Data_toolkit5

September 9, 2024

[1]:
```
#Q1 Q1.) Demonstrate three different methods for creating identical 2D arrays␣
↪in NumPy Provide the code for each method and the final output after each␣
↪method

#Method 1: Using np.array() with a list of lists
import numpy as np

array1 = np.array([[1, 2], [3, 4]])
print(array1)
```

```
[[1 2]
 [3 4]]
```

[2]:
```
#Method 2: Using np.zeros() and manually setting values

array2 = np.zeros((2, 2))
array2[0, 0] = 1
array2[0, 1] = 2
array2[1, 0] = 3
array2[1, 1] = 4
print(array2)
```

```
[[1. 2.]
 [3. 4.]]
```

[3]:
```
#Method 3: Using np.full() with a custom pattern

array3 = np.full((2, 2), 0)
array3[0, 0] = 1
array3[0, 1] = 2
array3[1, 0] = 3
array3[1, 1] = 4
print(array3)
```

```
[[1 2]
 [3 4]]
```

[4]:
```
#Q2.) 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.
import numpy as np

# Generate an array of 100 evenly spaced numbers between 1 and 10
array_1d = np.linspace(1, 10, 100)

# Reshape the 1D array into a 2D array with a shape of (10, 10)
array_2d = array_1d.reshape(10, 10)

# Output the 2D array
print(array_2d)
```

```
[[ 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.        ]]
```

[5]:
```
#Q3.) Explain the following terms:

#The difference in np.array, np.asarray and np.asanyarray.

#The difference between Deep copy and shallow copy.

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

#1.) np.array:

#It always makes a copy of the input unless copy=False is explicitly passed.
```

```
#Can convert the input into a NumPy array, even if it's already an array (i.e.,
 ↪it forces copying by default).
#Example:
import numpy as np
list_data = [1, 2, 3]
arr1 = np.array(list_data)
```

[6]:
```
#2.) np.asarray:

#It doesn't copy the data if the input is already an array (NumPy array or
 ↪compatible array).

#It's more efficient when you don't need to duplicate memory.
#Example:
#np_arr = np.array([1, 2, 3])
#arr2 = np.asarray(np_arr)  # Does not make a copy if np_arr is already an array
#3.) np.asanyarray:

#Similar to np.asarray, but it will pass through any subclass of ndarray (like
 ↪matrix) without converting it to a base ndarray.

#Useful when working with specific array subclasses

#Example:
#arr3 = np.asanyarray(np_arr)  # Will not copy if np_arr is already an array
#Difference Between Deep Copy and Shallow Copy

#Shallow Copy:

#A shallow copy creates a new object but does not recursively copy the objects
 ↪it contains.

#Changes to nested mutable elements (like lists) affect the original object.

#Example:
#import numpy as np
#arr_original = np.array([1, 2, 3])
#shallow_copy = arr_original  # Shallow copy, only copies reference
#shallow_copy[0] = 100  # Changes affect the original array
#print(arr_original)

#Deep Copy:

#A deep copy creates a new object and recursively copies all objects contained
 ↪within it.

#Modifications to nested elements don't affect the original object.
```

```
#Example:
#import numpy as np
#import copy
#arr_original = np.array([1, 2, 3])
#deep_copy = copy.deepcopy(arr_original)  # Deep copy
#deep_copy[0] = 100  # Does not affect the original array
#print(arr_original)
```

[7]:
```
#Q4.) Generate a 3*3 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
random_array = np.random.uniform(5, 20, (3, 3))

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

print(rounded_array)
```

```
[[12.61 17.35 11.19]
 [10.17 10.34 13.44]
 [19.17  6.1   7.49]]
```

[8]:
```
#Q5.) 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

# Create a NumPy array with random integers between 1 and 10 of shape (5, 6)
random_int_array = np.random.randint(1, 11, (5, 6))

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

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

# Output the results
print("Original Array:\n", random_int_array)
print("\nEven Integers:\n", even_integers)
print("\nOdd Integers:\n", odd_integers)
```

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

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

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

[9]:
```python
#Q6.) 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

# Step 1: Create a 3D NumPy array of shape (3, 3, 3) containing random integers between 1 and 10
array_3d = np.random.randint(1, 11, (3, 3, 3))

# Step 2: Find the indices of the maximum values along each depth level (third axis)
max_indices = np.argmax(array_3d, axis=2)

# Step 3: Create another array for element-wise multiplication (same shape for demonstration)
array_3d_2 = np.random.randint(1, 11, (3, 3, 3))

# Perform element-wise multiplication between both arrays
result_array = array_3d * array_3d_2

# Output the results
print("Original 3D Array:\n", array_3d)
print("\nIndices of Maximum Values Along Each Depth Level:\n", max_indices)
print("\nSecond 3D Array:\n", array_3d_2)
print("\nResult of Element-wise Multiplication:\n", result_array)
```

```
Original 3D Array:
 [[[ 2  4  8]
  [ 6 10  6]
  [ 2  2  5]]
```

```
 [[ 1  2  5]
  [ 6 10  8]
  [ 5 10 10]]

 [[ 5  1  7]
  [10  5  2]
  [ 9 10  6]]]

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

Second 3D Array:
 [[[ 9  3  4]
  [ 7  8  4]
  [ 3  2  4]]

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

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

Result of Element-wise Multiplication:
 [[[ 18  12  32]
  [ 42  80  24]
  [  6   4  20]]

 [[  8   2  40]
  [  6 100  64]
  [ 15  20  60]]

 [[ 10   7   7]
  [ 80  30   6]
  [ 27  90  36]]]
```

[10]:
```python
#Q7.) 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

# Load the dataset
data = pd.read_csv('People Data.csv')
```

```
# Clean and transform the 'Phone' column to remove non-numeric characters
data['Phone'] = data['Phone'].str.replace(r'\D', '', regex=True)

# Convert the 'Phone' column to a numeric data type
data['Phone'] = pd.to_numeric(data['Phone'], errors='coerce')

# Display the table attributes and data types of each column
print("Data Types of Each Column:\n")
print(data.dtypes)

# Show the first few rows of the dataset to confirm the transformation
print("\nCleaned Dataset:\n")
print(data.head())
```

```
Data Types of Each Column:

Index              int64
User Id           object
First Name        object
Last Name         object
Gender            object
Email             object
Phone            float64
Date of birth     object
Job Title         object
Salary             int64
dtype: object

Cleaned Dataset:

   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

                          Email         Phone Date of birth  \
0           pwarner@example.org  8.571398e+09    27-01-2014
1  fergusonkatherine@example.net           NaN    26-07-1931
2           fhoward@example.org  5.997821e+09    25-11-2013
3         zjohnston@example.com           NaN    17-11-2012
4              elin@example.net  3.904172e+13    15-04-1923

                  Job Title  Salary
0         Probation officer   90000
```

```
1                    Dancer   80000
2                      Copy   50000
3  Counselling psychologist   65000
4       Biomedical engineer  100000
```

[11]:
```python
#Q8.) Perform the following tasks using people dataset:
import pandas as pd

# Task a: Read the 'data.csv' file using pandas, skipping the first 50 rows
data = pd.read_csv('People Data.csv', skiprows=range(1, 51))

# Task b: Only read the columns: 'Last Name', 'Gender', 'Email', 'Phone', and␣
 ↪'Salary'
filtered_data = data[['Last Name', 'Gender', 'Email', 'Phone', 'Salary']]

# Task c: Display the first 10 rows of the filtered dataset
print("First 10 rows of the filtered dataset:\n")
print(filtered_data.head(10))

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

```
First 10 rows of the filtered dataset:

  Last Name  Gender                        Email                  Phone  \
0    Zavala    Male         pamela64@example.net  001-859-448-9935x54536
1     Carey  Female    dianashepherd@example.net    001-274-739-8470x814
2     Hobbs  Female    ingramtiffany@example.org       241.179.9509x498
3    Reilly    Male  carriecrawford@example.org      207.797.8345x6177
4    Conrad    Male  fuentesclaudia@example.net    001-599-042-7428x143
5      Cole    Male      kaneaudrey@example.org            663-280-5834
6   Donovan    Male   rebekahsantos@example.net                     NaN
7    Little  Female         craig28@example.com      125.219.3673x0076
8    Dawson  Female  connercourtney@example.net     650-748-3069x64529
9      Page    Male  harrygallagher@example.com       849.500.6331x717

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

```
9   60000

Last 5 values of the 'Salary' column:

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

[12]:
```python
#Q9.) 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

# Load the dataset
data = pd.read_csv('People Data.csv')

# Filter the dataset where 'Last Name' contains 'Duke', 'Gender' is 'Female', and 'Salary' is less than 85000
filtered_data = data[
    (data['Last Name'].str.contains('Duke', case=False, na=False)) &
    (data['Gender'] == 'Female') &
    (data['Salary'] < 85000)
]

# Display the filtered rows
print("Filtered Data:\n")
print(filtered_data)
```

```
Filtered Data:

     Index          User Id First Name Last Name  Gender  \
45      46  99A502C175C4EBd     Olivia      Duke  Female
210    211  DF17975CC0a0373    Katrina      Duke  Female
457    458  dcE1B7DE83c1076      Traci      Duke  Female
729    730  c9b482D7aa3e682     Lonnie      Duke  Female

                        Email                    Phone Date of birth  \
45         diana26@example.net  001-366-475-8607x04350    13-10-1934
210        robin78@example.com            740.434.0212    21-09-1935
457  perryhoffman@example.org     +1-903-596-0995x489    11-02-1997
729   kevinkramer@example.net            982.692.6257    12-05-2015

           Job Title  Salary
45           Dentist   60000
210  Producer, radio   50000
```

```
457         Herbalist   50000
729     Nurse, adult    70000
```

[13]:
```
#Q10.) 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 and 6
random_series = pd.Series(np.random.randint(1, 7, 35))

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

# Create the DataFrame from the reshaped data
df = pd.DataFrame(df_random, columns=['Col1', 'Col2', 'Col3', 'Col4', 'Col5'])

# Display the DataFrame
print(df)
```

```
   Col1  Col2  Col3  Col4  Col5
0     2     3     3     5     1
1     6     1     1     4     4
2     6     6     3     1     4
3     1     1     3     5     3
4     4     5     5     2     3
5     3     2     6     1     5
6     2     3     6     3     3
```

[14]:
```
#Q11.) Create two different Series, each of length 50, with the following␣
 ↪criteria:
import pandas as pd
import numpy as np

# First Series: Random numbers ranging from 10 to 50
series1 = pd.Series(np.random.randint(10, 51, 50))

# Second Series: Random numbers ranging from 100 to 1000
series2 = pd.Series(np.random.randint(100, 1001, 50))

# Creating a DataFrame by joining these Series by columns
df_series = pd.DataFrame({'col1': series1, 'col2': series2})

# Display the DataFrame
print(df_series)
```

```
    col1  col2
0     48   597
```

1	19	965
2	40	888
3	14	269
4	14	313
5	40	576
6	34	486
7	21	716
8	20	873
9	50	709
10	19	246
11	44	681
12	22	596
13	19	964
14	18	233
15	38	334
16	43	628
17	47	264
18	49	784
19	30	140
20	16	210
21	36	495
22	35	660
23	35	228
24	19	283
25	36	162
26	32	611
27	49	407
28	39	487
29	50	955
30	47	410
31	42	511
32	41	512
33	14	685
34	35	943
35	41	699
36	24	126
37	41	838
38	29	351
39	21	154
40	46	778
41	24	977
42	46	344
43	17	652
44	30	344
45	29	708
46	42	395
47	25	425
48	10	753

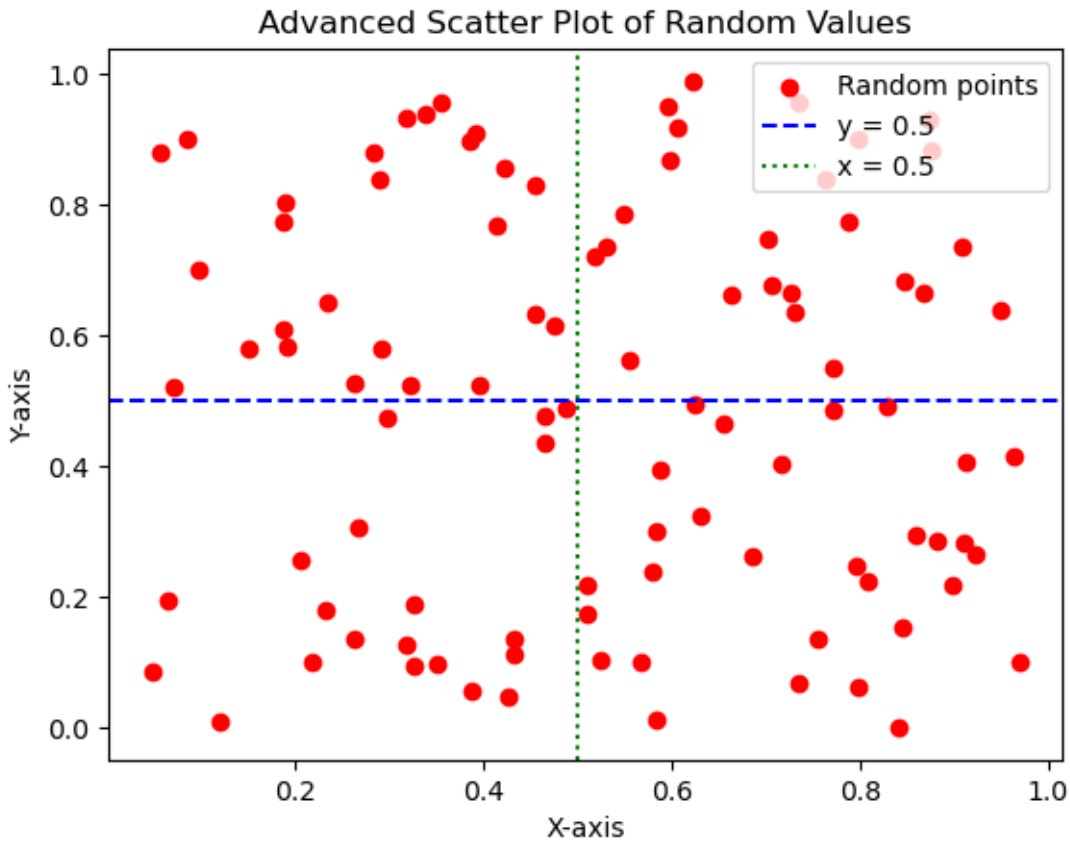
```
49    19   934
```

[15]:
```
#Q12.) 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

# Load the dataset
data = pd.read_csv('People Data.csv')

# Task a: Delete the 'Email', 'Phone', and 'Date of birth' columns from the␣
 ↪dataset
data_cleaned = data.drop(columns=['Email', 'Phone', 'Date of birth'],␣
 ↪errors='ignore')

# Task b: Delete the rows containing any missing values
data_cleaned = data_cleaned.dropna()

# Task d: Print the final output
print("Cleaned Dataset:\n")
print(data_cleaned)
```

```
Cleaned Dataset:

     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
..                              …     …
```

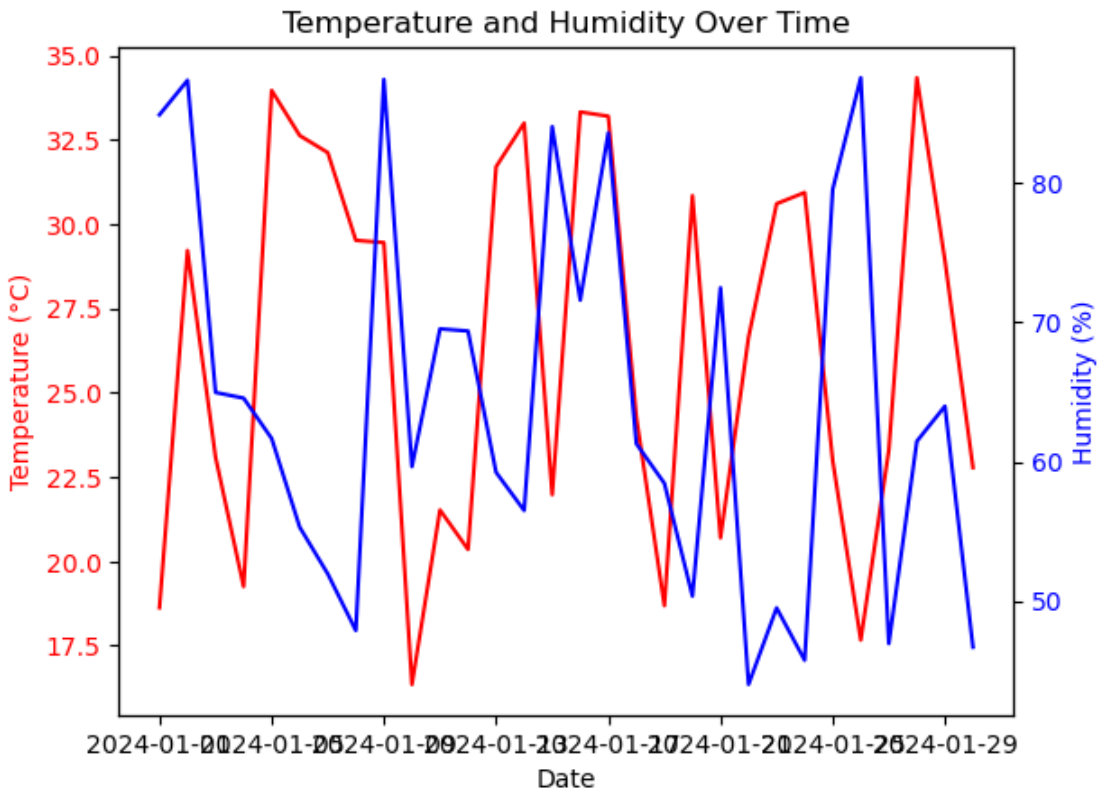
```
995                Personnel officer   90000
996          Education administrator   50000
997  Commercial/residential surveyor   60000
998                 Ambulance person  100000
999       Nurse, learning disability   90000

[1000 rows x 7 columns]
```

[16]:
```
#Q13.) 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:
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)

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

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

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

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

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

# Display the legend
plt.legend()

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

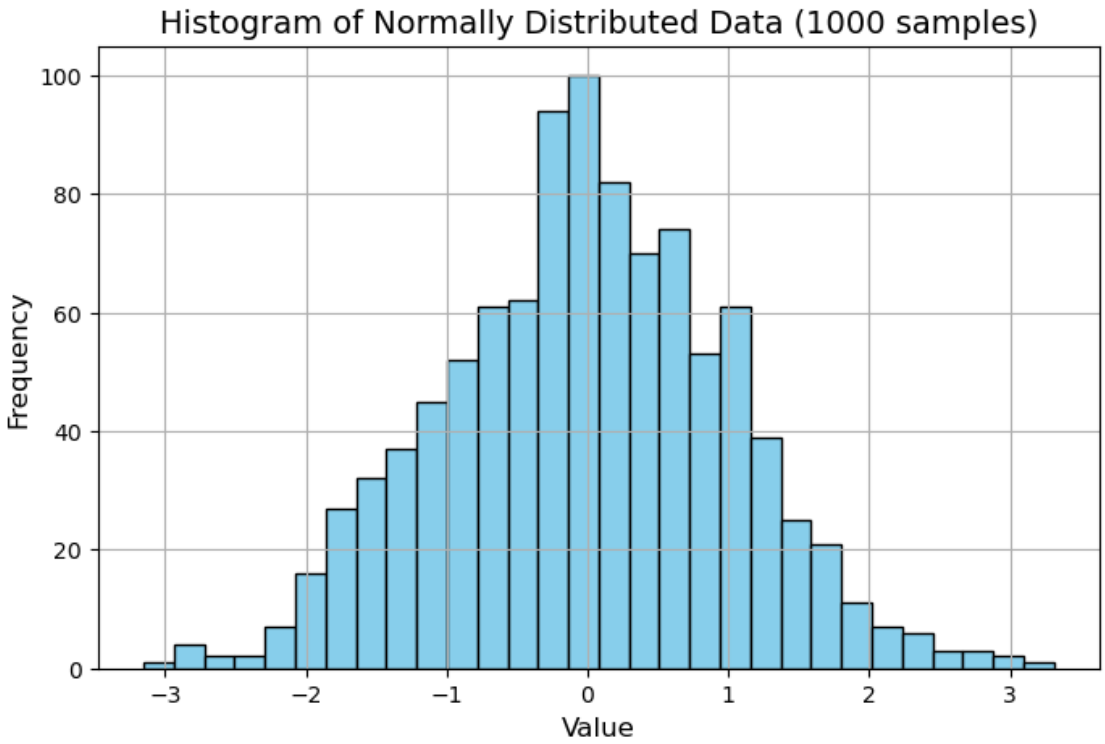


[17]:
```
#Q14.) Create a time-series dataset in a Pandas DataFrame with columns: 'Date',␣
 ↪'Temperature', 'Humidity' and Perform the following tasks using Matplotlib:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Step 1: Create a time-series dataset
date_range = pd.date_range(start='2024-01-01', periods=30, freq='D')  #␣
 ↪Generating 30 days of dates
temperature = np.random.uniform(15, 35, 30)  # Random temperatures between 15␣
 ↪and 35 degrees
humidity = np.random.uniform(40, 90, 30)     # Random humidity between 40% and␣
 ↪90%

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

```python
# Step 2: Plot the data with dual y-axes
fig, ax1 = plt.subplots()

# Plotting 'Temperature' on the left y-axis
ax1.plot(df['Date'], df['Temperature'], color='red', label='Temperature')
ax1.set_xlabel('Date')  # Label x-axis as 'Date'
ax1.set_ylabel('Temperature (°C)', color='red')
ax1.tick_params(axis='y', labelcolor='red')

# Create another y-axis that shares the same x-axis for 'Humidity'
ax2 = ax1.twinx()
ax2.plot(df['Date'], df['Humidity'], color='blue', label='Humidity')
ax2.set_ylabel('Humidity (%)', color='blue')
ax2.tick_params(axis='y', labelcolor='blue')

# Step 3: Set the title of the plot
plt.title('Temperature and Humidity Over Time')

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

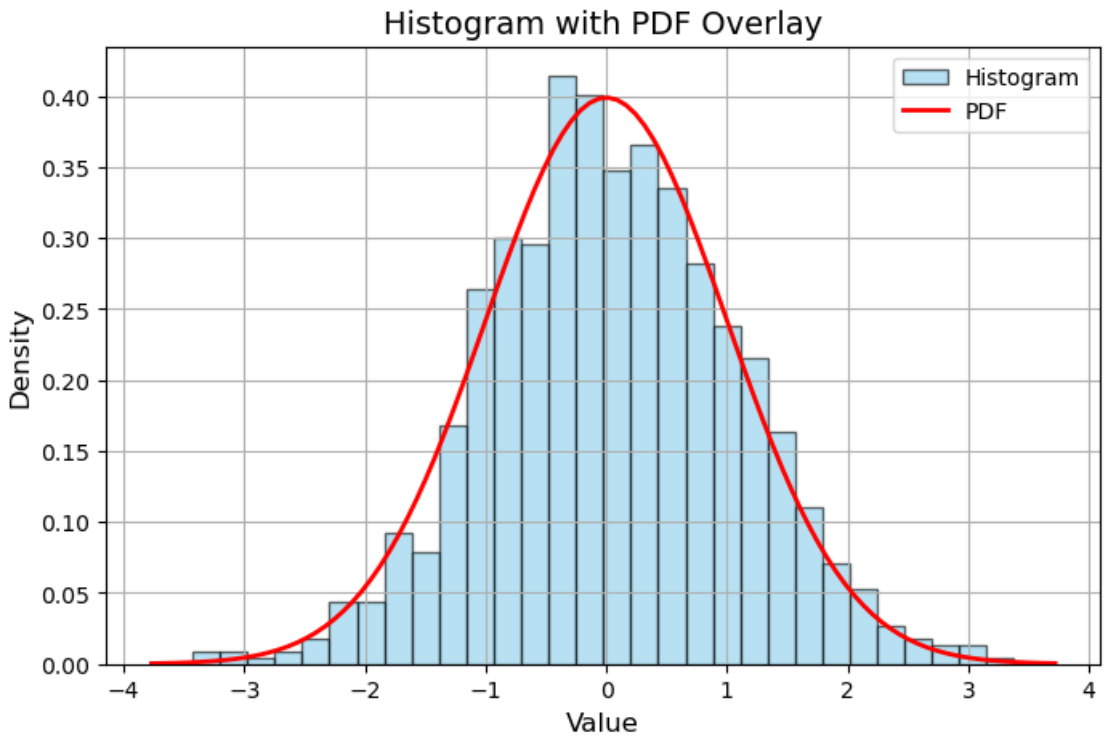


[20]:
```python
#Q15.) Create a NumPy arrayimport numpy as np
import numpy as np
import matplotlib.pyplot as plt

# a) Create a NumPy array containing 1000 samples from a normal distribution
# Mean = 0, Standard Deviation = 1 (standard normal distribution)
data = np.random.normal(loc=0, scale=1, size=1000)

# b) Plot the histogram of the data
plt.figure(figsize=(8, 5))
plt.hist(data, bins=30, color='skyblue', edgecolor='black')

# c) Customize the plot
plt.title('Histogram of Normally Distributed Data (1000 samples)', fontsize=14)
plt.xlabel('Value', fontsize=12)
plt.ylabel('Frequency', fontsize=12)

# d) Show the plot
plt.grid(True)
plt.show()
```



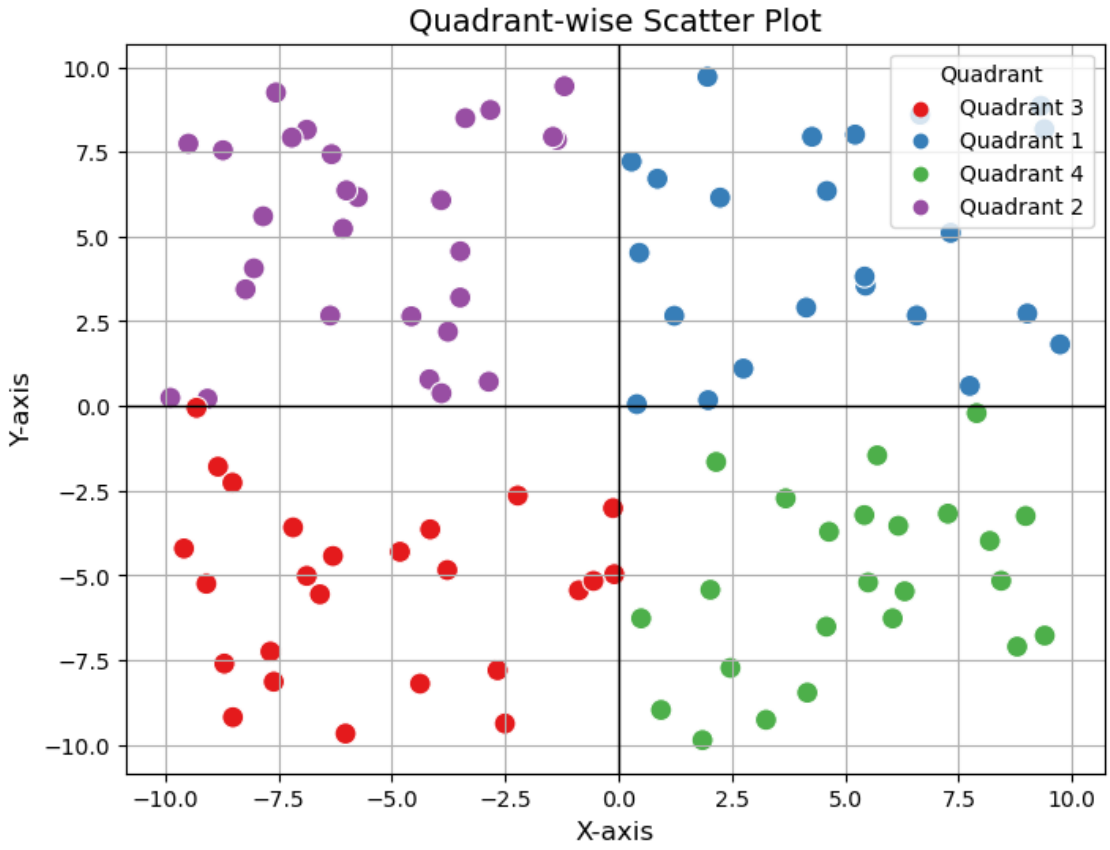
[21]:
```python
#Q16. Set the title of the plot as 'Histogram with PDF Overlay'.
import numpy as np
```

```
import matplotlib.pyplot as plt
import scipy.stats as stats

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

# b) Plot the histogram of the data with density=True to normalize it
plt.figure(figsize=(8, 5))
plt.hist(data, bins=30, color='skyblue', edgecolor='black', density=True,␣
 ↪alpha=0.6, label='Histogram')

# c) Generate the x values for the PDF (range based on the data)
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)

# d) Plot the PDF (probability density function) of the normal distribution
p = stats.norm.pdf(x, loc=0, scale=1)
plt.plot(x, p, 'r', linewidth=2, label='PDF')

# e) Customize the plot
plt.title('Histogram with PDF Overlay', fontsize=14)
plt.xlabel('Value', fontsize=12)
plt.ylabel('Density', fontsize=12)

# f) Add a legend
plt.legend()

# g) Show the plot
plt.grid(True)
plt.show()
```



[22]:
```python
#Q17 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

# a) Generate two random arrays (100 points) with values between -10 and 10
np.random.seed(42)  # For reproducibility
x = np.random.uniform(-10, 10, 100)
y = np.random.uniform(-10, 10, 100)

# b) Create a DataFrame with the x and y values
df = pd.DataFrame({'x': x, 'y': y})

# c) Define the quadrant based on x and y values
def assign_quadrant(row):
    if row['x'] >= 0 and row['y'] >= 0:
        return 'Quadrant 1'
    elif row['x'] < 0 and row['y'] >= 0:
        return 'Quadrant 2'
    elif row['x'] < 0 and row['y'] < 0:
```

```
        return 'Quadrant 3'
    else:
        return 'Quadrant 4'

df['Quadrant'] = df.apply(assign_quadrant, axis=1)

# d) Create a scatter plot using Seaborn, color points by 'Quadrant'
plt.figure(figsize=(8, 6))
sns.scatterplot(x='x', y='y', hue='Quadrant', data=df, palette='Set1', s=100)

# e) Customize the plot
plt.axhline(0, color='black',linewidth=1)  # Horizontal line at y=0
plt.axvline(0, color='black',linewidth=1)  # Vertical line at x=0
plt.title('Quadrant-wise Scatter Plot', fontsize=14)
plt.xlabel('X-axis', fontsize=12)
plt.ylabel('Y-axis', fontsize=12)

# f) Add a legend
plt.legend(title='Quadrant')

# g) Show the plot
plt.grid(True)
plt.show()
```



[23]:
```
#Q18 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'8
from bokeh.plotting import figure, show
from bokeh.io import output_notebook
from bokeh.models import ColumnDataSource, HoverTool
from bokeh.transform import factor_cmap
import random
import pandas as pd

# Initialize Bokeh output to display in the notebook
output_notebook()

# Generate random categorical data and values
categories = ['A', 'B', 'C', 'D', 'E']  # Categorical data
values = [random.randint(10, 100) for _ in range(5)]  # Random values

# Create a DataFrame to store data
data = pd.DataFrame({'categories': categories, 'values': values})

# Create a ColumnDataSource from the DataFrame
```

```
source = ColumnDataSource(data)

# Define a color map based on the values (using a gradient of colors)
colors = ['#718dbf', '#e84d60', '#c9d9d3', '#ddb7b1', '#80b1d3']
mapper = factor_cmap('categories', palette=colors, factors=categories)

# Create a figure with title and axis labels
p = figure(x_range=categories, title="Random Categorical Bar Chart",
           x_axis_label='Categories', y_axis_label='Values', height=400,␣
 ↪width=600)

# Add bar glyphs (using the color map to color based on values)
p.vbar(x='categories', top='values', width=0.8, source=source,
       fill_color=mapper, line_color='black')

# Add hover tooltips to display category and exact value
hover = HoverTool(tooltips=[('Category', '@categories'), ('Value', '@values')])
p.add_tools(hover)

# Show the plot
show(p)
```

[24]:
```
#Q19 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'8
from bokeh.plotting import figure, show
from bokeh.io import output_notebook
from bokeh.models import ColumnDataSource, HoverTool
from bokeh.transform import factor_cmap
import random
import pandas as pd

# Initialize Bokeh output to display in the notebook
output_notebook()

# Generate random categorical data and values
categories = ['A', 'B', 'C', 'D', 'E']  # Categorical data
values = [random.randint(10, 100) for _ in range(5)]  # Random values

# Create a DataFrame to store data
data = pd.DataFrame({'categories': categories, 'values': values})

# Create a ColumnDataSource from the DataFrame
source = ColumnDataSource(data)

# Define a color map based on the values (using a gradient of colors)
```

```
colors = ['#718dbf', '#e84d60', '#c9d9d3', '#ddb7b1', '#80b1d3']
mapper = factor_cmap('categories', palette=colors, factors=categories)

# Create a figure with title and axis labels
p = figure(x_range=categories, title="Random Categorical Bar Chart",
           x_axis_label='Categories', y_axis_label='Values', height=400,␣
 ↪width=600)

# Add bar glyphs (using the color map to color based on values)
p.vbar(x='categories', top='values', width=0.8, source=source,
       fill_color=mapper, line_color='black')

# Add hover tooltips to display category and exact value
hover = HoverTool(tooltips=[('Category', '@categories'), ('Value', '@values')])
p.add_tools(hover)

# Show the plot
show(p)
```

[28]:
```
#Q20.0 Using Plotly, create a basic line plot of a randomly generated dataset,␣
 ↪label the axes, and set the title as 'Simple Line Plot'8
import plotly.graph_objs as go
import numpy as np
import plotly.io as pio

# Generate random data
x = np.linspace(0, 10, 100)  # 100 points from 0 to 10
y = np.random.rand(100) * 10  # Random values between 0 and 10

# Create a line plot
line_plot = go.Figure()

# Add the trace (line)
line_plot.add_trace(go.Scatter(x=x, y=y, mode='lines', name='Random Data'))

# Set the layout (title and axes labels)
line_plot.update_layout(
    title='Simple Line Plot',
    xaxis_title='X-axis',
    yaxis_title='Y-axis'
)

# Display the plot
pio.show(line_plot)
```

[ ]:
```
#Q21 Using Plotly, create an interactive pie chart of randomly generated data,␣
 ↪add labels and percentages,
#set the title as 'Interactive Pie Chart'.
import numpy as np
import plotly.graph_objects as go

# Generate random data
np.random.seed(42)  # For reproducibility
labels = ['Category A', 'Category B', 'Category C', 'Category D', 'Category E']
values = np.random.randint(1, 100, size=len(labels))

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

# Update layout to add title
fig.update_layout(
    title='Interactive Pie Chart'
)

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

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: