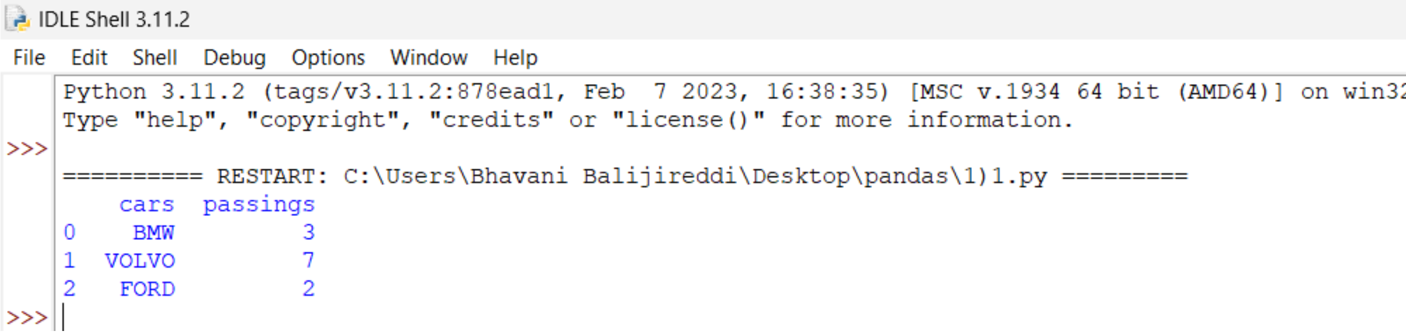
DATA EXPLORATION RECORD

April 29, 2023

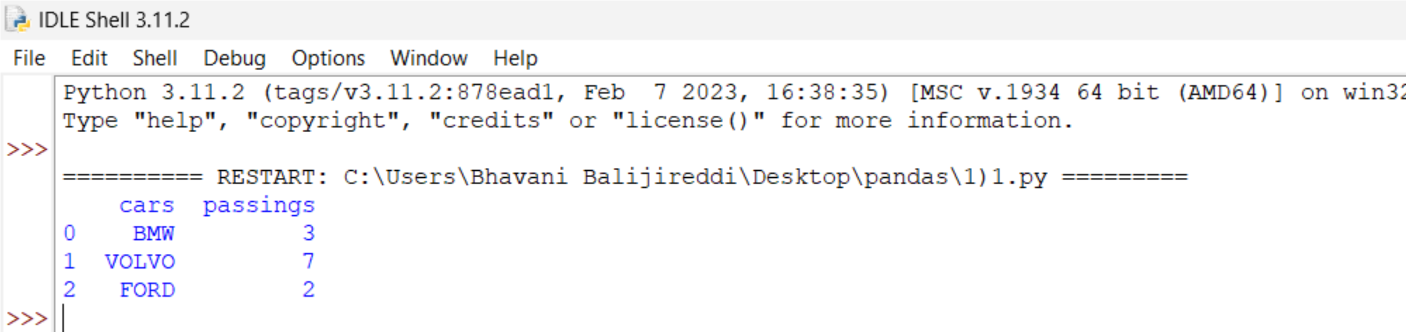
1.Introduction to the Data Exploration Components (Series and Data Frames) using Pandas in python
a.Import Pandas
b.Loading the data various formats (.XLS, .TXT, .CSV, JSON) using Pandas
c.Describe Data, Modify Data, Grouping Data, Filtering Data
d.Converting a variable to a different data type back to a CSV, JSON, or SQL

(a)Import Pandas
Aim:write a python program to import pandas
Description:
pandas is a popular open-source library for data manipulation and analysis in Python. It provides powerful and easy-to-use data structures, such as DataFrame and Series, that allow users to perform operations on tabular data with ease.
Program:
import pandas
mydataset={'cars':["BMW","VOLVO","FORD"], 'passings':[3,7,2]}
print(myvar)
Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win3
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\1)1.py =========
    cars  passings
0    BMW         3
1  VOLVO         7
2   FORD         2
>>>
```

Observed output:



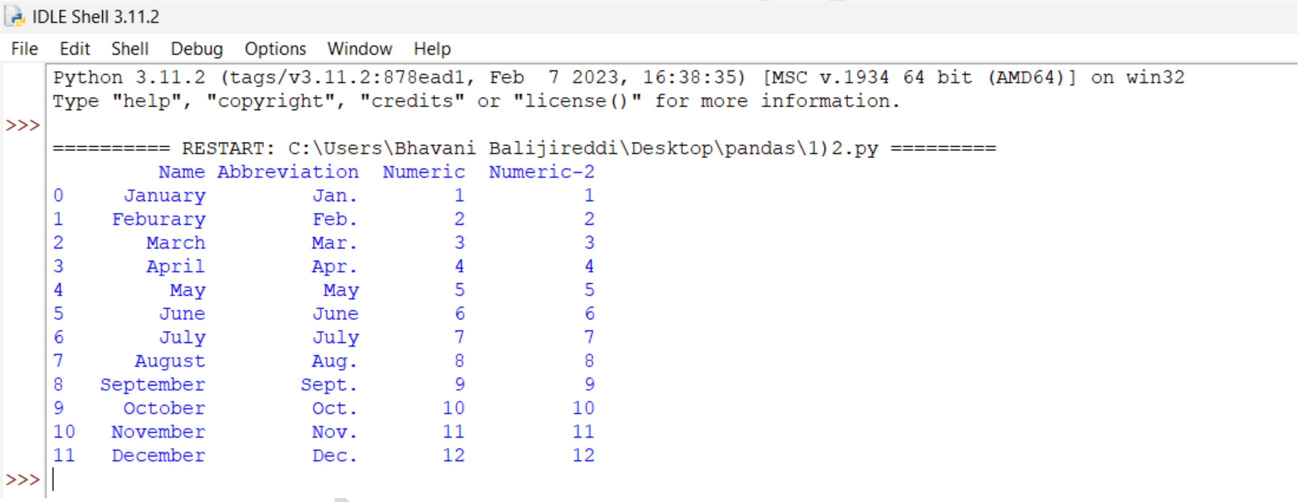
```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win3
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\1)1.py =========
    cars  passings
0    BMW         3
1  VOLVO         7
2   FORD         2
>>>
```

(b)Loading the data various formats (.XLS, .TXT, .CSV, JSON) using Pandas

Aim: Loading the data various formats (.XLS, .TXT, .CSV, JSON) using Pandas
Description:
1.For .XLS (Excel) files, you can use read_excel() function, specifying the file name and sheet name or sheet index to read data from.
2.For .TXT (text) files, you can use read_csv() function, specifying the delimiter used in the file (e.g., tab, comma, space), and other options such as encoding and skipping rows.
3.For .CSV (Comma-Separated Values) files, you can also use read_csv() function, specifying the delimiter used in the file (e.g., comma, semicolon), and other options such as encoding and skipping rows.
4.For JSON (JavaScript Object Notation) files, you can use read_json() function to load data into a Pandas DataFrame.
Program:

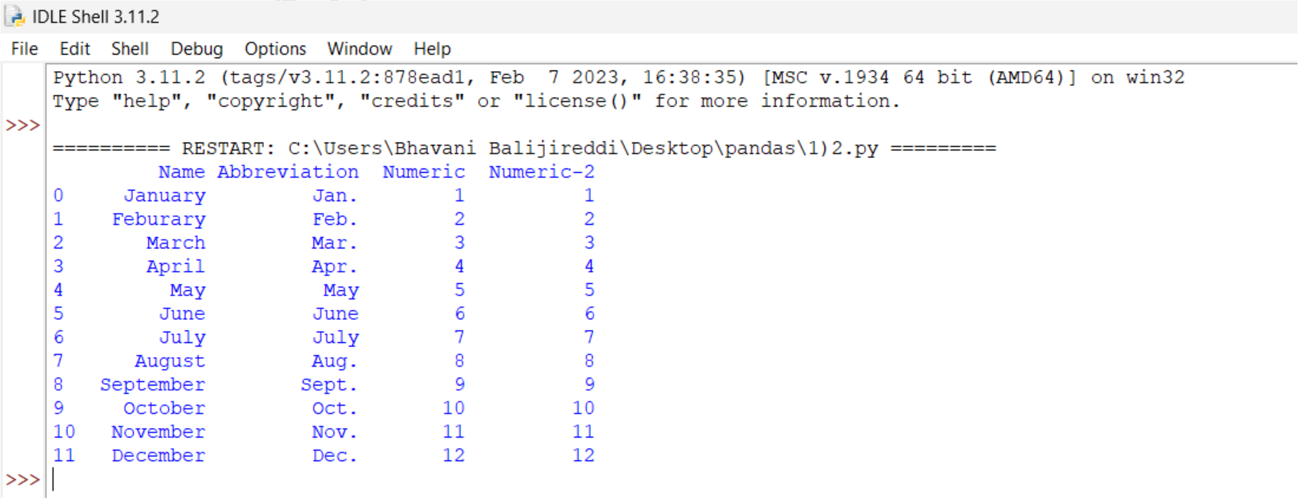
```
import pandas as pd
d=pd.read_csv("month.csv")
df=pd.DataFrame(d)
print(df)
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\1)2.py =========
         Name Abbreviation  Numeric  Numeric-2
0     January         Jan.        1          1
1    Feburary         Feb.        2          2
2       March         Mar.        3          3
3       April         Apr.        4          4
4         May          May        5          5
5        June         June        6          6
6        July         July        7          7
7      August         Aug.        8          8
8   September        Sept.        9          9
9     October         Oct.       10         10
10   November         Nov.       11         11
11   December         Dec.       12         12
>>> |
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\1)2.py =========
         Name Abbreviation  Numeric  Numeric-2
0     January         Jan.        1          1
1    Feburary         Feb.        2          2
2       March         Mar.        3          3
3       April         Apr.        4          4
4         May          May        5          5
5        June         June        6          6
6        July         July        7          7
7      August         Aug.        8          8
8   September        Sept.        9          9
9     October         Oct.       10         10
10   November         Nov.       11         11
11   December         Dec.       12         12
>>> |
```

(c)Describe Data, Modify Data, Grouping Data, Filtering Data

Aim: Describe Data, Modify Data, Grouping Data, Filtering Data

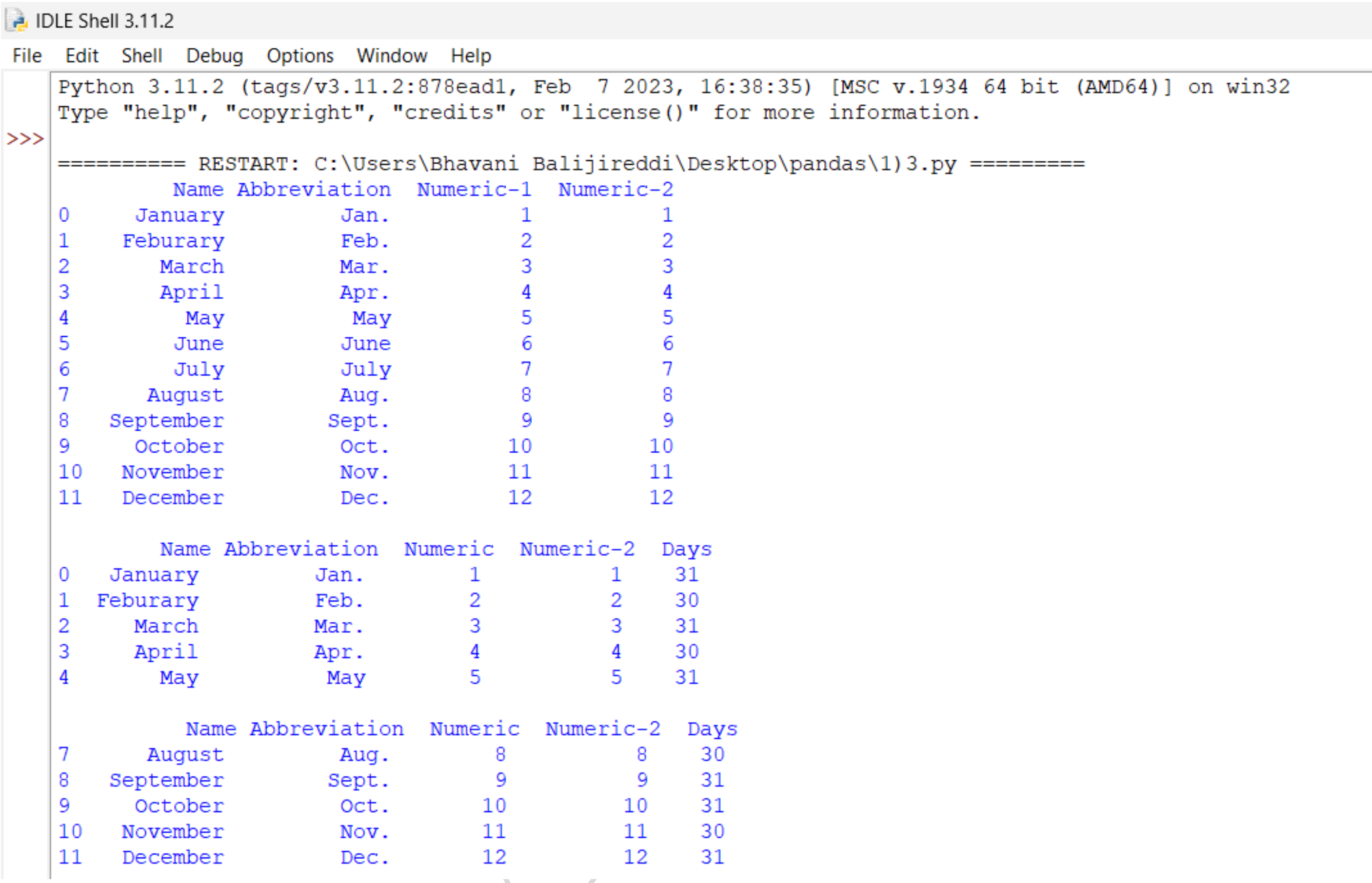
Description:

Modifying data refers to changing the values or structure of a dataset. This can include adding, removing, or updating data, as well as transforming data into a different format or type.
Grouping data involves organizing data into subsets based on common characteristics.
Filtering data refers to selecting a subset of data based on specific criteria.

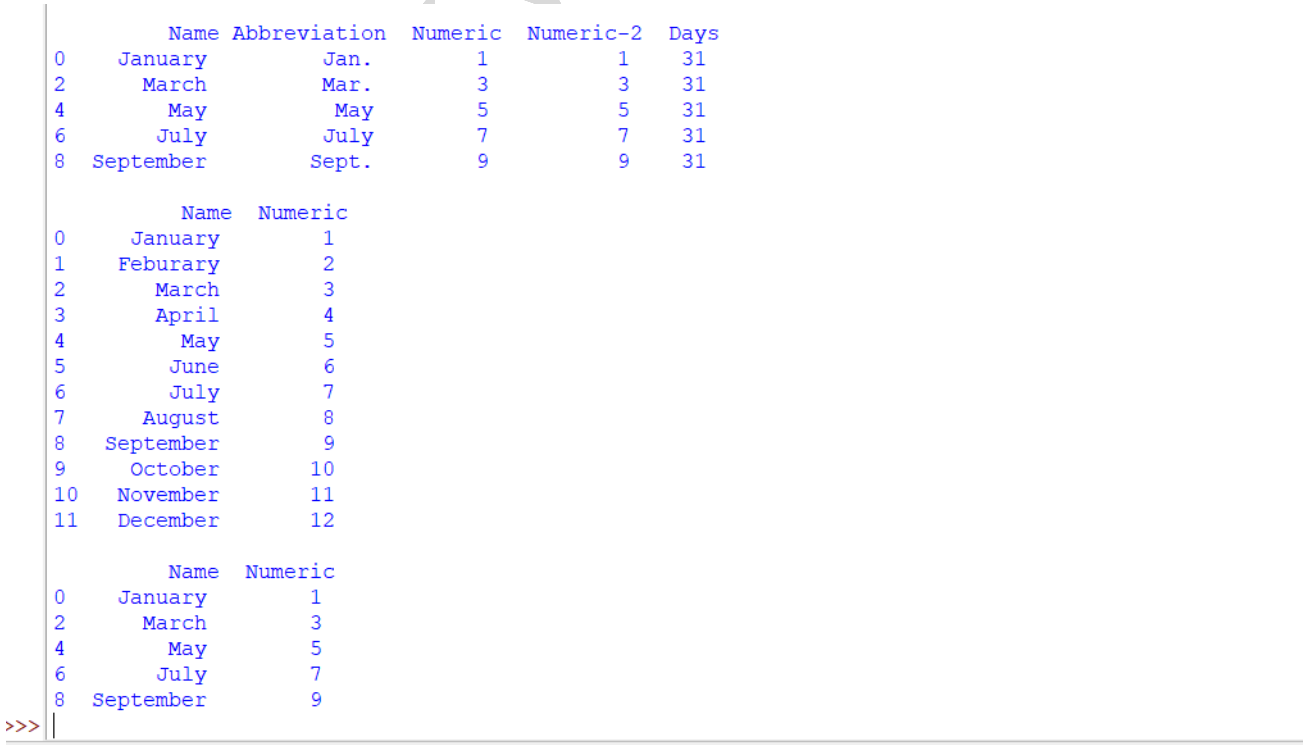
Program:

```
import pandas as pd
d=pd.read_csv(”month.csv”)
df=pd.DataFrame(d)
print(df.rename(columns={’Numeric’:’Numeric-1’}))
df[’Days’]=[31,30,31,30,31,30,31,30,31,31,30,31]
print( ’\n’,df.head())
print(’\n’,df.tail())
print(’\n’,df[0:10:2])
print(’\n’,df[[’Name’,’Numeric’]])
print(’\n’,df[[’Name’,’Numeric’]][0:10:2])
```

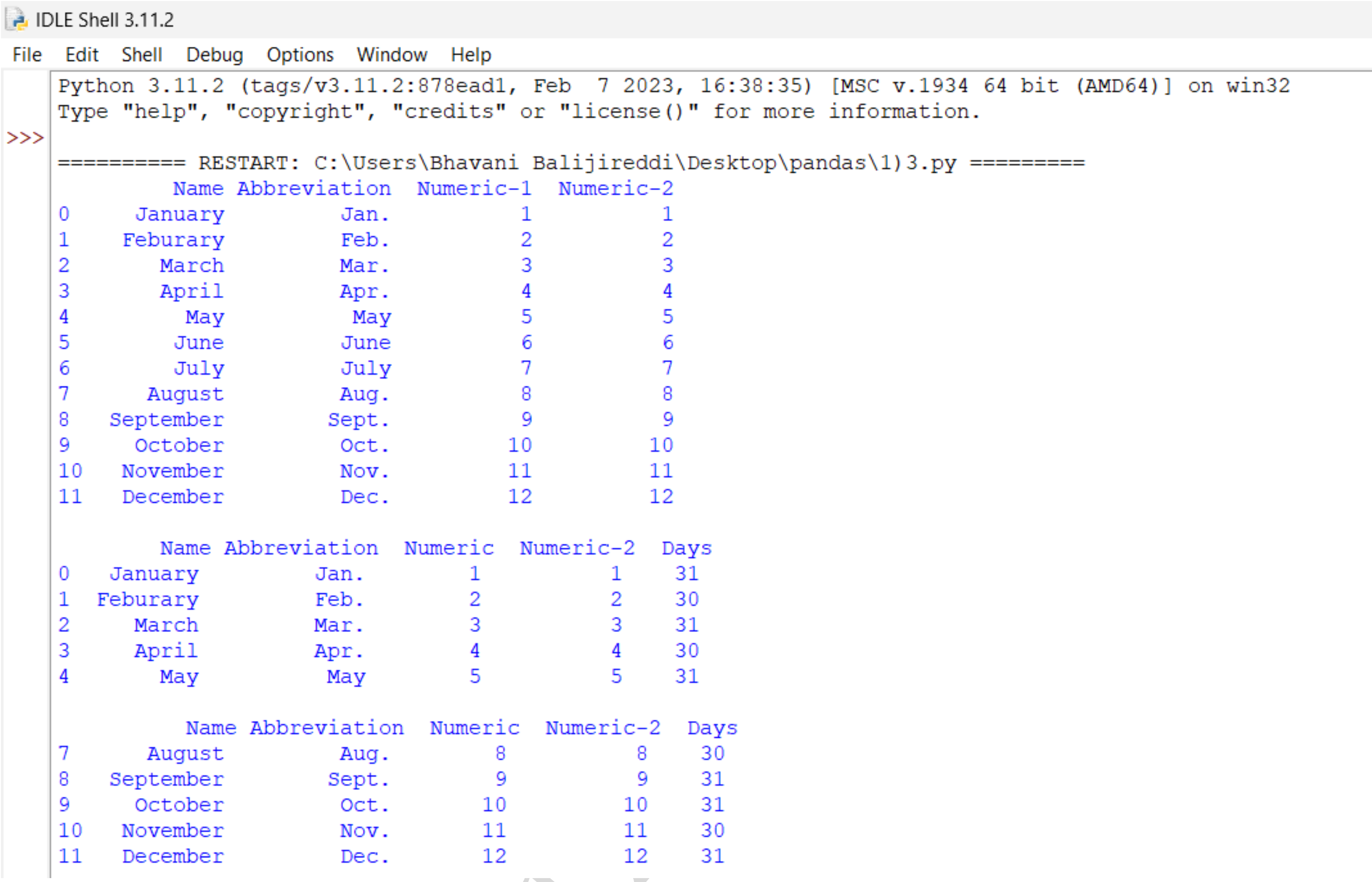
Expected output: observed output:



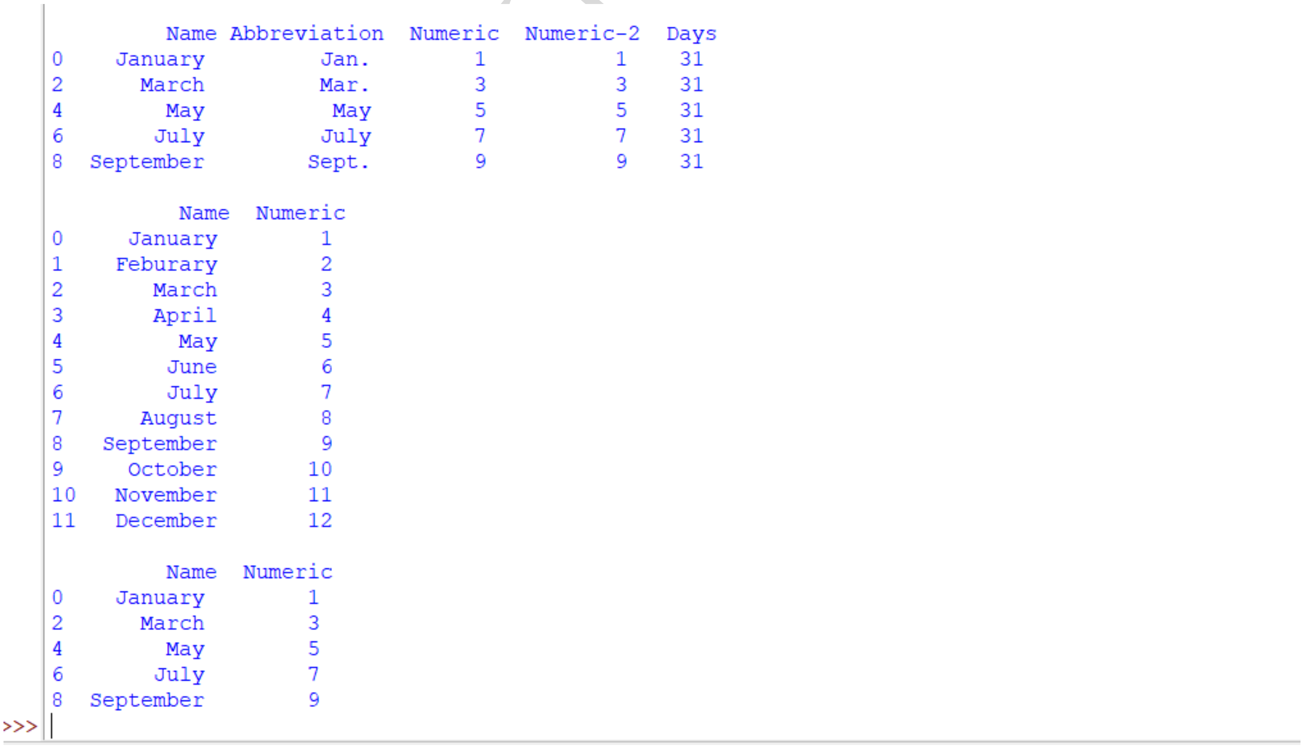
```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\1)3.py =========
         Name Abbreviation  Numeric-1  Numeric-2
0     January         Jan.          1          1
1    Feburary         Feb.          2          2
2       March         Mar.          3          3
3       April         Apr.          4          4
4         May          May          5          5
5        June         June          6          6
6        July         July          7          7
7      August         Aug.          8          8
8   September        Sept.          9          9
9     October         Oct.         10         10
10   November         Nov.         11         11
11   December         Dec.         12         12

       Name Abbreviation  Numeric  Numeric-2  Days
0   January         Jan.        1          1    31
1  Feburary         Feb.        2          2    30
2     March         Mar.        3          3    31
3     April         Apr.        4          4    30
4       May          May        5          5    31

         Name Abbreviation  Numeric  Numeric-2  Days
7      August         Aug.        8          8    30
8   September        Sept.        9          9    31
9     October         Oct.       10         10    31
10   November         Nov.       11         11    30
11   December         Dec.       12         12    31
```



```
        Name Abbreviation  Numeric  Numeric-2  Days
0    January         Jan.        1          1    31
2      March         Mar.        3          3    31
4        May          May        5          5    31
6       July         July        7          7    31
8  September        Sept.        9          9    31

         Name  Numeric
0     January        1
1    Feburary        2
2       March        3
3       April        4
4         May        5
5        June        6
6        July        7
7      August        8
8   September        9
9     October       10
10   November       11
11   December       12

        Name  Numeric
0    January        1
2      March        3
4        May        5
6       July        7
8  September        9
>>> 
```



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\1)3.py =========
         Name Abbreviation  Numeric-1  Numeric-2
0     January         Jan.          1          1
1    Feburary         Feb.          2          2
2       March         Mar.          3          3
3       April         Apr.          4          4
4         May          May          5          5
5        June         June          6          6
6        July         July          7          7
7      August         Aug.          8          8
8   September        Sept.          9          9
9     October         Oct.         10         10
10   November         Nov.         11         11
11   December         Dec.         12         12

        Name Abbreviation  Numeric  Numeric-2  Days
0   January         Jan.        1          1    31
1  Feburary         Feb.        2          2    30
2     March         Mar.        3          3    31
3     April         Apr.        4          4    30
4       May          May        5          5    31

          Name Abbreviation  Numeric  Numeric-2  Days
7      August         Aug.        8          8    30
8   September        Sept.        9          9    31
9     October         Oct.       10         10    31
10   November         Nov.       11         11    30
11   December         Dec.       12         12    31
```



```
         Name Abbreviation  Numeric  Numeric-2  Days
0    January         Jan.        1          1    31
2      March         Mar.        3          3    31
4        May          May        5          5    31
6       July         July        7          7    31
8  September        Sept.        9          9    31

          Name  Numeric
0     January        1
1    Feburary        2
2       March        3
3       April        4
4         May        5
5        June        6
6        July        7
7      August        8
8   September        9
9     October       10
10   November       11
11   December       12

         Name  Numeric
0    January        1
2      March        3
4        May        5
6       July        7
8  September        9
>>> 
```

(d)Converting a variable to a different data type back to a CSV, JSON, or SQL

Aim: Converting a variable to a different data type back to a CSV, JSON, or SQL

Description:

Converting a variable to a different data type and saving to CSV: You can use the pd.to_csv() function in Pandas to convert a variable (such as a DataFrame) to a CSV format. You specify the file name and other options such as the delimiter, encoding, and whether to include or exclude index in the CSV file. Once converted, the data can be saved to a CSV file using the to_csv() function.

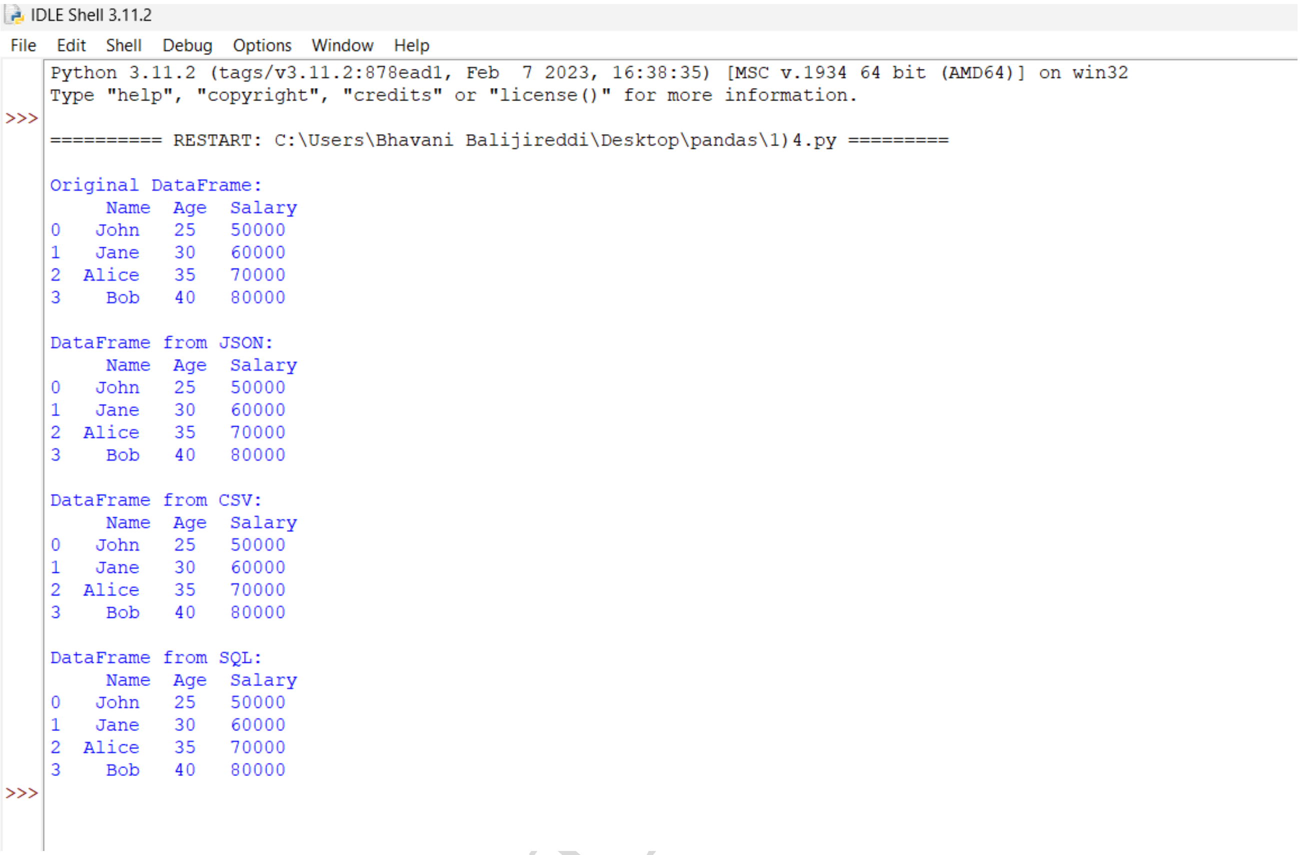
Converting a variable to a different data type and saving to JSON: You can use the pd.to_json() function in Pandas to convert a variable (such as a DataFrame) to a JSON format. You specify the file name and other options such as the JSON structure (orient), compression, and indentation level. Once converted, the data can be saved to a JSON file using the to_json() function.

Converting a variable to a different data type and saving to SQL: You can use the pd.to_sql() function in Pandas to convert a variable (such as a DataFrame) to a SQL format and save it to a SQL database. You need to specify the database connection details, table name, and other options such as if_exists (what to do if the table already exists), and whether to include or exclude index in the SQL table. Once converted, the data can be saved to a SQL table using the to_sql() function.

Program:

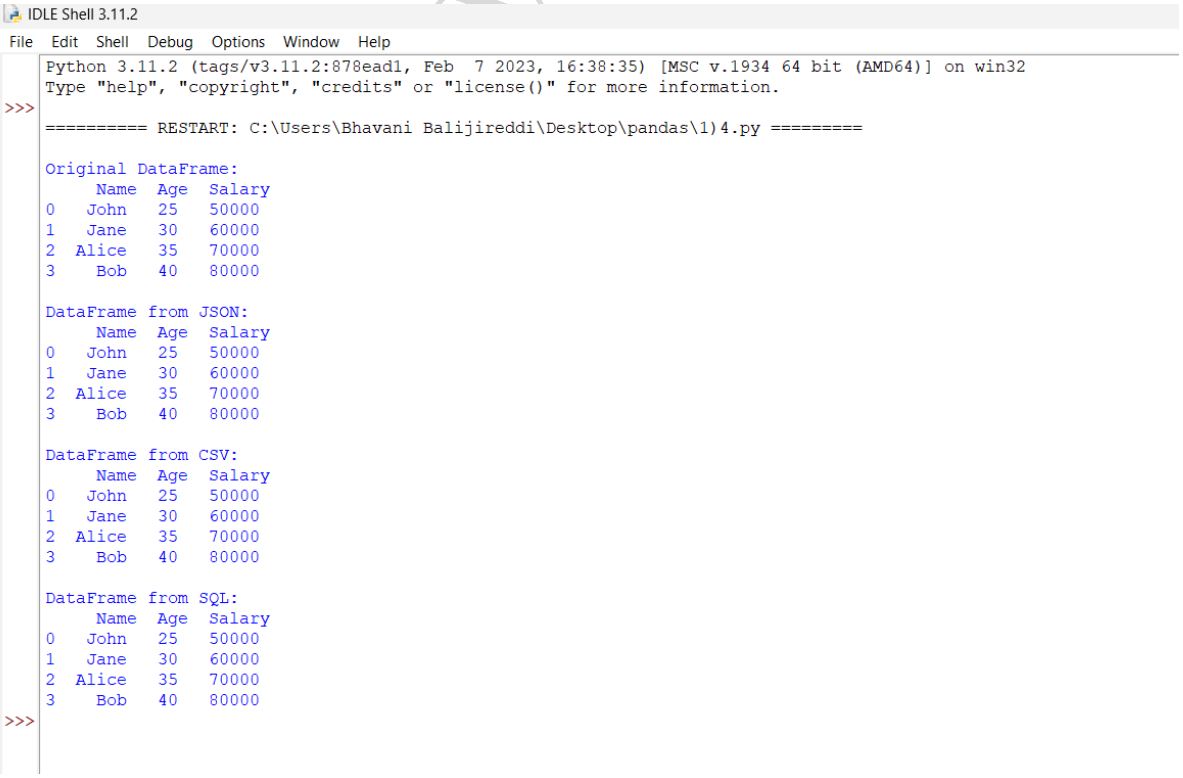
```
import pandas as pd
import io
import sqlite3
sample_data = {
’Name’: [’John’, ’Jane’, ’Alice’, ’Bob’],
’Age’: [25, 30, 35, 40],
’Salary’: [50000, 60000, 70000, 80000]
}
df = pd.DataFrame(sample_data)
json_data = df.to_json()
df_from_json = pd.read_json(json_data)
csv_data = df.to_csv(index=False)
df_from_csv = pd.read_csv(io.StringIO(csv_data))
conn = sqlite3.connect(’example.db’)
df.to_sql(’employee’, conn, if_exists=’replace’, index=False)
df_from_sql = pd.read_sql(’SELECT * FROM employee’, conn)
print(’\n Original DataFrame:\n’, df)
print(’\n DataFrame from JSON:\n’, df_from_json)
print(’\n DataFrame from CSV:\n’, df_from_csv)
print(’\n DataFrame from SQL:\n’, df_from_sql)
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\1)4.py =========

Original DataFrame:
    Name  Age  Salary
0   John   25   50000
1   Jane   30   60000
2  Alice   35   70000
3    Bob   40   80000

DataFrame from JSON:
    Name  Age  Salary
0   John   25   50000
1   Jane   30   60000
2  Alice   35   70000
3    Bob   40   80000

DataFrame from CSV:
    Name  Age  Salary
0   John   25   50000
1   Jane   30   60000
2  Alice   35   70000
3    Bob   40   80000

DataFrame from SQL:
    Name  Age  Salary
0   John   25   50000
1   Jane   30   60000
2  Alice   35   70000
3    Bob   40   80000
>>> 
```

observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\1)4.py =========

Original DataFrame:
    Name  Age  Salary
0   John   25   50000
1   Jane   30   60000
2  Alice   35   70000
3    Bob   40   80000

DataFrame from JSON:
    Name  Age  Salary
0   John   25   50000
1   Jane   30   60000
2  Alice   35   70000
3    Bob   40   80000

DataFrame from CSV:
    Name  Age  Salary
0   John   25   50000
1   Jane   30   60000
2  Alice   35   70000
3    Bob   40   80000

DataFrame from SQL:
    Name  Age  Salary
0   John   25   50000
1   Jane   30   60000
2  Alice   35   70000
3    Bob   40   80000
>>> 
```

2.Reading and writing files
a. Reading a CSV File
b. Writing content of data frames to CSV File
c. Reading an Excel File
d. Writing content of data frames to Excel File

(a)Reading a CSV File

Aim: Reading a CSV File using pandas

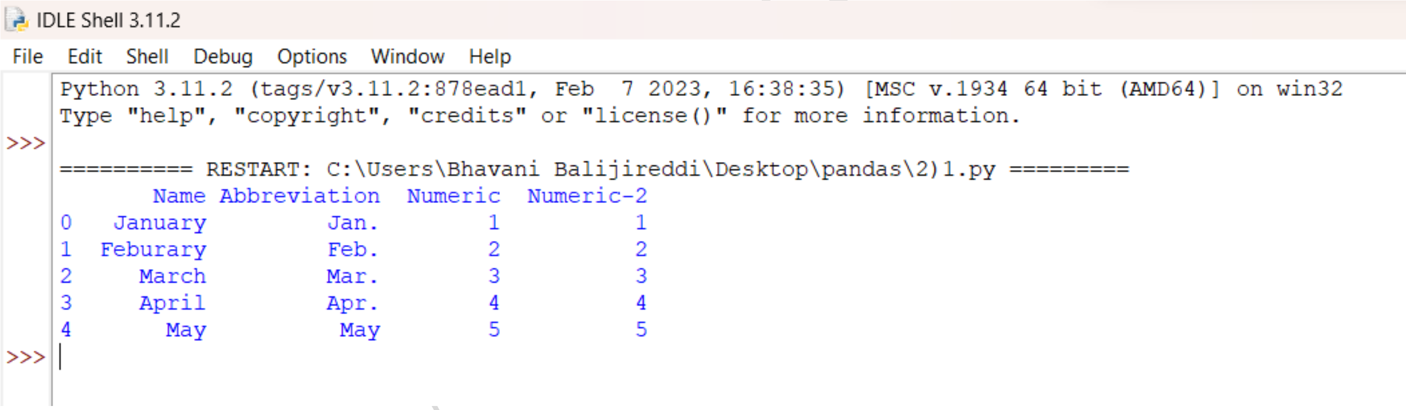
Description:

To read a CSV file, you can use a programming language such as Python or Java, and use libraries like pandas or csv to parse the contents of the file. Once the CSV file has been read, you can perform various operations on the data, such as filtering, sorting, and grouping, or you can save the data to a database or another file format.

Program:

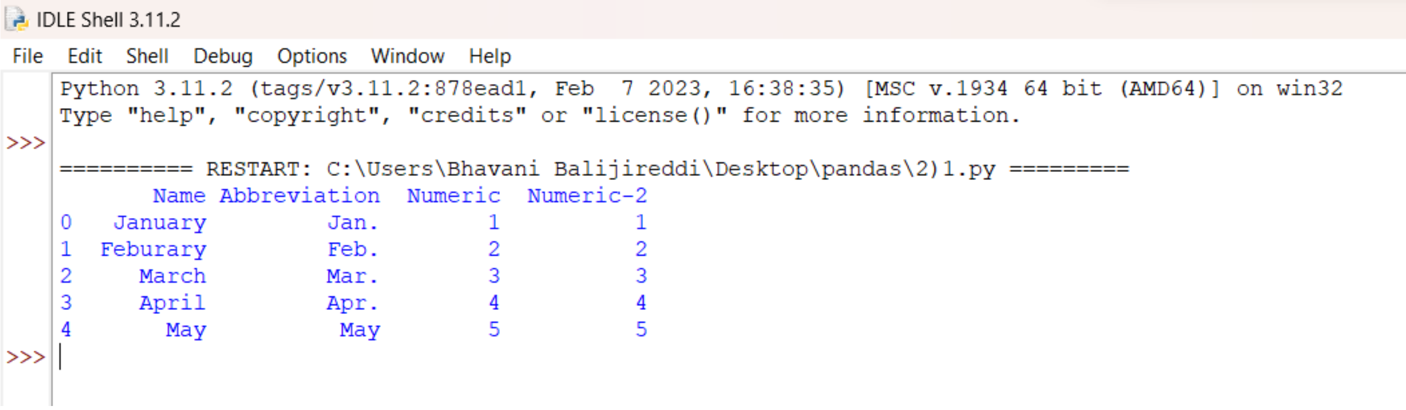
```
import pandas as pd
df = pd.read_csv(”month.csv”)
print(df.head())
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\2)1.py =========
       Name Abbreviation  Numeric  Numeric-2
0   January         Jan.        1          1
1  Feburary         Feb.        2          2
2     March         Mar.        3          3
3     April         Apr.        4          4
4       May          May        5          5
>>> 
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\2)1.py =========
       Name Abbreviation  Numeric  Numeric-2
0   January         Jan.        1          1
1  Feburary         Feb.        2          2
2     March         Mar.        3          3
3     April         Apr.        4          4
4       May          May        5          5
>>> 
```

(b)Writing content of data frames to CSV File

Aim: Writing content of data frames to CSV File

Description:

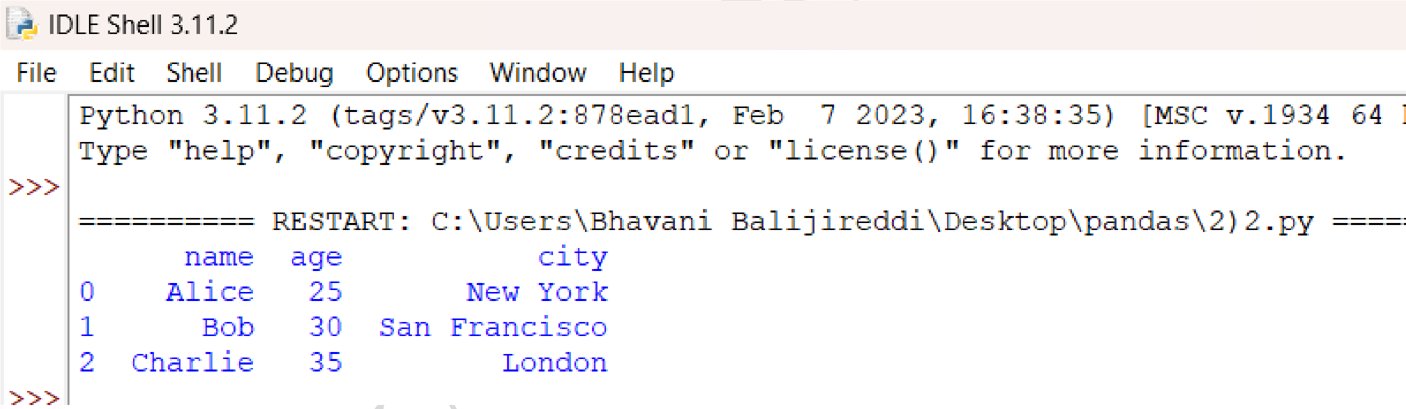
Writing the contents of a data frame to a CSV file involves converting the data frame into a CSV format and saving it to a file. This is a common operation when working with data in Python, as CSV files are a widely used format for storing and sharing data.

To write the contents of a data frame to a CSV file in Python, you can use the to_csv() method of the data frame object. This method takes a file path as its argument, and writes the contents of the data frame to a file at that path in CSV format.

Program:

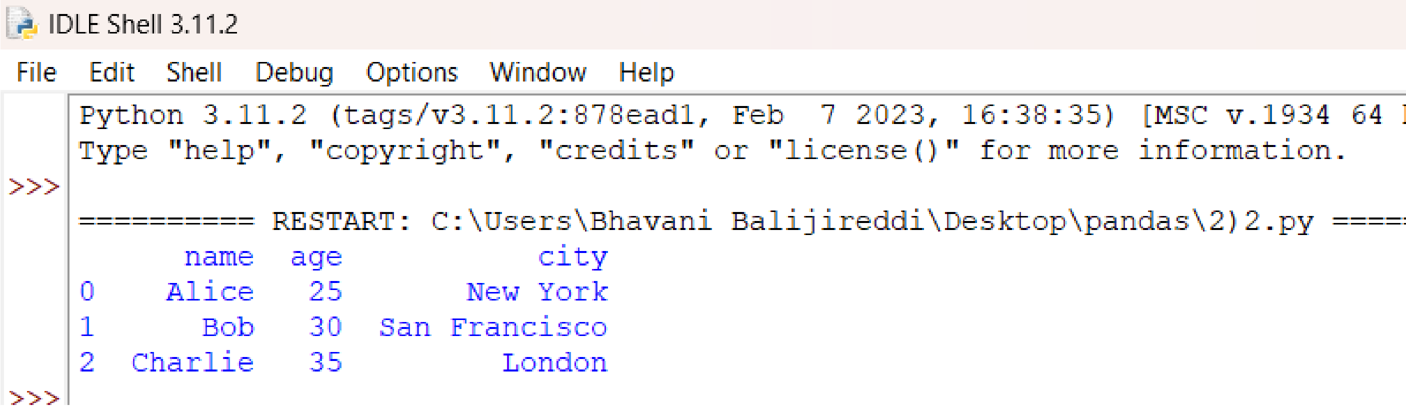
```
import pandas as pd
data = {'name': ['Alice', 'Bob', 'Charlie'],
'age': [25, 30, 35],
'city': ['New York', 'San Francisco', 'London'] }
df = pd.DataFrame(data)
df.to_csv('my_data.csv', index=False)
print(df)
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\2)2.py ====
      name  age           city
0    Alice   25       New York
1      Bob   30  San Francisco
2  Charlie   35         London
>>>
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\2)2.py ====
      name  age           city
0    Alice   25       New York
1      Bob   30  San Francisco
2  Charlie   35         London
>>>
```

(c) Reading an Excel File

Aim: Reading an Excel File

Description:

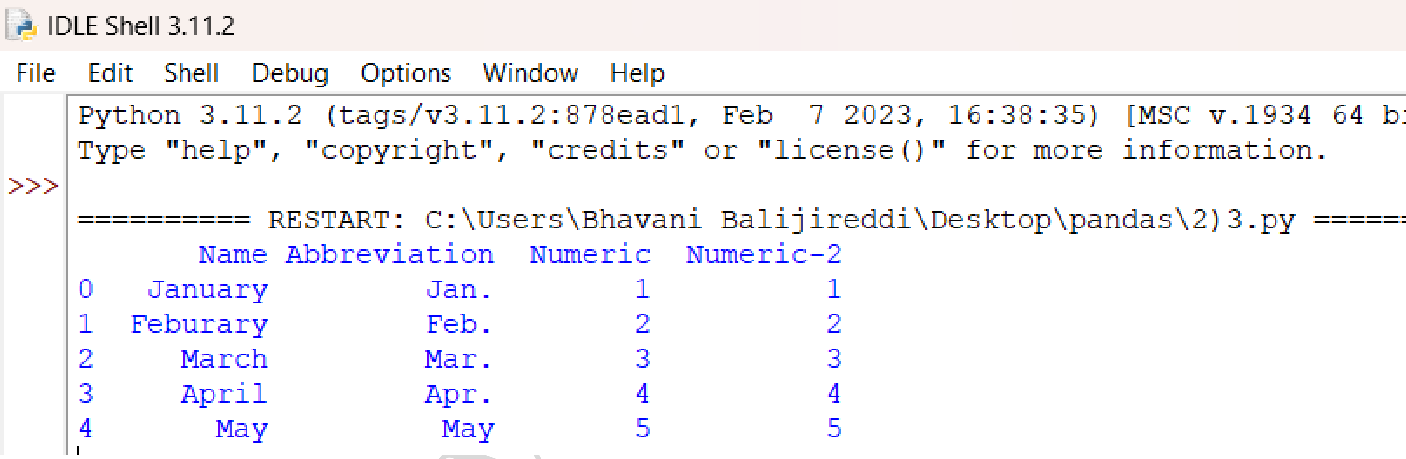
Reading an Excel file typically involves using a software library or module that can handle the file format, such as pandas in Python or Apache POI in Java.

The first step is to open the Excel file using the appropriate function or method provided by the library, specifying the file path and any additional options. Once the file is open, the data can be accessed using various functions or methods, such as selecting specific rows and columns or filtering based on certain conditions.

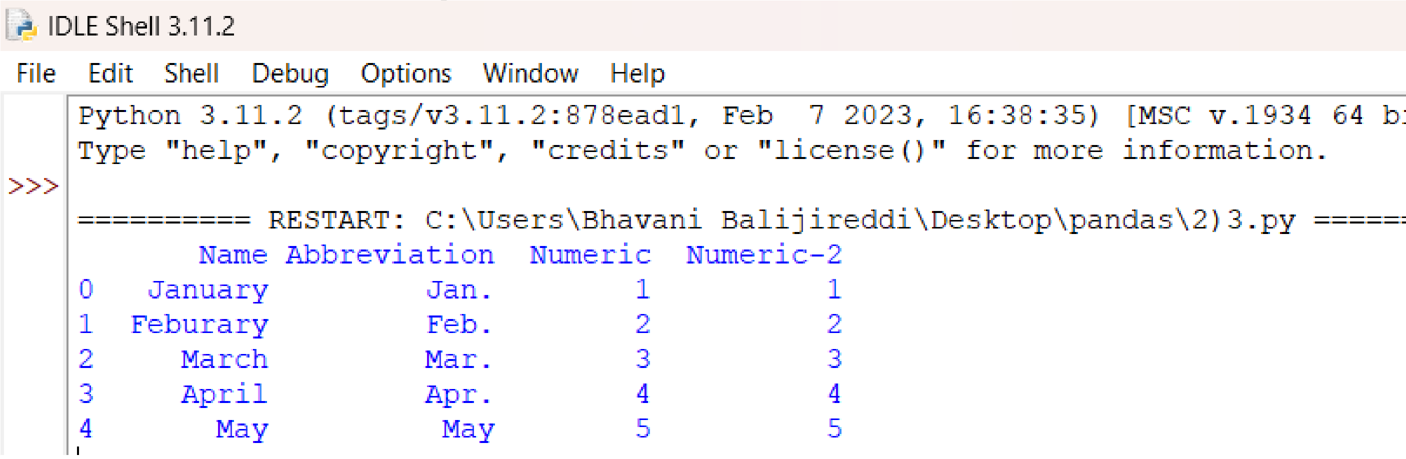
Program:

```
import pandas as pd
df = pd.read_excel('month.xlsx')
print(df.head())
```

Expected output: Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bi
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\2)3.py =====
       Name Abbreviation  Numeric  Numeric-2
0   January         Jan.        1          1
1  Feburary         Feb.        2          2
2     March         Mar.        3          3
3     April         Apr.        4          4
4       May          May        5          5
```

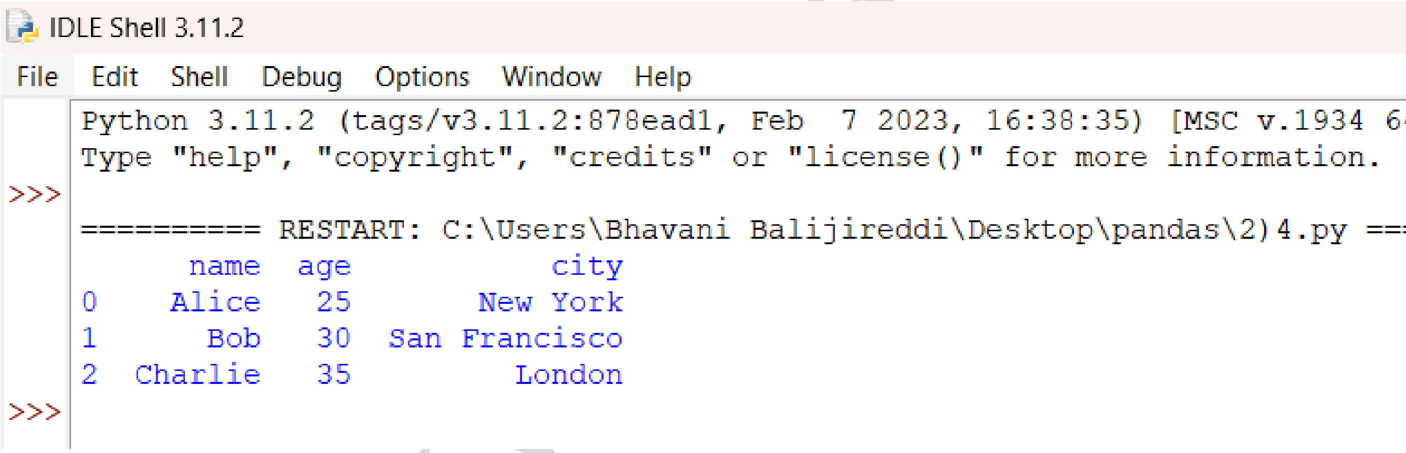


```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bi
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\2)3.py =====
       Name Abbreviation  Numeric  Numeric-2
0   January         Jan.        1          1
1  Feburary         Feb.        2          2
2     March         Mar.        3          3
3     April         Apr.        4          4
4       May          May        5          5
```

(d)Writing content of data frames to Excel File
Aim: Writing content of data frames to Excel File **Description:**
Writing the content of a data frame to an Excel file typically involves using a software library or module that can handle the file format, such as pandas in Python or Apache POI in Java.
The first step is to create an instance of the library’s Excel writer class, specifying the file path and any additional options such as the sheet name and data format. Then, the data frame can be written to the Excel file using the writer’s appropriate method or function, such as ’to_excel() ’in pandas.
Program:

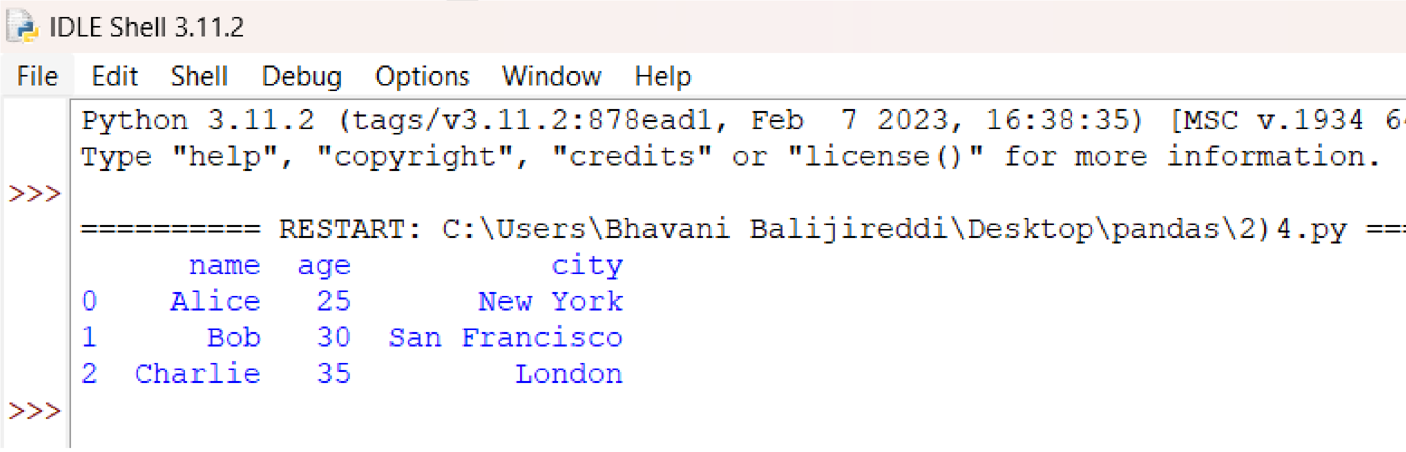
```
import pandas as pd
data = {’name’: [’Alice’, ’Bob’, ’Charlie’],
’age’: [25, 30, 35],
’city’: [’New York’, ’San Francisco’, ’London’]}
df = pd.DataFrame(data)
df.to_excel(’data.xlsx’, index=False)
print(df)
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 6
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\2)4.py ==
      name  age           city
0    Alice   25       New York
1      Bob   30  San Francisco
2  Charlie   35         London
>>>
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 6
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\2)4.py ==
      name  age           city
0    Alice   25       New York
1      Bob   30  San Francisco
2  Charlie   35         London
>>>
```

3.Getting the Dataset
a. Viewing your data
b. Data Set Description
c. Describe as category
d. Handling duplicates
e. Number of observations Per Category
f. Column cleanup

(a)Viewing your data

Aim: Viewing your data

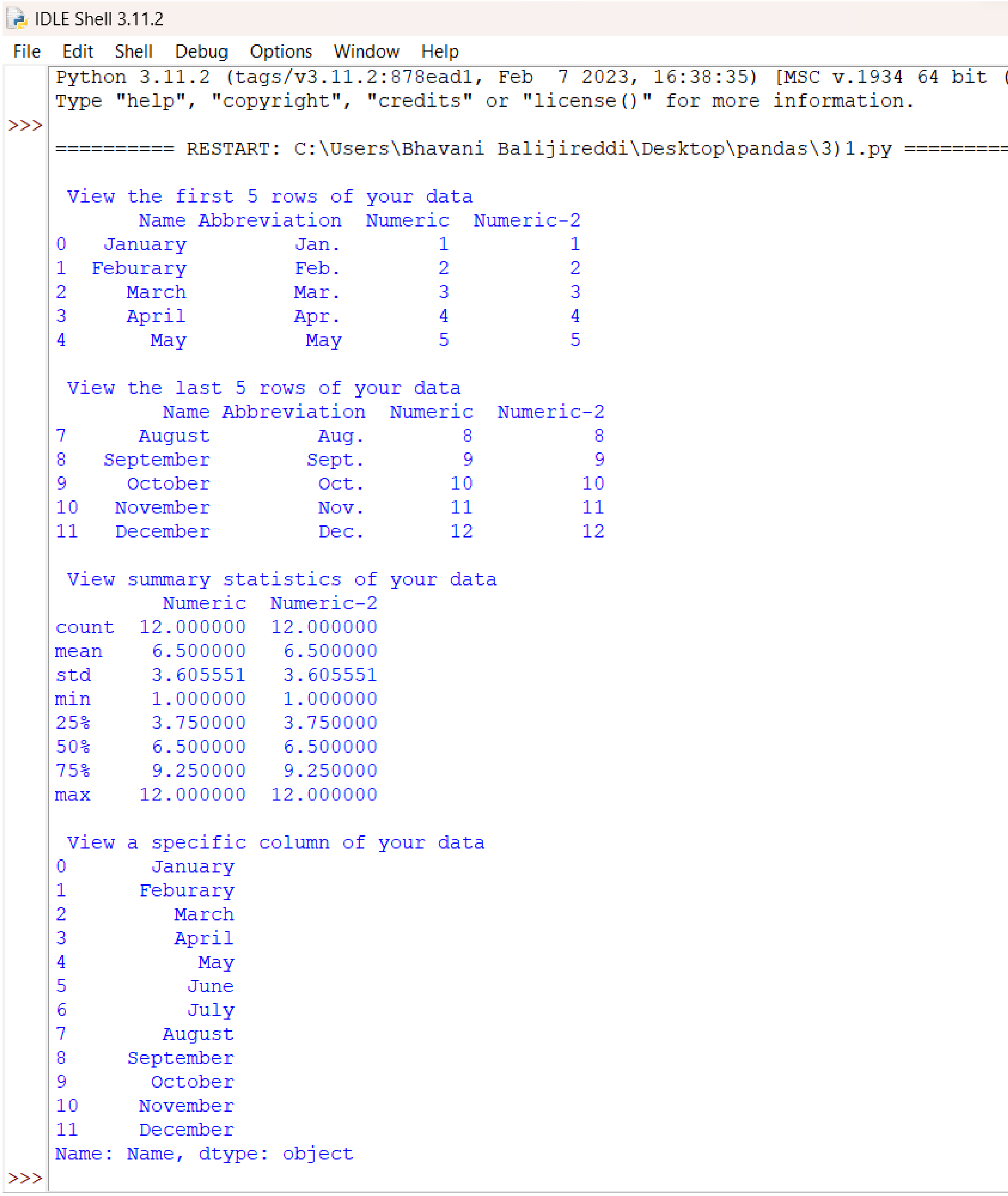
Description:

In most programming languages, including Python and R, you can use functions or methods provided by software libraries or modules to view your data. For example, in Python, you can use the ’head() ’function in pandas to view the first few rows of a data frame or ’describe() ’function to get a statistical summary of the data. Similarly, in R, you can use the ’head()’ function.

Program:

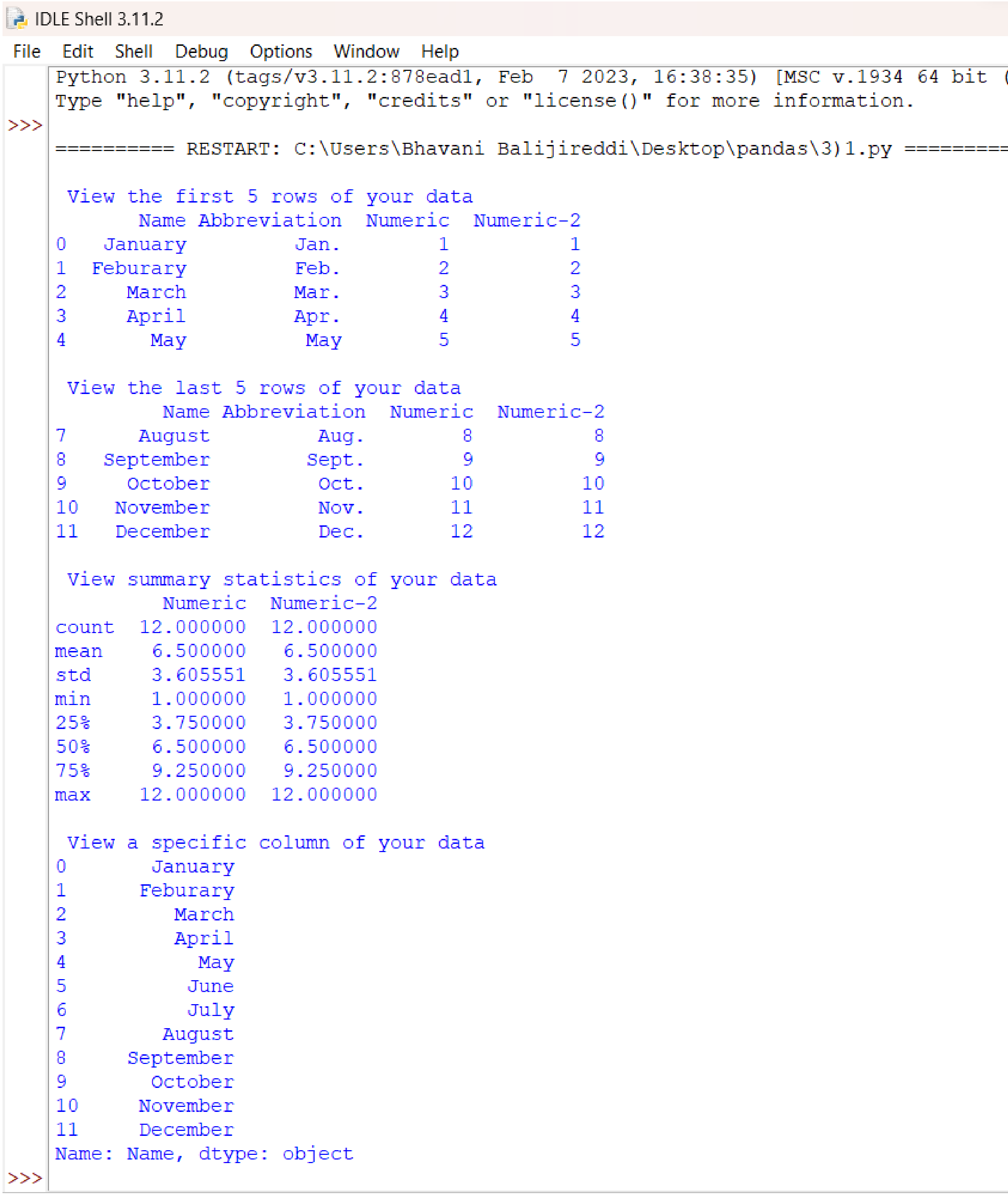
```
import pandas as pd
data = pd.read_csv(’month.csv’)
print(’\n’, ”View the first 5 rows of your data”)
print(data.head())
print(’\n’,”View the last 5 rows of your data”)
print(data.tail())
print(’\n’,”View summary statistics of your data”)
print(data.describe())
print(’\n’,”View a specific column of your data”)
print(data[’Name’])
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\3)1.py =========

 View the first 5 rows of your data
       Name Abbreviation  Numeric  Numeric-2
0   January         Jan.        1          1
1  Feburary         Feb.        2          2
2     March         Mar.        3          3
3     April         Apr.        4          4
4       May          May        5          5

 View the last 5 rows of your data
         Name Abbreviation  Numeric  Numeric-2
7      August         Aug.        8          8
8   September        Sept.        9          9
9     October         Oct.       10         10
10    November         Nov.       11         11
11    December         Dec.       12         12

 View summary statistics of your data
         Numeric  Numeric-2
count  12.000000  12.000000
mean    6.500000   6.500000
std     3.605551   3.605551
min     1.000000   1.000000
25%     3.750000   3.750000
50%     6.500000   6.500000
75%     9.250000   9.250000
max    12.000000  12.000000

 View a specific column of your data
0       January
1      Feburary
2         March
3         April
4           May
5          June
6          July
7        August
8     September
9       October
10     November
11     December
Name: Name, dtype: object
>>>
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\3)1.py =========

 View the first 5 rows of your data
       Name Abbreviation  Numeric  Numeric-2
0   January         Jan.        1          1
1  Feburary         Feb.        2          2
2     March         Mar.        3          3
3     April         Apr.        4          4
4       May          May        5          5

 View the last 5 rows of your data
         Name Abbreviation  Numeric  Numeric-2
7      August         Aug.        8          8
8   September        Sept.        9          9
9     October         Oct.       10         10
10   November         Nov.       11         11
11   December         Dec.       12         12

 View summary statistics of your data
         Numeric  Numeric-2
count  12.000000  12.000000
mean    6.500000   6.500000
std     3.605551   3.605551
min     1.000000   1.000000
25%     3.750000   3.750000
50%     6.500000   6.500000
75%     9.250000   9.250000
max    12.000000  12.000000

 View a specific column of your data
0       January
1      Feburary
2         March
3         April
4           May
5          June
6          July
7        August
8     September
9       October
10     November
11     December
Name: Name, dtype: object
>>>
```

(b)Data Set Description

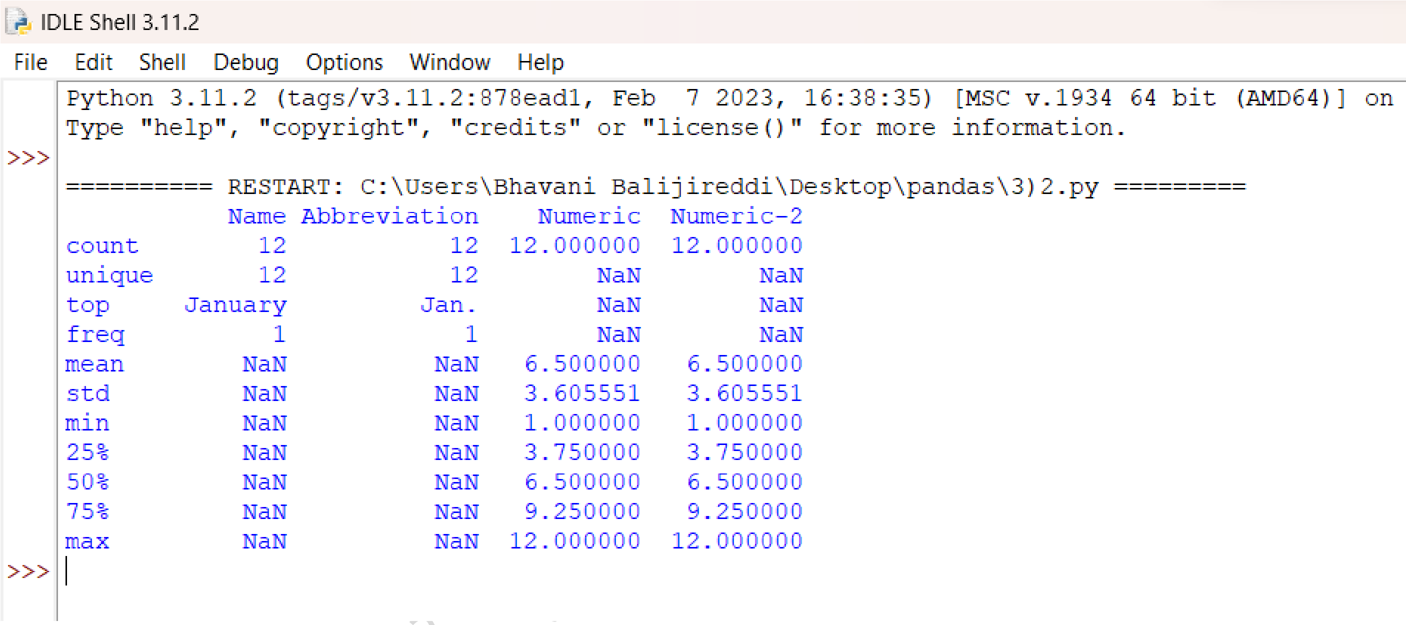
Aim: Data Set Description

Description:

The description typically includes information about the data collection process, such as the methodology, sampling techniques, and any limitations or biases in the data. It may also provide a summary of the variables included in the data set, such as their names, types, and units of measurement. **Program:**

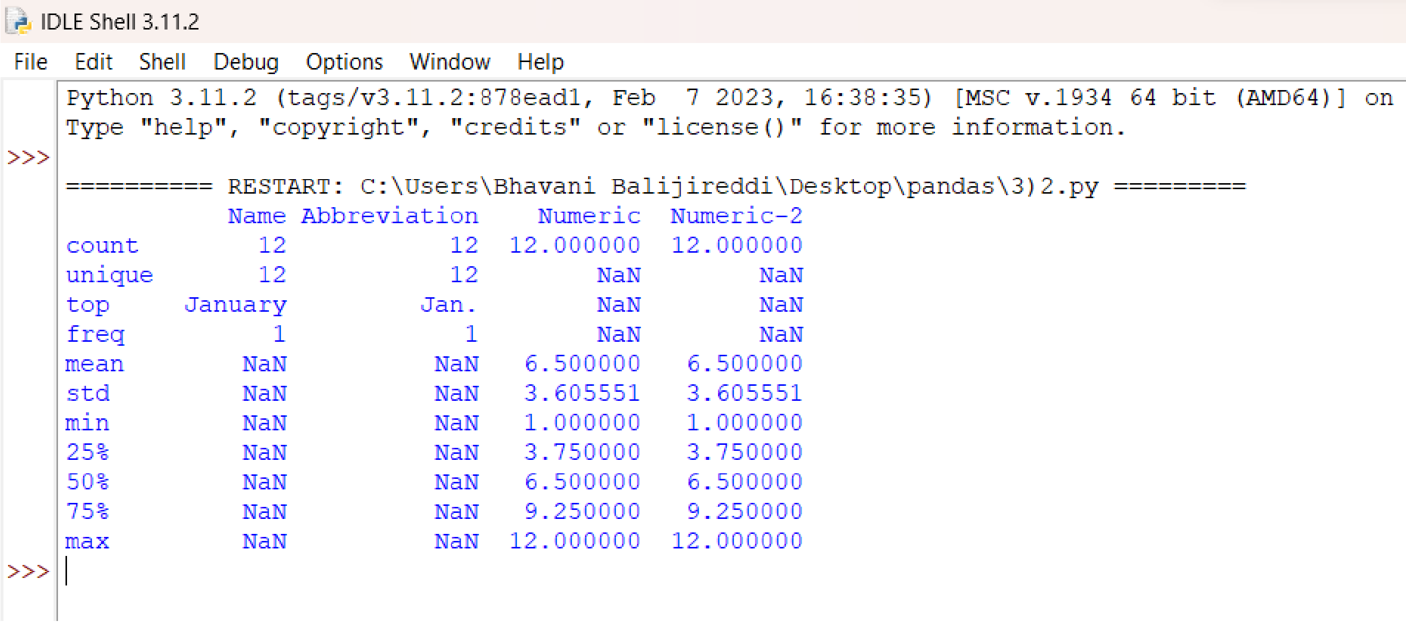
import pandas as pd
data = pd.read_csv('month.csv')
print(data.describe(include='all'))

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\3)2.py =========
           Name Abbreviation    Numeric  Numeric-2
count        12           12  12.000000  12.000000
unique       12           12        NaN        NaN
top     January         Jan.        NaN        NaN
freq          1            1        NaN        NaN
mean        NaN          NaN   6.500000   6.500000
std         NaN          NaN   3.605551   3.605551
min         NaN          NaN   1.000000   1.000000
25%         NaN          NaN   3.750000   3.750000
50%         NaN          NaN   6.500000   6.500000
75%         NaN          NaN   9.250000   9.250000
max         NaN          NaN  12.000000  12.000000
>>> 
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\3)2.py =========
           Name Abbreviation    Numeric  Numeric-2
count        12           12  12.000000  12.000000
unique       12           12        NaN        NaN
top     January         Jan.        NaN        NaN
freq          1            1        NaN        NaN
mean        NaN          NaN   6.500000   6.500000
std         NaN          NaN   3.605551   3.605551
min         NaN          NaN   1.000000   1.000000
25%         NaN          NaN   3.750000   3.750000
50%         NaN          NaN   6.500000   6.500000
75%         NaN          NaN   9.250000   9.250000
max         NaN          NaN  12.000000  12.000000
>>> 
```

(c)Describe as category

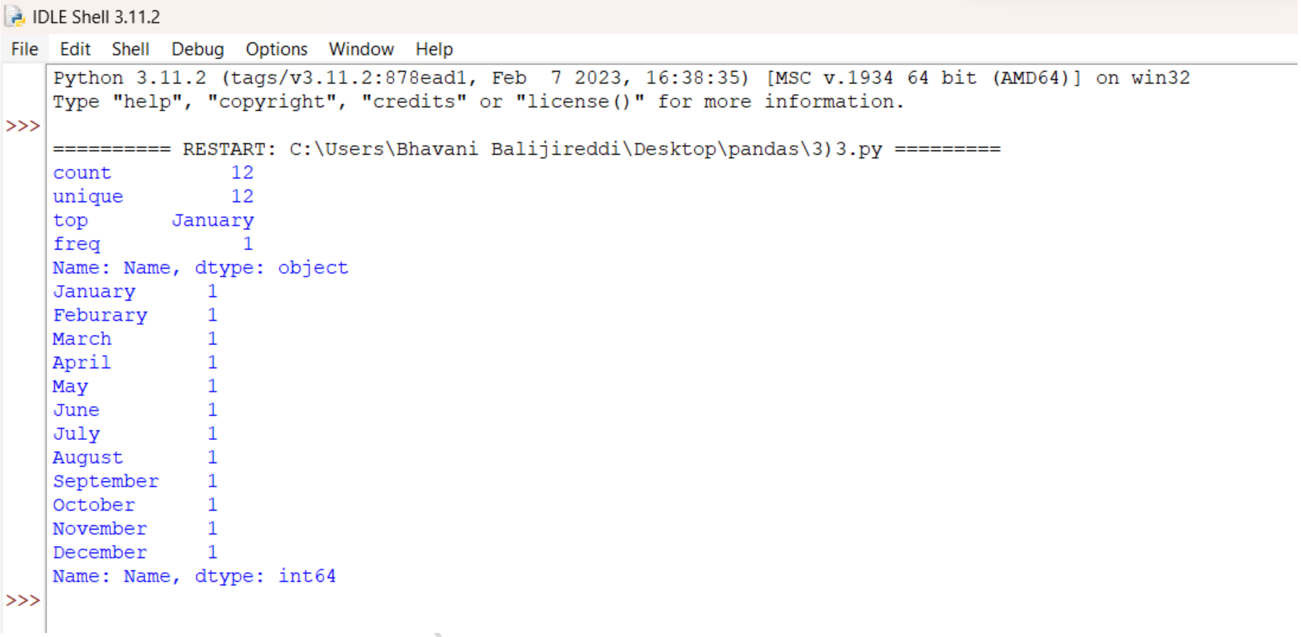
Aim: Describe as category

Description:

A category is a classification or grouping of things based on shared characteristics, traits, or attributes. It is a way to organize and simplify information by creating distinct classes or sets of items that share common features or properties. Categories can be hierarchical, with subcategories and supercategories, or they can be flat, with no hierarchical structure. **Program:**

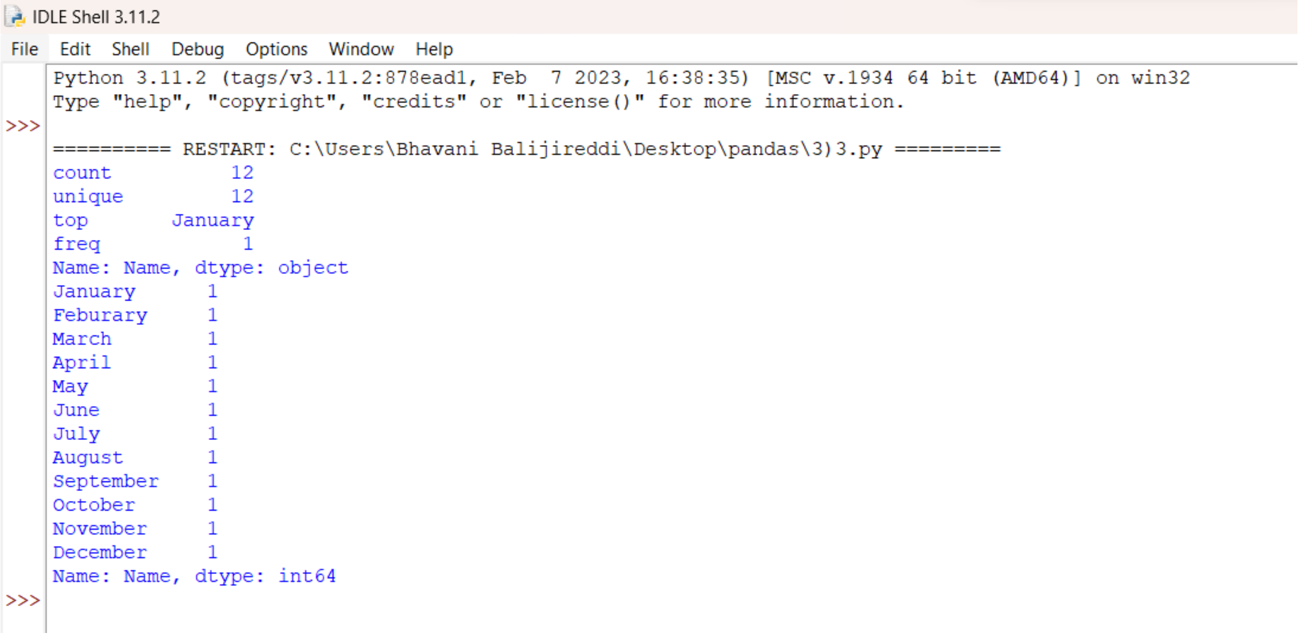
import pandas as pd
data = pd.read_csv(’month.csv’)
print(data[’Name’].describe(include=’Name’))
print(data[’Name’].value_counts())

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\3)3.py =========
count          12
unique         12
top       January
freq            1
Name: Name, dtype: object
January      1
Feburary     1
March        1
April        1
May          1
June         1
July         1
August       1
September    1
October      1
November     1
December     1
Name: Name, dtype: int64
>>>
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\3)3.py =========
count          12
unique         12
top       January
freq            1
Name: Name, dtype: object
January      1
Feburary     1
March        1
April        1
May          1
June         1
July         1
August       1
September    1
October      1
November     1
December     1
Name: Name, dtype: int64
>>>
```

(d) Handling duplicates

Aim: Handling duplicates

Description:

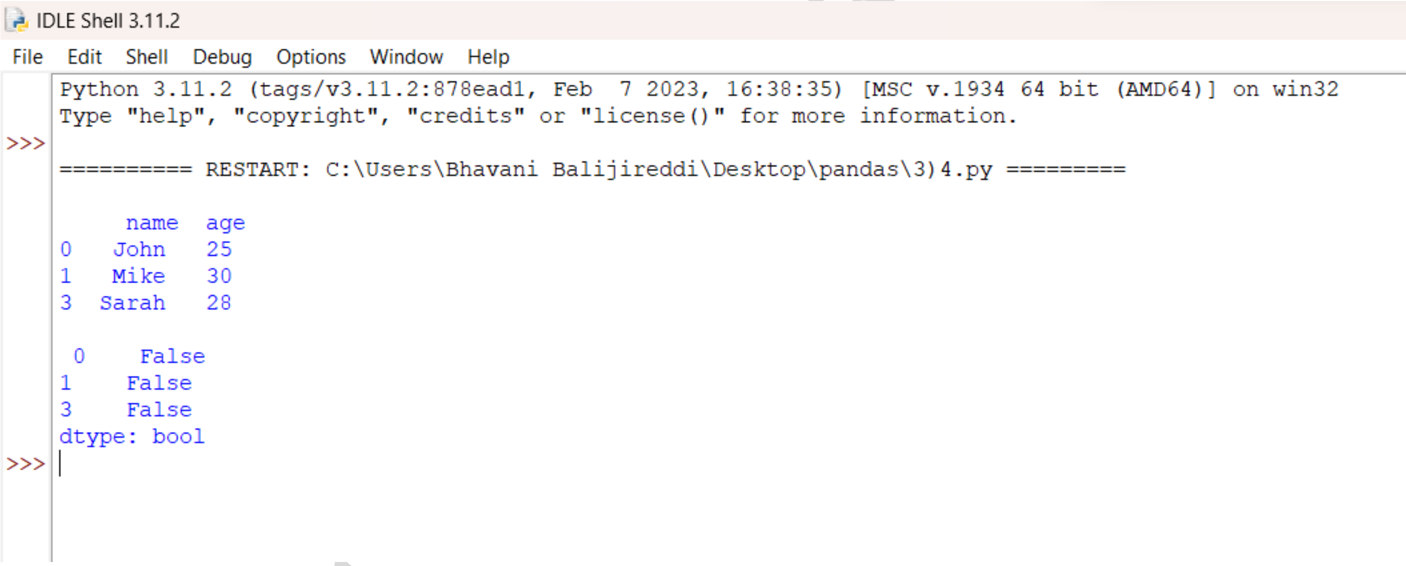
1.Handling duplicates refers to the process of identifying and managing duplicated data in a dataset.

2.To handle duplicates, there are various approaches that can be taken such as deleting or dropping duplicates, merging or aggregating duplicates, or flagging duplicates for further review. The choice of method often depends on the specific context and goals of the analysis or application.

Program:

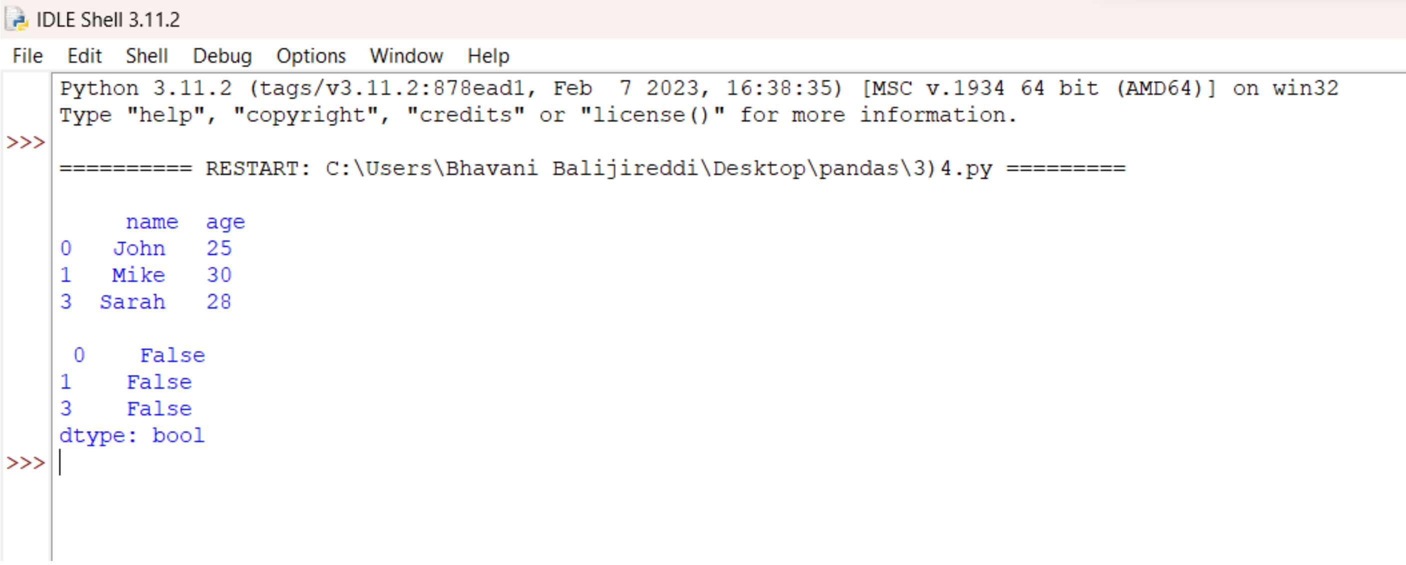
```
import pandas as pd data = {’name’: [’John’, ’Mike’, ’John’, ’Sarah’, ’Mike’], ’age’: [25, 30, 25, 28,
30]}
df = pd.DataFrame(data)
df = df.drop_duplicates()
print(’\n’,df)
duplicates = df.duplicated()
print(’\n’,duplicates)
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\3)4.py =========

     name  age
0    John   25
1    Mike   30
3   Sarah   28

 0     False
1     False
3     False
dtype: bool
>>> 
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\3)4.py =========

     name  age
0    John   25
1    Mike   30
3   Sarah   28

 0     False
1     False
3     False
dtype: bool
>>> 
```

(e)Number of observations Per Category

Aim: Number of observations Per Category

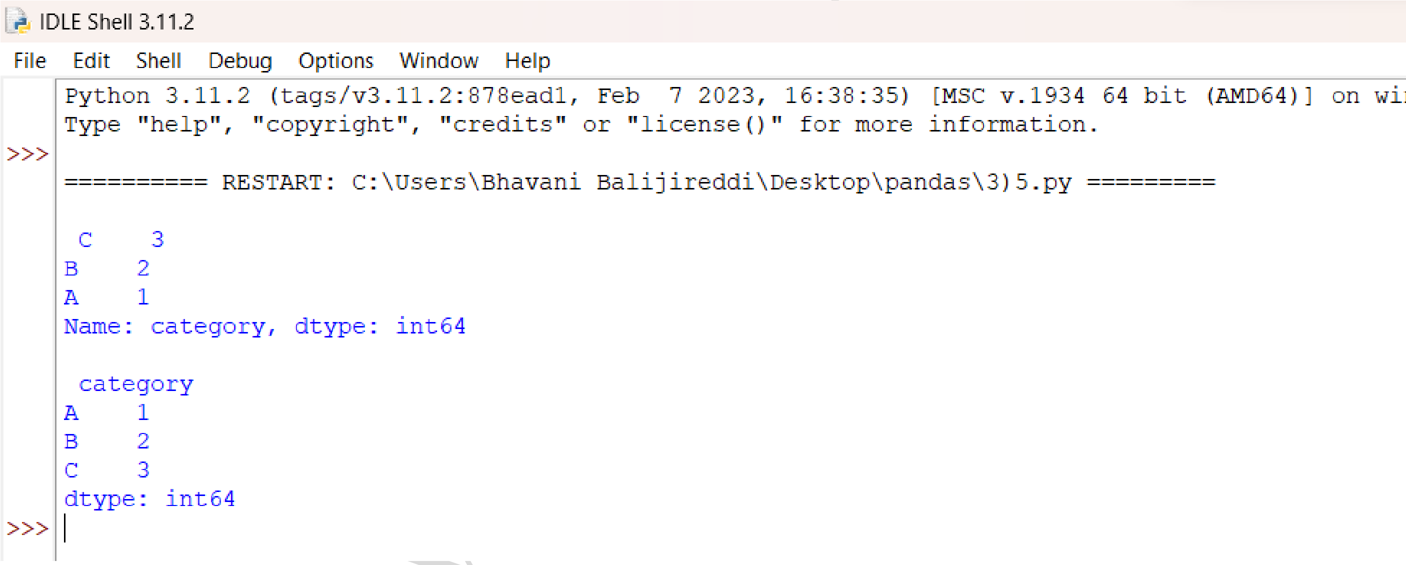
Description:

Number of observations per category refers to the count of data points or instances that belong to each category or group within a dataset. This information is useful in many data analysis tasks, such as understanding the distribution of data, identifying patterns or trends, and comparing different groups.

Program:

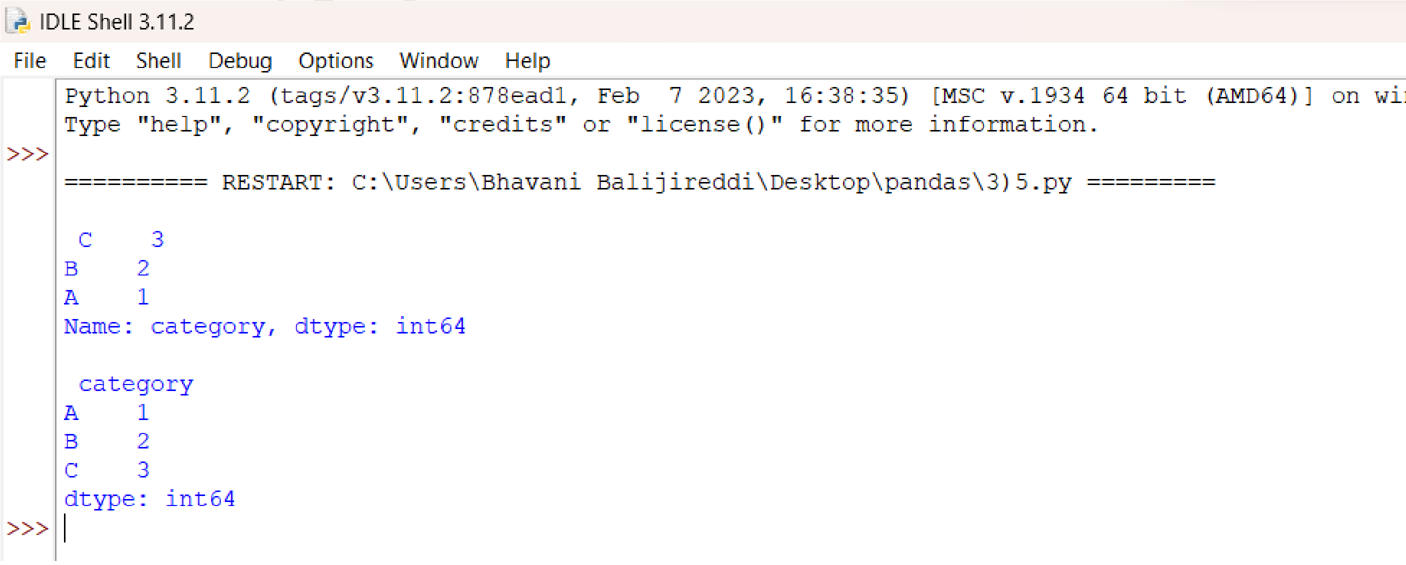
```
import pandas as pd
data = {’category’: [’A’, ’B’, ’B’, ’C’, ’C’, ’C’]}
df = pd.DataFrame(data)
counts = df[’category’].value_counts()
print(’\n’,counts)
counts = df.groupby(’category’).size()
print(’\n’,counts)
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on wi
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\3)5.py =========

 C    3
B    2
A    1
Name: category, dtype: int64

 category
A    1
B    2
C    3
dtype: int64
>>> 
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on wi
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\3)5.py =========

 C    3
B    2
A    1
Name: category, dtype: int64

 category
A    1
B    2
C    3
dtype: int64
>>> 
```

(f)Column cleanup

Aim: Column cleanup

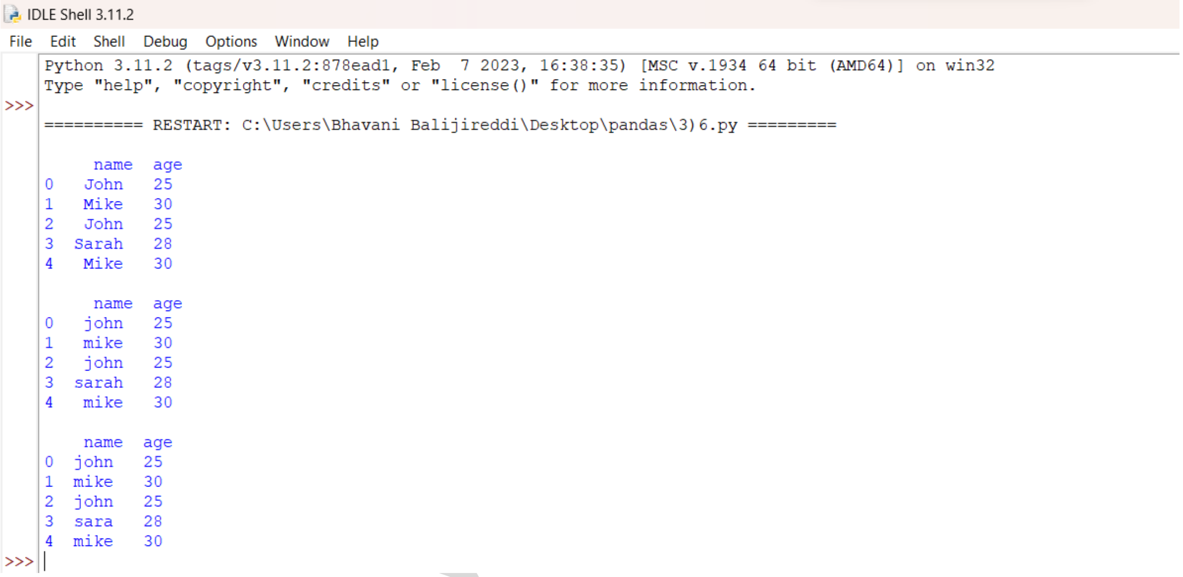
Description:

1.Column cleanup refers to the process of cleaning and transforming the data within individual columns of a dataset to improve their quality and usability. 2.Column cleanup can include tasks such as removing duplicates, correcting spelling errors, converting data types, and imputing missing values.

Program:

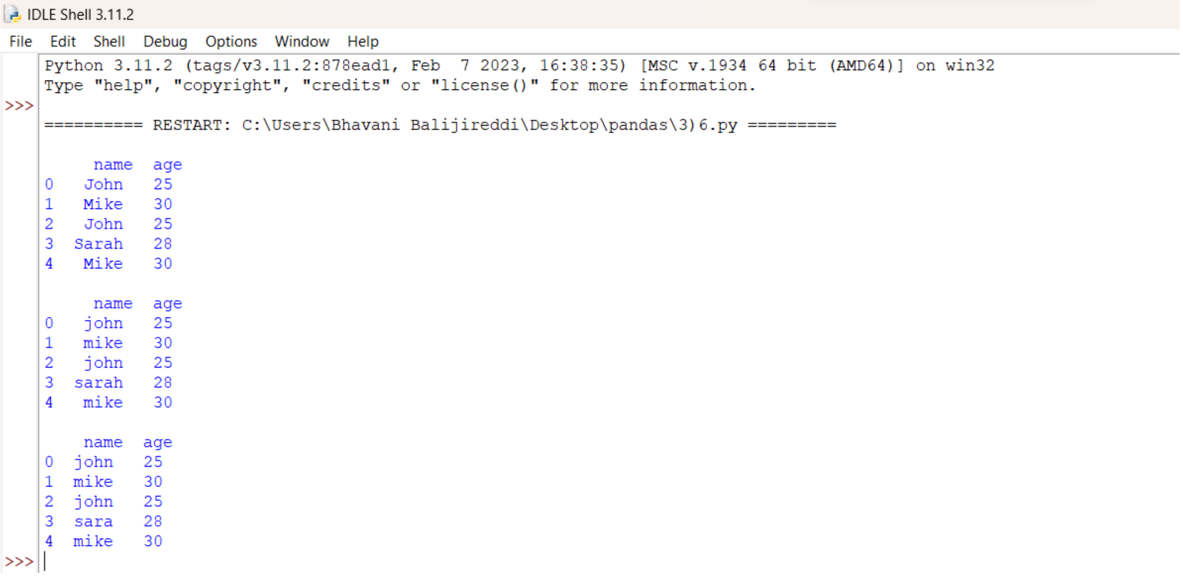
import pandas as pd
data = {'name': [' John', 'Mike ', ' John ', 'Sarah ', 'Mike '], 'age': [25, 30, 25, 28, 30]}
df = pd.DataFrame(data)
df['name'] = df['name'].str.strip()
print('\n',df)
df['name'] = df['name'].replace('sarah', 'sara')
print('\n',df)

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\3)6.py =========

     name  age
0   John   25
1   Mike   30
2   John   25
3  Sarah   28
4   Mike   30

     name  age
0   john   25
1   mike   30
2   john   25
3  sarah   28
4   mike   30

    name  age
0  john   25
1  mike   30
2  john   25
3  sara   28
4  mike   30
>>>
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\3)6.py =========

     name  age
0   John   25
1   Mike   30
2   John   25
3  Sarah   28
4   Mike   30

     name  age
0   john   25
1   mike   30
2   john   25
3  sarah   28
4   mike   30

    name  age
0  john   25
1  mike   30
2  john   25
3  sara   28
4  mike   30
>>>
```

4.Getting the Dataset continuation
a. Removing null values
b. Understanding your variables
c. Relationships between continuous variables
d. DataFrame slicing, selecting, extracting
e. Conditional selections

(a)Removing null values

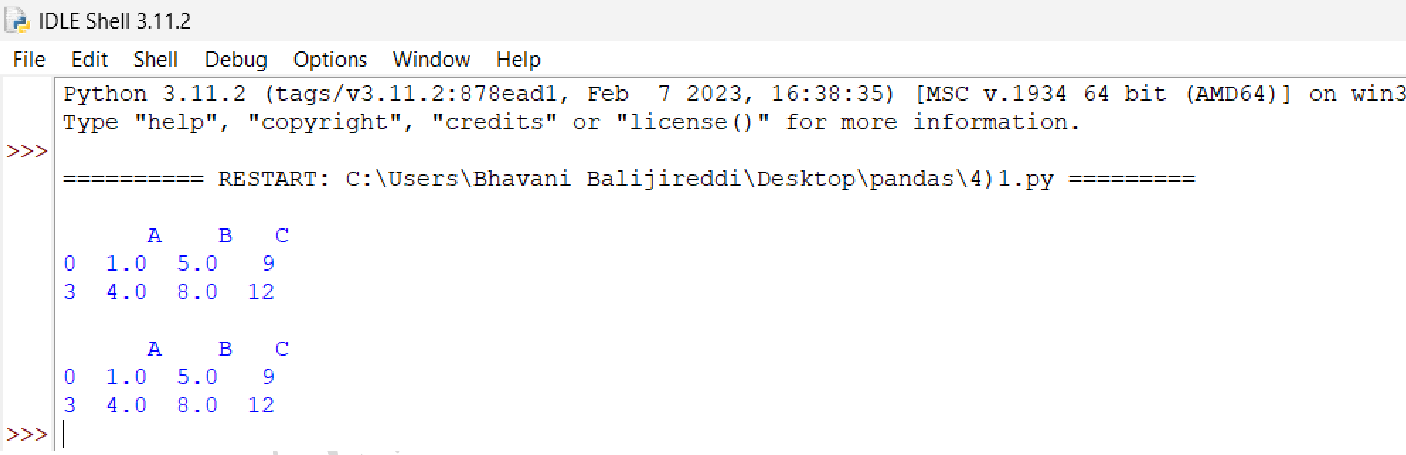
Aim: Removing null values

Description:

Removing null values refers to the process of identifying and eliminating missing or null values from a dataset. Null values, also known as missing values, are data points that are not available or have not been recorded for a particular variable or observation. **Program:**

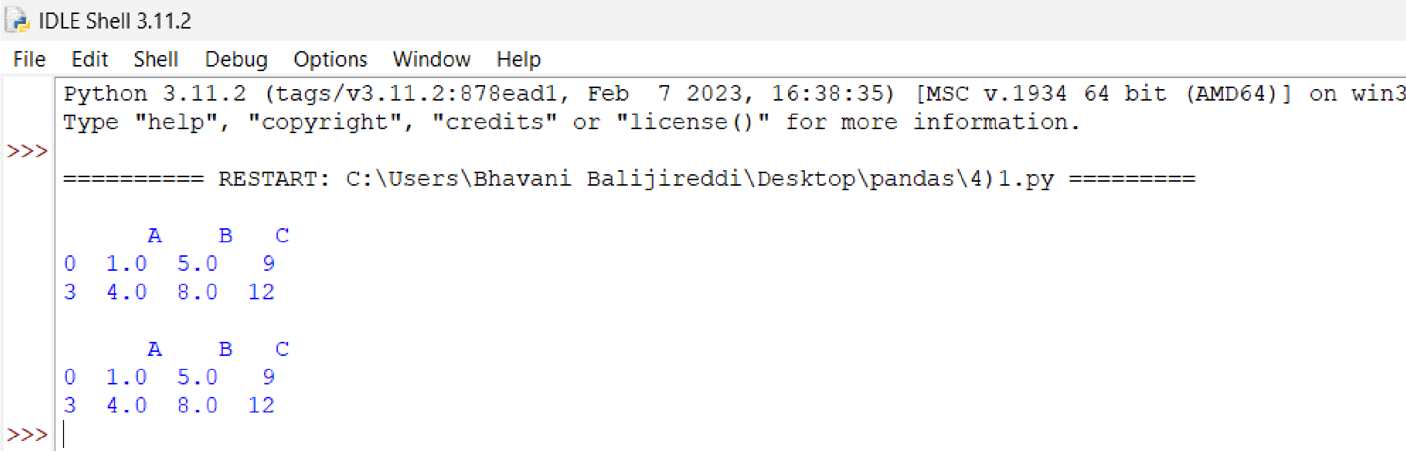
```
import pandas as pd
df = pd.DataFrame({’A’: [1, 2, None, 4],
’B’: [5, None, 7, 8],
’C’: [9, 10, 11, 12]})
df = df.dropna()
print(’\n’,df)
df = df.dropna(axis=1)
print(’\n’,df)
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win3
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\4)1.py =========

      A    B   C
0   1.0  5.0   9
3   4.0  8.0  12

      A    B   C
0   1.0  5.0   9
3   4.0  8.0  12
>>>
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win3
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\4)1.py =========

      A    B   C
0   1.0  5.0   9
3   4.0  8.0  12

      A    B   C
0   1.0  5.0   9
3   4.0  8.0  12
>>>
```

(b) Understanding your variables

Aim: Understanding your variables

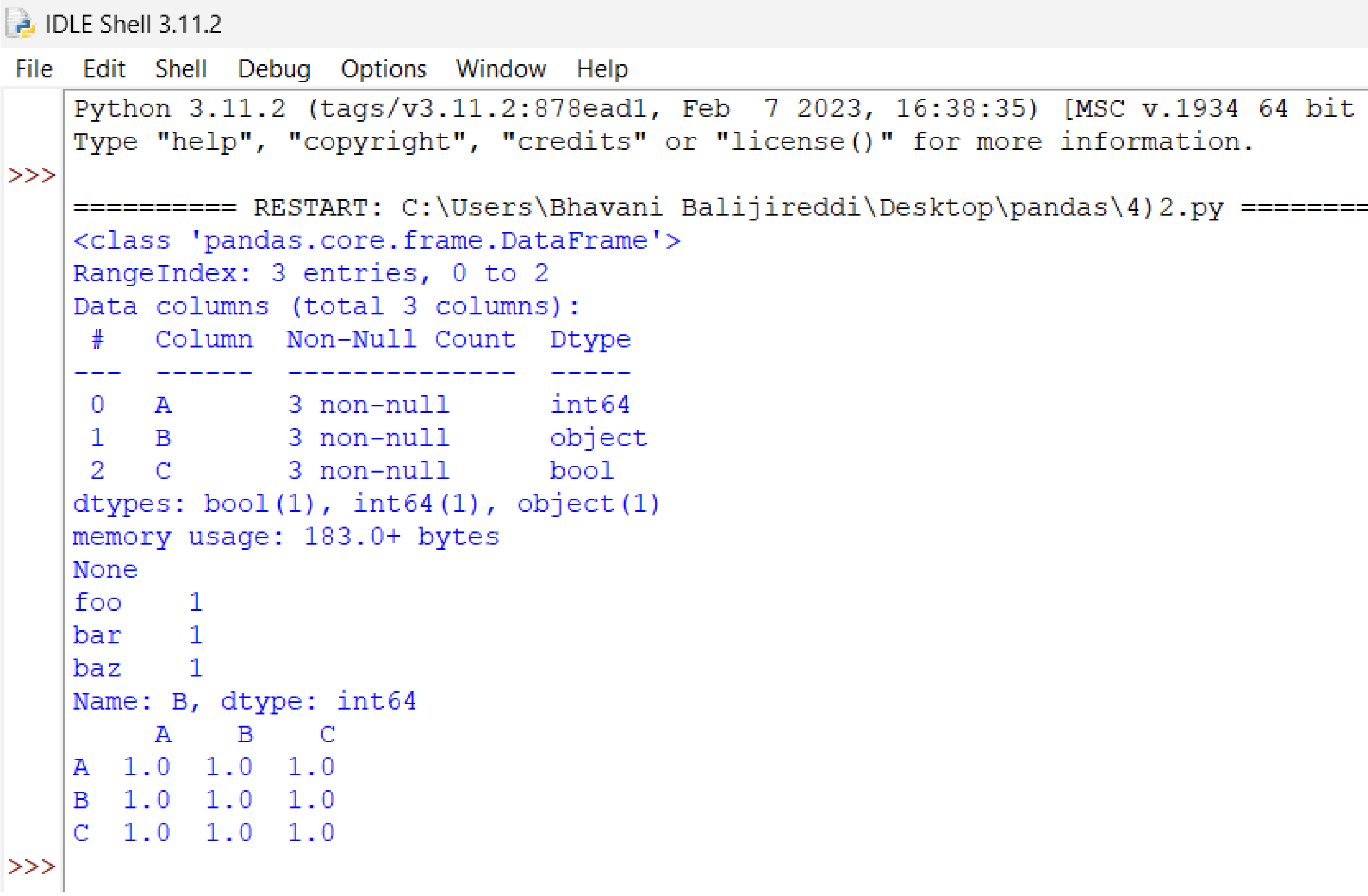
Description:

1.Understanding your variables is a crucial step in data analysis and modeling, which involves exploring and describing the characteristics and properties of the variables in a dataset. 2.Understanding your variables can help you to identify patterns, relationships, and outliers in the data, and to select appropriate analysis techniques or models.

Program:

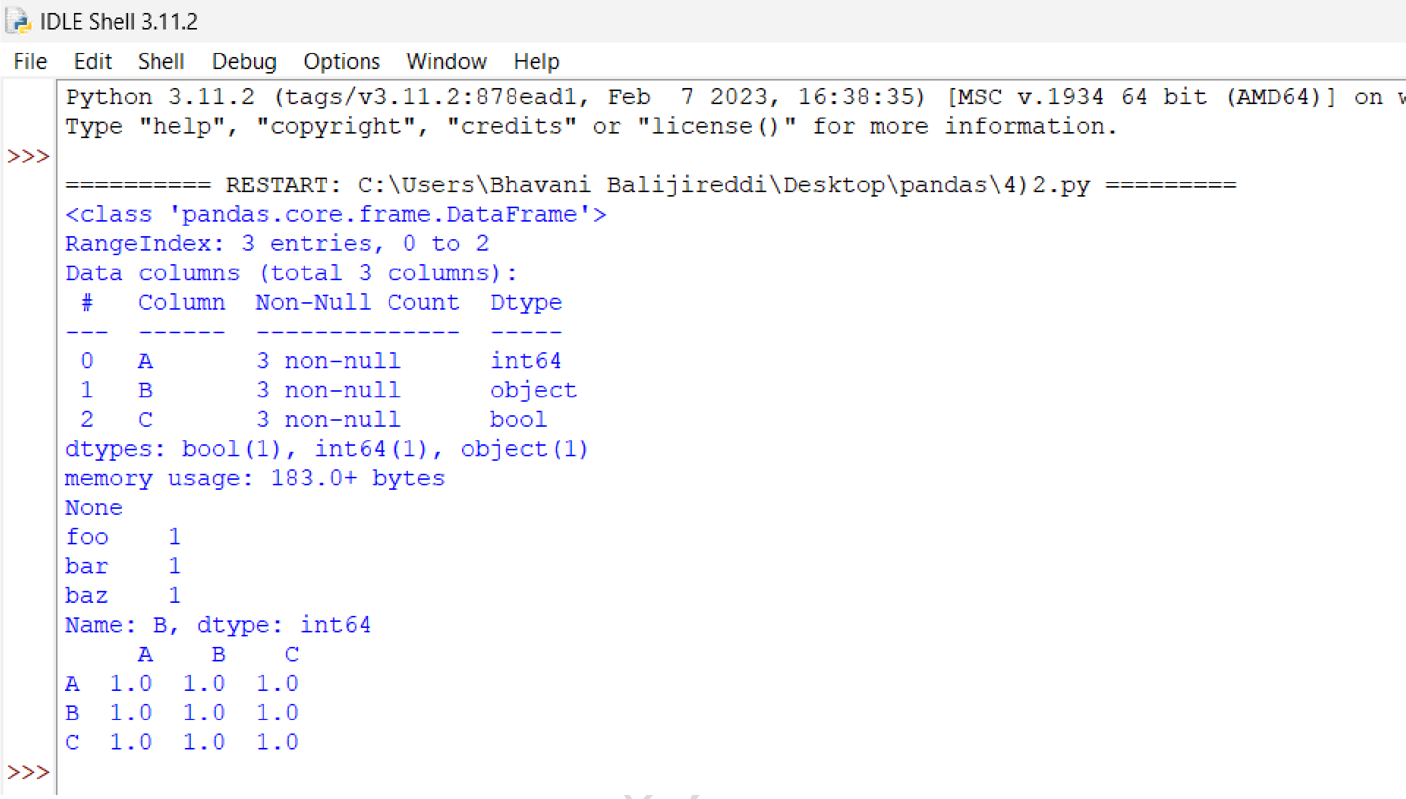
```
import pandas as pd
df = pd.DataFrame({’A’: [1, 2, 3],
’B’: [’foo’, ’bar’, ’baz’],
’C’: [True, False, True]})
df.info()
print(df.info())
df[’B’].value_counts()
print(df[’B’].value_counts())
df = pd.DataFrame({’A’: [1, 2, 3],
’B’: [4, 5, 6],
’C’: [7, 8, 9]})
print(df.corr())
```

Expected output:



```
    Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\4)2.py ========
    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 3 entries, 0 to 2
    Data columns (total 3 columns):
     #   Column  Non-Null Count  Dtype
    ---  ------  --------------  -----
     0   A       3 non-null      int64
     1   B       3 non-null      object
     2   C       3 non-null      bool
    dtypes: bool(1), int64(1), object(1)
    memory usage: 183.0+ bytes
    None
    foo    1
    bar    1
    baz    1
    Name: B, dtype: int64
         A    B    C
    A  1.0  1.0  1.0
    B  1.0  1.0  1.0
    C  1.0  1.0  1.0
>>>
```

Observed output:



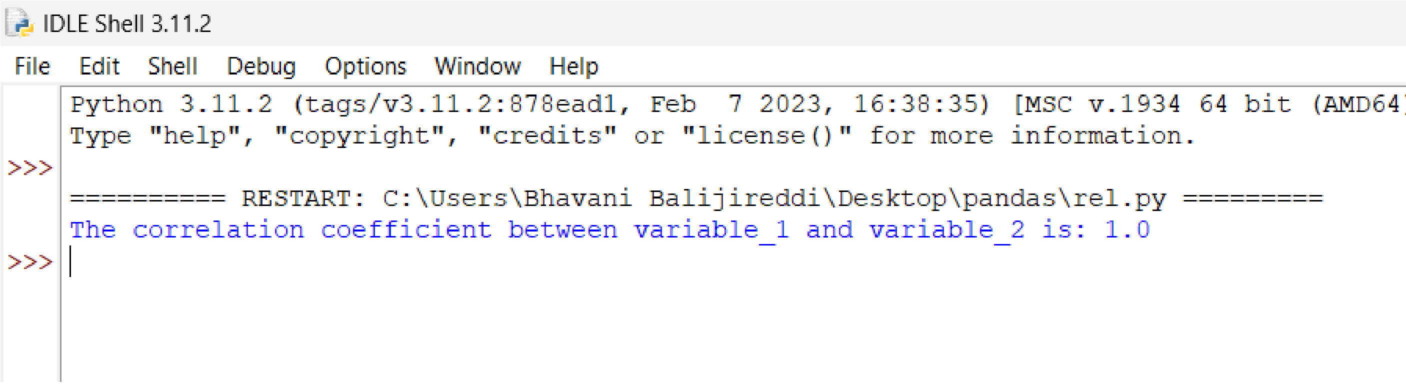
```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on w
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\4)2.py =========
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   A       3 non-null      int64 
 1   B       3 non-null      object
 2   C       3 non-null      bool  
dtypes: bool(1), int64(1), object(1)
memory usage: 183.0+ bytes
None
foo    1
bar    1
baz    1
Name: B, dtype: int64
     A    B    C
A  1.0  1.0  1.0
B  1.0  1.0  1.0
C  1.0  1.0  1.0
>>> 
```

(c)Relationships between continuous variables

Aim: Relationships between continuous variables
Description:
The relationship between continuous variables refers to the association or pattern of behavior that exists between two or more variables that are measured on a continuous scale. Continuous variables are those that can take on any value within a certain range, such as age, height, weight, and temperature.
Program:

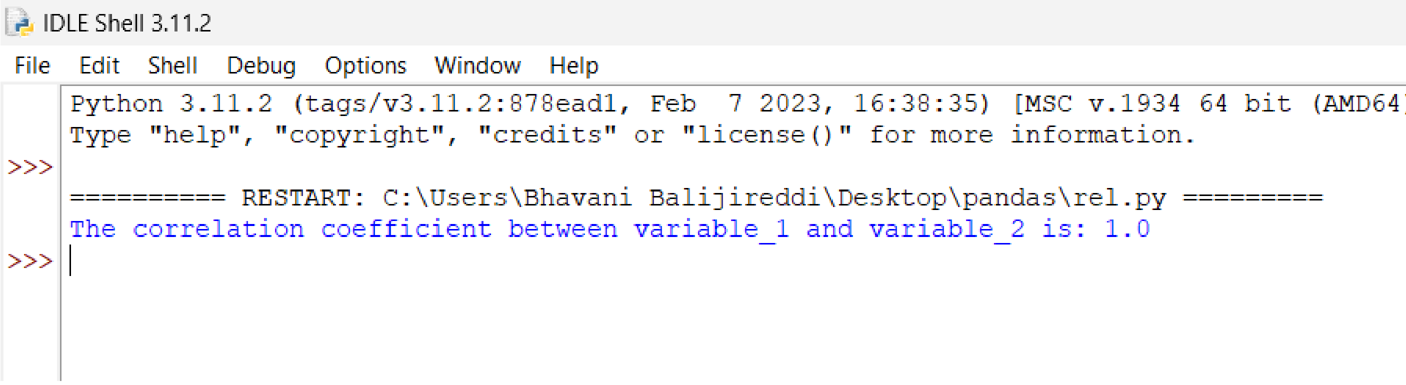
```
import pandas as pd
df = pd.DataFrame({'variable_1': [1, 2, 3, 4, 5],
'variable_2': [10, 15, 20, 25, 30]})
correlation_coefficient = df['variable_1'].corr(df['variable_2'])
print("The correlation coefficient between variable_1 and variable_2 is:", correlation_coefficient)
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64
Type "help", "copyright", "credits" or "license()" for more information.
>>> 
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\rel.py =========
The correlation coefficient between variable_1 and variable_2 is: 1.0
>>> |
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64
Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\rel.py =========
    The correlation coefficient between variable_1 and variable_2 is: 1.0
>>> |
```

(d)DataFrame slicing, selecting, extracting

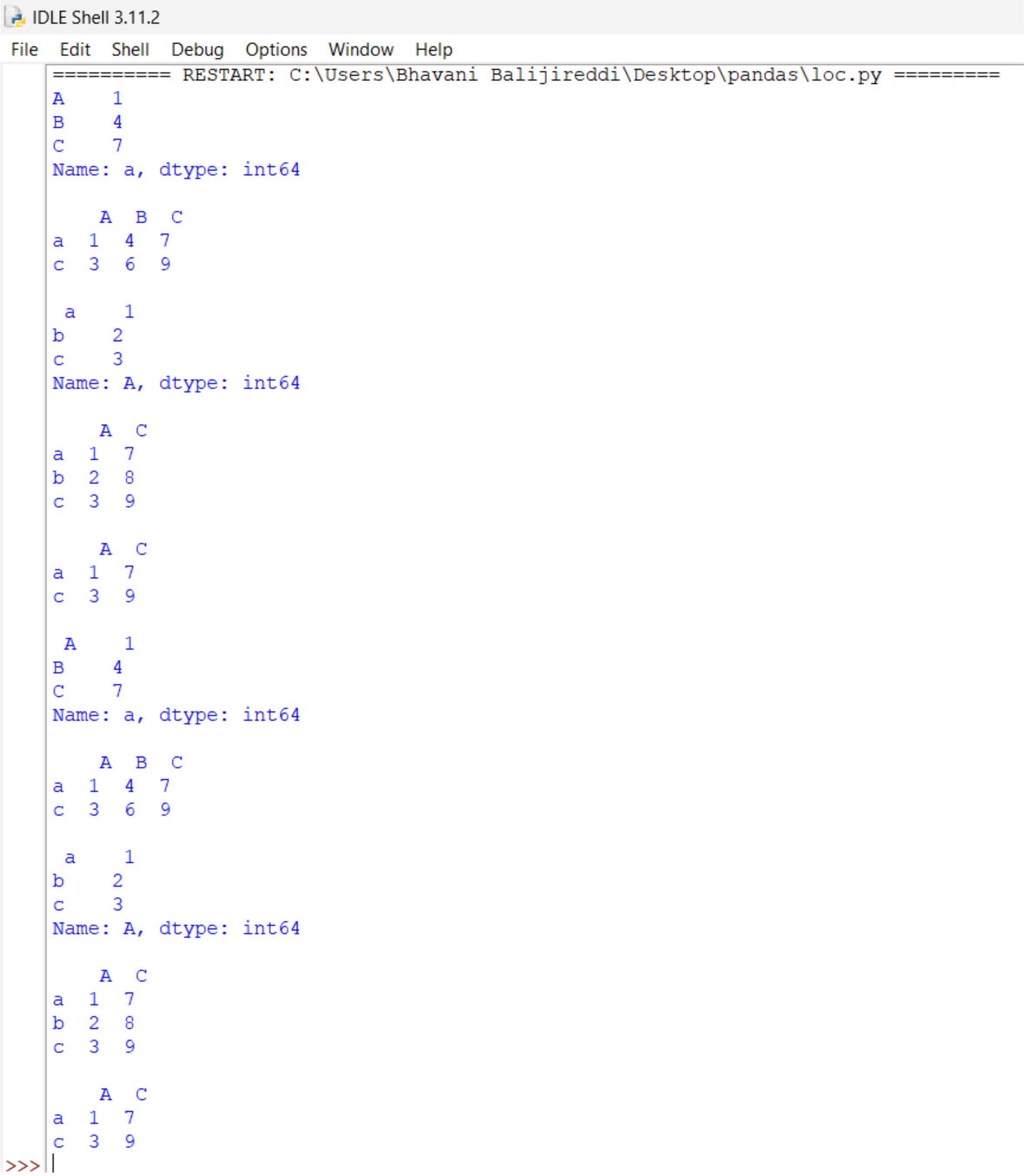
Aim: DataFrame slicing, selecting, extracting

Description:

1.Slicing involves selecting a subset of the rows and/or columns from a DataFrame based on a specific range of indices. This can be achieved using the ’.loc’. 2.Selecting involves filtering the rows and/or columns of a DataFrame based on certain conditions. 3.Extracting involves retrieving a specific column or row of data from a DataFrame. This can be done using the indexing operator’ [] ’, which allows you to select a column by label, or the’ .loc’ and ’.iloc ’accessor methods. **Program:**

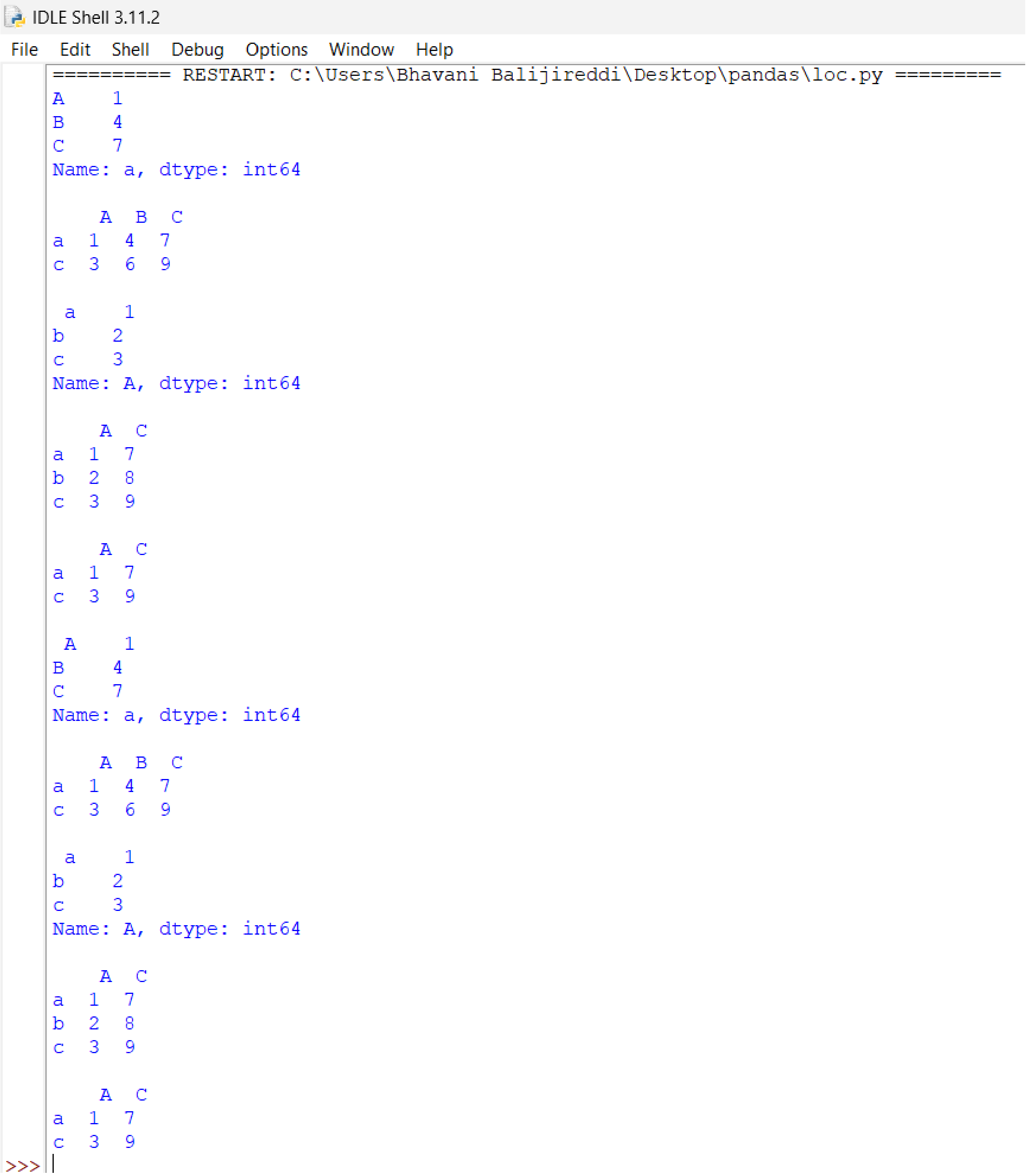
```
import pandas as pd
df = pd.DataFrame({’A’: [1, 2, 3],
’B’: [4, 5, 6],
’C’: [7, 8, 9]},
index=[’a’, ’b’, ’c’])
print(df.loc[’a’])
print(’\n’,df.loc[[’a’, ’c’]])
print(’\n’,df.loc[:, ’A’])
print(’\n’,df.loc[:, [’A’, ’C’]])
print(’\n’,df.loc[[’a’, ’c’], [’A’, ’C’]])
print(’\n’,df.iloc[0])
print(’\n’,df.iloc[[0, 2]])
print(’\n’,df.iloc[:, 0])
print(’\n’,df.iloc[:, [0, 2]])
print(’\n’,df.iloc[[0, 2], [0, 2]])
```

Expected output:



```
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\loc.py =========
A    1
B    4
C    7
Name: a, dtype: int64

    A  B  C
a  1  4  7
c  3  6  9

 a     1
b    2
c    3
Name: A, dtype: int64

    A  C
a  1  7
b  2  8
c  3  9

    A  C
a  1  7
c  3  9

 A     1
B    4
C    7
Name: a, dtype: int64

    A  B  C
a  1  4  7
c  3  6  9

 a     1
b    2
c    3
Name: A, dtype: int64

    A  C
a  1  7
b  2  8
c  3  9

    A  C
a  1  7
c  3  9
>>> |
```

Observed output:



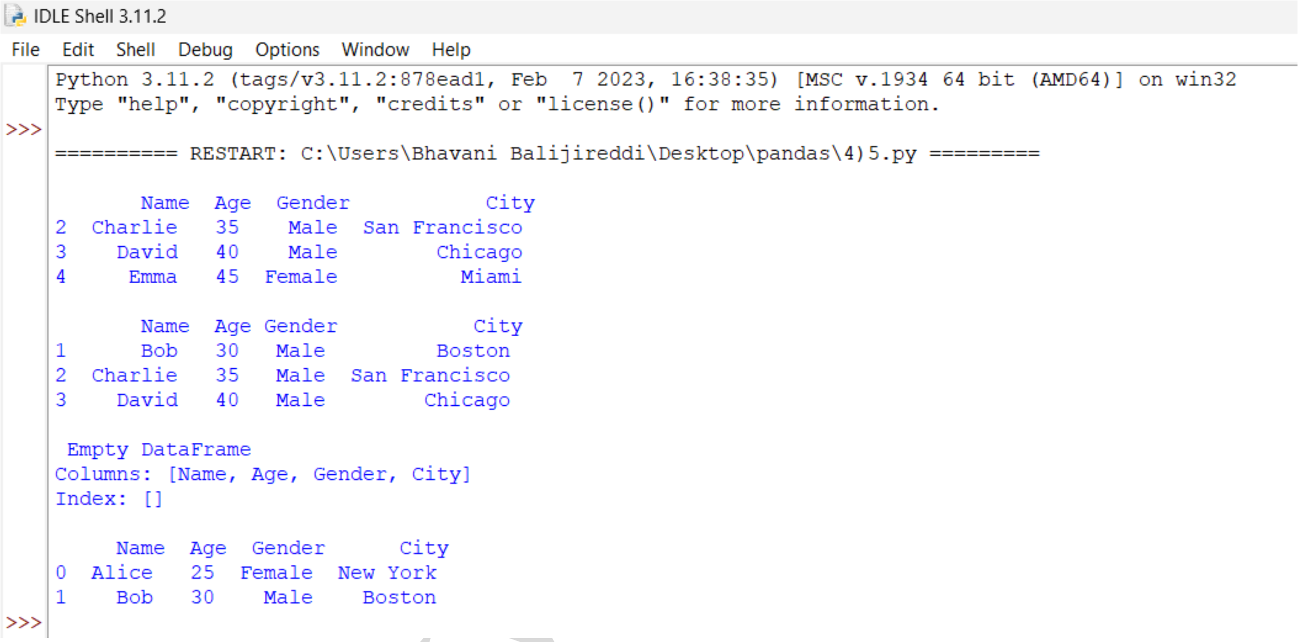
```
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\loc.py =========
A    1
B    4
C    7
Name: a, dtype: int64

   A  B  C
a  1  4  7
c  3  6  9

 a    1
b    2
c    3
Name: A, dtype: int64

   A  C
a  1  7
b  2  8
c  3  9

   A  C
a  1  7
c  3  9

 A    1
B    4
C    7
Name: a, dtype: int64

   A  B  C
a  1  4  7
c  3  6  9

 a    1
b    2
c    3
Name: A, dtype: int64

   A  C
a  1  7
b  2  8
c  3  9

   A  C
a  1  7
c  3  9
>>> |
```

(e)Conditional selections
Aim: Conditional selections
Description:
Conditional selection is a technique used in data analysis to extract specific subsets of data from a larger dataset based on certain conditions or criteria. In pandas, a popular Python library for data analysis, conditional selection can be achieved using Boolean indexing.

Program:

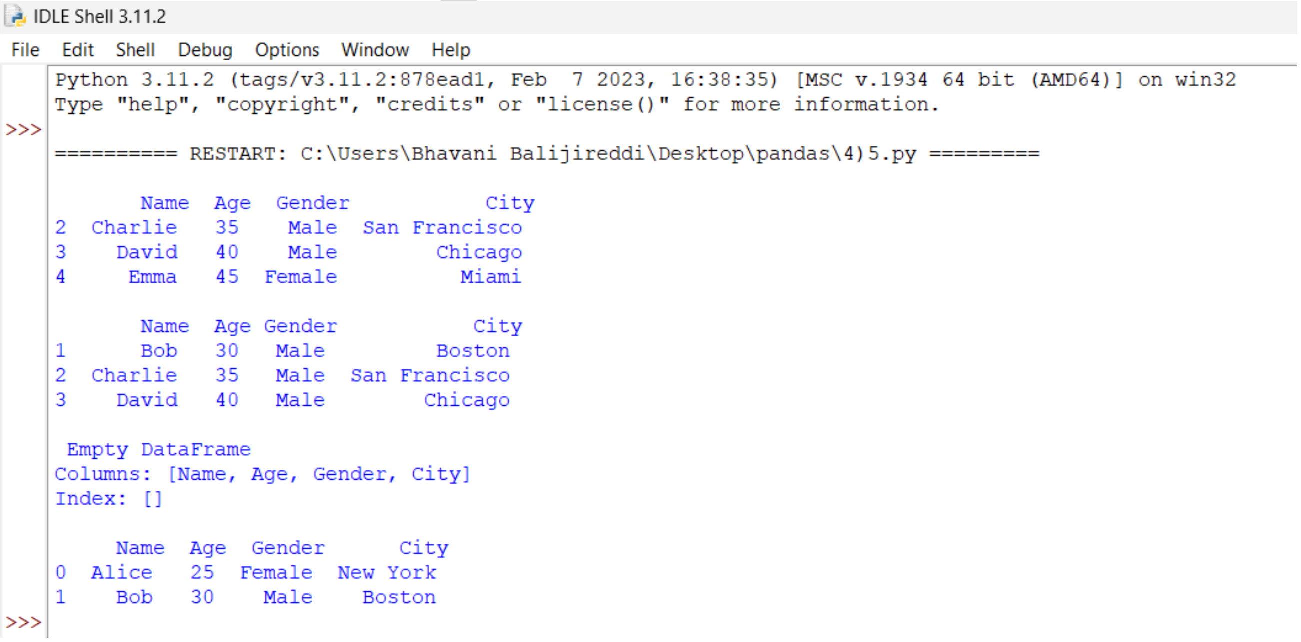
```
import pandas as pd
df = pd.DataFrame({
’Name’: [’Alice’, ’Bob’, ’Charlie’, ’David’, ’Emma’], ’Age’: [25, 30, 35, 40, 45],
’Gender’: [’Female’, ’Male’, ’Male’, ’Male’, ’Female’],
’City’: [’New York’, ’Boston’, ’San Francisco’, ’Chicago’, ’Miami’]})
print(’\n’, df[df[’Age’] ¿ 30])
print(’\n’,df.loc[df[’Gender’] == ’Male’])
print(’\n’,df.query(’Age ¿ 30 and City == ”Boston”’))
print(’\n’,df[df[’City’].isin([’New York’, ’Boston’])])
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\4)5.py =========

       Name  Age  Gender           City
2  Charlie   35    Male  San Francisco
3    David   40    Male        Chicago
4     Emma   45  Female          Miami

       Name  Age Gender           City
1      Bob   30   Male         Boston
2  Charlie   35   Male  San Francisco
3    David   40   Male        Chicago

 Empty DataFrame
Columns: [Name, Age, Gender, City]
Index: []

     Name  Age  Gender      City
0  Alice   25  Female  New York
1    Bob   30    Male    Boston
>>>
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\4)5.py =========

       Name  Age  Gender           City
2  Charlie   35    Male  San Francisco
3    David   40    Male        Chicago
4     Emma   45  Female          Miami

       Name  Age Gender           City
1      Bob   30   Male         Boston
2  Charlie   35   Male  San Francisco
3    David   40   Male        Chicago

 Empty DataFrame
Columns: [Name, Age, Gender, City]
Index: []

     Name  Age  Gender      City
0  Alice   25  Female  New York
1    Bob   30    Male    Boston
>>>
```

5.Getting Preview of DataFrame

a. Creating DataFrames from scratch
b. Looking at top n records
c. Looking at bottom n records
d. View columns names

(a)Creating DataFrames from scratch

Aim: Creating DataFrames from scratch

Description:

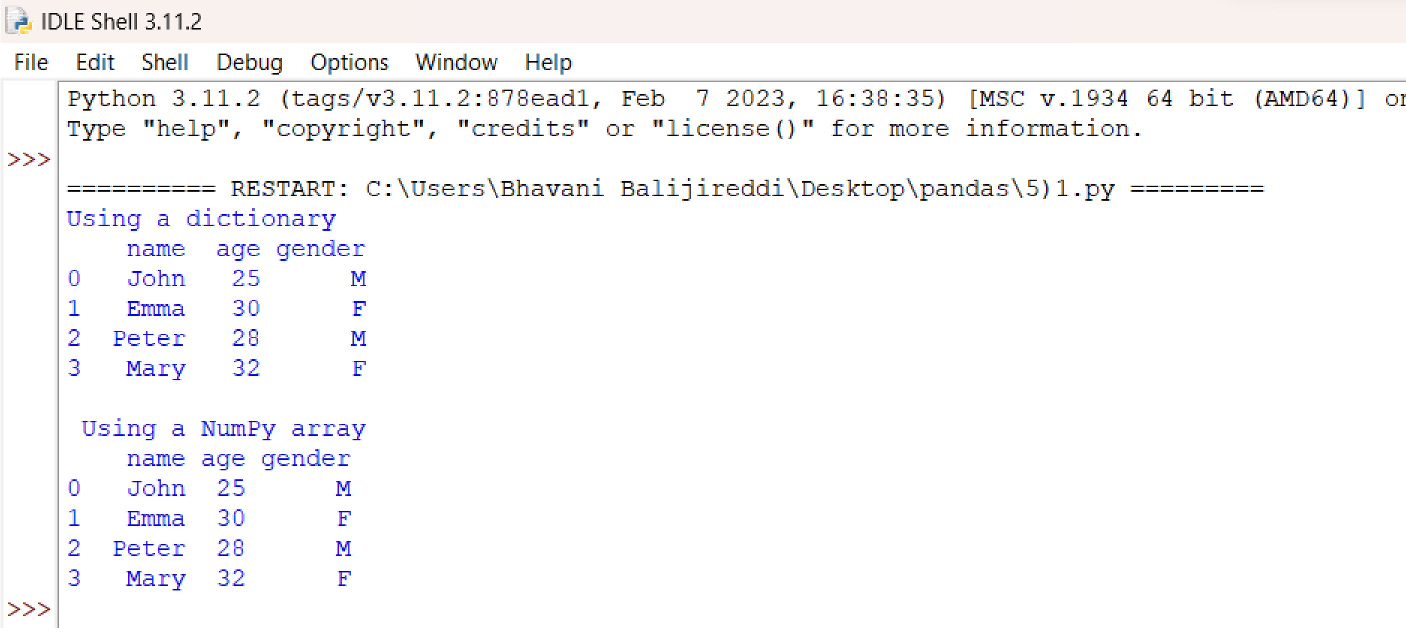
Creating DataFrames from scratch:

We can create DataFrames from scratch using various methods, such as creating a dictionary and converting it to a DataFrame, using a list of lists, or using NumPy arrays. Pandas provide the DataFrame() function to create a DataFrame from a data structure.

Program:

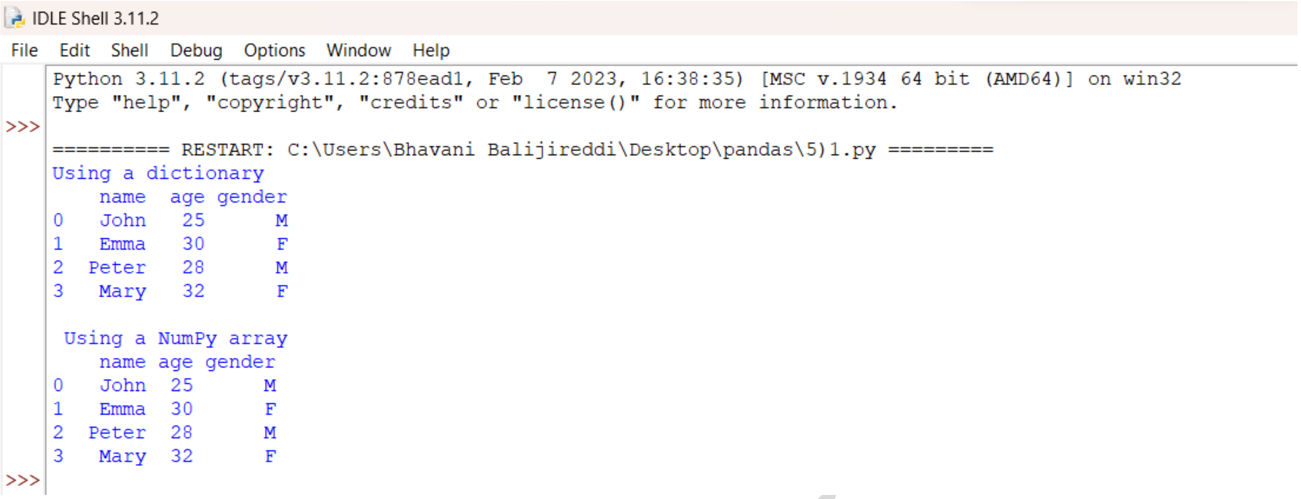
```
print(”Using a dictionary”)
import pandas as pd
data = {’name’: [’John’, ’Emma’, ’Peter’, ’Mary’],
’age’: [25, 30, 28, 32],
’gender’: [’M’, ’F’, ’M’, ’F’]}
df = pd.DataFrame(data)
print(df)
print(’\n’,”Using a NumPy array”)
import pandas as pd
import numpy as np
data = np.array([[’John’, 25, ’M’], [’Emma’, 30, ’F’], [’Peter’, 28, ’M’], [’Mary’, 32, ’F’]])
columns = [’name’, ’age’, ’gender’]
df = pd.DataFrame(data, columns=columns)
print(df)
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on
Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\5)1.py =========
    Using a dictionary
        name  age gender
    0   John   25      M
    1   Emma   30      F
    2  Peter   28      M
    3   Mary   32      F

     Using a NumPy array
        name age gender
    0   John  25      M
    1   Emma  30      F
    2  Peter  28      M
    3   Mary  32      F
>>>
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\5)1.py =========
Using a dictionary
    name  age gender
0   John   25      M
1   Emma   30      F
2  Peter   28      M
3   Mary   32      F

 Using a NumPy array
    name age gender
0   John  25      M
1   Emma  30      F
2  Peter  28      M
3   Mary  32      F
>>>
```

(b) Looking at top n records

Aim: Looking at top n records

Description:

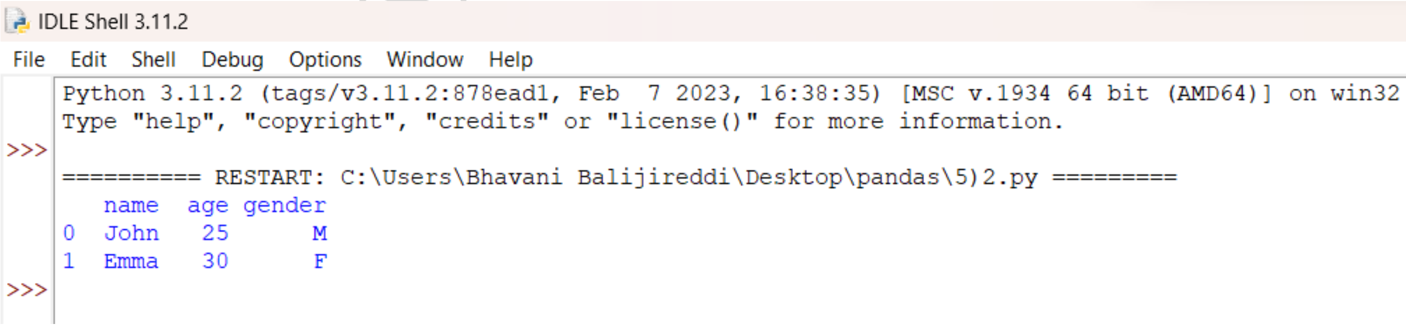
Looking at top n records:

To preview the top records of a DataFrame, we can use the head() function, which returns the first n rows of the DataFrame. By default, it returns the first five rows of the DataFrame, but we can specify the number of rows we want to see.

Program:

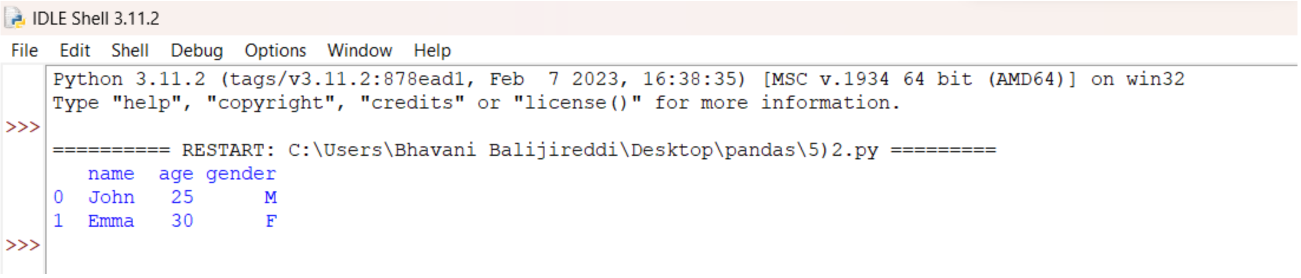
import pandas as pd
data = {'name': ['John', 'Emma', 'Peter', 'Mary','bunny','sunny'],
'age': [25, 30, 28, 32,26,40],
'gender': ['M', 'F', 'M', 'F','M','M']}
df = pd.DataFrame(data)
print(df.head(2))

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\5)2.py =========
    name  age gender
0   John   25      M
1   Emma   30      F
>>>
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\5)2.py =========
   name  age gender
0  John   25      M
1  Emma   30      F
>>>
```

(c)Looking at bottom n records

Aim: Looking at bottom n records

Description:

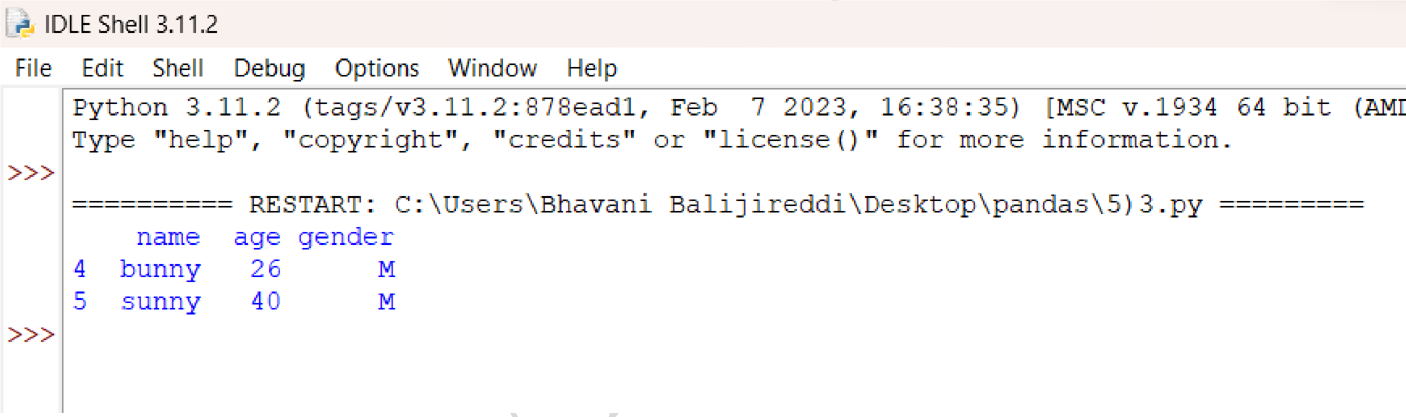
Looking at bottom n records:

To preview the bottom records of a DataFrame, we can use the tail() function, which returns the last n rows of the DataFrame. By default, it returns the last five rows of the DataFrame, but we can specify the number of rows we want to see.

Program:

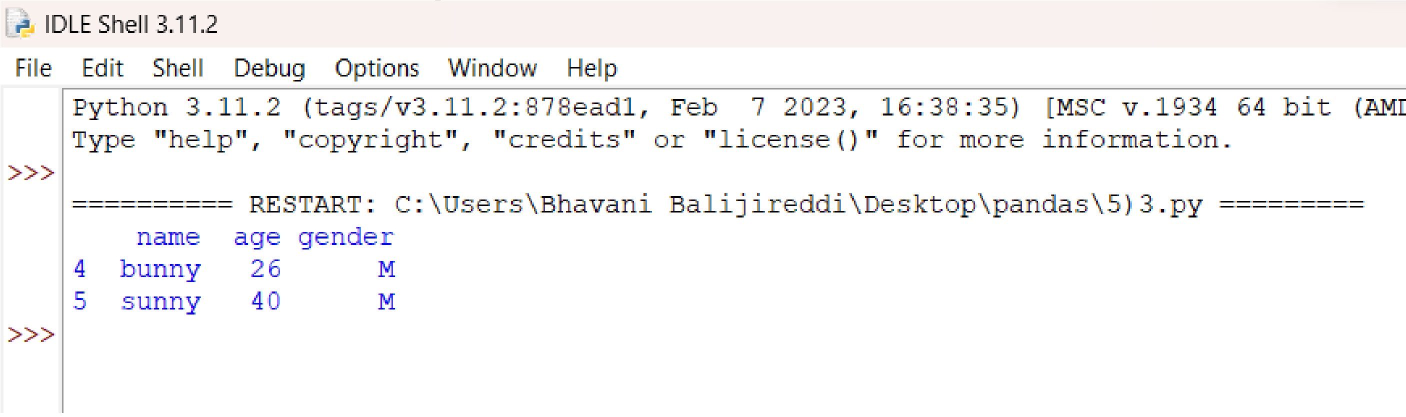
```
import pandas as pd
data = {’name’: [’John’, ’Emma’, ’Peter’, ’Mary’,’bunny’,’sunny’],
’age’: [25, 30, 28, 32,26,40],
’gender’: [’M’, ’F’, ’M’, ’F’,’M’,’M’]}
df = pd.DataFrame(data)
print(df.tail(2))
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\5)3.py =========
    name  age gender
4  bunny   26      M
5  sunny   40      M
>>>
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\5)3.py =========
    name  age gender
4  bunny   26      M
5  sunny   40      M
>>>
```

(d)View columns names

Aim: View columns names

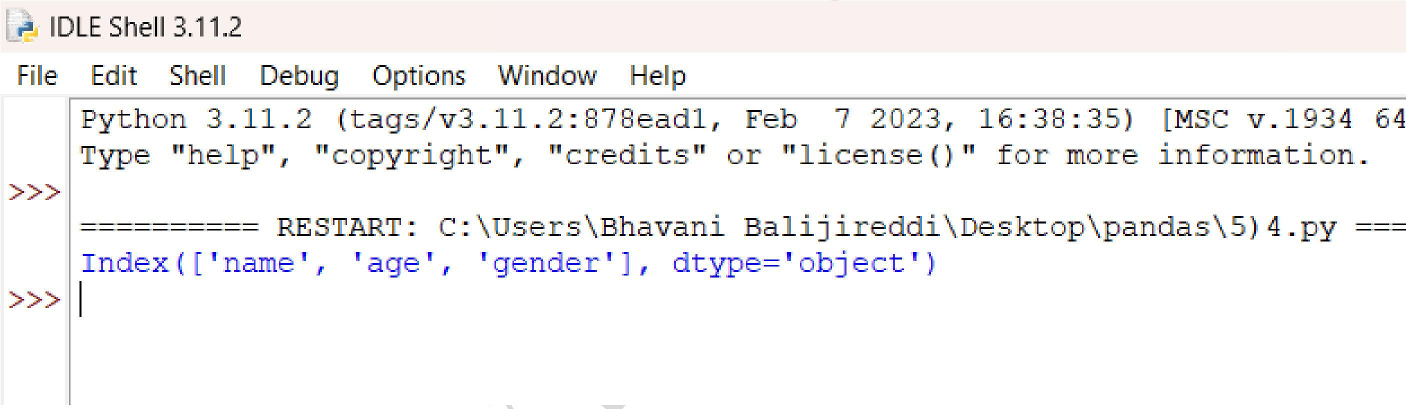
Description:

View column names:

We can view the column names of a DataFrame by using the columns attribute, which returns an index object containing the column names. Alternatively, we can use the head() function with a parameter of 0 to view the column names. This will return only the column names and not any data from the DataFrame. **Program:**

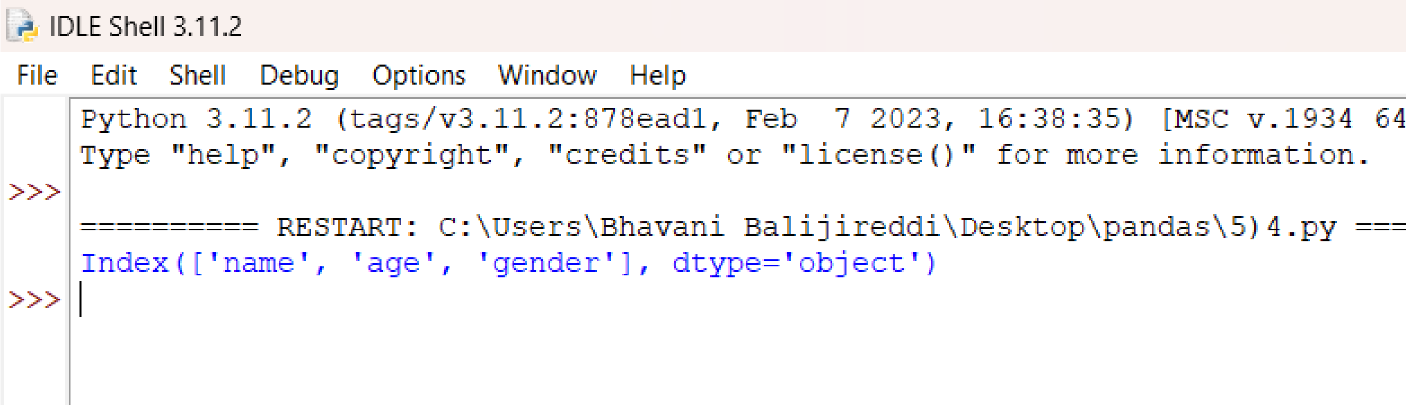
import pandas as pd
data = {'name': ['John', 'Emma', 'Peter', 'Mary'],
'age': [25, 30, 28, 32],
'gender': ['M', 'F', 'M', 'F']}
df = pd.DataFrame(data)
print(df.columns)

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\5)4.py ===
Index(['name', 'age', 'gender'], dtype='object')
>>>
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\5)4.py ===
Index(['name', 'age', 'gender'], dtype='object')
>>>
```

6.Creating New Columns, Rename Columns of Data Frames

a. Rename method helps to rename column of data frame
b. To rename the column of existing data frame set inplace=True

(a)Rename method helps to rename column of data frame

Aim: Rename method helps to rename column of data frame

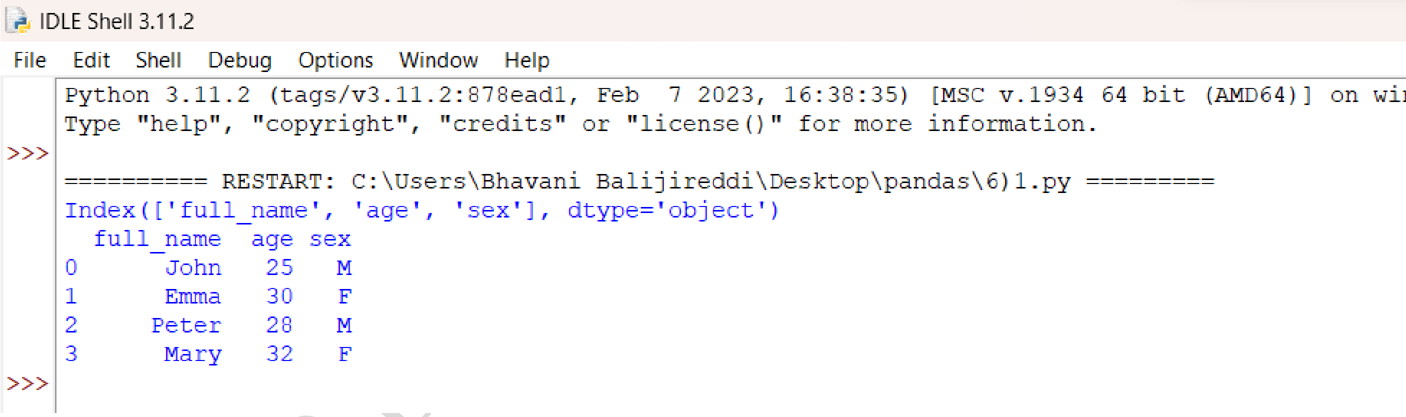
Description:

The rename() method is a powerful tool in pandas, a popular Python library for data analysis, that allows you to rename columns of a DataFrame. This method provides a convenient way to rename columns without having to modify the original DataFrame object.
To use the rename() method, you can pass a dictionary to the columns parameter where the keys represent the old column names and the values represent the new column names.

Program:

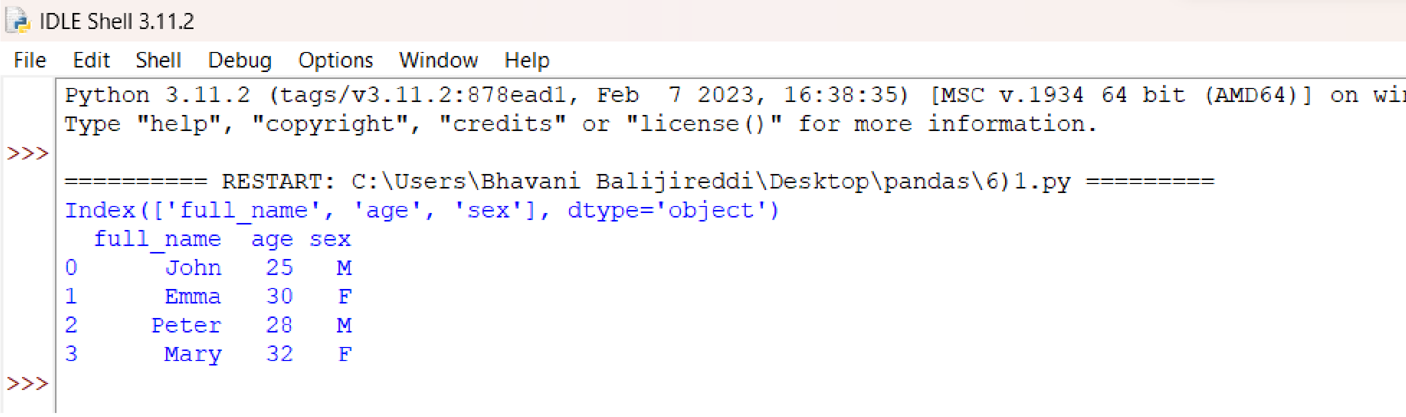
```
import pandas as pd
data = {’name’: [’John’, ’Emma’, ’Peter’, ’Mary’],
’age’: [25, 30, 28, 32],
’gender’: [’M’, ’F’, ’M’, ’F’]}
df = pd.DataFrame(data)
df = df.rename(columns={’name’: ’full_name’, ’gender’: ’sex’})
print(df.columns)
print(df)
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on wi
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\6)1.py =========
Index(['full_name', 'age', 'sex'], dtype='object')
  full_name  age sex
0      John   25   M
1      Emma   30   F
2     Peter   28   M
3      Mary   32   F
>>>
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64)] on wi
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\6)1.py =========
Index(['full_name', 'age', 'sex'], dtype='object')
  full_name  age sex
0      John   25   M
1      Emma   30   F
2     Peter   28   M
3      Mary   32   F
>>>
```

(b)To rename the column of existing data frame set inplace=True

Aim: To rename the column of existing data frame set inplace=True

Description:

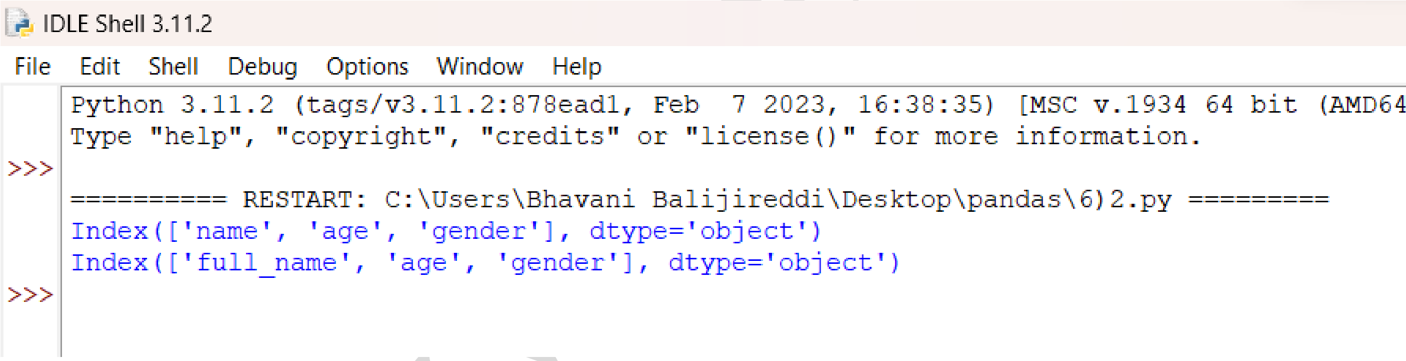
When you want to rename columns of an existing pandas DataFrame in place, you can use the rename() method with the inplace=True parameter. This parameter allows you to modify the original DataFrame object without creating a new object.

To rename columns in place, you can use the rename() method and set inplace=True to modify the original DataFrame object. For example, the following code renames the column ”old_name” to ”new_name” in a DataFrame called df.

Program:

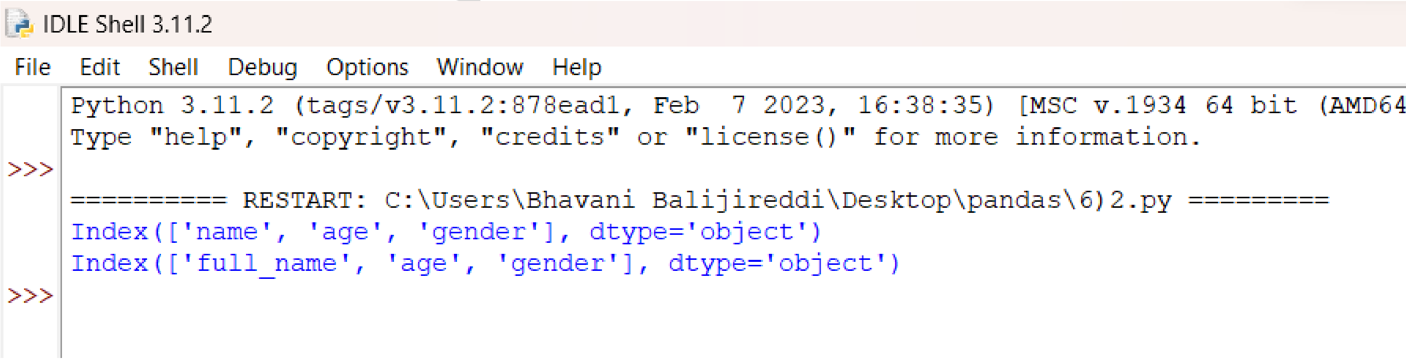
```
import pandas as pd
data = {’name’: [’John’, ’Emma’, ’Peter’, ’Mary’],
’age’: [25, 30, 28, 32],
’gender’: [’M’, ’F’, ’M’, ’F’]}
df = pd.DataFrame(data)
print(df.columns)
df.rename(columns={’name’: ’full_name’}, inplace=True)
print(df.columns
```

Expected output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\6)2.py =========
Index(['name', 'age', 'gender'], dtype='object')
Index(['full_name', 'age', 'gender'], dtype='object')
>>>
```

Observed output:



```
Python 3.11.2 (tags/v3.11.2:878ead1, Feb  7 2023, 16:38:35) [MSC v.1934 64 bit (AMD64
Type "help", "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: C:\Users\Bhavani Balijireddi\Desktop\pandas\6)2.py =========
Index(['name', 'age', 'gender'], dtype='object')
Index(['full_name', 'age', 'gender'], dtype='object')
>>>
```

7.Selecting Columns or Rows
a. Accessing sub data frames
b. Filtering Records

(a)Accessing sub data frames

Aim: Write a python program for Accessing sub data frames using pandas

Description:

1.Indexing operator []: You can use the indexing operator [] to select one or more columns from a data frame. For example, df[’column_name’] .

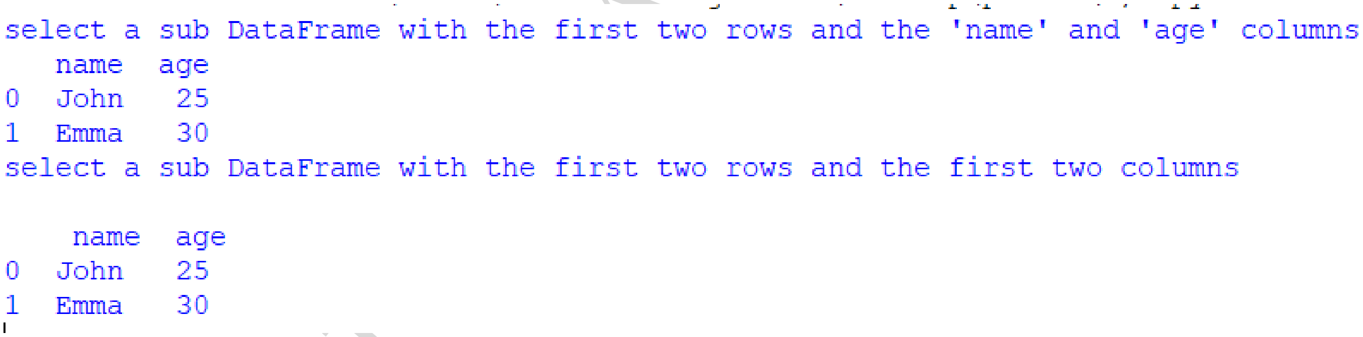
2.returns a sub data frame with the selected column, and df[[’column_1’, ’column_2’]] returns a sub data frame with the selected columns.

3.’.loc ’accessor method: You can use the .loc accessor method to select rows and columns by label. For example, df.loc[row_label, column_label] returns a sub data frame with the selected row and column.

Program:

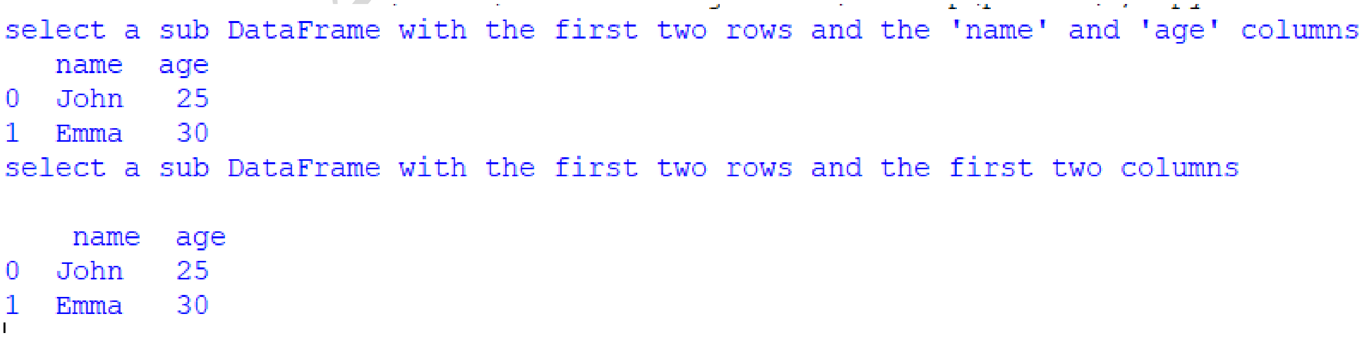
```
import pandas as pd
data = {’name’: [’John’, ’Emma’, ’Peter’, ’Mary’],
’age’: [25, 30, 28, 32],
’gender’: [’M’, ’F’, ’M’, ’F’]}
df = pd.DataFrame(data)
sub_df = df.loc[0:1, [’name’, ’age’]]
print(sub_df)
sub_df = df.iloc[0:2, 0:2]
print(’\n’,sub_df)
```

Expected output:



```
select a sub DataFrame with the first two rows and the 'name' and 'age' columns
   name  age
0  John   25
1  Emma   30
select a sub DataFrame with the first two rows and the first two columns

    name  age
0  John   25
1  Emma   30
```

Observed output:



```
select a sub DataFrame with the first two rows and the 'name' and 'age' columns
   name  age
0  John   25
1  Emma   30
select a sub DataFrame with the first two rows and the first two columns

    name  age
0  John   25
1  Emma   30
```

(b)Filtering Records

Aim: Write a python program for Filtering Records using pandas

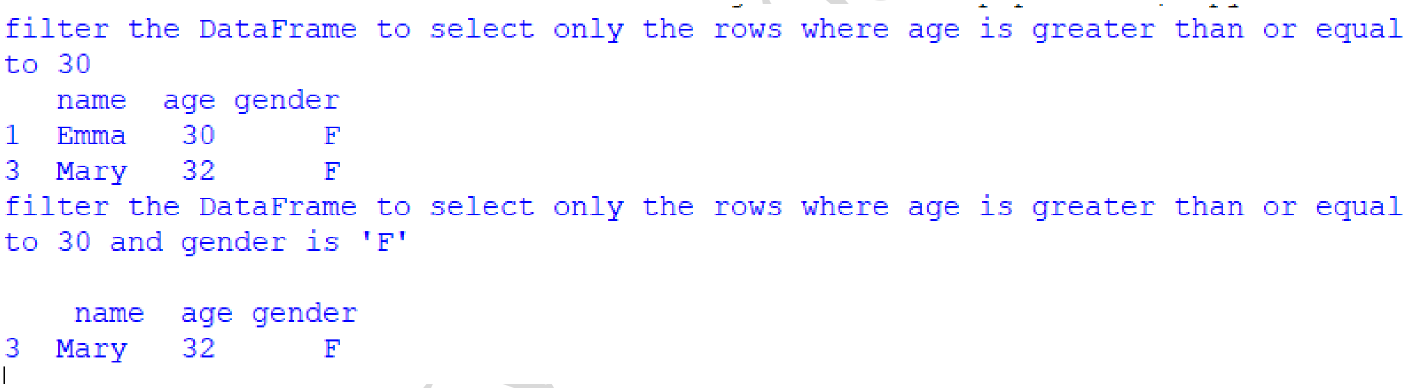
Description:

Filtering records: Filtering records involves selecting only the rows that meet specific criteria based on the values in the DataFrame. We can use boolean indexing to filter records. Boolean indexing involves creating a Boolean condition that evaluates to True or False for each row in the DataFrame. We can then use this condition to select the rows that meet the specified criteria.

Program:

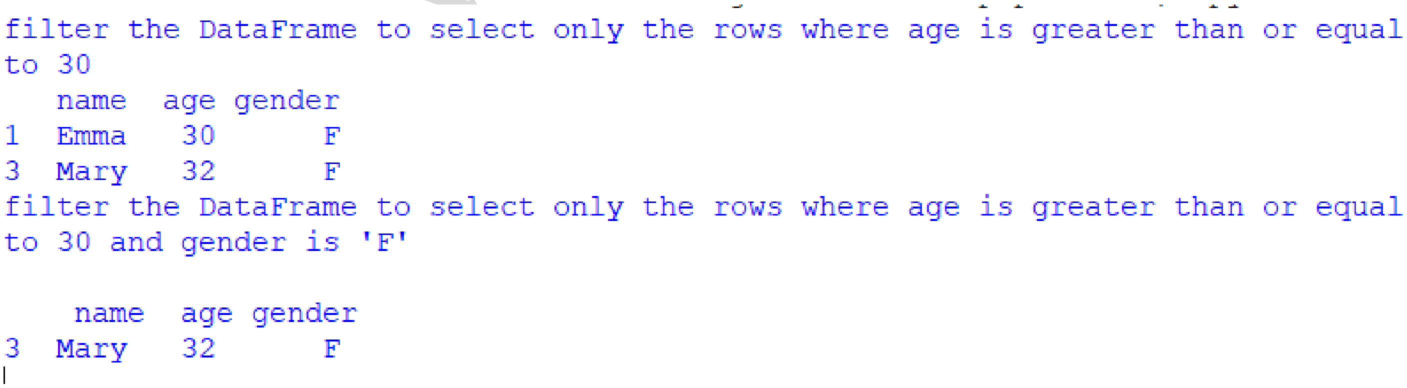
```
import pandas as pd
data = {’name’: [’John’, ’Emma’, ’Peter’, ’Mary’],
’age’: [25, 30, 28, 32],
’gender’: [’M’, ’F’, ’M’, ’F’]}
df = pd.DataFrame(data)
filtered_df = df[df[’age’] >= 30]
print(filtered_df)
filter_df = df[(df[’age’] >30) & (df[’gender’] == ’F’)]
print(’\n’,filter_df)
```

Expected output:



```
filter the DataFrame to select only the rows where age is greater than or equal
to 30
   name  age gender
1  Emma   30      F
3  Mary   32      F
filter the DataFrame to select only the rows where age is greater than or equal
to 30 and gender is 'F'

    name  age gender
3  Mary   32      F
```

Observed output:



```
filter the DataFrame to select only the rows where age is greater than or equal
to 30
   name  age gender
1  Emma   30      F
3  Mary   32      F
filter the DataFrame to select only the rows where age is greater than or equal
to 30 and gender is 'F'

    name  age gender
3  Mary   32      F
```

8. Handling Missing Values

a. Dropna

b. Fillna

c. Recognize and Treat missing values and outliers in Pandas

(a) Dropna

Aim: Write a python program for Dropna using pandas

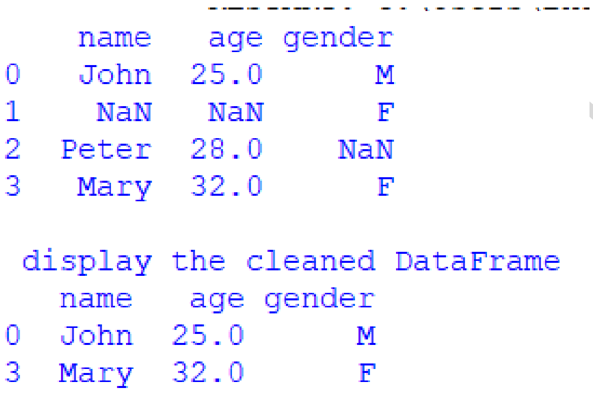
Description:

a. Dropna: Dropna is a method in Pandas that allows you to remove rows or columns with missing values from a DataFrame. This method can be used with the dropna() function, and you can specify the axis (0 for rows and 1 for columns) to drop rows or columns with missing values. It is a straightforward approach to handling missing values, but it may result in loss of data if you choose to drop rows or columns with missing values.

Program:

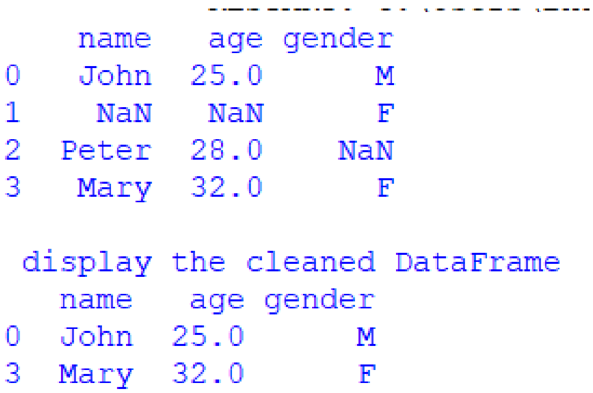
```
import pandas as pd
import numpy as np
data = {’name’: [’John’, np.nan, ’Peter’, ’Mary’],
’age’: [25, np.nan, 28, 32],
’gender’: [’M’, ’F’, np.nan, ’F’]}
df = pd.DataFrame(data)
print(df)
clean_df = df.dropna()
print(’\n’,”display the cleaned DataFrame”)
print(clean_df)
```

Expected output:



```
    name   age gender
0   John  25.0      M
1    NaN   NaN      F
2  Peter  28.0    NaN
3   Mary  32.0      F

 display the cleaned DataFrame
   name   age gender
0  John  25.0      M
3  Mary  32.0      F
```

Observed output:



```
    name   age gender
0   John  25.0      M
1    NaN   NaN      F
2  Peter  28.0    NaN
3   Mary  32.0      F

 display the cleaned DataFrame
   name   age gender
0  John  25.0      M
3  Mary  32.0      F
```

(b) Fillna

Aim: Write a python program for Fillna using pandas

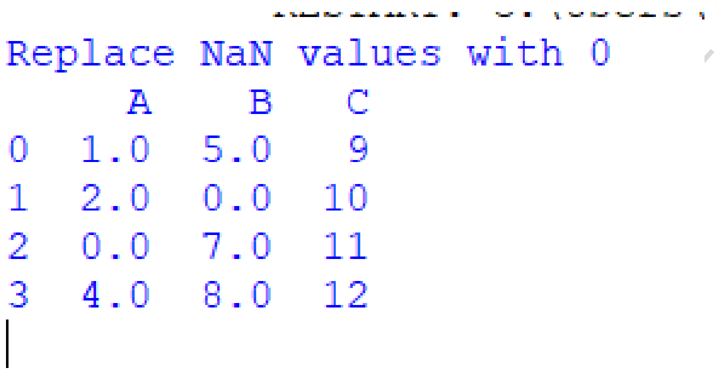
Description:

b. Fillna: Fillna is a method in Pandas that allows you to fill in missing values in a DataFrame with specified values or using various filling techniques. You can use the fillna() function on a DataFrame and specify the value or method to fill missing values. For example, you can fill missing values with a constant value, or use techniques such as forward fill (filling with the previous value) or backward fill (filling with the next value) to fill in missing values. You can also use statistical methods such as mean, median, or mode imputation to fill missing values based on the values of other data points in the same column.

Program:

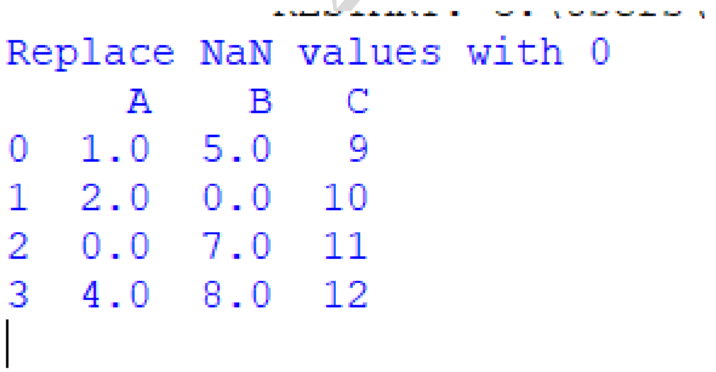
```
import pandas as pd
import numpy as np
df = pd.DataFrame({
'A': [1, 2, np.nan, 4],
'B': [5, np.nan, 7, 8],
'C': [9, 10, 11, 12]
})
print("Replace NaN values with 0")
df.fillna(0, inplace=True)
print(df)
```

Expected output:



```
Replace NaN values with 0
     A    B   C
0  1.0  5.0   9
1  2.0  0.0  10
2  0.0  7.0  11
3  4.0  8.0  12
```

Observed output:



```
Replace NaN values with 0
     A    B   C
0  1.0  5.0   9
1  2.0  0.0  10
2  0.0  7.0  11
3  4.0  8.0  12
```

(c) Recognize and Treat missing values and outliers in Pandas

Aim: Write a python program for Recognize and Treat missing values and outliers in Pandas

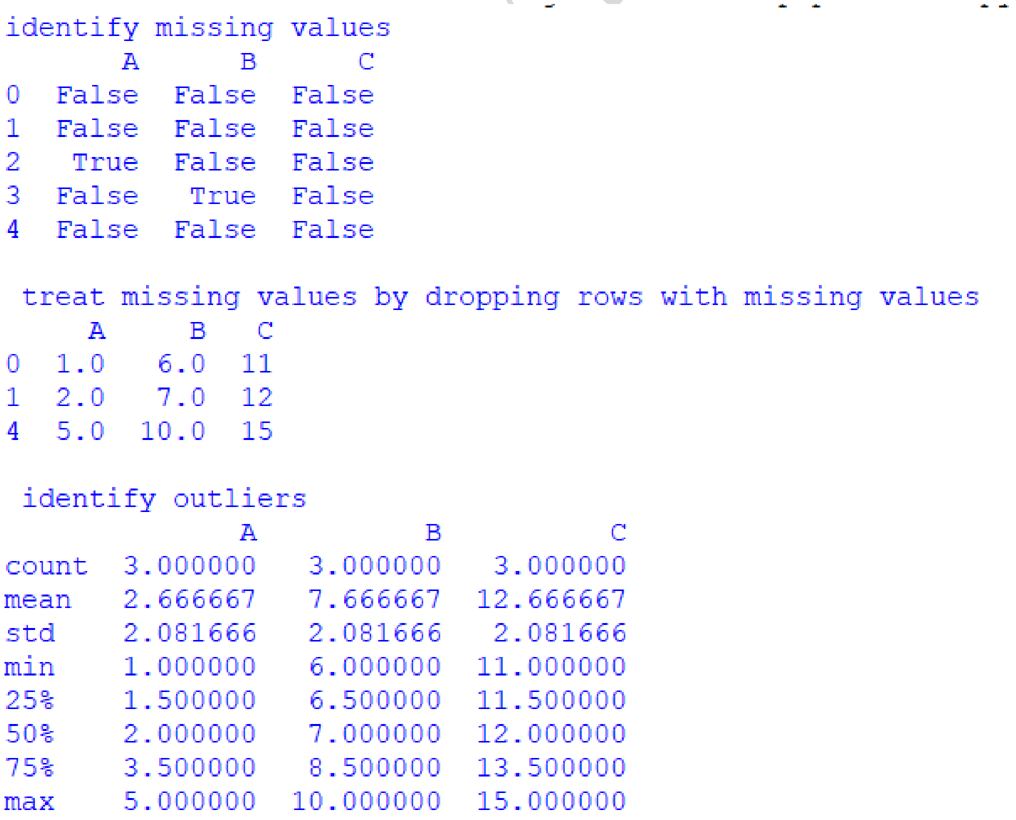
Description:

c. Recognize and Treat missing values and outliers: This step involves identifying and handling missing values and outliers in a DataFrame using various statistical methods and data visualization techniques. For example, you can use the isna() function in Pandas to identify missing values and then use statistical methods such as mean, median, or mode imputation to fill in missing values. Outliers can be identified using statistical techniques such as z-score or IQR (interquartile range) and visualized using data visualization techniques such as box plots, scatter plots, or histograms. Once identified, outliers can be treated using techniques such as winsorization (replacing extreme values with a predetermined threshold) or z-score normalization (scaling values based on their z-scores) to mitigate their impact on data analysis and modeling. Handling missing values and outliers appropriately is crucial to ensure data integrity and the reliability of analytical results.

Program:

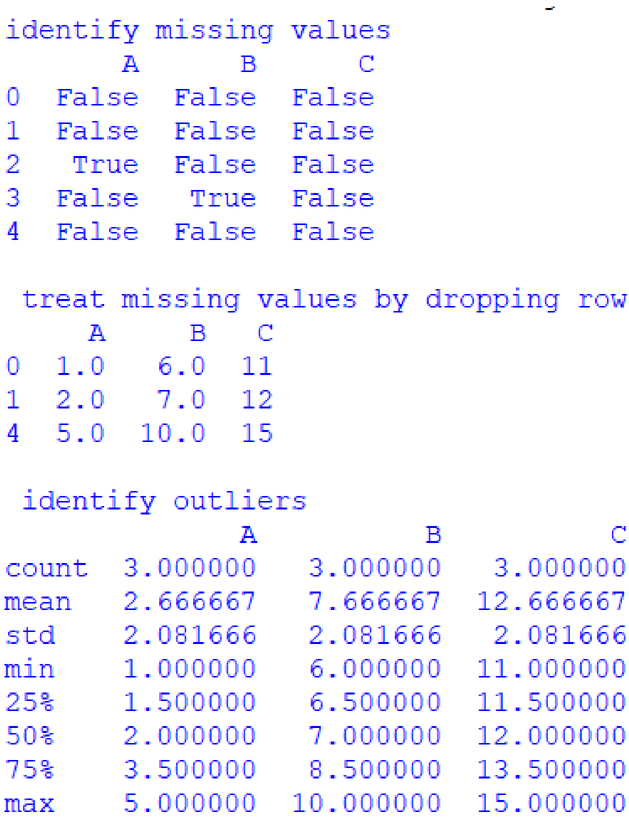
```
import pandas as pd
import numpy as np
data = {’A’: [1, 2, np.nan, 4, 5],
’B’: [6, 7, 8, np.nan, 10],
’C’: [11, 12, 13, 14, 15]}
df = pd.DataFrame(data)
print(”identify missing values”)
print(df.isnull())
print(’\n’,”treat missing values by dropping rows with missing values”)
df.dropna(inplace=True)
print(df)
print(’\n’,”identify outliers”)
print(df.describe())
```

Expected output:



```
identify missing values
       A      B      C
0  False  False  False
1  False  False  False
2   True  False  False
3  False   True  False
4  False  False  False

 treat missing values by dropping rows with missing values
     A     B   C
0  1.0   6.0  11
1  2.0   7.0  12
4  5.0  10.0  15

 identify outliers
              A          B          C
count  3.000000   3.000000   3.000000
mean   2.666667   7.666667  12.666667
std    2.081666   2.081666   2.081666
min    1.000000   6.000000  11.000000
25%    1.500000   6.500000  11.500000
50%    2.000000   7.000000  12.000000
75%    3.500000   8.500000  13.500000
max    5.000000  10.000000  15.000000
```

Observed output:



```
identify missing values
       A      B      C
0  False  False  False
1  False  False  False
2   True  False  False
3  False   True  False
4  False  False  False

 treat missing values by dropping rows with missing values
     A     B   C
0  1.0   6.0  11
1  2.0   7.0  12
4  5.0  10.0  15

 identify outliers
              A          B          C
count  3.000000   3.000000   3.000000
mean   2.666667   7.666667  12.666667
std    2.081666   2.081666   2.081666
min    1.000000   6.000000  11.000000
25%    1.500000   6.500000  11.500000
50%    2.000000   7.000000  12.000000
75%    3.500000   8.500000  13.500000
max    5.000000  10.000000  15.000000
```

9. Aggregate
a. Groupby
I. Splitting the data into groups
II. Applying a function to each group individually
III. Combining the result into a data structure
b. Pivot thable
c. Cross tab

(a)Groupby
I. Splitting the data into groups

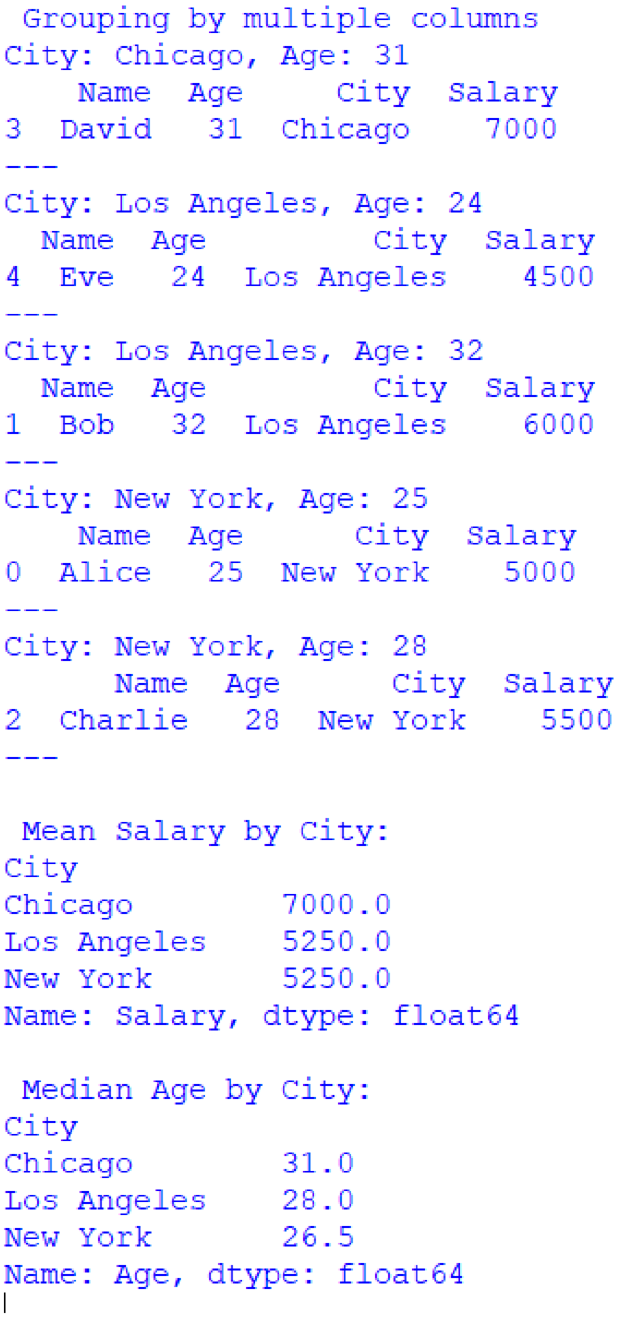
Aim: Write a python program for Splitting the data into groups using pandas

Description:
a. Groupby: Groupby is a powerful feature in Pandas that allows you to group data in a DataFrame based on one or more columns, and then apply various aggregate functions to the groups to generate summary statistics or perform data aggregation. The process of using groupby typically involves three main steps:
I. Splitting the data into groups: In this step, you specify the column(s) by which you want to group the data in the DataFrame. This creates a ”groupby” object that represents the groups.

Program:

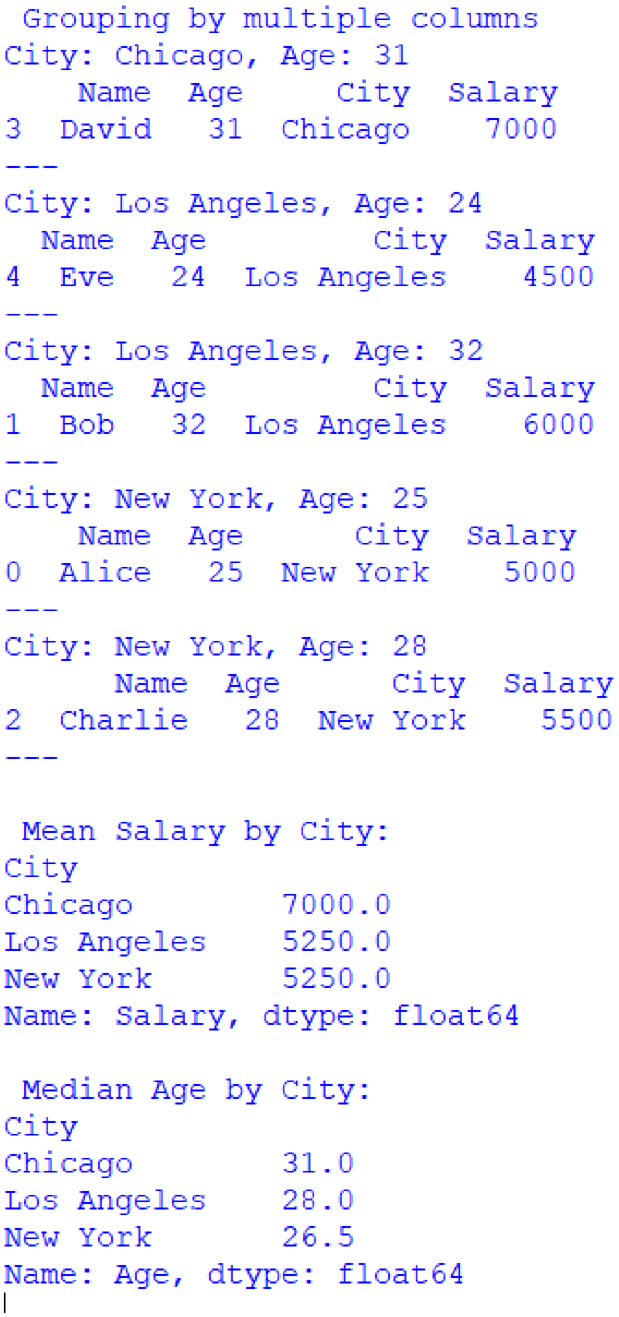
```
import pandas as pd
df = pd.DataFrame({
’Name’: [’Alice’, ’Bob’, ’Charlie’, ’David’, ’Eve’],
’Age’: [25, 32, 28, 31, 24],
’City’: [’New York’, ’Los Angeles’, ’New York’, ’Chicago’, ’Los Angeles’],
’Salary’: [5000, 6000, 5500, 7000, 4500]
})
print(’\n’,”Grouping by multiple columns”)
grouped = df.groupby([’City’, ’Age’])
for (city, age), group in grouped:
print(f’City: city, Age: age’)
print(group)
print(’—’)
grouped = df.groupby(’City’)
mean_salary = grouped[’Salary’].mean()
median_age = grouped[’Age’].median()
print(’\n’,’Mean Salary by City:’)
print(mean_salary)
print(’\n’,’Median Age by City:’)
print(median_age)
```

Expected output:



```
 Grouping by multiple columns
City: Chicago, Age: 31
    Name  Age     City  Salary
3  David   31  Chicago    7000
---
City: Los Angeles, Age: 24
  Name  Age         City  Salary
4  Eve   24  Los Angeles    4500
---
City: Los Angeles, Age: 32
  Name  Age         City  Salary
1  Bob   32  Los Angeles    6000
---
City: New York, Age: 25
    Name  Age      City  Salary
0  Alice   25  New York    5000
---
City: New York, Age: 28
      Name  Age      City  Salary
2  Charlie   28  New York    5500
---

 Mean Salary by City:
City
Chicago        7000.0
Los Angeles    5250.0
New York       5250.0
Name: Salary, dtype: float64

 Median Age by City:
City
Chicago        31.0
Los Angeles    28.0
New York       26.5
Name: Age, dtype: float64
```

Observed output:



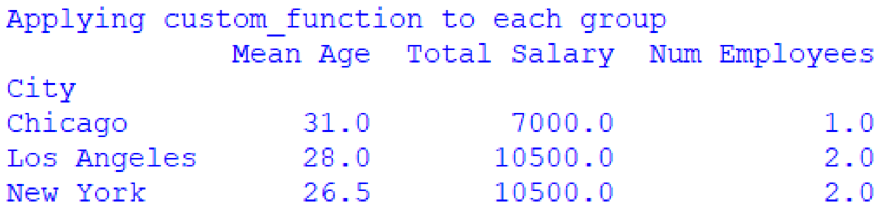
```
 Grouping by multiple columns
City: Chicago, Age: 31
    Name  Age     City  Salary
3  David   31  Chicago    7000
---
City: Los Angeles, Age: 24
  Name  Age         City  Salary
4  Eve   24  Los Angeles    4500
---
City: Los Angeles, Age: 32
  Name  Age         City  Salary
1  Bob   32  Los Angeles    6000
---
City: New York, Age: 25
    Name  Age      City  Salary
0  Alice   25  New York    5000
---
City: New York, Age: 28
      Name  Age      City  Salary
2  Charlie   28  New York    5500
---

 Mean Salary by City:
City
Chicago        7000.0
Los Angeles    5250.0
New York       5250.0
Name: Salary, dtype: float64

 Median Age by City:
City
Chicago        31.0
Los Angeles    28.0
New York       26.5
Name: Age, dtype: float64
```

II. Applying a function to each group individually

Aim: Write a python program for Applying a function to each group individually using pandas
Description:
a. Groupby: Groupby is a powerful feature in Pandas that allows you to group data in a DataFrame based on one or more columns, and then apply various aggregate functions to the groups to generate summary statistics or perform data aggregation. The process of using groupby typically involves three main steps:
II. Applying a function to each group individually: Once the data is grouped, you can apply various aggregate functions to each group individually. These functions can include basic statistical functions such as sum, mean, median, min, max, and count, as well as custom functions that you define.
Program:

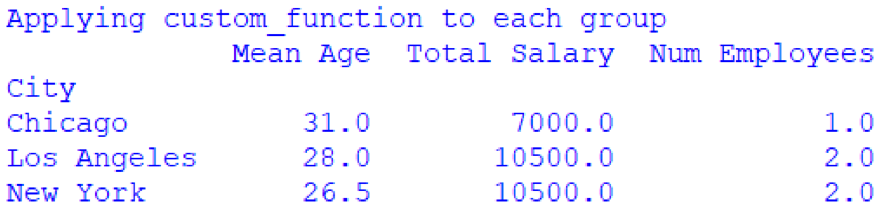
```
import pandas as pd
df = pd.DataFrame({
’Name’: [’Alice’, ’Bob’, ’Charlie’, ’David’, ’Eve’],
’Age’: [25, 32, 28, 31, 24],
’City’: [’New York’, ’Los Angeles’, ’New York’, ’Chicago’, ’Los Angeles’],
’Salary’: [5000, 6000, 5500, 7000, 4500]
})
def custom_function(group):
mean_age = group[’Age’].mean()
total_salary = group[’Salary’].sum()
num_employees = len(group)
result = {
’Mean Age’: mean_age,
’Total Salary’: total_salary,
’Num Employees’: num_employees
}
return pd.Series(result)
grouped = df.groupby(’City’)
print(”Applying custom_function to each group”)
result = grouped.apply(custom_function)
print(result)
```

Expected output:



```
Applying custom_function to each group
             Mean Age  Total Salary  Num Employees
City
Chicago          31.0        7000.0            1.0
Los Angeles      28.0       10500.0            2.0
New York         26.5       10500.0            2.0
```

Observed output:



```
Applying custom_function to each group
             Mean Age  Total Salary  Num Employees
City
Chicago          31.0        7000.0            1.0
Los Angeles      28.0       10500.0            2.0
New York         26.5       10500.0            2.0
```

III. Combining the result into a data structure

Aim: Write a python program for Combining the result into a data structure using pandas

Description:

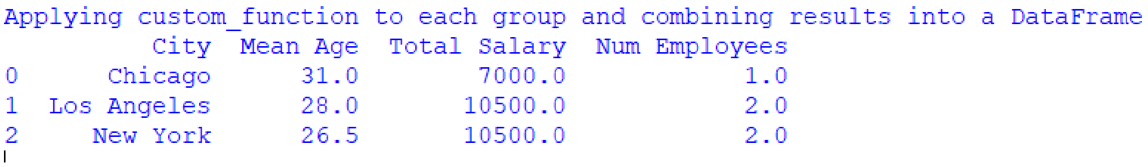
a. Groupby: Groupby is a powerful feature in Pandas that allows you to group data in a DataFrame based on one or more columns, and then apply various aggregate functions to the groups to generate summary statistics or perform data aggregation. The process of using groupby typically involves three main steps:

III. Combining the result into a data structure: After applying the aggregate functions to each group, the results are combined into a new data structure, typically a new DataFrame or a Series, where the groups are represented as index labels and the aggregated values are the corresponding data points. This new data structure provides a summary of the data for each group, allowing you to perform further analysis or generate visualizations.

Program:

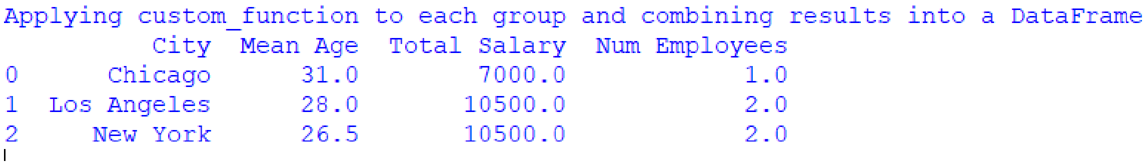
```
import pandas as pd
df = pd.DataFrame({
’Name’: [’Alice’, ’Bob’, ’Charlie’, ’David’, ’Eve’],
’Age’: [25, 32, 28, 31, 24],
’City’: [’New York’, ’Los Angeles’, ’New York’, ’Chicago’, ’Los Angeles’],
’Salary’: [5000, 6000, 5500, 7000, 4500]
})
def custom_function(group):
mean_age = group[’Age’].mean()
total_salary = group[’Salary’].sum()
num_employees = len(group)
result = {
’Mean Age’: mean_age,
’Total Salary’: total_salary,
’Num Employees’: num_employees
}
return pd.Series(result)
grouped = df.groupby(’City’)
print(”Applying custom_function to each group and combining results into a DataFrame”)
result = grouped.apply(custom_function).reset_index()
print(result)
```

Expected output:



```
Applying custom_function to each group and combining results into a DataFrame
          City  Mean Age  Total Salary  Num Employees
0      Chicago      31.0        7000.0            1.0
1  Los Angeles      28.0       10500.0            2.0
2     New York      26.5       10500.0            2.0
```

Observed output:



```
Applying custom_function to each group and combining results into a DataFrame
          City  Mean Age  Total Salary  Num Employees
0      Chicago      31.0        7000.0            1.0
1  Los Angeles      28.0       10500.0            2.0
2     New York      26.5       10500.0            2.0
```

b. Pivot thable

Aim: Write a python program for Pivot thable using pandas

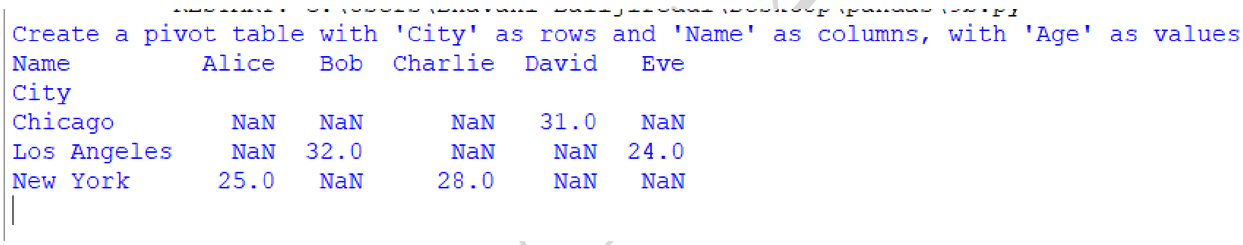
Description:

b. Pivot table: A pivot table is a feature in Pandas that allows you to transform a DataFrame by reorganizing the data and calculating summary statistics. It is similar to the concept of a pivot table in spreadsheet software like Microsoft Excel. With a pivot table, you can specify one or more columns as the index, columns, and values, and then apply various aggregate functions to calculate summary statistics for the values based on the index and columns. This can be useful for analyzing data with multiple dimensions and generating meaningful insights.

Program:

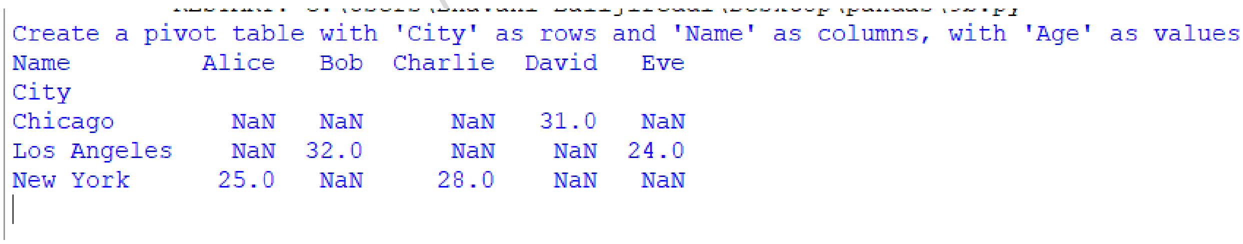
```
import pandas as pd
df = pd.DataFrame({
’Name’: [’Alice’, ’Bob’, ’Charlie’, ’David’, ’Eve’],
’Age’: [25, 32, 28, 31, 24],
’City’: [’New York’, ’Los Angeles’, ’New York’, ’Chicago’, ’Los Angeles’],
’Salary’: [5000, 6000, 5500, 7000, 4500]
})
print(”Create a pivot table with ’City’ as rows and ’Name’ as columns, with ’Age’ as values”)
pivot_table = df.pivot_table(index=’City’, columns=’Name’, values=’Age’)
print(pivot_table)
```

Expected output:



```
Create a pivot table with 'City' as rows and 'Name' as columns, with 'Age' as values
Name         Alice   Bob  Charlie  David   Eve
City
Chicago        NaN   NaN      NaN   31.0   NaN
Los Angeles    NaN  32.0      NaN    NaN  24.0
New York      25.0   NaN     28.0    NaN   NaN
```

Observed output:



```
Create a pivot table with 'City' as rows and 'Name' as columns, with 'Age' as values
Name         Alice   Bob  Charlie  David   Eve
City
Chicago        NaN   NaN      NaN   31.0   NaN
Los Angeles    NaN  32.0      NaN    NaN  24.0
New York      25.0   NaN     28.0    NaN   NaN
```

c. Cross tab

Aim: Write a python program for Cross tab using pandas

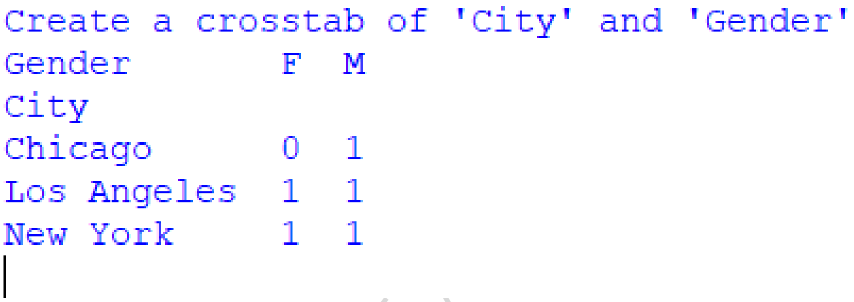
Description:

c. Cross tab: A cross tab, short for ”cross-tabulation,” is a method in Pandas that allows you to create a table of frequencies or contingency table by grouping and counting data based on two or more columns. It is commonly used to explore the relationship between two categorical variables and understand the distribution of data across different categories. The cross tab function in Pandas provides a convenient way to generate frequency tables, calculate row and column percentages, and perform other statistical calculations on categorical data. It is a useful tool for exploring and summarizing data with categorical variables in a tabular format, making it easier to identify patterns and trends in the data.

Program:

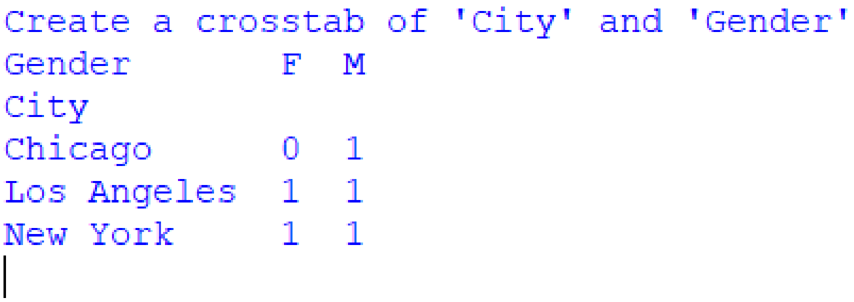
```
import pandas as pd
df = pd.DataFrame({
’Name’: [’Alice’, ’Bob’, ’Charlie’, ’David’, ’Eve’],
’Age’: [25, 32, 28, 31, 24],
’City’: [’New York’, ’Los Angeles’, ’New York’, ’Chicago’, ’Los Angeles’],
’Gender’: [’F’, ’M’, ’M’, ’M’, ’F’]
})
print(”Create a crosstab of ’City’ and ’Gender’”)
crosstab_result = pd.crosstab(df[’City’], df[’Gender’])
print(crosstab_result)
```

Expected output:



```
Create a crosstab of 'City' and 'Gender'
Gender       F  M
City
Chicago      0  1
Los Angeles  1  1
New York     1  1
```

Observed output:



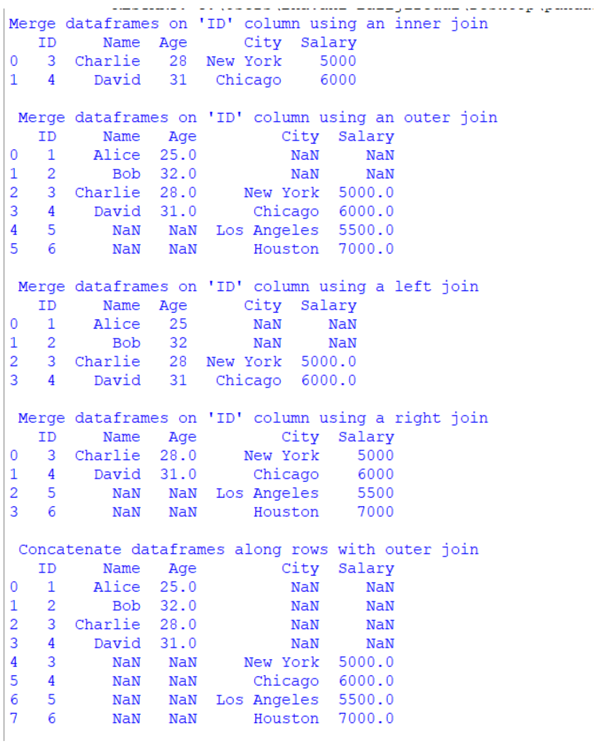
```
Create a crosstab of 'City' and 'Gender'
Gender       F  M
City
Chicago      0  1
Los Angeles  1  1
New York     1  1
```

10. Operations on Data Frames
a. Mearging/Concatenating Data Frames
b. Transpose a Data set or dataframe using Pandas
c. To sort a Pandas DataFrame
d. Remove duplicate values of a variable in a Pandas Dataframe

(a)Mearging/Concatenating Data Frames
Aim: Write a python program for Mearging/Concatenating Data Frames using pandas
Description:
a. Merging/Concatenating Data Frames: Merging or concatenating data frames in Pandas is the process of combining two or more data frames into a single data frame. This can be done based on common columns or keys, similar to a SQL join operation. Merging allows you to combine data from different data frames into a single data frame, which is useful for data integration, data consolidation, and data analysis tasks.
Program:

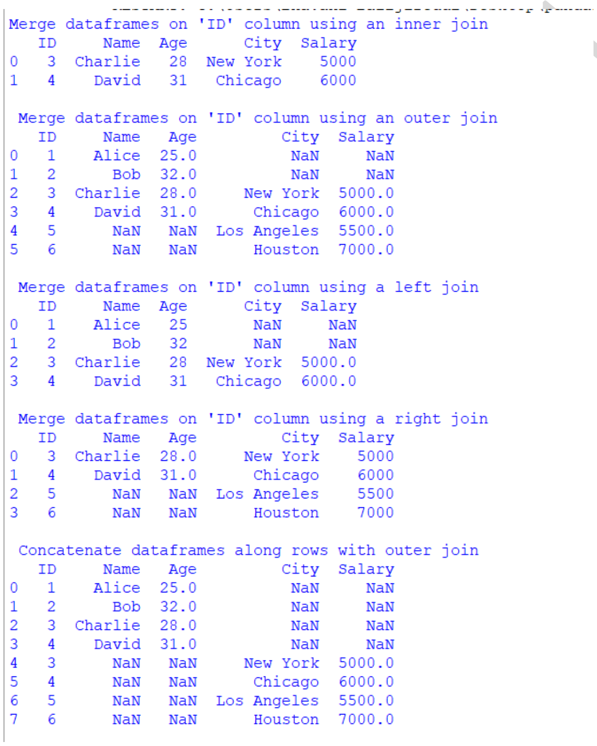
```
import pandas as pd
df1 = pd.DataFrame({’ID’: [1, 2, 3, 4],
’Name’: [’Alice’, ’Bob’, ’Charlie’, ’David’],
’Age’: [25, 32, 28, 31]})
df2 = pd.DataFrame({’ID’: [3, 4, 5, 6],
’City’: [’New York’, ’Chicago’, ’Los Angeles’, ’Houston’],
’Salary’: [5000, 6000, 5500, 7000]})
print(”Merge dataframes on ’ID’ column using an inner join”)
merged_df = pd.merge(df1, df2, on=’ID’, how=’inner’)
print(merged_df)
print(’\n’,”Merge dataframes on ’ID’ column using an outer join”)
merged_df = pd.merge(df1, df2, on=’ID’, how=’outer’)
print(merged_df)
print(’\n’,”Merge dataframes on ’ID’ column using a left join”)
merged_df = pd.merge(df1, df2, on=’ID’, how=’left’)
print(merged_df)
print(’\n’,”Merge dataframes on ’ID’ column using a right join”)
merged_df = pd.merge(df1, df2, on=’ID’, how=’right’)
print(merged_df)
print(’\n’,”Concatenate dataframes along rows with outer join”)
concatenated_df = pd.concat([df1, df2], ignore_index=True, sort=False)
print(concatenated_df)
```

Expected output:



```
Merge dataframes on 'ID' column using an inner join
   ID     Name  Age      City  Salary
0   3  Charlie   28  New York    5000
1   4    David   31   Chicago    6000

 Merge dataframes on 'ID' column using an outer join
   ID     Name   Age         City  Salary
0   1    Alice  25.0          NaN     NaN
1   2      Bob  32.0          NaN     NaN
2   3  Charlie  28.0     New York  5000.0
3   4    David  31.0      Chicago  6000.0
4   5      NaN   NaN  Los Angeles  5500.0
5   6      NaN   NaN      Houston  7000.0

 Merge dataframes on 'ID' column using a left join
   ID     Name  Age      City  Salary
0   1    Alice   25       NaN     NaN
1   2      Bob   32       NaN     NaN
2   3  Charlie   28  New York  5000.0
3   4    David   31   Chicago  6000.0

 Merge dataframes on 'ID' column using a right join
   ID     Name   Age         City  Salary
0   3  Charlie  28.0     New York    5000
1   4    David  31.0      Chicago    6000
2   5      NaN   NaN  Los Angeles    5500
3   6      NaN   NaN      Houston    7000

 Concatenate dataframes along rows with outer join
   ID     Name   Age         City  Salary
0   1    Alice  25.0          NaN     NaN
1   2      Bob  32.0          NaN     NaN
2   3  Charlie  28.0          NaN     NaN
3   4    David  31.0          NaN     NaN
4   3      NaN   NaN     New York  5000.0
5   4      NaN   NaN      Chicago  6000.0
6   5      NaN   NaN  Los Angeles  5500.0
7   6      NaN   NaN      Houston  7000.0
```

Observed output:



```
Merge dataframes on 'ID' column using an inner join
   ID     Name  Age      City  Salary
0   3  Charlie   28  New York    5000
1   4    David   31   Chicago    6000

 Merge dataframes on 'ID' column using an outer join
   ID     Name   Age         City  Salary
0   1    Alice  25.0          NaN     NaN
1   2      Bob  32.0          NaN     NaN
2   3  Charlie  28.0     New York  5000.0
3   4    David  31.0      Chicago  6000.0
4   5      NaN   NaN  Los Angeles  5500.0
5   6      NaN   NaN      Houston  7000.0

 Merge dataframes on 'ID' column using a left join
   ID     Name  Age      City  Salary
0   1    Alice   25       NaN     NaN
1   2      Bob   32       NaN     NaN
2   3  Charlie   28  New York  5000.0
3   4    David   31   Chicago  6000.0

 Merge dataframes on 'ID' column using a right join
   ID     Name   Age         City  Salary
0   3  Charlie  28.0     New York    5000
1   4    David  31.0      Chicago    6000
2   5      NaN   NaN  Los Angeles    5500
3   6      NaN   NaN      Houston    7000

 Concatenate dataframes along rows with outer join
   ID     Name   Age         City  Salary
0   1    Alice  25.0          NaN     NaN
1   2      Bob  32.0          NaN     NaN
2   3  Charlie  28.0          NaN     NaN
3   4    David  31.0          NaN     NaN
4   3      NaN   NaN     New York  5000.0
5   4      NaN   NaN      Chicago  6000.0
6   5      NaN   NaN  Los Angeles  5500.0
7   6      NaN   NaN      Houston  7000.0
```

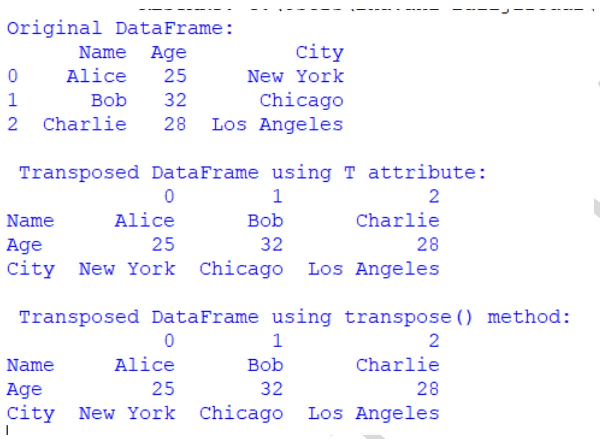
(b)Transpose a Data set or dataframe using Pandas
Aim: Write a python program for Transpose a Data set or dataframe using Pandas
Description:

b. Transpose a Data Set or DataFrame using Pandas: Transposing a data set or data frame in Pandas involves swapping the rows and columns, effectively rotating the data set or data frame by 90 degrees. This can be done using the transpose() function or the .T attribute in Pandas, and it is useful for reshaping data or changing the orientation of data for analysis or visualization purposes.

Program:

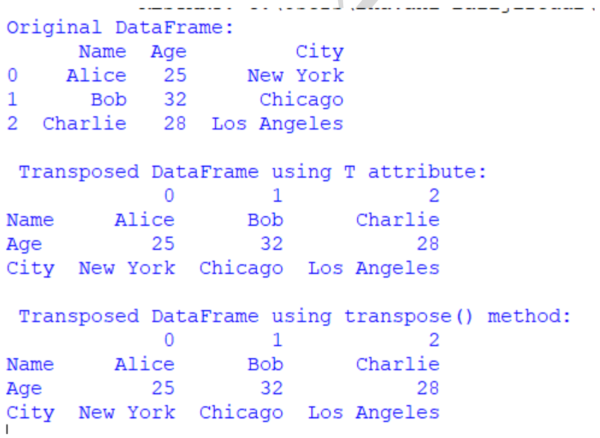
```
import pandas as pd
df = pd.DataFrame({’Name’: [’Alice’, ’Bob’, ’Charlie’],
’Age’: [25, 32, 28],
’City’: [’New York’, ’Chicago’, ’Los Angeles’]})
print(”Original DataFrame:”)
print(df)
transposed_df = df.T
print(’\n’,”Transposed DataFrame using T attribute:”)
print(transposed_df)
transposed_df2 = df.transpose()
print(’\n’,”Transposed DataFrame using transpose() method:”)
print(transposed_df2)
```

Expected output:



```
Original DataFrame:
      Name  Age         City
0    Alice   25     New York
1      Bob   32      Chicago
2  Charlie   28  Los Angeles

 Transposed DataFrame using T attribute:
             0        1            2
Name     Alice      Bob      Charlie
Age         25       32           28
City  New York  Chicago  Los Angeles

 Transposed DataFrame using transpose() method:
             0        1            2
Name     Alice      Bob      Charlie
Age         25       32           28
City  New York  Chicago  Los Angeles
```

Observed output:



```
Original DataFrame:
      Name  Age         City
0    Alice   25     New York
1      Bob   32      Chicago
2  Charlie   28  Los Angeles

 Transposed DataFrame using T attribute:
             0        1            2
Name     Alice      Bob      Charlie
Age         25       32           28
City  New York  Chicago  Los Angeles

 Transposed DataFrame using transpose() method:
             0        1            2
Name     Alice      Bob      Charlie
Age         25       32           28
City  New York  Chicago  Los Angeles
```

(c)To sort a Pandas DataFrame

Aim: Write a python program for To sort a Pandas DataFrame using pandas

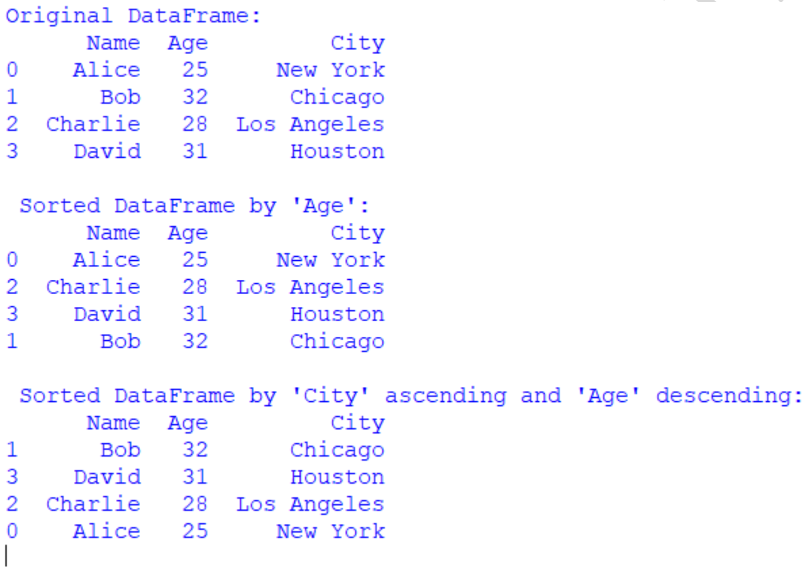
Description:

c. Sorting a Pandas DataFrame: Sorting a Pandas DataFrame involves arranging the rows or columns of the data frame in a specific order based on the values in one or more columns. This can be done using the sort_values() function in Pandas, which allows you to sort the data frame by one or more columns in ascending or descending order. Sorting data frames is useful for organizing data, identifying patterns or trends, and preparing data for analysis or visualization.

Program:

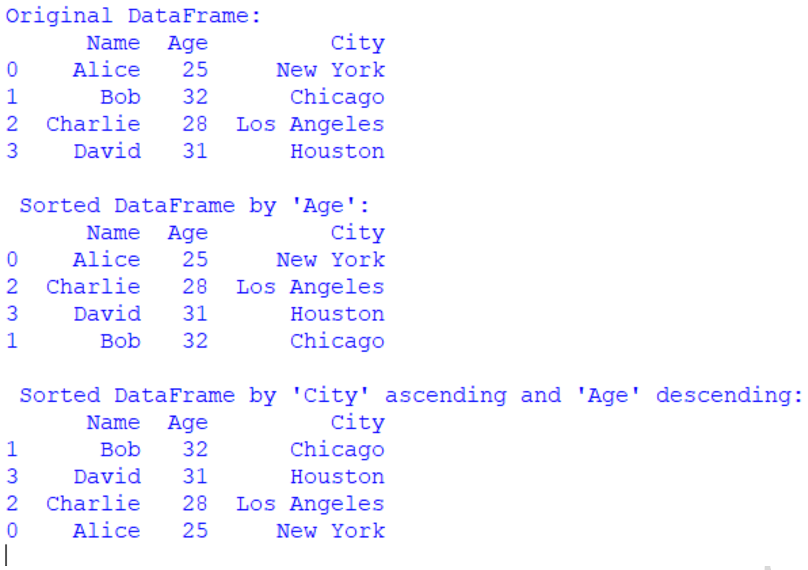
```
import pandas as pd
df = pd.DataFrame({’Name’: [’Alice’, ’Bob’, ’Charlie’, ’David’],
’Age’: [25, 32, 28, 31],
’City’: [’New York’, ’Chicago’, ’Los Angeles’, ’Houston’]})
print(”Original DataFrame:”)
print(df)
sorted_df = df.sort_values(by=’Age’)
print(’\n’,”Sorted DataFrame by ’Age’:”)
print(sorted_df)
sorted_df2 = df.sort_values(by=[’City’, ’Age’], ascending=[True, False])
print(’\n’,”Sorted DataFrame by ’City’ ascending and ’Age’ descending:”)
print(sorted_df2)
```

Expected output:



```
Original DataFrame:
      Name  Age         City
0    Alice   25     New York
1      Bob   32      Chicago
2  Charlie   28  Los Angeles
3    David   31      Houston

 Sorted DataFrame by 'Age':
      Name  Age         City
0    Alice   25     New York
2  Charlie   28  Los Angeles
3    David   31      Houston
1      Bob   32      Chicago

 Sorted DataFrame by 'City' ascending and 'Age' descending:
      Name  Age         City
1      Bob   32      Chicago
3    David   31      Houston
2  Charlie   28  Los Angeles
0    Alice   25     New York
```

Observed output:



```
Original DataFrame:
      Name  Age         City
0    Alice   25     New York
1      Bob   32      Chicago
2  Charlie   28  Los Angeles
3    David   31      Houston

 Sorted DataFrame by 'Age':
      Name  Age         City
0    Alice   25     New York
2  Charlie   28  Los Angeles
3    David   31      Houston
1      Bob   32      Chicago

 Sorted DataFrame by 'City' ascending and 'Age' descending:
      Name  Age         City
1      Bob   32      Chicago
3    David   31      Houston
2  Charlie   28  Los Angeles
0    Alice   25     New York
```

(d)Remove duplicate values of a variable in a Pandas Dataframe

Aim: Write a python program for Remove duplicate values of a variable in a Pandas Dataframe using pandas

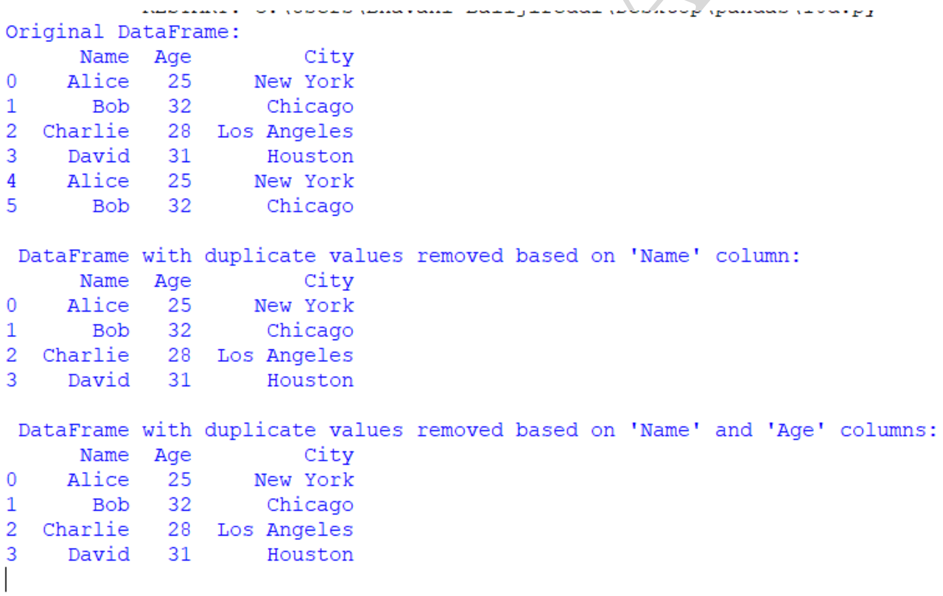
Description:

d. Removing Duplicate Values of a Variable in a Pandas DataFrame: Removing duplicate values of a variable in a Pandas DataFrame involves identifying and eliminating rows with identical values in one or more columns. This can be done using the drop_duplicates() function in Pandas, which allows you to identify and remove duplicate values based on specific columns or the entire data frame. Removing duplicate values is important for data cleaning, data quality assurance, and ensuring accurate and reliable data analysis results.

Program:

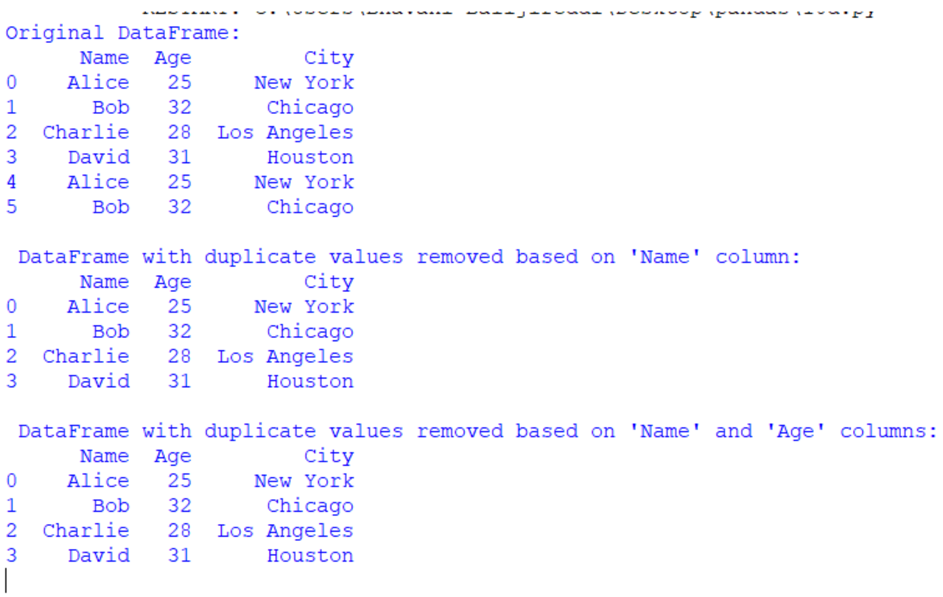
```
import pandas as pd
df = pd.DataFrame({
’Name’: [’Alice’, ’Bob’, ’Charlie’, ’David’, ’Alice’, ’Bob’],
’Age’: [25, 32, 28, 31, 25, 32],
’City’: [’New York’, ’Chicago’, ’Los Angeles’, ’Houston’, ’New York’, ’Chicago’]})
print(”Original DataFrame:”)
print(df)
df_no_duplicates = df.drop_duplicates(subset=’Name’)
print(’\n’,”DataFrame with duplicate values removed based on ’Name’ column:”)
print(df_no_duplicates)
df_no_duplicates2 = df.drop_duplicates(subset=[’Name’, ’Age’])
print(’\n’,”DataFrame with duplicate values removed based on ’Name’ and ’Age’ columns:”)
print(df_no_duplicates2)
```

Expected output:



```
Original DataFrame:
      Name  Age         City
0    Alice   25     New York
1      Bob   32      Chicago
2  Charlie   28  Los Angeles
3    David   31      Houston
4    Alice   25     New York
5      Bob   32      Chicago

 DataFrame with duplicate values removed based on 'Name' column:
      Name  Age         City
0    Alice   25     New York
1      Bob   32      Chicago
2  Charlie   28  Los Angeles
3    David   31      Houston

 DataFrame with duplicate values removed based on 'Name' and 'Age' columns:
      Name  Age         City
0    Alice   25     New York
1      Bob   32      Chicago
2  Charlie   28  Los Angeles
3    David   31      Houston
```

Observed output:



```
Original DataFrame:
      Name  Age         City
0    Alice   25     New York
1      Bob   32      Chicago
2  Charlie   28  Los Angeles
3    David   31      Houston
4    Alice   25     New York
5      Bob   32      Chicago

 DataFrame with duplicate values removed based on 'Name' column:
      Name  Age         City
0    Alice   25     New York
1      Bob   32      Chicago
2  Charlie   28  Los Angeles
3    David   31      Houston

 DataFrame with duplicate values removed based on 'Name' and 'Age' columns:
      Name  Age         City
0    Alice   25     New York
1      Bob   32      Chicago
2  Charlie   28  Los Angeles
3    David   31      Houston
```