JSPM UNIVERSITY PUNE

Faculty of Business Management and Commerce

School of Business Management

Introduction to Data Science (Code:230GCSM28_01) Lab Manual

MBA

Faculty: Mrs. Manisha Patil

JSPM UNIVERSITY PUNE

Faculty of Business Management and Commerce

JSPM UNIVERSITY

न हि ज्ञानेन सदृशं

School of Business management

Name: ____

Roll No: ____

Subject: ____

Faculty: ____

JSPM University Pune

F.Y. MBA

Course Type: SEC	**Lab Course Title: Introduction to Data Science**	
Course Code: 230GCAB0101	**Teaching Scheme:**	**Examination Scheme:**
Credits: 1	**Lecture (L): -1** **Tutorial (T): -** **Practical (P):** 2 **Experiential Learning (EL): -**	**Practical:**50 Marks **Oral: -**50 Marks

List of Laboratory Experiments

1.	**Installation of IDLE to work with python libraries.**
2.	**Introduction to python libraries for Data Science**
3.	**Draw a linear graph using Mat plot lib library from Python.**
4.	**Draw a bar graph using Mat plot lib. Write a code to change attributes width and Color of Bar graph.**
5.	**Draw a Pie chart using matplotlib.**
6.	**Understanding Array creation routines using NumPy.**
7.	**Develop a code to understand array indexing and slicing.**
8.	**Develop code to understand following operations on matrices.** **(i) Addition of two matrices (ii) Subtraction of two matrices (iii)multiplication of two matrices (iv)division of matrix elements (v)Dot product of two matrices**
9.	**Create Data Frame using Pandas and print number of columns and Number of Rows.**
10.	**Write code to Extract and Read Data with Pandas .csv file and .xls file.**
11.	**Design a desktop to display bar graph, line graph and pie chart for given data set using power BI.**

Virtual LAB Links:

Lab Name: Introduction to Data Science Lab

Link of the Virtual Lab: https://www.iiitmk.ac.in/DAVirtalLab/#work

Sr.No.	Name Of the Practical/Experiments	Date Of Practical	Signature
1	***EXPERIMENT NO.: 01***		
2	***EXPERIMENT NO.: 02***		
3	***EXPERIMENT NO.: 03***		
4	***EXPERIMENT NO.: 04***		
5	***EXPERIMENT NO.: 05***		
6	***EXPERIMENT NO.: 06***		
7	***EXPERIMENT NO.: 07***		
8	***EXPERIMENT NO.: 08***		
9	***EXPERIMENT NO.: 09***		
10	***EXPERIMENT NO.: 10***		
11	***EXPERIMENT NO.: 11***		

Assignment No:1

Installation of IDLE to work with python libraries

Theory:

IDLE Python Download:

To download IDLE (Integrated Development and Learning Environment) for Python, you can follow these steps:

1. Open your web browser and go to the official Python website at https://www.python.org/.

2. Click on the “Downloads” tab located in the navigation bar.

3. Scroll down to the Python releases section and locate the version you want to download. Select the version appropriate for your operating system (Windows, macOS, or Linux).

4. After selecting your operating system, you’ll see a list of files available for download. Look for the installer that matches your system architecture (32-bit or 64-bit) and click on the link to start the download.

5. Once the download is complete, locate the downloaded file and run the installer.

6. Follow the installation wizard’s instructions to install Python on your system. Make sure to check the box that says “Install IDLE” or “Add Python to PATH” during the installation process, depending on the options presented.

7. After the installation is complete, you can launch IDLE by searching for “IDLE” in your computer’s search bar or by locating it in the Python installation directory.

Assignment No.2

Introduction to python libraries for Data Science

Theory:

Python is a general purpose language and is often used for things other than data analysis and data science.

NumPy:

NumPy stands for Numerical Python. The most powerful feature of NumPy is n-dimensional array. This library also contains basic linear algebra functions, Fourier transforms, advanced random number capabilities and tools for integration with other low level languages like Fortran, C and C++. NumPy was created in 2005 by Travis Oliphant.

NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.

NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.This behavior is called locality of reference in computer science.

Pandas:

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real-world** data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open source data analysis/manipulation tool available in any language**. It is already well on its way toward this goal.

Pandas for structured data operations and manipulations. It is extensively used for data munging and preparation. Pandas were added relatively recently to Python and have been instrumental in boosting

Python's usage in data scientist community.

The two primary data structures of pandas, **Series** (1-dimensional) and **DataFrame** (2-dimensional), handle the vast majority of typical use cases in finance, statistics, social science, and many areas of engineering. For R users, **DataFrame** provides everything that R's data.frame provides and much more. pandas is built on top of NumPy and is intended to integrate well within a scientific computing environment with many other 3rd party libraries.

Matplotlib:

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

Matplotlib was created by John D. Hunter.

Matplotlib is an amazing visualization library in **Python** for 2D plots of arrays.

Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram, etc.

Types of Matplotlib

Matplotlib comes with a wide variety of plots. Plots help to understand trends, and patterns, and to make correlations. They're typically instruments for reasoning about quantitative information. Some of the sample plots are covered here.

- Matplotlib Line Plot
- Matplotlib Bar Plot
- Matplotlib Histograms Plot
- Matplotlib Scatter Plot
- Matplotlib Pie Charts
- Matplotlib Area Plot

Assignment No3.

Draw a linear graph using Mat plot lib library from Python and demonstrate different functions of Mat Plot Lib

Aim: To Understand the Python library for plotting graphs Matplotlib

Matplot lib:

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

Installation of Matplotlib

If you have Python and PIP already installed on a system, then installation of Matplotlib is very easy.

Install it using this command:

```
C:\Users\Your Name>pip install matplotlib.
```

If this command fails, then use a python distribution that already has Matplotlib installed, like Anaconda, Spyder etc.

Import Matplotlib

Once Matplotlib is installed, import it in your applications by adding
he import *module* statement:

```
import matplotlib
```

Pyplot

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

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

Now the Pyplot package can be referred to as plt.

Draw a line in a diagram

```
import matplotlib.pyplot as plt
x=[1,2,3,4]
y=[1,2,3,4]
```

```
plt.plot(x,y)
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
plt.show()
```

Add Grid Lines to a Plot

With Pyplot, you can use the grid() function to add grid lines to the plot.

```
plt.grid()
```

Example

Display only grid lines for the x-axis:

```
plt.grid(axis = 'x')
```

Example

Display only grid lines for the y-axis:

```
plt.grid(axis = 'y')
```

Matplotlib Subplot

Display Multiple Plots

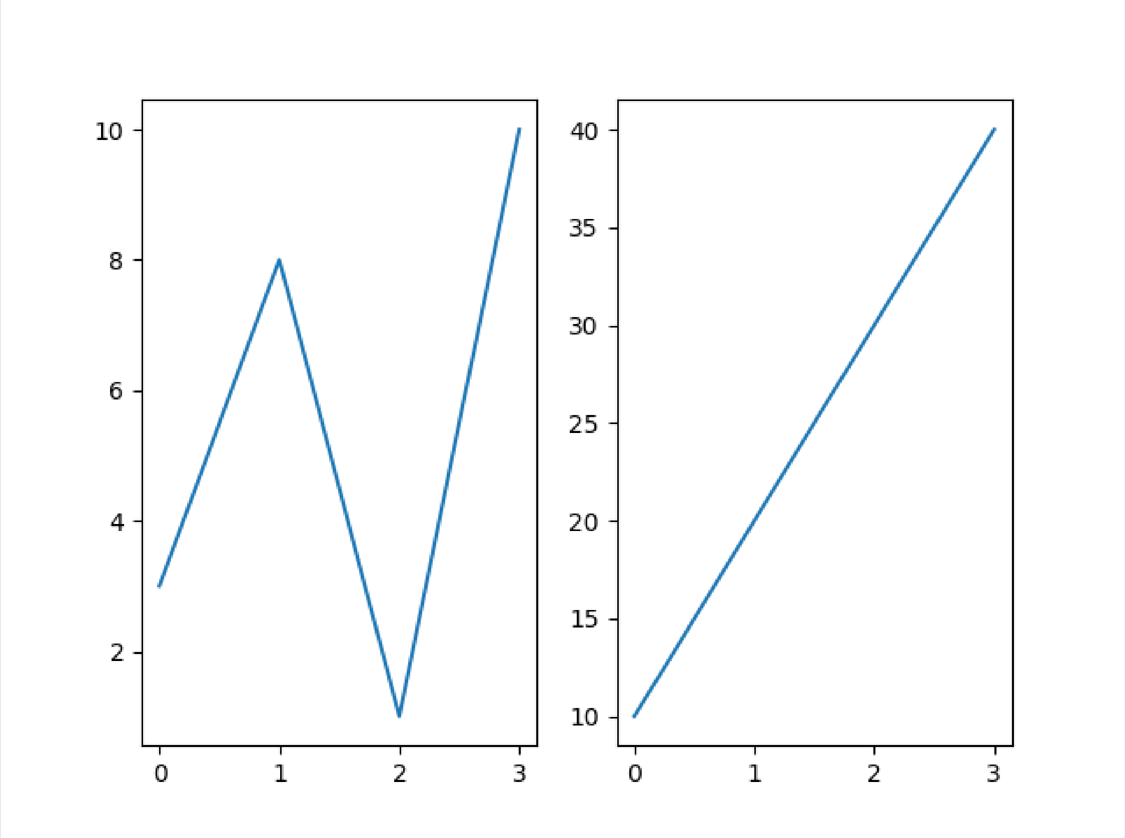
With the subplot() function you can draw multiple plots in one figure:

Example

Draw 2 plots:

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

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(1, 2, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)
plt.plot(x,y)

plt.show()
```

Result:



The subplot() Function

The subplot() function takes three arguments that describes the layout of the figure.

The layout is organized in rows and columns, which are represented by the *first* and *second* argument.

The third argument represents the index of the current plot.

```
plt.subplot(1, 2, 1)
#the figure has 1 row, 2 columns, and this plot is the first plot.
```

```
plt.subplot(1, 2, 2)
#the figure has 1 row, 2 columns, and this plot is the second plot.
```

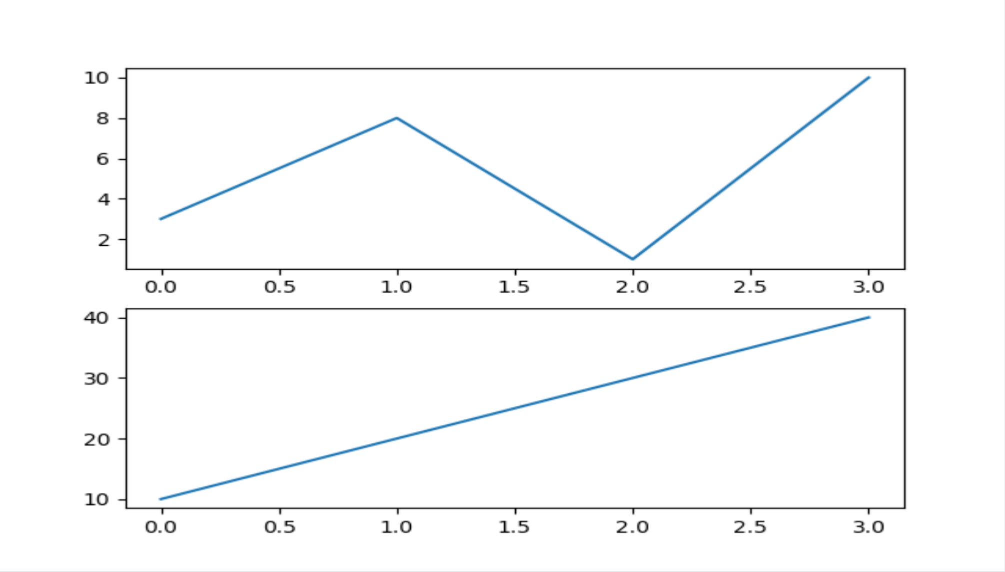
So, if we want a figure with 2 rows an 1 column (meaning that the two plots will be displayed on top of each other instead of side-by-side), we can write the syntax like this:

Example

Draw 2 plots on top of each other:

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

#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 1, 1)
plt.plot(x,y)

#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 1, 2)
plt.plot(x,y)

plt.show()
```

Result:



You can draw as many plots you like on one figure, just descibe the number of rows, columns, and the index of the plot.

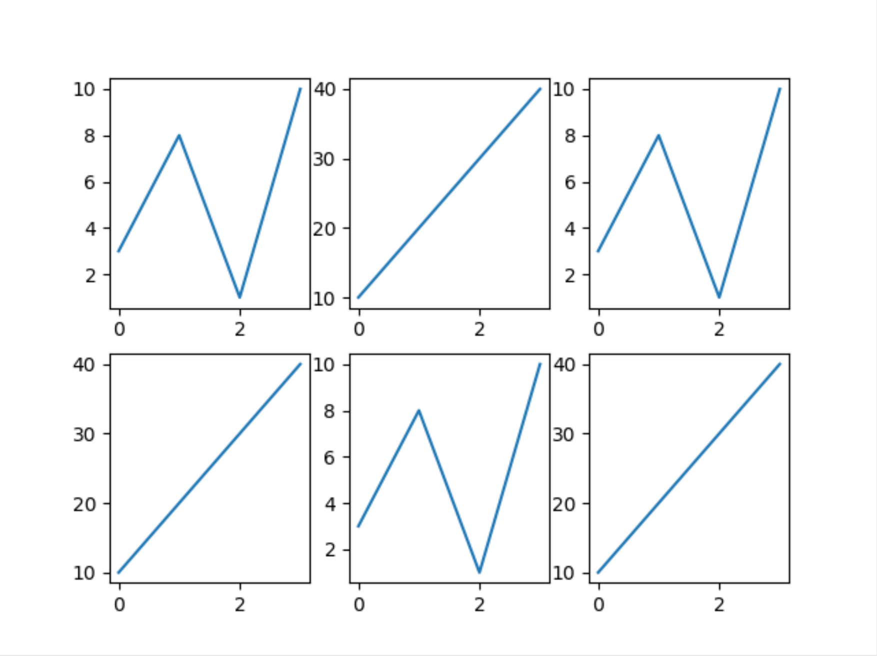
Example

Draw 6 plots:

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

```
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 1)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 2)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 3)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 4)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])

plt.subplot(2, 3, 5)
plt.plot(x,y)

x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])

plt.subplot(2, 3, 6)
plt.plot(x,y)

plt.show()
```

Result:



Title

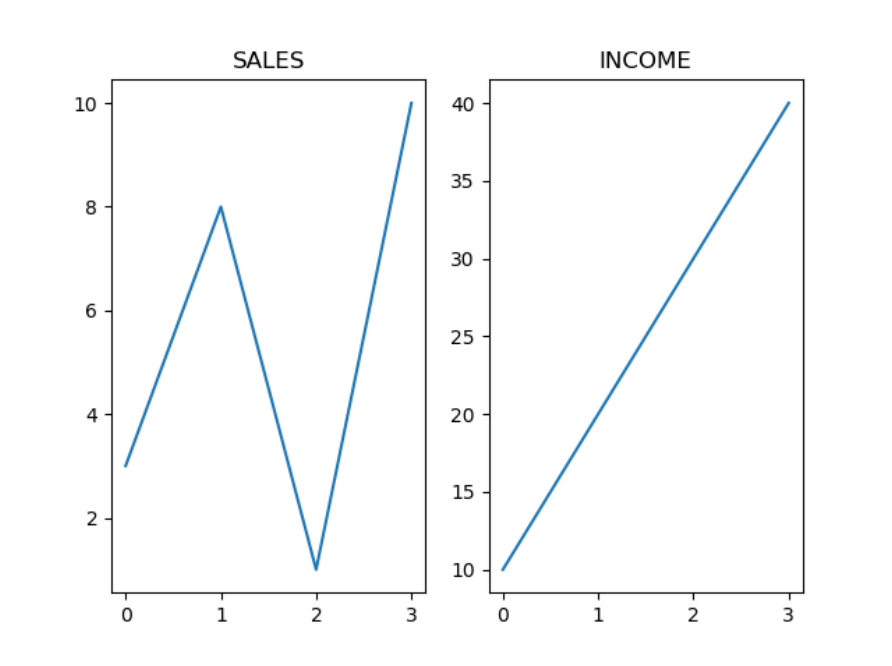
You can add a title to each plot with the title() function:

Example

2 plots, with titles:

```
import matplotlib.pyplot as plt
import numpy as np
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")
#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")
plt.show()
```

Result:



Super Title

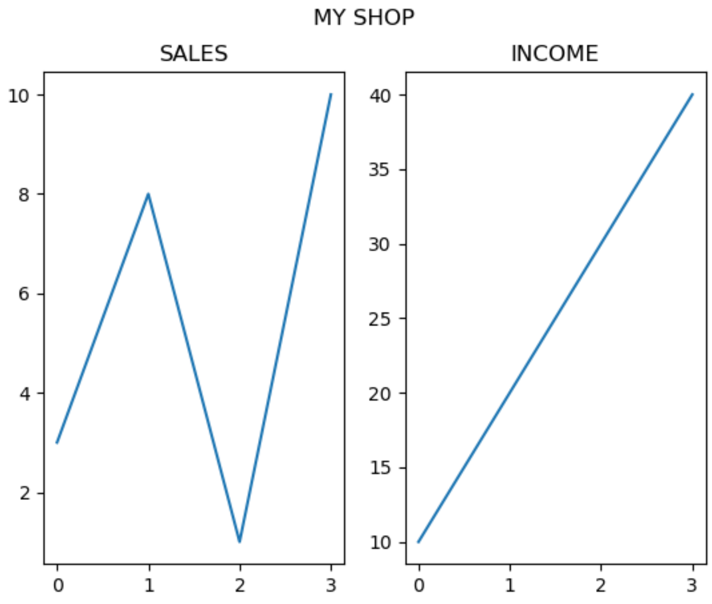
You can add a title to the entire figure with the suptitle() function:

Example

Add a title for the entire figure:

```
import matplotlib.pyplot as plt
import numpy as np
#plot 1:
x = np.array([0, 1, 2, 3])
y = np.array([3, 8, 1, 10])
plt.subplot(1, 2, 1)
plt.plot(x,y)
plt.title("SALES")
#plot 2:
x = np.array([0, 1, 2, 3])
y = np.array([10, 20, 30, 40])
plt.subplot(1, 2, 2)
plt.plot(x,y)
plt.title("INCOME")
plt.suptitle("MY SHOP")
plt.show()
```

Result:



Assignment No.4

Draw a bar graph using Matplot lib. Write a code to change attributes width and Color of Bar graph

Aim: To Understand the methods to draw and modify bargraphs using MatplotLib library in Python

Theory:

Matplotlib Bars

Creating Bars

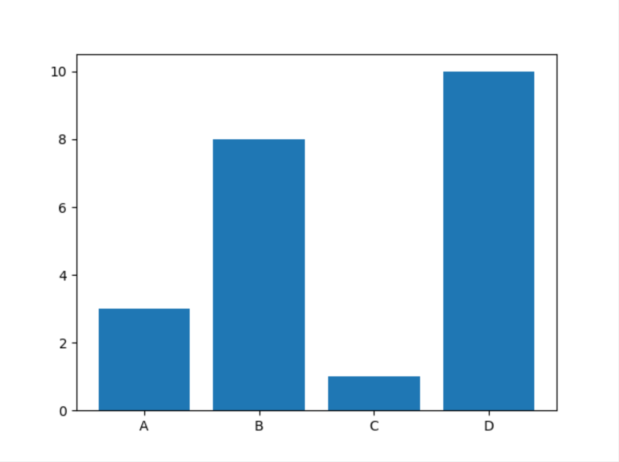
With Pyplot, you can use the bar() function to draw bar graphs:

Example

Draw 4 bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()
```

Result:



The bar() function takes arguments that describes the layout of the bars.

The categories and their values represented by the *first* and *second* argument as arrays.

Example

```
x = ["APPLES", "BANANAS"]
y = [400, 350]
plt.bar(x, y)
```

Horizontal Bars

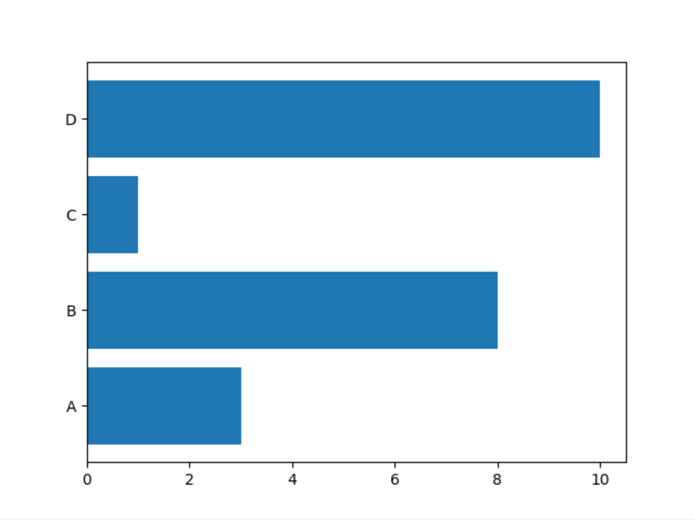
If you want the bars to be displayed horizontally instead of vertically, use the barh() function:

Example

Draw 4 horizontal bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y)
plt.show()
```

Result:



Bar Color

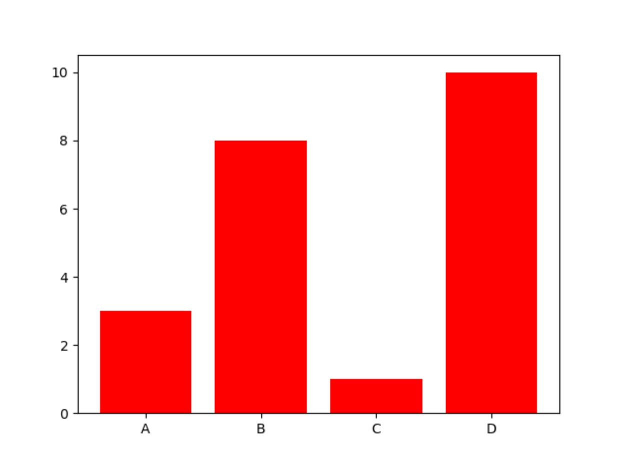
The bar() and barh() take the keyword argument color to set the color of the bars:

Example

Draw 4 red bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "red")
plt.show()
```

Result:



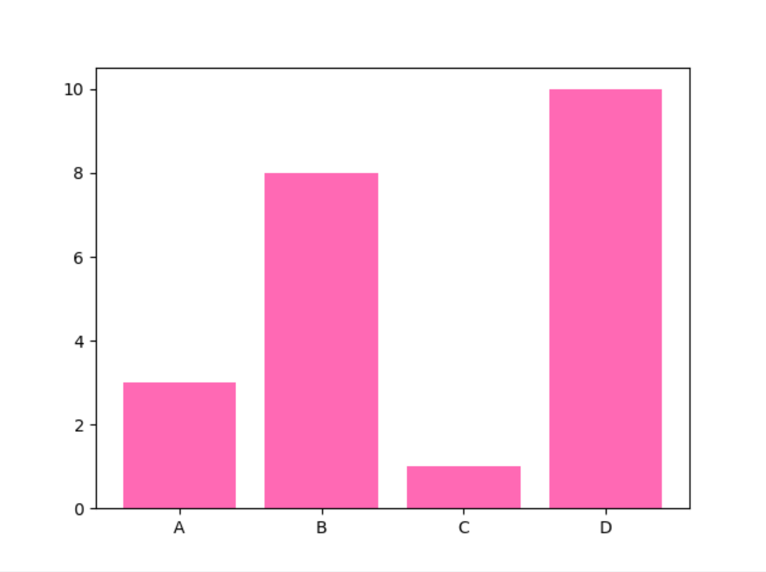
Color Names You can use any of the 140 supported color names.

Example

Draw 4 "hot pink" bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "hotpink")
plt.show()
```

Result:



Color Hex

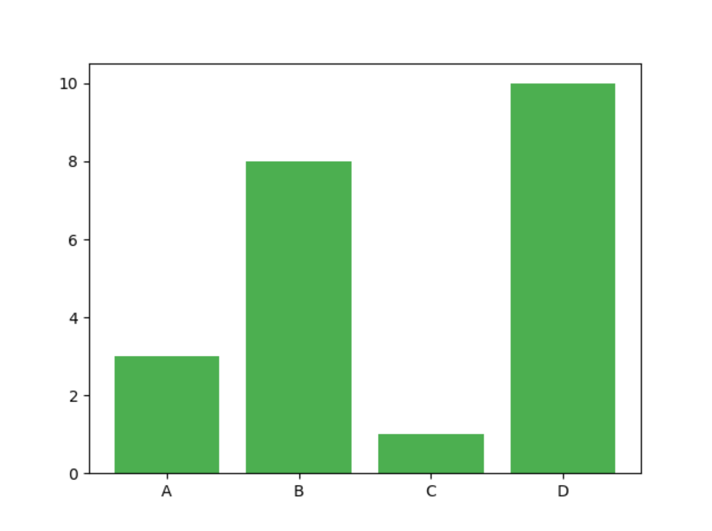
Or you can use Hexadecimal color values:

Example

Draw 4 bars with a beautiful green color:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, color = "#4CAF50")
plt.show()
```

Result:



Bar Width

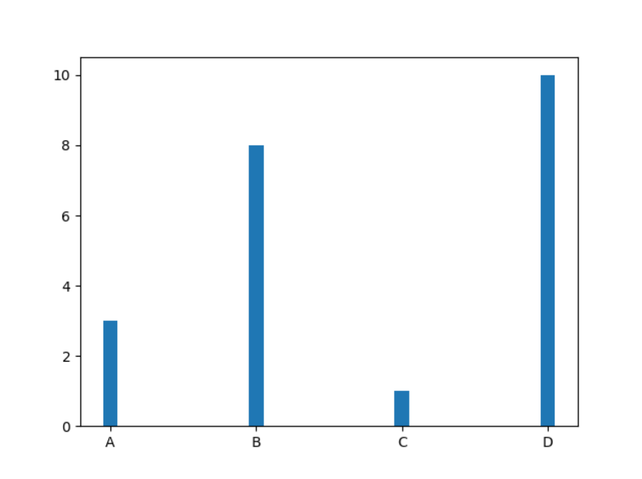
The bar() takes the keyword argument width to set the width of the bars:

Example

Draw 4 very thin bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x, y, width = 0.1)
plt.show()
```

Result:



The default width value is 0.8

Note: For horizontal bars, use height instead of width.

Bar Height

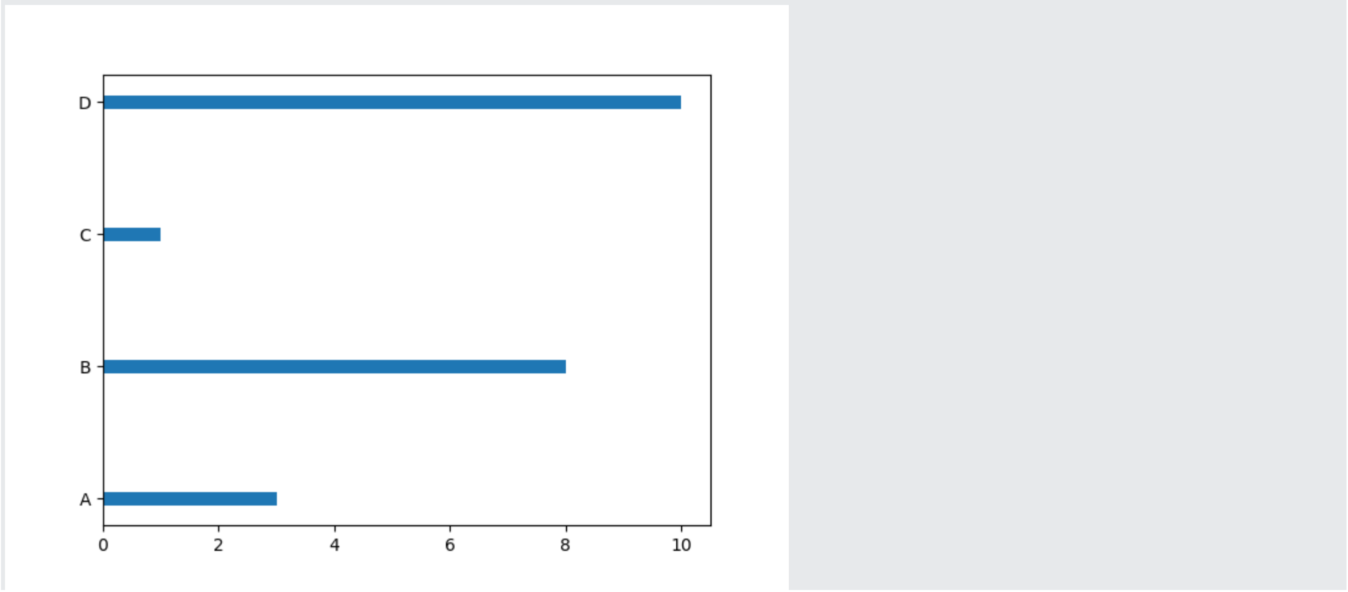
The barh() takes the keyword argument height to set the height of the bars:

Example

Draw 4 very thin bars:

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

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.barh(x, y, height = 0.1)
plt.show()
```

Result:


D
C
B
A
0
2
4
6
8
10

Assignment No.5:

Draw a Pie chart using matplotlib.

Aim: To find largest of three numbers using If else ladder concept of C Programming

Matplotlib Pie Charts

Creating Pie Charts

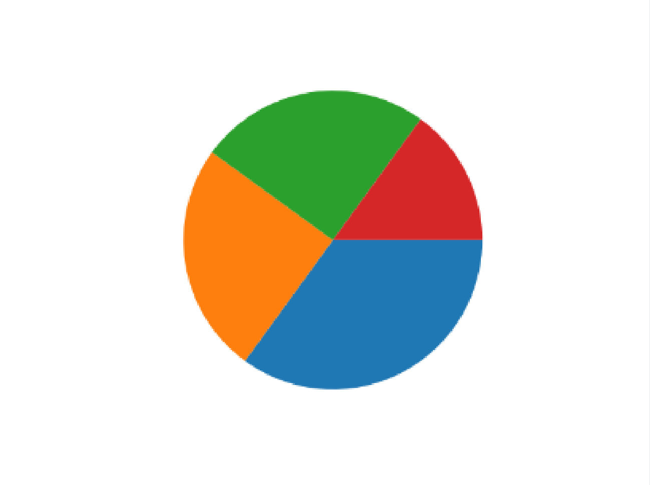
With Pyplot, you can use the pie() function to draw pie charts:

Example

A simple pie chart:

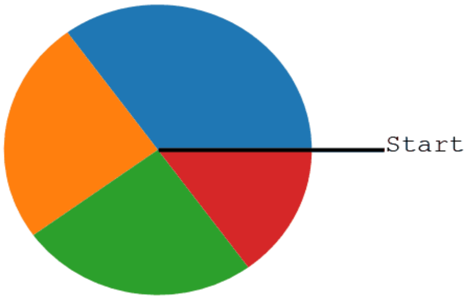
```
import matplotlib.pyplot as plt
import numpy as np

y = np.array([35, 25, 25, 15])

plt.pie(y)
plt.show()
```

Result:



As you can see the pie chart draws one piece (called a wedge) for each value in the array (in this case [35, 25, 25, 15]).

By default the plotting of the first wedge starts from the x-axis and moves *counterclockwise*:



Note: The size of each wedge is determined by comparing the value with all the other values, by using this formula:

The value divided by the sum of all values: x/sum(x)

Labels

Add labels to the pie chart with the label parameter.

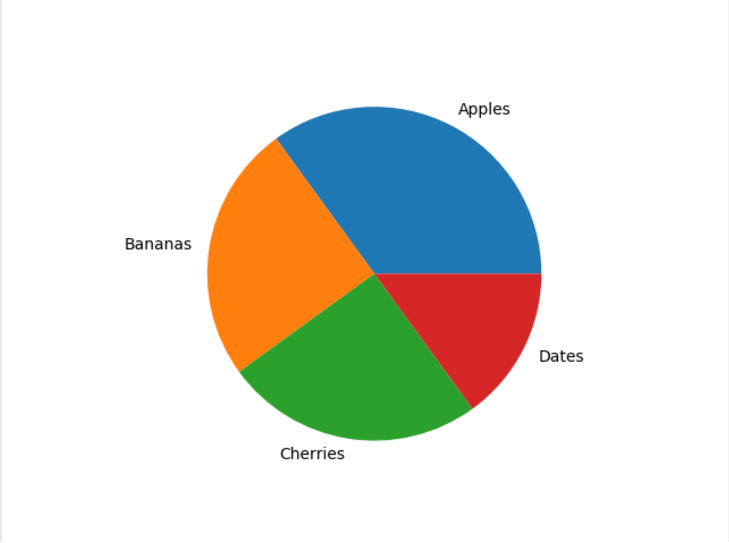
The label parameter must be an array with one label for each wedge:

Example

A simple pie chart:

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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels)
plt.show()
```

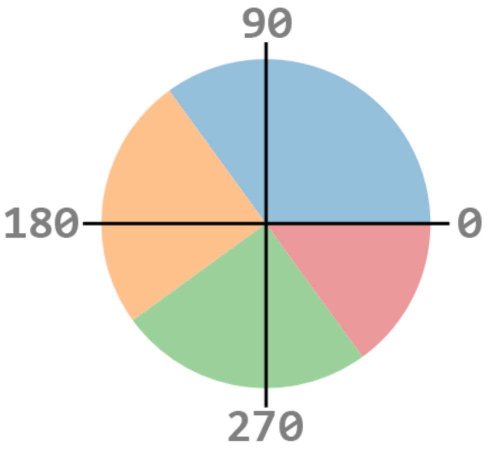
Result:



Start Angle

As mentioned the default start angle is at the x-axis, but you can change the start angle by specifying a startangle parameter.

The startangle parameter is defined with an angle in degrees, default angle is 0:



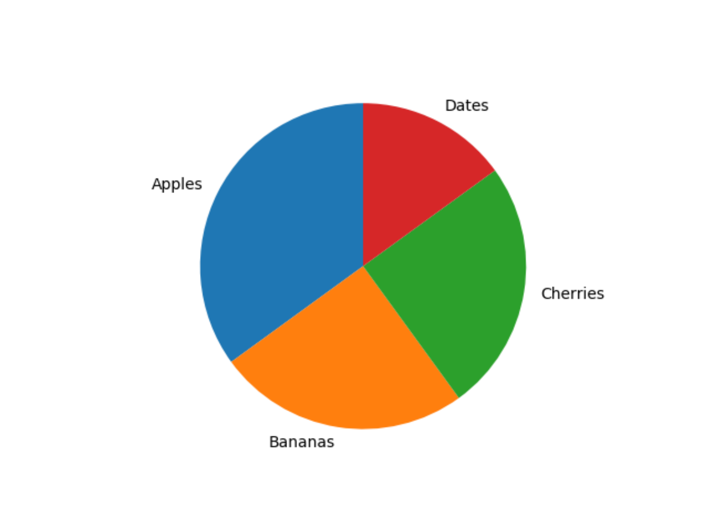
Example

Start the first wedge at 90 degrees:

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

y = np.array([35, 25, 25, 15])
```

```
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels, startangle = 90)
plt.show()
```

Result:



Explode

Maybe you want one of the wedges to stand out? The explode parameter allows you to do that.

The explode parameter, if specified, and not None, must be an array with one value for each wedge.

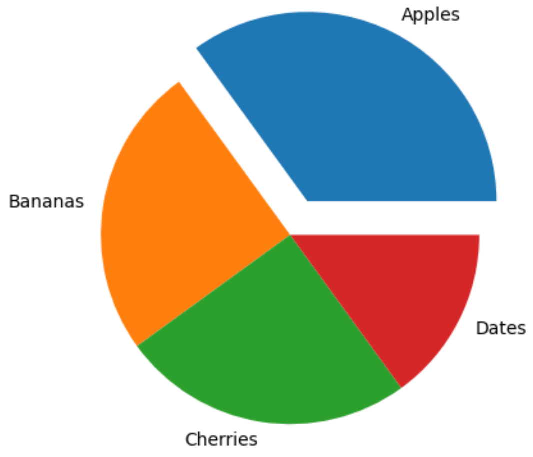
Each value represents how far from the center each wedge is displayed:

Example

Pull the "Apples" wedge 0.2 from the center of the pie:

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

y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
myexplode = [0.2, 0, 0, 0]

plt.pie(y, labels = mylabels, explode = myexplode)
plt.show()
```

Result:



Assignment No.6

Understanding Array creation routines using NumPy.

Aim: To understand creation of arrays using Numpy Library ndarray object.

Program:

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

NumPy stands for Numerical Python.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

NumPy Creating Arrays

Create a NumPy ndarray Object

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

We can create a NumPy ndarray object by using the array() function.

Example

```
import numpy as np

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

print(arr)

print(type(arr))
```

type(): This built-in Python function tells us the type of the object passed to it. Like in above code it shows that arr is numpy.ndarray type.

Dimensions in Arrays

A dimension in arrays is one level of array depth (nested arrays).

nested array: are arrays that have arrays as their elements.

0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

Example

Create a 0-D array with value 42

```
import numpy as np

arr = np.array(42)

print(arr)
```

1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

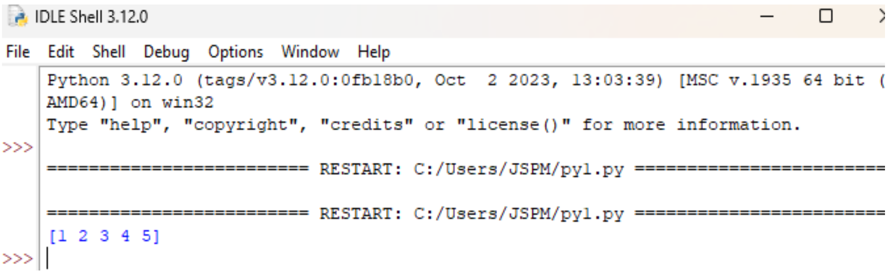
These are the most common and basic arrays.

Example

Create a 1-D array containing the values 1,2,3,4,5:

```
import numpy as np

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

print(arr)
```

Output:



```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======================== RESTART: C:/Users/JSPM/py1.py ========================

======================== RESTART: C:/Users/JSPM/py1.py ========================
[1 2 3 4 5]
>>> |
```

2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array.

These are often used to represent matrix or 2nd order tensors.

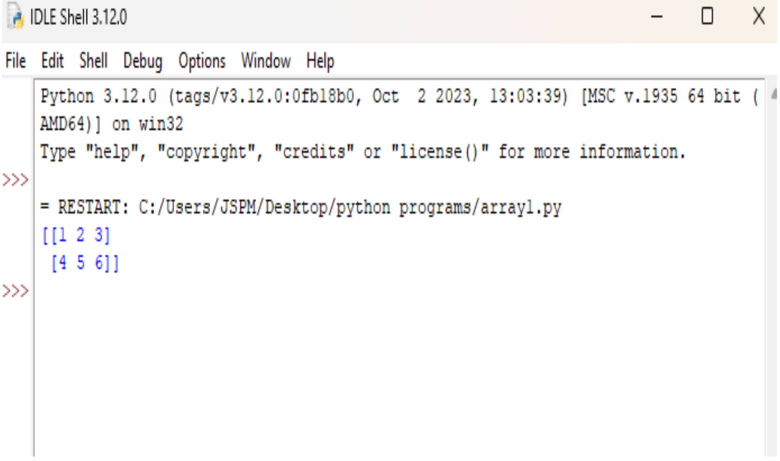
NumPy has a whole sub module dedicated towards matrix operations called numpy.mat

Example

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

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

print(arr)
```

Output:



```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/JSPM/Desktop/python programs/array1.py
[[1 2 3]
 [4 5 6]]
>>>
```

3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

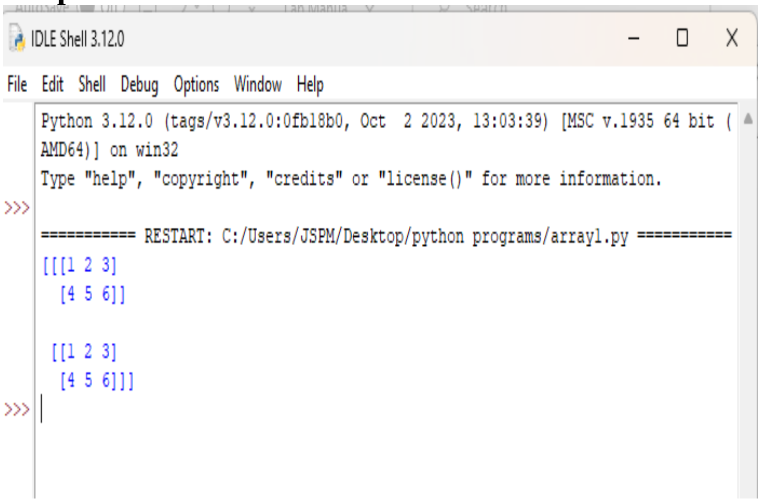
These are often used to represent a 3rd order tensor.

Example

Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

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

print(arr)
```

Output:



```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=========== RESTART: C:/Users/JSPM/Desktop/python programs/array1.py ===========
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
>>>
```

Check Number of Dimensions:

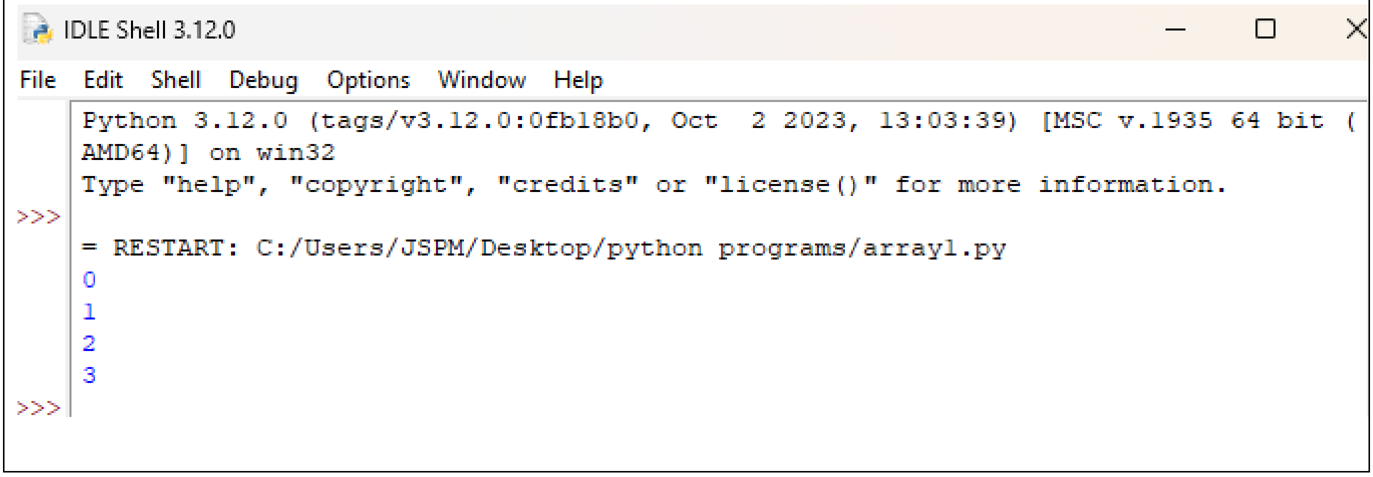
NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

Example

Check how many dimensions the arrays have:

```
import numpy as np

a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
c = np.array([[1, 2, 3], [4, 5, 6]])
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
```

Output:



```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/JSPM/Desktop/python programs/array1.py
0
1
2
3
>>>
```

Assignment No. 7

Develop a code to understand array indexing and slicing

Aim: Understanding methods from numpy package for array indexing and slicing

NumPy Array Indexing

Access Array Elements

Array indexing is the same as accessing an array element.

You can access an array element by referring to its index number.

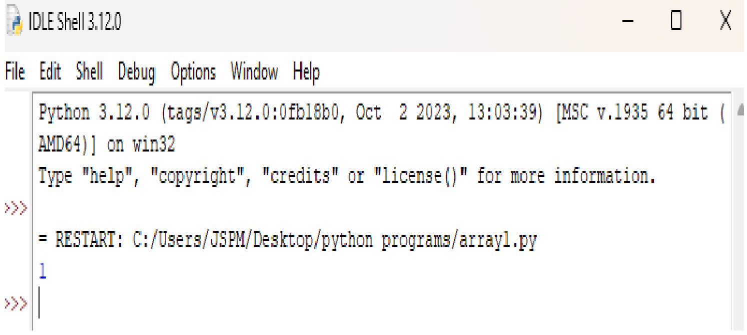
The indexes in NumPy arrays start with 0, meaning that the first element has index 0, and the second has index 1 etc.

Example

Get the first element from the following array:

```
import numpy as np

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

print(arr[0])
```

Output:

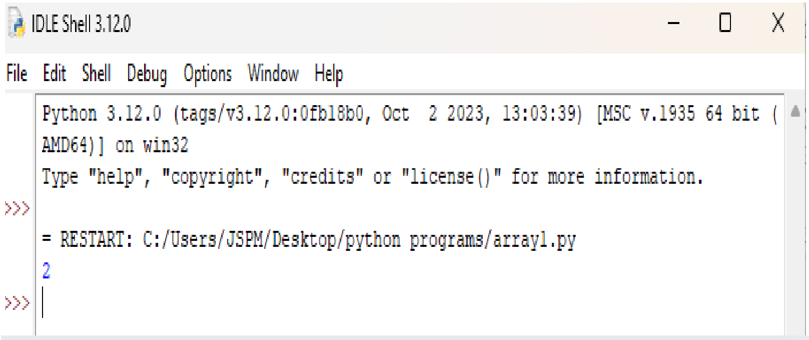


```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/JSPM/Desktop/python programs/array1.py
1
>>>
```

Example

Get the second element from the following array.

```
import numpy as np

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

print(arr[1])
```



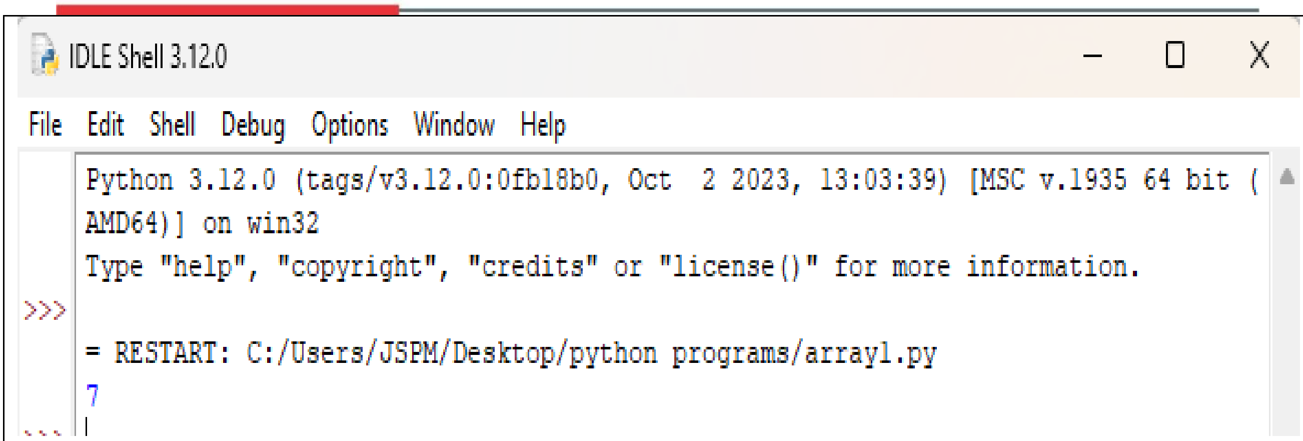
```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/JSPM/Desktop/python programs/array1.py
2
>>>
```

Example

Get third and fourth elements from the following array and add them.

```
import numpy as np

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

print(arr[2] + arr[3])
```

Output:



```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/JSPM/Desktop/python programs/array1.py
7
>>>
```

Access 2-D Arrays

To access elements from 2-D arrays we can use comma separated integers representing the dimension and the index of the element.

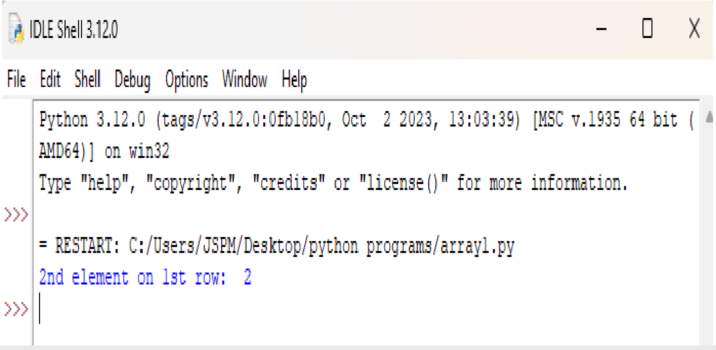
Think of 2-D arrays like a table with rows and columns, where the dimension represents the row and the index represents the column.

Example

Access the element on the first row, second column:

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print('2nd element on 1st row: ', arr[0, 1])
```

Output:



```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/JSPM/Desktop/python programs/array1.py
2nd element on 1st row:  2
>>>
```

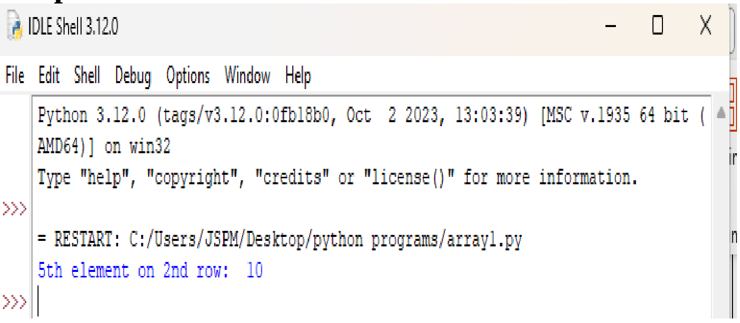
Example

Access the element on the 2nd row, 5th column:

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
```

```
print('5th element on 2nd row: ', arr[1, 4])
```

Output:



```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/JSPM/Desktop/python programs/array1.py
5th element on 2nd row:  10
>>>
```

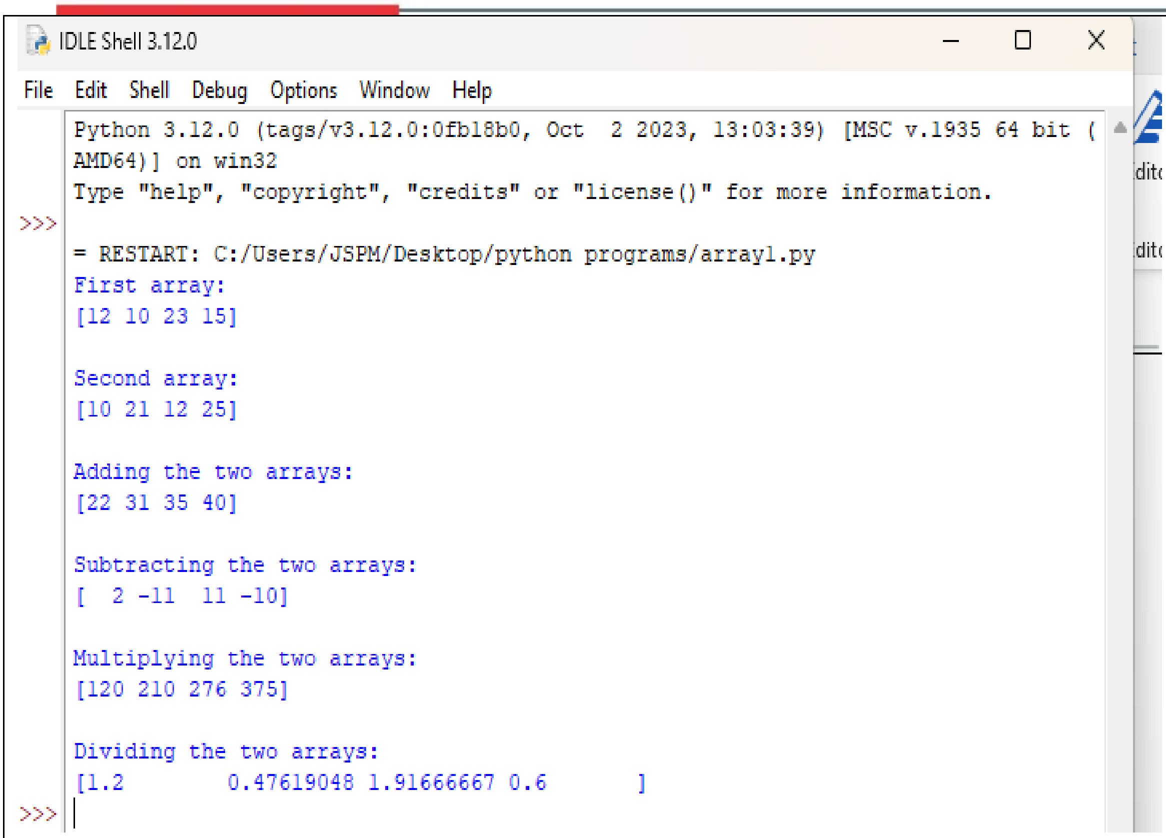
Develop code to understand following operations on matrices
(i) Addition of two matrices
(ii) Subtraction of two matrices
(iii)multiplication of two matrices
(iv)division of matrix elements
(v)Dot product of two matrices

Aim: To understand different arithmetic operations on arrays using numpy package

Code:

```
# Python code to perform arithmetic
# operations on NumPy array


import numpy as np


# Initializing the array
arr1 = np.array([12,10,23,15])

print('First array:')
print(arr1)

print('\nSecond array:')
arr2 = np.array([10,21,12,25])
print(arr2)

print('\nAdding the two arrays:')
print(np.add(arr1, arr2))

print('\nSubtracting the two arrays:')
print(np.subtract(arr1, arr2))

print('\nMultiplying the two arrays:')
print(np.multiply(arr1, arr2))

print('\nDividing the two arrays:')
print(np.divide(arr1, arr2))
```

Output:



```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
= RESTART: C:/Users/JSPM/Desktop/python programs/array1.py
First array:
[12 10 23 15]

Second array:
[10 21 12 25]

Adding the two arrays:
[22 31 35 40]

Subtracting the two arrays:
[  2 -11  11 -10]

Multiplying the two arrays:
[120 210 276 375]

Dividing the two arrays:
[1.2        0.47619048 1.91666667 0.6       ]
>>> 
```

Assignment no 9

Create Data Frame using Pandas and print number of columns and Number of Rows.

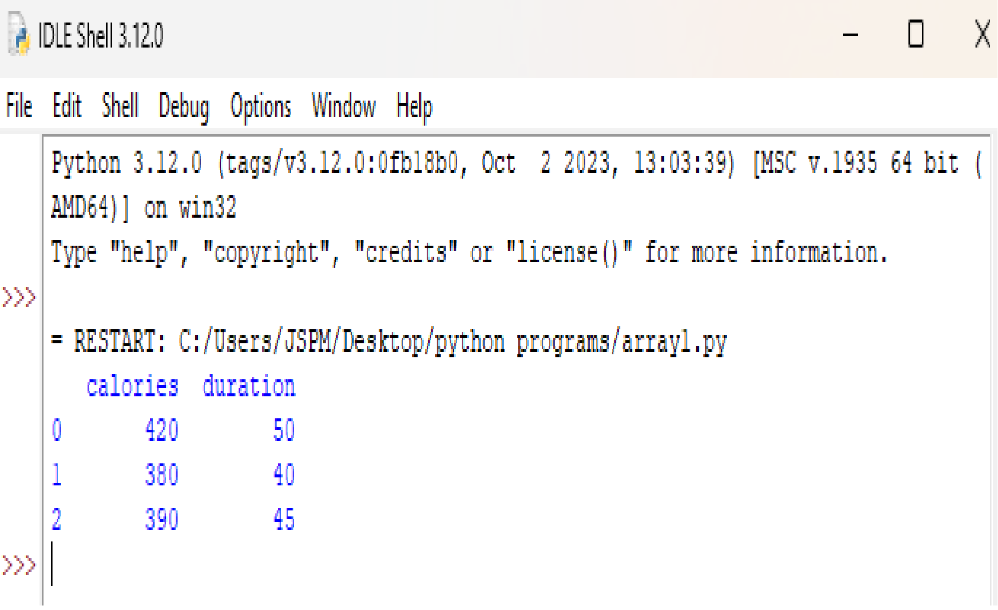
Aim : To understand the Pandas library to create data set and display data set.

Code:

Create a simple Pandas DataFrame:

```
import pandas as pd

data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

#load data into a DataFrame object:
df = pd.DataFrame(data)
print(df)
```

Output:



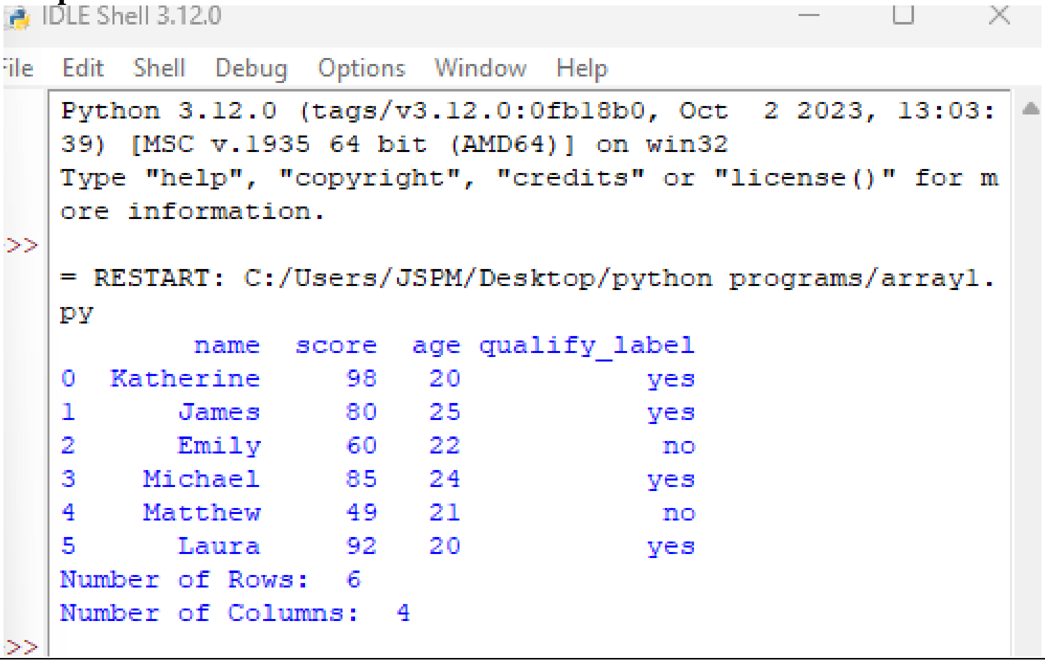
```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/JSPM/Desktop/python programs/array1.py
   calories  duration
0       420        50
1       380        40
2       390        45
>>>
```

Code to print number of rows and columns in data frame

```
# importing pandas
import pandas as pd
result_data = {'name': ['Katherine', 'James', 'Emily',
                        'Michael', 'Matthew', 'Laura'],
               'score': [98, 80, 60, 85, 49, 92],
               'age': [20, 25, 22, 24, 21, 20],
               'qualify_label': ['yes', 'yes', 'no',
                                 'yes', 'no', 'yes']}

# creating dataframe
df = pd.DataFrame(result_data, index=None)
```

```
# computing number of rows
rows = len(df.axes[0])

# computing number of columns
cols = len(df.axes[1])

print(df)
print("Number of Rows: ", rows)
print("Number of Columns: ", cols)
```

Output:



```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:
39) [MSC v.1935 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for m
ore information.
>>
= RESTART: C:/Users/JSPM/Desktop/python programs/array1.
py
        name  score  age qualify_label
0  Katherine     98   20           yes
1      James     80   25           yes
2      Emily     60   22            no
3    Michael     85   24           yes
4    Matthew     49   21            no
5      Laura     92   20           yes
Number of Rows:  6
Number of Columns:  4
>>
```

Assignment 10:

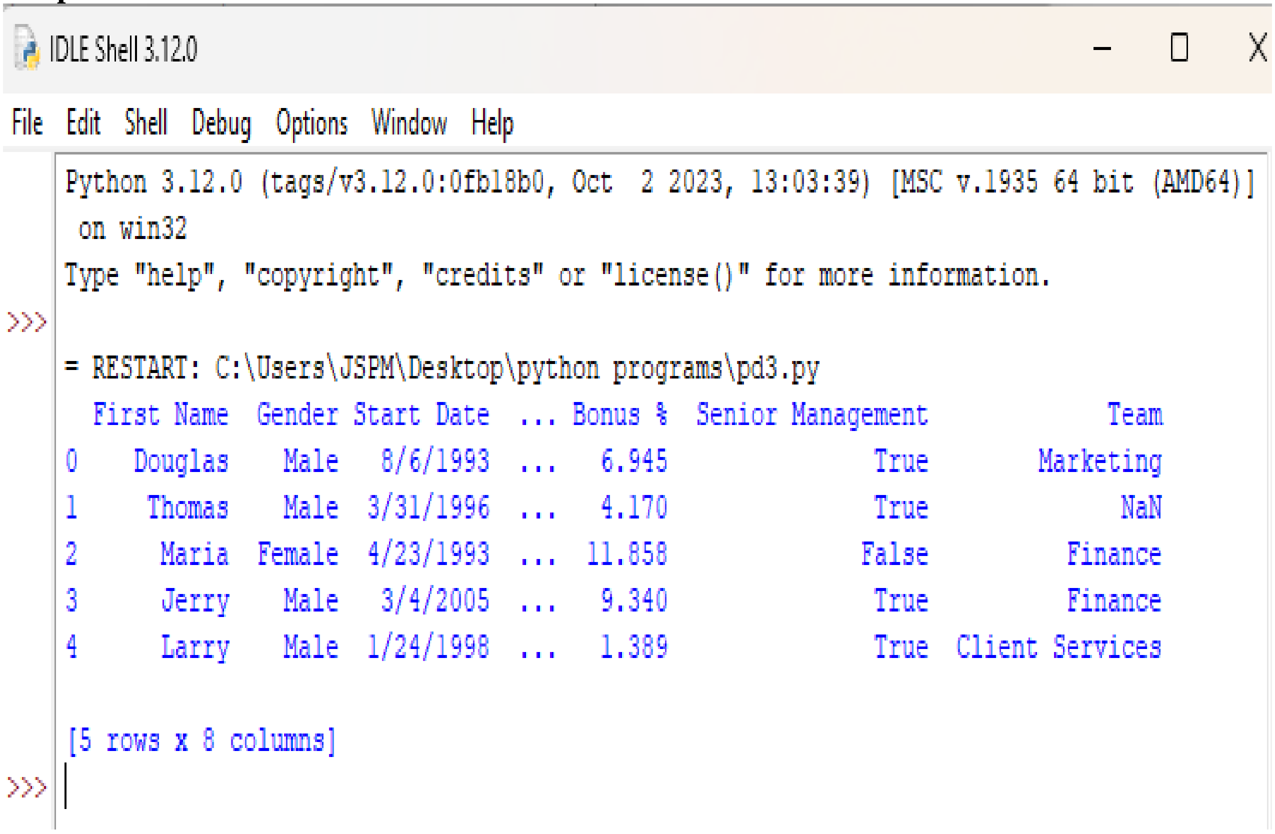
Write code to Extract and Read Data With Pandas .csv file and .xls file

Aim: To understand Pandas library to read data from .csv files and .xls files .

Code:

```
# read dataset using pandas

import pandas as pd

import numpy as np

df = pd.read_csv("D:/Manisha/employees.csv")

print(df.head())
```

Output:

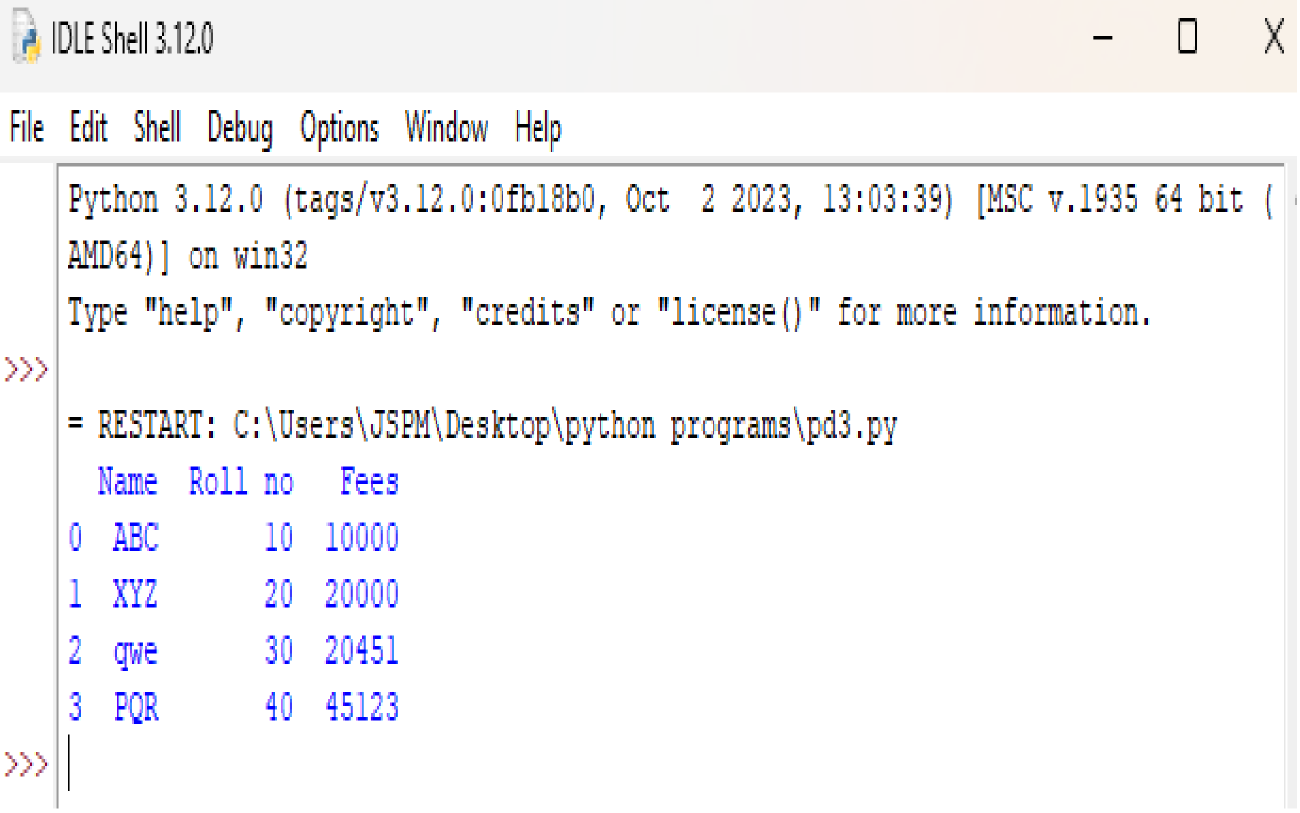


```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (AMD64)]
 on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
= RESTART: C:\Users\JSPM\Desktop\python programs\pd3.py
  First Name  Gender Start Date  ... Bonus %  Senior Management            Team
0    Douglas    Male   8/6/1993  ...   6.945               True       Marketing
1     Thomas    Male  3/31/1996  ...   4.170               True             NaN
2      Maria  Female  4/23/1993  ...  11.858              False         Finance
3      Jerry    Male   3/4/2005  ...   9.340               True         Finance
4      Larry    Male  1/24/1998  ...   1.389               True  Client Services

[5 rows x 8 columns]
>>> 
```

# read dataset using pandas

```
import pandas as pd
import numpy as np
df = pd.read_excel("D:/Manisha/book1.xlsx")
print(df.head())
```

Output:



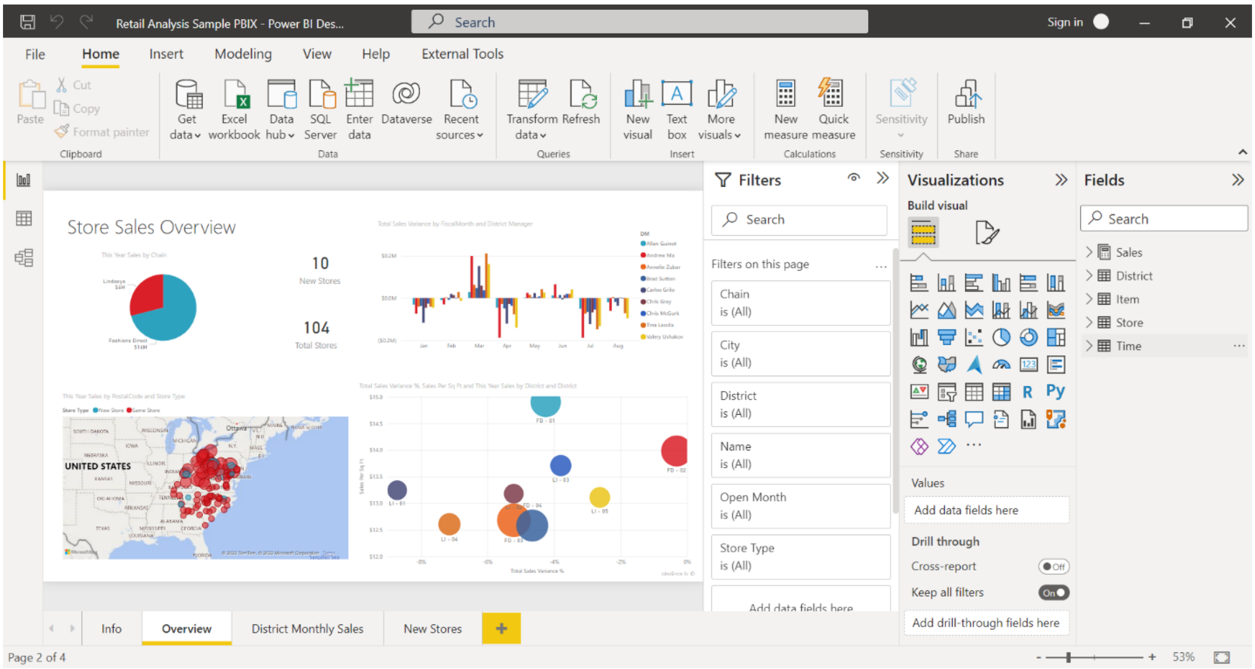
```
Python 3.12.0 (tags/v3.12.0:0fb18b0, Oct  2 2023, 13:03:39) [MSC v.1935 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\JSPM\Desktop\python programs\pd3.py
  Name  Roll no   Fees
0  ABC       10  10000
1  XYZ       20  20000
2  qwe       30  20451
3  PQR       40  45123
>>>
```

Assignment No:11

Design Desktop using Power BI

Steps: ***Data Visualization Using Power BI***

Power BI Desktop is a free application you install on your local computer that lets you connect to, transform, and visualize your data. With Power BI Desktop, you can connect to multiple different sources of data, and combine them (often called *modeling*) into a data model. This data model lets you build visuals, and collections of visuals you can share as reports, with other people inside your organization. Most users who work on business intelligence projects use Power BI Desktop to create reports, and then use the *Power BI service* to share their reports with others.



The most common uses for Power BI Desktop are as follows:

- Connect to data.
- Transform and clean data to create a data model.
- Create visuals, such as charts or graphs that provide visual representations of the data.
- Create reports that are collections of visuals on one or more report pages.
- Share reports with others by using the Power BI service.

People who are responsible for such tasks are often considered *data analysts* (sometimes referred to as *analysts*) or business intelligence professionals (often referred to as *report creators*). Many people who don't consider themselves an analyst or a report creator use Power BI Desktop to create compelling reports, or to pull data from various sources. They can build data models, and then share the reports with their coworkers and organizations.

Important

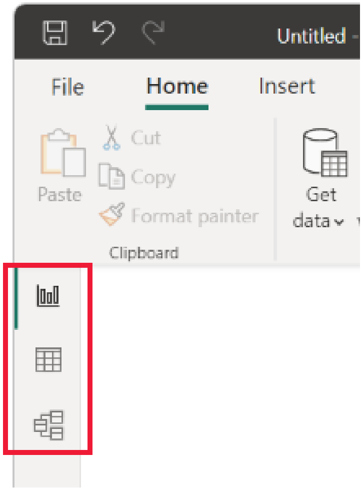
Power BI Desktop is updated and released on a monthly basis, incorporating customer feedback and new features. Only the most recent version of Power BI Desktop is supported; customers who contact support for Power BI Desktop will be asked to upgrade to the most recent version. You can get the most recent version of Power BI Desktop from the **Windows Store**, or as a single executable containing all supported languages that you **download** and install on your computer.

There are three views available in Power BI Desktop, which you select on the left side of the canvas. The views, shown in the order they appear, are as follows:

- **Report**: You create reports and visuals, where most of your creation time is spent.
- **Data**: You see the tables, measures, and other data used in the data model associated with your report, and transform the data for best use in the report's model.

- **Model**: You see and manage the relationships among tables in your data model.

The following image shows the three views, as displayed along the left side of the canvas:



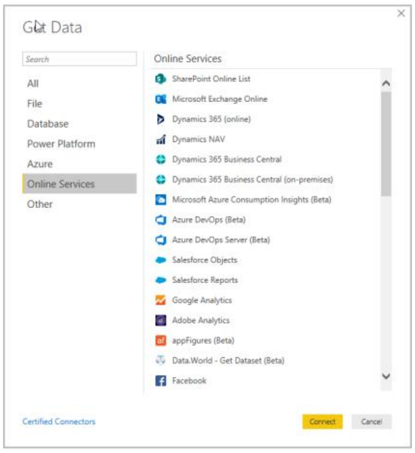
Connect to data

To get started with Power BI Desktop, the first step is to connect to data. There are many different data sources you can connect to from Power BI Desktop.

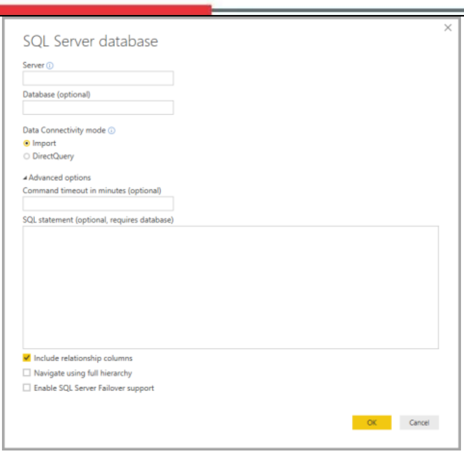
To connect to data:

1. From the **Home** ribbon, select **Get Data** > **More**.

   The **Get Data** window appears, showing the many categories to which Power BI Desktop can connect.



2. When you select a data type, you're prompted for information, such as the URL and credentials, necessary for Power BI Desktop to connect to the data source on your behalf.



SQL Server database

Server

Database (optional)

Data Connectivity mode
● Import
○ DirectQuery

Advanced options
Command timeout in minutes (optional)

SQL statement (optional, requires database)

☑ Include relationship columns
□ Navigate using full hierarchy
□ Enable SQL Server Failover support

3. After you connect to one or more data sources, you may want to transform the data so it's useful for you.

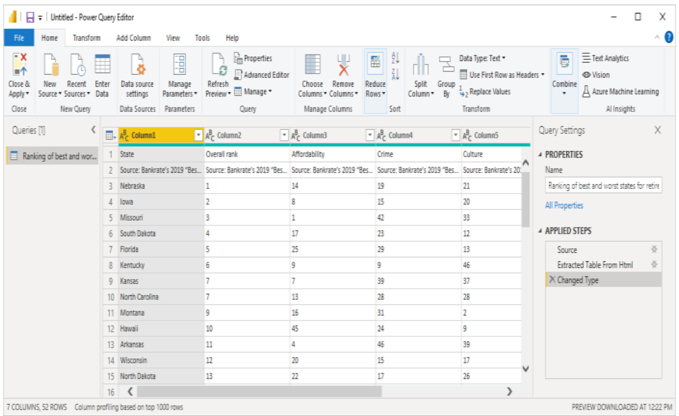
Transform and clean data, create a model

In Power BI Desktop, you can clean and transform data using the built-in Power Query Editor. With Power Query Editor, you make changes to your data, such as changing a data type, removing columns, or combining data from multiple sources. It's like sculpting: you start with a large block of clay (or data), then shave off pieces or add others as needed, until the shape of the data is how you want it.

To start Power Query Editor:

- On the **Home** ribbon, in the **Queries** section, select **Transform data**.

  The **Power Query Editor** window appears.



	Column1	Column2	Column3	Column4	Column5
1	State	Overall rank	Affordability	Crime	Culture
2	Source: Bankrate's 2019 "Bes...	Source: Bankrate's 2019 "Bes...	Source: Bankrate's 2019 "Bes...	Source: Bankrate's 2019 "Bes...	Source: Bankrate's 20:
3	Nebraska	1	14	19	21
4	Iowa	2	8	15	20
5	Missouri	3	1	42	33
6	South Dakota	4	17	23	12
7	Florida	5	25	29	13
8	Kentucky	6	9	9	46
9	Kansas	7	7	39	37
10	North Carolina	7	13	28	28
11	Montana	9	16	31	2
12	Hawaii	10	45	24	9
13	Arkansas	11	4	46	39
14	Wisconsin	12	20	15	17
15	North Dakota	13	22	17	26
16					

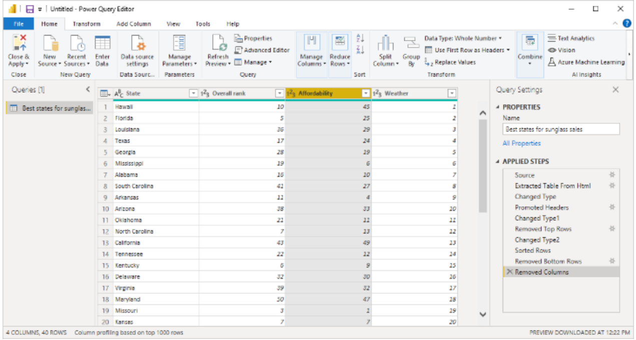
PROPERTIES
Name
Ranking of best and worst states for retire
All Properties

APPLIED STEPS
Source
Extracted Table From Html
Changed Type

7 COLUMNS, 52 ROWS Column profiling based on top 1000 rows
PREVIEW DOWNLOADED AT 12:22 PM

Each step you take in transforming data (such as renaming a table, transforming a data type, or deleting a column) is recorded by Power Query Editor. Every time this query connects to the data source, those steps are carried out so that the data is always shaped the way you specify.

The following image shows the **Power Query Editor** window for a query that was shaped, and turned into a model.



Name: Best states for sunglass sales

APPLIED STEPS:
- Source
- Extracted Table From Html
- Changed Type
- Promoted Headers
- Changed Type1
- Removed Top Rows
- Changed Type2
- Sorted Rows
- Removed Bottom Rows
- Removed Columns

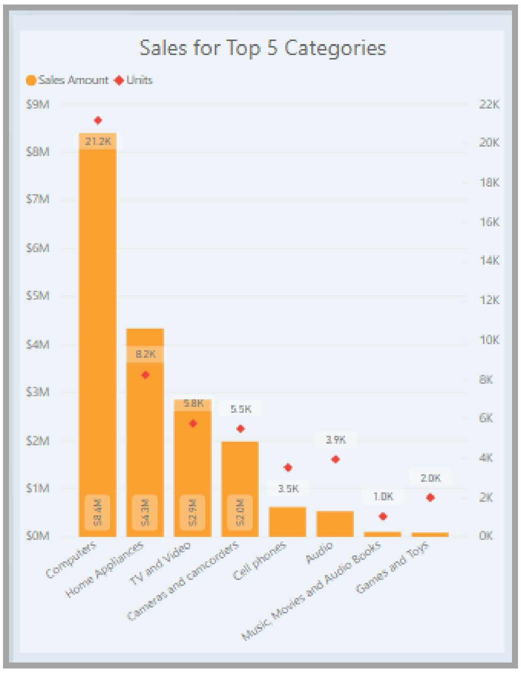
	State	Overall rank	Affordability	Weather
1	Hawaii	10	45	1
2	Florida	5	25	2
3	Louisiana	36	29	3
4	Texas	17	24	4
5	Georgia	28	19	5
6	Mississippi	19	6	6
7	Alabama	16	10	7
8	South Carolina	41	27	8
9	Arkansas	11	4	9
10	Arizona	38	33	10
11	Oklahoma	21	11	11
12	North Carolina	7	13	12
13	California	43	49	13
14	Tennessee	22	12	14
15	Kentucky	6	9	15
16	Delaware	32	30	16
17	Virginia	39	32	17
18	Maryland	50	47	18
19	Missouri	3	1	19
20	Kansas	7	7	20

4 COLUMNS, 40 ROWS Column profiling based on top 1000 rows

Once your data is how you want it, you can create visuals.

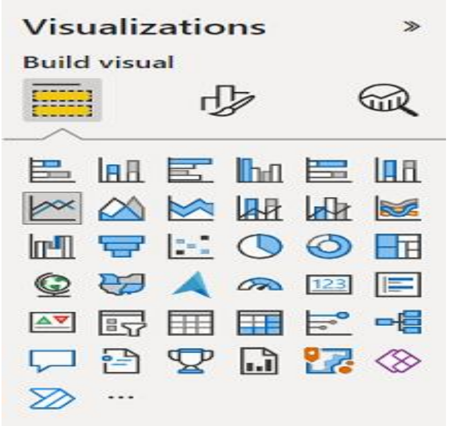
Create visuals

After you have a data model, you can drag *fields* onto the report canvas to create *visuals*. A visual is a graphic representation of the data in your model. There are many different types of visuals to choose from in Power BI Desktop. The following visual shows a simple column chart.



To create or change a visual:

- From the **Visualizations** pane, select the **Build visual** icon.



If you already have a visual selected on the report canvas, the selected visual changes to the type you selected.

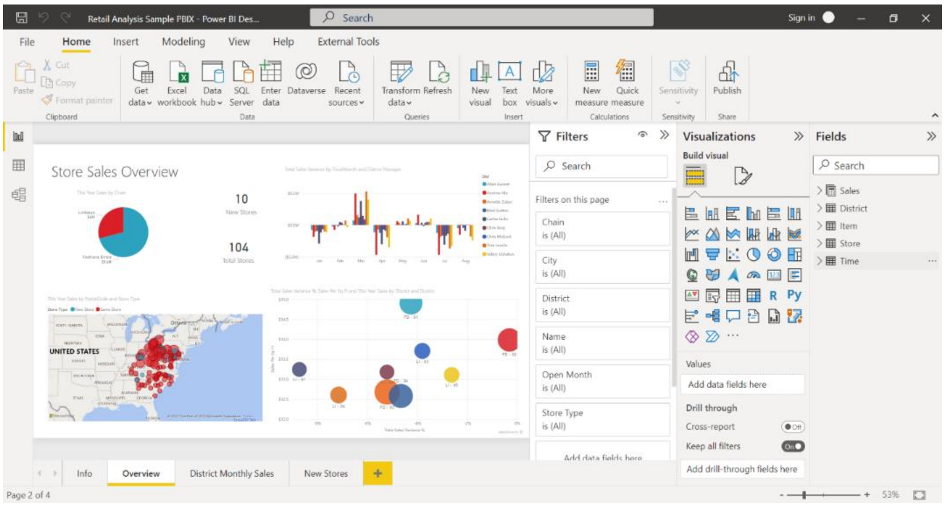
If no visual is selected on the canvas, a new visual is created based on your selection.

Create reports

More often, you'll want to create a collection of visuals that show various aspects of the data you've used to create your model in Power BI Desktop. A collection of visuals, in one Power BI Desktop file, is called a *report*. A report can have one or more pages, just like an Excel file can have one or more worksheets.

With Power BI Desktop you can create complex and visually rich reports, using data from multiple sources, all in one report that you can share with others in your organization.

In the following image, you see the first page of a Power BI Desktop report, named **Overview**, as seen on the tab near the bottom of the image.

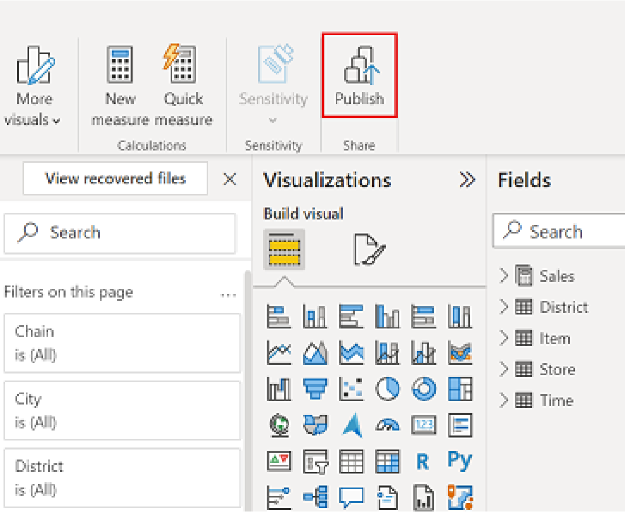


Share reports

After a report is ready to share with others, you can *publish* the report to the Power BI service, and make it available to anyone in your organization who has a Power BI license.

To publish a Power BI Desktop report:

1. Select **Publish** from the **Home** ribbon.



Power BI Desktop connects you to the Power BI service with your Power BI account.

2. You're prompted to select where in the Power BI service you'd like to share the report. For example, your workspace, a team workspace, or some other location in the Power BI service.

   You must have a Power BI license to share reports to the Power BI service.