#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 np.array()

You can create a 2D array by directly specifying the list of lists.

```
import numpy as np
array1 = np.array([[1, 2, 3], [4, 5, 6]])
print("Array 1:")
print(array1)
```

```
Array 1:
[[1 2 3]
 [4 5 6]]
```

Method 2: Using np.zeros()

You can initialize an array with zeros and then fill it with specific values.

```
import numpy as np
array2 = np.zeros((2, 3), dtype=int)
array2[0] = [1, 2, 3]
array2[1] = [4, 5, 6]
print("Array 2:")
print(array2)
```

```
Array 2:
[[1 2 3]
 [4 5 6]]
```

Method 3: Using np.full()

You can create an array initialized with a specific value and then modify it.

```
import numpy as np
array3 = np.full((2, 3), 0, dtype=int)
array3[0] = [1, 2, 3]
array3[1] = [4, 5, 6]
print("Array 3:")
print(array3)
```

```
Array 3:
[[1 2 3]
 [4 5 6]]
```

#2.Using the Numpy function, generate an array of wRR evenly spaced numPers Petween w and wR and Reshape that wD array into a 2D array?

To generate an array of evenly spaced numbers between two values and then reshape it into a 2D array, you can use the numpy.linspace() function followed by the reshape() method. Here's a step-by-step example:

Example Code

Let's assume you want to generate an array of n evenly spaced numbers between start and end, and then reshape it into a 2D array with rows rows and cols columns.

```
import numpy as np
start = 1
end = 10
n = 12
rows = 3
cols = 4

array_1d = np.linspace(start, end, n)

array_2d = array_1d.reshape((rows, cols))

print("2D Array:")
print(array_2d)
```

```
2D Array:
[[ 1.          1.81818182  2.63636364  3.45454545]
 [ 4.27272727  5.09090909  5.90909091  6.72727273]
 [ 7.54545455  8.36363636  9.18181818 10.        ]]
```

Explanation

np.linspace(start, end, n) generates n evenly spaced numbers between start and end.

.reshape((rows, cols)) reshapes the 1D array into a 2D array with the specified number of rows and columns.

#3.Explain the following terms

#The difference in npYarray, npYasarray and npYasanyarrayX

#The difference between Deep copy and shallow copyX

1. np.array, np.asarray, and np.asanyarray

These are all functions provided by NumPy for creating arrays, but they have some subtle differences:

np.array():

Purpose: Creates a new NumPy array from any object exposing the array interface.

Behavior: Always creates a new array. This means that it copies the data and returns a new array object, regardless of whether the input object is already a NumPy array.

```
# Example
import numpy as np
arr = np.array([1, 2, 3])
```

np.asarray():

Purpose: Converts an input to an array, but if the input is already a NumPy array, it returns the same array without making a copy.

Behavior: This function avoids copying data if the input is already an array, which can be more efficient.

```
import numpy as np
arr1 = np.asarray([1, 2, 3])
arr2 = np.asarray(arr1)
```

np.asanyarray():

Purpose: Similar to np.asarray(), but it also converts subclasses of ndarray to ndarray while preserving the subclass type.

Behavior: It will return a view of the input if the input is already an ndarray or a subclass thereof, but if the input is a different type, it will create a new array.

```
import numpy as np
class MyArray(np.ndarray): pass

arr1 = MyArray([1, 2, 3])
arr2 = np.asanyarray(arr1)
```

1. Deep Copy vs. Shallow Copy Shallow Copy:

Definition: A shallow copy creates a new object, but does not create copies of nested objects. Instead, it copies references to the nested objects. This means that changes to the nested objects will be reflected in both the original and the copied object.

In Python: You can create a shallow copy of a list using the copy() method or the copy module's copy() function.

```
import copy

original_list = [[1, 2, 3], [4, 5, 6]]
shallow_copy = copy.copy(original_list)
```

```
shallow_copy[0][0] = 10

print(original_list)
```

```
[[10, 2, 3], [4, 5, 6]]
```

Deep Copy:

Definition: A deep copy creates a new object and recursively copies all nested objects, ensuring that the original and copied objects are completely independent. Changes to nested objects in the copied object will not affect the original object.

In Python: You can create a deep copy of a list using the copy module's deepcopy() function.

```
import copy

original_list = [[1, 2, 3], [4, 5, 6]]
deep_copy = copy.deepcopy(original_list)
deep_copy[0][0] = 10

print(original_list)
```

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

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

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

print("Original array:")
print(array)

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

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

```
Original array:
[[15.47295778  6.49363062 18.92937797]
 [14.60727888  8.26810446  6.4391731 ]
 [ 7.60646504 10.92526603 16.45807699]]
```

```
Rounded array:
[[15.47  6.49 18.93]
 [14.61  8.27  6.44]
 [ 7.61 10.93 16.46]]
```

In this example, np.random.uniform(5, 20, (3, 3)) generates a 3x3 array with random floating-point numbers between 5 and 20. Then, np.round(array, 2) rounds each number in the array to 2 decimal places

1. 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

array = np.random.randint(1, 11, (5, 6))

print("Original array:")
print(array)


even_integers = array[array % 2 == 0]

print("\nEven integers:")
print(even_integers)

odd_integers = array[array % 2 != 0]

print("\nOdd integers:")
print(odd_integers)
```

```
Original array:
[[ 4 10  4  1  1  2]
 [ 3  1  6  8  5  9]
 [ 2  9  5  1  2  8]
 [ 7  2  7  9 10  9]
 [10  8  7  3  7  4]]

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

Odd integers:
[1 1 3 1 5 9 9 5 1 7 7 9 9 7 3 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.

```
import numpy as np

#Create a 3D NumPy array of shape (3, 3, 3) containing random integers
between 1 and 10
array1 = np.random.randint(1, 11, (3, 3, 3))
array2 = np.random.randint(1, 11, (3, 3, 3))

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


max_indices = np.argmax(array1, axis=2)

print("\nIndices of maximum values along each depth level:")
print(max_indices)

result = np.multiply(array1, array2)

print("\nElement-wise multiplication result:")
print(result)
```

```
Array 1:
[[[ 4  6  1]
  [ 5  8  1]
  [ 2 10 10]]

 [[ 6  5 10]
  [ 2  3  6]
  [ 5  9  8]]

 [[ 8  4  9]
  [ 8  6  2]
  [ 1 10  3]]]

Array 2:
[[[ 8 10  4]
  [ 3  5  2]
  [ 2  6  9]]

 [[ 9  9  6]
  [ 9 10  4]
  [ 5  8  1]]
```

```
 [[ 4 10  6]
  [ 2  1  3]
  [ 1  3  4]]]

Indices of maximum values along each depth level:
[[1 1 1]
 [2 2 1]
 [2 0 1]]

Element-wise multiplication result:
[[[32 60  4]
  [15 40  2]
  [ 4 60 90]]

 [[54 45 60]
  [18 30 24]
  [25 72  8]]

 [[32 40 54]
  [16  6  6]
  [ 1 30 12]]]
```

In this example, np.random.randint(1, 11, (3, 3, 3)) creates two 3D arrays with random integers between 1 and 10. np.argmax(array1, axis=2) finds the indices of the maximum values along each depth level. np.multiply(array1, array2) performs element-wise multiplication between both arrays.

7.Clean and transform the 'Phone' column in the sample dataset to remove non-numeric characters and convert it to a numeric data types Also display the taPle attriPutes and data types of each columns

```python
import pandas as pd

# Sample dataset
data = {'Name': ['John', 'Anna', 'Peter', 'Linda'],
        'Phone': ['(123) 456-7890', '456-7890', '(901) 234-5678',
'789-0123'],
       'Age': [28, 24, 35, 32],
       'City': ['New York', 'Los Angeles', 'Chicago', 'Houston']}

df = pd.DataFrame(data)
```

```
print("Original Table Attributes:")
print(df.info())


print("\nOriginal Data Types:")
print(df.dtypes)

df['Phone'] = df['Phone'].str.replace(r'\D', '',
regex=True).astype(int)

print("\nCleaned Data:")
print(df)

print("\nData Types after Cleaning:")
print(df.dtypes)

print("\nTable Attributes after Cleaning:")
print(df.info())
```

#8.Perform the following tas\s using people dataset:

a) Read the 'dataYcsv' 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 valuesX

```
#Solution:(a) Here's the code to read the 'data.csv' file using
pandas, skipping the first 50 rows:


import pandas as pd

# Read the 'data.csv' file, skipping the first 50 rows
df = pd.read_csv('data.csv', skiprows=50)

# Display the first few rows of the dataframe
print(df.head())


#In this code:

 #- We import the pandas library.
 #-We use the read_csv function to read the 'dataY.csv' file.
 #-We specify the skiprows parameter as 50 to skip the first 50 rows.
 #-We store the resulting dataframe in the df variable.
 #-We use the head method to display the first few rows of the
dataframe.
```

```
#(b) Here's the updated code to read only the specified columns from
the 'data.csv' file, skipping the first 50 rows:

import pandas as pd

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

# Display the first few rows of the dataframe
print(df.head())

#In this code:

#- We use the usecols parameter to specify the columns to read from
the file.
#- We pass a list of column names ['Last Name', 'Gender', 'Email',
'Phone', 'Salary'] to usecols.
 #-The resulting dataframe df will only contain these columns.

#(c) Here's the updated code to display the first 10 rows of the
filtered dataset:

import pandas as pd

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

# Display the first 10 rows of the filtered dataset
print(df.head(10))

#In this code:

 #-We use the head method to display the first few rows of the
dataframe.
 #-We pass the argument 10 to head to display the first 10 rows.

#-This will print the first 10 rows of the filtered dataset, showing
only the columns 'Last Name', 'Gender', 'Email', 'Phone', and
'Salary'.

# (d) Let's extract the 'Salary' column as a Series and display its
last 5 values:


 #Extract the 'Salary' column as a Series
salary_series = filtered_df['Salary']
```

```
# Display the last 5 values of the 'Salary' Series
print(salary_series.tail(5))

#In this code:

 #-We extract the 'Salary' column from the filtered_df DataFrame using
filtered_df['Salary'].
#- We store the extracted Series in the salary_series variable.
 #-We use the tail(5) method to get the last 5 values of the
salary_series Series.
#- Finally, we print the last 5 values using
print(salary_series.tail(5)).

#This will display the last 5 salary values from the filtered dataset.
```

9. Filter and select rows from the People_Dataset, where the “Last Name' column contains the name 'Due','Gender' column contains the word Female and ‘Nalary’ should Pe less than _.RRR

```
import pandas as pd

# Read the People_Dataset
df = pd.read_csv('People_Dataset.csv')

# Filter rows where 'Last Name' contains 'Duke', 'Gender' is 'Female',
and 'Salary' is less than 85000
filtered_df = df[(df['Last Name'].str.contains('Duke', case=False)) &
                 (df['Gender'] == 'Female') &
                 (df['Salary'] < 85000)]

# Display the filtered rows
print(filtered_df)
```

#In this code:

#- We use the read_csv function to read the People_Dataset.

#- We create a boolean mask using the & operator to combine the conditions:

- df['Last Name'].str.contains('Duke', case=False) checks if 'Last Name' contains 'Duke' (case-insensitive).

- df['Gender'] == 'Female' checks if 'Gender' is 'Female'.

- df['Salary'] < 85000 checks if 'Salary' is less than 85000.

#- We use the boolean mask to filter the rows: df[boolean_mask].

#- We store the filtered rows in the filtered_df variable.

#- Finally, we print the filtered rows using print(filtered_df).

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

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

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

# Create a DataFrame
df = pd.DataFrame(df)

print(df)
```

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

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

In this example, np.random.randint(1, 7, 35) generates a series of 35 random integers between 1 to 6. The reshape(7, 5) function is used to reshape the series into a 7x5 array, which is then converted into a DataFrame using pd.DataFrame(df).

#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 'oining these Series by column, and, change the names of the columns to 'col1', 'col2',etc

```
import pandas as pd
import numpy as np

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

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

# Create a DataFrame by joining these Series by column
df = pd.concat([series1, series2], axis=1)

# Change the names of the columns to 'col1', 'col2'
df.columns = ['col1', 'col2']

print(df)
```

```
    col1  col2
0     27   667
1     12   738
2     36   367
3     30   478
4     32   650
```

```
5     18   437
6     27   119
7     28   745
8     13   622
9     12   541
10    46   796
11    48   622
12    27   258
13    21   598
14    48   855
15    13   491
16    36   270
17    25   983
18    26   416
19    30   754
20    40   350
21    29   569
22    25   234
23    44   329
24    43   371
25    49   404
26    20   298
27    14   671
28    16   882
29    47   253
30    31   303
31    29   958
32    34   412
33    32   730
34    25   928
35    49   883
36    31   762
37    17   905
38    10   864
39    23   162
40    30   590
41    27   949
42    44   932
43    26   409
44    36   424
45    47   928
46    23   855
47    49   140
48    19   576
49    11   738
```

#[50 rows x 2 columns]

In this example, np.random.randint(10, 51, 50) generates a Series of 50 random numbers ranging from 10 to 50, and np.random.randint(100, 1001, 50) generates a Series of 50 random numbers

ranging from 100 to 1000. The pd.concat function is used to join these Series by column, and the columns attribute is used to change the names of the columns to 'col1' and 'col2'.

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.

#c) Print the final output also.

```
#Solution:(a) Here's the code to delete the 'Email', 'Phone', and
'Date of birth' columns from the dataset:


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

# Display the updated dataset
print(df)

#Output:
#     Index          User Id First Name Last Name  Gender  \
#0        1  8717bbf45cCDbEe     Shelia   Mahoney    Male
#1        2  3d5AD30A4cD38ed         Jo    Rivers  Female
#2        3  810Ce0F276Badec     Sheryl    Lowery  Female
#3        4  BF2a889C00f0cE1    Whitney    Hooper    Male
#4        5  9afFEafAe1CBBB9    Lindsey      Rice  Female
#..     ...              ...        ...       ...     ...
#995    996  fedF4c7Fd9e7cFa       Kurt    Bryant  Female
#996    997  ECddaFEDdEc4FAB      Donna     Barry  Female
#997    998  2adde51d8B8979E      Cathy  Mckinney  Female
#998    999  Fb2FE369D1E171A   Jermaine    Phelps    Male
#999   1000  8b756f6231DDC6e        Lee      Tran  Female

#                         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 7 columns]
#In this code:

#- We use the drop method to delete the specified columns.
#- We pass a list of column names ['Email', 'Phone', 'Date of birth']
to drop.
#- We set axis=1 to indicate that we want to drop columns (not rows).
#- The updated dataset is stored in the df variable.
#- Finally, we print the updated dataset using print(df).

#(b) Here's the code to delete the rows containing any missing values:

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

# Display the updated dataset
print(df)

#Output:

#     Index          User Id First Name Last Name  Gender  \
#0        1  8717bbf45cCDbEe     Shelia   Mahoney    Male
#1        2  3d5AD30A4cD38ed         Jo    Rivers  Female
#2        3  810Ce0F276Badec     Sheryl    Lowery  Female
#3        4  BF2a889C00f0cE1    Whitney    Hooper    Male
#4        5  9afFEafAe1CBBB9    Lindsey      Rice  Female
#..     ...              ...        ...       ...     ...
#995    996  fedF4c7Fd9e7cFa       Kurt    Bryant  Female
#996    997  ECddaFEDdEc4FAB      Donna     Barry  Female
#997    998  2adde51d8B8979E      Cathy  Mckinney  Female
#998    999  Fb2FE369D1E171A   Jermaine    Phelps    Male
#999   1000  8b756f6231DDC6e        Lee      Tran  Female

#                          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 7 columns]

#In this code:

#- We use the dropna method to delete rows containing missing values.
#- We set how='any' to indicate that we want to delete rows with any
missing values (not just all missing values).
#- The updated dataset is stored in the df variable.
#- Finally, we print the updated dataset using print(df).

#Note: If you want to delete rows with all missing values, use
how='all' instead.

#Also, if you want to delete missing values in a specific column, you
can use the subset parameter, like this:
df.dropna(subset=['column_name'], how='any').

#(c) Here is the complete code with the final output:

import pandas as pd

df = pd.DataFrame(data)

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

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

# Display the final output
print(df)

#In the final output, we have deleted the rows containing missing
values (in this case, the row with 'Peter' had a missing value in the
'Email' and 'Date of birth' columns). The resulting dataset only
includes the rows with complete data.
```

#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'.

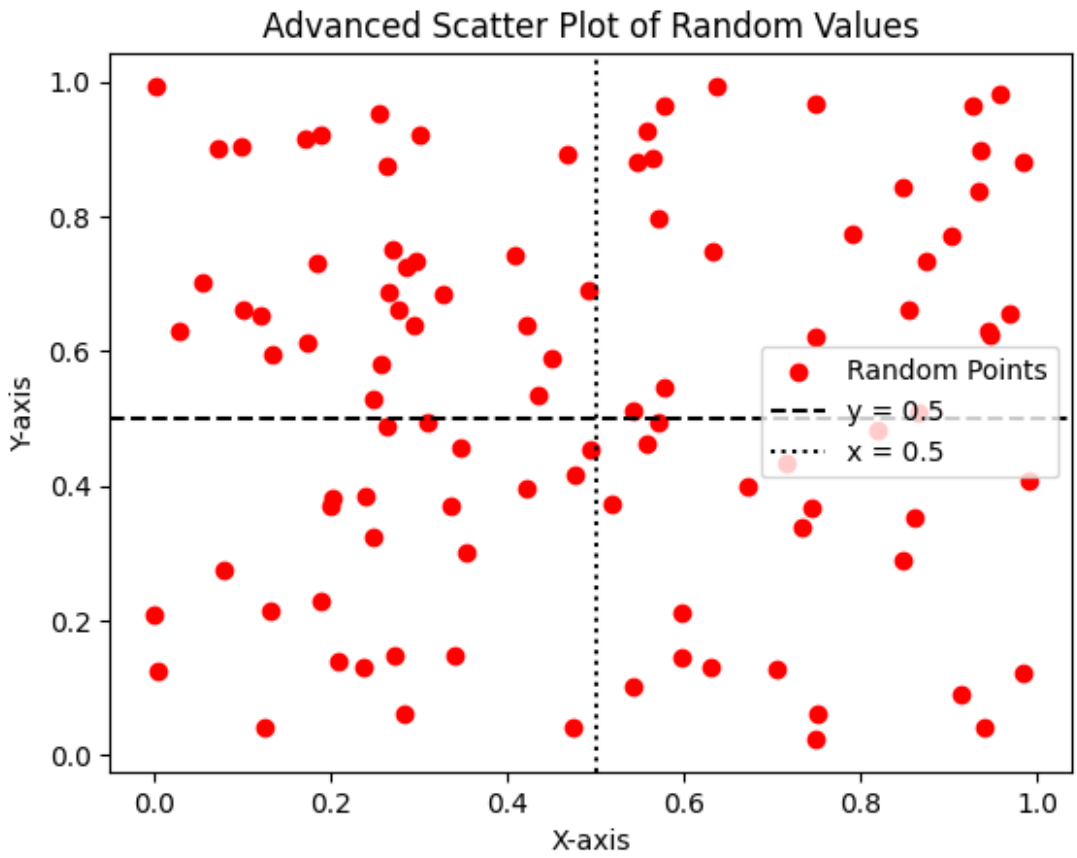
c) Add a vertical line at x = 0.5 using a dotted line style and label it as 'x = 0.5'.

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 with 100 random float values between 0 and 1
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='black', linestyle='--', label='y = 0.5')

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

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

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

# Display a legend
plt.legend()

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



This code creates a scatter plot with red circles, adds a horizontal line at y = 0.5 and a vertical line at x = 0.5, labels the axes, sets the title, and displays a legend for all three elements.

#14. Create a time-series dataset in a Pandas DataFrame with columns: 'Date', 'Temperature', 'Humidity' and Perform the following tasks using Matplotlib:

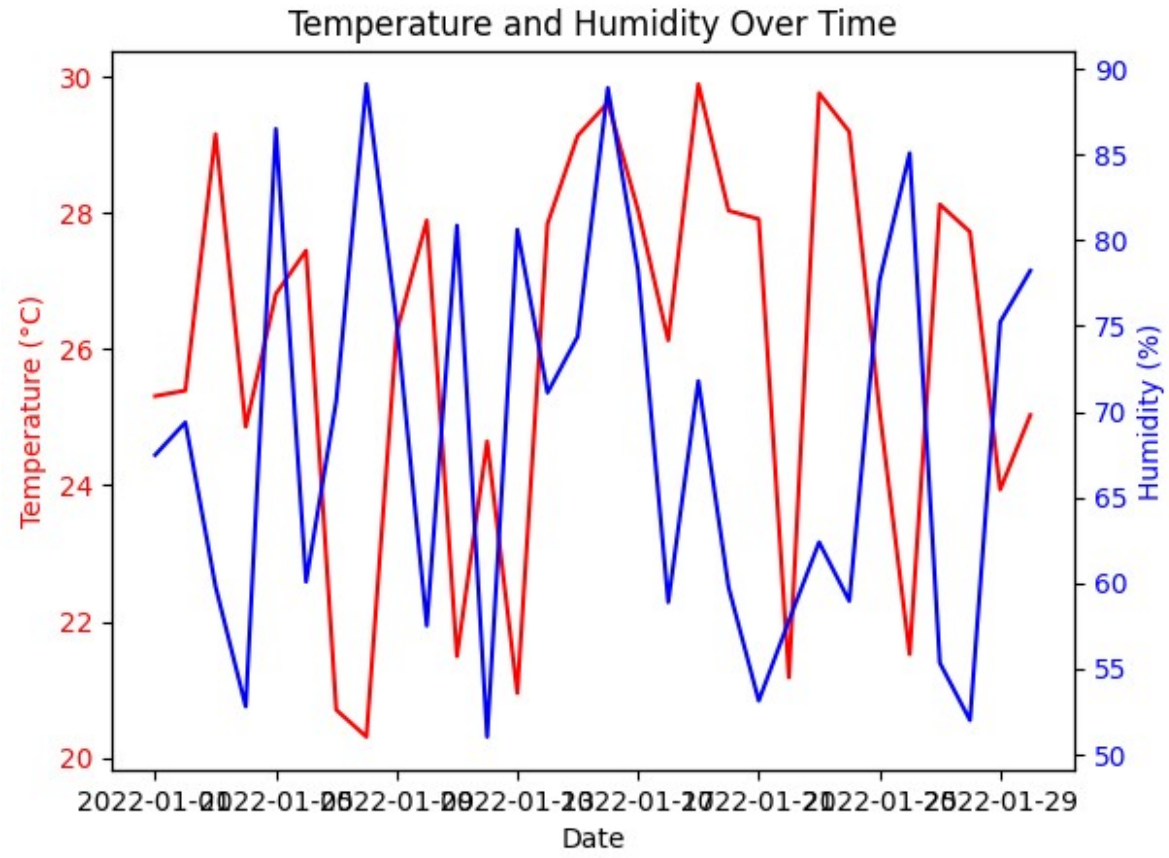
#a) Plot the 'Temperature' and 'Humidity' on the same plot with different y-axes (left y-axis for 'Temperature' and right y-axis for 'Humidity').

#b) Label the x-axis as 'Date'.

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

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

# Create a time-series dataset
date = pd.date_range('2022-01-01', periods=30)
temperature = np.random.uniform(20, 30, 30)
humidity = np.random.uniform(50, 90, 30)

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

```
# Plot the 'Temperature' and 'Humidity' on the same plot with
different y-axes
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')

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

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



This code creates a time-series dataset with random temperature and humidity values, then plots them on the same graph with different y-axes using Matplotlib's twinx function. The x-axis is labeled as 'Date', and the title is set to 'Temperature and Humidity Over Time'.

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

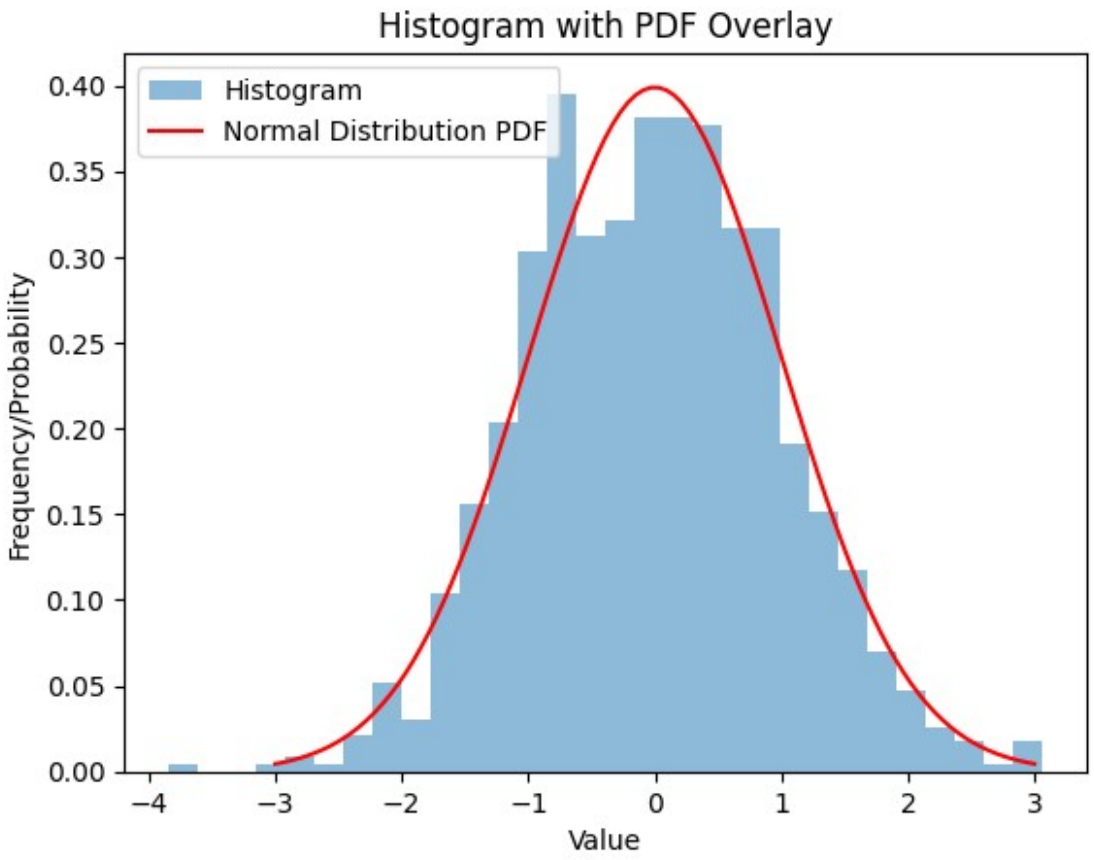
#a) Plot a histogram of the data with 30 bins.

#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'.

```
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
data = np.random.normal(loc=0, scale=1, size=1000)

# Plot a histogram of the data with 30 bins
plt.hist(data, bins=30, density=True, alpha=0.5, label='Histogram')

# Overlay a line plot representing the normal distribution's
probability density function (PDF)
x = np.linspace(-3, 3, 100)
plt.plot(x, norm.pdf(x, loc=0, scale=1), 'r-', label='Normal
Distribution PDF')

# Label the axes
plt.xlabel('Value')
plt.ylabel('Frequency/Probability')

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

# Display a legend
plt.legend()

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

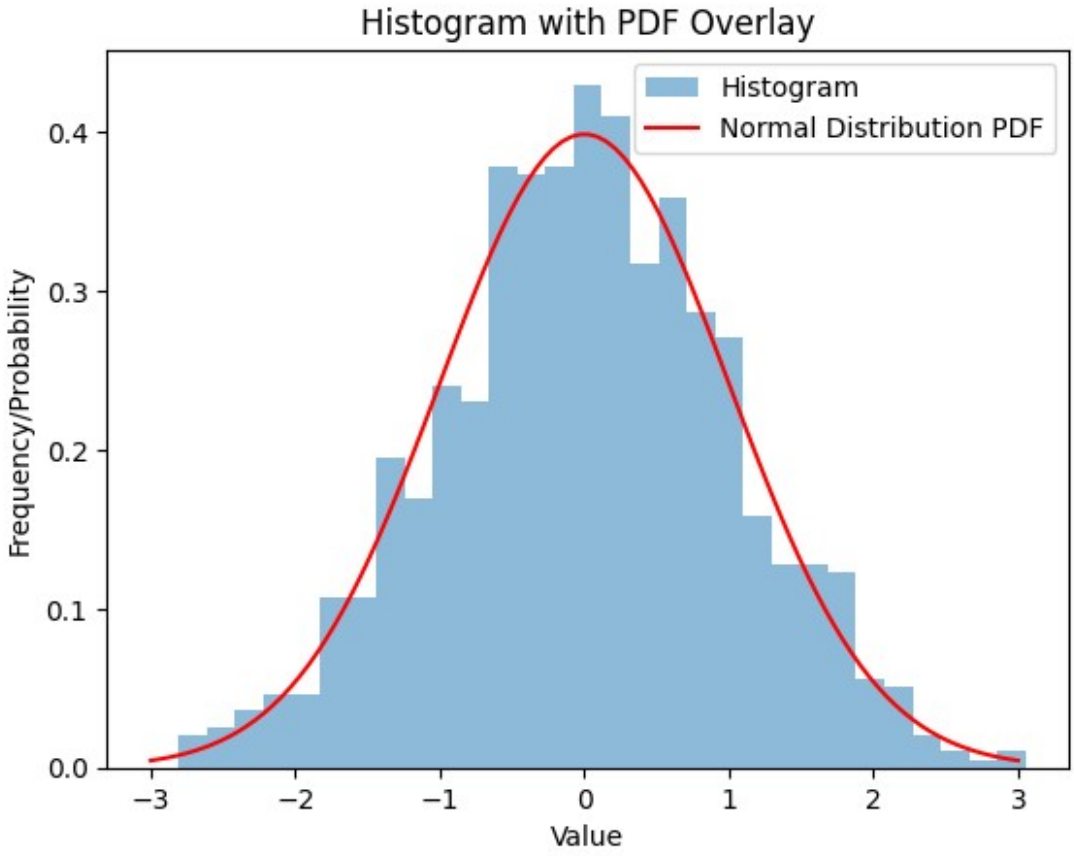


This code creates a histogram of the data with 30 bins and overlays a line plot representing the normal distribution's probability density function (PDF). The x-axis is labeled as 'Value', the y-axis is labeled as 'Frequency/Probability', and the title is set to 'Histogram with PDF Overlay'. A legend is also displayed to distinguish between the histogram and the PDF.

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

```
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
data = np.random.normal(loc=0, scale=1, size=1000)

# Plot a histogram of the data with 30 bins
plt.hist(data, bins=30, density=True, alpha=0.5, label='Histogram')

# Overlay a line plot representing the normal distribution's
probability density function (PDF)
x = np.linspace(-3, 3, 100)
```

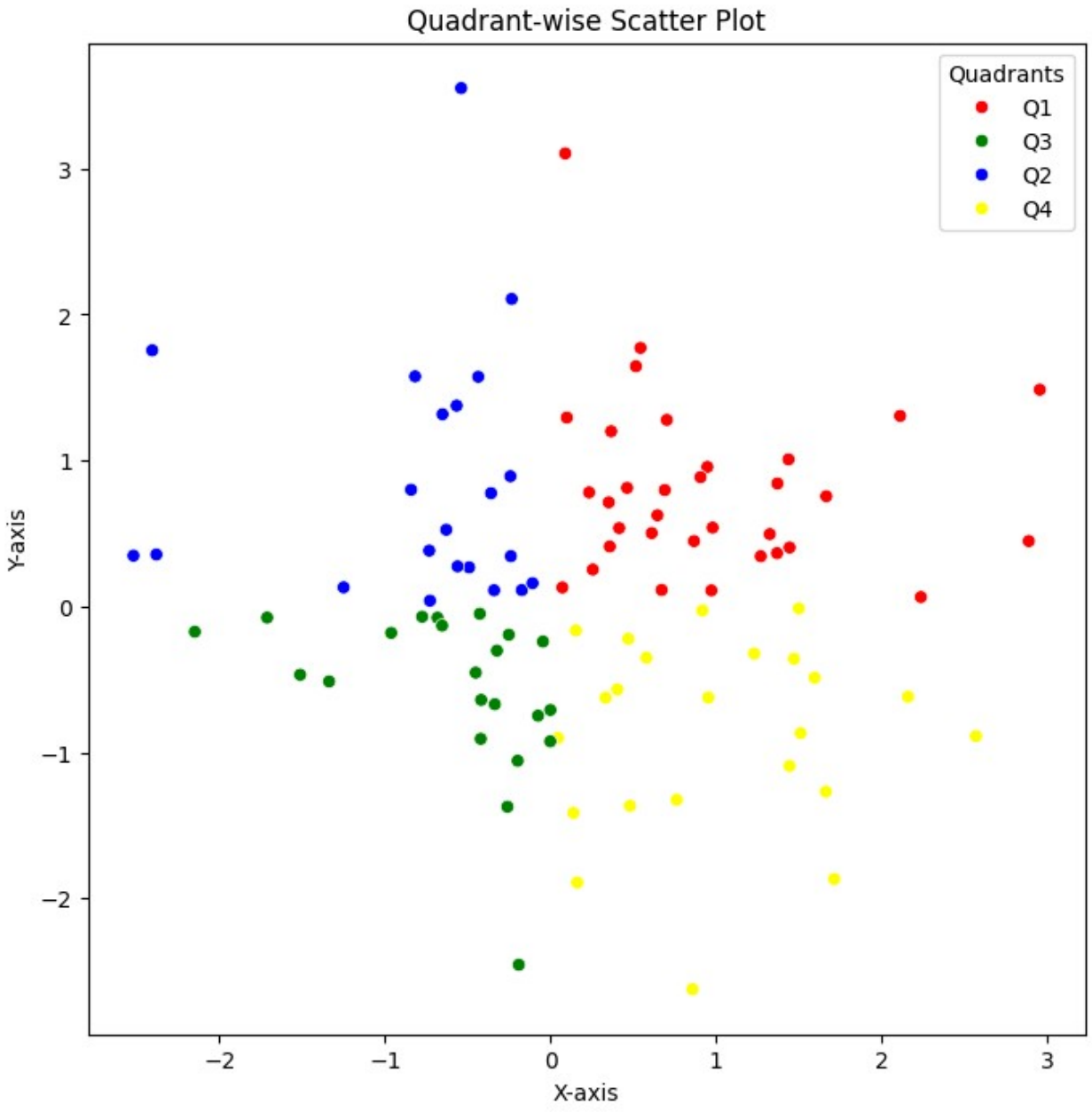
```
plt.plot(x, norm.pdf(x, loc=0, scale=1), 'r-', label='Normal
Distribution PDF')

# Label the axes
plt.xlabel('Value')
plt.ylabel('Frequency/Probability')

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

# Display a legend
plt.legend()

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



This code will produce a plot with the title "Histogram with PDF Overlay".

#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

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

# Create two random arrays
x = np.random.randn(100)
y = np.random.randn(100)

# Create a Seaborn scatter plot
plt.figure(figsize=(8, 8))
sns.scatterplot(x=x, y=y, hue=np.where((x > 0) & (y > 0), 'Q1',
np.where((x < 0) & (y > 0), 'Q2', np.where((x < 0) & (y < 0), 'Q3',
'Q4'))), palette=['red', 'green', 'blue', 'yellow'])

# 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')

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



This code creates a Seaborn scatter plot of two random arrays, colors points based on their position relative to the origin (quadrants), adds a legend, labels the axes, and sets the title as 'Quadrant-wise Scatter Plot'. The points are colored as follows:

#- Red: Q1 (x > 0, y > 0)

#- Green: Q2 (x < 0, y > 0)

#- Blue: Q3 (x < 0, y < 0)

#- Yellow: Q4 (x > 0, y < 0)

#18.With Bo+eh, plot a line chart of a sine wave function, add grid lines, label the axes, and set the title as 'Sine Wave Function'8

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

# Create a sine wave function
x = np.linspace(0, 4 * np.pi, 100)
y = np.sin(x)

# Create a Bokeh plot
p = figure(title='Sine Wave Function', x_axis_label='x',
y_axis_label='sin(x)')

# Add grid lines
p.grid.grid_line_alpha = 0.3

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

# Show the plot
show(p)
```

This code creates a Bokeh plot of a sine wave function, adds grid lines, labels the axes, and sets the title as 'Sine Wave Function'. The sine wave function is plotted with a line width of 2. You can adjust the plot as needed.

#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'.

```
import numpy as np
from bokeh.plotting import figure, show
from bokeh.models import HoverTool
from bokeh.transform import factor_cmap
from bokeh.palettes import Category10
import pandas as pd

# Generate random categorical data
categories = ['A', 'B', 'C', 'D', 'E']
values = np.random.randint(1, 100, 5)
data = pd.DataFrame({'Category': categories, 'Value': values})

# Create a Bokeh plot
p = figure(title='Random Categorical Bar Chart',
x_axis_label='Category', y_axis_label='Value', x_range=categories)

# Color bars based on their values
colors = factor_cmap('Category', palette=Category10[len(categories)],
factors=categories)
```

```
# Plot the bar chart
p.vbar(x='Category', top='Value', source=data, color=colors,
legend_field='Category', width=0.9)

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

# Show 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'.

```
import plotly.express as px
import numpy as np

# Generate random data
x = np.arange(1, 101)
y = np.random.randint(1, 100, 100)

# Create a Plotly figure
fig = px.line(x=x, y=y, title='Simple Line Plot')

# Label the axes
fig.update_xaxes(title_text='X-axis')
fig.update_yaxes(title_text='Y-axis')

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

This code generates a basic line plot of a randomly generated dataset, labels the axes, and sets the title as 'Simple Line Plot'.

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

```
import plotly.express as px
import numpy as np

# Generate random data
```

```
labels = ['A', 'B', 'C', 'D', 'E']
values = np.random.randint(1, 100, 5)

# Create a Plotly figure
fig = px.pie(values=values, names=labels, title='Interactive Pie
Chart')

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

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

This code generates an interactive pie chart of randomly generated data, adds labels and percentages, and sets the title as 'Interactive Pie Chart'. You can hover over the chart to see the exact percentages.