Data

May 27, 2024

[1]:
```
# 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.



import numpy as np

arr = np.array([[1,2,3,4,5]])
print(arr)
```

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

[2]:
```
ar_dia = np.eye(3,4)
print(ar_dia)
```

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

[3]:
```
ar_dia.ndim
```

[3]:
```
2
```

[4]:
```
np.zeros(5)
```

[4]:
```
array([0., 0., 0., 0., 0.])
```

[5]:
```
# Q2. Using the Numpy function, generate an array of 100 evenly spaced numPers␣
 ↪Between 1 and 10 and
# Reshape that 1D array into a 2D array.
```

[6]:
```
arr_1d = np.linspace(1,10,100)
arr_1d
```

[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.        ])
```

[7]:

```
arr_1d
```

[7]:

```
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.        ])
```

[8]:

```
arr_2d = arr_1d.reshape(10,10)
arr_2d
```

[8]:

```
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.        ]])
```

```
[10]: arr_2d
```

```
[10]: 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.        ]])
```

```
[11]: # Q3. Explain the following terms.
      # 1.The difference in npYarray, np.asarray and np.asanyarray.
      # 2.The difference between Deep copy and shallow copy.
```

```
[12]: # np.array() method creates copy of an existing objects.
      # >>> np.asarray()  create a new object only when needed.
```

[13]:

```
# deep copy >>> In Shallow copy, a copy of the original object is stored and␣
 ↪only the reference address is finally copied.
# In Deep copy, the copy of the original object and the repetitive copies both␣
 ↪are stored.
```

[14]:

```
# Q4. Generate a 3x3 array with random floating-point numPers between 5 and 20.␣
 ↪then, round each number in
#the array to 2 decimal places.
```

[15]:

```
array = np.random.uniform(5, 20, (3, 3))
```

[17]:

```
rounded_array = np.round(array, 2)
```

[18]:

```
rounded_array
```

[18]:

```
array([[ 7.29, 11.65,  8.77],
       [ 7.33,  7.12,  5.25],
       [18.74, 10.62, 16.27]])
```

[19]:

```
# Q5. Create a NumPy array with random integers Petween w and wR 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.
```

[21]:

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

even_numbers = array[array% 2 ==0]

odd_numbers = array[array % 2 !=0]

print("Original Array")
print(array)
print("\nEven Numbers:")
print(even_numbers)
print("\nOdd Numbers:")
print(odd_numbers)
```

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

```
Even Numbers:
[ 4  8  6 10  8  6  2  2 10  8  2  2]

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

[22]:

```
# 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.
```

[25]:

```
array1 = np.random.randint(1, 11, size=(3, 3, 3))
array2 = np.random.randint(1, 11, size=(3, 3, 3))

max_indices = np.argmax(array1, axis=2)
element_wise_multiplication = np.multiply(array1, array2)

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

print("\nIndices of Maximam Values along each depth Level:")
print(max_indices)

print("\nElement-wise Multiplication between Array 1 and Array 2:")
print(element_wise_multiplication)
```

```
Original Arrays:
Array 1:
[[[ 3  5  1]
  [ 5  6  4]
  [ 9  4  5]]

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

 [[ 4  7  6]
  [10  4  1]
  [ 5  9  3]]]

Array 2:
[[[ 4  1 10]
```

```
  [ 9 10  4]
  [ 8  2  6]]

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

 [[10  4  6]
  [ 2  8 10]
  [ 5  3 10]]]

Indices of Maximam Values along each depth Level:
[[1 1 0]
 [1 2 0]
 [1 0 1]]

Element-wise Multiplication between Array 1 and Array 2:
[[[12  5 10]
  [45 60 16]
  [72  8 30]]

 [[32 54 14]
  [42 36 18]
  [16  6 27]]

 [[40 28 36]
  [20 32 10]
  [25 27 30]]]
```

[26]:

```
# 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 taPle attriPutes and data␣
 ↪types of each column.
```

[1]:

```
import pandas as pd



data = {'Name': ['John', 'Alice', 'Bob'],
        'Age': [30, 25, 35],
        'Phone': ['(123) 456-7890', '+1-987-654-3210', '555-7890']}
df = pd.DataFrame(data)

print("Table attributes and data types before transformation:")
print(df.dtypes)

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

```
print("\nTable attributes and data types after transformation:")
print(df.dtypes)
print("\nDataFrame after cleaning and transforming the 'Phone' column:")
print(df)
```

```
Table attributes and data types before transformation:
Name     object
Age       int64
Phone    object
dtype: object

Table attributes and data types after transformation:
Name     object
Age       int64
Phone     int64
dtype: object

DataFrame after cleaning and transforming the 'Phone' column:
    Name  Age        Phone
0   John   30   1234567890
1  Alice   25  19876543210
2    Bob   35      5557890
```

[ ]:

```
#Question 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 values.
```

[3]:

```
import pandas as pd

df = pd.read_csv('data.csv', skiprows=50)

columns_to_read = ['Last Name', 'Gender', 'Email', 'Phone', 'Salary']
df_filtered = df[columns_to_read]


print("First 10 rows of the filtered dataset:")
print(df_filtered.head(10))

salary_series = df_filtered['Salary']
```

```
print("\nLast 5 values of the 'Salary' column:")
print(salary_series.tail(5))
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Cell In[3], line 3
      1 import pandas as pd
----> 3 df = pd.read_csv('data.csv', skiprows=50)
      5 columns_to_read = ['Last Name', 'Gender', 'Email', 'Phone', 'Salary']
      6 df_filtered = df[columns_to_read]

File /opt/conda/lib/python3.10/site-packages/pandas/util/_decorators.py:211, in
  ↪deprecate_kwarg.<locals>._deprecate_kwarg.<locals>.wrapper(*args, **kwargs)
    209     else:
    210         kwargs[new_arg_name] = new_arg_value
--> 211 return func(*args, **kwargs)

File /opt/conda/lib/python3.10/site-packages/pandas/util/_decorators.py:331, in
  ↪deprecate_nonkeyword_arguments.<locals>.decorate.<locals>.wrapper(*args,␣
  ↪**kwargs)
    325 if len(args) > num_allow_args:
    326     warnings.warn(
    327         msg.format(arguments=_format_argument_list(allow_args)),
    328         FutureWarning,
    329         stacklevel=find_stack_level(),
    330     )
--> 331 return func(*args, **kwargs)

File /opt/conda/lib/python3.10/site-packages/pandas/io/parsers/readers.py:950,␣
  ↪in read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col,␣
  ↪usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters,␣
  ↪true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows,␣
  ↪na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates ␣
  ↪infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates,␣
  ↪iterator, chunksize, compression, thousands, decimal, lineterminator,␣
  ↪quotechar, quoting, doublequote, escapechar, comment, encoding,␣
  ↪encoding_errors, dialect, error_bad_lines, warn_bad_lines, on_bad_lines,␣
  ↪delim_whitespace, low_memory, memory_map, float_precision, storage_options)
    935 kwds_defaults = _refine_defaults_read(
    936     dialect,
    937     delimiter,
   (…)
    946     defaults={"delimiter": ","},
    947 )
    948 kwds.update(kwds_defaults)
--> 950 return _read(filepath_or_buffer, kwds)

File /opt/conda/lib/python3.10/site-packages/pandas/io/parsers/readers.py:605,␣
  ↪in _read(filepath_or_buffer, kwds)
    602 _validate_names(kwds.get("names", None))
    604 # Create the parser.
--> 605 parser = TextFileReader(filepath_or_buffer, **kwds)
    607 if chunksize or iterator:
    608     return parser

File /opt/conda/lib/python3.10/site-packages/pandas/io/parsers/readers.py:1442,␣
  ↪in TextFileReader.__init__(self, f, engine, **kwds)
   1439     self.options["has_index_names"] = kwds["has_index_names"]
   1441 self.handles: IOHandles | None = None
-> 1442 self._engine = self._make_engine(f, self.engine)

File /opt/conda/lib/python3.10/site-packages/pandas/io/parsers/readers.py:1735,␣
  ↪in TextFileReader._make_engine(self, f, engine)
   1733     if "b" not in mode:
   1734         mode += "b"
-> 1735 self.handles = get_handle(
   1736     f,
   1737     mode,
   1738     encoding=self.options.get("encoding", None),
   1739     compression=self.options.get("compression", None),
   1740     memory_map=self.options.get("memory_map", False),
   1741     is_text=is_text,
   1742     errors=self.options.get("encoding_errors", "strict"),
   1743     storage_options=self.options.get("storage_options", None),
   1744 )
   1745 assert self.handles is not None
   1746 f = self.handles.handle

File /opt/conda/lib/python3.10/site-packages/pandas/io/common.py:856, in␣
  ↪get_handle(path_or_buf, mode, encoding, compression, memory_map, is_text,␣
  ↪errors, storage_options)
    851 elif isinstance(handle, str):
    852     # Check whether the filename is to be opened in binary mode.
    853     # Binary mode does not support 'encoding' and 'newline'.
    854     if ioargs.encoding and "b" not in ioargs.mode:
    855         # Encoding
--> 856         handle = open(
    857             handle,
    858             ioargs.mode,
    859             encoding=ioargs.encoding,
    860             errors=errors,
    861             newline="",
    862         )
    863     else:
    864         # Binary mode
    865         handle = open(handle, ioargs.mode)
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'data.csv'
```

[ ]:

```
# 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 Pe less than␣
 ↪85000.
```

[ ]:

```
import pandas as pd

filtered_df = People_Dataset[(People_Dataset['Last Name'].str.contains('Duke'))␣
 ↪&
                             (People_Dataset['Gender'].str.contains('Female')) &
                             (People_Dataset['Salary'] < 85000)]


print(filtered_df)
```

[ ]:

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

[4]:

```
import pandas as pd
import numpy as np


random_integers = np.random.randint(1, 7, size=35)


random_matrix = random_integers.reshape(7, 5)


df = pd.DataFrame(random_matrix)


print(df)
```

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

[ ]:

```
# Q. 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.
```

[5]:

```
import pandas as pd
import numpy as np


np.random.seed(0)


series1 = pd.Series(np.random.randint(10, 51, size=50), name='col1')


series2 = pd.Series(np.random.randint(100, 1001, size=50), name='col2')


df = pd.concat([series1, series2], axis=1)


print(df)
```

```
    col1  col2
0     10   153
1     13   650
2     13   588
3     49   856
4     19   373
5     29   435
6     31   488
7     46   717
8     33   142
9     16   542
10    34   643
11    34   988
12    22   357
13    11   421
14    48   157
15    49   391
16    33   970
17    34   219
18    27   879
19    47   530
```

```
20     35    182
21     23    191
22     18    996
23     19    498
24     30    711
25     26    665
26     15    733
27     25    184
28     10    303
29     28    424
30     45    874
31     34    147
32     39    739
33     29    231
34     29    968
35     24    280
36     49    946
37     42    243
38     11    760
39     19    327
40     42    891
41     41    819
42     20    473
43     33    953
44     45    660
45     21    405
46     38    681
47     44    269
48     10    775
49     10    548
```

[ ]:

```
# Q. 12. Perform the following operations using people data set:

#a) Delete the 'Email', 'Phone', and 'Date of birth' columns from the dataset.

#b) Delete the rows containing any missing values.

#d) Print the final output also.
```

[ ]:

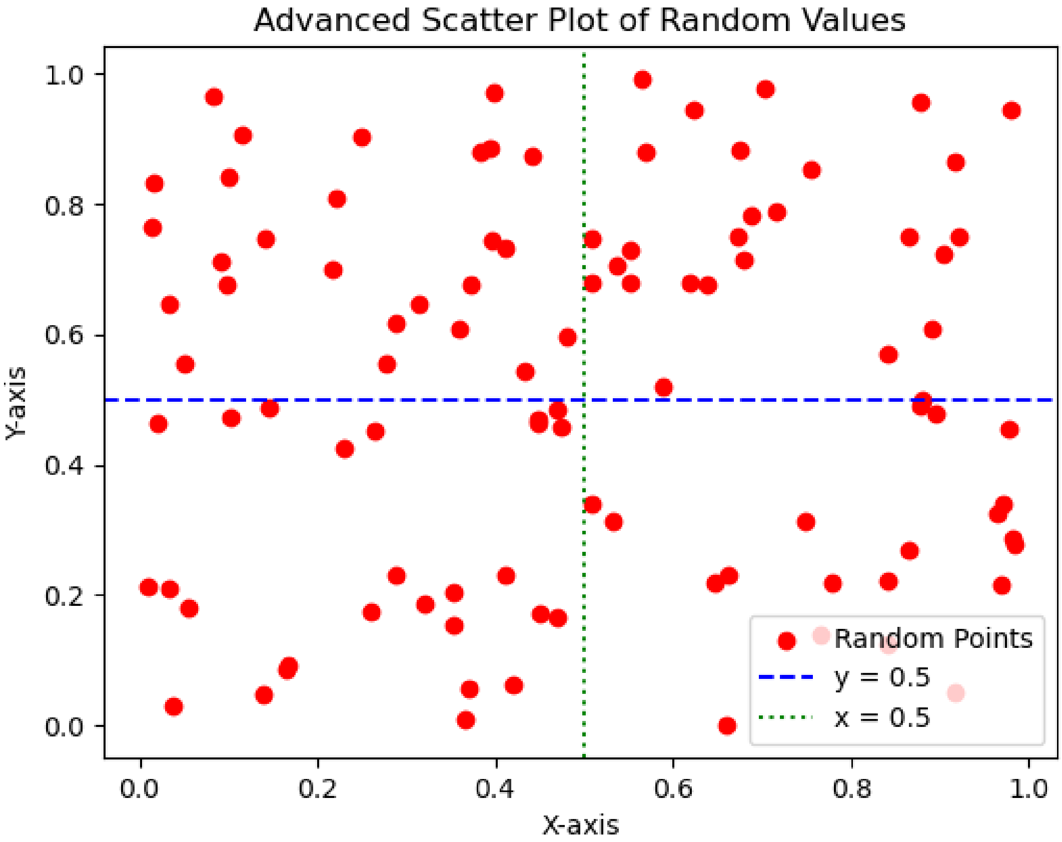
```
import pandas as pd

people_data.drop(['Email', 'Phone', 'Date of birth'], axis=1, inplace=True)

people_data.dropna(inplace=True)

print(people_data)
```

[ ]:

```
# Q.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.
```

[6]:

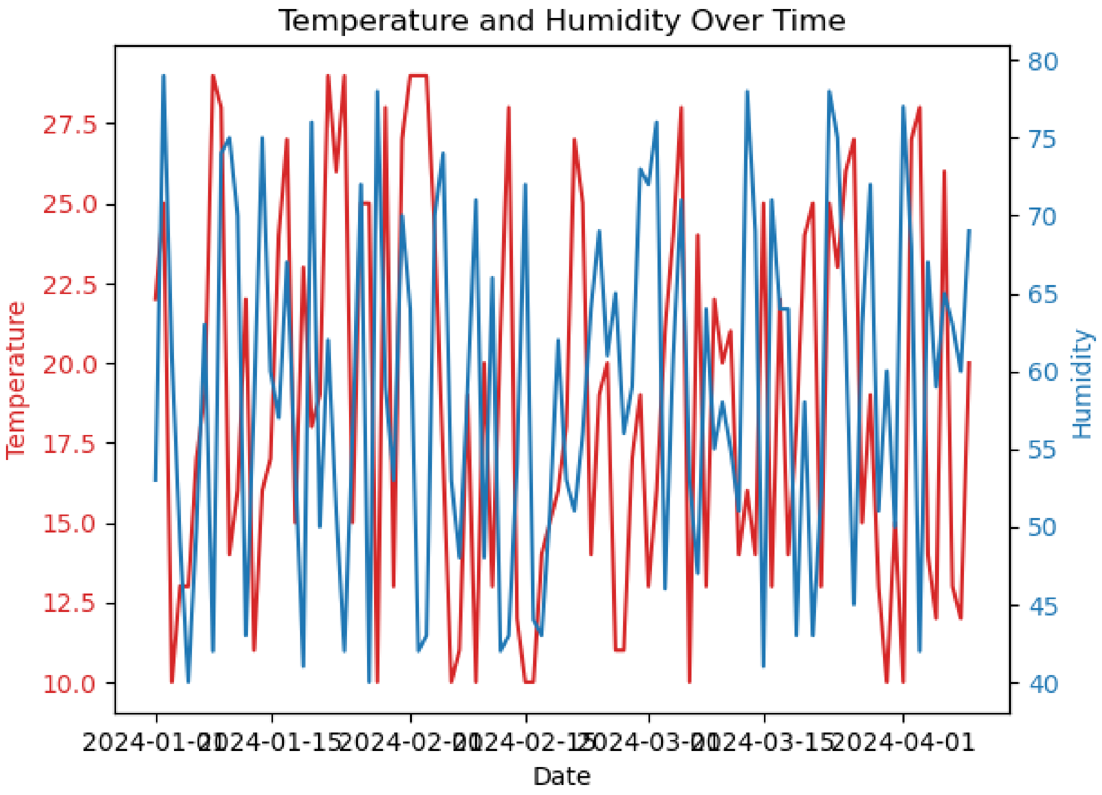
```
import numpy as np
import matplotlib.pyplot as plt

x = np.random.rand(100)
y = np.random.rand(100)

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

plt.axhline(y=0.5, linestyle='--', color='blue', label='y = 0.5')

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

plt.xlabel('X-axis')
plt.ylabel('Y-axis')

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

plt.legend()

plt.show()
```



[ ]:
```
# Q. 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'.
```

[7]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np


np.random.seed(0)
dates = pd.date_range('2024-01-01', periods=100)
temperature = np.random.randint(10, 30, size=100)
```

```
humidity = np.random.randint(40, 80, size=100)

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

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)

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

plt.show()
```

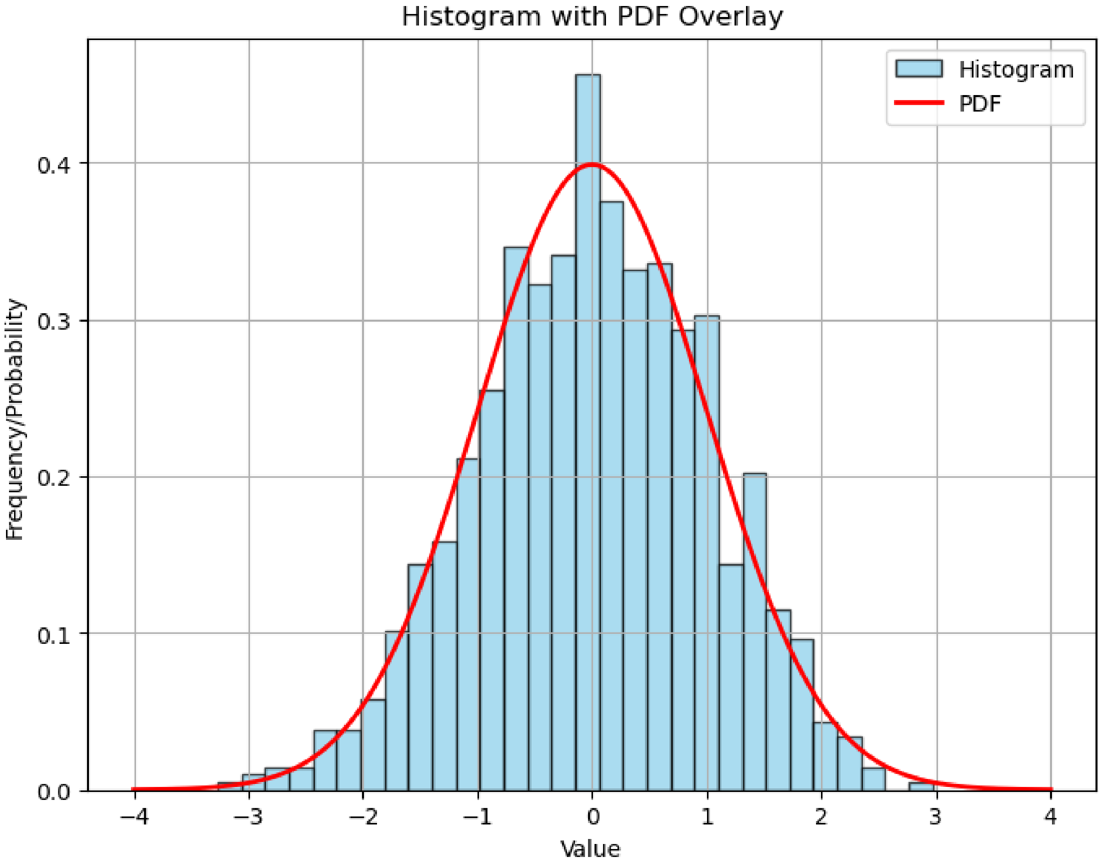


```
[ ]: # Q.15  Create a NumPy array data containing 1000 samples from a normal␣
      ↪distribution. Perform the following
     #tasks using Matplotlib:

     #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

     data = np.random.randn(1000)

     plt.figure(figsize=(8, 6))
     plt.hist(data, bins=30, density=True, color='skyblue', edgecolor='black',␣
      ↪alpha=0.7, label='Histogram')

     x = np.linspace(-4, 4, 1000)
     pdf = 1/(np.sqrt(2*np.pi)) * np.exp(-0.5*x**2)
     plt.plot(x, pdf, color='red', linewidth=2, label='PDF')

     plt.xlabel('Value')
     plt.ylabel('Frequency/Probability')

     plt.title('Histogram with PDF Overlay')

     plt.legend()

     plt.grid(True)
     plt.show()
```

```
[ ]: # Q.16 Set the title of the plot as 'Histogram with PDF Overlay'.
```

```
[8]: import numpy as np
     import matplotlib.pyplot as plt

     data = np.random.randn(1000)

     plt.figure(figsize=(8, 6))
     plt.hist(data, bins=30, density=True, color='skyblue', edgecolor='black',␣
      ↪alpha=0.7, label='Histogram')

     x = np.linspace(-4, 4, 1000)
     pdf = 1/(np.sqrt(2*np.pi)) * np.exp(-0.5*x**2)
     plt.plot(x, pdf, color='red', linewidth=2, label='PDF')

     plt.xlabel('Value')
     plt.ylabel('Frequency/Probability')

     plt.title('Histogram with PDF Overlay')

     plt.legend()

     plt.grid(True)
     plt.show()
```

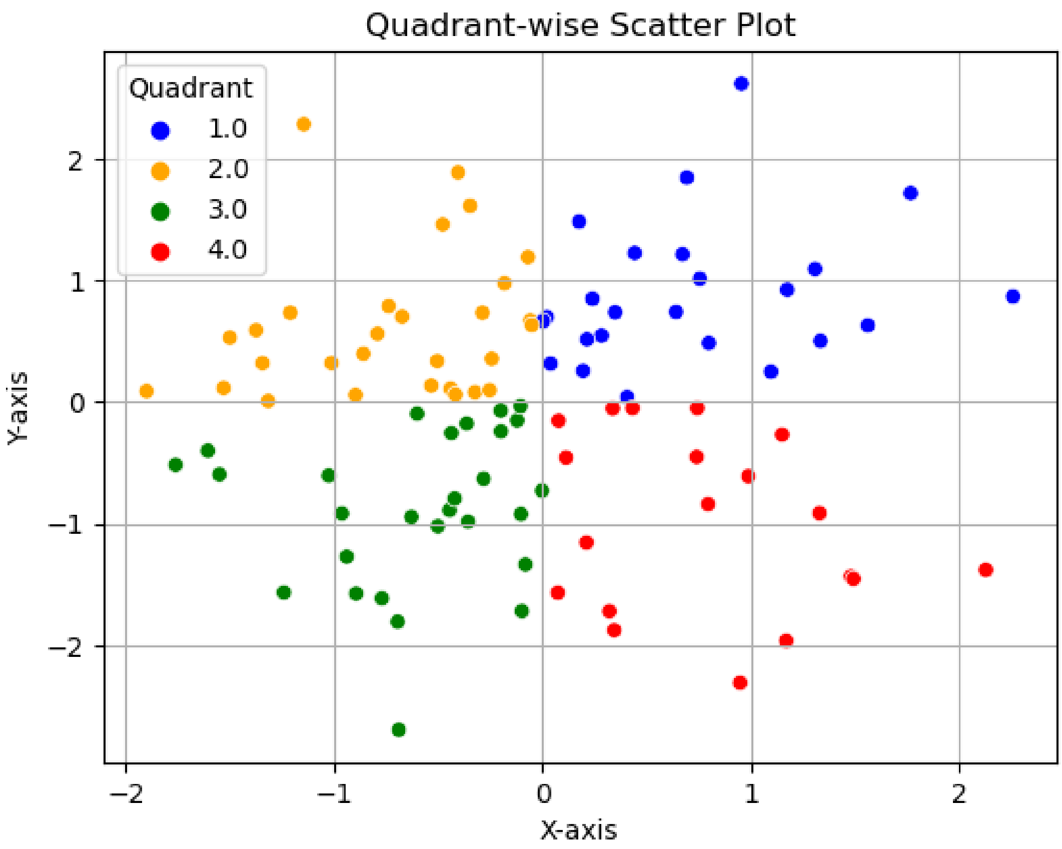


[ ]:
```
# Q.17 Create a Seaborn scatter plot of two random arrays, color points based␣
 ↪on their position relative to the
```

```
#origin (quadrants), add a legend, label the axes, and set the title as␣
 ↪'Quadrant-wise Scatter Plot'.
```

[9]:

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

x = np.random.randn(100)
y = np.random.randn(100)

quadrants = np.zeros(len(x))
quadrants[(x > 0) & (y > 0)] = 1  # Quadrant 1
quadrants[(x < 0) & (y > 0)] = 2  # Quadrant 2
quadrants[(x < 0) & (y < 0)] = 3  # Quadrant 3
quadrants[(x > 0) & (y < 0)] = 4  # Quadrant 4

data = {'x': x, 'y': y, 'quadrant': quadrants}
df = pd.DataFrame(data)

palette = {1: 'blue', 2: 'orange', 3: 'green', 4: 'red'}

sns.scatterplot(data=df, x='x', y='y', hue='quadrant', palette=palette,␣
 ↪legend='full')

plt.xlabel('X-axis')
plt.ylabel('Y-axis')

plt.title('Quadrant-wise Scatter Plot')

plt.legend(title='Quadrant')
plt.grid(True)
plt.show()
```



```
[ ]: # Q. 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'.
```

```
[ ]: from bokeh.plotting import figure, show
     from bokeh.models import ColumnDataSource

     import numpy as np

     x = np.linspace(0, 2*np.pi, 100)
     y = np.sin(x)

     source = ColumnDataSource(data=dict(x=x, y=y))

     p = figure(title="Sine Wave Function", x_axis_label='X-axis',␣
      ↪y_axis_label='Y-axis')

     p.line('x', 'y', source=source, line_width=2)

     p.grid.visible = True
```

```python

show(p)
```

[ ]:

```python
# Q.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'.
```

[ ]:

```python
from bokeh.plotting import figure, show
from bokeh.models import HoverTool
import random

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

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

p.vbar(x=categories, top=values, width=0.5, color=["blue", "orange", "green",␣
 ↪"red", "purple"])

hover = HoverTool()
hover.tooltips = [('Value', '@top')]
p.add_tools(hover)

p.xaxis.major_label_orientation = 1.2

show(p)
```

[ ]:

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

[ ]:

```python
import plotly.graph_objs as go
import numpy as np

x = np.linspace(0, 10, 100)
y = np.random.randn(100)

trace = go.Scatter(x=x, y=y, mode='lines', name='Random Data')

layout = go.Layout(title='Simple Line Plot', xaxis=dict(title='X-axis'),␣
 ↪yaxis=dict(title='Y-axis'))

fig = go.Figure(data=[trace], layout=layout)
```

```
fig.show()
```

[ ]:

```
# Q. 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.graph_objs as go
import numpy as np

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

trace = go.Pie(labels=labels, values=values, hoverinfo='label+percent',␣
 ↪textinfo='value+percent', textfont_size=20)

layout = go.Layout(title='Interactive Pie Chart')

fig = go.Figure(data=[trace], layout=layout)

fig.show()
```