DATA EXPLORATION

May 1, 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 program to import pandas using python

DESCRIPTION :
Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks.Import pandas is a statement in Python that allows you to import the pandas library into your Python code. Pandas is an open-source data manipulation and analysis library that provides data structures for efficiently storing and manipulating large datasets.

PROGRAM :

```
import pandas
data = {
'CHN': 'COUNTRY': 'China', 'POP': 1_398.72, 'AREA': 9_596.96, 'GDP': 12_234.78, 'CONT':
'Asia',
'IND': {'COUNTRY': 'India', 'POP': 1_351.16, 'AREA': 3_287.26, 'GDP': 2_575.67, 'CONT':
'Asia', 'IND_DAY': '1947-08-15'},
'USA': {'COUNTRY': 'US', 'POP': 329.74, 'AREA': 9_833.52, 'GDP': 19_485.39, 'CONT':
'N.America', 'IND_DAY': '1776-07-04'},}
columns = ('COUNTRY', 'POP', 'AREA', 'GDP', 'CONT', 'IND_DAY')
import pandas as pd
df = pd.DataFrame(data=data).T
df
```

EXPECTED OUTPUT :

	COUNTRY	POP	AREA	GDP	CONT	IND_DAY
CHN	China	1398.72	9596.96	12234.78	Asia	NaN
IND	India	1351.16	3287.26	2575.67	Asia	1947-08-15
USA	US	329.74	9833.52	19485.39	N.America	1776-07-04

OBSERVED OUTPUT :

	COUNTRY	POP	AREA	GDP	CONT	IND_DAY
CHN	China	1398.72	9596.96	12234.78	Asia	NaN
IND	India	1351.16	3287.26	2575.67	Asia	1947-08-15
USA	US	329.74	9833.52	19485.39	N.America	1776-07-04

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

AIM : Write a python program for loading the data various formats (.XLS, .TXT, .CSV, JSON) using Pandas

DESCRIPTION :

CSV (comma-separated values) files are a popular way of storing and sharing data. To read a CSV file using pandas, you can use the read_csv() function.Excel files are another common way of storing data. To read an Excel file using pandas, you can use the read_excel() function.

Text files are a generic way of storing data, and can be stored in various formats such as TSV, space-delimited, etc. To read a text file using pandas, you can use the read_table() function.JSON (JavaScript Object Notation) files are commonly used for storing and exchanging data on the web. To read a JSON file using pandas, you can use the read_json() function.

PROGRAM :

CSV :

```
import pandas as pd
df = pd.read_csv('data.csv')
df
```

EXPECTED OUTPUT :

	COUNTRY	POP	AREA	GDP	CONT	IND_DAY
CHN	China	1398.72	9596.96	12234.78	Asia	NaN
IND	India	1351.16	3287.26	2575.67	Asia	1947-08-15
USA	US	329.74	9833.52	19485.39	N.America	1776-07-04

OBSERVED OUTPUT :

	COUNTRY	POP	AREA	GDP	CONT	IND_DAY
CHN	China	1398.72	9596.96	12234.78	Asia	NaN
IND	India	1351.16	3287.26	2575.67	Asia	1947-08-15
USA	US	329.74	9833.52	19485.39	N.America	1776-07-04

XLS :

```
import pandas as pd
df = pd.read_excel('data.xlsx')
df
```

EXPECTED OUTPUT :

	COUNTRY	POP	AREA	GDP	CONT	IND_DAY
CHN	China	1398.72	9596.96	12234.78	Asia	NaN
IND	India	1351.16	3287.26	2575.67	Asia	1947-08-15
USA	US	329.74	9833.52	19485.39	N.America	1776-07-04

OBSERVED OUTPUT :

	COUNTRY	POP	AREA	GDP	CONT	IND_DAY
CHN	China	1398.72	9596.96	12234.78	Asia	NaN
IND	India	1351.16	3287.26	2575.67	Asia	1947-08-15
USA	US	329.74	9833.52	19485.39	N.America	1776-07-04

TXT :

```
import pandas as pd
df = pd.read_csv('data.txt')
df
```

EXPECTED OUTPUT :

	COUNTRY	POP	AREA	GDP	CONT	IND_DAY
CHN	China	1398.72	9596.96	12234.78	Asia	NaN
IND	India	1351.16	3287.26	2575.67	Asia	1947-08-15
USA	US	329.74	9833.52	19485.39	N.America	1776-07-04

OBSERVED OUTPUT :

	COUNTRY	POP	AREA	GDP	CONT	IND_DAY
CHN	China	1398.72	9596.96	12234.78	Asia	NaN
IND	India	1351.16	3287.26	2575.67	Asia	1947-08-15
USA	US	329.74	9833.52	19485.39	N.America	1776-07-04

JSON :

```
import pandas as pd
df = pd.read_json('data.txt')
df
```

EXPECTED OUTPUT :

	CHN	IND	USA
COUNTRY	China	India	US
POP	1398.72	1351.16	329.74
AREA	9596.96	3287.26	9833.52
GDP	12234.78	2575.67	19485.39
CONT	Asia	Asia	N.America
IND_DAY	NaN	1947-08-15	1776-07-04

OBSERVED OUTPUT :

	CHN	IND	USA
COUNTRY	China	India	US
POP	1398.72	1351.16	329.74
AREA	9596.96	3287.26	9833.52
GDP	12234.78	2575.67	19485.39
CONT	Asia	Asia	N.America
IND_DAY	NaN	1947-08-15	1776-07-04

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

AIM : Write a python program to Describe Data, Modify Data, Grouping Data, Filtering Data using pandas

DESCRIPTION :

Pandas provides several functions to describe the statistical properties of your data. For example, you can use describe() to get a summary of the central tendency, dispersion, and shape of your data.Pandas provides several functions to modify your data. For example, you can use fillna() to fill missing values with a specific value or method.

Pandas provides several functions to group your data by one or more columns. For example, you can use groupby() to group your data by a specific column and then apply a function to each group.Pandas provides several functions to filter your data based on specific conditions. For example, you can use loc[] to filter your data by specific values or conditions.

PROGRAM :

```
import pandas as pd
df = pd.DataFrame({
'Name': ['John', 'Mary', 'Peter', 'Ann', 'Tom'],
'Age': [25, 32, 18, 40, 28],
'Gender': ['M', 'F', 'M', 'F', 'M'],
'City': ['New York', 'Los Angeles', 'San Francisco', 'Chicago', 'Miami'],
'Salary': [50000, 75000, 40000, 90000, 60000]})
print("Describing the data:")
print(df.describe())
print("Modifying the data:")
df.loc[df['Name'] == 'Peter', 'Salary'] = 45000
print(df)
print("Grouping the data:")
grouped = df.groupby(['Gender'])
for group_name, group in grouped:
print(f'Group Name: group_name")
print(group)
print("Filtering the data:")
filtered = df[(df['Age'] > 25) & (df['Salary'] > 55000)]
print(filtered)
```

EXPECTED OUTPUT :

```
Describing the data:
             Age        Salary
count   5.000000      5.000000
mean   28.600000  63000.000000
std     8.173127  19874.606914
min    18.000000  40000.000000
25%    25.000000  50000.000000
50%    28.000000  60000.000000
75%    32.000000  75000.000000
max    40.000000  90000.000000

Modifying the data:
    Name  Age Gender           City  Salary
0   John   25      M       New York   50000
1   Mary   32      F    Los Angeles   75000
2  Peter   18      M  San Francisco   45000
3    Ann   40      F        Chicago   90000
4    Tom   28      M          Miami   60000

Grouping the data:
```

```
Group Name: F
   Name  Age Gender         City  Salary
1  Mary   32      F  Los Angeles   75000
3   Ann   40      F      Chicago   90000

Group Name: M
    Name  Age Gender           City  Salary
0   John   25      M       New York   50000
2  Peter   18      M  San Francisco   45000
4    Tom   28      M          Miami   60000

Filtering the data:
   Name  Age Gender         City  Salary
1  Mary   32      F  Los Angeles   75000
3   Ann   40      F      Chicago   90000
4   Tom   28      M        Miami   60000
```

OBSERVED OUTPUT :

```
Describing the data:
              Age        Salary
count    5.000000      5.000000
mean    28.600000  63000.000000
std      8.173127  19874.606914
min     18.000000  40000.000000
25%     25.000000  50000.000000
50%     28.000000  60000.000000
75%     32.000000  75000.000000
max     40.000000  90000.000000

Modifying the data:
    Name  Age Gender           City  Salary
0   John   25      M       New York   50000
1   Mary   32      F    Los Angeles   75000
2  Peter   18      M  San Francisco   45000
3    Ann   40      F        Chicago   90000
4    Tom   28      M          Miami   60000

Grouping the data:
```

```
Group Name: F
   Name  Age Gender         City  Salary
1  Mary   32      F  Los Angeles   75000
3   Ann   40      F      Chicago   90000

Group Name: M
    Name  Age Gender           City  Salary
0   John   25      M       New York   50000
2  Peter   18      M  San Francisco   45000
4    Tom   28      M          Miami   60000

Filtering the data:
   Name  Age Gender         City  Salary
1  Mary   32      F  Los Angeles   75000
3   Ann   40      F      Chicago   90000
4   Tom   28      M        Miami   60000
```

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

AIM : Write a python program to converting a variable to a different data type back to a CSV,JSON, or SQL using pandas

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('Original DataFrame:', df)
print('DataFrame from JSON:', df_from_json)
print('DataFrame from CSV:', df_from_csv)
print('DataFrame from SQL:', df_from_sql)

EXPECTED OUTPUT :

```
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 :

```
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 : Write a program for Reading a CSV File using python

DESCRIPTION :
Reading from a CSV file is done using the reader object. The CSV file is opened as a text file with Python's built-in open() function, which returns a file object. This is then passed to the reader , which does the heavy lifting.

PROGRAM :

```
import pandas as pd
data = pd.read_csv('data.csv')
print(data)
```

EXPECTED OUTPUT :

	Unnamed: 0	COUNTRY	POP	AREA	GDP	CONT	IND_DAY
0	CHN	China	1398.72	9596.96	12234.78	Asia	NaN
1	IND	India	1351.16	3287.26	2575.67	Asia	1947-08-15
2	USA	US	329.74	9833.52	19485.39	N.America	1776-07-04

OBSERVED OUTPUT :

	Unnamed: 0	COUNTRY	POP	AREA	GDP	CONT	IND_DAY
0	CHN	China	1398.72	9596.96	12234.78	Asia	NaN
1	IND	India	1351.16	3287.26	2575.67	Asia	1947-08-15
2	USA	US	329.74	9833.52	19485.39	N.America	1776-07-04

(b) Writing content of data frames to CSV File

AIM : Write a program for Writing content of data frames to CSV File using python

DESCRIPTION :
Which pandas can be used in a Python script to write the contents of a Dataframe to a csv file.
Pandas DataFrame to_csv() function exports the DataFrame to CSV format. If a file argument is provided, the output will be the CSV file. Otherwise, the return value is a CSV format like string.

PROGRAM :

```
import pandas as pd
data = {'name': ['Alice', 'Bob', 'Charlie'], 'age': [25, 30, 35]}
df = pd.DataFrame(data)
df.to_csv('sample.csv')
print(df)
```

EXPECTED OUTPUT :

```
      name  age
0    Alice   25
1      Bob   30
2  Charlie   35
```

OBSERVED OUTPUT :

```
      name  age
0    Alice   25
1      Bob   30
2  Charlie   35
```

(c) Reading an Excel File

AIM : Write a program Reading an Excel File using python

DESCRIPTION : In Python, we can work with the data in the excel sheet with the help of the pandas module. There is a function called the pandas read_excel function for reading the excel file. There are lots of parameters for this function, like "io", "sheet_name", "dtype", etc., for reading the data in different ways. We can also get a specific part of the data using pandas read_excel function parameters.

PROGRAM :

```
import pandas as pd
data = pd.read_excel('data.xlsx')
print(data)
```

EXPECTED OUTPUT :

```
  Unnamed: 0 COUNTRY      POP     AREA       GDP       CONT     IND_DAY
0        CHN   China  1398.72  9596.96  12234.78       Asia         NaN
1        IND   India  1351.16  3287.26   2575.67       Asia  1947-08-15
2        USA      US   329.74  9833.52  19485.39  N.America  1776-07-04
```

OBSERVED OUTPUT :

```
  Unnamed: 0 COUNTRY      POP     AREA       GDP       CONT     IND_DAY
0        CHN   China  1398.72  9596.96  12234.78       Asia         NaN
1        IND   India  1351.16  3287.26   2575.67       Asia  1947-08-15
2        USA      US   329.74  9833.52  19485.39  N.America  1776-07-04
```

(d) Writing content of data frames to Excel File

AIM : Write a program for Writing content of data frames to Excel File using python

DESCRIPTION :
The Writing content of data frames to an Excel file is a common operation in data analysis and data science. It involves exporting the contents of a data frame, which is a data structure in Python that organizes data in rows and columns, to an Excel file format.To write the content of a data frame to an Excel file in Python, you can use the pandas library.

PROGRAM :

```
import pandas as pd
data = {'name': ['Alice', 'Bob', 'Charlie'], 'age': [25, 30, 35]}
df = pd.DataFrame(data)
df.to_excel('sample.xlsx')
print(df)
```

EXPECTED OUTPUT :

```
      name  age
0    Alice   25
1      Bob   30
2  Charlie   35
```

OBSERVED OUTPUT :

```
      name  age
0    Alice   25
1      Bob   30
2  Charlie   35
```

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 : Write a program for Viewing your data using python

DESCRIPTION :
The head() function is used for Viewing data in pandas from starting rows and tail() is used to view the data from the bottom, by default it views the 5 rows. Using describe() we can get the description of the data, like mean, standard deviation, percentage of the same data, etc..

PROGRAM :

```
import pandas as pd
data = pd.read_csv('data.csv')
print(data)
```

EXPECTED OUTPUT :

	Unnamed: 0	COUNTRY	POP	AREA	GDP	CONT	IND_DAY
0	CHN	China	1398.72	9596.96	12234.78	Asia	NaN
1	IND	India	1351.16	3287.26	2575.67	Asia	1947-08-15
2	USA	US	329.74	9833.52	19485.39	N.America	1776-07-04

OBSERVED OUTPUT :

	Unnamed: 0	COUNTRY	POP	AREA	GDP	CONT	IND_DAY
0	CHN	China	1398.72	9596.96	12234.78	Asia	NaN
1	IND	India	1351.16	3287.26	2575.67	Asia	1947-08-15
2	USA	US	329.74	9833.52	19485.39	N.America	1776-07-04

(b) Data Set Description

AIM : Write a program for Data Set Description using python

DESCRIPTION :
Pandas DataFrame describe() Method.The describe() method returns description of the data in the DataFrame. If the DataFrame contains numerical data, the description contains these information for each column: count - The number of not-empty values. mean - The average (mean) value.

PROGRAM :

```
import pandas as pd
data = pd.read_csv('data.csv')
description = data.describe()
print(description)
```

EXPECTED OUTPUT :

	Age	Salary
count	9.000000	9.000000
mean	30.444444	62222.222222
std	8.589399	16791.201400
min	18.000000	40000.000000
25%	25.000000	50000.000000
50%	29.000000	60000.000000
75%	35.000000	75000.000000
max	45.000000	90000.000000

OBSERVED OUTPUT :

	Age	Salary
count	9.000000	9.000000
mean	30.444444	62222.222222
std	8.589399	16791.201400
min	18.000000	40000.000000
25%	25.000000	50000.000000
50%	29.000000	60000.000000
75%	35.000000	75000.000000
max	45.000000	90000.000000

(c) Describe as category

AIM : Write a program for Describe as category using python

DESCRIPTION :
In python to describe dataset as category Categorical variables can take on only a limited, and usually fixed number of possible values. Besides the fixed length, categorical data might have an order but cannot perform numerical operation. Categorical are a Pandas data type.

PROGRAM :

```
import pandas as pd
data = pd.read_csv('data.csv')
cat_var = 'City'
print("Value counts for", cat_var, ":",
data[cat_var].value_counts())
```

EXPECTED OUTPUT :

```
Value counts for City :
 New York         1
Los Angeles       1
San Francisco     1
Chicago           1
Miami             1
Houston           1
Boston            1
Denver            1
Seattle           1
Name: City, dtype: int64
```

OBSERVED OUTPUT :

```
Value counts for City :
 New York         1
Los Angeles       1
San Francisco     1
Chicago           1
Miami             1
Houston           1
Boston            1
Denver            1
Seattle           1
Name: City, dtype: int64
```

(d) Handling duplicates

AIM : Write a program for Handling duplicates using python

DESCRIPTION :
You can identify such duplicate rows in a Pandas dataframe by calling the duplicated function. The duplicated function returns a Boolean series with value True indicating a duplicate row. By default the first row in a duplicated set is marked as False and all others marked as True.

PROGRAM :

```
import pandas as pd
data = pd.read_csv('data.csv')
print("Number of duplicate rows:", data.duplicated().sum())

import pandas as pd
data = pd.read_csv('data.csv')
data.drop_duplicates(inplace=True)
print("Number of rows after removing duplicates:", len(data))
```

EXPECTED OUTPUT :

```
Number of duplicate rows: 0
Number of rows after removing duplicates: 9
```

OBSERVED OUTPUT :

```
Number of duplicate rows: 0
Number of rows after removing duplicates: 9
```

(e) Number of observations Per Category

AIM : Write a program for Number of observations Per Category using python

DESCRIPTION :

In Python Pandas, "number of observations per category" refers to the count of data points (observations) that belong to each category or group in a categorical variable. This can be calculated using the "groupby" method in Pandas, which groups the data by the categories in the categorical variable and then applies a function (such as "count") to calculate the number of observations in each group.

PROGRAM :

```
import pandas as pd
data = pd.read_csv('data.csv')
cat_counts = data[cat_var].value_counts()
print("Counts per category for", cat_var, ":", cat_counts)
```

EXPECTED OUTPUT :

```
Counts per category for City :
 New York         1
Los Angeles       1
San Francisco     1
Chicago           1
Miami             1
Houston           1
Boston            1
Denver            1
Seattle           1
Name: City, dtype: int64
```

OBSERVED OUTPUT :

```
Counts per category for City :
 New York         1
Los Angeles       1
San Francisco     1
Chicago           1
Miami             1
Houston           1
Boston            1
Denver            1
Seattle           1
Name: City, dtype: int64
```

(f) Column cleanup

AIM : Write a program for Column cleanup using python

DESCRIPTION :
In Python Pandas, "column cleanup" refers to the process of preparing a dataset by making sure that each column contains accurate and consistent data, and is in the appropriate format for analysis.

PROGRAM :

```
import pandas as pd
data = pd.read_csv('data.csv')
columns_to_remove = ['Gender', 'Age']
data.drop(columns=columns_to_remove, inplace=True)
```

print(data)

EXPECTED OUTPUT :

```
    Name           City  Salary
0   John       New York   50000
1   Mary    Los Angeles   75000
2  Peter  San Francisco   40000
3    Ann        Chicago   90000
4    Tom          Miami   60000
5   Jake        Houston   55000
6   Emma         Boston   65000
7  Steve         Denver   80000
8   Lisa        Seattle   45000
```

OBSERVED OUTPUT :

```
    Name           City  Salary
0   John       New York   50000
1   Mary    Los Angeles   75000
2  Peter  San Francisco   40000
3    Ann        Chicago   90000
4    Tom          Miami   60000
5   Jake        Houston   55000
6   Emma         Boston   65000
7  Steve         Denver   80000
8   Lisa        Seattle   45000
```

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 : Write a program for removing null values using python

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({'Column1': [1, 2, None, 4], 'Column2': [5, None, 7, 8]})
print('Original dataframe:', df)
df = df.dropna()
print('Dataframe after removing null values:', df)
```

EXPECTED OUTPUT :

```
Original dataframe:
    Column1  Column2
0      1.0      5.0
1      2.0      NaN
2      NaN      7.0
3      4.0      8.0
Dataframe after removing null values:
    Column1  Column2
0      1.0      5.0
3      4.0      8.0
```

OBSERVED OUTPUT :

```
Original dataframe:
    Column1  Column2
0      1.0      5.0
1      2.0      NaN
2      NaN      7.0
3      4.0      8.0
Dataframe after removing null values:
    Column1  Column2
0      1.0      5.0
3      4.0      8.0
```

(b) Understanding your variables

AIM : Write a program for Understanding your variables using python

DESCRIPTION :
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.
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({'Name': ['Alice', 'Bob', 'Charlie', 'David'],
'Age': [25, 30, 35, 40],
'Gender': ['F', 'M', 'M', 'M'],
'Salary': [50000, 70000, 60000, 80000]})
print('Number of unique values in each column:', df.nunique())
```

EXPECTED OUTPUT :

```
Number of unique values in each column:
 Name      4
Age        4
Gender     2
Salary     4
dtype: int64
```

OBSERVED OUTPUT :

```
Number of unique values in each column:
 Name      4
Age        4
Gender     2
Salary     4
dtype: int64
```

(c) Relationships between continuous variables

AIM : Write a program for Relationships between continuous variables using python

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 :

```
The correlation coefficient between variable_1 and variable_2 is: 1.0
```

OBSERVED OUTPUT :

```
The correlation coefficient between variable_1 and variable_2 is: 1.0
```

(d) DataFrame slicing, selecting, extracting

AIM : Write a program for DataFrame slicing, selecting, extracting using python

DESCRIPTION :

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'.

Selecting involves filtering the rows and/or columns of a DataFrame based on certain conditions.

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
data = {'Name': ['John', 'Mary', 'Bob', 'Alice', 'Kate'],
'Age': [25, 30, 35, 40, 45],
'Gender': ['Male', 'Female', 'Male', 'Female', 'Female'],
'Salary': [50000, 60000, 70000, 80000, 90000]}
df = pd.DataFrame(data)
print(df.iloc[2:4])
print(df.loc[:, 'Name'])
print(df[df['Age'] > 30])
```

EXPECTED OUTPUT :

```
    Name  Age  Gender  Salary
2    Bob   35    Male   70000
3  Alice   40  Female   80000
0     John
1     Mary
2      Bob
3    Alice
4     Kate
Name: Name, dtype: object
    Name  Age  Gender  Salary
2    Bob   35    Male   70000
3  Alice   40  Female   80000
4   Kate   45  Female   90000
```

OBSERVED OUTPUT :

```
    Name  Age  Gender  Salary
2    Bob   35    Male   70000
3  Alice   40  Female   80000
0   John
1   Mary
2    Bob
3  Alice
4   Kate
Name: Name, dtype: object
    Name  Age  Gender  Salary
2    Bob   35    Male   70000
3  Alice   40  Female   80000
4   Kate   45  Female   90000
```

(e) Conditional selections

AIM : Write a program for Conditional selections using python

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
data = {'Name': ['John', 'Mary', 'Bob', 'Alice', 'Kate'],
'Age': [25, 30, 35, 40, 45],
'Gender': ['Male', 'Female', 'Male', 'Female', 'Female'],
'Salary': [50000, 60000, 70000, 80000, 90000]}
df = pd.DataFrame(data)
age_above_30 = df[df['Age'] ¿ 30]
print("Rows where Age is greater than 30:")
print(age_above_30)
female_above_70k = df[(df['Gender'] == 'Female') & (df['Salary'] ¿ 70000)]
print("Rows where Gender is Female and Salary is greater than 70000:")
print(female_above_70k)
```

EXPECTED OUTPUT :

```
Rows where Age is greater than 30:
    Name  Age  Gender  Salary
2    Bob   35    Male   70000
3  Alice   40  Female   80000
4   Kate   45  Female   90000

Rows where Gender is Female and Salary is greater than 70000:
    Name  Age  Gender  Salary
3  Alice   40  Female   80000
4   Kate   45  Female   90000
```

OBSERVED OUTPUT :

```
Rows where Age is greater than 30:
    Name  Age  Gender  Salary
2    Bob   35    Male   70000
3  Alice   40  Female   80000
4   Kate   45  Female   90000

Rows where Gender is Female and Salary is greater than 70000:
    Name  Age  Gender  Salary
3  Alice   40  Female   80000
4   Kate   45  Female   90000
```

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 : Write a program for Creating DataFrames from scratch using python

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 :

```
import pandas as pd
df = pd.DataFrame()
df['Name'] = ['John', 'Sarah', 'Michael', 'David']
df['Age'] = [25, 30, 35, 40]
df['Gender'] = ['Male', 'Female', 'Male', 'Male']
print(df)
```

EXPECTED OUTPUT :

```
      Name  Age  Gender
0     John   25    Male
1    Sarah   30  Female
2  Michael   35    Male
3    David   40    Male
```

OBSERVED OUTPUT :

```
      Name  Age  Gender
0     John   25    Male
1    Sarah   30  Female
2  Michael   35    Male
3    David   40    Male
```

(b) Looking at top n records

AIM : Write a program for Looking at top n records using python

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', 'Sarah', 'Michael', 'David', 'Emily'],
'Age': [25, 30, 35, 40, 45],
'Gender': ['Male', 'Female', 'Male', 'Male', 'Female']}
df = pd.DataFrame(data)
top_n = 3
top_records = df.head(top_n)
print(top_records)
```

EXPECTED OUTPUT :

```
      Name  Age  Gender
0     John   25    Male
1    Sarah   30  Female
2  Michael   35    Male
```

OBSERVED OUTPUT :

```
      Name  Age  Gender
0     John   25    Male
1    Sarah   30  Female
2  Michael   35    Male
```

(c) Looking at bottom n records

AIM : Write a program for Looking at bottom n records using python

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', 'Sarah', 'Michael', 'David', 'Emily'],
'Age': [25, 30, 35, 40, 45],
'Gender': ['Male', 'Female', 'Male', 'Male', 'Female']}
df = pd.DataFrame(data)
bottom_n = 2
bottom_records = df.tail(bottom_n)
print(bottom_records)
```

EXPECTED OUTPUT :

```
    Name  Age  Gender
3  David   40    Male
4  Emily   45  Female
```

OBSERVED OUTPUT :

```
    Name  Age  Gender
3  David   40    Male
4  Emily   45  Female
```

(d) View columns names

AIM : Write a program for View columns names using python

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', 'Sarah', 'Michael', 'David'],
'Age': [25, 30, 35, 40],
'Gender': ['Male', 'Female', 'Male', 'Male']}
df = pd.DataFrame(data)
columns = df.columns
print(columns)
```

EXPECTED OUTPUT :

```
Index(['Name', 'Age', 'Gender'], dtype='object')
```

OBSERVED OUTPUT :

```
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 : Write a program Rename method helps to rename column of data frame using python

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

PROGRAM :

```
import pandas as pd
data = {'Name': ['John', 'Sarah', 'Michael', 'David'], 'Age': [25, 30, 35, 40], 'Gender': ['Male', 'Female', 'Male', 'Male']}
df = pd.DataFrame(data)
df = df.rename(columns={'Age': 'Years'})
print(df.columns)
```

EXPECTED OUTPUT :

```
Index(['Name', 'Years', 'Gender'], dtype='object')
```

OBSERVED OUTPUT :

```
Index(['Name', 'Years', 'Gender'], dtype='object')
```

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

AIM : Write a program for To rename the column of existing data frame set inplace=True using python

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', 'Sarah', 'Michael', 'David'], 'Age': [25, 30, 35, 40], 'Gender': ['Male', 'Female', 'Male', 'Male']}
df = pd.DataFrame(data)
df.rename(columns='Age': 'Years', inplace=True)
print(df.columns)
```

EXPECTED OUTPUT :

```
Index(['Name', 'Years', 'Gender'], dtype='object')
```

OBSERVED OUTPUT :

```
Index(['Name', 'Years', 'Gender'], dtype='object')
```

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

(a) Accessing sub data frames

AIM : Write a program for Accessing sub data frames using python

DESCRIPTION :

In pandas, a sub DataFrame can be created by selecting specific rows and columns from an existing DataFrame. This can be done using various methods such as integer indexing, label indexing, boolean indexing, and using the loc and iloc functions. The loc function is used for label-based indexing, while the iloc function is used for integer-based indexing. Once a sub DataFrame is created, it can be used for further analysis or manipulation. Overall, accessing sub DataFrames using pandas is an essential technique that allows users to work with specific subsets of data within a larger dataset.

PROGRAM :

```
import pandas as pd
data = {
'Name': ['John', 'Sam', 'Sarah', 'Mike', 'David'],
'Age': [25, 31, 29, 26, 28],
'City': ['New York', 'London', 'Paris', 'Sydney', 'Tokyo'],
'Salary': [50000, 70000, 60000, 55000, 80000]
}
df = pd.DataFrame(data)
sub_df = df[df['Age'] ≥ 28]
print('Original dataframe:')
print(df)
print('Sub-dataframe where Age ≥ 28 :')
print(sub_df)
```

EXPECTED OUTPUT :

```
Original dataframe:
    Name  Age      City  Salary
0   John   25  New York   50000
1    Sam   31    London   70000
2  Sarah   29     Paris   60000
3   Mike   26    Sydney   55000
4  David   28     Tokyo   80000
Sub-dataframe where Age >= 28:
    Name  Age    City  Salary
1    Sam   31  London   70000
2  Sarah   29   Paris   60000
4  David   28   Tokyo   80000
```

OBSERVED OUTPUT :

```
Original dataframe:
    Name  Age      City  Salary
0   John   25  New York   50000
1    Sam   31    London   70000
2  Sarah   29     Paris   60000
3   Mike   26    Sydney   55000
4  David   28     Tokyo   80000
Sub-dataframe where Age >= 28:
    Name  Age    City  Salary
1    Sam   31  London   70000
2  Sarah   29   Paris   60000
4  David   28   Tokyo   80000
```

(b) Filtering Records

AIM : Write a program for Filtering Records using python

DESCRIPTION :

Filtering records using pandas in Python involves selecting a subset of data from a larger dataset based on certain criteria. This can be done by applying conditions on specific columns of the DataFrame using boolean indexing or by using query function in pandas. The filtered data can then be further analyzed or used for other purposes. Overall, filtering records using pandas is a powerful tool that allows users to efficiently manipulate and analyze data in Python.

PROGRAM :

```
import pandas as pd
data = {
'Name': ['John', 'Sam', 'Sarah', 'Mike', 'David'],
'Age': [25, 31, 29, 26, 28],
'City': ['New York', 'London', 'Paris', 'Sydney', 'Tokyo'],
'Salary': [50000, 70000, 60000, 55000, 80000]
}
df = pd.DataFrame(data)
filtered_df = df[(df['Age'] ≥ 28)(df['Salary'] ≥ 60000)]
print('Original dataframe:')
print(df)
print('Filtered dataframe where Age ≥ 28andSalary ≥ 60000 :')
print(filtered_df)
```

EXPECTED OUTPUT :

```
Original dataframe:
    Name  Age      City  Salary
0   John   25  New York   50000
1    Sam   31    London   70000
2  Sarah   29     Paris   60000
3   Mike   26    Sydney   55000
4  David   28     Tokyo   80000
Filtered dataframe where Age >= 28 and Salary >= 60000:
    Name  Age    City  Salary
1    Sam   31  London   70000
2  Sarah   29   Paris   60000
4  David   28   Tokyo   80000
```

OBSERVED OUTPUT :

```
Original dataframe:
    Name  Age      City  Salary
0   John   25  New York   50000
1    Sam   31    London   70000
2  Sarah   29     Paris   60000
3   Mike   26    Sydney   55000
4  David   28     Tokyo   80000
Filtered dataframe where Age >= 28 and Salary >= 60000:
    Name  Age    City  Salary
1    Sam   31  London   70000
2  Sarah   29   Paris   60000
4  David   28   Tokyo   80000
```

8. Handling Missing Values
a. Dropna
b. Fillna
c. Recognize and Treat missing values and outliers in Pandas

(a) Dropna

AIM : Write a program for Dropna using python

DESCRIPTION :

The dropna() function in pandas is used to remove rows or columns from a DataFrame that contain missing or null values. By default, it removes any rows that contain at least one NaN value, but the behavior can be customized using different parameters. This function is useful for cleaning and preparing data for analysis or modeling, as null values can create issues with computations and statistical measures. Overall, dropna() is an essential function in the pandas library that provides a powerful tool for data cleaning and manipulation in Python.

PROGRAM :

```
import pandas as pd
import numpy as np
data = {
'Name': ['John', 'Sam', np.nan, 'Mike', 'David'],
'Age': [25, 31, 29, np.nan, 28],
'City': ['New York', 'London', np.nan, 'Sydney', 'Tokyo'],
'Salary': [50000, np.nan, 60000, 55000, 80000]
}
df = pd.DataFrame(data)
df1 = df.dropna()
print('Original dataframe:')
print(df)
print('Dataframe after dropping NaN values:')
print(df1)
```

EXPECTED OUTPUT :

```
Original dataframe:
    Name   Age      City   Salary
0   John  25.0  New York  50000.0
1    Sam  31.0    London      NaN
2    NaN  29.0       NaN  60000.0
3   Mike   NaN    Sydney  55000.0
4  David  28.0     Tokyo  80000.0
Dataframe after dropping NaN values:
    Name   Age      City   Salary
0   John  25.0  New York  50000.0
4  David  28.0     Tokyo  80000.0
```

OBSERVED OUTPUT :

```
Original dataframe:
    Name   Age      City   Salary
0   John  25.0  New York  50000.0
1    Sam  31.0    London      NaN
2    NaN  29.0       NaN  60000.0
3   Mike   NaN    Sydney  55000.0
4  David  28.0     Tokyo  80000.0
Dataframe after dropping NaN values:
    Name   Age      City   Salary
0   John  25.0  New York  50000.0
4  David  28.0     Tokyo  80000.0
```

(b) Fillna

AIM : Write a program for Fillna using python

DESCRIPTION :

The fillna() function in pandas is used to fill missing or null values in a DataFrame with a specified value or a method such as forward or backward fill. It can also be used to fill missing values with the mean, median, or mode of a column. This function is useful for cleaning and preparing data for analysis or modeling, as null values can create issues with computations and statistical measures. Overall, fillna() is an essential function in the pandas library that provides a powerful tool for data cleaning and manipulation in Python.

PROGRAM :

```
import pandas as pd
import numpy as np
data = {
'Name': ['John', 'Sam', np.nan, 'Mike', 'David'],
'Age': [25, 31, 29, np.nan, 28],
'City': ['New York', 'London', np.nan, 'Sydney', 'Tokyo'],
'Salary': [50000, np.nan, 60000, 55000, 80000]
}
df = pd.DataFrame(data)
df['Age'] = df['Age'].fillna(df['Age'].mean())
df['Salary'] = df['Salary'].fillna(df['Salary'].mean())
df['Name'] = df['Name'].fillna('Unknown')
df['City'] = df['City'].fillna('Unknown')
print('Dataframe after filling NaN values:')
print(df)
```

EXPECTED OUTPUT :

```
Dataframe after filling NaN values:
      Name    Age      City   Salary
0     John  25.00  New York  50000.0
1      Sam  31.00    London  61250.0
2  Unknown  29.00   Unknown  60000.0
3     Mike  28.25    Sydney  55000.0
4    David  28.00     Tokyo  80000.0
```

OBSERVED OUTPUT :

```
Dataframe after filling NaN values:
      Name    Age      City   Salary
0     John  25.00  New York  50000.0
1      Sam  31.00    London  61250.0
2  Unknown  29.00   Unknown  60000.0
3     Mike  28.25    Sydney  55000.0
4    David  28.00     Tokyo  80000.0
```

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

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

DESCRIPTION :

In pandas, missing values and outliers can be identified using various functions such as isnull(), notnull(), and describe(). Once identified, missing values can be treated using the fillna() function to fill them with appropriate values or using the dropna() function to remove them from the DataFrame. Outliers can be treated by removing them using different methods such as the Z-score method or using percentile-based filtering. In addition, visualizations such as scatter plots and box plots can be used to identify outliers and missing values. Overall, recognizing and treating missing values and outliers is an essential step in data cleaning and preparation, which is a critical aspect of data analysis and modeling in pandas.

PROGRAM :

```
import pandas as pd
import numpy as np
data = {
'Name': ['John', 'Sam', np.nan, 'Mike', 'David'],
'Age': [25, 31, 29, np.nan, 28],
'City': ['New York', 'London', np.nan, 'Sydney', 'Tokyo'],
'Salary': [50000, np.nan, 60000, 55000, 80000],
'Sales': [1000, 1200, 1300, 200, 15000]
}
df = pd.DataFrame(data)
print('Number of missing values in the dataframe:')
print(df.isnull().sum())
```

```
df['Age'] = df['Age'].fillna(df['Age'].mean())
df['Salary'] = df['Salary'].fillna(df['Salary'].mean())
df['City'] = df['City'].fillna('Unknown')
df['Name'] = df['Name'].fillna('Unknown')
q = df['Sales'].quantile(0.99)
df['Sales'] = np.where(df['Sales'] >q, q, df['Sales'])
print('Dataframe after treating missing values and outliers:')
print(df)
```

EXPECTED OUTPUT :

```
Number of missing values in the dataframe:
Name      1
Age       1
City      1
Salary    1
Sales     0
dtype: int64
Dataframe after treating missing values and outliers:
      Name    Age      City   Salary    Sales
0     John  25.00  New York  50000.0   1000.0
1      Sam  31.00    London  61250.0   1200.0
2  Unknown  29.00   Unknown  60000.0   1300.0
3     Mike  28.25    Sydney  55000.0    200.0
4    David  28.00     Tokyo  80000.0  14452.0
```

OBSERVED OUTPUT :

```
Number of missing values in the dataframe:
Name      1
Age       1
City      1
Salary    1
Sales     0
dtype: int64
Dataframe after treating missing values and outliers:
      Name    Age      City   Salary    Sales
0     John  25.00  New York  50000.0   1000.0
1      Sam  31.00    London  61250.0   1200.0
2  Unknown  29.00   Unknown  60000.0   1300.0
3     Mike  28.25    Sydney  55000.0    200.0
4    David  28.00     Tokyo  80000.0  14452.0
```

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 program for Splitting the data into groups using python

DESCRIPTION :

In pandas, data can be split into groups using the groupby() function. This function groups rows based on a specified column or multiple columns and creates a GroupBy object. The GroupBy object can then be used to perform various aggregation functions such as sum, mean, max, min, and count, among others. The groupby() function can also be used with the apply() function to apply a custom function to each group. Overall, splitting data into groups using pandas is an essential technique that allows users to perform in-depth analysis and gain insights into the relationships between variables in a dataset.

PROGRAM :

```
import itertools
data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
groups = itertools.groupby(data, lambda x: (x-1)//3)
for key, group in groups:
print("Group : ".format(key+1, list(group)))
```

EXPECTED OUTPUT :

```
Group 1: [1, 2, 3]
Group 2: [4, 5, 6]
Group 3: [7, 8, 9]
Group 4: [10]
```

OBSERVED OUTPUT :

```
Group 1: [1, 2, 3]
Group 2: [4, 5, 6]
Group 3: [7, 8, 9]
Group 4: [10]
```

ii. Applying a function to each group individually

AIM : Write a program for Applying a function to each group individually using python

DESCRIPTION :

In pandas, applying a function to each group individually can be done using the apply() function. The apply() function is used to apply a specified function to each group of a GroupBy object. The function can be a built-in function or a custom function created by the user. The apply() function can also be used with lambda functions for quick and simple operations. Overall, applying a function to each group individually using pandas is an essential technique that allows users to perform complex analysis and gain deeper insights into the relationships between variables in a dataset.

PROGRAM :

```
import itertools
data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
def process_group(key, group):
print("Processing group {}: {}".format(key+1, list(group)))
groups = itertools.groupby(data, lambda x: (x-1)//3)
for key, group in groups:
process_group(key, group)
```

EXPECTED OUTPUT :

```
Processing group 1: [1, 2, 3]
Processing group 2: [4, 5, 6]
Processing group 3: [7, 8, 9]
Processing group 4: [10]
```

OBSERVED OUTPUT :

```
Processing group 1: [1, 2, 3]
Processing group 2: [4, 5, 6]
Processing group 3: [7, 8, 9]
Processing group 4: [10]
```

iii. Combining the result into a data structure

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

DESCRIPTION :

In pandas, combining the results of multiple operations into a single data structure can be done using the concat(), merge(), and join() functions. The concat() function is used to combine DataFrames vertically or horizontally, while the merge() function is used to combine DataFrames based on a specified column or index. The join() function is used to join DataFrames based on a specified index. These functions allow users to combine data from different sources and perform more complex analysis and modeling. Overall, combining the results into a data structure using pandas is an essential technique that allows users to work with larger datasets and gain deeper insights into the data.

PROGRAM :

```
import itertools
data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
def process_group(key, group):
return sum(group)
groups = itertools.groupby(data, lambda x: (x-1)//3)
results = [process_group(key, group) for key, group in groups]
```

print("Results:", results)

EXPECTED OUTPUT :

```
Results: [6, 15, 24, 10]
```

OBSERVED OUTPUT :

```
Results: [6, 15, 24, 10]
```

(b) Pivot thable

AIM : Write a program for Pivot thable using python

DESCRIPTION :

In pandas, a pivot table is a way to summarize and aggregate data in a DataFrame by grouping data according to multiple variables and calculating summary statistics for each group. The pivot_table() function in pandas is used to create a pivot table from a DataFrame. The pivot_table() function allows users to specify which variables to use for the rows, columns, and values in the pivot table. Users can also specify how to aggregate the data using functions such as sum(), mean(), count(), and others. Pivot tables are useful for analyzing and visualizing complex datasets and can provide insights into relationships between variables. Overall, pivot tables are an essential tool in pandas for data analysis and modeling.

PROGRAM :

```
import pandas as pd
df = pd.DataFrame({
'Name': ['Alice', 'Bob', 'Charlie', 'Alice', 'Bob', 'Charlie'],
'Month': ['Jan', 'Jan', 'Jan', 'Feb', 'Feb', 'Feb'],
'Sales': [100, 200, 150, 300, 250, 200]
})
pivot_table = pd.pivot_table(df, values='Sales', index='Name', columns='Month')
print(pivot_table)
```

EXPECTED OUTPUT

```
Month     Feb  Jan
Name
Alice     300  100
Bob       250  200
Charlie   200  150
```

OBSERVED OUTPUT :

```
Month     Feb   Jan
Name
Alice     300   100
Bob       250   200
Charlie   200   150
```

(c) Cross tab

AIM : Write a program forCross tab using python

DESCRIPTION :

In pandas, a cross tabulation table or crosstab is a way to summarize and compare the frequency or count of two or more variables. The crosstab() function in pandas is used to create a cross tabulation table from a DataFrame. The crosstab() function allows users to specify the row and column variables to use in the table, as well as any additional options such as normalization or aggregation functions. Cross tabulation tables are useful for analyzing and comparing categorical data, identifying patterns and trends, and gaining insights into relationships between variables. Overall, crosstabs are an essential tool in pandas for data analysis and visualization.

PROGRAM :

```
import pandas as pd
df = pd.DataFrame({
'Name': ['Alice', 'Bob', 'Charlie', 'Alice', 'Bob', 'Charlie'],
'Gender': ['F', 'M', 'M', 'F', 'M', 'F'],
'Sales': [100, 200, 150, 300, 250, 200]
})
cross_tab = pd.crosstab(df['Name'], df['Gender'],
values=df['Sales'], aggfunc='sum')
print(cross_tab)
```

EXPECTED OUTPUT :

Month Name	Feb	Jan
Alice	300	100
Bob	250	200
Charlie	200	150

OBSERVED OUTPUT :

Month Name	Feb	Jan
Alice	300	100
Bob	250	200
Charlie	200	150

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 program for Mearging/Concatenating Data Frames using python

DESCRIPTION :

In pandas, merging and concatenating DataFrames are techniques used to combine multiple DataFrames into a single one. The concat() function is used to concatenate DataFrames vertically or horizontally by adding rows or columns respectively. The merge() function is used to merge two DataFrames based on one or more common columns. The merge() function is similar to the SQL join operation and can perform inner, outer, left, and right joins. By merging and concatenating DataFrames, users can combine data from different sources and perform more complex analysis and modeling. Overall, merging and concatenating DataFrames are essential techniques in pandas for data manipulation and analysis.

PROGRAM :

```
import pandas as pd
df1 = pd.DataFrame({
'Name': ['Alice', 'Bob', 'Charlie'],
'Age': [25, 30, 35],
'Salary': [50000, 60000, 70000]
})
df2 = pd.DataFrame(
'Name': ['Alice', 'Bob', 'David'],
'Location': ['New York', 'San Francisco', 'London'],
'Company': ['XYZ Inc', 'ABC Corp', 'MNO LLC']
)
merged_df = pd.merge(df1, df2, on='Name', how='outer')
concat_df = pd.concat([df1, df2], ignore_index=True)
print('Merged Data Frame:', merged_df)
print('Data Frame:', concat_df)
```

EXPECTED OUTPUT :

```
Merged Data Frame:
      Name   Age   Salary       Location   Company
0    Alice  25.0  50000.0       New York   XYZ Inc
1      Bob  30.0  60000.0  San Francisco  ABC Corp
2  Charlie  35.0  70000.0            NaN       NaN
3    David   NaN      NaN         London   MNO LLC

Concatenated Data Frame:
      Name   Age   Salary       Location   Company
0    Alice  25.0  50000.0            NaN       NaN
1      Bob  30.0  60000.0            NaN       NaN
2  Charlie  35.0  70000.0            NaN       NaN
3    Alice   NaN      NaN       New York   XYZ Inc
4      Bob   NaN      NaN  San Francisco  ABC Corp
5    David   NaN      NaN         London   MNO LLC
```

OBSERVED OUTPUT :

```
Merged Data Frame:
      Name   Age   Salary       Location   Company
0    Alice  25.0  50000.0       New York   XYZ Inc
1      Bob  30.0  60000.0  San Francisco  ABC Corp
2  Charlie  35.0  70000.0            NaN       NaN
3    David   NaN      NaN         London   MNO LLC

Concatenated Data Frame:
      Name   Age   Salary       Location   Company
0    Alice  25.0  50000.0            NaN       NaN
1      Bob  30.0  60000.0            NaN       NaN
2  Charlie  35.0  70000.0            NaN       NaN
3    Alice   NaN      NaN       New York   XYZ Inc
4      Bob   NaN      NaN  San Francisco  ABC Corp
5    David   NaN      NaN         London   MNO LLC
```

(b) Transpose a Data set or dataframe using Pandas

AIM : Write a program for Transpose a Data set or dataframe using Pandas using python

DESCRIPTION :

In pandas, transposing a dataset or DataFrame means converting the rows into columns and columns into rows. This is done using the transpose() function, which swaps the rows and columns of a DataFrame or Series. The transpose operation is useful when you want to change the orientation of the data or when you want to group data in a different way. The transpose function can be used on a DataFrame or a Series, and it returns a new object with the transposed data. Overall, transposing a DataFrame is an essential technique in pandas for data manipulation and analysis.

PROGRAM :

```
import pandas as pd
df = pd.DataFrame({
'Name': ['Alice', 'Bob', 'Charlie'],
'Age': [25, 30, 35],
'Salary': [50000, 60000, 70000]
})
transposed_df = df.transpose()
print('Original Data Frame:', df)
print('Data Frame:', transposed_df)
```

EXPECTED OUTPUT :

```
Original Data Frame:
      Name  Age  Salary
0    Alice   25   50000
1      Bob   30   60000
2  Charlie   35   70000

Transposed Data Frame:
             0      1        2
Name     Alice    Bob  Charlie
Age         25     30       35
Salary   50000  60000    70000
```

OBSERVED OUTPUT :

```
Original Data Frame:
       Name  Age  Salary
0    Alice   25   50000
1      Bob   30   60000
2  Charlie   35   70000

Transposed Data Frame:
             0      1        2
Name     Alice    Bob  Charlie
Age         25     30       35
Salary   50000  60000    70000
```

(c) To sort a Pandas DataFrame

AIM : Write a program for To sort a Pandas DataFrame using python

DESCRIPTION :

In pandas, sorting a DataFrame means arranging the rows of a DataFrame in a particular order based on the values in one or more columns. The sort_values() function in pandas is used to sort a DataFrame by one or more columns. Users can specify the column(s) to sort by, and whether to sort in ascending or descending order. By default, sort_values() sorts in ascending order, but users can change the order by specifying ascending=False. The sort_index() function is used to sort a DataFrame by the index. Sorting a DataFrame is useful when you want to group or aggregate data, or when you want to order data for visualization or modeling. Overall, sorting a DataFrame is an essential technique in pandas for data manipulation and analysis.

PROGRAM :

```
import pandas as pd
df = pd.DataFrame({
'Name': ['Alice', 'Bob', 'Charlie'],
'Age': [25, 30, 35],
'Salary': [50000, 60000, 70000]
})
sorted_df = df.sort_values('Name')
print('Sorted Data Frame:', sorted_df)
```

EXPECTED OUTPUT :

```
Sorted Data Frame:
       Name  Age  Salary
0    Alice   25   50000
1      Bob   30   60000
2  Charlie   35   70000
```

OBSERVED OUTPUT :

```
Sorted Data Frame:
       Name  Age  Salary
0    Alice   25   50000
1      Bob   30   60000
2  Charlie   35   70000
```

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

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

DESCRIPTION :

In pandas, removing duplicate values means deleting rows in a DataFrame that have the same values in all columns. The drop_duplicates() function in pandas is used to remove duplicate rows from a DataFrame. Users can specify which columns to consider when dropping duplicates, and whether to keep the first or last occurrence of a duplicated row. By default, drop_duplicates() keeps the first occurrence of a duplicated row and removes all subsequent occurrences. Removing duplicate values is useful when you want to ensure data quality or when you want to eliminate redundancy in a DataFrame. Overall, removing duplicate values is an essential technique in pandas for data manipulation and analysis.

PROGRAM :

```
import pandas as pd
df = pd.DataFrame({
'Name': ['Alice', 'Bob', 'Charlie', 'Bob', 'Charlie'],
'Age': [25, 30, 35, 30, 35],
'Salary': [50000, 60000, 70000, 60000, 70000]
})
unique_names_df = df.drop_duplicates(subset=['Name'])
print('Data Frame with Unique Names:',unique_name_df)
```

EXPECTED OUTPUT :

```
Data Frame with Unique Names:
      Name  Age  Salary
0    Alice   25   50000
1      Bob   30   60000
2  Charlie   35   70000
```

OBSERVED OUTPUT :

```
Data Frame with Unique Names:
      Name  Age  Salary
0    Alice   25   50000
1      Bob   30   60000
2  Charlie   35   70000
```

11.Applying Function to element, column or data frame
a. Map
b. Apply
c. ApplyMap

(a) Map

AIM : Write a python program for Map using pandas.

DESCRIPTION :
In pandas, mapping means transforming the values in a DataFrame or Series based on a specified function or dictionary. The map() function in pandas is used to apply a function or dictionary to each element in a Series. The map() function is useful when you want to replace or transform values in a Series based on some criteria or logic. Additionally, the apply() function can be used to apply a function to each row or column in a DataFrame. The applymap() function can be used to apply a function to each element in a DataFrame. Mapping is useful when you want to manipulate the values in a DataFrame or Series in a flexible way, and it is an essential technique in pandas for data manipulation and analysis.

PROGRAM:

```
import pandas as pd
df = pd.DataFrame({
'Name': ['Alice', 'Bob', 'Charlie'],
'Age': [25, 30, 35],
'Salary': [50000, 60000, 70000]
})
def add_prefix(name):
return 'Mrs. ' + name
df['Name'] = df['Name'].map(add_prefix)
df = df.applymap(lambda x: str(x) + '$')
print('Modified Data Frame:', df)
```

EXPECTED OUTPUT:

```
Modified Data Frame:
             Name  Age  Salary
0    Mrs. Alice$  25$  50000$
1      Mrs. Bob$  30$  60000$
2  Mrs. Charlie$  35$  70000$
```

OBSERVED OUTPUT:

```
Modified Data Frame:
             Name  Age  Salary
0    Mrs. Alice$  25$  50000$
1      Mrs. Bob$  30$  60000$
2  Mrs. Charlie$  35$  70000$
```

(b) Apply

AIM : Write a python program for Apply using pandas.

DESCRIPTION :

In pandas, applying a function means executing a user-defined or built-in function on each element or row in a DataFrame or Series. The apply() function in pandas is used to apply a function along an axis of a DataFrame or Series. The apply() function is useful when you want to transform or manipulate data in a flexible and customized way. The apply() function can be used with built-in functions, lambda functions, or user-defined functions. Additionally, the applymap() function can be used to apply a function to each element in a DataFrame, and the agg() function can be used to apply multiple functions to a DataFrame simultaneously. Applying functions is useful when you want to transform data based on some criteria or logic, and it is an essential technique in pandas for data manipulation and analysis.

PROGRAM :

```
import pandas as pd
data = {'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]}
df = pd.DataFrame(data)
def add_one(x):
return x + 1
df_apply_elements = df.applymap(add_one)
print("Data Frame after applying the function to elements:", df_apply_elements)
df_apply_columns = df.apply(add_one)
print("Data Frame after applying the function to columns:", df_apply_columns)
def sum_two(x, y):
return x + y
df_apply_df = df.apply(sum_two, args=(2,))
print("Data Frame after applying the function to data frame:", df_apply_df)
```

EXPECTED OUTPUT:

```
Data Frame after applying the function to elements:
   A  B   C
0  2  5   8
1  3  6   9
2  4  7  10
Data Frame after applying the function to columns:
   A  B   C
0  2  5   8
1  3  6   9
2  4  7  10
Data Frame after applying the function to data frame:
   A  B   C
0  3  6   9
1  4  7  10
2  5  8  11
```

OBSERVED OUTPUT:

```
Data Frame after applying the function to elements:
    A  B   C
0   2  5   8
1   3  6   9
2   4  7  10
Data Frame after applying the function to columns:
    A  B   C
0   2  5   8
1   3  6   9
2   4  7  10
Data Frame after applying the function to data frame:
    A  B   C
0   3  6   9
1   4  7  10
2   5  8  11
```

(c) ApplyMap

AIM : Write a python program for ApplyMap using pandas
DESCRIPTION :

In pandas, applymap() function is used to apply a function to each element of a DataFrame. The applymap() function can be used with built-in functions, lambda functions, or user-defined functions. The applymap() function is useful when you want to transform or manipulate data in a flexible and customized way. This function is similar to the map() function, but map() works on series while applymap() works on the whole DataFrame. applymap() is a convenient way to apply a function to each element in a DataFrame, without having to iterate through each element individually. Additionally, the apply() function can be used to apply a function along an axis of a DataFrame or Series, and the agg() function can be used to apply multiple functions to a DataFrame simultaneously. Overall, applymap() is an essential technique in pandas for data manipulation and analysis.

PROGRAM :

```
import pandas as pd
data = 'A': [1, 2, 3], 'B': [4, 5, 6], 'C': [7, 8, 9]
df = pd.DataFrame(data)
def add_one(x):
return x + 1
df_applymap = df.applymap(add_one)
print("Data Frame after applying the function to elements:", df_applymap)
```

EXPECTED OUTPUT:

```
Data Frame after applying the function to elements:
    A  B   C
0   2  5   8
1   3  6   9
2   4  7  10
```

OBSERVED OUTPUT:

```
Data Frame after applying the function to elements:
    A  B   C
0   2  5   8
1   3  6   9
2   4  7  10
```

12. Basic Stats
a. Describe
b. Convariance
c. Correlation

(a) Describe

AIM : Write a python program for Describe using pandas.

DESCRIPTION :

PROGRAM :

```
import pandas as pd
df = pd.read_csv('data.csv')
summary = df.describe()
print(summary)
```

EXPECTED OUTPUT:

	Age	Salary
count	9.000000	9.000000
mean	30.444444	62222.222222
std	8.589399	16791.201400
min	18.000000	40000.000000
25%	25.000000	50000.000000
50%	29.000000	60000.000000
75%	35.000000	75000.000000
max	45.000000	90000.000000

OBSERVED OUTPUT:

	Age	Salary
count	9.000000	9.000000
mean	30.444444	62222.222222
std	8.589399	16791.201400
min	18.000000	40000.000000
25%	25.000000	50000.000000
50%	29.000000	60000.000000
75%	35.000000	75000.000000
max	45.000000	90000.000000

(b) Convariance

AIM : Write a python program for Convariance using pandas.

DESCRIPTION :

PROGRAM :

```
import pandas as pd
df = pd.read_csv('my_dataset.csv')
x = df['column_x'] y = df['column_y']
covariance = x.cov(y)
```

print("The covariance between x and y is:", covariance)

EXPECTED OUTPUT:

```
The covariance between x and y is: 25.0
```

OBSERVED OUTPUT:

```
The covariance between x and y is: 25.0
```

(c) Describe

AIM : Write a python program for Correlation Describe using pandas.

DESCRIPTION :

PROGRAM :

```
import pandas as pd
df = pd.read_csv('my_dataset.csv')
x = df['column_x']
y = df['column_y']
correlation = x.corr(y)
print("The correlation between x and y is:", correlation)
```

EXPECTED OUTPUT:

```
The correlation between x and y is: 1.0
```

OBSERVED OUTPUT:

```
The correlation between x and y is: 1.0
```