prod\_6 | SHP\_7625 | Cust\_1818 | 2337.8900 | 0.09 | 43 | 729.34 | 14.30 | 0.37 |
| **4** | Ord\_5485 | Prod\_17 | SHP\_7664 | Cust\_1818 | 4233.1500 | 0.08 | 35 | 1219.87 | 26.30 | 0.38 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **8394** | Ord\_5353 | Prod\_4 | SHP\_7479 | Cust\_1798 | 2841.4395 | 0.08 | 28 | 374.63 | 7.69 | 0.59 |
| **8395** | Ord\_5411 | Prod\_6 | SHP\_7555 | Cust\_1798 | 127.1600 | 0.10 | 20 | -74.03 | 6.92 | 0.37 |
| **8396** | Ord\_5388 | Prod\_6 | SHP\_7524 | Cust\_1798 | 243.0500 | 0.02 | 39 | -70.85 | 5.35 | 0.40 |
| **8397** | Ord\_5348 | Prod\_15 | SHP\_7469 | Cust\_1798 | 3872.8700 | 0.03 | 23 | 565.34 | 30.00 | 0.62 |
| **8398** | Ord\_5459 | Prod\_6 | SHP\_7628 | Cust\_1798 | 603.6900 | 0.00 | 47 | 131.39 | 4.86 | 0.38 |

8399 rows × 10 columns

In [31]:

market\_df.head()

Out[31]:

|  | **Ord\_id** | **Prod\_id** | **Ship\_id** | **Cust\_id** | **Sales** | **Discount** | **Order\_Quantity** | **Profit** | **Shipping\_Cost** | **Product\_Base\_Margin** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Ord\_5446 | Prod\_16 | SHP\_7609 | Cust\_1818 | 136.81 | 0.01 | 23 | -30.51 | 3.60 | 0.56 |
| **1** | Ord\_5406 | Prod\_13 | SHP\_7549 | Cust\_1818 | 42.27 | 0.01 | 13 | 4.56 | 0.93 | 0.54 |
| **2** | Ord\_5446 | Prod\_4 | SHP\_7610 | Cust\_1818 | 4701.69 | 0.00 | 26 | 1148.90 | 2.50 | 0.59 |
| **3** | Ord\_5456 | Prod\_6 | SHP\_7625 | Cust\_1818 | 2337.89 | 0.09 | 43 | 729.34 | 14.30 | 0.37 |
| **4** | Ord\_5485 | Prod\_17 | SHP\_7664 | Cust\_1818 | 4233.15 | 0.08 | 35 | 1219.87 | 26.30 | 0.38 |

In [32]:

market\_df

Out[32]:

|  | **Ord\_id** | **Prod\_id** | **Ship\_id** | **Cust\_id** | **Sales** | **Discount** | **Order\_Quantity** | **Profit** | **Shipping\_Cost** | **Product\_Base\_Margin** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | Ord\_5446 | Prod\_16 | SHP\_7609 | Cust\_1818 | 136.8100 | 0.01 | 23 | -30.51 | 3.60 | 0.56 |
| **1** | Ord\_5406 | Prod\_13 | SHP\_7549 | Cust\_1818 | 42.2700 | 0.01 | 13 | 4.56 | 0.93 | 0.54 |
| **2** | Ord\_5446 | Prod\_4 | SHP\_7610 | Cust\_1818 | 4701.6900 | 0.00 | 26 | 1148.90 | 2.50 | 0.59 |
| **3** | Ord\_5456 | Prod\_6 | SHP\_7625 | Cust\_1818 | 2337.8900 | 0.09 | 43 | 729.34 | 14.30 | 0.37 |
| **4** | Ord\_5485 | Prod\_17 | SHP\_7664 | Cust\_1818 | 4233.1500 | 0.08 | 35 | 1219.87 | 26.30 | 0.38 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **8394** | Ord\_5353 | Prod\_4 | SHP\_7479 | Cust\_1798 | 2841.4395 | 0.08 | 28 | 374.63 | 7.69 | 0.59 |
| **8395** | Ord\_5411 | Prod\_6 | SHP\_7555 | Cust\_1798 | 127.1600 | 0.10 | 20 | -74.03 | 6.92 | 0.37 |
| **8396** | Ord\_5388 | Prod\_6 | SHP\_7524 | Cust\_1798 | 243.0500 | 0.02 | 39 | -70.85 | 5.35 | 0.40 |
| **8397** | Ord\_5348 | Prod\_15 | SHP\_7469 | Cust\_1798 | 3872.8700 | 0.03 | 23 | 565.34 | 30.00 | 0.62 |
| **8398** | Ord\_5459 | Prod\_6 | SHP\_7628 | Cust\_1798 | 603.6900 | 0.00 | 47 | 131.39 | 4.86 | 0.38 |

8399 rows × 10 columns

In [ ]:

​

market\_df.loc[market\_df['Sales'] **>** 3000]

​

Out[36]:

|  | **Ord\_id** | **Prod\_id** | **Ship\_id** | **Cust\_id** | **Sales** | **Discount** | **Order\_Quantity** | **Profit** | **Shipping\_Cost** | **Product\_Base\_Margin** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2** | Ord\_5446 | Prod\_4 | SHP\_7610 | Cust\_1818 | 4701.6900 | 0.00 | 26 | 1148.90 | 2.50 | 0.59 |
| **4** | Ord\_5485 | Prod\_17 | SHP\_7664 | Cust\_1818 | 4233.1500 | 0.08 | 35 | 1219.87 | 26.30 | 0.38 |
| **7** | Ord\_4725 | Prod\_4 | SHP\_6593 | Cust\_1641 | 3410.1575 | 0.10 | 48 | 1137.91 | 0.99 | 0.55 |
| **10** | Ord\_4743 | Prod\_2 | SHP\_6615 | Cust\_1641 | 4072.0100 | 0.01 | 43 | 1675.98 | 0.99 | 0.56 |
| **13** | Ord\_2207 | Prod\_11 | SHP\_3093 | Cust\_839 | 3364.2480 | 0.10 | 15 | -693.23 | 61.76 | 0.78 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **8366** | Ord\_3593 | Prod\_3 | SHP\_4974 | Cust\_1274 | 12073.0600 | 0.03 | 39 | 5081.87 | 19.99 | 0.38 |
| **8367** | Ord\_3593 | Prod\_15 | SHP\_4975 | Cust\_1274 | 6685.0500 | 0.09 | 25 | 1653.60 | 24.49 | NaN |
| **8371** | Ord\_2624 | Prod\_4 | SHP\_3591 | Cust\_1006 | 4924.1350 | 0.07 | 28 | 1049.54 | 8.99 | 0.58 |
| **8383** | Ord\_2722 | Prod\_1 | SHP\_3731 | Cust\_1006 | 3508.3300 | 0.04 | 21 | -546.98 | 35.00 | 0.85 |
| **8397** | Ord\_5348 | Prod\_15 | SHP\_7469 | Cust\_1798 | 3872.8700 | 0.03 | 23 | 565.34 | 30.00 | 0.62 |

1359 rows × 10 columns

In [37]:

market\_df.loc[(market\_df['Sales'] **>** 2000) **&** (market\_df['Sales'] **<** 3000) **&** (market\_df['Profit'] **>** 100), :]

​

Out[37]:

|  | **Ord\_id** | **Prod\_id** | **Ship\_id** | **Cust\_id** | **Sales** | **Discount** | **Order\_Quantity** | **Profit** | **Shipping\_Cost** | **Product\_Base\_Margin** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **3** | Ord\_5456 | Prod\_6 | SHP\_7625 | Cust\_1818 | 2337.8900 | 0.09 | 43 | 729.34 | 14.30 | 0.37 |
| **81** | Ord\_5205 | Prod\_4 | SHP\_7274 | Cust\_1749 | 2546.5235 | 0.09 | 26 | 210.00 | 7.69 | 0.59 |
| **109** | Ord\_139 | Prod\_17 | SHP\_186 | Cust\_45 | 2671.2100 | 0.06 | 14 | 636.18 | 15.59 | 0.36 |
| **110** | Ord\_239 | Prod\_4 | SHP\_332 | Cust\_45 | 2157.3085 | 0.00 | 38 | 519.25 | 5.31 | 0.57 |
| **141** | Ord\_1673 | Prod\_17 | SHP\_2314 | Cust\_498 | 2027.5500 | 0.04 | 14 | 537.40 | 13.99 | 0.37 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **8338** | Ord\_2107 | Prod\_2 | SHP\_2882 | Cust\_785 | 2409.9600 | 0.07 | 32 | 575.10 | 4.50 | 0.59 |
| **8350** | Ord\_3570 | Prod\_4 | SHP\_4942 | Cust\_1266 | 2094.9780 | 0.06 | 44 | 697.29 | 1.25 | 0.55 |
| **8354** | Ord\_3592 | Prod\_4 | SHP\_4973 | Cust\_1266 | 2614.3705 | 0.07 | 25 | 384.01 | 7.69 | 0.58 |
| **8381** | Ord\_2696 | Prod\_4 | SHP\_3691 | Cust\_1006 | 2836.0505 | 0.01 | 25 | 561.13 | 8.99 | 0.59 |
| **8394** | Ord\_5353 | Prod\_4 | SHP\_7479 | Cust\_1798 | 2841.4395 | 0.08 | 28 | 374.63 | 7.69 | 0.59 |

328 rows × 10 columns

In [38]:

market\_df.loc[(market\_df['Sales'] **>** 2000) **|** (market\_df['Profit'] **>** 100), :]

​

Out[38]:

|  | **Ord\_id** | **Prod\_id** | **Ship\_id** | **Cust\_id** | **Sales** | **Discount** | **Order\_Quantity** | **Profit** | **Shipping\_Cost** | **Product\_Base\_Margin** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **2** | Ord\_5446 | Prod\_4 | SHP\_7610 | Cust\_1818 | 4701.6900 | 0.00 | 26 | 1148.90 | 2.50 | 0.59 |
| **3** | Ord\_5456 | Prod\_6 | SHP\_7625 | Cust\_1818 | 2337.8900 | 0.09 | 43 | 729.34 | 14.30 | 0.37 |
| **4** | Ord\_5485 | Prod\_17 | SHP\_7664 | Cust\_1818 | 4233.1500 | 0.08 | 35 | 1219.87 | 26.30 | 0.38 |
| **7** | Ord\_4725 | Prod\_4 | SHP\_6593 | Cust\_1641 | 3410.1575 | 0.10 | 48 | 1137.91 | 0.99 | 0.55 |
| **10** | Ord\_4743 | Prod\_2 | SHP\_6615 | Cust\_1641 | 4072.0100 | 0.01 | 43 | 1675.98 | 0.99 | 0.56 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **8383** | Ord\_2722 | Prod\_1 | SHP\_3731 | Cust\_1006 | 3508.3300 | 0.04 | 21 | -546.98 | 35.00 | 0.85 |
| **8385** | Ord\_1833 | Prod\_3 | SHP\_2527 | Cust\_637 | 611.1600 | 0.04 | 46 | 100.22 | 4.98 | 0.40 |
| **8394** | Ord\_5353 | Prod\_4 | SHP\_7479 | Cust\_1798 | 2841.4395 | 0.08 | 28 | 374.63 | 7.69 | 0.59 |
| **8397** | Ord\_5348 | Prod\_15 | SHP\_7469 | Cust\_1798 | 3872.8700 | 0.03 | 23 | 565.34 | 30.00 | 0.62 |
| **8398** | Ord\_5459 | Prod\_6 | SHP\_7628 | Cust\_1798 | 603.6900 | 0.00 | 47 | 131.39 | 4.86 | 0.38 |

3009 rows × 10 columns

**import** pandas **as** pd

**import** numpy **as** np

s**=**pd.Series([2,3,4,5,6])

print(s)

print(type(s))

0 2

1 3

2 4

3 5

4 6

dtype: int64

<class 'pandas.core.series.Series'>

In [7]:

char\_series**=**pd.Series(['a','b','af'])

char\_series

​

Out[7]:

0 a

1 b

2 af

dtype: object

In [10]:

data\_series**=**pd.date\_range(start**=**'19-02-2024',end**=**'25-02-2024')

data\_series

​

Out[10]:

DatetimeIndex(['2024-02-19', '2024-02-20', '2024-02-21', '2024-02-22',

'2024-02-23', '2024-02-24', '2024-02-25'],

dtype='datetime64[ns]', freq='D')

In [11]:

type(data\_series)

Out[11]:

pandas.core.indexes.datetimes.DatetimeIndex

In [12]:

s[3]

Out[12]:

5

In [13]:

s[2:]

Out[13]:

2 4

3 5

4 6

dtype: int64

In [15]:

s[[1,3]]

Out[15]:

1 3

3 5

dtype: int64

In [21]:

pd.Series([0,1,2],index **=** ['a','b','c'])

Out[21]:

a 0

b 1

c 2

dtype: int64

In [25]:

pd.Series(np.array(range(0,10))**\*\***2,index**=**range(0,10))

​

Out[25]:

0 0

1 1

2 4

3 9

4 16

5 25

6 36

7 49

8 64

9 81

dtype: int32

In [1]:

df.set\_index('id',inplace**=True**)

df

Out[56]:

|  | **name** | **age** | **occ** |
| --- | --- | --- | --- |
| **id** |  |  |  |
| **1** | vin | 22 | doc |
| **2** | tim | 20 | eng |
| **3** | abc | 19 | doc |

MATPLOTLIP AND PYTHON

**import** matplotlib.pyplot **as** plt

​

*Simple plot*

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

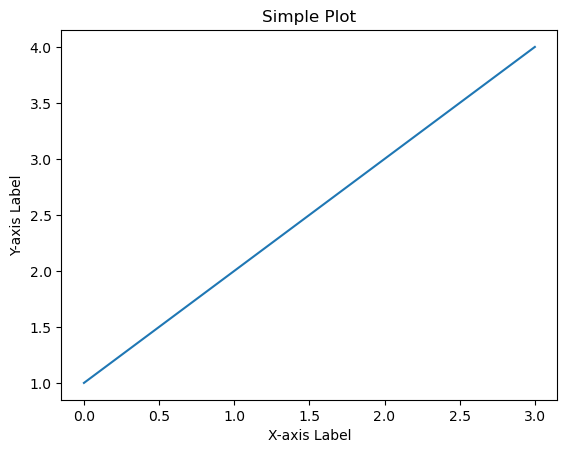
plt.ylabel('Y-axis Label')

plt.xlabel('X-axis Label')

plt.title('Simple Plot')

plt.show()

​



In [25]:

**import** matplotlib.pyplot **as** plt

​

*Plot with formatting*

plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')

plt.axis([0, 6, 0, 20])

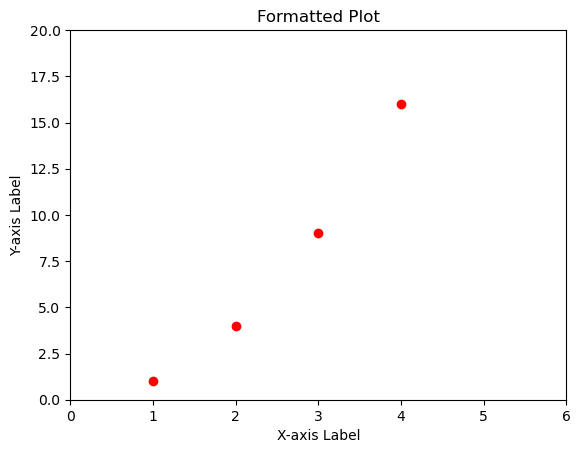
plt.xlabel('X-axis Label')

plt.ylabel('Y-axis Label')

plt.title('Formatted Plot')

plt.show()

​



In [26]:

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

​

*Plotting multiple lines*

t **=** np.arange(0., 5., 0.2)

plt.plot(t, t, 'r--', label**=**'Line 1')

plt.plot(t, t**\*\***2, 'bs', label**=**'Line 2')

plt.plot(t, t**\*\***3, 'g^', label**=**'Line 3')

plt.legend()

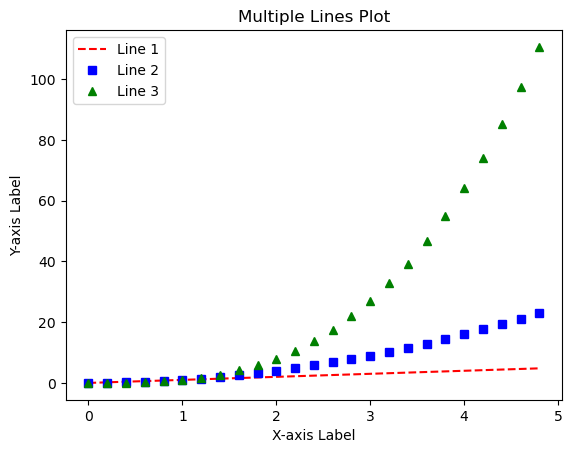
plt.xlabel('X-axis Label')

plt.ylabel('Y-axis Label')

plt.title('Multiple Lines Plot')

plt.show()

​



In [28]:

**import** matplotlib.pyplot **as** plt

​

*Customizing line properties*

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

y **=** [10, 20, 25, 30]

​

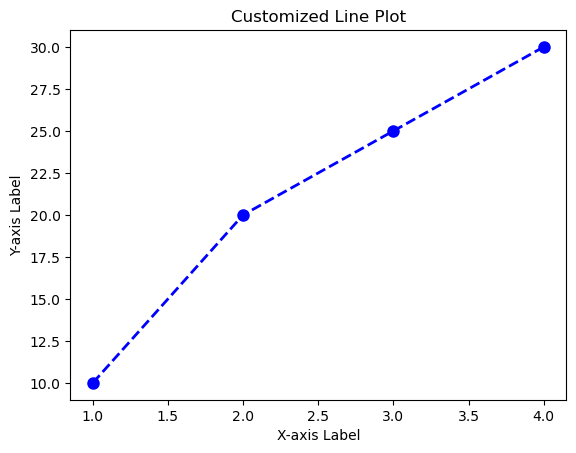
plt.plot(x, y, linewidth**=**2.0, color**=**'blue', linestyle**=**'--', marker**=**'o', markersize**=**8)

plt.xlabel('X-axis Label')

plt.ylabel('Y-axis Label')

plt.title('Customized Line Plot')

plt.show()

​

**import** matplotlib.pyplot **as** plt

​

**def** categorical\_plot(names, values):

fig, axs **=** plt.subplots(1, 3, figsize**=**(9, 3))

​

axs[0].bar(names, values)

axs[0].set\_title('Bar Plot')

​

axs[1].scatter(names, values)

axs[1].set\_title('')

​

axs[2].plot(names, values)

axs[2].set\_title('Line Plot')

​

fig.suptitle('Categorical Plotting')

plt.show()

​

​

names **=** ['group\_a', 'group\_b', 'group\_c']

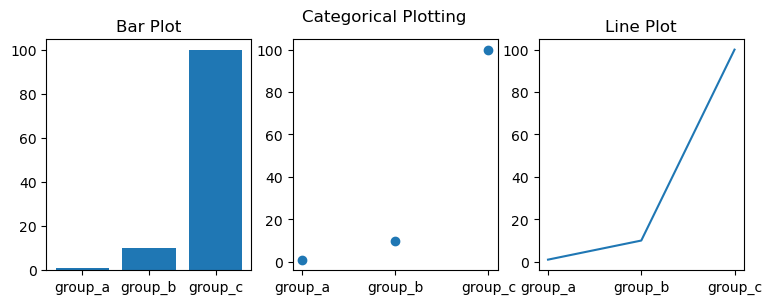
values **=** [1, 10, 100]

​

​

categorical\_plot(names, values)

​



In [29]:

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

​

*Working with multiple figures and axes*

**def** f(t):

**return** np.exp(**-**t) **\*** np.cos(2**\***np.pi**\***t)

​

t1 **=** np.arange(0.0, 5.0, 0.1)

t2 **=** np.arange(0.0, 5.0, 0.02)

​

plt.figure()

plt.subplot(211)

plt.plot(t1, f(t1), 'bo', t2, f(t2), 'k')

plt.title('Subplot 1')

​

plt.subplot(212)

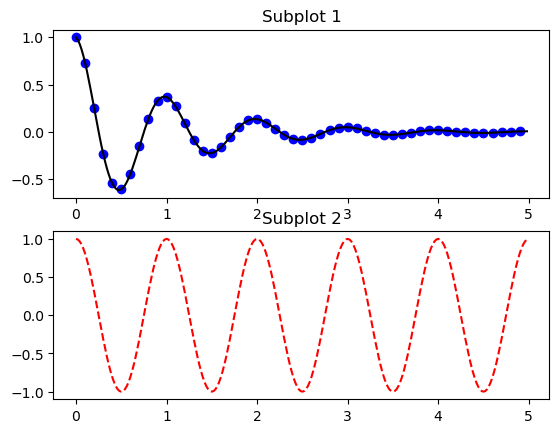
plt.plot(t2, np.cos(2**\***np.pi**\***t2), 'r--')

plt.title('Subplot 2')

​

plt.show()

​



In [31]:

**import** matplotlib.pyplot **as** plt

​

*Working with text*

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

plt.ylabel('Y-axis Label')

plt.xlabel('X-axis Label')

plt.title('Plot with Text')

​

plt.text(2, 3, 'Sample Text', fontsize**=**12, color**=**'red')

plt.show()

​

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

​

**def** plot\_scatter():

a\_values **=** np.arange(50)

c\_values **=** np.random.randint(0, 50, 50)

d\_values **=** np.random.randn(50)

b\_values **=** a\_values **+** 10 **\*** np.random.randn(50)

d\_values\_abs **=** np.abs(d\_values) **\*** 100

​

plt.scatter(a\_values, b\_values, c**=**c\_values, s**=**d\_values\_abs)

plt.xlabel('entry a')

plt.ylabel('entry b')

plt.show()

​

plot\_scatter()

​

