ques 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 Using the Numpy

```
#Method 1: Using numpy.array()
import numpy as np
# Creating a 2D array using numpy.array()
array1 = np.array([[1, 2, 3], [4, 5, 6]])
print("Method 1 - Using np.array():")
print(array1)
```

```
Method 1 - Using np.array():
[[1 2 3]
 [4 5 6]]
```

```
#Method 2: Using numpy.zeros()
# Creating a 2D array using numpy.zeros()
array2 = np.zeros((2, 3), dtype=int)
array2[0] = [1, 2, 3]
array2[1] = [4, 5, 6]
print("\nMethod 2 - Using np.zeros():")
print(array2)
```

```

Method 2 - Using np.zeros():
[[1 2 3]
 [4 5 6]]
```

```
# Creating a 2D array using numpy.full()
array3 = np.full((2, 3), 0)
array3[0] = [1, 2, 3]
array3[1] = [4, 5, 6]
print("\nMethod 3 - Using np.full():")
print(array3)
```

```

Method 3 - Using np.full():
[[1 2 3]
 [4 5 6]]
```

ques 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

You can generate an array of 100 evenly spaced numbers between 1 and 10 using numpy.linspace() and then reshape it into a 2D array using the reshape() method. Here's how to do it:

```
import numpy as np
array_1d = np.linspace(1, 10, 100)
array_2d = array_1d.reshape(10, 10)
print("1D Array:")
print(array_1d)
print("\n2D Array:")
print(array_2d)
```

```
1D Array:
[ 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  1.45454545
   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.        ]]
```

ques 3.(i) explain The difference in np.array, np.asarray and np.asanyarray

1. np.array

Functionality: Creates a new NumPy array from a given object (like a list or tuple).

Behavior: Always returns a new array.

Copies the data from the input, regardless of the input type (e.g., lists, tuples).

You can specify the data type using the dtype parameter.

Use Case: When you need a new array that is independent of the original data.

```
import numpy as np
a = [1, 2, 3]
b = np.array(a)
print(b)
```

```
[1 2 3]
```

2. np.asarray

Functionality: Converts the input to an array, but does not create a copy if the input is already an array (unless the input is of a different type).

Behavior: If the input is already a NumPy array, it returns the original array without making a copy.

If the input is not an array, it behaves like np.array and creates a new array.

Use Case: When you want to ensure that the output is a NumPy array but want to avoid unnecessary copying if the input is already an array.

```
a = np.array([1, 2, 3])
b = np.asarray(a)
print(b is a)
c = [4, 5, 6]
d = np.asarray(c)
print(d)
```

```
True
[4 5 6]
```

3. np.asanyarray

Functionality: Similar to np.asarray, but it allows for subclassing of arrays.

Behavior: If the input is an array subclass (like np.matrix), it returns that subclass. Otherwise, it behaves like np.asarray and returns a base array.

Use Case: When you want to ensure the result is an array, but you may want to preserve subclasses of arrays.

```
a = np.matrix([[1, 2], [3, 4]])
b = np.asanyarray(a)
print(type(b))
c = [7, 8, 9]
d = np.asanyarray(c)
print(type(d))
```

```
<class 'numpy.matrix'>
<class 'numpy.ndarray'>
```

np.array: Always creates a new array, making a copy of the data.

np.asarray: Returns the input as an array without copying if it is already an array.

np.asanyarray: Similar to asarray, but it retains subclasses of arrays (e.g., np.matrix).

ques 3.(ii) explain The difference between Deep copy and shallow copy

Shallow Copy

Definition: A shallow copy creates a new object that references the original data, but does not create a copy of the data itself.

Behavior: If you modify the data in the shallow copy, it will also affect the original array, because both the shallow copy and the original array refer to the same data in memory.

Usage in NumPy: You can create a shallow copy using methods like np.view() or by simply assigning the array to a new variable.

Shallow Copy: Creates a new array object that references the same data.

Modifications to the shallow copy affect the original array.

Created with methods like np.view().

```
import numpy as np
original = np.array([[1, 2, 3], [4, 5, 6]])
shallow_copy = original.view()
shallow_copy[0, 0] = 10

print("Original Array:")
print(original)
print("Shallow Copy:")
print(shallow_copy)
```

```
Original Array:
[[10  2  3]
 [ 4  5  6]]
Shallow Copy:
[[10  2  3]
 [ 4  5  6]]
```

Deep Copy

Definition: A deep copy creates a new object and also recursively copies all data from the original object, ensuring that the new object is completely independent of the original.

Behavior: Modifications to the deep copy will not affect the original array, because the data is stored separately in memory.

Usage in NumPy: You can create a deep copy using the np.copy() function.

Deep Copy:

Creates a new array object with its own separate data.

Modifications to the deep copy do not affect the original array.

Created with np.copy().

```
deep_copy = np.copy(original)
deep_copy[0, 0] = 20
print("Original Array:")
print(original)
print("Deep Copy:")
print(deep_copy)
```

```
Original Array:
[[10  2  3]
 [ 4  5  6]]
Deep Copy:
[[20  2  3]
 [ 4  5  6]]
```

ques 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.

```
import numpy as np
random_array = np.random.uniform(5, 20, size=(3, 3))
rounded_array = np.round(random_array, 2)
print("Random Array:")
print(random_array)
print("\nRounded Array:")
print(rounded_array)
```

```
Random Array:
[[ 6.27423949 12.14875882 12.53015381]
 [ 9.96256496 19.37449185  6.28620443]
 [14.61763374 14.05547731 18.19209007]]

Rounded Array:
[[ 6.27 12.15 12.53]
 [ 9.96 19.37  6.29]
 [14.62 14.06 18.19]]
```

ques 5. create a NumPy array with random integers between 1 and 10 of shape (5, 6). After creating the array perform the following operations:

a)Extract all even integers from array.

b))Extract all odd integers from array.

```
import numpy as np
random_array = np.random.randint(1, 11, size=(5, 6))
print("Random Array:")
print(random_array)
# a) Extract all even integers from the array
even_integers = random_array[random_array % 2 == 0]
print("\nEven Integers:")
print(even_integers)
# b) Extract all odd integers from the array
odd_integers = random_array[random_array % 2 != 0]
print("\nOdd Integers:")
print(odd_integers)
```

```
Random Array:
[[ 5  1 10  4  6  5]
 [ 9  4  6  8  1  9]
 [ 3 10  5  9  7  7]
 [ 9  7  9  5  2 10]
 [ 4  8  5  4 10  2]]

Even Integers:
[10  4  6  4  6  8 10  2 10  4  8  4 10  2]

Odd Integers:
[5 1 5 9 1 9 3 5 9 7 7 9 7 9 5 5]
```

ques 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

```
import numpy as np
array_3d = np.random.randint(1, 11, size=(3, 3, 3))
print("3D Array:")
print(array_3d)
# a) Find the indices of the maximum values along each depth level (third axis)
max_indices = np.argmax(array_3d, axis=2)
print("\nIndices of Maximum Values Along Each Depth Level:")
print(max_indices)
```

```
# b) Perform element-wise multiplication of the array with itself
elementwise_multiplication = array_3d * array_3d
print("\nElement-wise Multiplication of the Array:")
print(elementwise_multiplication)
```

```
3D Array:
[[[ 6  6  1]
  [ 2  7  5]
  [ 9  2  8]]

 [[ 7 10 10]
  [ 4 10  3]
  [ 7  1  9]]

 [[ 4 10  1]
  [ 9  4  4]
  [ 9 10 10]]]

Indices of Maximum Values Along Each Depth Level:
[[0 1 0]
 [1 1 2]
 [1 0 1]]

Element-wise Multiplication of the Array:
[[[ 36  36   1]
  [  4  49  25]
  [ 81   4  64]]

 [[ 49 100 100]
  [ 16 100   9]
  [ 49   1  81]]

 [[ 16 100   1]
  [ 81  16  16]
  [ 81 100 100]]]
```

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

```
import pandas as pd
import numpy as np
data = pd.read_csv('https://drive.google.com/uc?id=13x8f8HNKieSRAzxTIzAojaYp8Up8cefk')
data.head()
# Create a DataFrame
df = pd.DataFrame(data)
# Display the original DataFrame
print("Original DataFrame:")
print(df)
# Step 1: Clean the 'Phone' column
# Remove non-numeric characters
df['Phone'] = df['Phone'].str.replace(r'\D', '', regex=True)
# Step 2: Convert to numeric data type
df['Phone'] = pd.to_numeric(df['Phone'], errors='coerce')
# Display the cleaned DataFrame
print("\nCleaned DataFrame:")
print(df)
# Step 3: Display the DataFrame attributes and data types
print("\nDataFrame Attributes and Data Types:")
print(df.dtypes)
# Displaying the DataFrame as a NumPy array
numpy_array = df.to_numpy()
print("\nDataFrame as NumPy Array:")
print(numpy_array)
```

```
995        lyonsdaisy@example.net  2.177529e+08    05-01-1959
996       dariusbryan@example.com  1.149711e+13    06-10-2001
997        georgechan@example.org  1.750774e+15    13-05-1918
998           wanda04@example.net  9.152922e+09    31-08-1971
999       deannablack@example.org  7.975254e+13    24-01-1947

                          Job Title  Salary
0                 Probation officer   90000
1                            Dancer   80000
2                              Copy   50000
3           Counselling psychologist   65000
4                Biomedical engineer  100000
..                              ...     ...
995                Personnel officer   90000
996          Education administrator   50000
997  Commercial/residential surveyor   60000
998                 Ambulance person  100000
999       Nurse, learning disability   90000

[1000 rows x 10 columns]

DataFrame Attributes and Data Types:
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

DataFrame as NumPy Array:
[[1 '8717bbf45cCDbEe' 'Shelia' ... '27-01-2014' 'Probation officer' 90000]
 [2 '3d5AD30A4cD38ed' 'Jo' ... '26-07-1931' 'Dancer' 80000]
 [3 '810Ce0F276Badec' 'Sheryl' ... '25-11-2013' 'Copy' 50000]
 ...
 [998 '2adde51d8B8979E' 'Cathy' ... '13-05-1918'
  'Commercial/residential surveyor' 60000]
 [999 'Fb2FE369D1E171A' 'Jermaine' ... '31-08-1971' 'Ambulance person'
  100000]
 [1000 '8b756f6231DDC6e' 'Lee' ... '24-01-1947'
  'Nurse, learning disability' 90000]]
```

ques 8. Perform the following tasks using people dataset:

a) Read the 'data.csv' file using pandas, skipping the first 50 rows.

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

```
import pandas as pd
import numpy as np

# a) Read the 'data.csv' file using pandas, skipping the first 50 rows.
df = pd.read_csv('https://drive.google.com/uc?id=13x8f8HNKieSRAzxTIzAojaYp8Up8cefk',skiprows=50,header=None)

# b) Only read the columns: 'Last Name', 'Gender', 'Email', 'Phone', and 'Salary'.
filtered_df = df[[2, 4, 5, 6, 9]]

# c) Display the first 10 rows of the filtered dataset.
print("First 10 rows of the filtered dataset:")
print(filtered_df.head(10))

# d) Extract the 'Salary' column as a Series and display its last 5 values in numpy.
salary_series = filtered_df[9]
last_5_salaries = salary_series.tail(5).to_numpy()

print("\nLast 5 values of the 'Salary' column:")
print(last_5_salaries)
```

```
First 10 rows of the filtered dataset:
           2       4                            5                        6  \
0     George  Female  douglascontreras@example.net    +1-326-669-0118x4341
1         Jo    Male          pamela64@example.net  001-859-448-9935x54536
```

```
2     Joshua  Female     dianashepherd@example.net    001-274-739-8470x814
3     Rickey  Female     ingramtiffany@example.org        241.179.9509x498
4      Robyn    Male    carriecrawford@example.org       207.797.8345x6177
5  Christina    Male    fuentesclaudia@example.net    001-599-042-7428x143
6     Shelby    Male        kaneaudrey@example.org            663-280-5834
7      Steve    Male     rebekahsantos@example.net                     NaN
8       Gina  Female            craig28@example.com       125.219.3673x0076
9     Connie  Female    connercourtney@example.net      650-748-3069x64529

        9
0   70000
1   80000
2   70000
3   60000
4  100000
5   50000
6   85000
7   65000
8   60000
9   60000

Last 5 values of the 'Salary' column:
[ 90000  50000  60000 100000  90000]
```

ques 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 85000

```
import pandas as pd
import numpy as np

# Read the dataset (make sure to adjust the filename/path as necessary)
df = pd.read_csv('https://drive.google.com/uc?id=13x8f8HNKieSRAzxTIzAojaYp8Up8cefk')

# Filter the DataFrame based on the specified conditions
filtered_df = df[
    (df['Last Name'].str.contains('Duke', case=False)) &  # 'Last Name' contains 'Duke'
    (df['Gender'] == 'Female') &                           # 'Gender' is 'Female'
    (df['Salary'] < 85000)                                 # 'Salary' is less than 85000
]

# Convert the filtered DataFrame to a NumPy array
result_array = filtered_df.to_numpy()

# Display the filtered results
print("Filtered Results:")
print(result_array)
```

```
Filtered Results:
[[46 '99A502C175C4EBd' 'Olivia' 'Duke' 'Female' 'diana26@example.net'
  '001-366-475-8607x04350' '13-10-1934' 'Dentist' 60000]
 [211 'DF17975CC0a0373' 'Katrina' 'Duke' 'Female' 'robin78@example.com'
  '740.434.0212' '21-09-1935' 'Producer, radio' 50000]
 [458 'dcE1B7DE83c1076' 'Traci' 'Duke' 'Female'
  'perryhoffman@example.org' '+1-903-596-0995x489' '11-02-1997'
  'Herbalist' 50000]
 [730 'c9b482D7aa3e682' 'Lonnie' 'Duke' 'Female'
  'kevinkramer@example.net' '982.692.6257' '12-05-2015' 'Nurse, adult'
  70000]]
```

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

```
import pandas as pd
import numpy as np
random_integers = np.random.randint(1, 7, size=35)
# Create a DataFrame from the random integers, reshaping it to 7 rows and 5 columns
df = pd.DataFrame(random_integers.reshape(7, 5), columns=[f'Col {i+1}' for i in range(5)])
print("7x5 DataFrame:")
print(df)
```

```
7x5 DataFrame:
   Col 1  Col 2  Col 3  Col 4  Col 5
0      5      2      5      1      4
1      3      1      1      1      6
2      5      2      6      3      2
3      1      6      6      3      1
```

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

ques 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', etc

```
import pandas as pd
import numpy as np

# a) Create the first Series with random numbers ranging from 10 to 50
series1 = pd.Series(np.random.randint(10, 51, size=50))

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

# c) Create a DataFrame by joining these Series by column and rename the columns
df = pd.DataFrame({'col1': series1, 'col2': series2})
print("DataFrame:")
print(df)
```

```
DataFrame:
    col1  col2
0     27   615
1     26   893
2     35   595
3     27   579
4     37   213
5     34   784
6     50   431
7     33   295
8     31   620
9     29   254
10    11   478
11    34   796
12    26   359
13    48   673
14    23   410
15    31   205
16    32   769
17    42   450
18    38   185
19    18   399
20    24   900
21    14   502
22    13   645
23    12   779
24    18   860
25    23   930
26    46   326
27    40   602
28    31   436
29    11   182
30    13   226
31    20   478
32    34   171
33    27   687
34    46   805
35    48   235
36    34   490
37    13   568
38    49   810
39    17   586
40    50   297
41    35   491
42    45   522
43    26   430
44    45   770
45    17   347
46    17   960
47    10   927
48    26   574
49    33   301
```

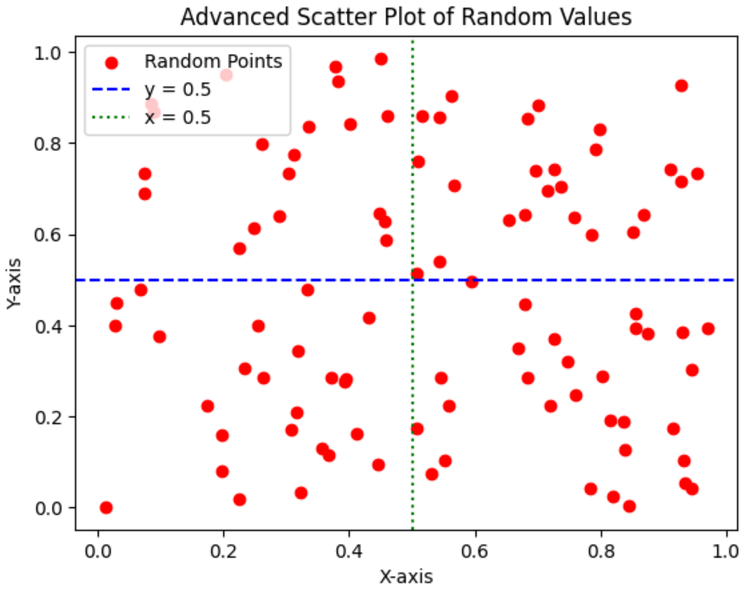
ques 12.Perform the following operations using people data set: a) Delete the 'Email', 'Phone', and 'Date of birth' columns from the dataset. b) Delete the rows containing any missing values. d) Print the final output also

```
import pandas as pd
import numpy as np
df = pd.read_csv('https://drive.google.com/uc?id=13x8f8HNKieSRAzxTIzAojaYp8Up8cefk')

# a) Delete the 'Email', 'Phone', and 'Date of birth' columns
df.drop(columns=['Email', 'Phone', 'Date of birth'], inplace=True)

# b) Delete the rows containing any missing values
df.dropna(inplace=True)

# d) Convert the final DataFrame to a NumPy array and print it
final_output = df.to_numpy()
print("Final Output as NumPy Array:")
print(final_output)
```

```
Final Output as NumPy Array:
[[1 '8717bbf45cCDbEe' 'Shelia' ... 'Male' 'Probation officer' 90000]
 [2 '3d5AD30A4cD38ed' 'Jo' ... 'Female' 'Dancer' 80000]
 [3 '810Ce0F276Badec' 'Sheryl' ... 'Female' 'Copy' 50000]
 ...
 [998 '2adde51d8B8979E' 'Cathy' ... 'Female'
  'Commercial/residential surveyor' 60000]
 [999 'Fb2FE369D1E171A' 'Jermaine' ... 'Male' 'Ambulance person' 100000]
 [1000 '8b756f6231DDC6e' 'Lee' ... 'Female' 'Nurse, learning disability'
  90000]]
```

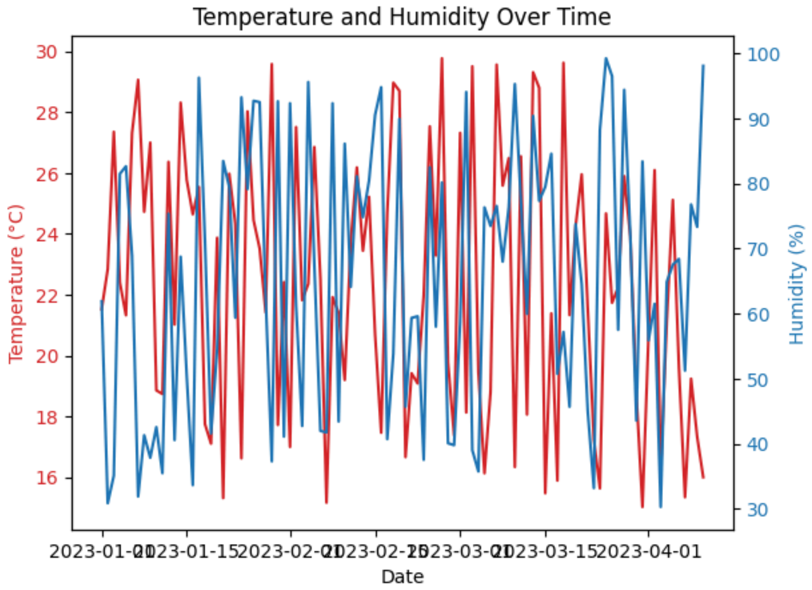
ques 13. Create two NumPy arrays, x and y, each containing 100 random float values between 0 and 1. Perform the following tasks using Matplotlib and NumPy: a) Create a scatter plot using x and y, setting the color of the points to red and the marker style to 'o'. b) Add a horizontal line at y = 0.5 using 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.

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

# Create two NumPy arrays, x and y, each containing 100 random float values between 0 and 1
x = np.random.rand(100)
y = np.random.rand(100)

# a) Create a scatter plot using x and y
plt.scatter(x, y, color='red', marker='o', label='Random Points')

# b) Add a horizontal line at y = 0.5
plt.axhline(y=0.5, color='blue', linestyle='--', label='y = 0.5')

# c) Add a vertical line at x = 0.5
plt.axvline(x=0.5, color='green', linestyle=':', label='x = 0.5')

# d) Label the x-axis and y-axis
plt.xlabel('X-axis')
plt.ylabel('Y-axis')

# e) Set the title of the plot
plt.title('Advanced Scatter Plot of Random Values')

# f) Display a legend
plt.legend()
plt.show()
```



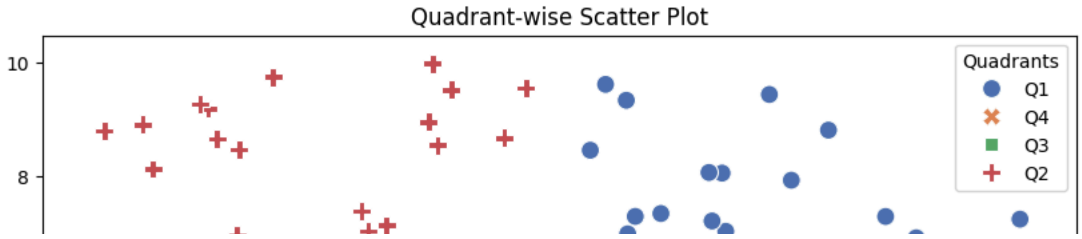
ques 14.Create a time-series dataset in a Pandas DataFrame with columns: 'Date', 'Temperature', 'Humidity' and Perform the following tasks using Matplotlib: right y-axis for 'Humidity'). b) Label the x-axis as 'Date'. a) Plot the 'Temperature' and 'Humidity' on the same plot with different y-axes (left y-axis for 'Temperature' and c) Set the title of the plot as 'Temperature and Humidity Over Time

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

# Create a time-series dataset
dates = pd.date_range(start='2023-01-01', periods=100)
temperature = np.random.uniform(low=15, high=30, size=100)  # Random temperatures between 15 and 30 degrees Celsius
humidity = np.random.uniform(low=30, high=100, size=100)    # Random humidity between 30% and 100%

# Create a DataFrame
df = pd.DataFrame({
    'Date': dates,
    'Temperature': temperature,
    'Humidity': humidity
})

# Plotting
fig, ax1 = plt.subplots()

# a) Plot 'Temperature' on the left y-axis
ax1.set_xlabel('Date')
ax1.set_ylabel('Temperature (°C)', color='tab:red')
ax1.plot(df['Date'], df['Temperature'], color='tab:red', label='Temperature')
ax1.tick_params(axis='y', labelcolor='tab:red')

# b) Create a second y-axis for 'Humidity'
ax2 = ax1.twinx()
ax2.set_ylabel('Humidity (%)', color='tab:blue')
ax2.plot(df['Date'], df['Humidity'], color='tab:blue', label='Humidity')
ax2.tick_params(axis='y', labelcolor='tab:blue')

# c) Set the title of the plot
plt.title('Temperature and Humidity Over Time')
plt.show()
```



ques 16.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'.

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

# Create two random arrays
np.random.seed(0)  # For reproducibility
x = np.random.rand(100) * 10  # Random values for x-axis
y = np.random.rand(100) * 10  # Random values for y-axis

# Create a DataFrame and define a column for quadrant
df = pd.DataFrame({'x': x, 'y': y})

# Determine the quadrant for each point
df['Quadrant'] = np.where((df['x'] >= 5) & (df['y'] >= 5), 'Q1',  # Quadrant 1
                          np.where((df['x'] < 5) & (df['y'] >= 5), 'Q2',  # Quadrant 2
                                   np.where((df['x'] < 5) & (df['y'] < 5), 'Q3',  # Quadrant 3
                                            'Q4')))  # Quadrant 4

# Create a scatter plot
plt.figure(figsize=(10, 6))
scatter = sns.scatterplot(data=df, x='x', y='y', hue='Quadrant', palette='deep', style='Quadrant', s=100)
plt.legend(title='Quadrants')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Quadrant-wise Scatter Plot')
plt.show()
```


Quadrant-wise Scatter Plot
10
8
Quadrants
Q1
Q4
Q3
Q2