Data toolkit

May 27, 2024

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

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

[2]:
```
# Method 1
array_zeros = np.zeros((3,2))  # method 1 - using np.zeros
print(array_zeros)

# method 2
array_one = np.ones((3,2))     # method 2 - using np.one
print(array_one)

# method 3
value = 5
array_full = np.full((3,2),value)   # method 3 - using np.full
print(array_full)
```

```
[[0. 0.]
 [0. 0.]
 [0. 0.]]
[[1. 1.]
 [1. 1.]
 [1. 1.]]
[[5 5]
 [5 5]
 [5 5]]
```

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

[3]:
```
evenly_spaced = np.linspace(1,10,100)
```

[4]:
```
evenly_spaced
```

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

```
[5]: reshaped_array = evenly_spaced.reshape(10,10)
```

```
[6]: reshaped_array
```

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

3. Explain the following terms:

- The difference in np.array, np.asarray and np.asanyarray.
- The difference between Deep copy and shallow copy.

ANS - Difference in np.array, np.asarray, and np.asanyarray:

- np.array

np.array always creates a new array, regardless of the input type.

It converts input data (lists, tuples, etc.) into an ndarray.

[7]:
```
list_data = [1, 2, 3]

array_from_list = np.array(list_data)
```

[8]:
```
array_from_list
```

[8]:
```
array([1, 2, 3])
```

- np.asarray() np.asarray converts the input into an array if it's not already an array. If the input is already an array, np.asarray doesn't create a new copy; it returns the original array.

[9]:
```
list_data = [1, 2, 3]
array_from_list = np.asarray(list_data)
```

[10]:
```
array_from_list
```

[10]:
```
array([1, 2, 3])
```

[11]:
```
list_data = (1, 2, 3)
array_from_list = np.asarray(list_data)
```

[12]:
```
array_from_list
```

[12]:
```
array([1, 2, 3])
```

- np.asanarray

np.asanyarray converts the input into an array if it's not already an array. Unlike np.asarray, it may not always create a new copy, even if the input is an array. It only creates a new copy if necessary to satisfy the requirements.

[13]:
```
list_data = [1, 2, 3]
array_from_list = np.asanyarray(list_data)
```

[14]:
```
array_from_list
```

[14]:
```
array([1, 2, 3])
```

[15]:

```python
list_data = (1, 2, 3)
array_from_list = np.asanyarray(list_data)
```

[16]:

```python
array_from_list
```

[16]:

```
array([1, 2, 3])
```

- The difference between Deep copy and shallow copy:
- Shallow copy

Shallow copy creates a new object, but it doesn't create copies of nested objects. Instead, it copies references to the nested objects. So, changes made to the nested objects in one copy will affect the other copy.

[17]:

```python
import copy
list1 = [1, [2, 3], 4]
shallow_copy_list1 = copy.copy(list1)  # shallow copy
```

[18]:

```python
shallow_copy_list1
```

[18]:

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

- Deep copy

Deep copy creates a new object and recursively copies all nested objects within it. It means it creates an entirely new copy of the original data structure, including all nested objects. Hence, changes made in one copy will not affect the other copy.

[19]:

```python
import copy
list1 = [1, [2, 3], 4]
deep_copy_list1 = copy.deepcopy(list1)   # Deep copy
```

[20]:

```python
deep_copy_list1
```

[20]:

```
[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.

[21]:

```python
# Generate random floating-point numbers between 5 and 20
random_array = np.random.uniform(5, 20, size=(3, 3))

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

print(rounded_array)
```

```
[[12.87  8.12 19.5 ]
 [16.46  7.51  8.84]
 [16.6   11.72 16.26]]
```

5. Create a Numpy array with random integers Petween 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.

[22]:
```
random_array = np.random.randint(1,10,size = (5,6))

# Extract all even integers from the array
even_int = random_array[random_array % 2 == 0]

# Extract all odd integers from the array

odd_int = random_array[random_array % 2 !=0]

print(even_int)
print(odd_int)
```

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

6 . Create a 3D NumPy array of shape (3, 3, 3) containing random integers Petween 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.

[23]:
```
random_3d_array = np.random.randint(1,10 , size = (3,3,3))

# a) Find the indices of the maximum values along each depth level (third axis)
max_indices = np.argmax(random_3d_array, axis=2)

# b) Perform element-wise multiplication between the original array and its␣
 ↪transpose
elementwise_multiplication = random_3d_array * random_3d_array.transpose(1, 2,␣
 ↪0)
```

[24]:
```
elementwise_multiplication
```

[24]:
```
array([[[25, 16, 49],
        [16, 15, 63],
        [42, 10, 42]],

       [[ 4, 45,  6],
        [27, 16,  4],
        [27,  5, 64]],
       [[42, 21, 54],
        [15, 20, 24],
        [63, 24,  9]]])
```

[25]:
```
max_indices
```

[25]:
```
array([[1, 2, 2],
       [1, 0, 2],
       [0, 2, 0]])
```

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.

[26]:
```
df = pd.read_csv("People Data.csv")
```

[27]:
```
df.head()
```

[27]:
```
   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         857.139.8239    27-01-2014
1  fergusonkatherine@example.net                  NaN    26-07-1931
2            fhoward@example.org        (599)782-0605    25-11-2013
3          zjohnston@example.com                  NaN    17-11-2012
4               elin@example.net  (390)417-1635x3010    15-04-1923

                  Job Title  Salary
0         Probation officer   90000
1                    Dancer   80000
2                      Copy   50000
3  Counselling psychologist   65000
4       Biomedical engineer  100000
```

[28]:
```
df.duplicated().sum()
```

[28]:
```
0
```

[29]:
```
df.dtypes
```

[29]:
```
Index             int64
User Id          object
First Name       object
```

```
Last Name        object
Gender           object
Email            object
Phone            object
Date of birth    object
Job Title        object
Salary            int64
dtype: object
```

[30]:
```
# Step 2: Remove non-numeric characters
df['Phone'] = df['Phone'].str.replace(r'\D', '', regex=True)
```

[31]:
```
df['Phone']
```

[31]:
```
0            8571398239
1                   NaN
2            5997820605
3                   NaN
4        39041716353010
             …
995          0217752933
996     0011497107799721
997     1750774412833265
998          9152922254
999      079752542467259
Name: Phone, Length: 1000, dtype: object
```

[32]:
```
# Step 3: Convert to numeric data type
df['Phone'] = pd.to_numeric(df['Phone'], errors='coerce')
```

[33]:
```
df['Phone']
```

[33]:
```
0      8.571398e+09
1               NaN
2      5.997821e+09
3               NaN
4      3.904172e+13
           …
995    2.177529e+08
996    1.149711e+13
997    1.750774e+15
998    9.152922e+09
999    7.975254e+13
Name: Phone, Length: 1000, dtype: float64
```

[34]:
```
df['Phone']
```

```
[34]: 0      8.571398e+09
      1               NaN
      2      5.997821e+09
      3               NaN
      4      3.904172e+13
                 …
      995    2.177529e+08
      996    1.149711e+13
      997    1.750774e+15
      998    9.152922e+09
      999    7.975254e+13
      Name: Phone, Length: 1000, dtype: float64
```

```
[35]: # Step 4: Display table attributes and data types
      print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Index          1000 non-null   int64
 1   User Id        1000 non-null   object
 2   First Name     1000 non-null   object
 3   Last Name      1000 non-null   object
 4   Gender         1000 non-null   object
 5   Email          1000 non-null   object
 6   Phone          979 non-null    float64
 7   Date of birth  1000 non-null   object
 8   Job Title      1000 non-null   object
 9   Salary         1000 non-null   int64
dtypes: float64(1), int64(2), object(7)
memory usage: 78.2+ KB
None
```

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.

```
[36]: # Read the 'data.csv' file using pandas, skipping the first 50 rows.
      pd.read_csv("People Data.csv",skiprows=50)
```

```
[36]:        50  afF3018e9cdd1dA     George    Mercer  Female   \
      0      51  CccE5DAb6E288e5         Jo    Zavala    Male
```

```
1      52  DfBDc3621D4bcec     Joshua     Carey  Female
2      53  f55b0A249f5E44D     Rickey     Hobbs  Female
3      54  Ed71DcfaBFd0beE      Robyn    Reilly    Male
4      55  FDaFD0c3f5387EC  Christina    Conrad    Male
..    …              …        …       …      …
945   996  fedF4c7Fd9e7cFa       Kurt    Bryant  Female
946   997  ECddaFEDdEc4FAB      Donna     Barry  Female
947   998  2adde51d8B8979E      Cathy  Mckinney  Female
948   999  Fb2FE369D1E171A   Jermaine    Phelps    Male
949  1000  8b756f6231DDC6e        Lee      Tran  Female

    douglascontreras@example.net    +1-326-669-0118x4341  11-09-1941  \
0           pamela64@example.net  001-859-448-9935x54536  23-11-1992
1      dianashepherd@example.net    001-274-739-8470x814  07-01-1915
2      ingramtiffany@example.org        241.179.9509x498  01-07-1910
3     carriecrawford@example.org       207.797.8345x6177  27-07-1982
4     fuentesclaudia@example.net    001-599-042-7428x143  06-01-1998
..                           …                     …          …
945       lyonsdaisy@example.net            021.775.2933  05-01-1959
946      dariusbryan@example.com    001-149-710-7799x721  06-10-2001
947       georgechan@example.org   +1-750-774-4128x33265  13-05-1918
948          wanda04@example.net           (915)292-2254  31-08-1971
949      deannablack@example.org      079.752.5424x67259  24-01-1947

             Human resources officer   70000
0                       Nurse, adult   80000
1                Seismic interpreter   70000
2                          Barrister   60000
3               Engineer, structural  100000
4                    Producer, radio   50000
..                               …       …
945                Personnel officer   90000
946          Education administrator   50000
947  Commercial/residential surveyor   60000
948                 Ambulance person  100000
949       Nurse, learning disability   90000

[950 rows x 10 columns]
```

[37]:
```
df = pd.read_csv('People Data.csv').head()
df
```

[37]:
```
   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        857.139.8239    27-01-2014
1  fergusonkatherine@example.net                 NaN    26-07-1931
2            fhoward@example.org       (599)782-0605    25-11-2013
3          zjohnston@example.com                 NaN    17-11-2012
4               elin@example.net  (390)417-1635x3010    15-04-1923

                  Job Title  Salary
0         Probation officer   90000
1                    Dancer   80000
2                      Copy   50000
3  Counselling psychologist   65000
4       Biomedical engineer  100000
```

```
[ ]:
```

```
[38]: # Only read the columns: 'Last Name', ‘Gender’,’Email’,‘Phone’ and ‘Salary’␣
       ↪from the file.
      df = pd.read_csv('People Data.csv', usecols=['Last Name', 'Gender', 'Email',␣
       ↪'Phone', 'Salary'])
      df
```

```
[38]:    Last Name  Gender                           Email                  Phone  \
      0     Mahoney    Male             pwarner@example.org           857.139.8239
      1      Rivers  Female   fergusonkatherine@example.net                    NaN
      2      Lowery  Female             fhoward@example.org          (599)782-0605
      3      Hooper    Male           zjohnston@example.com                    NaN
      4        Rice  Female                elin@example.net     (390)417-1635x3010
      ..        …     …                            …                    …
      995    Bryant  Female          lyonsdaisy@example.net           021.775.2933
      996     Barry  Female         dariusbryan@example.com    001-149-710-7799x721
      997  Mckinney  Female          georgechan@example.org  +1-750-774-4128x33265
      998    Phelps    Male             wanda04@example.net          (915)292-2254
      999      Tran  Female         deannablack@example.org     079.752.5424x67259

           Salary
      0     90000
      1     80000
      2     50000
      3     65000
      4    100000
      ..      …
      995   90000
      996   50000
      997   60000
```

```
998  100000
999   90000

[1000 rows x 5 columns]
```

[39]:

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

[39]:

```
  Last Name  Gender                         Email              Phone  Salary
0   Mahoney    Male           pwarner@example.org       857.139.8239   90000
1    Rivers  Female  fergusonkatherine@example.net               NaN   80000
2    Lowery  Female           fhoward@example.org      (599)782-0605   50000
3    Hooper    Male         zjohnston@example.com               NaN   65000
4      Rice  Female              elin@example.net  (390)417-1635x3010  100000
5  Caldwell    Male          kaitlin13@example.net         8537800927   50000
6   Hoffman    Male         jeffharvey@example.com  093.655.7480x7895   60000
7  Andersen    Male           alicia33@example.org         4709522945   65000
8      Mays    Male             jake50@example.com       013.820.4758   50000
9  Mitchell    Male      lanechristina@example.net  (560)903-5068x4985   50000
```

[40]:

```
# Extract the ‘Salary’' column as a Series and display its last 5 value

df['Salary'].tail()
```

[40]:

```
995     90000
996     50000
997     60000
998    100000
999     90000
Name: Salary, dtype: int64
```

9. Filter and select rows from the People_Dataset, where the “Last Name’ column contains the name ‘Duke’, ‘Gender’ column contains the word Female and ‘Salary’ should Pe less than 85000 .

[41]:

```
df = pd.read_csv('People Data.csv' , usecols = ['Last Name','Gender','Salary'])
df
```

[41]:

```
    Last Name  Gender  Salary
0     Mahoney    Male   90000
1      Rivers  Female   80000
2      Lowery  Female   50000
3      Hooper    Male   65000
4        Rice  Female  100000
..        …       …      …
995    Bryant  Female   90000
996     Barry  Female   50000
```

```
997  Mckinney  Female   60000
998    Phelps    Male  100000
999      Tran  Female   90000

[1000 rows x 3 columns]
```

[42]:

```
filterd_df = df[(df['Last Name'] == 'Duke') &(df['Gender'] == 'Female' ) &␣
  ↪(df['Salary'] <85000)]
filterd_df
```

[42]:

```
    Last Name  Gender  Salary
45       Duke  Female   60000
210      Duke  Female   50000
457      Duke  Female   50000
729      Duke  Female   70000
```

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

[43]:

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

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

# Convert the numpy array back to a pandas DataFrame

df = pd.DataFrame(df , columns = ['A','B','C','D','E'])

# Display The DataFrame
print(df)
```

```
   A  B  C  D  E
0  1  3  3  1  5
1  5  5  3  6  3
2  5  5  4  5  3
3  4  6  2  5  1
4  2  5  2  6  1
5  6  2  5  2  6
6  1  6  3  2  4
```

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 ‘jining these Series by column, and, change the names of the columns to ’col1’, ‘col2’.

[44]:

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

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


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

# Display the Data Frame
print(df)
```

```
    col1  col2
0     26   238
1     17   548
2     46   830
3     18   924
4     16   932
5     32   732
6     47   713
7     46   389
8     44   576
9     49   518
10    35   260
11    44   479
12    16   177
13    18   592
14    23   705
15    14   539
16    25   299
17    37   955
18    10   788
19    13   539
20    43   488
21    42   371
22    12   645
23    40   595
24    15   224
25    17   124
26    12   457
27    31   483
28    29   776
29    27   587
30    31   886
31    19   668
32    36   575
```

```
33     21    721
34     36    413
35     49    439
36     19    463
37     39    209
38     16    539
39     31    529
40     48    772
41     12    396
42     16    859
43     11    481
44     25    290
45     48    414
46     33    702
47     48    650
48     39    756
49     12    831
```

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.

[45]:
```python
# Read the data source
df = pd.read_csv('People Data.csv')

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

# Delete the rows containing any missing values.

df = df.dropna()

# Print the final output also.

df
```

[45]:
```
     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]
```

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'.

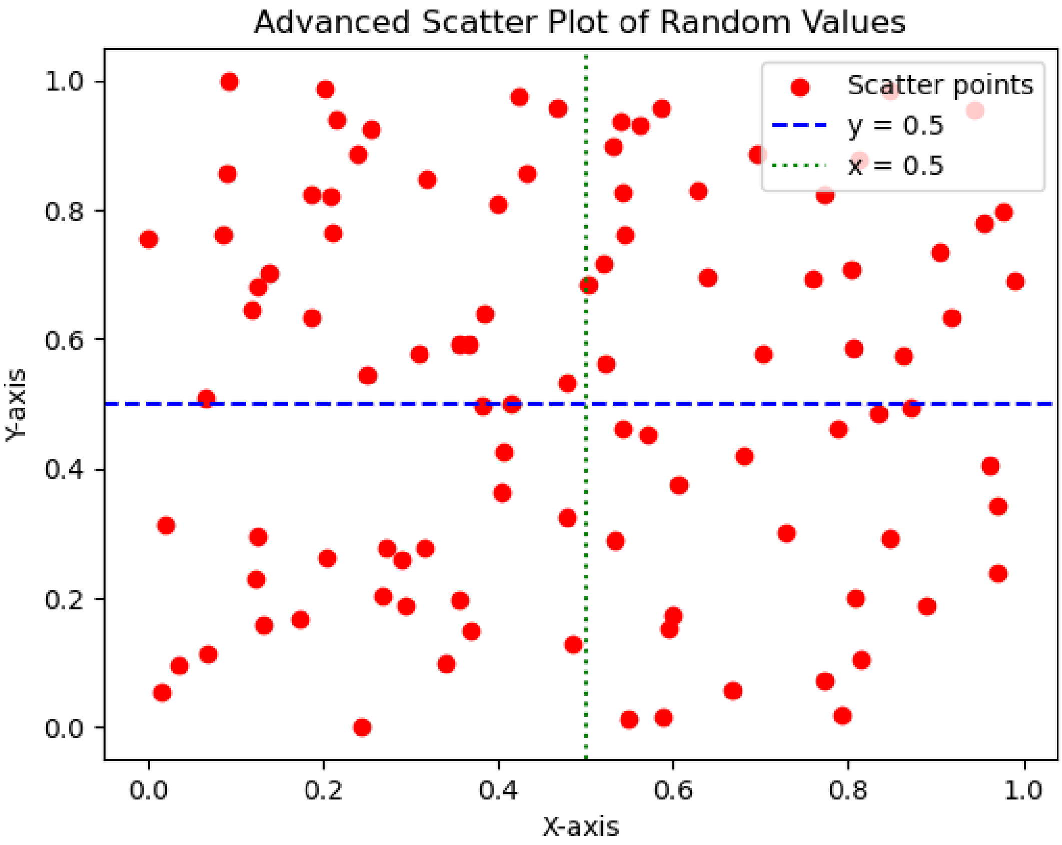
e) Set the title of the plot as 'Advanced Scatter Plot of Random Values'.

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

[46]:

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

# a) Create a scatter plot using x and y, setting the color of the points to␣
 ↪red and the marker style to 'o'
plt.scatter(x, y, color='red', marker='o', label='Scatter points')

# b) Add a horizontal line at y = 0.5 using a dashed line style and label it as␣
 ↪'y = 0.5'
plt.axhline(y=0.5, color='blue', linestyle='--', label='y = 0.5')

# c) Add a vertical line at x = 0.5 using a dotted line style and label it as␣
 ↪'x = 0.5'
```

```python
plt.axvline(x=0.5, color='green', linestyle=':', label='x = 0.5')

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

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

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

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



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

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

b) Label the x-axis as ‘Date’.

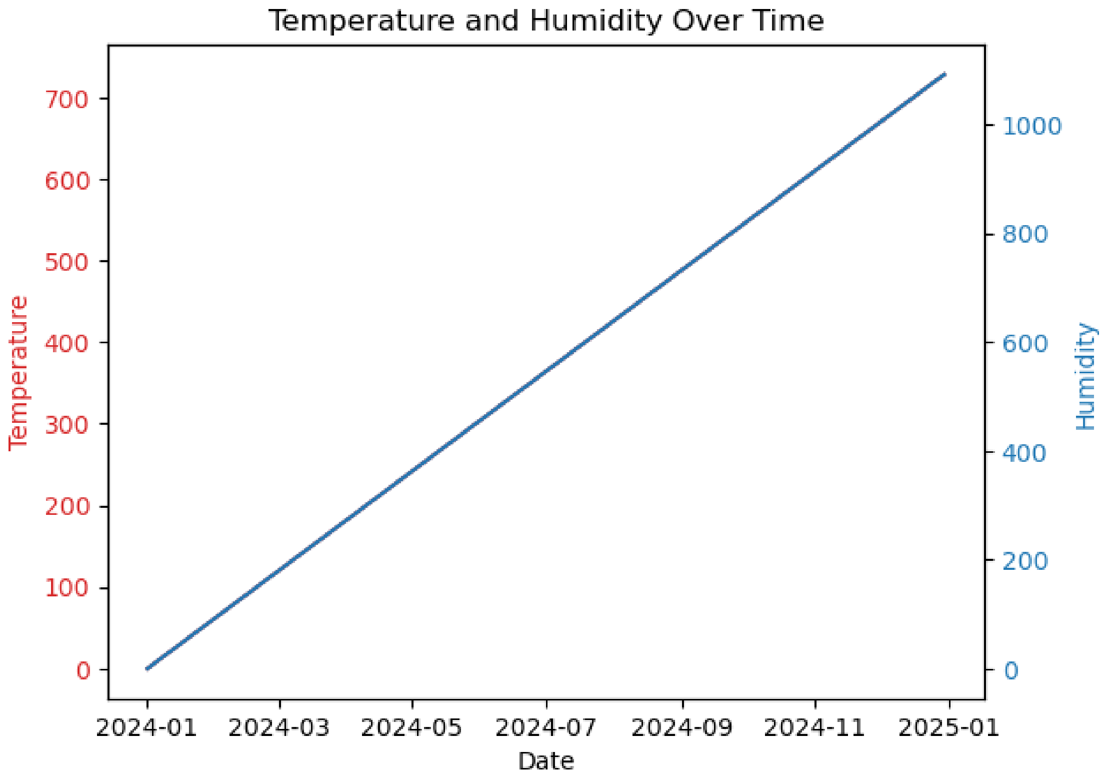
c) Set the title of the plot as ‘Temperature and Humidity Over Time’.

[47]:
```
# Create a time-series dataset in a Pandas DataFrame
data = {
    'Date': pd.date_range(start='2024-01-01', periods=365),
    'Temperature': pd.Series(range(365)) * 2,
    'Humidity': pd.Series(range(365)) * 3
}
df = pd.DataFrame(data)

 # Display Data Frame
df
```

[47]:
```
           Date  Temperature  Humidity
0    2024-01-01            0         0
1    2024-01-02            2         3
2    2024-01-03            4         6
3    2024-01-04            6         9
4    2024-01-05            8        12
..          …           …       …
360  2024-12-26          720      1080
361  2024-12-27          722      1083
362  2024-12-28          724      1086
363  2024-12-29          726      1089
364  2024-12-30          728      1092

[365 rows x 3 columns]
```

[48]:
```
# Plot the 'Temperature' and 'Humidity' on the same plot with different y-axes
fig, ax1 = plt.subplots()

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

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

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

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



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).

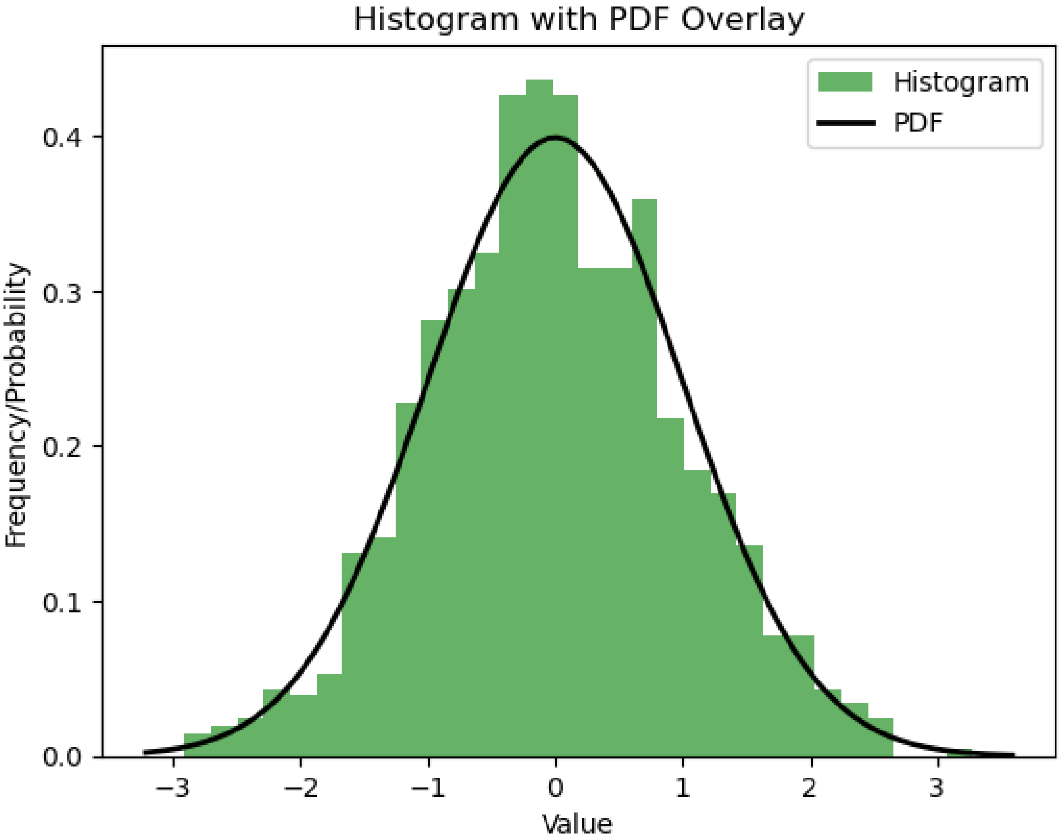
c) Label the x-axis as 'Value' and the y-axis as 'Frequency/Probability'.

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

[49]:

```python
# Generate a Numpy array containing 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=0.6, color='g', label='Histogram')

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

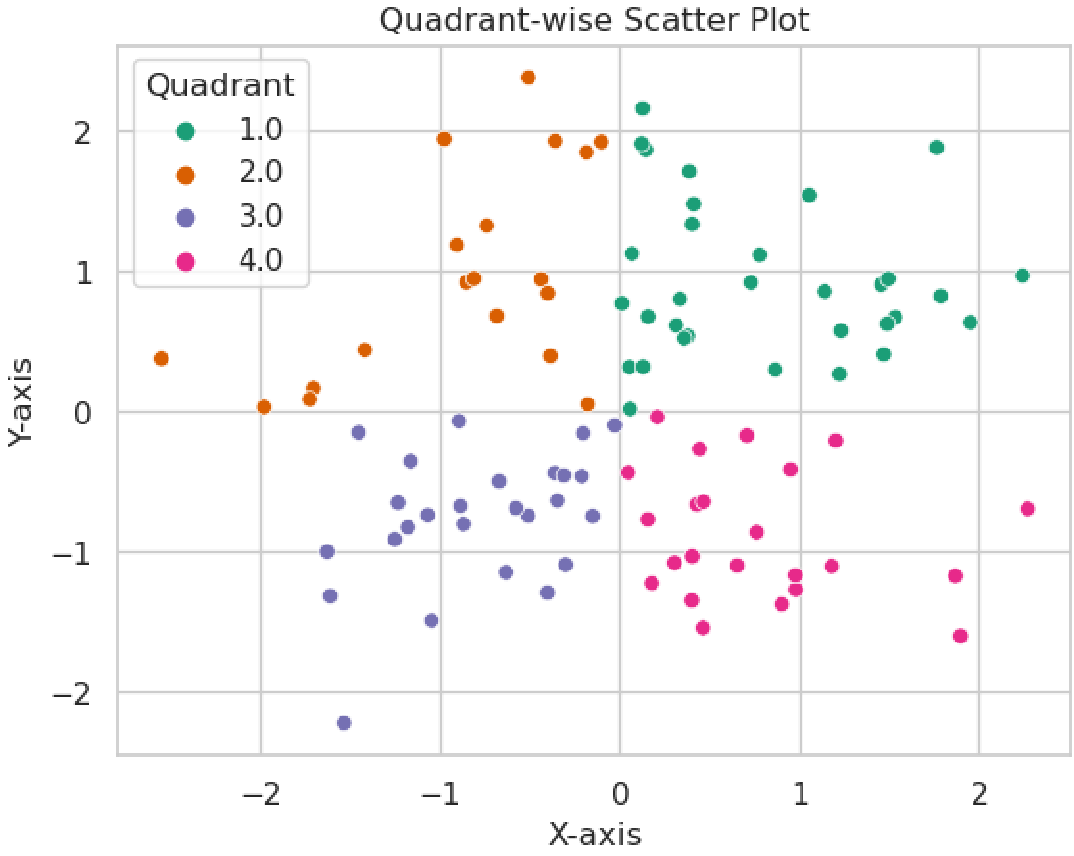
```
p = np.exp(-0.5 * x**2) / np.sqrt(2 * np.pi)
plt.plot(x, p, 'k', linewidth=2, label='PDF')

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

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

# Display legend
plt.legend()

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



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

[56]:

```python
# Generate random data
np.random.seed(0)
x = np.random.randn(100)
y = np.random.randn(100)

# Determine the quadrant for each point
quadrants = np.zeros_like(x)
quadrants[(x >= 0) & (y >= 0)] = 1  # Quadrant I
quadrants[(x < 0) & (y >= 0)] = 2   # Quadrant II
quadrants[(x < 0) & (y < 0)] = 3    # Quadrant III
quadrants[(x >= 0) & (y < 0)] = 4   # Quadrant IV

# Create a DataFrame for plotting
data = {'x': x, 'y': y, 'quadrant': quadrants}
df = pd.DataFrame(data)

# Set style
sns.set(style="whitegrid")

# Create the scatter plot
sns.scatterplot(x='x', y='y', hue='quadrant', palette='Dark2', data=df)

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

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

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

# Show plot
plt.show()
```



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

[59]:

```
from bokeh.plotting import figure, show
from bokeh.io import output_notebook

# Define the sine wave function
def sine_wave(x):
    return np.sin(x)

# Generate x values
x = np.linspace(0, 4*np.pi, 100)

# Generate y values using the sine wave function
y = sine_wave(x)

# Initialize the Bokeh figure
p = figure(title='Sine Wave Function', x_axis_label='x', y_axis_label='y')

# Add the line plot
```

```
p.line(x, y, line_width=2)

# Add grid lines
p.grid.grid_line_alpha = 0.5

# Show the plot
output_notebook()
show(p)
```

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

[60]:

```
from bokeh.models import HoverTool
import random

# Generate random categorical data
categories = ['A', 'B', 'C', 'D', 'E']
values = [random.randint(1, 10) for _ in range(len(categories))]

# Create a DataFrame
data = {'Categories': categories, 'Values': values}
df = pd.DataFrame(data)

# Initialize the Bokeh figure
p = figure(x_range=categories, title='Random Categorical Bar Chart',␣
  ↪x_axis_label='Categories', y_axis_label='Values')

# Create the bar chart
p.vbar(x='Categories', top='Values', width=0.5, color='blue', source=df)

# Add hover tooltips to display exact values
hover = HoverTool()
hover.tooltips = [('Value', '@Values')]
p.add_tools(hover)

# Label the axes
p.xaxis.major_label_orientation = 1
p.yaxis.axis_label_text_font_style = 'normal'

# Show the plot
show(p)
```

[ ]: