1.Demonstrate three different methods for creating identical 2D arrays in NumPy Provide the code for each method and the final output after each method

In [1]:
```
#three methods to creating arrays
#-->np.array() method
#-->np.asarray() method
#-->np.asanyarray() method


#creating with np.array() method

import numpy as np
mat1=np.array([[1,2,3],[4,5,6],[7,8,9]])
print("mat1 2d array created using np.array():\n",mat1)
mat2=np.asarray([[1,2,3],[4,5,6],[7,8,9]])
print("mat2 2d array created using np.asarray():\n",mat2)
mat3=np.asanyarray([[1,2,3],[4,5,6],[7,8,9]])
print('mat3 2d array created using np.asanyarray():\n',mat3)
```

```
mat1 2d array created using np.array():
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
mat2 2d array created using np.asarray():
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
mat3 2d array created using np.asanyarray():
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

2.Using the Numpy function, generate an array of 100 evenly spaced numbers between 1 and 10 and Reshape that 1D array into a 2D array.

In [2]:

```
#we can create array with  100 evenly spaced num between  1 and 10
#by using function np.linspace()
#with the help of np.linspace() we can get evenly
#spaced numbers between any two num
#np.linspace(start,stop,num=no_of _datapoints_you_want,
#endpoint=True/False,dtype)
#num means no of data points you what between that two numbers
#endpoint =true means rifgt boundary  num should iclude
#retstep=True  means it gives step size


a=np.linspace(1,10,num=100,endpoint=True)
b=a.reshape(5,20)      #5 rows and 20 columns
#reshape() is used to change the shape of ndarray
#a is 1d array
#on reshaping a, b array(2d) is obtained
print("100 evenly spaced num between  1 and 10:",a)

print()
print("2d array:",b)
```

```
100 evenly spaced num between  1 and 10: [ 1.         1.09090909  1.18181818
1.27272727  1.36363636  1.45454545
  1.54545455  1.63636364  1.72727273  1.81818182  1.90909091  2.
  2.09090909  2.18181818  2.27272727  2.36363636  2.45454545  2.54545455
  2.63636364  2.72727273  2.81818182  2.90909091  3.          3.09090909
  3.18181818  3.27272727  3.36363636  3.45454545  3.54545455  3.63636364
  3.72727273  3.81818182  3.90909091  4.          4.09090909  4.18181818
  4.27272727  4.36363636  4.45454545  4.54545455  4.63636364  4.72727273
  4.81818182  4.90909091  5.          5.09090909  5.18181818  5.27272727
  5.36363636  5.45454545  5.54545455  5.63636364  5.72727273  5.81818182
  5.90909091  6.          6.09090909  6.18181818  6.27272727  6.36363636
  6.45454545  6.54545455  6.63636364  6.72727273  6.81818182  6.90909091
  7.          7.09090909  7.18181818  7.27272727  7.36363636  7.45454545
  7.54545455  7.63636364  7.72727273  7.81818182  7.90909091  8.
  8.09090909  8.18181818  8.27272727  8.36363636  8.45454545  8.54545455
  8.63636364  8.72727273  8.81818182  8.90909091  9.          9.09090909
  9.18181818  9.27272727  9.36363636  9.45454545  9.54545455  9.63636364
  9.72727273  9.81818182  9.90909091 10.        ]

2d array: [[ 1.         1.09090909  1.18181818  1.27272727  1.36363636  1.45
454545
   1.54545455  1.63636364  1.72727273  1.81818182  1.90909091  2.
   2.09090909  2.18181818  2.27272727  2.36363636  2.45454545  2.54545455
   2.63636364  2.72727273]
 [ 2.81818182  2.90909091  3.          3.09090909  3.18181818  3.27272727
   3.36363636  3.45454545  3.54545455  3.63636364  3.72727273  3.81818182
   3.90909091  4.          4.09090909  4.18181818  4.27272727  4.36363636
   4.45454545  4.54545455]
 [ 4.63636364  4.72727273  4.81818182  4.90909091  5.          5.09090909
   5.18181818  5.27272727  5.36363636  5.45454545  5.54545455  5.63636364
   5.72727273  5.81818182  5.90909091  6.          6.09090909  6.18181818
   6.27272727  6.36363636]
 [ 6.45454545  6.54545455  6.63636364  6.72727273  6.81818182  6.90909091
   7.          7.09090909  7.18181818  7.27272727  7.36363636  7.45454545
   7.54545455  7.63636364  7.72727273  7.81818182  7.90909091  8.
   8.09090909  8.18181818]
 [ 8.27272727  8.36363636  8.45454545  8.54545455  8.63636364  8.72727273
   8.81818182  8.90909091  9.          9.09090909  9.18181818  9.27272727
   9.36363636  9.45454545  9.54545455  9.63636364  9.72727273  9.81818182
   9.90909091 10.        ]]
```

3 Explain the following terms -->The difference in nparray, npasarray and npasanyarray -->The difference between Deep copy and shallow copyy

-->The difference in nparray, npasarray and npasanyarray

In [3]:
```
#3the differences in anparray,np.asarray,np.asanyarray()

#np.arraY()
"""
--> np.array()--- this method is mainly used to create ndarray.
every time it will create a new ndarray object. np.array()
takes a iterable object and converts it into ndarray that
iterable object may be a ndarray. so np.asaary() method always create
new ndarray object.
"""
#code

a=np.array([[1,2,3],[4,5,6]])  #creating 2d array
b=np.array(a) #passing an 2d array
print("np.array()")
print('a and b is(objects equal/not equal) :',a is b)


#np.asarray()
"""
-->np.asaary()--- is also used to create ndarrays. but it is mainly used
to convert the existing ndarray into another ndarray
without generating new object in memory. simply it manipulates
the existing ndrray if already it is a ndarray. if it is not ndarray
it will creates a new ndarray ,this time a new object will gets creates.

"""

a=np.array([[1,2,3],[4,5,6]])  #creating 2d array
b=np.asarray(a) #passing an 2d array
print("np.asarray()")
print('a and b is(objects equal/not equal) :',a is b)


#np.asanyarray()

"""
--> np.asanyarray()--- is also used to create a nd arrays.
this function is used mainly when we want to convert input to an array
but it pass ndarray subclasses through. input can be lists
,tuples,list of tuples,ndarrays etc..,
if passed  is ndarray  or a subclass of ndarray
,it is returned without new copy .

"""
#for example matrx is a subclass  of ndarray
#if we pass matrix in np.asanyarray() it wont create a
#new object it will returns the object with same memory location

a=np.matrix([[1,2],[3,4]])
print("np.asanyarray()")
b=np.asanyarray(a)
print('a and b is (objects equal/not equal):',a is b)
```

```
np.array()
a and b is(objects equal/not equal) : False
np.asarray()
a and b is(objects equal/not equal) : True
np.asanyarray()
a and b is (objects equal/not equal): True
```

-->the differences between deep copy and shallow copy

In [4]:
```
#deepcopy
"""
-->deep copy ----- deep copy means we will create a new object with
different memory location ,this new object will have the
contents same  as the original data structure . but on making
changes in any of this two data structure does not
reflect on other data structure.
that means here  two data structures will
have diffrent memory locations .
"""

import copy
l1=[1,2,3,4]
l2=copy.deepcopy(l1)
print("is l2 and l1 are same:(deepcopy)" ,l1 is l2)
l2[0]=100  #making changes in l2 list
print("l1 is:",l1)
print("l2 is :",l2)
print()


#shallowcopy
"""
-->shallow copy ----- shallow copy means we won't create a new object
with different memory location ,the object created by shallow copy will have th
contents same  as the original data structure  and shares the
memory location . so on making changes in any of
this two data structure it reflect on other data structure.
that means here  two data structures will have same memory locations .
"""


l1=[1,2,3,4]
l2=l1
print("is l2 and l1 are same:(shallowcopy)" ,l1 is l2)
l2[0]=100  #making changes in l2 list
print("l1 is:",l1)
print("l2 is :",l2)
```

```
is l2 and l1 are same:(deepcopy) False
l1 is: [1, 2, 3, 4]
l2 is : [100, 2, 3, 4]

is l2 and l1 are same:(shallowcopy) True
l1 is: [100, 2, 3, 4]
l2 is : [100, 2, 3, 4]
```

4. Generate a 3x3 array with random floating-point numbers between 5 and 20 then, round each number in the array to 2 decimal places.

In [5]:
```
import numpy as np
#we can use numpy.uniform function which is used
#to generate floating point numbers
#it has syntax numpy.random.uniform(low,high,size)
#size means shape of array
# Generating a 3x3 array with random floating-point
#numbers between 5 and 20
arr= np.random.uniform(5, 20, (3, 3))

# Round each number in the array to 2 decimal places
#round() is used to round values
rounded_arr= np.round(arr, 2)

print(rounded_arr)
```

```
[[19.41 12.82 16.25]
 [14.64 12.65 14.06]
 [17.91 19.25  7.4 ]]
```

5.Create a NumPy array with random integers between 1 and 10 of shape (5,6). After creating the array perform the following opeartions a)extract all even integers from array b)extract all odd integers from array.

In [6]:
```
import numpy as np
#we can use numpy.random.randint() to generate integers over an interval
#syntax is numpy.random.randint(low,high,size)
#low-->1  high-->11 (to include 10 also because it is exclusive)
#shape-->(5,6)
# Creating  a NumPy array with random integers between 1 and 10 of
#shape (5, 6)
random_int_array = np.random.randint(1, 11, (5, 6))

# Extract all even integers from the array
even_integers =random_int_array[random_int_array % 2 == 0]

# Extract all odd integers from the array
odd_integers = random_int_array[random_int_array % 2 != 0]

#printing even and odd integers
print("Original Array:\n", random_int_array)
print("Even Integers:\n", even_integers)
print("Odd Integers:\n", odd_integers)
```

```
Original Array:
 [[ 1  3 10  1  6  2]
 [ 8  6  9  4  6  7]
 [ 6  3  3 10  2  4]
 [ 7  3  9 10  2 10]
 [ 5  3  1  7 10 10]]
Even Integers:
 [10  6  2  8  6  4  6  6 10  2  4 10  2 10 10 10]
Odd Integers:
 [1 3 1 9 7 3 3 7 3 9 5 3 1 7]
```

6.create a 3d NumPy array of shape (3,3,3) containing random integers between 1 and 10. Perform the following operations:

a) Find the indices of the maximum values along each depth level (third axis).

b) Perform element-wise multiplication of between both array

In [7]:
```python
import numpy as np

# Creating a 3D NumPy array with random integers between 1 and 10 of shape (3,
arr_3d = np.random.randint(1, 11, (3, 3,3))
print("Original 3D Array:\n", arr_3d)



#-->Find the indices of the maximum values along each depth level (third axis).
"""
The numpy.argmax function is used to
find the indices of the maximum values along a specified axis in a NumPy array.
syntax:numpy.argmax(arr,axis=0/1/2.,)
2-->thirdaxis
0-->column wise
1-->row wise
"""
max_indices = np.argmax(arr_3d, axis=2)
print("Indices of Maximum Values along each depth level:\n", max_indices)
print()
#--->performing element wise operation on between both array
result_arr = np.multiply(arr_3d,max_indices)
print("Element-wise Multiplication Result:\n", result_arr)
```

```
Original 3D Array:
 [[[ 1  2  1]
  [ 2  5  1]
  [ 2  6  2]]

 [[ 8  9  9]
  [ 6  1  1]
  [ 1  3  2]]

 [[10  6  5]
  [ 4  1  7]
  [ 2  4  9]]]
Indices of Maximum Values along each depth level:
 [[1 1 1]
 [1 0 1]
 [0 2 2]]

Element-wise Multiplication Result:
 [[[ 1  2  1]
  [ 2  0  1]
  [ 0 12  4]]

 [[ 8  9  9]
  [ 6  0  1]
  [ 0  6  4]]

 [[10  6  5]
  [ 4  0  7]
  [ 0  8 18]]]
```

7.clean and transform the 'phone' column in the sample dataset to remove non-numeric charactersand convert it to a numeric data type. Also display the table attributes and data types of each column.

In [8]:
```
import pandas as pd
import numpy as np
import re
#importing dataset
df=pd.read_csv(r"C:\Users\sai kiran\Downloads\people_data.csv")
df['Phone']=df['Phone'].str.replace(r'\D+', '').astype(float)  #using regessior
df['Phone'].fillna(0000000,inplace=True)  #null values of numbers
df['Phone']
#
```

```
C:\Users\sai kiran\AppData\Local\Temp\ipykernel_9364\1562689098.py:6: FutureW
arning: The default value of regex will change from True to False in a future
version.
  df['Phone']=df['Phone'].str.replace(r'\D+', '').astype(float)  #using reges
sion re package
```

Out[8]:
```
0      8.571398e+09
1      0.000000e+00
2      5.997821e+09
3      0.000000e+00
4      3.904172e+13
           ...
995    2.177529e+08
996    1.149711e+13
997    1.750774e+15
998    9.152922e+09
999    7.975254e+13
Name: Phone, Length: 1000, dtype: float64
```

In [9]:
```
#displaying  the table attributes and data types of each column.
print("Table Attributes:")
print(f"Number of rows: {len(df)}")
print(f"Number of columns: {len(df.columns)}")
print("\nData Types of Each Column:")
print(df.dtypes)
```

```
Table Attributes:
Number of rows: 1000
Number of columns: 10

Data Types of Each Column:
Index              int64
User Id           object
First Name        object
Last Name         object
Gender            object
Email             object
Phone            float64
Date of Birth     object
Job Title         object
Salary             int64
dtype: object
```

In [10]:
```
#describe about peopledata
df.describe(include='all')
```

Out[10]:

	Index	User Id	First Name	Last Name	Gender	Email	Phone
count	1000.000000	1000	1000	1000	1000	1000	1.000000e+03
unique	NaN	1000	526	628	2	1000	NaN
top	NaN	8717bbf45cCDbEe	Lydia	Duke	Male	pwarner@example.org	NaN
freq	NaN	1	6	6	506	1	NaN
mean	500.500000	NaN	NaN	NaN	NaN	NaN	1.813169e+14
std	288.819436	NaN	NaN	NaN	NaN	NaN	4.160588e+14
min	1.000000	NaN	NaN	NaN	NaN	NaN	0.000000e+00
25%	250.750000	NaN	NaN	NaN	NaN	NaN	6.544581e+09
50%	500.500000	NaN	NaN	NaN	NaN	NaN	6.192731e+12
75%	750.250000	NaN	NaN	NaN	NaN	NaN	9.052905e+13
max	1000.000000	NaN	NaN	NaN	NaN	NaN	1.944627e+15

8.Perform the following tasks using people dataset:

b) Only read the columns: 'Last Name', 'Gender','Email','Phone' and 'Salary' from the file.

c) Display the first 10 rows of the filtered dataset.

d) Extract the 'Salary" column as a Series and display its last 5 values

In [11]:
```
import pandas as pd
import numpy as np
#skiprows is used to skip that no of first rows
#skiprows=range(1,51) -->skips first 50 rows
#use cols is used to read that specify columns only
df=pd.read_csv(r"C:\Users\sai kiran\Downloads\people_data.csv",
skiprows=range(1,51),usecols=['Last Name','Gender','Email','Phone','Salary'])
```

In [12]:
```
#displaying first 10 rows of the filtered dataset
#head() is used to get first n no of rows
#df.head(count)
d=df.head(10)
d
```

Out[12]:

	Last Name	Gender	Email	Phone	Salary
0	Zavala	Male	pamela64@example.net	001-859-448-9935x54536	60000
1	Carey	Female	dianashepherd@example.net	001-274-739-6470x814	70000
2	Hobbs	Female	ingramtiffany@example.org	241.179.9509x498	60000
3	Reilly	Male	carriecrawford@example.org	207.297.8345x6177	100000
4	Conrad	Male	fuentesclaudia@example.net	001-692-042-7428x143	50000
5	Cole	Male	kaneaudrey@example.org	663-280-5634	85000
6	Donovan	Male	rioskahsantos@example.net	NaN	65000
7	Little	Female	craig28@example.com	125.219.3675x0076	60000
8	Dawson	Female	conrancourtney@example.net	650-748-3069x64529	60000
9	Page	Male	harrygallagher@example.com	849.920.6331x717	60000

In [13]:
```
#Extracting  the 'Salary'' column as a Series and display its last 5 values
d=df['Salary']
#By using tail we can get last n rows
d.tail(5)
```

Out[13]:
```
945     90000
946     50000
947     60000
948    100000
949     90000
Name: Salary, dtype: int64
```

9. Filter and select rows from the People_Dataset, where the "Last Name' column contains the name 'Duke', 'Gender' column contains the word Female and 'salary' should be less than

In [14]:

```
#filtering the dataframe based on the mentioned conditions
d=df[(df['Gender']=='Female') & ( df['Last Name']=='Duke') & (df['Salary']<850
#displaying the dataframe
d
```

Out[14]:

	Last Name	Gender	Email	Phone	Salary
160	Duke	Female	robin78@example.com	740.434.0212	50000
407	Duke	Female	perryhoffman@example.org	+1-903-596-0995x489	50000
679	Duke	Female	kevinkramer@example.net	982.692.6257	70000

10.Create a 7*5 Dataframe in Pandas using a series generated from 35 random integers between integers 1 to 6?

In [15]:

```
import pandas as pd
import numpy as np

# Generating 35 random integers between 1 to 6 using random.randint()
#random.randint() genrates random int values in provided range
#here upper value is excludes that is why we have used 7 instead of 6
#random.randint(low,high,size)
random_integers = np.random.randint(1, 7, size=35)

# Reshaping the random integers into a 7x5 array
data = random_integers.reshape(7, 5)

# Creatig a DataFrame
df = pd.DataFrame(data)

# Displaying the DataFrame
print(df)
```

```
   0  1  2  3  4
0  3  4  1  3  2
1  5  2  5  5  6
2  2  3  5  5  3
3  1  6  1  4  1
4  1  2  4  3  4
5  3  5  3  3  1
6  6  2  6  3  2
```

11.Create two different Series, each of length 50, with the following criteria:

a) The first Series should contain random numbers ranging from 10 to 50.

b) The second Series should contain random numbers ranging from 100 to 1000.

c) Create a DataFrame by joining these Series by column, and, change the names of the columns to 'col1', 'col2'.

In [16]:

```python
import pandas as pd
import numpy as np

# Creating the first Series with random numbers ranging from 10 to 50
series1 = pd.Series(np.random.randint(10, 51, size=50), name='col1')

# Creating the second Series with random numbers ranging from 100 to 1000
series2 = pd.Series(np.random.randint(100, 1001, size=50), name='col2')

# Combining the two Series into a DataFrame
#concat() is used to join to series
#axis=1 -->colum  wise
df = pd.concat([series1, series2], axis=1)


# Display the DataFrame
print(df)
```

	col1	col2
0	22	607
1	35	865
2	14	330
3	35	376
4	20	259
5	46	389
6	21	643
7	33	345
8	31	238
9	47	415
10	13	140
11	44	295
12	21	852
13	39	652
14	43	894
15	29	221
16	36	212
17	26	713
18	38	110
19	22	107
20	41	185
21	36	656
22	17	681
23	34	328
24	26	535
25	43	840
26	14	584
27	26	839
28	25	819
29	25	126
30	40	655
31	44	258
32	27	533
33	43	984
34	32	490
35	11	977
36	44	791
37	47	976
38	25	967
39	39	859
40	19	835
41	19	350
42	16	799
43	46	331
44	40	712
45	37	957
46	39	380
47	12	299
48	18	486
49	44	978

12.perform the following operations using people data set:

a) Delete the 'Email', 'Phone', and 'Date of birth' columns from the dataset.

b) Delete the rows containing any missing values.

d) Print the final output also

In [17]:
```
import pandas as pd
import numpy as np

df=pd.read_csv(r"C:\Users\sai kiran\Downloads\people_data.csv")
#Deleting  the 'Email', 'Phone', and 'Date of birth' columns from the dataset.
d=df.drop(['Email','Phone','Date of birth'],axis=1)


# Delete the rows containing any missing values.
d=d.dropna(axis=0)

#Print the final output
d
```

Out[17]:

	Index	User Id	First Name	Last Name	Gender	Job Title	Salary
0	1	8717bbf45cCDbEe	Sheila	Mahoney	Male	Probation officer	90000
1	2	3d5AD30A4cD38ed	Jo	Rivers	Female	Dancer	80000
2	3	810Ce0F276Badec	Sheryl	Lowery	Female	Copy	50000
3	4	BF2a889C00f0cE1	Whitney	Hooper	Male	Counselling psychologist	65000
4	5	9afFEafAe1CBBB9	Lindsey	Rice	Female	Biomedical engineer	100000
...	...	...	...	...	...	...	...
995	996	fedF4c7Fd9e7cFa	Kurt	Bryant	Female	Personnel officer	90000
996	997	ECddaFEDdEc4FAB	Donna	Barry	Female	Education administrator	50000
997	998	2adde51d8B8979E	Cathy	Mckinney	Female	Commercial/residential surveyor	60000
998	999	Fb2FE369D1E171A	Jermaine	Phelps	Male	Ambulance person	100000
999	1000	8b756f6231DDC6e	Lee	Tran	Female	Nurse, learning disability	90000

1000 rows × 7 columns

13. Create two NumPy arrays, x and y, each containing 100 random float values between 0 and 1. Perform the following tasks using Matplotlib and NumPy:

a) Create a scatter plot using x and y, setting the color of the points to red and the marker style to 'o'.

b) Add a horizontal line at y = 0.5 using a dashed line style and label it as 'y = 0.5'.

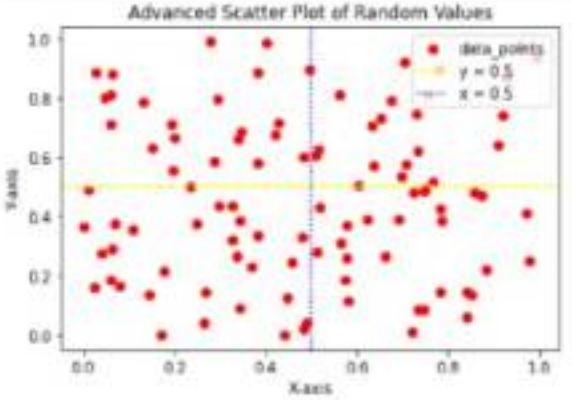
c) Add a vertical line at x = 0.5 using a dotted line style and label it as 'x = 0.5'.

d) Label the x-axis as 'X-axis' and the y-axis as 'Y-axis'.

e) Set the title of the plot as 'Advanced Scatter Plot of Random Values'.

f) Display a legend for the scatter plot, the horizontal line, and the vertical line.

In [18]:
```
import numpy as np
import matplotlib.pyplot as plt

# Generating random float values between 0 and 1 for x and y by using
#random.rand()
x = np.random.rand(100)
y = np.random.rand(100)

# Creating the scatter plot
plt.scatter(x, y, color='red', marker='o', label='data_points')

# Adding a horizontal line at y = 0.5
#With the help of axh we can draw horizontal line
plt.axhline(y=0.5, linestyle='--', color='yellow', label='y = 0.5')

# Adding a vertical line at x = 0.5
#With the help of axv line we can draw vertical line
plt.axvline(x=0.5, linestyle=':', color='blue', label='x = 0.5')

# Label the axes
plt.xlabel('X-axis')
plt.ylabel('Y-axis')

# Setting the title
plt.title('Advanced Scatter Plot of Random Values')

# Displaying legend
plt.legend()

# Showing the plot
plt.show()
```

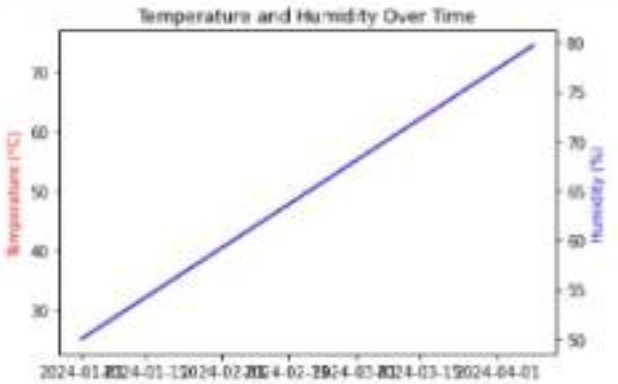


14.Create a time-series dataset in a Pandas DataFrame with columns: 'Date', 'Temperature', 'Humidity' and Perform the following tasks using Matplotlib: a) Plot the 'Temperature' and 'Humidity' on the same plot with different y-axes (left y-axis for 'Temperature' and right y-axis for 'Humidity').

b) Label the x-axis as 'Date'

c) Set the title of the plot as 'Temperature and Humidity Over Time

In [19]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Create a time-series dataset
# Let's create a sample dataset for demonstration
data = {
    'Date': pd.date_range(start='2024-01-01', periods=100),
    'Temperature': [25 + i * 0.5 for i in range(100)],
    # Sample temperature data
    'Humidity': [50 + i * 0.3 for i in range(100)]
    # Sample humidity data
}

df = pd.DataFrame(data)

# Step 2: Plot 'Temperature' and 'Humidity'
#on the same plot with different y-axes
fig, ax1 = plt.subplots()

# Plot Temperature on the left y-axis
ax1.plot(df['Date'], df['Temperature'], color='red')
ax1.set_ylabel('Temperature (°C)', color='red')

# Create another y-axis for Humidity on the right
ax2 = ax1.twinx()
ax2.plot(df['Date'], df['Humidity'], color='blue')
ax2.set_ylabel('Humidity (%)', color='blue')

# Step 3: Label the x-axis and set the title of the plot
plt.xlabel('Date')
plt.title('Temperature and Humidity Over Time')


# Show the plot
plt.show()
```



15. Create a NumPy array data containing 1000 samples from a normal distribution. Perform

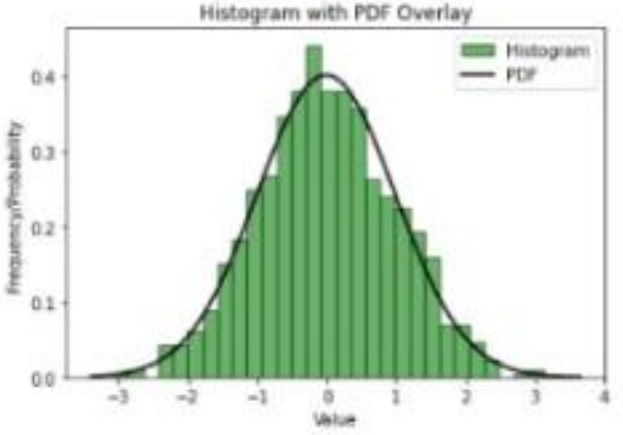
a) Plot a histogram of the data with 30 bins.

b) Overlay a line plot representing the normal distribution's probability density function (PDF).

c) Label the x-axis as 'Value' and the y-axis as 'Frequency/Probability'.

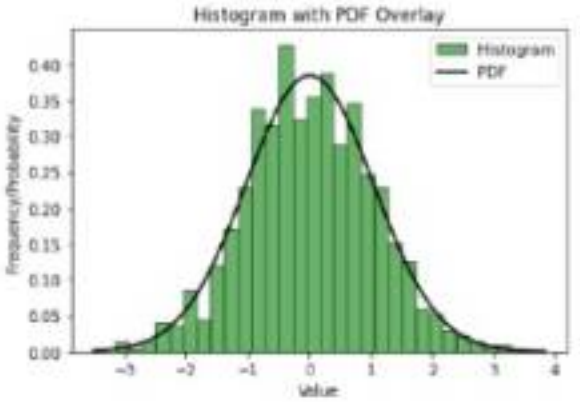
d) Set the title of the plot as 'Histogram with PDF Overlay'.

In [20]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

#crreating  a NumPy array data  containing 1000 samples
#from a normal distribution
data = np.random.normal(loc=0, scale=1, size=1000)

#plotting a histogram of the data with 30 bins
plt.hist(data, bins=30, density=True, alpha=0.6, color='g',
         edgecolor='black', label='Histogram')

#Overlay a line plot representing the normal distribution's
#probability density function (PDF)
xmin, xmax = plt.xlin()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, np.mean(data), np.std(data))
plt.plot(x, p, 'k', linewidth=2, label='PDF')

# Label the x-axis as 'Value' and the y-axis as 'Frequency/Probability'
plt.xlabel('Value')
plt.ylabel('Frequency/Probability')

#Setting the title of the plot as 'Histogram with PDF Overlay'
plt.title('Histogram with PDF Overlay')

# Add legend
plt.legend()

# Show the plot
plt.show()
```



16.Set the title of the plot as 'Histogram with PDF Overlay

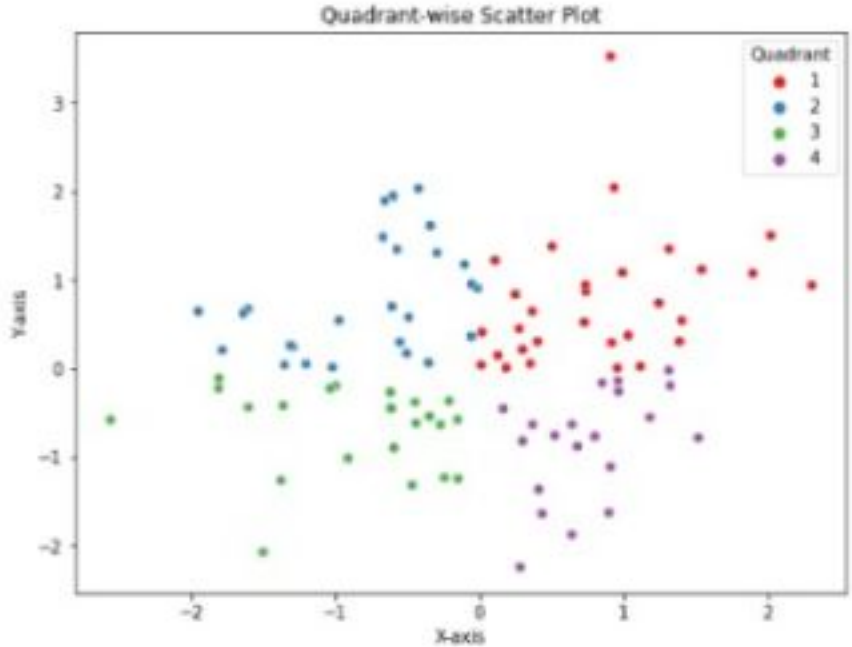
In [21]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm


#creating  a NumPy array data  containing 1000 samples
#from a normal distribution
data = np.random.normal(loc=0, scale=1, size=1000)

#plotting a histogram of the data with 30 bins
plt.hist(data, bins=30, density=True, alpha=0.6, color='g',
         edgecolor='black', label='Histogram')

#Overlay a line plot representing the normal distribution's
#probability density function (PDF)
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, np.mean(data), np.std(data))
plt.plot(x, p, 'k', linewidth=2, label='PDF')

# Label the x-axis as 'Value' and the y-axis as 'Frequency/Probability'
plt.xlabel('Value')
plt.ylabel('Frequency/Probability')

#Setting the title of the plot as 'Histogram with PDF Overlay'
plt.title('Histogram with PDF Overlay')

# Add legend
plt.legend()

# Show the plot
plt.show()
```



17.Create a Seaborn scatter plot of two random arrays, color points based on their position relative to the origin (quadrants), add a legend, label the axes, and set the title as 'Quadrant-wise Scatter Plot.

In [22]:

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

# firstly Generating two random arrays
#using numpy.random.randn to generate random numbers.
#numpy.random.randn generates an array of specified shape filled with
#random samples from a standard normal distribution (mean = 0, standard deviati
x = np.random.randn(100)
y = np.random.randn(100)

# Determining the quadrants
quadrant = np.zeros_like(x, dtype=int)
quadrant[(x > 0) & (y > 0)] = 1  # First quadrant
quadrant[(x < 0) & (y > 0)] = 2  # Second quadrant
quadrant[(x < 0) & (y < 0)] = 3  # Third quadrant
quadrant[(x > 0) & (y < 0)] = 4  # Fourth quadrant

# using  a Seaborn scatter plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x=x, y=y, hue=quadrant, palette='Set1', legend='full')

# Adding legend
plt.legend(title='Quadrant')

# Label the axes
plt.xlabel('X-axis')
plt.ylabel('Y-axis')

# Setting the title
plt.title('Quadrant-wise Scatter Plot')

# Show the plot
plt.show()
```



18.With Bokeh, plot a line chart of a sine wave function, add grid lines, label the axes, and set the title as 'Sine Wave Function.

In [23]:
```
from bokeh.plotting import figure, show
from bokeh.models import Title

import numpy as np

# Generating data for the sine wave function
x = np.linspace(0, 4*np.pi, 100)
y = np.sin(x)

# Creating a Bokeh figure
p = figure(title='Sine Wave Function', x_axis_label='X-axis',
           y_axis_label='Y-axis')

# Plotting the sine wave
p.line(x, y, line_width=2)

# Adding grid lines
p.grid.grid_line_alpha = 0.5

# Show the plot
show(p)
```

19.Using Bokeh, generate a bar chart of randomly generated categorical data, color bars based on their values, add hover tooltips to display exact values, label the axes, and set the title as 'Random Categorical Bar Chart.

In [24]:
```
from bokeh.plotting import figure, show
from bokeh.models import HoverTool

import random

# Generate random categorical data
"""We start by defining a list of categories (categories) and
generating random values (values) for each category.
For demonstration purposes, we use the random.randint function to\
generate integers between 1 and 10 for each category."""
categories = ['a','b','c','d']
values = [random.randint(1, 10) for _ in range(len(categories))]

# Creating a Bokeh figure
p = figure(x_range=categories, title='Random Categorical Bar Chart',
           x_axis_label='Categories', y_axis_label='Values')

# Plotting the bars

"""
vbar glyph method to plot vertical bars for each category.
We pass the x coordinates (categories), top coordinates (values),
and width of the bars.
We also specify colors for each bar using the color parameter.

"""
p.vbar(x=categories, top=values, width=0.9,
       color=["#FF5733", "#FFC300", "#DAF7A6", "#A0A0A0"])

# Adding hover tooltips
"""
We added a hover tool to display tooltips when hovering over the bars.
The tooltip displays the value (top) of each bar.
"""
tooltips = [("Value", "@top")]
p.add_tools(HoverTool(tooltips=tooltips))

# Set the y-axis range to include the maximum value
p.y_range.start = 0
p.y_range.end = max(values) + 1

# Showing the plot
show(p)
```

20.Using Plotly, create a basic line plot of a randomly generated dataset, label the axes, and set the title as 'Simple Line Plot'.

In [25]:

```
import plotly.graph_objects as go
import numpy as np

# Generating random data

""" to generate random data for the x and y coordinates.
Here, np.linspace(0, 10, 100) generates 100 equally spaced points between 0
and 10 for the x-axis, and np.random.randn(100)
generates 100 random numbers from a standard normal distribution for
the y-axis.
"""
x = np.linspace(0, 10, 100)
y = np.random.randn(100)

# Creating the line plot
fig = go.Figure()

# Adding a trace for the line plot
fig.add_trace(go.Scatter(x=x, y=y, mode='lines'))

# Adding labels to the axes
fig.update_layout(xaxis_title='X-axis', yaxis_title='Y-axis')

# Setting the title
fig.update_layout(title='Simple Line Plot')


fig.show()
```

21. Using Plotly, create an interactive pie chart of randomly generated data, add labels and percentages, set the title as 'Interactive Pie Chart'.

In [26]:

```
import plotly.graph_objects as go
import numpy as np

# Generating random data

"""
The labels list contains the category names,
and the values array contains the
corresponding numerical values. We use NumPy's
np.random.randint function to generate random integers
between 1 and 10 for each category.
python

"""
labels = ['dsa', 'ml', 'nlp', 'dp']
values = np.random.randint(1, 10, size=len(labels))

# Creating the pie chart
fig = go.Figure(data=[go.Pie(labels=labels, values=values,
            textinfo='label+percent', insidetextorientation='radial')])

# Setingt the title
fig.update_layout(title='Interactive Pie Chart')

# Showing the plot
fig.show()
```

Interactive Pie Chart