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.

```
Three Methods to Create Identical 2D Arrays in NumPy
Method 1: Using np.ones()

import numpy as np

# Create a 3x3 array of ones
array1 = np.ones((3, 3))

print(array1)
Use code with caution.

Output:

[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
Method 2: Using np.zeros() and assignment

# Create a 3x3 array of zeros
array2 = np.zeros((3, 3))

# Assign the same value to all elements
array2[:] = 2

print(array2)
Use code with caution.

Output:

[[2. 2. 2.]
 [2. 2. 2.]
 [2. 2. 2.]]
Method 3: Using np.full()

# Create a 3x3 array filled with the value 3
array3 = np.full((3, 3), 3)
```

```
print(array3)
Use code with caution.

Output:

[[3 3 3]
 [3 3 3]
 [3 3 3]]
```

2. 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
numbers = np.linspace(1, 10, 100)

# Reshape the 1D array into a 2D array with 10 rows and 10 columns
reshaped_array = numbers.reshape(10, 10)

print(reshaped_array)
```

3.Explain the following terms

(a). The difference in np.array, np.asarray and np.asanyarray.

The `np.array`, `np.asarray`, and `np.asanyarray` functions in NumPy are all used to create NumPy arrays from various input data types. However, they have subtle differences in their behavior:

np.array:

- **Converts any input data type to a NumPy array.**
- **Copies the data if necessary.**
- **If the input is already a NumPy array, it returns a view of the original array.**

np.asarray:

- **Converts any input data type to a NumPy array.**
- **Tries to avoid copying the data if possible.**
- **If the input is already a NumPy array, it returns the original array without copying.**

np.asanyarray:

- **Converts any input data type to a NumPy array.**
- **Tries to avoid copying the data if possible.**
- **If the input is already a NumPy array, it returns the original array without copying, even if it's not contiguous.**

In summary, `np.array` always copies the data if necessary, while `np.asarray` and `np.asanyarray` try to avoid copying if possible. The main difference between `np.asarray` and `np.asanyarray` is that `np.asanyarray` will return a non-contiguous view of the original array if it's already a NumPy array but not contiguous.

Here's a table summarizing the differences:

Function	Behavior
np.array	Always copies data if necessary
np.asarray	Tries to avoid copying data if possible
np.asanyarray	Tries to avoid copying data if possible, even if the input is a non-contiguous NumPy array

b. The difference between Deep copy and shallow copy.

Deep Copy vs. Shallow Copy:

In Python, when you create a copy of an object, it can be either a deep copy or a shallow copy. The key difference lies in how the object's contents are copied.

Shallow Copy:

- A shallow copy creates a new object but references the same underlying data as the original object.
- If the original object contains mutable data structures (like lists or dictionaries), modifying the copy will also modify the original object.

Deep Copy:

- A deep copy creates a new object and recursively copies all the contents of the original object, including nested objects.
- If the original object contains mutable data structures, modifying the copy will not affect the original object.

Example:

```
import copy

original_list = [1, 2, [3, 4]]
shallow_copy = copy.copy(original_list)
deep_copy = copy.deepcopy(original_list)

# Modify
 the original list
original_list[2][0] = 5

print("Original list:", original_list)
print("Shallow copy:", shallow_copy)
print("Deep copy:", deep_copy)
```

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
random_array = np.random.uniform(5, 20, (3, 3))

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

print(rounded_array)
```

5. Create a NumPy array with random integers between 1 and 10 of shape (5, 6). After creating the array.

```
import numpy as np
```

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

# Print the original array
print("Original array:")
print(random_array)

# Extract all even integers
even_integers = random_array[random_array % 2 == 0]

# Print even integers
print("Even integers:")
print(even_integers)

# Extract all odd integers
odd_integers = random_array[random_array % 2 != 0]

# Print odd integers
print("Odd integers:")
print(odd_integers)
```

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)

```
import numpy as np

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

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

# b) Perform element-wise multiplication of the array with itself
multiplied_array = array * array

# Print the original array, max indices, and multiplied array
print("Original 3D array:\n", array)
```

```
print("\nIndices of maximum values along each depth level:\n", max_indices)
print("\nElement-wise multiplied array:\n", multiplied_array)
```

8. 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

df = pd.read_csv('People Data.csv')

# Function to clean and convert 'Phone' column, including handling null values
def clean_phone_column(df):
    # Fill null or empty values with a default phone number (e.g., '0000000000')
    df['Phone'] = df['Phone'].fillna('0000000000').replace('', '0000000000')

    # Remove non-numeric characters using regex (including dots, parentheses, dashes)
    df['Phone'] = df['Phone'].str.replace(r'\D', '', regex=True)

    # Convert the 'Phone' column to numeric (integer type)
    df['Phone'] = pd.to_numeric(df['Phone'], errors='coerce')  # 'coerce' will handle any remaining issues by setting invalid parsing to NaN

    return df

# Clean the 'Phone' column
df = clean_phone_column(df)

# Display the cleaned DataFrame
print("Cleaned DataFrame:\n", df)

# Display table attributes and data types
print("\nTable Attributes and Data Types:")
print(df.dtypes)
```

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

```
import pandas as pd
df = pd.read_csv('D:\\Laptop data\\pw_skills_Data_Analysis\\python\\python
assignment\\8th assignment\\People Data.csv')

# a) Read the 'data.csv' file, skipping the first 50 rows

df_only_50 = pd.read_csv('D:\\Laptop
data\\pw_skills_Data_Analysis\\python\\python assignment\\8th assignment\\People
Data.csv',skiprows=50)


# b) Only read the specified columns: 'Last Name', 'Gender', 'Email', 'Phone',
and 'Salary'
df_filtered = df[['Last Name','Gender','Email','Phone','Salary']]

# c) Display the first 10 rows of the filtered dataset
print("First 10 rows of the filtered dataset:")
print(df_filtered.head(10))

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

9. Filter and select rows from the People_Dataset, where the "Last Name' column contains the name 'Duke', 'Gender' column contains the word Female and 'Salary' should be less than 85000.

```
import pandas as pd
```

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

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 35 random integers between 1 and 6
random_numbers = np.random.randint(1, 7, size=35)

# Reshape the array into a 7x5 matrix
data = random_numbers.reshape(7, 5)

# Create a DataFrame from the data
df = pd.DataFrame(data)

# Print the DataFrame
print(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 joining these Series by column, and, change the names of the columns to 'col1', 'col2', etc.

```
import pandas as pd
import numpy as np

# Create two Series with random numbers
series1 = pd.Series(np.random.randint(10, 51, size=50))
series2 = pd.Series(np.random.randint(100, 1001, size=50))

# Create a DataFrame from the Series
df = pd.concat([series1, series2], axis=1)

# Rename the columns
df.columns = ['col1', 'col2']

# Print the DataFrame
print(df)
```

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 = pd.read_csv('D:\\Laptop
data\\pw_skills_Data_Analysis\\python\\python
assignment\\8th assignment\\People Data.csv')


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

# b) Delete rows with missing values
people_data = people_data.dropna()

# Print the final output
print(people_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 horizontal and vertical lines
plt.axhline(y=0.5, color='green', linestyle='--', label='y =
0.5')
```

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

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

# Display legend
plt.legend()

# 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.

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

# Sample time-series dataset
data = {
    'Date': pd.date_range(start='2023-01-01', periods=10, freq='D'),
    'Temperature': [30, 32, 31, 29, 28, 35, 34, 33, 31, 30],
    'Humidity': [65, 70, 72, 68, 67, 66, 64, 63, 62, 60]
}

# Create DataFrame
```

```
df = pd.DataFrame(data)

# Plotting with two y-axes
fig, ax1 = plt.subplots()

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

# Create a second y-axis for Humidity
ax2 = ax1.twinx()
ax2.plot(df['Date'], df['Humidity'], color='tab:blue', label='Humidity')
ax2.set_ylabel('Humidity (%)', color='tab:blue')
ax2.tick_params(axis='y', labelcolor='tab:blue')

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

# Show 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'.

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

```
from scipy.stats import norm

# a) Generate data with 1000 samples from a normal distribution
data = np.random.normal(loc=0, scale=1, size=1000)

# b) Plot the histogram with 30 bins and overlay the PDF
plt.hist(data, bins=30, density=True, alpha=0.6, color='skyblue',
edgecolor='black', label='Data Histogram')

# Overlay PDF
xmin, xmax = plt.xlim()  # Get x-axis range for plotting PDF
x = np.linspace(xmin, xmax, 100)
pdf = norm.pdf(x, np.mean(data), np.std(data))
plt.plot(x, pdf, color='red', linewidth=2, label='Normal PDF')

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

# d) Set the title
plt.title('Histogram with PDF Overlay')
plt.legend()

# Show 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'.

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

# Generate two random arrays for x and y coordinates
np.random.seed(0)  # For reproducibility
x = np.random.uniform(-10, 10, 100)
y = np.random.uniform(-10, 10, 100)
```

```
# Determine quadrant for each point
def get_quadrant(x, y):
    if x > 0 and y > 0:
        return 'Quadrant I'
    elif x < 0 and y > 0:
        return 'Quadrant II'
    elif x < 0 and y < 0:
        return 'Quadrant III'
    elif x > 0 and y < 0:
        return 'Quadrant IV'
    else:
        return 'On Axis'

# Create DataFrame for plotting
df = pd.DataFrame({'x': x, 'y': y})
df['Quadrant'] = [get_quadrant(xi, yi) for xi, yi in zip(df['x'], df['y'])]

# Create Seaborn scatter plot with color based on quadrant
plt.figure(figsize=(8, 6))
scatter_plot = sns.scatterplot(data=df, x='x', y='y', hue='Quadrant',
palette='tab10', s=70, edgecolor='k')

# Label axes and set the title
plt.xlabel('X-Axis')
plt.ylabel('Y-Axis')
plt.title('Quadrant-wise Scatter Plot')
plt.axhline(0, color='black',linewidth=0.5)  # Add origin lines
plt.axvline(0, color='black',linewidth=0.5)
plt.legend(title='Position Relative to Origin')

# 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'.

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

```
# Display plot inline in notebook
output_notebook()

# Generate data for the sine wave
x = np.linspace(0, 4 * np.pi, 100)
y = np.sin(x)

# Create a Bokeh figure
p = figure(title="Sine Wave Function", width=700, height=400)

# Plot the sine wave
p.line(x, y, line_width=2, color="blue", legend_label="y = sin(x)")

# Add grid lines (enabled by default)
p.grid.grid_line_color = "gray"

# Label the axes
p.xaxis.axis_label = "X-Axis"
p.yaxis.axis_label = "Y-Axis"

# Show the plot
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'.

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

# Display plot inline in notebook
output_notebook()
```

```
# Generate random categorical data
categories = ['A', 'B', 'C', 'D', 'E']
values = np.random.randint(10, 100, size=len(categories))

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

# Create a ColumnDataSource
source = ColumnDataSource(df)

# Create a Bokeh figure for the bar chart
p = figure(x_range=categories, title="Random Categorical Bar Chart", width=700,
height=400)

# Color map based on values
color_mapper = factor_cmap('Category', palette="Viridis256", factors=categories)

# Plot the bars
p.vbar(x='Category', top='Value', width=0.5, source=source, color=color_mapper)

# Add hover tool to show exact values
hover = HoverTool()
hover.tooltips = [("Category", "@Category"), ("Value", "@Value")]
p.add_tools(hover)

# Label the axes
p.xaxis.axis_label = "Category"
p.yaxis.axis_label = "Value"

# 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.graph_objects as go
import numpy as np

# Generate random data for the line plot
```

```
x = np.arange(0, 50)
y = np.random.randint(0, 100, size=50)

# Create a Plotly line plot
fig = go.Figure()

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

# Label the axes and set the title
fig.update_layout(
    title='Simple Line Plot',
    xaxis_title='X-Axis',
    yaxis_title='Y-Axis'
)

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

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_objects as go
import numpy as np

# Generate random data for the pie chart
labels = ['Category A', 'Category B', 'Category C', 'Category D']
values = np.random.randint(10, 100, size=len(labels))

# Create a Plotly pie chart
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=0)])

# Add title and enable display of percentages
fig.update_traces(textinfo='percent+label')

# Set the title
fig.update_layout(title='Interactive Pie Chart')

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