Experiment No.1-Part B

Title: Understanding Data with Programming Exercise

Batch: FDS-2 **Roll No.: 16010422234** **Name: Chandana Ramesh Galgali**

Experiment No.:1 -B

Aim: Understanding of the Data Handling technique with Programming exercise

Resources needed: Python, Data, Numpy and Pandas library

Theory:

There are several programming languages that are commonly used for data handling, analysis, and manipulation tasks. Some of the most popular data handling programming languages are Python, R, Scala, Matlab, Julia and SQL. In this practical session we will handle python based data handling commands. Let us see features of python.

Python:

Python is one of the most widely used programming languages for data handling and analysis. It has rich libraries like Pandas, NumPy, Matplotlib, Seaborn, and SciPy that offer comprehensive support for data manipulation, numerical computing, and data visualization. Python's readability and user-friendly syntax make it a top choice for data scientists, researchers, and analysts. In this session we will learn some important python libraries namely, Numpy and Pandas.

NumPy:

NumPy, which stands for "Numerical Python," is a fundamental package for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with an extensive collection of mathematical functions to operate on these arrays efficiently. NumPy is the backbone of many other Python libraries for scientific computing and data analysis.

Key features of NumPy:

- N-dimensional array object (numpy.ndarray): Efficient data structure for storing and manipulating large datasets, making it suitable for numerical computations and data analysis.
- Broadcasting: NumPy automatically broadcasts arrays with different shapes for element-wise operations, reducing the need for explicit loops.
- Mathematical Functions: NumPy offers a wide range of mathematical functions like trigonometric, exponential, statistical, and linear algebra functions, among others.
- Indexing and Slicing: NumPy supports advanced indexing techniques to access specific elements or subsets of data quickly.
- Integration with C/C++: NumPy is written in C and Python, which provides the advantage of speed and efficiency in numerical computations.

Pandas:

Pandas is an open-source data manipulation library built on top of NumPy. It provides high-level data structures and functions to make working with structured data easy and intuitive. The two primary data structures in Pandas are the Series and DataFrame.

Key features of Pandas :

- Series: A one-dimensional labeled array, similar to a NumPy array but with additional indexing capabilities, making it more versatile for data analysis.
- DataFrame: A two-dimensional labeled data structure with rows and columns, resembling a spreadsheet or SQL table, which is the central data structure for data analysis in Pandas.
- Data Import/Export: Pandas can read and write data from various file formats like CSV, Excel, SQL databases, JSON, and HTML tables.
- Data Exploration: Pandas provides functions to explore, clean, and preprocess data, such as handling missing values, filtering, grouping, and aggregation.
- Time-Series Analysis: Pandas has robust support for handling time-series data, making it popular for financial and temporal data analysis.
- Data Visualization: Although not a visualization library, Pandas integrates well with other visualization libraries like Matplotlib and Seaborn, allowing easy creation of basic visualizations directly from DataFrames and Series.
- Together, NumPy and Pandas form a powerful combination for data analysis and data manipulation tasks in Python. NumPy handles the underlying numerical computations and efficient data storage, while Pandas provides high-level structures and functions that simplify data handling and analysis tasks. They are fundamental libraries in the Python data science ecosystem and are widely used by data scientists, researchers, and analysts.

Both Pandas and NumPy offer a wide range of commands and functions for data analysis.

Exercise Data Handling Commands with Python:

Both Pandas and NumPy offer a wide range of commands and functions for data analysis. Here are some essential commands from each library commonly used in data analysis:

Pandas:

1. Creating DataFrames:

   - `pd.DataFrame(data, columns=['col1', 'col2', ...])`: Creates a DataFrame from data (a dictionary, list of lists, or a NumPy array) with specified column names.

2. **Data Exploration:**

   - `df.head(n)`: Returns the first n rows of the DataFrame.
   - `df.info()`: Provides information about the DataFrame, including data types and non-null counts.
   - `df.describe()`: Generates summary statistics for numerical columns.

- `df.shape`: Returns the dimensions of the DataFrame (rows, columns).
- `df.columns`: Returns the column names.

3. **Data Selection and Filtering:**
   - `df['column_name']`: Selects a specific column from the DataFrame.
   - `df.loc[row_index, 'column_name']`: Accesses specific data using labels for both rows and columns.
   - `df.iloc[row_index, col_index]`: Accesses specific data using integer-based indexing.
   - `df[df['column_name'] > value]`: Filters data based on a condition.

4. **Data Aggregation:**
   - `df.groupby('column_name').agg({'other_column': 'mean'})`: Grouping and aggregation based on specific columns.
   - `df['column_name'].value_counts()`: Calculates the frequency of unique values in a column.

5. **Data Cleaning:**
   - `df.isnull()`: Checks for missing values in the DataFrame.
   - `df.dropna()`: Drops rows with missing values.
   - `df.fillna(value)`: Fills missing values with a specific value.

6. **Data Visualization:**
   - `df['column_name'].plot(kind='hist')`: Creates a histogram for a specific column.
   - `df.plot(x='x_column', y='y_column', kind='scatter')`: Creates a scatter plot.

NumPy:

1. **Creating Arrays:**
   - `np.array([1, 2, 3])`: Creates a 1-dimensional NumPy array.
   - `np.zeros((m, n))`: Creates an m x n array filled with zeros.
   - `np.ones((m, n))`: Creates an m x n array filled with ones.
   - `np.arange(start, stop, step)`: Creates an array with values from start to stop (exclusive) with a given step size.

2. **Array Operations:**
   - `np.mean(arr)`: Calculates the mean of the array.
   - `np.median(arr)`: Calculates the median of the array.
   - `np.sum(arr)`: Calculates the sum of the array elements.
   - `np.std(arr)`: Calculates the standard deviation of the array.

- `np.unique(arr)`: Returns the unique elements of the array.

3. **Array Indexing and Slicing:**
   - `arr[index]`: Accesses specific elements of the array using indexing.
   - `arr[start:stop]`: Slices the array from start to stop (exclusive).

4. **Array Broadcasting:**
   - NumPy automatically broadcasts arrays with different shapes for element-wise operations.

5. **Mathematical Functions:**
   - `np.sin(arr)`, `np.cos(arr)`, `np.exp(arr)`, etc.: Various mathematical functions that operate element-wise on the array.

Procedure / Approach /Algorithm / Activity Diagram:

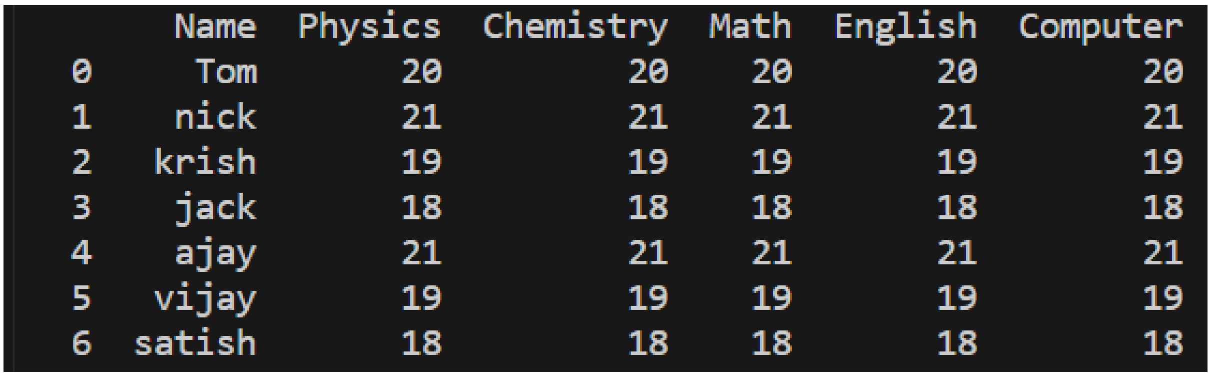
1. Install Numpy and Pandas library
2. Execute the data handling pandas commands
3. Execute the data handling numpy commands

Results: (Program printout with output / Document printout as per the format)

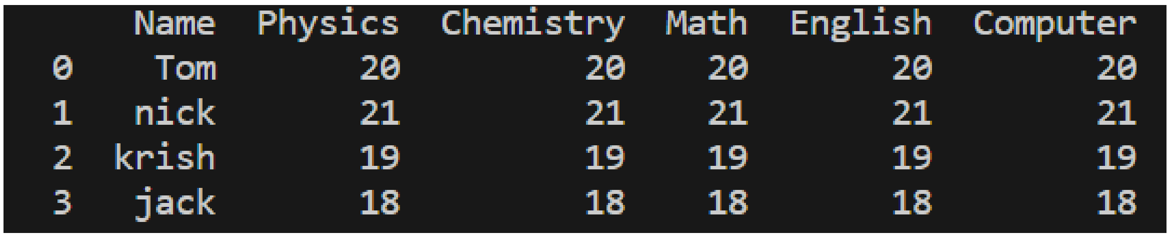
Execution of the data handling pandas commands:

```
import pandas as pd
#Initialize dictionary of student data
Student = {'Name':['Tom', 'nick', 'krish',
'jack','ajay','vijay','satish'],
        'Physics':[20, 21, 19, 18,21, 19, 18],
        'Chemistry':[20, 21, 19, 18,21, 19, 18],
        'Math':[20, 21, 19, 18,21, 19, 18],
        'English':[20, 21, 19, 18,21, 19, 18],
        'Computer':[20, 21, 19, 18,21, 19, 18]
        }
#1.Creating DataFrame:
df = pd.DataFrame(Student)
print(df) #Print the output with index
#2.Data Exploration:
print(df.head(4)) #Returns the first n rows of the DataFrame.
print(df.info()) #Provides information about the DataFrame, including
data types and non-null counts.

print(df.describe()) #Generates summary statistics for numerical
columns.

print(df.shape) #Returns the dimensions of the DataFrame (rows,
columns).

print(df.columns) #Returns the column names.


#3.Data Selection and Filtering:

print(df['Name']) #Selects a specific column from the DataFrame.

print(df.loc[0, 'Physics']) #Accesses specific data using labels for
both rows and columns.

print(df.iloc[3, 0]) #Accesses specific data using integer-based
indexing.

print(df[df['Computer'] >19 ])  #Filters data based on a condition.


#4.Data Aggregation:

print(df.groupby('Name').agg({'English': 'mean'})) #Grouping and
aggregation based on specific columns.

print(df['Chemistry'].value_counts()) #Calculates the frequency of
unique values in a column.


#5.Data Cleaning:

print(df.isnull()) #Checks for missing values in the DataFrame.

print(df.dropna()) #Drops rows with missing values.

print(df.fillna(0)) #Fills missing values with a specific value.


#6.Data Visualization:

print(df['Computer'].plot(kind='hist')) #Creates a histogram for a
specific column.

print(df.plot(x='Name', y='Math', kind='scatter')) #Creates a scatter
plot.
```

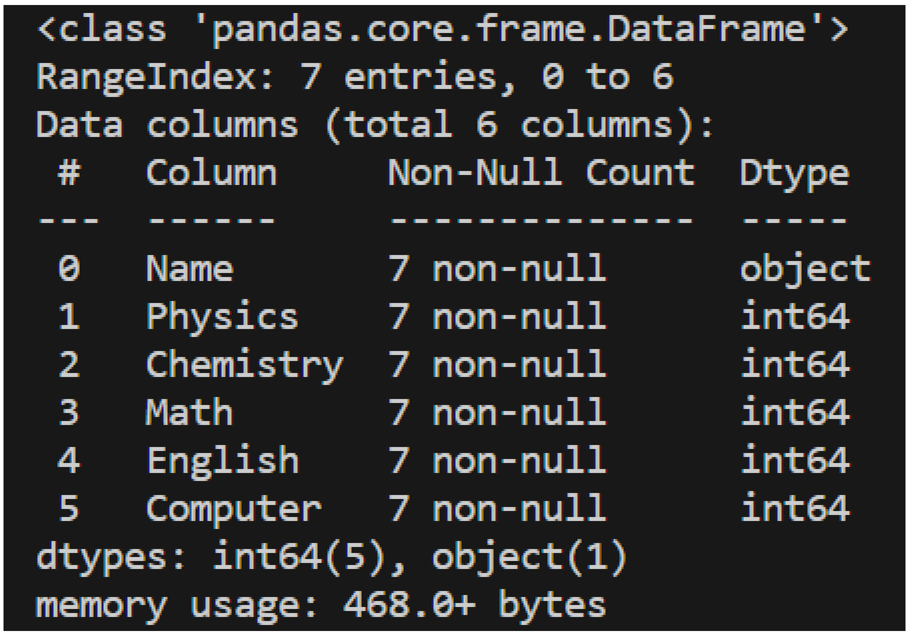
Output:



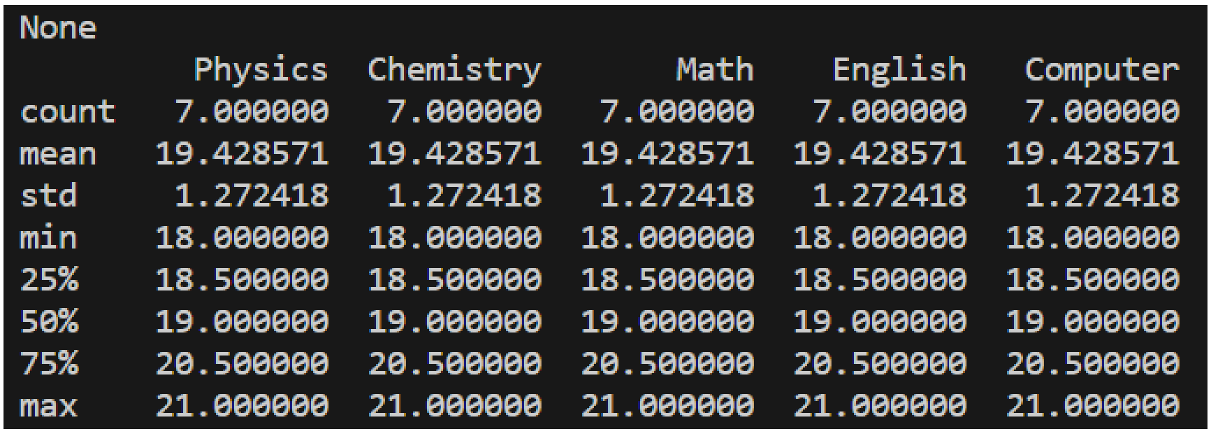
```
     Name  Physics  Chemistry  Math  English  Computer
0     Tom       20         20    20       20        20
1    nick       21         21    21       21        21
2   krish       19         19    19       19        19
3    jack       18         18    18       18        18
4    ajay       21         21    21       21        21
5   vijay       19         19    19       19        19
6  satish       18         18    18       18        18
```



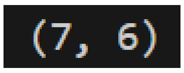
```
    Name  Physics  Chemistry  Math  English  Computer
0    Tom       20         20    20       20        20
1   nick       21         21    21       21        21
2  krish       19         19    19       19        19
3   jack       18         18    18       18        18
```



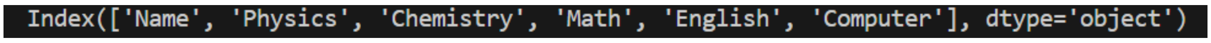
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   Name       7 non-null      object
 1   Physics    7 non-null      int64 
 2   Chemistry  7 non-null      int64 
 3   Math       7 non-null      int64 
 4   English    7 non-null      int64 
 5   Computer   7 non-null      int64 
dtypes: int64(5), object(1)
memory usage: 468.0+ bytes
```



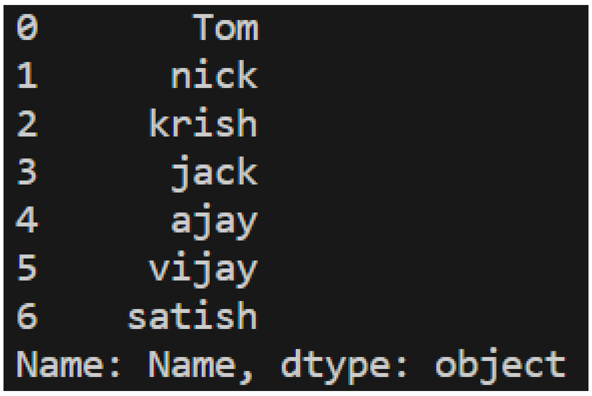
```
None
         Physics  Chemistry       Math    English   Computer
count   7.000000   7.000000   7.000000   7.000000   7.000000
mean   19.428571  19.428571  19.428571  19.428571  19.428571
std     1.272418   1.272418   1.272418   1.272418   1.272418
min    18.000000  18.000000  18.000000  18.000000  18.000000
25%    18.500000  18.500000  18.500000  18.500000  18.500000
50%    19.000000  19.000000  19.000000  19.000000  19.000000
75%    20.500000  20.500000  20.500000  20.500000  20.500000
max    21.000000  21.000000  21.000000  21.000000  21.000000
```



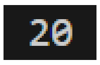
```
(7, 6)
```



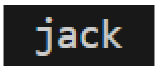
```
Index(['Name', 'Physics', 'Chemistry', 'Math', 'English', 'Computer'], dtype='object')
```



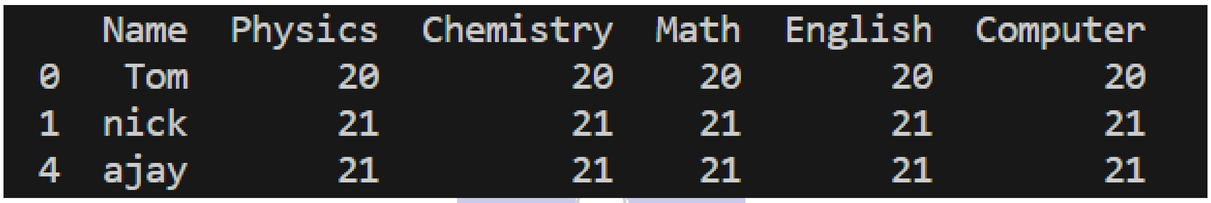
```
0        Tom
1       nick
2      krish
3       jack
4       ajay
5      vijay
6     satish
Name: Name, dtype: object
```



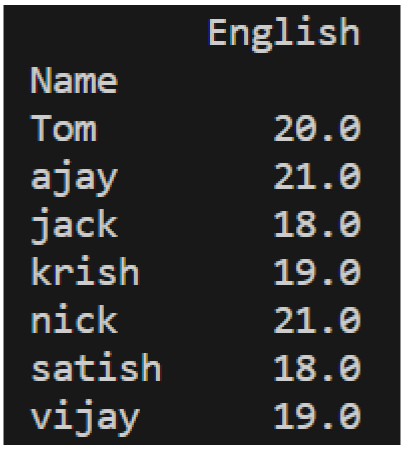
```
20
```



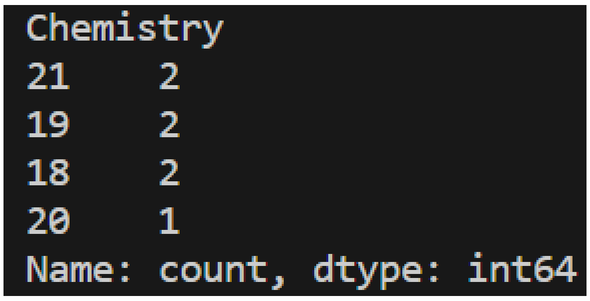
```
jack
```



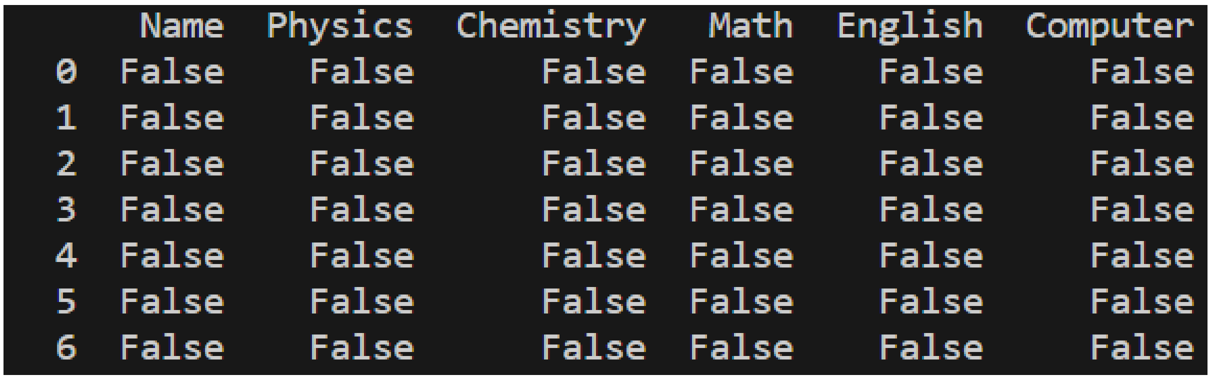
```
   Name  Physics  Chemistry  Math  English  Computer
0   Tom       20         20    20       20        20
1  nick       21         21    21       21        21
4  ajay       21         21    21       21        21
```



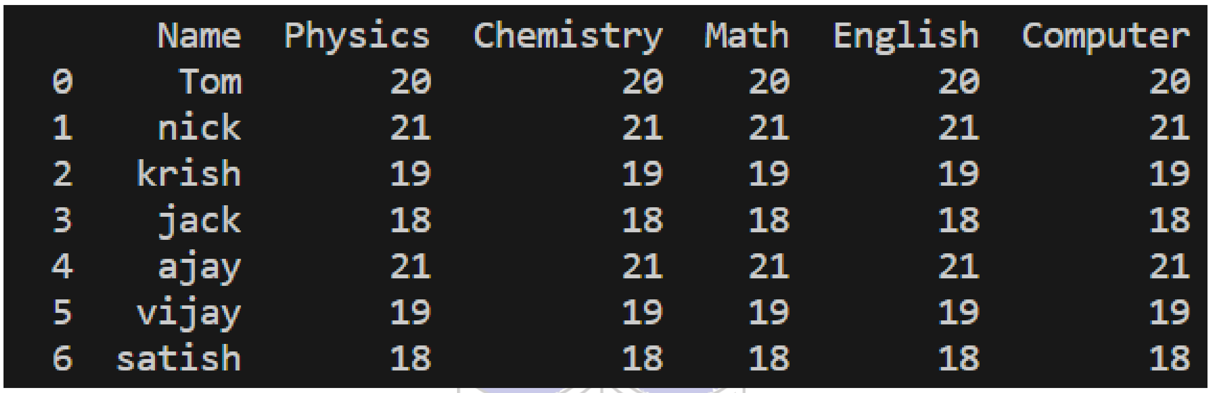
```
        English
Name
Tom        20.0
ajay       21.0
jack       18.0
krish      19.0
nick       21.0
satish     18.0
vijay      19.0
```



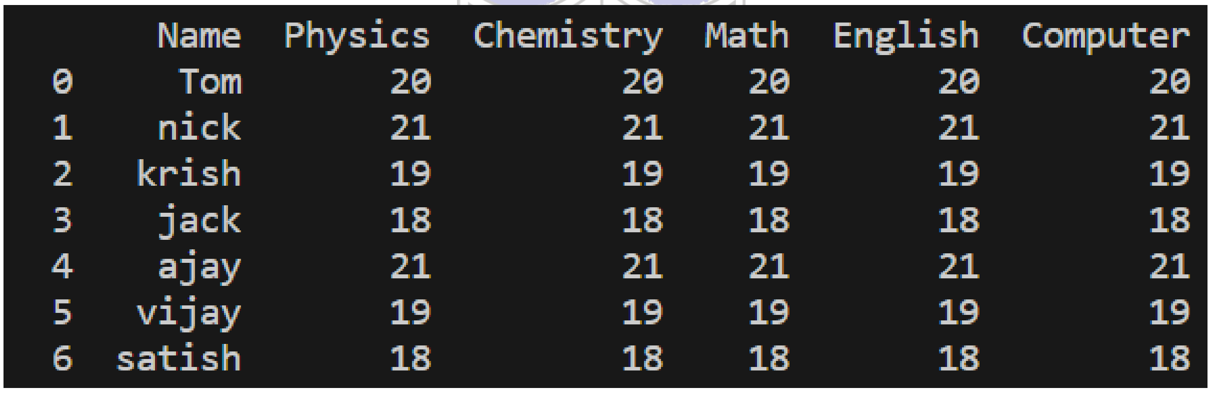
```
Chemistry
21     2
19     2
18     2
20     1
Name: count, dtype: int64
```



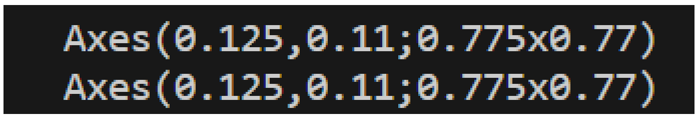
```
    Name  Physics  Chemistry   Math  English  Computer
0  False    False      False  False    False     False
1  False    False      False  False    False     False
2  False    False      False  False    False     False
3  False    False      False  False    False     False
4  False    False      False  False    False     False
5  False    False      False  False    False     False
6  False    False      False  False    False     False
```



```
     Name  Physics  Chemistry  Math  English  Computer
0     Tom       20         20    20       20        20
1    nick       21         21    21       21        21
2   krish       19         19    19       19        19
3    jack       18         18    18       18        18
4    ajay       21         21    21       21        21
5   vijay       19         19    19       19        19
6  satish       18         18    18       18        18
```



```
     Name  Physics  Chemistry  Math  English  Computer
0     Tom       20         20    20       20        20
1    nick       21         21    21       21        21
2   krish       19         19    19       19        19
3    jack       18         18    18       18        18
4    ajay       21         21    21       21        21
5   vijay       19         19    19       19        19
6  satish       18         18    18       18        18
```

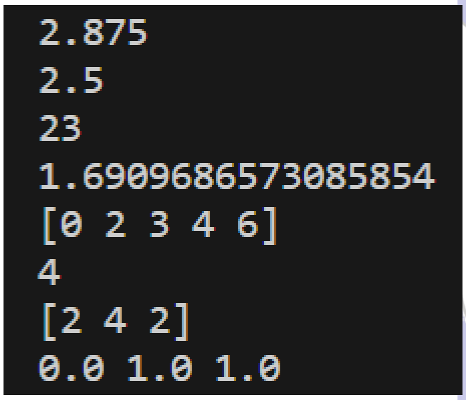


```
Axes(0.125,0.11;0.775x0.77)
Axes(0.125,0.11;0.775x0.77)
```

Execution of the data handling numpy commands:

```
import numpy as np


#1.Creating Arrays:

arr=np.array([4, 0, 2, 4, 2, 3, 2, 6]) #Creates a 1-dimensional NumPy
array.


#2.Array Operations:

print(np.mean(arr)) #Calculates the mean of the array.

print(np.median(arr)) #Calculates the median of the array.
print(np.sum(arr)) #Calculates the sum of the array elements.

print(np.std(arr)) #Calculates the standard deviation of the array.

print(np.unique(arr)) #Returns the unique elements of the array.


#3.Array Indexing and Slicing:

print(arr[3]) #Accesses specific elements of the array using indexing.

print(arr[2:5]) #Slices the array from start to stop (exclusive).


#5.Mathematical Functions:

print(np.sin(arr[1]), np.cos(arr[1]), np.exp(arr[1])) #Various
mathematical functions that operate element-wise on the array.
```

Output:



```
2.875
2.5
23
1.6909686573085854
[0 2 3 4 6]
4
[2 4 2]
0.0 1.0 1.0
```

Questions:

1. **How do you load data from a CSV file into a Pandas DataFrame?**

   To load data from a CSV file into a Pandas DataFrame, you can use the read_csv() function provided by the Pandas library. The read_csv() function reads the CSV file and creates a DataFrame object df containing the data from the file. Finally, you can use the print() function to display the contents of the DataFrame.

2. **When to prefer numpy over pandas? Describe the situation in your own words.**

   Numpy and Pandas are both powerful libraries in Python for data manipulation and analysis, but they have different use cases and excel in different scenarios.

   You might prefer to use Numpy over Pandas in the following situations:

   1. Mathematical and numerical operations: Numpy is designed for efficient numerical computations and provides a multidimensional array object called ndarray. If you need to perform complex mathematical operations, linear

algebra, or numerical computations, Numpy's array operations are generally faster and more efficient than Pandas.

2. Handling large datasets: Numpy arrays are more memory-efficient compared to Pandas DataFrames. If you are working with large datasets that don't require complex data manipulation or analysis, using Numpy can help reduce memory usage and improve performance.
3. Speed and performance: Numpy is generally faster than Pandas for basic array operations and mathematical computations. If you are working with large arrays and performance is a critical factor, Numpy can provide significant speed advantages.
4. Low-level access and control: Numpy allows for low-level access and control over the data, such as direct indexing and slicing of arrays. If you need fine-grained control over the data at a lower level, Numpy provides more flexibility.

On the other hand, Pandas is more suitable in the following scenarios:

1. Data manipulation and analysis: Pandas provides a high-level, easy-to-use interface for data manipulation and analysis. It offers powerful data structures like DataFrames that allow for efficient handling of structured data, including indexing, filtering, grouping, and merging. If your focus is on data exploration, cleaning, and analysis, Pandas provides a wide range of functions and methods that simplify these tasks.
2. Working with heterogeneous data: Pandas DataFrames can handle different data types within a single structure, making it suitable for working with heterogeneous datasets. It provides convenient methods for handling missing data, transforming data, and dealing with time series data.
3. Data visualization: Pandas integrates well with other libraries like Matplotlib and Seaborn, making it easy to visualize and plot data. It provides built-in functions for generating various types of plots, histograms, and statistical visualizations.

In summary, Numpy is preferred when you need to perform numerical computations, work with large datasets, or require low-level control over the data. Pandas, on the other hand, is more suitable for data manipulation, analysis, and visualization tasks, especially when dealing with structured and heterogeneous data.

Outcomes: Summarize the data.

Conclusion: (Conclusion to be based on the objectives and outcomes achieved)

The experiment provided valuable hands-on experience in data handling techniques, empowering participants to effectively work with and analyze data using programming.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

1. *"Jake VanderPlas",* **Python Data Science Handbook,O** O'Reilly Publication
2. "Wes McKinney", Python for Data Analysis, 2nd Edition, O'Reilly Media, Inc. ISBN: 9781491957660
3. NumPy Documentation
4. NumPy user guide — NumPy v1.25 Manual
5. User Guide — pandas 2.0.3 documentation (pydata.org)
6. 10 minutes to pandas — pandas 2.0.3 documentation (pydata.org)
7. pandas documentation — pandas 2.0.3 documentation (pydata.org)