Vidyavardhini's College of Engineering and Technology

Department of Artificial Intelligence & Data Science

AY: 2024-25

Class:	**SE**	**Semester:**	**IV**
Course Code:	**CSL405**	**Course Name:**	**Skills Based Python Programming Lab**

Name of Student:	**Shravani Sandeep Raut**
Roll No. :	**48**
Experiment No.:	**12**
Title of the Experiment:	**Program to demonstrate DataFrame using Pandas**
Date of Performance:	**01/04/2025**
Date of Submission:	**08/04/2025**

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty : Mr. Raunak Joshi

Signature :

Date:

Aim: Program to demonstrate DataFrame using Pandas

Theory:

Pandas is a powerful open-source data analysis and manipulation library for Python. It provides flexible and efficient data structures for working with structured data. One of its core data structures is the **DataFrame**, which is a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure with labeled axes (rows and columns).

Introduction to Pandas DataFrame:

- **Pandas DataFrame:** A two-dimensional, labeled data structure similar to a table in a database, an Excel spreadsheet, or a data frame in R.
- **Key Features:**
    - **Labeled Rows and Columns:** Each row and column can have custom labels, enabling intuitive data access and manipulation.
    - **Heterogeneous Data:** Columns can contain different data types (integers, floats, strings, etc.).
    - **Size Mutable:** Rows and columns can be added or removed easily.
    - **Data Alignment:** Automatically aligns data based on labels during operations, handling missing data gracefully.

Creating a DataFrame:

A DataFrame can be created from various data structures, including:

- Lists of dictionaries.
- Dictionaries of lists or dictionaries.
- NumPy arrays.
- Existing Pandas Series.
- External data sources (e.g., CSV, Excel, SQL databases).

Implementation:

```
import pandas as pd
```

In [13]:

```
# Step 1: Creating a DataFrame using a dictionary
data_dict = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Age': [23, 30, 35, 40, 28],
    'City': ['New York', 'Los Angeles', 'Chicago', 'Houston', 'Phoenix'],
    'Salary': [70000, 80000, 120000, 100000, 95000]
}

df_from_dict = pd.DataFrame(data_dict)
```

In [14]:

```
# Step 2: Display the DataFrame
print("DataFrame from Dictionary:")
print(df_from_dict)
```

```
DataFrame from Dictionary:
      Name  Age         City  Salary
0    Alice   23     New York   70000
1      Bob   30  Los Angeles   80000
2  Charlie   35      Chicago  120000
3    David   40      Houston  100000
4      Eva   28      Phoenix   95000
```

In [15]:

```
# Step 3: Accessing a column (e.g., 'Age')
print("\nAccessing the 'Age' column:")
print(df_from_dict['Age'])
```

```
Accessing the 'Age' column:
0    23
1    30
2    35
3    40
4    28
Name: Age, dtype: int64
```

In [16]:

```
# Step 4: Accessing multiple columns (e.g., 'Name' and 'Salary')
print("\nAccessing 'Name' and 'Salary' columns:")
print(df_from_dict[['Name', 'Salary']])
```

```
Accessing 'Name' and 'Salary' columns:
      Name  Salary
0    Alice   70000
1      Bob   80000
2  Charlie  120000
3    David  100000
4      Eva   95000
```

In [17]:

```
# Step 5: Filtering rows based on a condition (e.g., Age > 30)
print("\nFiltering rows where Age > 30:")
filtered_df = df_from_dict[df_from_dict['Age'] > 30]
print(filtered_df)
```

```
Filtering rows where Age > 30:
      Name  Age     City  Salary
2  Charlie   35  Chicago  120000
3    David   40  Houston  100000
```

In [18]:

```
# Step 6: Adding a new column (e.g., 'Tax' calculated as 10% of Salary)
df_from_dict['Tax'] = df_from_dict['Salary'] * 0.1
print("\nDataFrame with a new 'Tax' column:")
print(df_from_dict)
```

```
DataFrame with a new 'Tax' column:
      Name  Age         City  Salary      Tax
0    Alice   23     New York   70000   7000.0
1      Bob   30  Los Angeles   80000   8000.0
2  Charlie   35      Chicago  120000  12000.0
3    David   40      Houston  100000  10000.0
4      Eva   28      Phoenix   95000   9500.0
```

In [19]:

```
# Step 7: Applying a function to a column (e.g., 'Salary' increase by 5%)
df_from_dict['Salary'] = df_from_dict['Salary'] * 1.05
print("\nDataFrame with 'Salary' increased by 5%:")
print(df_from_dict)
```

```
DataFrame with 'Salary' increased by 5%:
      Name  Age         City    Salary      Tax
0    Alice   23     New York   73500.0   7000.0
1      Bob   30  Los Angeles   84000.0   8000.0
2  Charlie   35      Chicago  126000.0  12000.0
3    David   40      Houston  105000.0  10000.0
4      Eva   28      Phoenix   99750.0   9500.0
```

In [20]:

```
# Step 8: Dropping a column (e.g., 'Tax')
df_from_dict = df_from_dict.drop('Tax', axis=1)
print("\nDataFrame after dropping the 'Tax' column:")
print(df_from_dict)
```

```
DataFrame after dropping the 'Tax' column:
      Name  Age         City    Salary
0    Alice   23     New York   73500.0
1      Bob   30  Los Angeles   84000.0
2  Charlie   35      Chicago  126000.0
3    David   40      Houston  105000.0
4      Eva   28      Phoenix   99750.0
```

In [21]:

```
# Step 9: Handling missing data (NaN values)
df_with_nan = pd.DataFrame({
    'Product': ['Apple', 'Banana', 'Cherry', 'Date'],
    'Price': [1.2, None, 2.5, None]
})
print("\nDataFrame with missing values:")
print(df_with_nan)
```

```
DataFrame with missing values:
  Product  Price
0   Apple    1.2
1  Banana    NaN
2  Cherry    2.5
3    Date    NaN
```

Conclusion :

Pandas DataFrame is an essential tool for data analysis and manipulation in Python. It offers robust functionalities for creating, accessing, manipulating, and analyzing structured data. With its integration with data visualization and statistical analysis libraries, DataFrame provides a complete environment for data-driven applications and research.