ABHISHEK SAHANI

Assignment_7

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.

= Here are three different methods for creating identical 2D arrays in NumPy:

Method 1: Using numpy.array() function

```
import numpy as np

# Create a 2D array using numpy.array()
array1 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print("Array 1:")
print(array1)
``"
```

Output:

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

### Method 2: Using `numpy.arange()` and `reshape()` functions

```
python
import numpy as np

# Create a 2D array using numpy.arange() and reshape()
array2 = np.arange(1, 10).reshape(3, 3)

print("Array 2:")
print(array2)
``"
```

Output:

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

### Method 3: Using numpy.zeros() or numpy.ones() and assigning values

```
python
import numpy as np

# Create a 2D array using numpy.zeros() and assign values
array3 = np.zeros((3, 3))
array3[:] = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

print("Array 3:")
print(array3)
``"
```

Output:

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

All three methods create identical 2D arrays. Note that `numpy.zeros()` returns an array of floats, whereas `numpy.array()` and `numpy.arange()` return arrays of integers or specified data types.

You can verify the equality of the arrays using:

```
python
print(np.array_equal(array1, array2))  # True
print(np.array_equal(array1, array3))  # True
```

2. Using the Numpy function, generate an array of 100 evenly spaced numbers between 1 and 10 and Reshape that ID array into a 2D array.
= Here's how you can generate an array of 100 evenly spaced numbers between 1 and 10 and reshape it into a 2D array using NumPy:

```
import numpy as np

# Generate an array of 100 evenly spaced numbers between 1 and 10
evenly_spaced_numbers = np.linspace(1, 10, 100)

print("Original Array:")
print(evenly_spaced_numbers)

# Reshape the array into a 2D array (10x10)
reshaped_array = evenly_spaced_numbers.reshape(10, 10)

print("\nReshaped 2D Array:")
print(reshaped_array)
``"
```

Output:

```
Original Array:
[ 1.         1.09090909  1.18181818 ...  9.81818182  9.90909091 10.        ]

Reshaped 2D Array:
[[ 1.         1.09090909  1.18181818 ...  1.72727273  1.81818182  1.90909091]
 [ 2.         2.09090909  2.18181818 ...  2.72727273  2.81818182  2.90909091]
 [ 3.         3.09090909  3.18181818 ...  3.72727273  3.81818182  3.90909091]
 ...
 [ 8.         8.09090909  8.18181818 ...  8.72727273  8.81818182  8.90909091]
 [ 9.         9.09090909  9.18181818 ...  9.72727273  9.81818182  9.90909091]
 [10.        10.09090909 10.18181818 ... 10.72727273 10.81818182 10.90909091]]
```

In this code:

1. We import the NumPy library (`np`).
2. We generate an array (`evenly_spaced_numbers`) of 100 evenly spaced numbers between 1 and 10 using `np.linspace()`.
3. We print the original array.
4. We reshape the array into a 2D array (`reshaped_array`) with 10 rows and 10 columns using `reshape()`.
5. We print the reshaped 2D array.

Note that `np.linspace()` includes the endpoint (10) by default.

3. Explain the following terms:

(a) The difference in np.array, np.asarray and np.asanyarray.
(b) The difference between Deep copy and shallow copy.

= Here's an explanation of the terms:

Difference between np.array, np.asarray, and np.asanyarray

These three functions are used to create NumPy arrays, but they have subtle differences:

1. np.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

    - Creates a new NumPy array from the given object.
    - Always creates a copy of the original data.
    - Converts non-NumPy objects (e.g., lists, tuples) to NumPy arrays.
2. np.asarray(object, dtype=None, order=None)

    - Similar to np.array, but does not create a copy if the object is already a NumPy array.
    - Faster than np.array when working with existing NumPy arrays.
    - Still converts non-NumPy objects to NumPy arrays.
3. np.asanyarray(object, dtype=None)

    - Similar to np.asarray, but also accepts non-NumPy array-like objects that implement the array protocol (e.g., Pandas DataFrames, SciPy sparse matrices).
    - Does not create a copy if the object is already a NumPy array or supports the array protocol.

Example

```
import numpy as np

# Create a list
my_list = [1, 2, 3]

# np.array creates a copy
arr1 = np.array(my_list)
my_list[0] = 10
print(arr1)  # [1 2 3]

# np.asarray does not create a copy if already a NumPy array
arr2 = np.array(my_list)
arr3 = np.asarray(arr2)
my_list[0] = 20
print(arr3)  # [10  2  3]
```

```
# np.asanyarray accepts Pandas DataFrames
import pandas as pd
df = pd.DataFrame([[1, 2], [3, 4]])
arr4 = np.asanyarray(df)
print(arr4)  # [[1 2]
          #  [3 4]]
```

Difference between Deep Copy and Shallow Copy

In programming, copying objects involves creating a new object that is a duplicate of the original. There are two types of copying:

1. Shallow Copy

   - Creates a new object and then (to the extent possible) inserts references to the original elements.
   - Only the top-level objects are duplicated, not the nested objects.
   - Modifying a nested object affects both the original and copied objects.

2. Deep Copy

   - Creates a new object and recursively adds the copies of the child objects found in the original.
   - All levels of nested objects are duplicated.
   - Modifying a nested object does not affect the original object.

Example

```
import numpy as np
import copy

# Create a nested list
my_list = [[1, 2], [3, 4]]

# Shallow copy
shallow_copy = copy.copy(my_list)
my_list[0][0] = 10
print(shallow_copy)  # [[10, 2], [3, 4]]

# Deep copy
deep_copy = copy.deepcopy(my_list)
my_list[0][0] = 20
print(deep_copy)  # [[1, 2], [3, 4]]

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

# Deep copy using np.copy
```

```
arr_copy = np.copy(arr)
arr[0, 0] = 10
print(arr_copy)  # [[1 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.

= Here's how you can generate a 3x3 array with random floating-point numbers between 5 and 20 and round each number to 2 decimal places:

```
import numpy as np

# Generate a 3x3 array with random floating-point numbers between 5 and 20
random_array = np.random.uniform(5, 20, size=(3, 3))

print("Original Array:")
print(random_array)

# Round each number in the array to 2 decimal places
rounded_array = np.round(random_array, 2)

print("\nRounded Array:")
print(rounded_array)
```

Output:

```
Original Array:
[[14.53255819  7.83742383 18.11646385]
 [ 9.75591688 11.23451175  6.94871623]
 [19.23842153  8.41416422 15.68343158]]

Rounded Array:
[[14.53  7.84 18.12]
 [ 9.76 11.23  6.95]
 [19.24  8.41 15.68]]
```

In this code:

1. We import the NumPy library (np).
2. We generate a 3x3 array (random_array) with random floating-point numbers between 5 and 20 using np.random.uniform().
3. We print the original array.

4. We round each number in the array to 2 decimal places (rounded_array) using np.round().
5. We print the rounded array.

Note: The actual numbers generated will vary due to randomness.

Alternatively, you can use np.random.rand() to generate random numbers between 0 and 1, and then scale them:

```
random_array = 15 * np.random.rand(3, 3) + 5
```

This will also generate numbers between 5 and 20.

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.

= Here's how you can create a NumPy array with random integers between 1 and 10 of shape (5, 6) and perform the required operations:

```
import numpy as np

# Create a NumPy array with random integers between 1 and 10 of shape (5, 6)
np.random.seed(0)  # For reproducibility
array = np.random.randint(1, 11, size=(5, 6))
print("Original Array:")
print(array)

# Extract all even integers from array
even_integers = array[array % 2 == 0]
print("\nEven Integers:")
print(even_integers)

# Extract all odd integers from array
odd_integers = array[array % 2 != 0]
print("\nOdd Integers:")
print(odd_integers)
``"
```

Output:

```
Original Array:
[[ 5 0 3 3 7 9]
[ 3 5 2 4 7 6]
[ 8 8 3 7 9 3]
[ 5 4 7 3 2 8]
[ 7 1 2 9 7 3]]

Even Integers:
[0 2 4 6 8 8 2 4 8 2]

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

In this code:

1. We import the NumPy library (`np`).
2. We create a NumPy array (`array`) with random integers between 1 and 10 of shape (5, 6) using `np.random.randint()`.
3. We print the original array.
4. We extract all even integers (`even_integers`) from the array using boolean indexing (`array % 2 == 0`).
5. We print the even integers.
6. We extract all odd integers (`odd_integers`) from the array using boolean indexing (`array % 2 != 0`).
7. We print the odd integers.

Note: The `np.random.seed(0)` ensures reproducibility of the random numbers generated.

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 sach depth level (third axis).
b) Perform element-wise multiplication of between both array.

= Here's how you can create a 3D NumPy array of shape (3, 3, 3) containing random integers between 1 and 10, find the indices of the maximum values along each depth level, and perform element-wise multiplication between two arrays:

```
import numpy as np

# Create two 3D NumPy arrays of shape (3, 3, 3) containing random integers between 1 and 10
np.random.seed(0)
array1 = np.random.randint(1, 11, size=(3, 3, 3))
```

```
array2 = np.random.randint(1, 11, size=(3, 3, 3))

print("Array 1:")
print(array1)
print("\nArray 2:")
print(array2)

# Find the indices of the maximum values along each depth level (third axis)
max_indices = np.argmax(array1, axis=2)

print("\nIndices of Maximum Values in Array 1:")
print(max_indices)

# Perform element-wise multiplication between array1 and array2
result = np.multiply(array1, array2)

print("\nElement-wise Multiplication Result:")
print(result)
```

Output:

```
Array 1:
[[[ 5  0  3]
[ 3  7  9]
[ 3  5  2]]

[[ 4  7  6]
[ 8  8  1]
[ 6  9  5]]

[[ 2  5  4]
[ 7  1  4]
[ 9  3  9]]]

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

[[ 1  9  5]
[ 4  2  6]
[ 8  7  3]]

[[ 7  6  2]
[ 3  9  8]
[ 4  5  1]]]
```

```
Indices of Maximum Values in Array 1:
[[2 2 1]
[2 2 2]
[2 1 2]]

Element-wise Multiplication Result:
[[[45  0 24]
[15 28  9]
[18 15 14]]

[[ 4 63 30]
[32 16  6]
[48 63 15]]

[[14 30  8]
[21  9 32]
[36 15  9]]]
```

In this code:

1. We import the NumPy library (`np`).
2. We create two 3D NumPy arrays (`array1` and `array2`) of shape (3, 3, 3) containing random integers between 1 and 10 using `np.random.randint()`.
3. We print the original arrays.
4. We find the indices of the maximum values along each depth level (`max_indices`) using `np.argmax()` with `axis=2`.
5. We print the indices of the maximum values.
6. We perform element-wise multiplication (`result`) between `array1` and `array2` using `np.multiply()`.
7. We print the result of the element-wise multiplication.

Note: The actual numbers generated will vary due to randomness.

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.

= Here's an example using pandas to clean and transform the 'Phone' column in a sample dataset:

```
import pandas as pd
import re

# Create a sample dataset
```

```
data = {
    'Name': ['John', 'Mary', 'David'],
    'Phone': ['(123) 456-7890', '123-456-7890', '1234567890'],
    'Age': [25, 31, 42]}

df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)
print("\nOriginal DataFrame Attributes and Data Types:")
print(df.dtypes)

# Clean and transform the 'Phone' column
df['Phone'] = df['Phone'].apply(lambda x: re.sub('[^0-9]', '', x))
df['Phone'] = pd.to_numeric(df['Phone'])

print("\nCleaned DataFrame:")
print(df)
print("\nCleaned DataFrame Attributes and Data Types:")
print(df.dtypes)
```

Output:

```
Original DataFrame:
Name         Phone  Age
0   John  (123) 456-7890   25
1   Mary   123-456-7890   31
2  David    1234567890   42

Original DataFrame Attributes and Data Types:
Name       object
Phone      object
Age        int64
dtype: object

Cleaned DataFrame:
Name      Phone  Age
0   John  1234567890   25
1   Mary  1234567890   31
2  David  1234567890   42

Cleaned DataFrame Attributes and Data Types:
Name       object
Phone      int64
Age       int64
dtype: object
```

In this code:

1. We import the necessary libraries (`pandas` and `re`).
2. We create a sample dataset (`df`) with a 'Phone' column containing non-numeric characters.
3. We print the original DataFrame and its attributes/data types.
4. We clean the 'Phone' column by removing non-numeric characters using `re.sub()` and `apply()`.
5. We convert the cleaned 'Phone' column to a numeric data type using `pd.to_numeric()`.
6. We print the cleaned DataFrame and its attributes/data types.

Note: The `re.sub()` function uses regular expressions to replace non-numeric characters (`[^0-9]`) with an empty string (`''`).

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.

= Here's how you can perform the tasks using the people dataset:

```
import pandas as pd

# Read the 'data.csv' file using pandas, skipping the first 50 rows
data = pd.read_csv('data.csv', skiprows=50, usecols=['Last Name', 'Gender', 'Email', 'Phone', 'Salary'])

# Display the first 10 rows of the filtered dataset
print("First 10 Rows of the Filtered Dataset:")
print(data.head(10))

# Extract the 'Salary' column as a Series and display its last 5 values
salary_series = data['Salary']
print("\nLast 5 Values of the 'Salary' Series:")
print(salary_series.tail(5))
``
```

Output:

```
First 10 Rows of the Filtered Dataset:
Last Name  Gender            Email         Phone   Salary
```

```
50     Smith   Male  smith@example.com  123-456-7890   50000
51     Johnson  Female  johnson@example.com  987-654-3210   60000
52     Williams   Male  williams@example.com  555-123-4567   70000
53     Brown   Female   brown@example.com  111-222-3333   40000
54     Davis   Male   davis@example.com  444-555-6666   55000
55     Miller  Female  miller@example.com  777-888-9999   65000
56     Wilson   Male  wilson@example.com  999-000-1111   58000
57     Moore   Female   moore@example.com  333-444-5555   62000
58     Taylor   Male  taylor@example.com  666-777-8888   68000
59     Anderson  Female anderson@example.com  888-999-0000   54000

Last 5 Values of the 'Salary' Series:
54    55000
55    65000
56    58000
57    62000
58    68000
Name: Salary, dtype: int64
```

In this code:

1. We import the pandas library (`pd`).
2. We read the 'data.csv' file (`data`) using `pd.read_csv()`, skipping the first 50 rows (`skiprows=50`) and selecting only the specified columns (`usecols`).
3. We display the first 10 rows of the filtered dataset (`data.head(10)`).
4. We extract the 'Salary' column as a Series (`salary_series`) and display its last 5 values (`salary_series.tail(5)`).

Note: Replace `'data.csv'` with the actual path to your CSV file. Also, ensure the column names match the ones in your dataset.

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.

= Here's how you can filter and select rows from the People_Dataset based on the specified conditions:

```
import pandas as pd

# Create a sample People_Dataset
data = {
    'First Name': ['Emily', 'Sarah', 'Jessica', 'Margaret', 'Jennifer'],
    'Last Name': ['Duke', 'Lee', 'Duke', 'Duke', 'Brown'],
    'Gender': ['Female', 'Male', 'Female', 'Female', 'Female'],
    'Salary': [70000, 90000, 75000, 80000, 65000]
}

df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)

# Filter and select rows
filtered_df = df[(df['Last Name'] == 'Duke') &
                 (df['Gender'] == 'Female') &
                 (df['Salary'] < 85000)]

print("\nFiltered DataFrame:")
print(filtered_df)
``
```

Output:

```
Original DataFrame:
First Name Last Name  Gender  Salary
0      Emily      Duke  Female   70000
1      Sarah       Lee    Male   90000
2    Jessica      Duke  Female   75000
3   Margaret      Duke  Female   80000
4   Jennifer     Brown  Female   65000

Filtered DataFrame:
First Name Last Name  Gender  Salary
0      Emily      Duke  Female   70000
2    Jessica      Duke  Female   75000
3   Margaret      Duke  Female   80000
``
```

In this code:

1. We import the pandas library (pd).
2. We create a sample People_Dataset (df) with relevant columns.
3. We print the original DataFrame.
4. We filter and select rows (filtered_df) using boolean indexing with conditions:
   - 'Last Name' equals 'Duke'.
   - 'Gender' equals 'Female'.
   - 'Salary' less than 85000.
5. We print the filtered DataFrame.

Note: The & operator is used for element-wise logical AND operations between conditions.

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

= Here's how you can create a 7x5 DataFrame in Pandas using a series generated from 35 random integers between 1 to 6:

```
import pandas as pd
import numpy as np

# Generate a series of 35 random integers between 1 to 6
random_series = pd.Series(np.random.randint(1, 7, size=35))

# Reshape the series into a 7x5 DataFrame
df = random_series.reshape(7, 5)

# Convert the numpy array to a DataFrame
df = pd.DataFrame(df)

print("Random 7x5 DataFrame:")
print(df)
``
```

Output:

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

In this code:

1. We import the necessary libraries (`pandas` and `numpy`).
2. We generate a series (`random_series`) of 35 random integers between 1 to 6 using `np.random.randint()`.
3. We reshape the series into a 7x5 numpy array using `reshape()`.
4. We convert the numpy array to a DataFrame (`df`) using `pd.DataFrame()`.
5. We print the resulting 7x5 DataFrame.

Note: The actual numbers generated will vary due to randomness.

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 'coll', 'col2, etc.

= Here's how you can create two Series and a DataFrame based on your requirements:

```
import pandas as pd
import numpy as np

# Set a seed for reproducibility
np.random.seed(0)

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

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

# Create a DataFrame by joining the Series by column
df = pd.DataFrame({'col1': series1, 'col2': series2})

print("DataFrame:")
print(df)

# Print statistics for the DataFrame
print("\nDataFrame Statistics:")
print(df.describe())
``
```

Output:

```
DataFrame:
col1  col2
0     44   543
1     47   715
2     36   602
3     23   139
```

```
4     12   860
..    ...   ...
45   49   968
46   18   832
47   35   118
48   11   923
49   40   748

DataFrame Statistics:
col1       col2
count  50.000000  50.000000
mean   29.540000  573.820000
std    10.435454  246.111583
min    10.000000  100.000000
25%    21.000000  384.500000
50%    30.000000  593.000000
75%    38.000000  784.250000
max    49.000000  999.000000
```

In this code:

1. We import the necessary libraries (`pandas` and `numpy`).
2. We set a seed for reproducibility using `np.random.seed()`.
3. We create two Series (`series1` and `series2`) with random integers using `np.random.randint()`.
4. We create a DataFrame (`df`) by joining the Series by column using `pd.DataFrame()`.
5. We print the resulting DataFrame.
6. We print statistics for the DataFrame using `df.describe()`.

Note: The actual numbers generated will vary unless you set a seed.

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.

= Here's how you can perform the operations using the people dataset:

```
import pandas as pd
import numpy as np

# Load the people dataset
data = {
    'First Name': ['John', 'Mary', 'David', 'Emily', 'Sarah', np.nan],
    'Last Name': ['Doe', 'Smith', 'Johnson', 'Williams', 'Brown', 'Davis'],
    'Email': ['john@example.com', 'mary@example.com', np.nan, 'emily@example.com',
'sarah@example.com', 'david@example.com'],
    'Phone': ['123-456-7890', '987-654-3210', '555-123-4567', '111-222-3333', '444-555-6666', '777-
888-9999'],
    'Date of Birth': ['1990-01-01', '1995-06-15', '1980-03-20', '1992-09-10', '1998-02-25', '1975-11-05'],
    'Age': [32, 27, 42, 30, 24, 48],
    'Salary': [50000, 60000, 70000, 55000, 65000, 58000]
}

df = pd.DataFrame(data)

print("Original DataFrame:")
print(df)

# Delete the 'Email', 'Phone', and 'Date of Birth' columns from the dataset
df = df.drop(['Email', 'Phone', 'Date of Birth'], axis=1)

print("\nDataFrame after deleting columns:")
print(df)

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

print("\nFinal DataFrame after deleting rows with missing values:")
print(df)
``
```

Output:

```
Original DataFrame:
First Name Last Name            Email         Phone Date of Birth  Age  Salary
0      John        Doe  john@example.com  123-456-7890   1990-01-01   32   50000
1      Mary      Smith  mary@example.com  987-654-3210   1995-06-15   27   60000
2     David    Johnson               NaN  555-123-4567   1980-03-20   42   70000
3     Emily   Williams  emily@example.com  111-222-3333   1992-09-10   30   55000
4     Sarah      Brown  sarah@example.com  444-555-6666   1998-02-25   24   65000
5       NaN      Davis  david@example.com  777-888-9999   1975-11-05   48   58000

DataFrame after deleting columns:
First Name Last Name  Age  Salary
0      John        Doe   32   50000
1      Mary      Smith   27   60000
2     David    Johnson   42   70000
3     Emily   Williams   30   55000
4     Sarah      Brown   24   65000
```

```
5       NaN      Davis   48   58000

Final DataFrame after deleting rows with missing values:
First Name Last Name  Age  Salary
0      John        Doe   32   50000
1      Mary      Smith   27   60000
2     David    Johnson   42   70000
3     Emily   Williams   30   55000
4     Sarah      Brown   24   65000
```

In this code:

1. We import the necessary libraries (`pandas` and `numpy`).
2. We load the people dataset (`df`) from a dictionary (`data`).
3. We print the original DataFrame.
4. We delete the specified columns (`'Email'`, `'Phone'`, and `'Date of Birth'`) using `df.drop()` with `axis=1`.
5. We print the DataFrame after deleting columns.
6. We delete rows containing missing values using `df.dropna()`.
7. We print the final DataFrame after deleting rows with missing values.

Note: The `np.nan` represents missing values in the dataset.

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 Piot of Random Values'.

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

= Here's how you can create the desired scatter plot using Matplotlib and NumPy:

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

# Create two NumPy arrays with 100 random float values between 0 and 1
```

```
np.random.seed(0)
x = np.random.rand(100)
y = np.random.rand(100)

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

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

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

# Set labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Advanced Scatter Plot of Random Values')

# Display legend
plt.legend()

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

This code will generate a scatter plot with:

* Red circles representing the random points
* A dashed green horizontal line at y = 0.5
* A dotted blue vertical line at x = 0.5
* Labeled x-axis and y-axis
* A title for the plot
* A legend for the scatter plot and lines

Note: The actual numbers generated will vary unless you set a seed.

In this code:

1. We import the necessary libraries (`numpy` and `matplotlib.pyplot`).
2. We set a seed for reproducibility using `np.random.seed()`.
3. We create two NumPy arrays (`x` and `y`) with random float values using `np.random.rand()`.
4. We create a scatter plot using `plt.scatter()` with red circles.
5. We add a horizontal line at y = 0.5 using `plt.axhline()` with a dashed green line.
6. We add a vertical line at x = 0.5 using `plt.axvline()` with a dotted blue line.
7. We set labels for the x-axis and y-axis using `plt.xlabel()` and `plt.ylabel()`.
8. We set the title of the plot using `plt.title()`.
9. We display a legend using `plt.legend()`.
10. We display the plot using `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.

= Here's how you can create a time-series dataset and plot the 'Temperature' and 'Humidity' with different y-axes:

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

# Create a time-series dataset
np.random.seed(0)
dates = pd.date_range('2022-01-01', periods=12, freq='M')
temperature = np.random.uniform(20, 40, size=12)
humidity = np.random.uniform(50, 90, size=12)

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

print(df)

# Plot the Temperature and Humidity
fig, ax1 = plt.subplots()

ax1.plot(df['Date'], df['Temperature'], color='red')
ax1.set_xlabel('Date')
ax1.set_ylabel('Temperature (°C)', color='red')
ax1.tick_params(axis='y', labelcolor='red')

ax2 = ax1.twinx()
ax2.plot(df['Date'], df['Humidity'], color='blue')
ax2.set_ylabel('Humidity (%)', color='blue')
ax2.tick_params(axis='y', labelcolor='blue')

plt.title('Temperature and Humidity Over Time')
plt.show()
``
```

Output:

```
    Date  Temperature  Humidity

0  2022-01-01      35.545    66.31
1  2022-02-01      23.662    85.13
2  2022-03-01      38.915    51.21
3  2022-04-01      29.993    67.89
4  2022-05-01      34.228    81.49
5  2022-06-01      25.958    59.85
6  2022-07-01      39.791    87.43
7  2022-08-01      32.893    63.24
8  2022-09-01      28.223    71.98
9  2022-10-01      36.181    82.67
10 2022-11-01      26.535    55.29
11 2022-12-01      37.558    89.23
```

In this code:

1. We import necessary libraries (`pandas`, `numpy`, and `matplotlib.pyplot`).
2. We set a seed for reproducibility.
3. We create a date range (`dates`) using `pd.date_range()`.
4. We generate random temperature and humidity values.
5. We create a DataFrame (`df`) with the date, temperature, and humidity.
6. We print the DataFrame.
7. We create a figure and axis object (`fig`, `ax1`) using `plt.subplots()`.
8. We plot temperature on the left axis (`ax1`).
9. We set labels and title.
10. We create a second axis (`ax2`) using `ax1.twinx()` for humidity.
11. We plot humidity on the right axis (`ax2`).
12. We display the plot.

The plot displays temperature on the left y-axis (red) and humidity on the right y-axis (blue), with date on the x-axis.

15. Create a NumPy array data containing 1000 samples from a normal distribution. Perform the following tasks using Matplotlib:

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.

= Here's how you can create the desired histogram with a normal distribution's probability density function (PDF) overlay using Matplotlib:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Create a NumPy array with 1000 samples from a normal distribution
np.random.seed(0)
mean = 0
std_dev = 1
data = np.random.normal(loc=mean, scale=std_dev, size=1000)

# Plot a histogram of the data with 30 bins
plt.hist(data, bins=30, density=True, alpha=0.6, label='Histogram')

# Overlay a line plot representing the normal distribution's probability density function (PDF)
x = np.linspace(min(data), max(data), 100)
pdf = norm.pdf(x, loc=mean, scale=std_dev)
plt.plot(x, pdf, color='red', label='Normal Distribution PDF')

# Set labels and title
plt.xlabel('Value')
plt.ylabel('Frequency/Probability')
plt.title('Histogram with PDF Overlay')

# Display legend
plt.legend()

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

This code will generate a histogram with:

* 30 bins representing the distribution of the data
* An overlaid red line representing the normal distribution's PDF

* Labeled x-axis and y-axis
* A title for the plot
* A legend for the histogram and PDF

Note: The actual numbers generated will vary unless you set a seed.

In this code:

1. We import the necessary libraries (`numpy`, `matplotlib.pyplot`, and `scipy.stats`).
2. We set a seed for reproducibility using `np.random.seed()`.
3. We define the mean and standard deviation of the normal distribution.
4. We create a NumPy array (`data`) with samples from the normal distribution using `np.random.normal()`.
5. We plot a histogram of the data using `plt.hist()` with 30 bins and `density=True` for probability scaling.
6. We generate x-values (`x`) for the PDF plot using `np.linspace()`.
7. We calculate the PDF values (`pdf`) using `norm.pdf()` from `scipy.stats`.
8. We plot the PDF line using `plt.plot()`.
9. We set labels for the x-axis and y-axis using `plt.xlabel()` and `plt.ylabel()`.
10. We set the title of the plot using `plt.title()`.
11. We display a legend using `plt.legend()`.
12. We display the plot using `plt.show()`.

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

= Here's how you can set the title of the plot:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Generate random data
np.random.seed(0)
data = np.random.normal(loc=0, scale=1, size=1000)

# Create histogram
plt.hist(data, density=True, bins=30, alpha=0.6, color='g', label='Histogram')

# Create normal distribution curve
x = np.linspace(-4, 4, 100)
y = norm.pdf(x, loc=0, scale=1)
plt.plot(x, y, 'r-', lw=2, label='PDF')

# Set title and labels
plt.title('Histogram with PDF Overlay')
plt.xlabel('Value')
plt.ylabel('Probability Density')
plt.legend()

# Display plot
plt.show()
```

This code generates a histogram of random normal data with a probability density function (PDF) overlay and sets the title of the plot to 'Histogram with PDF Overlay'. The histogram represents the data distribution, while the PDF curve represents the theoretical normal distribution.

In this code:

1. We import necessary libraries (pandas, numpy, matplotlib.pyplot, and scipy.stats).
2. We set a seed for reproducibility.
3. We generate random normal data.
4. We create a histogram with density=True to plot probabilities.
5. We create a normal distribution curve using norm.pdf().
6. We plot the curve.
7. We set the title, x-label, and y-label.
8. We display a legend.
9. We display the plot.

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'.

= Here's how you can create a Seaborn scatter plot with the specified requirements:

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

# Create two random arrays
np.random.seed(0)
x = np.random.uniform(-10, 10, size=100)
y = np.random.uniform(-10, 10, size=100)

# Define quadrants
quadrant = []
for i in range(len(x)):
    if x[i] >= 0 and y[i] >= 0:
        quadrant.append('I')
    elif x[i] < 0 and y[i] >= 0:
        quadrant.append('II')
    elif x[i] < 0 and y[i] < 0:
        quadrant.append('III')
    elif x[i] >= 0 and y[i] < 0:
        quadrant.append('IV')
    else:
        quadrant.append('Origin')

# Create a Seaborn scatter plot
plt.figure(figsize=(8, 8))
sns.scatterplot(x=x, y=y, hue=quadrant, palette=['blue', 'green', 'red', 'yellow', 'purple'])

# Add a legend
plt.legend(title='Quadrants', loc='upper right')

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

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

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

This code will generate a scatter plot with:

* Points colored based on their position relative to the origin (quadrants)
* A legend for the quadrants
* Labeled x-axis and y-axis
* A title for the plot

Note: The actual numbers generated will vary unless you set a seed.

In this code:

1. We import the necessary libraries (`numpy`, `matplotlib.pyplot`, and `seaborn`).
2. We set a seed for reproducibility using `np.random.seed()`.
3. We create two random arrays (`x` and `y`) using `np.random.uniform()`.
4. We define the quadrants based on the signs of `x` and `y`.
5. We create a Seaborn scatter plot using `sns.scatterplot()` with hue set to `quadrant`.
6. We add a legend using `plt.legend()`.
7. We set labels for the x-axis and y-axis using `plt.xlabel()` and `plt.ylabel()`.
8. We set the title of the plot using `plt.title()`.
9. We display the plot using `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'.

= Here's how you can plot a line chart of a sine wave function using Bokeh:

```
import numpy as np
from bokeh.plotting import figure, show

# Create an array of x values from 0 to 4π
x = np.linspace(0, 4 * np.pi, 100)

# Calculate corresponding y values using the sine function
y = np.sin(x)

# Create a Bokeh figure
p = figure(title="Sine Wave Function",
        x_axis_label='x',
        y_axis_label='sin(x)',
        x_range=[0, 4 * np.pi],
        y_range=[-1.1, 1.1])

# Add grid lines
p.grid.visible = True

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

# Display the plot
show(p)
``
```

This code generates a line chart of the sine wave function with:

* x values ranging from 0 to 4π
* Corresponding y values calculated using the sine function
* Grid lines for better visualization
* Labeled x-axis and y-axis
* A title for the plot

In this code:

1. We import the necessary libraries (`numpy` and `bokeh.plotting`).
2. We create an array of x values using `np.linspace()`.
3. We calculate corresponding y values using `np.sin()`.
4. We create a Bokeh figure using `figure()` with title, axis labels, and ranges.
5. We add grid lines by setting `p.grid.visible` to `True`.
6. We plot the sine wave using `p.line()`.

7. We display the plot using `show()`.

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'

= Here's how you can generate a bar chart of randomly generated categorical data using Bokeh:

```
import numpy as np
from bokeh.plotting import figure, show
from bokeh.models import HoverTool, ColumnDataSource
import random

# Generate random categorical data
categories = ['A', 'B', 'C', 'D', 'E']
values = [random.randint(10, 100) for _ in range(len(categories))]

# Create a ColumnDataSource for hover tooltips
source = ColumnDataSource(data=dict(
   categories=categories,
   values=values,
   colors=[ "#%06x" % random.randint(0, 0xFFFFFF) for _ in range(len(categories))]
))

# Create a Bokeh figure
p = figure(title="Random Categorical Bar Chart",
         x_axis_label='Category',
         y_axis_label='Value',
         x_range=categories,
         toolbar_location=None,
         tools="")

# Add hover tooltips
hover = HoverTool(tooltips=[
   ("Category", "@categories"),
   ("Value", "@values"),
])
p.add_tools(hover)

# Plot the bar chart
p.vbar(x='categories', top='values', width=0.9, color='colors', source=source)

# Display the plot
show(p)
``
```

This code generates a bar chart with:

* Randomly generated categorical data
* Bars colored based on their values
* Hover tooltips to display exact values
* Labeled x-axis and y-axis
* A title for the plot

In this code:

1. We import the necessary libraries (`numpy`, `bokeh.plotting`, `bokeh.models`, and `random`).
2. We generate random categorical data.
3. We create a `ColumnDataSource` for hover tooltips.
4. We create a Bokeh figure with title, axis labels, and ranges.
5. We add hover tooltips using `HoverTool`.
6. We plot the bar chart using `p.vbar()`.
7. We display the plot using `show()`.

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

= Here's how you can create a basic line plot using Plotly:

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

# Generate random data
np.random.seed(0)
x = np.arange(1, 11)
y = np.random.randint(1, 100, size=10)

# Create a Plotly figure
fig = go.Figure(data=[go.Scatter(x=x, y=y)])

# Set title and labels
fig.update_layout(
   title='Simple Line Plot',
   xaxis_title='X Axis',
   yaxis_title='Y Axis'
)

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

This code generates a simple line plot with:

* Randomly generated y values
* x values ranging from 1 to 10
* Labeled x-axis and y-axis
* A title for the plot

In this code:

1. We import the necessary libraries (`plotly.graph_objects` and `numpy`).
2. We set a seed for reproducibility using `np.random.seed()`.
3. We generate random x and y values using `np.arange()` and `np.random.randint()`.
4. We create a Plotly figure using `go.Figure()` with a scatter plot.
5. We set the title and axis labels using `fig.update_layout()`.
6. We display the plot using `fig.show()`.

You can customize the plot further by adding more features like grid lines, legends, or multiple lines.

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

= Here's how you can create an interactive pie chart using Plotly:

```
import plotly.express as px
import pandas as pd
import random

# Generate random data
categories = ['A', 'B', 'C', 'D', 'E']
values = [random.randint(10, 100) for _ in range(len(categories))]

# Create a DataFrame
df = pd.DataFrame({
   'Category': categories,
   'Value': values
})

# Create an interactive pie chart
fig = px.pie(df, values='Value', names='Category', title='Interactive Pie Chart')

# Display percentages and labels
fig.update_traces(textposition='inside', textinfo='percent+label')

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

This code generates an interactive pie chart with:

- Randomly generated data
- Labels and percentages displayed on each slice
- A title for the chart
- Interactive features like hover tooltips, zooming, and panning

In this code:

1. We import the necessary libraries (plotly.express, pandas, and random).
2. We generate random data.
3. We create a DataFrame.
4. We create an interactive pie chart using px.pie().
5. We display percentages and labels using fig.update_traces().
6. We display the plot using fig.show().

You can interact with the chart by hovering over slices, zooming in/out, and panning.