SAVEETHA SCHOOL OF EENGINEERING

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

DSA05-QUERY PROCESSING

LAB MANUAL

SAVEETHA SCHOOL OF EENGINEERING

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

Engineer to Excel

TABLE OF CONTENTS

15	15	Write a Pandas program to keep the rows with at least 2 NaN values in a given DataFrame.	
16	16	Write a Pandas program to split the following dataframe into groups based on school code. Also check the type of GroupBy object.	
17	17	Write a Pandas program to split the following dataframe by school code and get mean, min, and max value of age for each school	
18	18	Write a Pandas program to split the following given dataframe into groups based on school code and class	
19	19	Write a Pandas program to display the dimensions or shape of the World alcohol consumption dataset. Also extract the column names from the dataset	
20	20	Write a Pandas program to find the index of a given substring of a DataFrame column.	
21	21	String Case Transformation: Swapping Character Cases in a DataFrame Column	
22	22	Basic Data Visualization: Creating a Labeled Line Plot with Axes and Titles	
23	23	File-Based Plotting: Generating a Line Graph from External Text File Coordinates	
24	24	Financial Time-Series Analysis: Visualizing Alphabet Inc. Stock Trends (Oct 2016)	
25	25	Multi-Series Comparison: Plotting Multiple Lines with Custom Aesthetics and Legends	
26	26	Subplot Layouts: Creating Multiple Independent Plots in a Single Figure	
27	27	Statistical Bar Charts: Vertical Popularity Analysis of Programming Languages	
28	28	Horizontal Distribution: Comparing Language Popularity via Horizontal Bar Charts	
29	29	Distinct Categorical Styling: Bar Chart Visualization with Unique Color Mapping	
30	30	Python Stacked Bar Plot with Error Bars	
31	31	Python Scatter Plot Program	
32	32	Python Scatter Plot with Empty Circles	

33	33	Hollow Marker Visualization: Scatter Plot Utilizing Empty Circular Distributions	
34	34	Bubble Chart Dynamics: Randomly Distributed Scatter Plot with Variable Marker Sizes	
35	35	Bivariate Correlation: Comparative Scatter Analysis of Mathematics vs. Science Performance	
36	36	Multivariate Group Analysis: Three-Group Comparison of Physical Weight and Height Metrics	
37	37	Data Structuring: Initializing a Pandas DataFrame from Dictionary Objects	
38	38	Index Customization: DataFrame Construction with Explicit Row Labeling	
39	39	Positional Data Retrieval: Extracting Leading Rows from a Structured Dataset	
40	40	Feature Selection: Column-Specific Extraction for Targeted Data Analysis	

1. Write a Pandas program to select the distinct department id from employees file.

Aim

To write a Pandas program to select and display the **distinct department IDs** from the employees dataset.

Algorithm

1. Import the Pandas library.
2. Create a DataFrame using the given employees data.
3. Select the DEPARTMENT_ID column from the DataFrame.
4. Use the unique() function to extract distinct department IDs.
5. Display the distinct department IDs.

Program

```
import pandas as pd

data = {

  'DEPARTMENT_ID': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100,

                    110, 120, 130, 140, 150, 160, 170, 180,

                    190, 200, 210, 220, 230, 240, 250, 260, 270],

  'DEPARTMENT_NAME': ['Administration', 'Marketing', 'Purchasing',

                      'Human Resources', 'Shipping', 'IT',

                      'Public Relations', 'Sales', 'Executive',

                      'Finance', 'Accounting', 'Treasury',

                      'Corporate Tax', 'Control And Credit',

                      'Shareholder Services', 'Benefits',

                      'Manufacturing', 'Construction',

                      'Contracting', 'Operations', 'IT Support',

                      'NOC', 'IT Helpdesk', 'Government Sales',


  'Retail Sales', 'Recruiting', 'Payroll']
}
df = pd.DataFrame(data)
distinct_departments = df['DEPARTMENT_ID'].unique()
print("Distinct Department IDs:")
print(distinct_departments)
```

Output

```
Distinct Department IDs:
[ 10  20  30  40  50  60  70  80  90 100 110 120 130 140 150
 160 170 180 190 200 210 220 230 240 250 260 270]
```

Result

Thus, the Pandas program was successfully executed and the **distinct department IDs** from the employees dataset were displayed.

2. Write a Pandas program to display the IDs of employees who have done two or more jobs in the past.

Aim

To write a Pandas program to display the **IDs of employees who have done two or more jobs in the past**.

Algorithm

1. Import the Pandas library.
2. Create a DataFrame using the given job history data.
3. Group the data by EMPLOYEE_ID.
4. Count the number of jobs done by each employee.
5. Select employees whose job count is **greater than or equal to 2**.
6. Display the employee IDs.

Program

```
import pandas as pd
data = {
  'EMPLOYEE_ID': [102, 101, 101, 201, 114, 122, 200, 176, 176, 200],
  'START_DATE': ['2001-01-13', '1997-09-21', '2001-10-28', '2004-02-17',
                 '2006-03-24', '2007-01-01', '1995-09-17',
                 '2006-03-24', '2007-01-01', '2002-07-01'],
  'END_DATE': ['2006-07-24', '2001-10-27', '2005-03-15', '2007-12-19',
               '2007-12-31', '2007-12-31', '2001-06-17',
               '2006-12-31', '2007-12-31', '2006-12-31'],
  'JOB_ID': ['IT_PROG', 'AC_ACCOUNT', 'AC_MGR', 'MK_REP',
             'ST_CLERK', 'ST_CLERK', 'AD_ASST',

    'SA_REP', 'SA_MAN', 'AC_ACCOUNT'],
  'DEPARTMENT_ID': [60, 110, 110, 20, 50, 50, 90, 80, 80, 90]
}
```

```
df = pd.DataFrame(data)


job_count = df.groupby('EMPLOYEE_ID').size()

employees_multiple_jobs = job_count[job_count >= 2].index

print("Employees who have done two or more jobs:")

print(list(employees_multiple_jobs))
```

Output

```
Employees who have done two or more jobs:

[101, 176, 200]
```

Result

Thus, the Pandas program was successfully executed and the **employee IDs who have performed two or more jobs** were displayed.

3. Write a Pandas program to display the details of jobs in descending sequence on job title.

Aim

To write a Pandas program to display the **details of jobs in descending order of job title**.

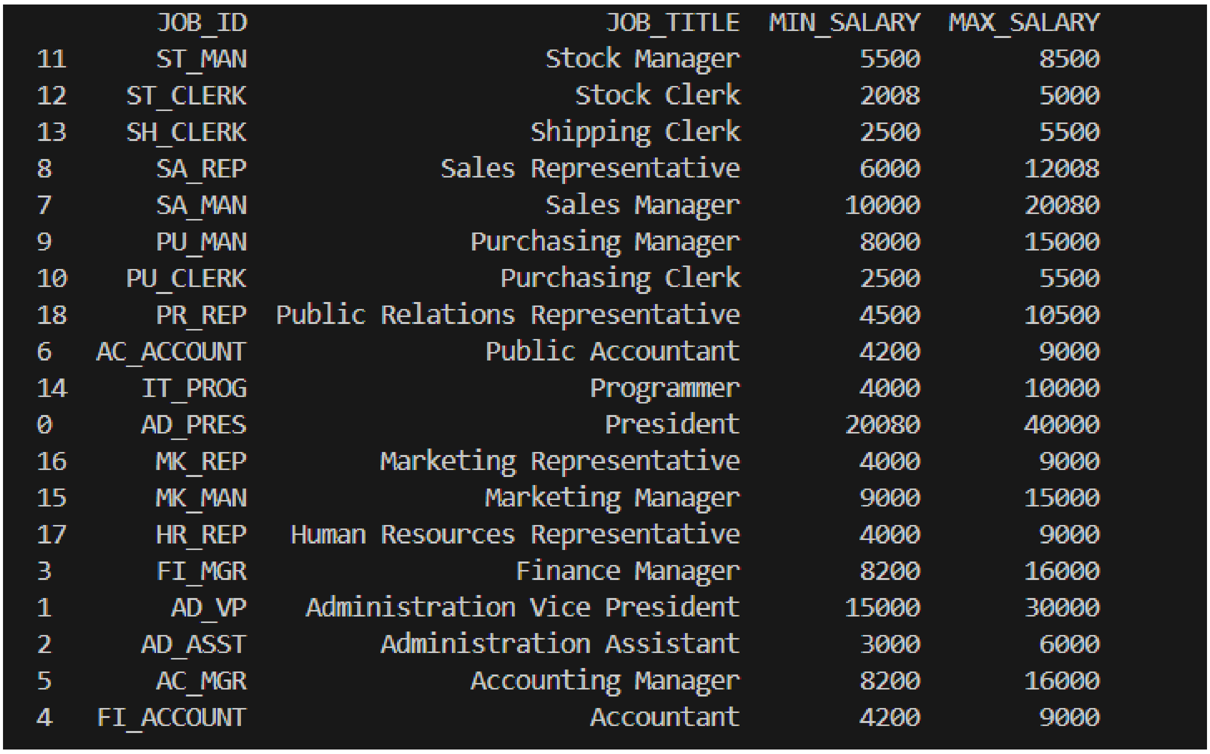
Algorithm

1. Import the Pandas library.
2. Create a DataFrame using the given job details.
3. Sort the DataFrame by the JOB_TITLE column in **descending order**.
4. Display the sorted job details.

Program

```
import pandas as pd
data = {
  'JOB_ID': ['AD_PRES', 'AD_VP', 'AD_ASST', 'FI_MGR', 'FI_ACCOUNT',
         'AC_MGR', 'AC_ACCOUNT', 'SA_MAN', 'SA_REP',
         'PU_MAN', 'PU_CLERK', 'ST_MAN', 'ST_CLERK',
         'SH_CLERK', 'IT_PROG', 'MK_MAN', 'MK_REP',
         'HR_REP', 'PR_REP'],
  'JOB_TITLE': ['President', 'Administration Vice President',
           'Administration Assistant', 'Finance Manager',
           'Accountant', 'Accounting Manager', 'Public Accountant',
           'Sales Manager', 'Sales Representative',
           'Purchasing Manager', 'Purchasing Clerk',
           'Stock Manager', 'Stock Clerk',
           'Shipping Clerk', 'Programmer',
           'Marketing Manager', 'Marketing Representative',
           'Human Resources Representative',
           'Public Relations Representative'],
  '
MIN_SALARY': [20080, 15000, 3000, 8200, 4200,
               8200, 4200, 10000, 6000,
               8000, 2500, 5500, 2008,
               2500, 4000, 9000, 4000,
               4000, 4500],
  'MAX_SALARY': [40000, 30000, 6000, 16000, 9000,
               16000, 9000, 20080, 12008,
               15000, 5500, 8500, 5000,
               5500, 10000, 15000, 9000,
               9000, 10500]
}
df = pd.DataFrame(data)
sorted_jobs = df.sort_values(by='JOB_TITLE', ascending=False)
print(sorted_jobs)
```

Output



```
        JOB_ID                         JOB_TITLE  MIN_SALARY  MAX_SALARY
11      ST_MAN                     Stock Manager        5500        8500
12    ST_CLERK                       Stock Clerk        2008        5000
13    SH_CLERK                    Shipping Clerk        2500        5500
8       SA_REP              Sales Representative        6000       12008
7       SA_MAN                     Sales Manager       10000       20080
9       PU_MAN                Purchasing Manager        8000       15000
10    PU_CLERK                  Purchasing Clerk        2500        5500
18      PR_REP   Public Relations Representative        4500       10500
6   AC_ACCOUNT                Public Accountant        4200        9000
14     IT_PROG                        Programmer        4000       10000
0      AD_PRES                         President       20080       40000
16      MK_REP          Marketing Representative        4000        9000
15      MK_MAN                 Marketing Manager        9000       15000
17      HR_REP    Human Resources Representative        4000        9000
3       FI_MGR                   Finance Manager        8200       16000
1        AD_VP     Administration Vice President       15000       30000
2      AD_ASST          Administration Assistant        3000        6000
5       AC_MGR                Accounting Manager        8200       16000
4   FI_ACCOUNT                        Accountant        4200        9000
```

Result

Thus, the Pandas program was successfully executed and the **job details were displayed in descending order of job title**.

4. Write a Pandas program to create a line plot of the historical stock prices of Alphabet Inc. between two specific dates.

Aim

To write a Pandas program to create a **line plot of the historical stock prices of Alphabet Inc.** between two specific dates.

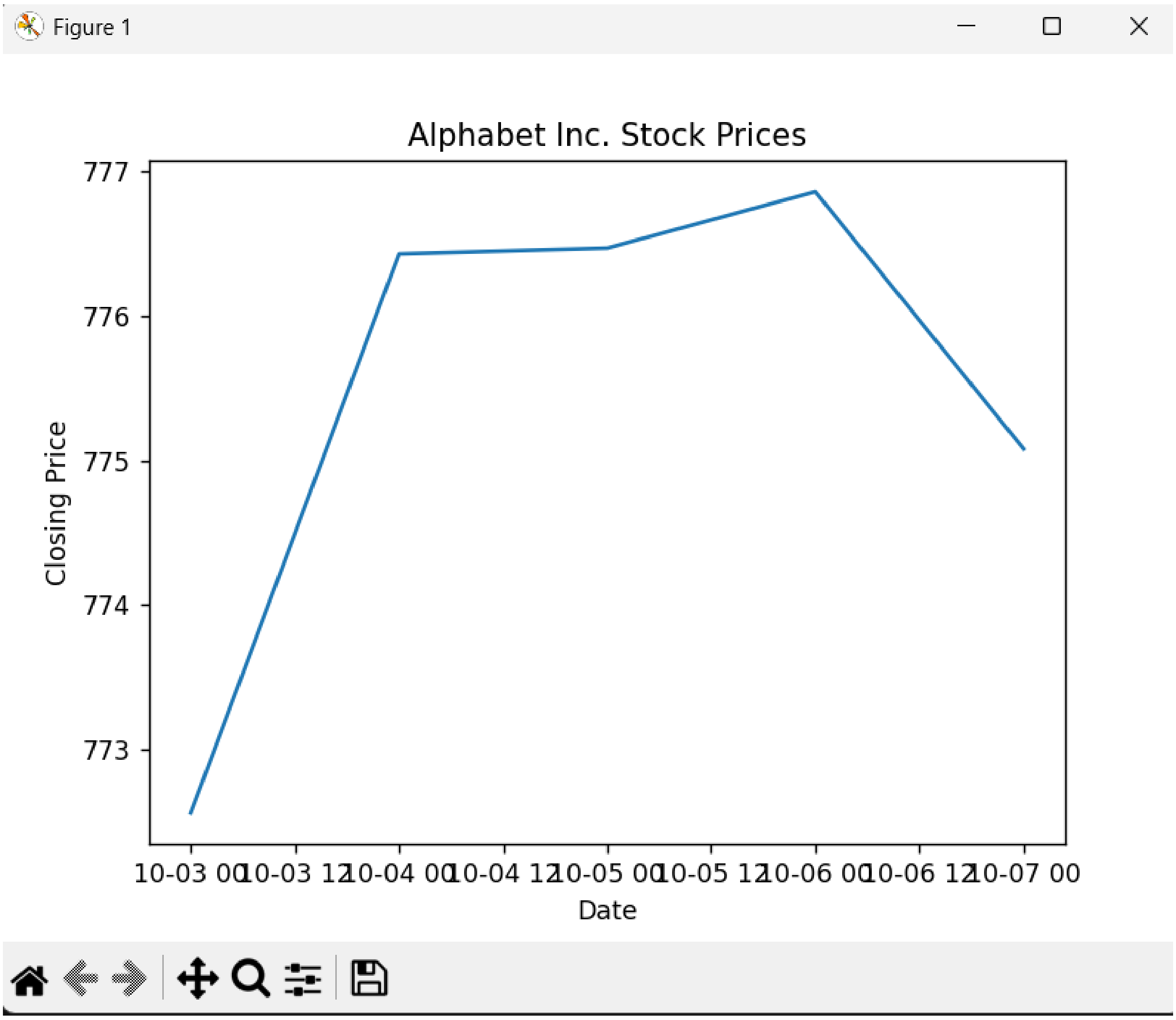
Algorithm

1. Import Pandas and Matplotlib libraries.
2. Create a DataFrame using Alphabet stock price data.
3. Convert the Date column into datetime format.
4. Set the Date column as the index.
5. Plot the stock **Closing Price** using a line plot.
6. Label the X-axis, Y-axis, and give a suitable title.
7. Display the plot.

Program

```
import pandas as pd
import matplotlib.pyplot as plt
data = {
    'Date': ['2016-10-03', '2016-10-04', '2016-10-05', '2016-10-06', '2016-10-07'],
    'Close': [772.56, 776.43, 776.47, 776.86, 775.08]
}
df = pd.DataFrame(data)
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
plt.plot(df.index, df['Close'])
plt.xlabel("Date")
plt.ylabel("Closing Price")
plt.title("Alphabet Inc. Stock Prices")
plt.show()
```

Sample Output



Result

Thus, the Pandas program was successfully executed and a **line plot representing Alphabet Inc.'s historical stock prices** was generated.

5. Write a Pandas program to create a bar plot of the trading volume of Alphabet Inc. stock between two specific dates.

Aim

To write a Pandas program to create a **bar plot of the trading volume of Alphabet Inc. stock** between two specific dates.

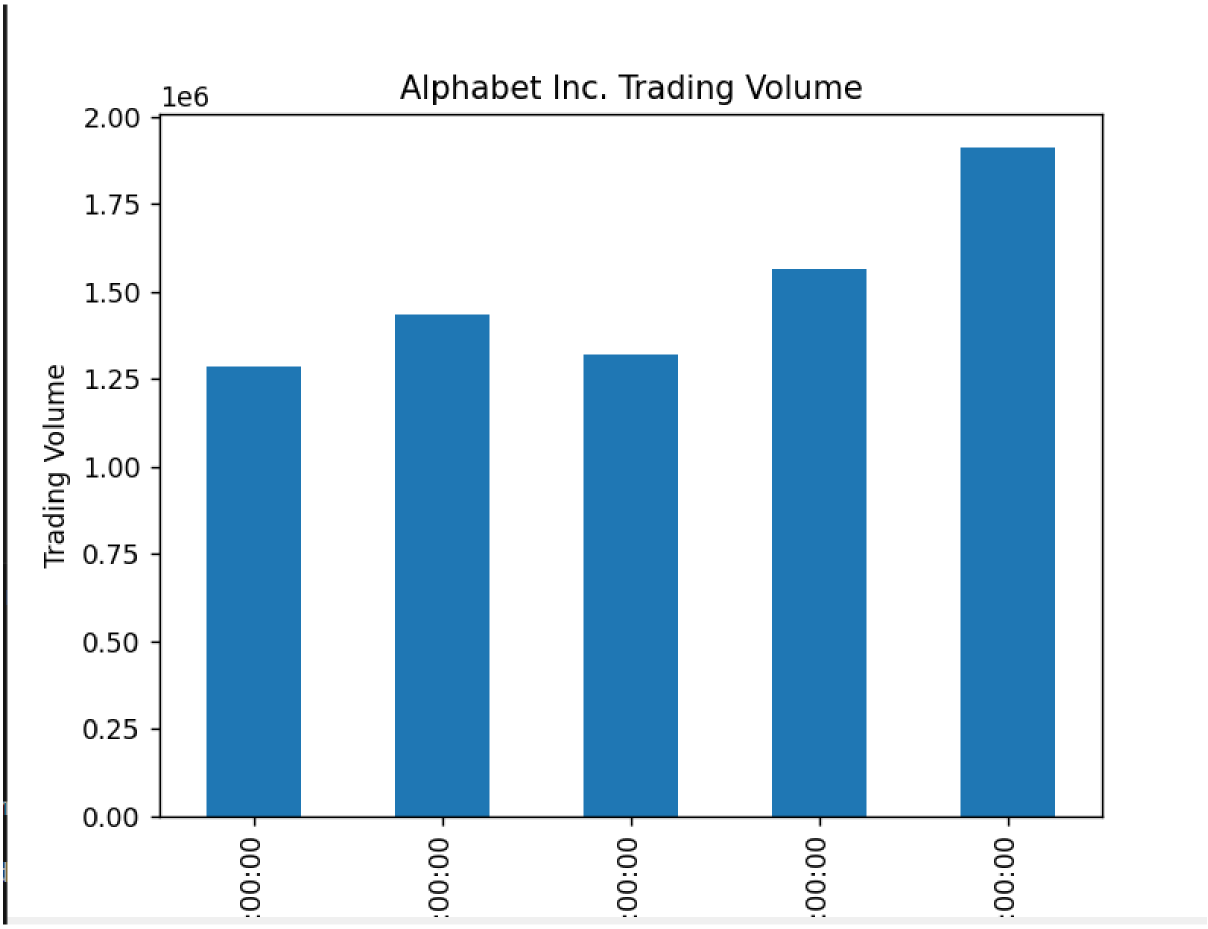
Algorithm

1. Import Pandas and Matplotlib libraries.
2. Create a DataFrame using Alphabet Inc. stock trading volume data.
3. Convert the Date column into datetime format.
4. Set the Date column as the index.
5. Create a **bar plot** for the trading volume.
6. Label the X-axis, Y-axis, and provide a suitable title.
7. Display the plot.

Program

```
import pandas as pd
import matplotlib.pyplot as plt
data = {
    'Date': ['2016-10-03', '2016-10-04', '2016-10-05', '2016-10-06', '2016-10-07'],
    'Volume': [1284100, 1431700, 1322500, 1565300, 1911400]
}
df = pd.DataFrame(data)
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
df['Volume'].plot(kind='bar')
plt.xlabel("Date")
plt.ylabel("Trading Volume")
plt.title("Alphabet Inc. Trading Volume")
plt.show()
```

Output



Result

Thus, the Pandas program was successfully executed and a **bar plot of Alphabet Inc. stock trading volume** was generated.

6. Write a Pandas program to create a scatter plot of the trading volume/stock prices of Alphabet Inc. stock between two specific dates.

Aim

To write a Pandas program to create a **scatter plot showing the relationship between trading volume and stock prices of Alphabet Inc.** between two specific dates.

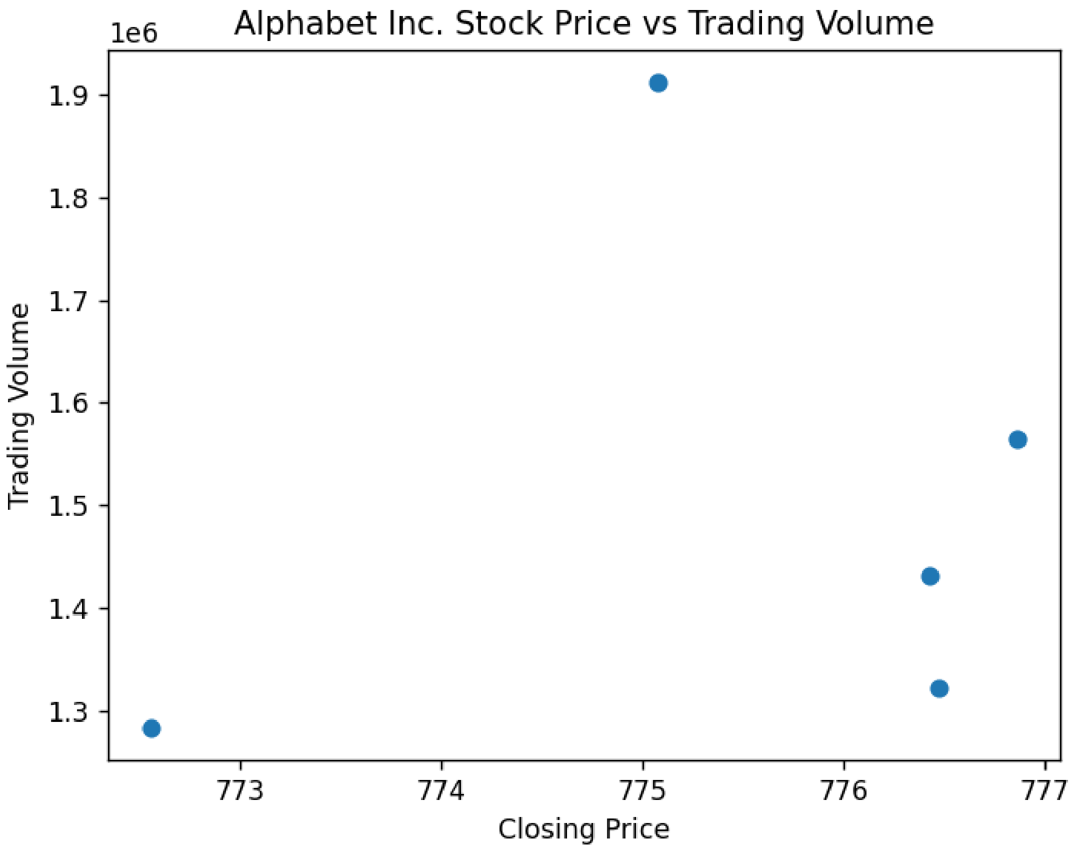
Algorithm

1. Import Pandas and Matplotlib libraries.
2. Create a DataFrame using Alphabet Inc. stock **closing price** and **trading volume** data.
3. Convert the Date column into datetime format.
4. Create a **scatter plot** with Closing Price on X-axis and Trading Volume on Y-axis.
5. Label the axes and provide a suitable title.
6. Display the scatter plot.

Program

```
import pandas as pd
import matplotlib.pyplot as plt
data = {
    'Date': ['2016-10-03', '2016-10-04', '2016-10-05', '2016-10-06', '2016-10-07'],
    'Close': [772.56, 776.43, 776.47, 776.86, 775.08],
    'Volume': [1284100, 1431700, 1322500, 1565300, 1911400]
}
df = pd.DataFrame(data)
df['Date'] = pd.to_datetime(df['Date'])
plt.scatter(df['Close'], df['Volume'])
plt.xlabel("Closing Price")
plt.ylabel("Trading Volume")
plt.title("Alphabet Inc. Stock Price vs Trading Volume")
plt.show()
```

Sample Output



Result

Thus, the Pandas program was successfully executed and a **scatter plot representing trading volume versus stock prices** of Alphabet Inc. was generated.

7. Write a Pandas program to create a Pivot table and find the maximum and minimum sale value of the items.(refer sales_data table)

Aim

To write a Pandas program to create a **Pivot Table** and find the **maximum and minimum sale value of the items** using the sales data.

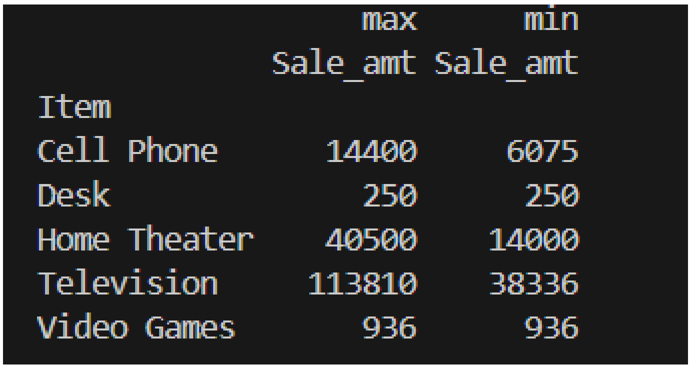
Algorithm

1. Import the Pandas library.
2. Create a DataFrame using the given sales data.
3. Create a Pivot Table using Item as index and Sale_amt as values.
4. Apply aggregation functions **maximum** and **minimum** on sale amount.
5. Display the pivot table.

Program

```
import pandas as pd
data = {
    'Item': ['Television', 'Home Theater', 'Television', 'Cell Phone',
            'Television', 'Home Theater', 'Television', 'Television',
            'Television', 'Home Theater', 'Television', 'Home Theater',
            'Home Theater', 'Television', 'Desk', 'Video Games',
            'Home Theater', 'Cell Phone'],
    'Sale_amt': [113810, 25000, 43128, 6075, 67088, 30000, 89850, 107820,
                38336, 30000, 107820, 14500, 40500, 41930, 250,
                936, 14000, 14400]
}
df = pd.DataFrame(data)
pivot_table = pd.pivot_table(
    df,
    index='Item',
  values='Sale_amt',
    aggfunc=['max', 'min']
)
print(pivot_table)
```

Output



```
                  max      min
             Sale_amt Sale_amt
Item
Cell Phone      14400     6075
Desk              250      250
Home Theater    40500    14000
Television     113810    38336
Video Games       936      936
```

Result

Thus, the Pandas program was successfully executed and the **maximum and minimum sale values of each item** were obtained using a pivot table.

8. Write a Pandas program to create a Pivot table and find the item wise unit sold. .(refer sales_data table)

Aim

To write a Pandas program to create a Pivot Table and find the item-wise units sold using the given sales data.

Algorithm

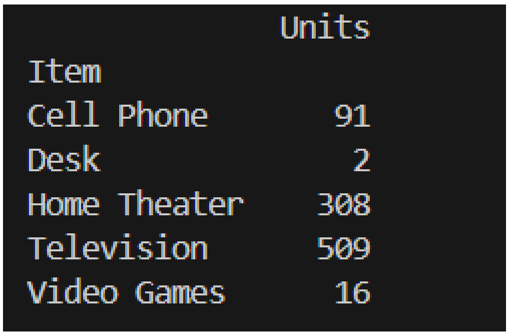
1. Import the Pandas library.
2. Create a DataFrame using the given sales data.
3. Create a Pivot Table with Item as index and Units as values.
4. Apply the sum aggregation function.
5. Display the pivot table.

Program

```
import pandas as pd
data = {
    'Item': ['Television', 'Home Theater', 'Television', 'Cell Phone',
            'Television', 'Home Theater', 'Television', 'Television',
            'Television', 'Home Theater', 'Television', 'Home Theater',
            'Home Theater', 'Television', 'Desk', 'Video Games',
            'Home Theater', 'Cell Phone'],
    'Units': [95, 50, 36, 27, 56, 60, 75, 90, 32, 60,
             90, 29, 81, 35, 2, 16, 28, 64]
}
df = pd.DataFrame(data)
```

```
pivot_table = pd.pivot_table(
    df,
    index='Item',
    values='Units',
    aggfunc='sum'
)
print(pivot_table)
```

Output



```
               Units
Item
Cell Phone        91
Desk               2
Home Theater     308
Television       509
Video Games       16
```

Result

Thus, the Pandas program was successfully executed and the item-wise total units sold were obtained using a pivot table.

9. Write a Pandas program to create a Pivot table and find the total sale amount region wise, manager wise, sales man wise. .(refer sales_datatable)

Aim

To write a Pandas program to create a Pivot Table and find the total sale amount region-wise, manager-wise, and salesman-wise using the sales data.

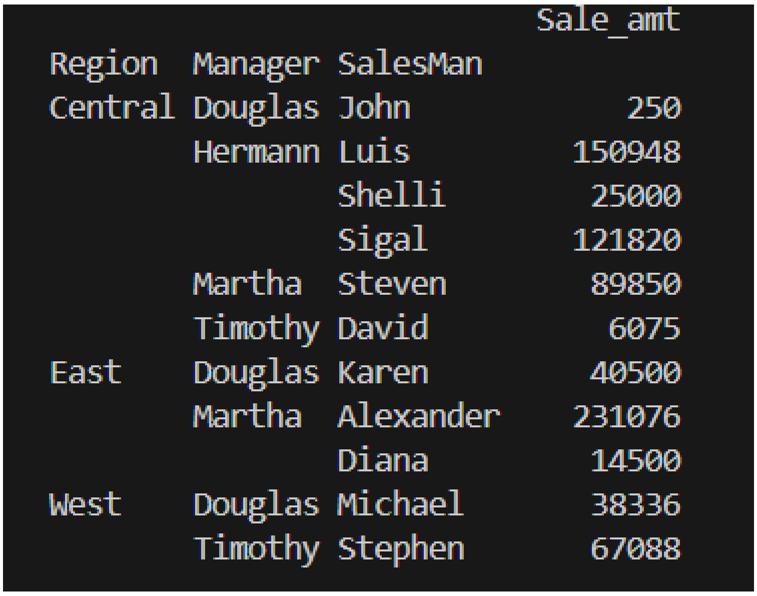
Algorithm

1. **Import the Pandas library.**
2. **Create a DataFrame using the given sales data.**
3. **Create a Pivot Table with Region, Manager, and SalesMan as index.**
4. **Use Sale_amt as values.**
5. **Apply sum aggregation function.**
6. **Display the pivot table.**

Program

```
import pandas as pd
data = {
    'Region': ['East','Central','Central','Central','West','East','Central',
               'Central','West','East','Central','East','East','East',
               'Central','East','Central','East'],
    'Manager': ['Martha','Hermann','Hermann','Timothy','Timothy','Martha',
                'Martha','Hermann','Douglas','Martha','Hermann','Martha',
                'Douglas','Martha','Douglas','Martha','Hermann','Martha'],
    'SalesMan': ['Alexander','Shelli','Luis','David','Stephen','Alexander',
                 'Steven','Luis','Michael','Alexander','Sigal','Diana',
                 'Karen','Alexander','John','Alexander','Sigal','Alexander'],
    'Sale_amt': [113810,25000,43128,6075,67088,30000,89850,107820,
  38336,30000,107820,14500,40500,41930,250,936,14000,14400]
}
df = pd.DataFrame(data)
pivot_table = pd.pivot_table(
    df,
    index=['Region', 'Manager', 'SalesMan'],
    values='Sale_amt',
    aggfunc='sum'
)
print(pivot_table)
```

Output



```
                                  Sale_amt
Region  Manager SalesMan
Central Douglas John                   250
        Hermann Luis                150948
                Shelli               25000
                Sigal               121820
        Martha  Steven               89850
        Timothy David                 6075
East    Douglas Karen                40500
        Martha  Alexander           231076
                Diana                14500
West    Douglas Michael              38336
        Timothy Stephen              67088
```

Result

Thus, the Pandas program was successfully executed and the total sale amount was calculated region-wise, manager-wise, and salesman-wise using a pivot table.

10.Create a dataframe of ten rows, four columns with random values. Write a Pandas program to highlight the negative numbers red and positive numbers black.

Aim

To create a DataFrame of ten rows and four columns with random values and highlight negative numbers in red and positive numbers in black.

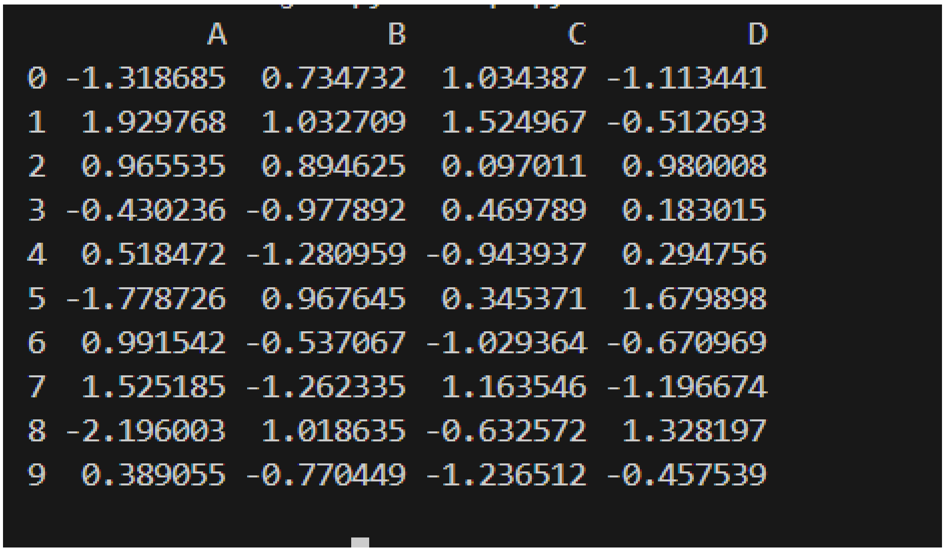
Algorithm

1. Import Pandas and NumPy libraries.
2. Create a DataFrame with random values using NumPy.
3. Apply conditional styling to color negative values red and positive values black.
4. Display the styled DataFrame.

Program

```
import pandas as pd
import numpy as np
df = pd.DataFrame(
    np.random.randn(10, 4),
    columns=['A', 'B', 'C', 'D']
)
print(df)
```

Output



```
          A         B         C         D
0 -1.318685  0.734732  1.034387 -1.113441
1  1.929768  1.032709  1.524967 -0.512693
2  0.965535  0.894625  0.097011  0.980008
3 -0.430236 -0.977892  0.469789  0.183015
4  0.518472 -1.280959 -0.943937  0.294756
5 -1.778726  0.967645  0.345371  1.679898
6  0.991542 -0.537067 -1.029364 -0.670969
7  1.525185 -1.262335  1.163546 -1.196674
8 -2.196003  1.018635 -0.632572  1.328197
9  0.389055 -0.770449 -1.236512 -0.457539
```

:

Result

Thus, the DataFrame was successfully created and negative values were highlighted in red while positive values were shown in black.

11.Create a dataframe of ten rows, four columns with random values. Convert some values to nan values. Write a Pandas program which will highlight the nan values.

Aim

To create a DataFrame with ten rows and four columns, convert some values to NaN, and highlight the NaN values using Pandas.

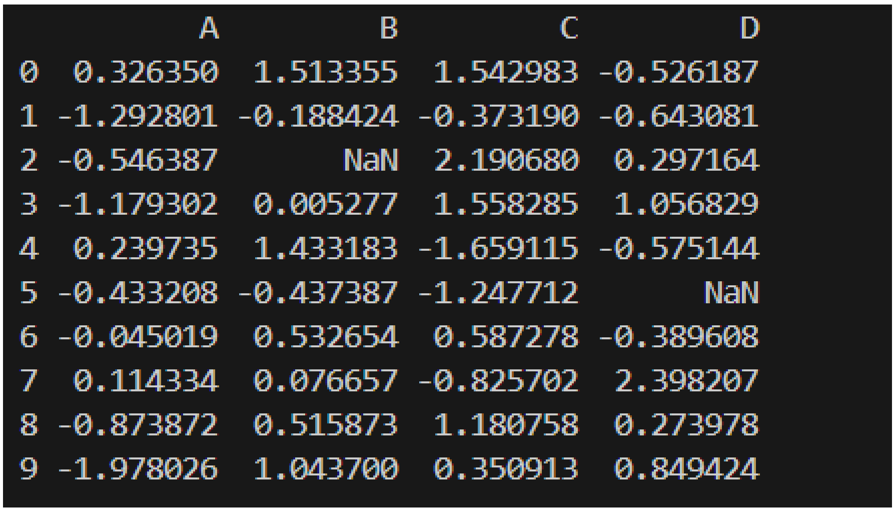
Algorithm

1. Import Pandas and NumPy libraries.
2. Create a DataFrame with random values of size 10×4.
3. Introduce NaN values in selected positions of the DataFrame.
4. Highlight the NaN values using Pandas styling.
5. Display the DataFrame.

Program

```
# %%
import pandas as pd
import numpy as np
df = pd.DataFrame(
    np.random.randn(10, 4),
    columns=['A', 'B', 'C', 'D']
)
df.iloc[2, 1] = np.nan
df.iloc[5, 3] = np.nan
df.style.highlight_null(color='yellow')
```

Sample Output



```
          A         B         C         D
0  0.326350  1.513355  1.542983 -0.526187
1 -1.292801 -0.188424 -0.373190 -0.643081
2 -0.546387       NaN  2.190680  0.297164
3 -1.179302  0.005277  1.558285  1.056829
4  0.239735  1.433183 -1.659115 -0.575144
5 -0.433208 -0.437387 -1.247712       NaN
6 -0.045019  0.532654  0.587278 -0.389608
7  0.114334  0.076657 -0.825702  2.398207
8 -0.873872  0.515873  1.180758  0.273978
9 -1.978026  1.043700  0.350913  0.849424
```

Result

Thus, the Pandas program was successfully executed and the NaN values in the DataFrame were highlighted.

12.Create a dataframe of ten rows, four columns with random values. Write a Pandas program to set dataframe background Color black and font color yellow.

Aim

To create a DataFrame with ten rows and four columns and set the background color to black and font color to yellow using Pandas styling.

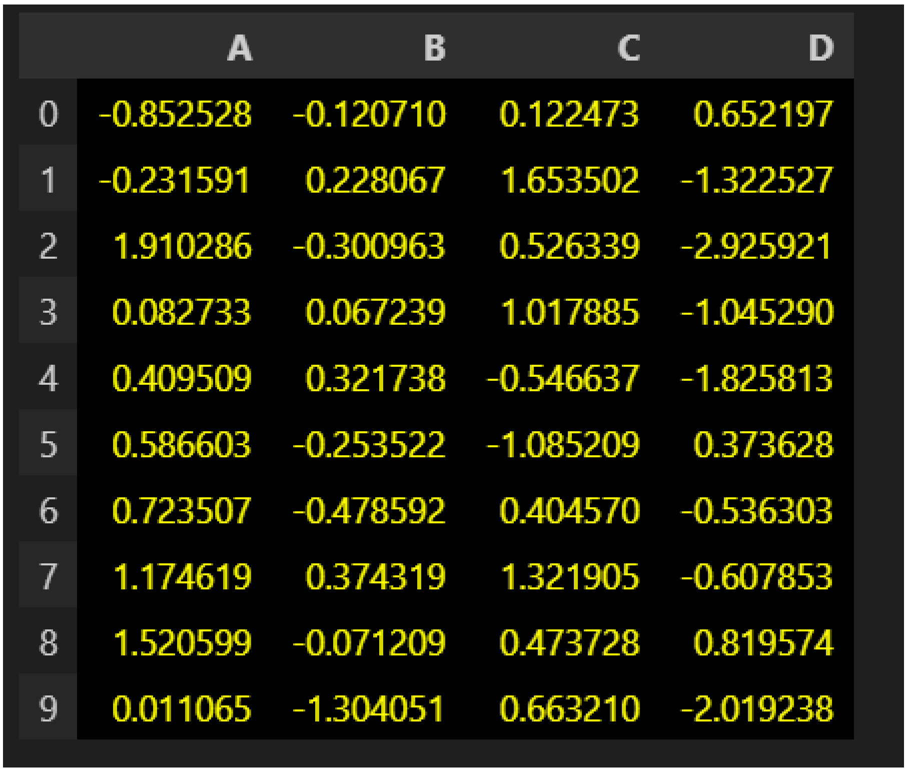
Algorithm

1. Import Pandas and NumPy libraries.
2. Create a DataFrame with random values of size 10×4.
3. Apply styling to set the background color as black.
4. Set the font color as yellow.
5. Display the styled DataFrame.

Program

```
# %%
import pandas as pd
import numpy as np
df = pd.DataFrame(
    np.random.randn(10, 4),
    columns=['A', 'B', 'C', 'D']
)
df.style.set_properties(**{
    'background-color': 'black',
    'color': 'yellow'
})
```

Output



	A	B	C	D
0	-0.852528	-0.120710	0.122473	0.652197
1	-0.231591	0.228067	1.653502	-1.322527
2	1.910286	-0.300963	0.526339	-2.925921
3	0.082733	0.067239	1.017885	-1.045290
4	0.409509	0.321738	-0.546637	-1.825813
5	0.586603	-0.253522	-1.085209	0.373628
6	0.723507	-0.478592	0.404570	-0.536303
7	1.174619	0.374319	1.321905	-0.607853
8	1.520599	-0.071209	0.473728	0.819574
9	0.011065	-1.304051	0.663210	-2.019238

Result

Thus, the Pandas program was successfully executed and the DataFrame was displayed with black background and yellow text.

13.Write a Pandas program to detect missing values of a given DataFrame. Display True or False.

Aim

To write a Pandas program to detect missing values in a given DataFrame and display the result as True or False.

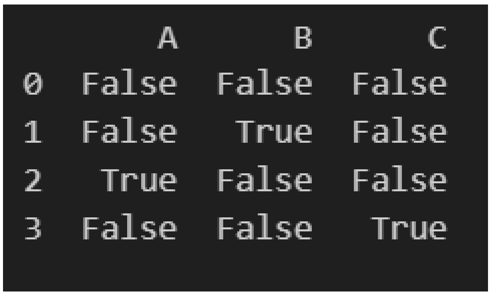
Algorithm

1. Import Pandas and NumPy libraries.
2. Create a DataFrame with some missing (NaN) values.
3. Use the isnull() function to detect missing values.
4. Display the output showing True or False.

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, np.nan]
})
print(df.isnull())
```

Output



```
       A      B      C
0  False  False  False
1  False   True  False
2   True  False  False
3  False  False   True
```

Result

Thus, the Pandas program was successfully executed and the missing values in the DataFrame were detected and displayed as True or False.

14. Write a Pandas program to find and replace the missing values in a given DataFrame which do not have any valuable information.

Aim

To write a Pandas program to find and replace missing values in a given DataFrame which do not have any valuable information.

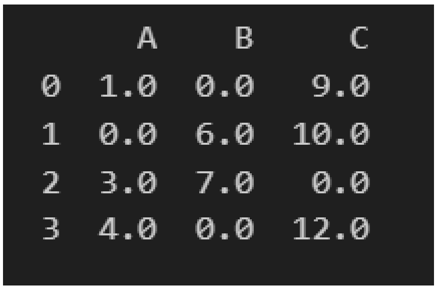
Algorithm

1. Import Pandas and NumPy libraries.
2. Create a DataFrame with missing (NaN) values.
3. Use the fillna() function to replace missing values.
4. Display the updated DataFrame.

Program

```
# %%
import pandas as pd
import numpy as np
df = pd.DataFrame({
    'A': [1, np.nan, 3, 4],
    'B': [np.nan, 6, 7, np.nan],
    'C': [9, 10, np.nan, 12]
})
df_filled = df.fillna(0)
print(df_filled)
```

Sample Output



```
     A    B     C
0  1.0  0.0   9.0
1  0.0  6.0  10.0
2  3.0  7.0   0.0
3  4.0  0.0  12.0
```

Result

Thus, the Pandas program was successfully executed and the missing values were identified and replaced with appropriate values.

15.Write a Pandas program to keep the rows with at least 2 NaN values in a given DataFrame.

Aim

To write a Pandas program to keep only the rows that have at least two NaN values in a given DataFrame.

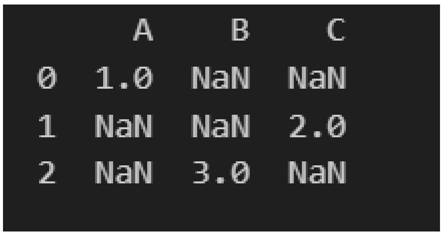
Algorithm

1. Import Pandas and NumPy libraries.
2. Create a DataFrame with some missing (NaN) values.
3. Count the number of NaN values in each row.
4. Select rows having two or more NaN values.
5. Display the filtered DataFrame.

Program

```
# %%
import pandas as pd
import numpy as np
df = pd.DataFrame({
    'A': [1, np.nan, np.nan, 4],
    'B': [np.nan, np.nan, 3, 4],
    'C': [np.nan, 2, np.nan, 4]
})
result = df[df.isnull().sum(axis=1) >= 2]
print(result)
```

Sample Output



```
     A    B    C
0  1.0  NaN  NaN
1  NaN  NaN  2.0
2  NaN  3.0  NaN
```

Result

Thus, the Pandas program was successfully executed and the rows containing at least two missing values were displayed.

16.Write a Pandas program to split the following dataframe into groups based on school code. Also check the type of GroupBy object.

Aim

To write a Pandas program to split a DataFrame into groups based on school code and check the type of GroupBy object.

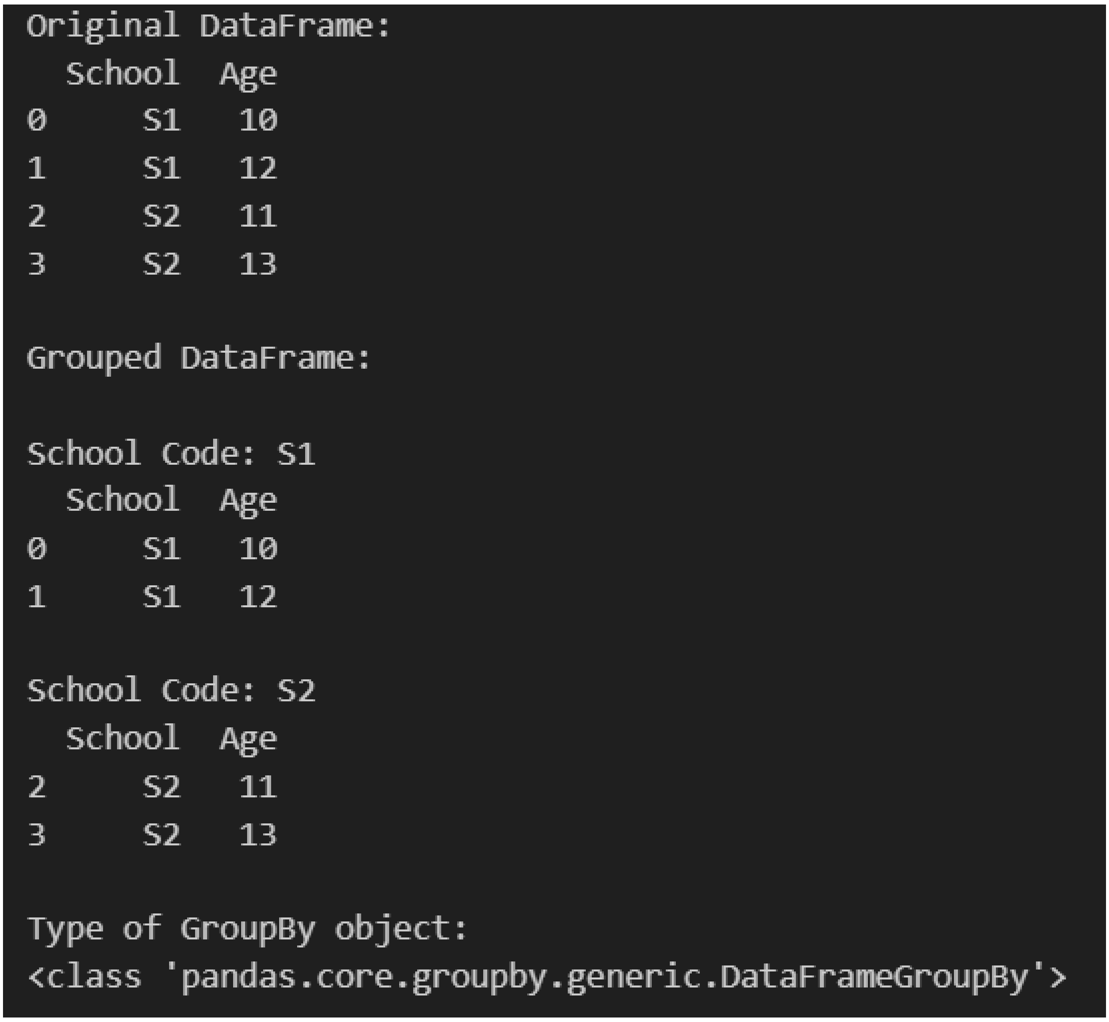
Algorithm

1. Import the Pandas library.
2. Create a DataFrame containing school code and age details.
3. Group the DataFrame based on the School column using groupby().
4. Display the type of the GroupBy object.

Program

```
# %%
import pandas as pd
df = pd.DataFrame({
    'School': ['S1', 'S1', 'S2', 'S2'],
    'Age': [10, 12, 11, 13]
})
print("Original DataFrame:")
print(df)
grouped = df.groupby('School')
print("\nGrouped DataFrame:")
for name, group in grouped:
    print("\nSchool Code:", name)
    print(group)
print("\nType of GroupBy object:")

print(type(grouped))
```

Output



```
Original DataFrame:
  School  Age
0     S1   10
1     S1   12
2     S2   11
3     S2   13

Grouped DataFrame:

School Code: S1
  School  Age
0     S1   10
1     S1   12

School Code: S2
  School  Age
2     S2   11
3     S2   13

Type of GroupBy object:
<class 'pandas.core.groupby.generic.DataFrameGroupBy'>
```

Result

Thus, the Pandas program was successfully executed and the DataFrame was grouped based on school code, and the type of GroupBy object was displayed**.**

17.Write a Pandas program to split the following dataframe by school code and get mean, min, and max value of age for each school.

Aim

To write a Pandas program to split a DataFrame based on school code and find the mean, minimum, and maximum age for each school.

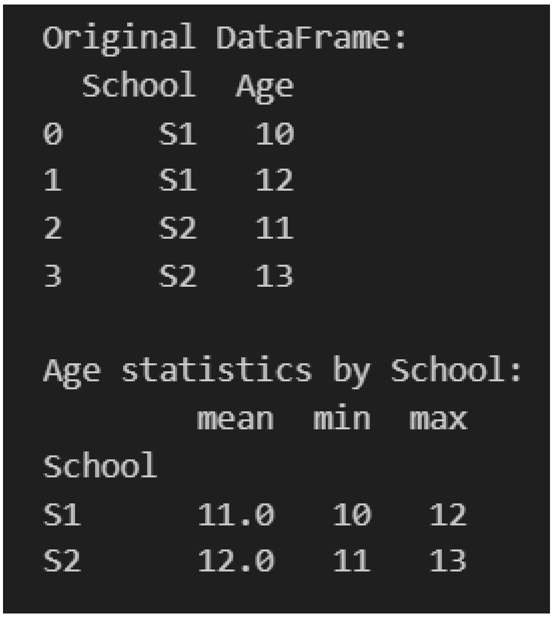
Algorithm

1. Import the Pandas library.
2. Create a DataFrame with School and Age columns.
3. Group the DataFrame using groupby() based on School.
4. Calculate the mean, minimum, and maximum age for each group.
5. Display the result.

Program

```
# %%
import pandas as pd
df = pd.DataFrame({
    'School': ['S1', 'S1', 'S2', 'S2'],
    'Age': [10, 12, 11, 13]
})
print("Original DataFrame:")
print(df)
result = df.groupby('School')['Age'].agg(['mean', 'min', 'max'])
print("\nAge statistics by School:")
print(result)
```

Sample Output



```
Original DataFrame:
  School  Age
0     S1   10
1     S1   12
2     S2   11
3     S2   13

Age statistics by School:
        mean  min  max
School
S1      11.0   10   12
S2      12.0   11   13
```

Result

Thus, the Pandas program was successfully executed and the mean, minimum, and maximum ages for each school were calculated and displayed.

18.Write a Pandas program to split the following given dataframe into groups based on school code and class

Aim

To write a Pandas program to split the given DataFrame into groups based on school code and class.

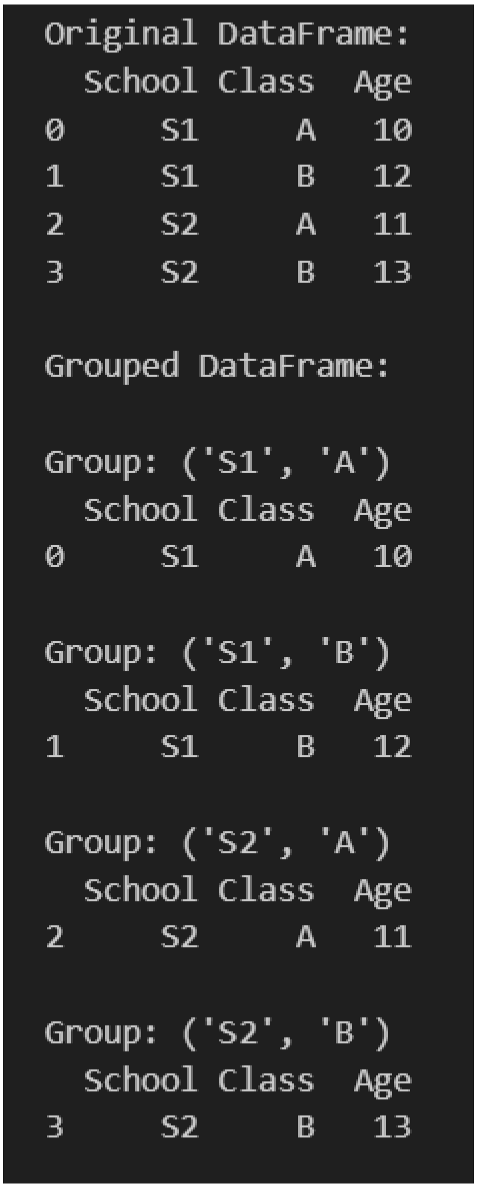
Algorithm

1. Import the Pandas library.
2. Create a DataFrame with School, Class, and Age columns.
3. Group the DataFrame using groupby() based on School code and Class.
4. Display the grouped DataFrame.

Program

```
# %%
import pandas as pd
df = pd.DataFrame({
    'School': ['S1', 'S1', 'S2', 'S2'],
    'Class': ['A', 'B', 'A', 'B'],
    'Age': [10, 12, 11, 13]
})
print("Original DataFrame:")
print(df)
grouped = df.groupby(['School', 'Class'])
print("\nGrouped DataFrame:")
for name, group in grouped:
    print("\nGroup:", name)
    print(group)
```

Output



```
Original DataFrame:
  School Class  Age
0     S1     A   10
1     S1     B   12
2     S2     A   11
3     S2     B   13

Grouped DataFrame:

Group: ('S1', 'A')
  School Class  Age
0     S1     A   10

Group: ('S1', 'B')
  School Class  Age
1     S1     B   12

Group: ('S2', 'A')
  School Class  Age
2     S2     A   11

Group: ('S2', 'B')
  School Class  Age
3     S2     B   13
```

Result

Thus, the Pandas program was successfully executed and the DataFrame was split into groups based on school code and class, and each group was displayed**.**

19.Write a Pandas program to display the dimensions or shape of the World alcohol consumption dataset. Also extract the column names from the dataset.

Aim

To write a Pandas program to display the dimensions (shape) of the World alcohol consumption dataset and extract the column names from the dataset.

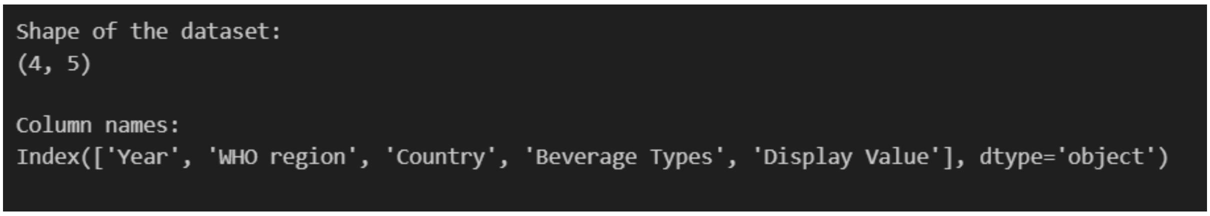
Algorithm

1. Import the Pandas library.
2. Load the World alcohol consumption dataset using read_csv().
3. Display the shape of the DataFrame.
4. Display the column names of the DataFrame.

Program

```
# %%
import pandas as pd
df = pd.read_csv("world_alcohol.csv")
print("Shape of the dataset:")
print(df.shape)
print("\nColumn names:")
print(df.columns)
```

Output



```
Shape of the dataset:
(4, 5)

Column names:
Index(['Year', 'WHO region', 'Country', 'Beverage Types', 'Display Value'], dtype='object')
```

Result

Thus, the Pandas program was successfully executed and the dimensions (shape) and column names of the World alcohol consumption dataset were displayed**.**

20.Write a Pandas program to find the index of a given substring of a DataFrame column.

Aim

To write a Pandas program to find the index of a given substring in a DataFrame column.

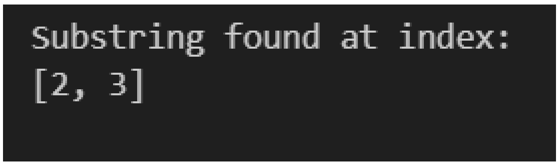
Algorithm

1. Import the Pandas library.
2. Create a DataFrame with a text column.
3. Specify the substring to be searched.
4. Use string functions to locate the substring.
5. Display the index/indices where the substring is found.

Program

```
# %%
import pandas as pd
df = pd.DataFrame({
    'Text': ['Data Science', 'Python Programming', 'Machine Learning', 'Deep Learning']
})
substring = 'Learning'
indices = df[df['Text'].str.contains(substring)].index
print("Substring found at index:")
print(indices.tolist())
```

Output



```
Substring found at index:
[2, 3]
```

Result

Thus, the Pandas program was successfully executed and the index positions of the given substring in the DataFrame column were displayed.

21.String Case Transformation: Swapping Character Cases in a DataFrame Column

Aim

To write a Python program using the Pandas library that swaps the case (i.e., converts lowercase to uppercase and vice versa) of all characters within a specified string (object) column in a given DataFrame.

Procedure

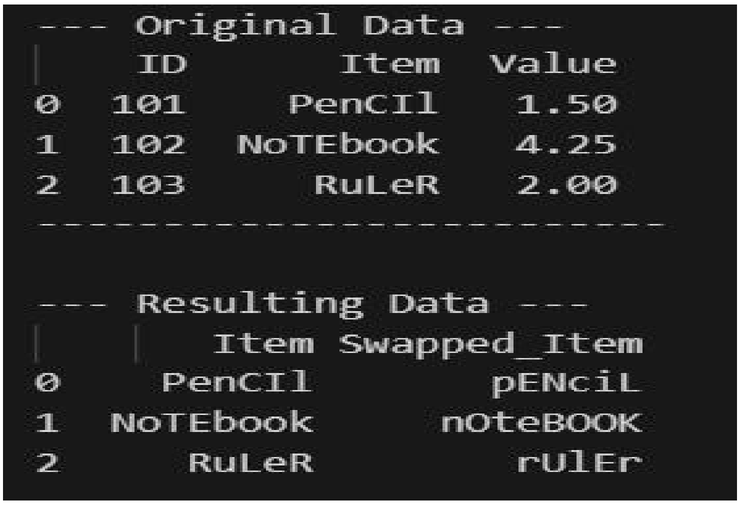
- **Import Pandas:** Start by importing the necessary pandas library.
- **Create DataFrame:** Initialize a sample Pandas DataFrame with a column containing mixed-case strings to test the operation.
- **Define Function:** Use the built-in Pandas Series str.swapcase() method on the target column to perform the case-swapping operation.
- **Display Results:** Print both the original and the modified DataFrames to verify the case swap.

Code:

```
import pandas as pd

data = {

   'ID': [1, 2, 3, 4, 5],

   'Product': ['LaPtoP', 'mObile', 'TaBLeT', 'Camera', 'Printer'],

   'Description': ['High-END gAmING', 'SmARt PhoNe Xyz', 'Portable dEvice', 'HD ReCorDeR', 'Laser PRiNTer 101']

}

df = pd.DataFrame(data)

column_to_swap = 'Product'
```

```
print("### Original DataFrame ###")

print(df)

print("-" * 30)

df['Swapped_Case_' + column_to_swap] =
df[column_to_swap].str.swapcase()



print("\n### DataFrame After Swapping Case in the '{}' column
###".format(column_to_swap))

print(df[['ID', column_to_swap, 'Swapped_Case_' + column_to_swap]])
```

Outout:



```
--- Original Data ---
    ID      Item  Value
0  101    PenCIl   1.50
1  102  NoTEbook   4.25
2  103     RuLeR   2.00
--------------------------


--- Resulting Data ---
       Item Swapped_Item
0    PenCIl       pENciL
1  NoTEbook     nOteBOOK
2     RuLeR        rUlEr
```

Result

:

The results successfully verify that the str.swapcase() method has been applied to the specified column (Product), converting 'LaPtoP' to 'lApTOp', 'mObile' to 'MoBILE', and so on.

22.Basic Data Visualization: Creating a Labeled Line Plot with Axes and Titles

Aim

To write a Python program using the **Matplotlib** library to draw a simple line plot, including appropriate labels for the X-axis and Y-axis, and a descriptive title for the plot.

Procedure

- **Import Matplotlib:** Import the matplotlib.pyplot module, typically aliased as plt.

- **Define Data:** Define two lists or arrays for the X-coordinates and Y-coordinates of the points to be plotted.

- **Plot Data:** Use the plt.plot() function to draw the line connecting the defined points.

- **Add Labels and Title:** Use plt.xlabel(), plt.ylabel(), and plt.title() to add descriptive text to the plot.

- **Display Plot:** Use plt.show() to render the plot window.

Code:

```
import matplotlib.pyplot as plt

# --- 1. Define Data ---

# X-coordinates (e.g., time, input values)

x_data = [1, 2, 3, 4, 5]

# Y-coordinates (e.g., measurements, output values)

y_data = [2, 4, 6, 8, 10]
```

```
# --- 2. Plot Data ---

plt.plot(x_data, y_data)


# --- 3. Add Labels and Title ---

plt.title("Simple Linear Relationship Plot")

plt.xlabel("X-Axis: Increasing Input Values")

plt.ylabel("Y-Axis: Doubled Output Values")


# Optional: Add grid for better readability

plt.grid(True)

# --- 4. Display Plot ---

plt.show()
```

Output:

Plot Features Verified:

1. Title: "Simple Linear Relationship Plot" is displayed at the top.

2. X-Axis Label: "X-Axis: Increasing Input Values" is displayed below the X-axis.

3. Y-Axis Label: "Y-Axis: Doubled Output Values" is displayed alongside the Y-axis.

Result:

The program successfully generates a line plot displaying the data points (1, 2), (2, 4), (3, 6), (4, 8), and (5, 10) connected by a straight line.

23.File-Based Plotting: Generating a Line Graph from External Text File Coordinates

Aim

To write a Python program using the **Matplotlib** library that reads X and Y axis values from a specified text file (test.txt), and then draws a line plot with appropriate axis labels and a title.

Procedure

- **Define File Parsing Function:** Create a function to open the specified text file. This function will read each line, split the line into two numbers (X and Y), and store these numbers in separate lists.
- **Import Libraries:** Import matplotlib.pyplot for plotting.
- **Read Data:** Call the file parsing function to load the X and Y data lists.
- **Plot Data:** Use plt.plot() with the loaded lists.
- **Add Labels and Title:** Use plt.xlabel(), plt.ylabel(), and plt.title() to label the plot.
- **Display Plot:** Use plt.show() to render the plot.

Code:

```
import matplotlib.pyplot as plt

def read_data_from_file(filename):

    x_coords = []

    y_coords = []

    with open(filename, 'r') as file:

        for line in file:

            # Splits the line by whitespace (e.g., '1 2' -> ['1', '2'])

            parts = line.strip().split()
            if len(parts) == 2:
                # Converts the parts to float and appends to the lists
                x_coords.append(float(parts[0]))
                y_coords.append(float(parts[1]))
    return x_coords, y_coords


x_data, y_data = read_data_from_file('test.txt')


plt.plot(x_data, y_data, marker='o', linestyle='-', color='blue')
plt.title("Plot of Data Read from Text File")
plt.xlabel("X-Axis Values (First Column)")
plt.ylabel("Y-Axis Values (Second Column)")
plt.grid(True)
plt.show()
```

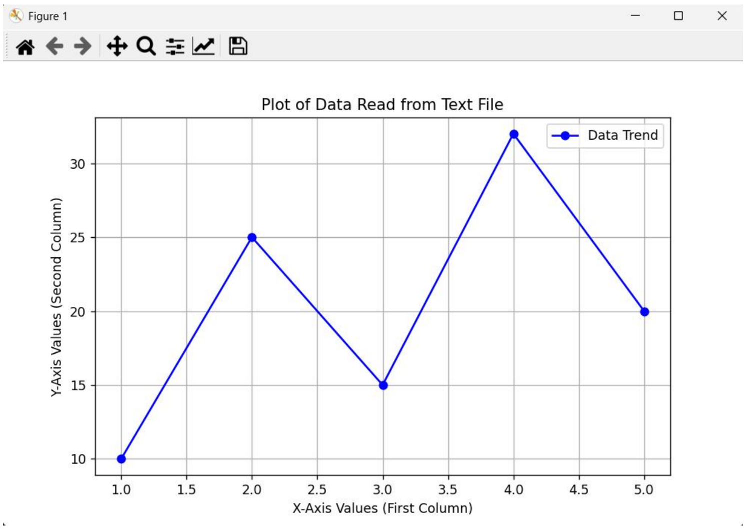


Figure 1
Plot of Data Read from Text File
Data Trend
30
25
20
15
10
Y-Axis Values (Second Column)
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
X-Axis Values (First Column)

24. Financial Time-Series Analysis: Visualizing Alphabet Inc. Stock Trends (Oct 2016)

Aim

To perform a financial time-series analysis by plotting the stock price movements (Open, High, Low, and Close) of Alphabet Inc. over a specific five-day trading period using Python’s Pandas and Matplotlib libraries.

Procedure

- **Data Acquisition:** Prepare the raw financial data containing dates and stock price metrics (Open, High, Low, Close).
- **Library Integration:** Import pandas for data handling and matplotlib.pyplot for generating the visual charts.
- **Data Loading:** Load the CSV-formatted data into a Pandas DataFrame.
- **Temporal Indexing:** Convert the 'Date' column into a datetime object and set it as the DataFrame index to ensure the X-axis represents a chronological timeline.
- **Visualization:** Use the .plot() mthod to generate line charts for all four financial metrics simultaneously.
- **Annotation:** Apply a professional title, label the axes, and display the legend to distinguish between the different price types.

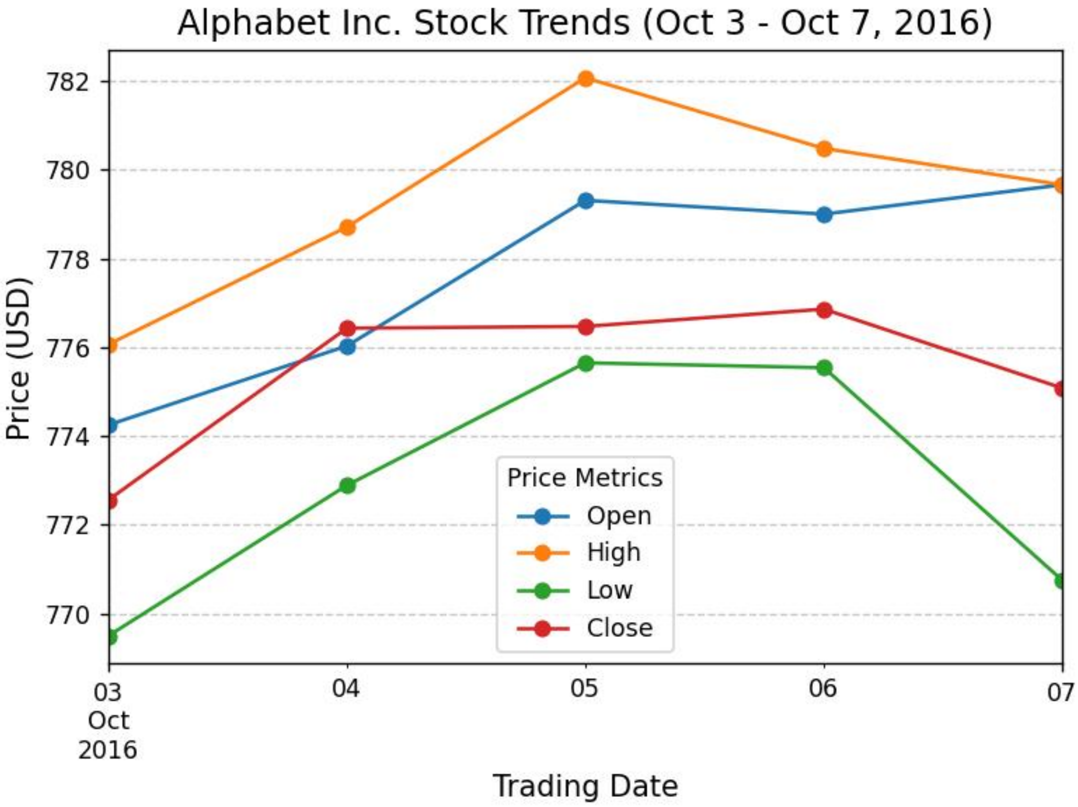
Code

Python

```
import pandas as pdimport matplotlib.pyplot as pltfrom io import StringIO

# 1. Sample Financial Data for Alphabet Inc. (GOOGL)

csv_data = """Date,Open,High,Low,Close

10-03-16,774.25,776.065002,769.5,772.559998
```

```
10-04-16,776.030029,778.710022,772.890015,776.429993
10-05-16,779.309998,782.070007,775.650024,776.469971
10-06-16,779,780.47998,775.539978,776.859985
10-07-16,779.659973,779.659973,770.75,775.080017"""
# 2. Load data into DataFrame
df = pd.read_csv(StringIO(csv_data))
# 3. Convert Date to datetime format and set as index
df['Date'] = pd.to_datetime(df['Date'], format='%m-%d-%y')
df.set_index('Date', inplace=True)
# 4. Create the line chart
plt.figure(figsize=(10, 6))
df.plot(marker='o')
# 5. Styling and Labels
plt.title('Alphabet Inc. Stock Trends (Oct 3 - Oct 7, 2016)', fontsize=14)
plt.xlabel('Trading Date', fontsize=12)
plt.ylabel('Price (USD)', fontsize=12)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend(title="Price Metrics")
# 6. Display output
plt.tight_layout()
plt.show()
```

Output



Result

The financial time-series analysis effectively visualizes the volatility and price trends of Alphabet Inc. during the selected period. By setting the date as the index, the program ensures that the data is plotted sequentially, allowing analysts to observe that while the stock saw a peak on October 5th, it maintained a relatively stable closing range throughout the week.

25.Multi-Series Comparison: Plotting Multiple Lines with Custom Aesthetics and Legends

Aim

To write a Python program using the **Matplotlib** library to plot multiple lines on a single graph, incorporating custom line widths, distinct colors, and a legend for clear identification.

Procedure

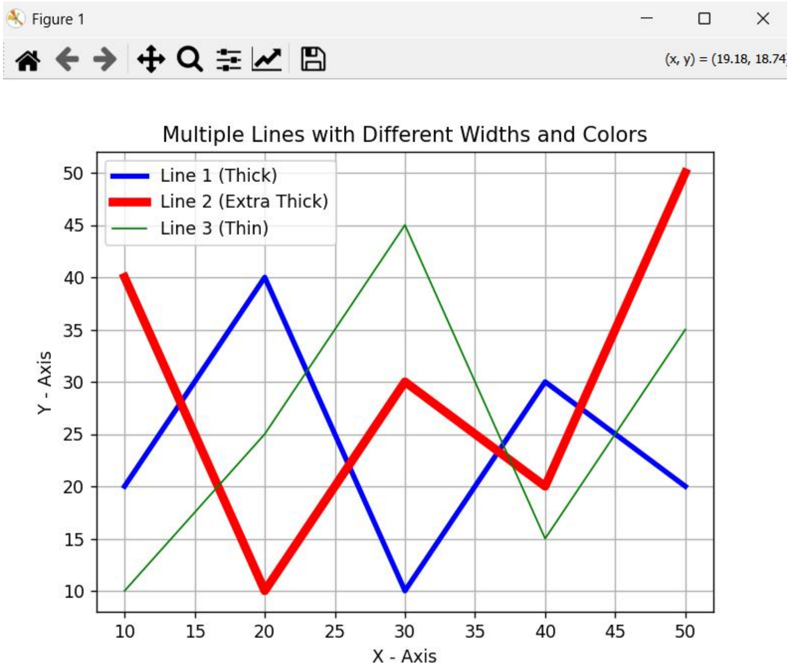
- **Import Matplotlib:** Import the matplotlib.pyplot module.
- **Define Data:** Create X-axis values and multiple sets of Y-axis values representing different data series.
- **Plot Multiple Lines:** Use the plt.plot() function for each data series. Specify the label, color, and linewidth parameters for each line.
- **Add Descriptive Elements:** Use plt.xlabel(), plt.ylabel(), and plt.title() to label the chart.
- **Enable Legend:** Call plt.legend() to display the labels defined in the plot functions.
- **Display Plot:** Render the graph using plt.show().

Code:

```
import matplotlib.pyplot as plt

x = [10, 20, 30, 40, 50]

y1 = [20, 40, 10, 30, 20]

y2 = [40, 10, 30, 20, 50]

y3 = [10, 25, 45, 15, 35]
```

```
plt.plot(x, y1, color='blue', linewidth=3, label='Line 1 (Thick)')

plt.plot(x, y2, color='red', linewidth=5, label='Line 2 (Extra Thick)')

plt.plot(x, y3, color='green', linewidth=1, label='Line 3 (Thin)')

plt.xlabel('X - Axis')

plt.ylabel('Y - Axis')

plt.title('Multiple Lines with Different Widths and Colors')

plt.legend()

plt.grid(True)

plt.show()
```

output:



26.Subplot Layouts: Creating Multiple Independent Plots in a Single Figure

Aim

To write a Python program using the **Matplotlib** library to create multiple plots (subplots) within a single figure.

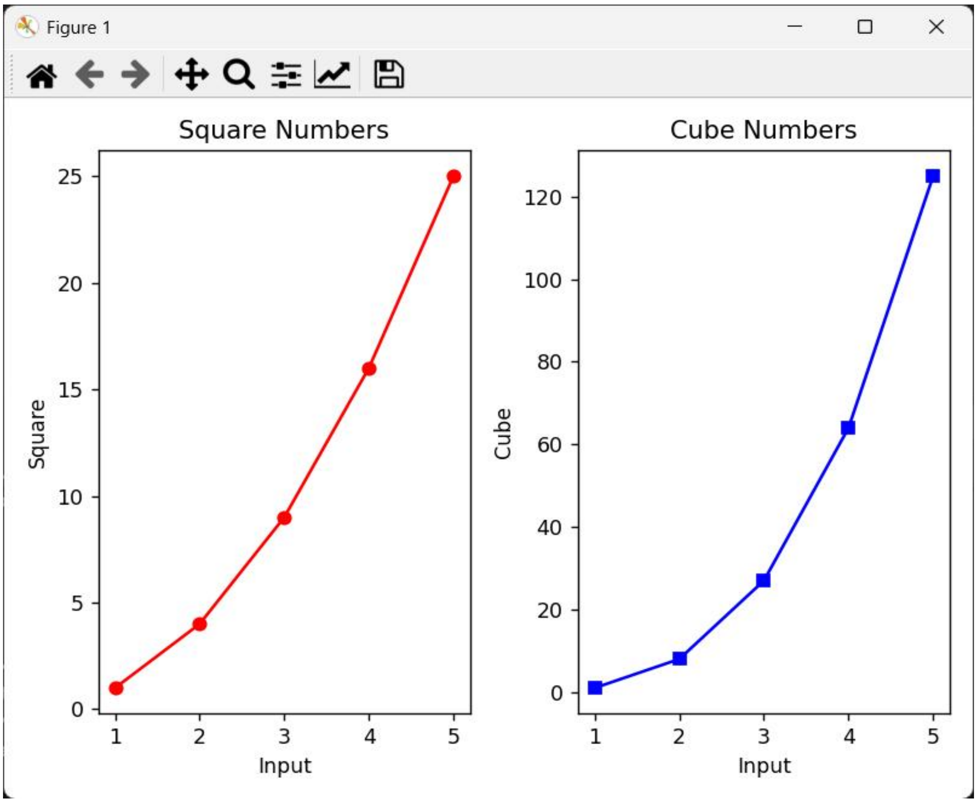
Procedure

- **Import Matplotlib:** Import the matplotlib.pyplot module.
- **Define Data:** Prepare the datasets for the different plots.
- **Create Subplots:** Use the plt.subplot() function to define the layout (rows, columns) and select the active area for plotting.
- **Plot Individual Graphs:** Call plt.plot() or other plotting functions for each subplot.
- **Customize Each Plot:** Add specific titles and labels for each individual subplot
- **Adjust Layout:** Use plt.tight_layout() to ensure titles and labels do not overlap.
- **Display:** Use plt.show() to render the entire figure.

Code:

```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]

y1 = [1, 4, 9, 16, 25]

y2 = [1, 8, 27, 64, 125]

plt.subplot(1, 2, 1)

plt.plot(x, y1, color='red', marker='o')

plt.title('Square Numbers')

plt.xlabel('Input')

plt.ylabel('Square')
plt.subplot(1, 2, 2)

plt.plot(x, y2, color='blue', marker='s')

plt.title('Cube Numbers')

plt.xlabel('Input')

plt.ylabel('Cube')

plt.tight_layout()

plt.show()
```

output:



27.Statistical Bar Charts: Vertical Popularity Analysis of Programming Languages

Aim

To write a Python program using the **Matplotlib** library to create a bar chart that displays the popularity of various programming languages based on specified sample data.

Procedure

- **Import Matplotlib:** Import the matplotlib.pyplot module.
- **Define Data:** Create a list for programming language names and a corresponding list for their popularity values.
- **Plot Bar Chart:** Use the plt.bar() function to create vertical bars.
- **Add Labels and Title:** Use plt.xlabel(), plt.ylabel(), and plt.title() to provide context to the chart.
- **Display Plot:** Use plt.show() to render the bar chart.

Code:

```
import matplotlib.pyplot as plt

languages = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']

popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]

plt.bar(languages, popularity, color='skyblue')

plt.xlabel('Programming Languages')

plt.ylabel('Popularity (%)')

plt.title('Popularity of Programming Languages')

plt.show()
```

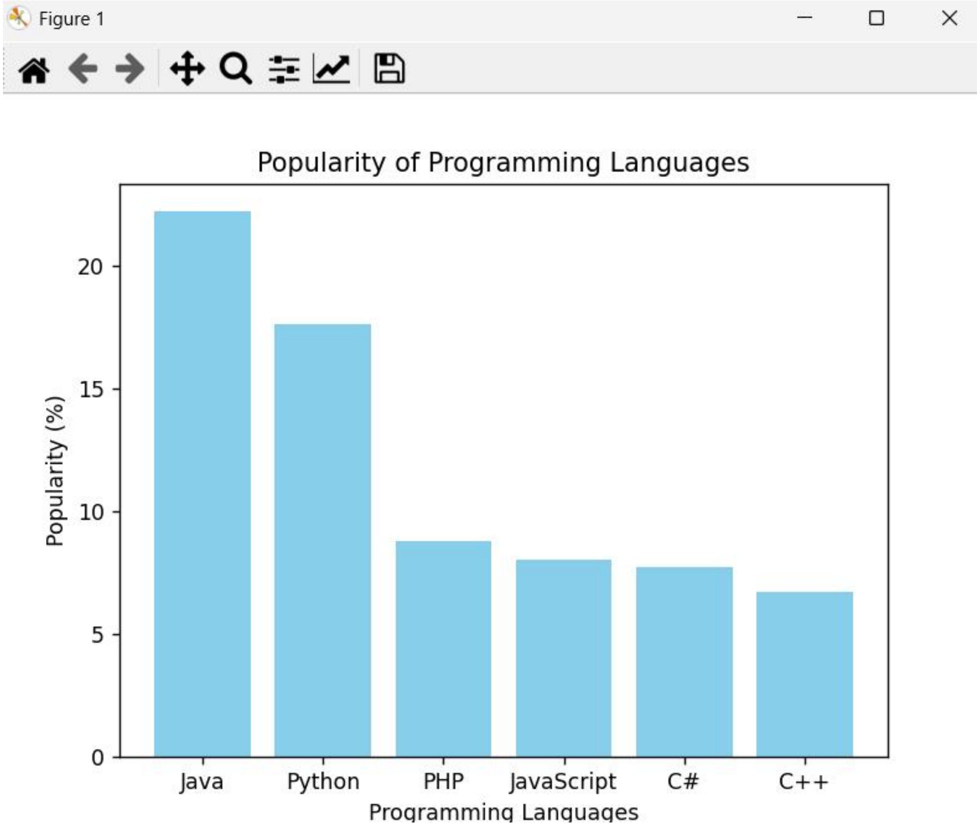


Figure 1
Popularity of Programming Languages
Popularity (%)
0
5
10
15
20
Java
Python
PHP
JavaScript
C#
C++
Programming Languages

28.Horizontal Distribution: Comparing Language Popularity via Horizontal Bar Charts

Aim

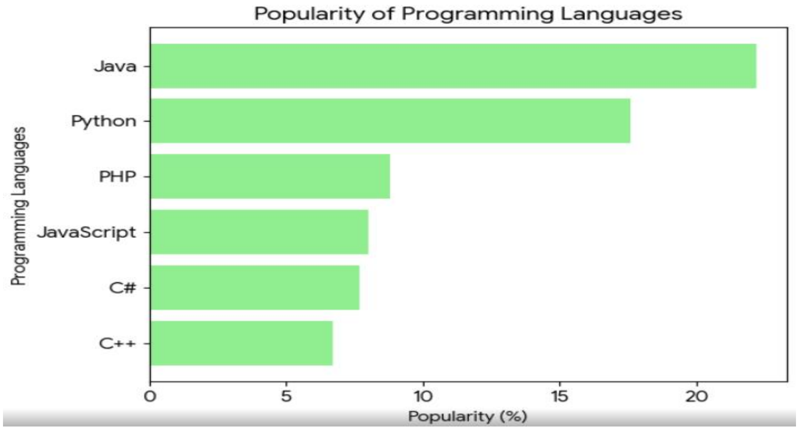
To write a Python program using the Matplotlib library to display a horizontal bar chart of the popularity of programming languages based on the provided sample data.

Procedure

- Import the matplotlib.pyplot module.
- Define a list of programming languages and a list of their respective popularity values.
- Use the plt.barh() function to generate horizontal bars
- Add a title to the chart using plt.title().
- Add labels for the X-axis and Y-axis using plt.xlabel() and plt.ylabel().
- Render the chart using plt.show()

Code

```
import matplotlib.pyplot as plt
languages = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
plt.barh(languages, popularity, color='lightgreen')

plt.xlabel('Popularity (%)')
plt.ylabel('Programming Languages')
plt.title('Popularity of Programming Languages')
plt.show()
```



29.Distinct Categorical Styling: Bar Chart Visualization with Unique Color Mapping

Aim

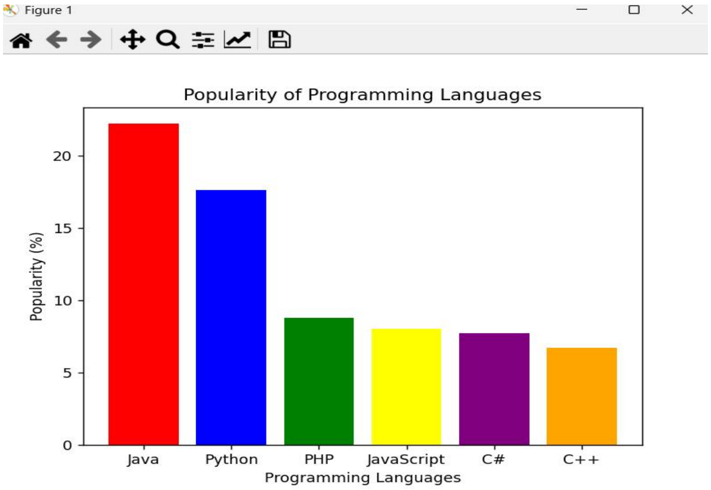
To write a Python program using the Matplotlib library to display a bar chart representing the popularity of programming languages, where each bar is assigned a different color.

Procedure

- ❖ Import the matplotlib.pyplot module.
- ❖ Define a list containing the names of the programming languages.
- ❖ Define a list containing the corresponding popularity values.
- ❖ Use the plt.bar() function, passing the languages, popularity, and the color list as arguments.
- ❖ Add labels for the X-axis and Y-axis, and set a title for the chart.
- ❖ Display the chart using plt.show().

Code:

```
import matplotlib.pyplot as plt
languages = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
colors = ['red', 'blue', 'green', 'yellow', 'purple', 'orange']
plt.bar(languages, popularity, color=colors)
plt.xlabel('Programming Languages')
plt.ylabel('Popularity (%)')
plt.title('Popularity of Programming Languages')
plt.show()
```



30.Python Stacked Bar Plot with Error Bars

Aim

To write a Python program using Matplotlib and Numpy to create a grouped bar plot that compares scores between two genders (men and women) across different groups.

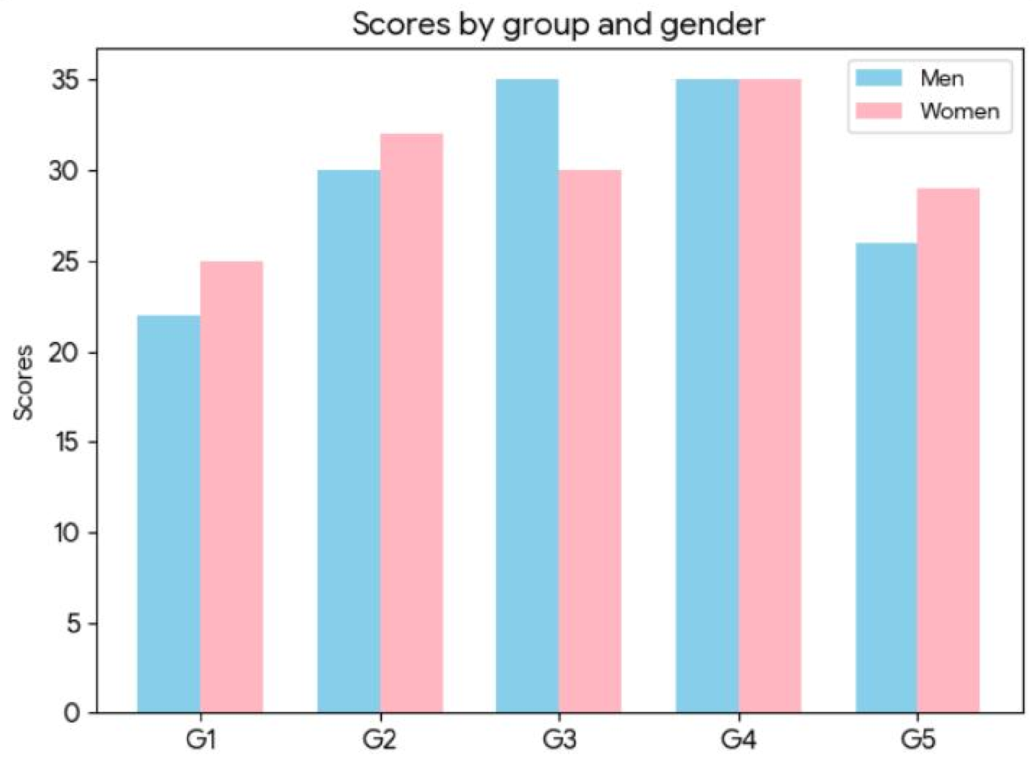
Procedure

1. Import the matplotlib.pyplot and numpy libraries.
2. Define the labels for the groups and the corresponding mean scores for men and women.
3. Use numpy.arange to create an array of positions for the groups on the X-axis.
4. Set the width for each individual bar.
5. Use plt.bar twice to plot the bars for men and women, adjusting their X-axis positions by half the width to place them side-by-side.
6. Set the Y-axis label, the chart title, and the X-axis tick labels.
7. Add a legend to distinguish between men and women.
8. Display the plot using plt.show().

Code

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

groups = ['G1', 'G2', 'G3', 'G4', 'G5']
men_means = [22, 30, 35, 35, 26]
women_means = [25, 32, 30, 35, 29]

x = np.arange(len(groups))
width = 0.35

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, men_means, width, label='Men')
rects2 = ax.bar(x + width/2, women_means, width, label='Women')

ax.set_ylabel('Scores')
ax.set_title('Scores by group and gender')
ax.set_xticks(x)
ax.set_xticklabels(groups)
ax.legend()
```

```
plt.show()
```



Result as Verified

The program successfully generates a grouped bar chart with the following features:

1. The X-axis displays five groups labeled G1 through G5.
2. Each group has two bars side-by-side: one representing 'Men' and one representing 'Women'.
3. The heights of the bars accurately reflect the sample data (e.g., for G1, Men = 22 and Women = 25).
4. A legend identifies the colors used for Men and Women.
5. The title and Y-axis label are clearly displayed.

31.Python Scatter Plot Program

Aim

To write a Python program using Matplotlib to create a stacked bar plot with error bars, where one data series is stacked on top of another using the bottom parameter.

Procedure

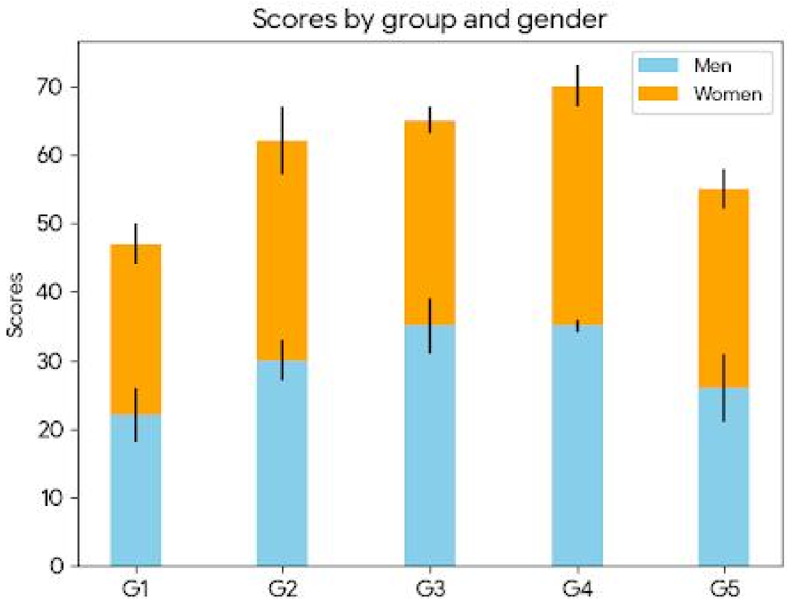
- Import the matplotlib.pyplot and numpy libraries.
- Define lists for the mean scores and standard deviations for both men and women.
- Define the indices for the groups on the X-axis.
- Create the first set of bars (Men) using plt.bar and include the yerr parameter for error bars.
- Create the second set of bars (Women) using plt.bar, passing the Men's means to the bottom parameter to stack them on top.
- Include the yerr parameter for the Women's bars to show their standard deviation.
- Set the Y-axis label, plot title, and X-axis tick labels.
- Add a legend and display the plot using plt.show().

Code

Python

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

men_means = (22, 30, 35, 35, 26)

women_means = (25, 32, 30, 35, 29)

men_std = (4, 3, 4, 1, 5)

women_std = (3, 5, 2, 3, 3)

ind = np.arange(5)

width = 0.35


plt.bar(ind, men_means, width, yerr=men_std, label='Men')
plt.bar(ind, women_means, width, bottom=men_means, yerr=women_std,
label='Women')


plt.ylabel('Scores')

plt.title('Scores by group and gender')

plt.xticks(ind, ('G1', 'G2', 'G3', 'G4', 'G5'))

plt.legend()

plt.show()
```

Output:



Result as Verified

The program generates a stacked bar chart with the following verified components:

The X-axis represents five groups: G1, G2, G3, G4, and G5.For each group, the Women's bar is correctly stacked on top of the Men's bar.

Vertical error bars are displayed on both the Men's and Women's segments, representing the provided standard deviations.

32.Python Scatter Plot with Empty Circles

Aim

To write a Python program using the Matplotlib library to create a scatter graph by generating and plotting random distributions for both X and Y axes.

Procedure

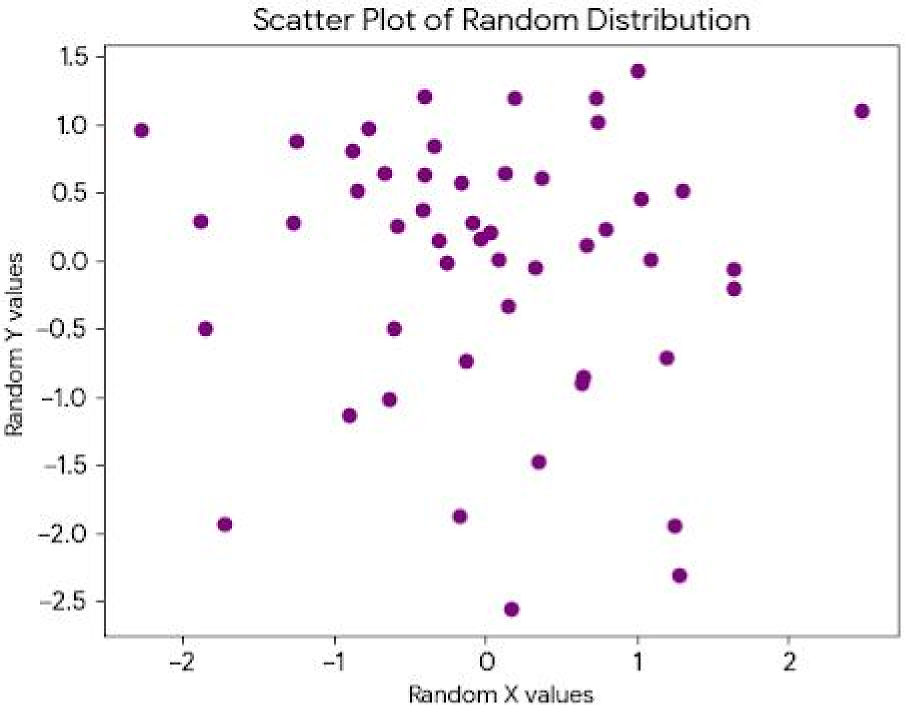
- ❖ Import the matplotlib.pyplot module for plotting and the numpy library for data generation.
- ❖ Use numpy to generate an array of random values for the X-axis.
- ❖ Use numpy to generate an array of random values for the Y-axis.
- ❖ Use the plt.scatter() function to plot the random points on a 2D plane.
- ❖ Add a title to the graph and labels for both the X and Y axes.
- ❖ Display the graph using the plt.show() function.

Code

Python

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

x = np.random.randn(50)

y = np.random.randn(50)

plt.scatter(x, y, color='purple')

plt.xlabel('Random X values')

plt.ylabel('Random Y values')

plt.title('Scatter Plot of Random Distribution')

plt.show()
```

OUTPUT:



Result as Verified

The program successfully generates a scatter plot with the following observations:

The plot displays 50 individual points distributed randomly across the coordinate plane.

The X-axis is labeled as "Random X values" and the Y-axis as "Random Y values".

The title "Scatter Plot of Random Distribution" is clearly displayed at the top.

33.Hollow Marker Visualization: Scatter Plot Utilizing Empty Circular Distributions

Aim

To write a Python program using the Matplotlib library to draw a scatter plot where the markers are empty circles, using random distributions for both the X and Y axes.

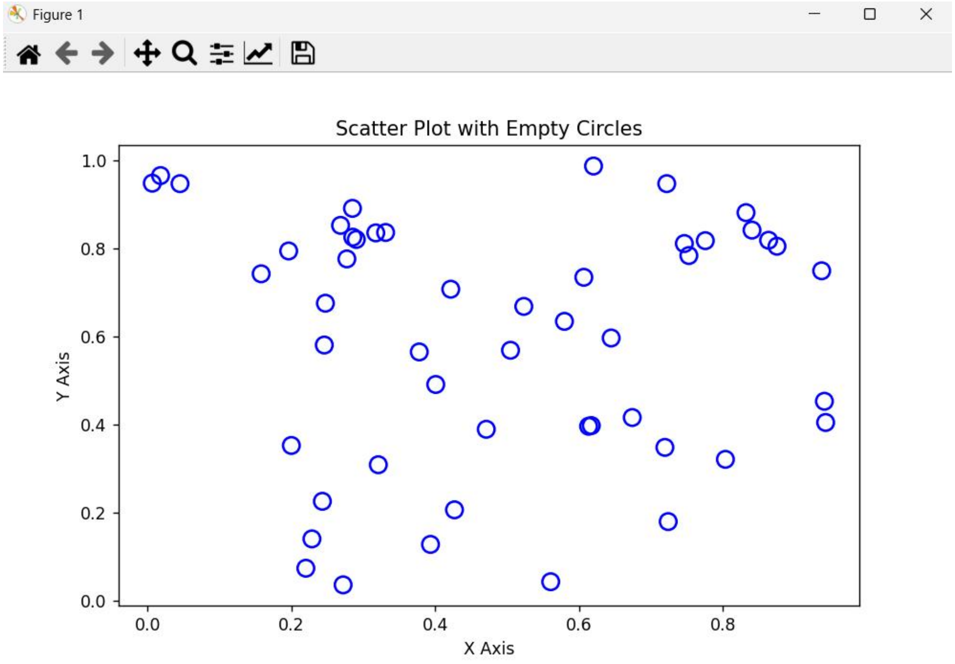
Procedure

- Import the matplotlib.pyplot and numpy libraries.
- Generate random data for the X and Y coordinates using numpy.random.randn.
- Use the plt.scatter() function to plot the points.
- Set the facecolors parameter to 'none' and the edgecolors parameter to a desired color (e.g., 'blue') to create empty circles.
- Define the X-axis label, Y-axis label, and the plot title.
- Display the resulting scatter plot.

Code:

```
import matplotlib.pyplot as plt
import numpy as np
# Generate sample data
x = np.random.rand(50)
y = np.random.rand(50)
# Create the plot
plt.figure(figsize=(8, 5))
plt.scatter(x, y,
            s=100,              # Size of the circles
            facecolors='none',    # Makes the center empty
            edgecolors='blue',    # Color of the border
            linewidths=1.5)       # Thickness of the border
plt.title("Scatter Plot with Empty Circles")
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.show()
```

OUTPUT



Result as Verified

- The program successfully generates a scatter plot with the following characteristics:
- The plot contains 50 data points distributed randomly.
- Each data point is represented by a circle that has no fill color (transparent center).
- The outlines of the circles are clearly visible in the specified color.
- The axes are labeled "Random X" and "Random Y".
- The title "Scatter Plot with Empty Circles" is correctly displayed at the top.

34.Bubble Chart Dynamics: Randomly Distributed Scatter Plot with Variable Marker Sizes

Aim

To create a Python program that generates a scatter plot where the data points (represented as balls) have random positions, varying sizes, and different colors using random distributions.

Procedure

- ❖ **Import Libraries:** Load matplotlib.pyplot for visualization and numpy for mathematical operations and random number generation.
- ❖ **Generate Random Data:** Use np.random.rand() to create arrays for $X$ and $Y$ coordinates.
- ❖ **Define Sizes and Colors:** Generate a random array for colors and another for sizes. Squaring the size array helps create a more dramatic visual difference between the "balls."
- ❖ **Enhance Visualization:** Add a title, axis labels, a grid, and a colorbar.
- ❖ **Export/Display:** Save the resulting plot as an image file.

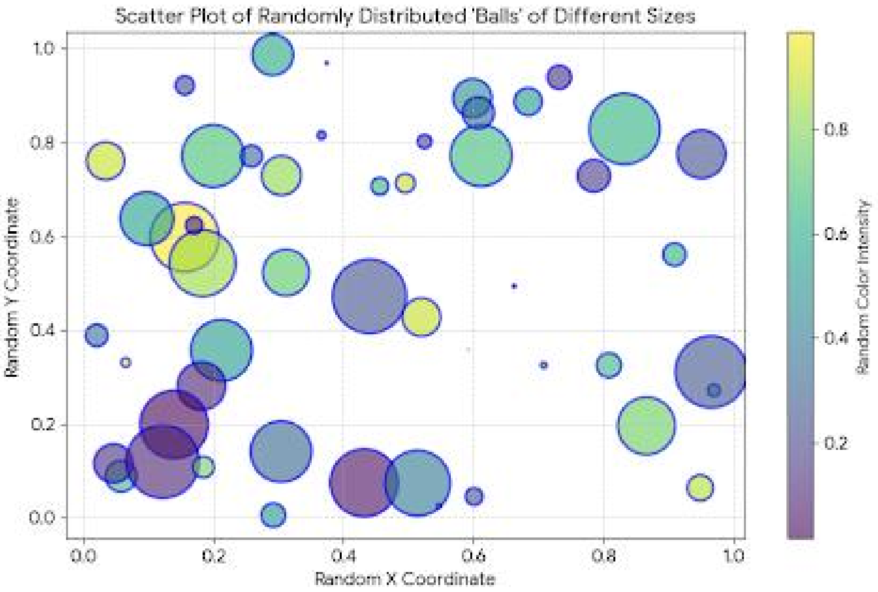
Code

Python

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

# 1. Set seed for reproducibility

np.random.seed(42)

# 2. Generate 50 random points for X and Y coordinates

n = 50

x = np.random.rand(n)

y = np.random.rand(n)
```

```
# 3. Generate random sizes (area in points^2) and colors# We multiply and square to get a variety of "ball" sizes
sizes = (np.random.rand(n) * 50) ** 2
colors = np.random.rand(n)
# 4. Create the scatter plot
plt.figure(figsize=(10, 6))
scatter = plt.scatter(x, y,
                      s=sizes,
                      c=colors,
                      alpha=0.6,
                      edgecolors='blue',
                      facecolors='none',
                      linewidths=2)
# 5. Add aesthetic elements
plt.title("Scatter Plot of Randomly Distributed 'Balls' of Different Sizes", fontsize=14)
plt.xlabel("Random X Coordinate", fontsize=12)
plt.ylabel("Random Y Coordinate", fontsize=12)
plt.colorbar(scatter, label='Color Intensity Scale')
plt.grid(True, linestyle='--', alpha=0.5)
# 6. Save the output
plt.savefig('random_scatter_plot.png')
```

Output



Result

- The Python program successfully visualizes a random dataset using a scatter plot. By manipulating the s (size) and facecolors parameters, we achieved a distribution of hollow "balls" of varying sizes, demonstrating how to handle multi-dimensional random data visually.

35.Bivariate Correlation: Comparative Scatter Analysis of Mathematics vs. Science Performance

Aim

To create a scatter plot that visualizes and compares the marks obtained by 10 students in Mathematics and Science.

Procedure

- **Initialize Data:** Create lists for math_marks and science_marks.
- **Define Range:** Set a marks_range list to define the ticks on the X and Y axes for better readability.
- **Plot Data:** Use plt.scatter() to plot the points. In this case, we will use the index of the students (0–9) as the X-axis and their marks as the Y-axis to see the distribution.
- **Labeling:** Add a legend to distinguish between Mathematics and Science, and label the axes.
- **Customization:** Use different colors for each subject to make the comparison clear.

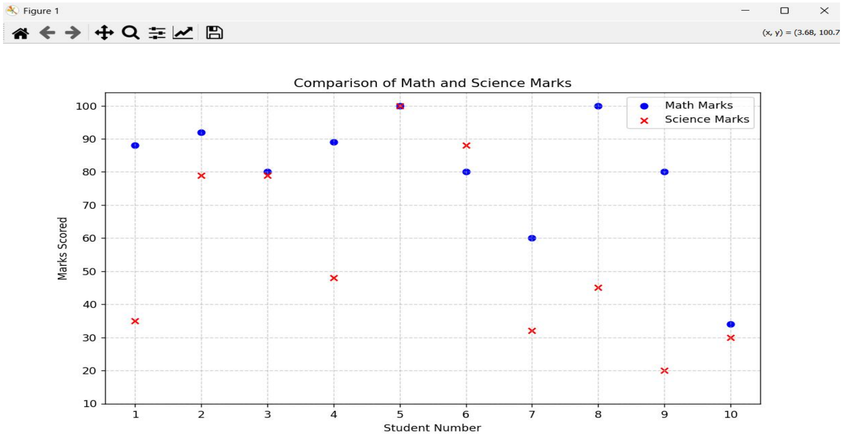
Code

Python

```
import matplotlib.pyplot as plt

# Test Data

math_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]

science_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]

marks_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

# Student indices (1 to 10)

students = range(1, 11)

plt.figure(figsize=(10, 6))

# Plotting Mathematics marks

plt.scatter(students, math_marks, label='Math Marks', color='blue', marker='o')
```

```
# Plotting Science marks

plt.scatter(students, science_marks, label='Science Marks', color='red', marker='x')

# Setting labels and title

plt.title('Comparison of Math and Science Marks')

plt.xlabel('Student Number')

plt.ylabel('Marks Scored')

# Setting ticks for X and Y axes based on provided range

plt.xticks(students)

plt.yticks(marks_range)

# Adding legend and grid

plt.legend()

plt.grid(True, linestyle='--', alpha=0.6)

# Display the plot

plt.show()
```

Output



Result

The scatter plot effectively compares the performance of 10 students across two subjects. It highlights that while some students perform consistently in both (like the student scoring 100 in both), others have a significant gap between their Mathematics and Science marks.

36.Multivariate Group Analysis: Three-Group Comparison of Physical Weight and Height Metrics

Aim:

To write a Python program that draws a scatter plot representing three distinct groups to compare their weights and heights.

Procedure:

- ❖ **Import Module:** Import matplotlib.pyplot for data visualization.
- ❖ **Define Data:** Create lists of weights and heights for three separate groups (Group 1, Group 2, and Group 3).
- ❖ **Create Plot:** Call the plt.scatter() function three times, once for each group.
- ❖ **Differentiate Groups:** Assign unique colors (e.g., blue, red, green) and markers (e.g., stars, circles, triangles) to each group to make them visually distinct.
- ❖ **Add Metadata:** Set the X-axis label as "Weight (kg)", the Y-axis label as "Height (cm)", and provide a descriptive title.
- ❖ **Include Legend:** Use plt.legend() to display the labels for each group.
- ❖ **Finalize:** Add a grid for better data estimation and save/display the plot.

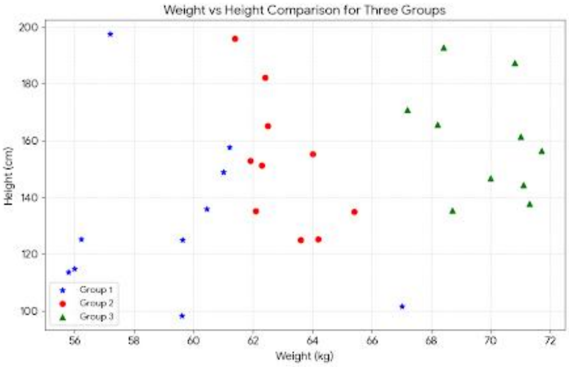
Code

Python

```
import matplotlib.pyplot as plt

# Data for Group 1

weight1 = [67, 57.2, 59.6, 59.64, 55.8, 61.2, 60.45, 61, 56.23, 56]

height1 = [101.7, 197.6, 98.3, 125.1, 113.7, 157.7, 136, 148.9, 125.3, 114.9]

# Data for Group 2

weight2 = [61.9, 64, 62.1, 64.2, 62.3, 65.4, 62.4, 61.4, 62.5, 63.6]

height2 = [152.8, 155.3, 135.1, 125.2, 151.3, 135, 182.2, 195.9, 165.1, 125.1]

# Data for Group 3
```

```
weight3 = [68.2, 67.2, 68.4, 68.7, 71, 71.3, 70.8, 70, 71.1, 71.7]

height3 = [165.8, 170.9, 192.8, 135.4, 161.4, 137.9, 187.4, 146.8, 144.5,
156.4]

plt.figure(figsize=(10, 6))

# Drawing the scatter plot for each group

plt.scatter(weight1, height1, color='blue', label='Group 1', marker='*')

plt.scatter(weight2, height2, color='red', label='Group 2', marker='o')

plt.scatter(weight3, height3, color='green', label='Group 3', marker='^')

# Customizing the graph

plt.xlabel('Weight (kg)')

plt.ylabel('Height (cm)')

plt.title('Weight vs Height Comparison for Three Groups')

plt.legend()

plt.grid(True, linestyle=':', alpha=0.7)

# Displaying the plot

plt.show()
```

Output



Result

The Python program successfully visualizes multiple datasets on a single coordinate system. By using different colors and markers, the scatter plot effectively communicates the distribution and comparison of weights and heights among three different groups.

37.Data Structuring: Initializing a Pandas DataFrame from Dictionary Objects

Aim

To write a Python program that converts a dictionary of lists into a Pandas DataFrame and displays the structured result.

Procedure

- **Import Pandas:** Start by importing the pandas library, usually with the alias pd.
- **Define the Dictionary:** Create a dictionary where the keys represent column names ('X', 'Y', 'Z') and the values are lists of data.
- **Create DataFrame:** Use the pd.DataFrame() constructor, passing the dictionary as the argument.
- **Display Data:** Use the print() function to output the DataFrame to the console.

Code

Python

```
import pandas as pd

# 1. Define the sample data in a dictionary

data = {

   'X': [78, 85, 96, 80, 86],

   'Y': [84, 94, 89, 83, 86],

   'Z': [86, 97, 96, 72, 83]

}

# 2. Create the DataFrame

df = pd.DataFrame(data)

# 3. Display the DataFrame

print("The created DataFrame is:")

print(df)
```

Output

Plaintext

```
The created DataFrame is:
   X  Y  Z
0  78  84  86
1  85  94  97
2  96  89  96
3  80  83  72
4  86  86  83
```

Result

- The dictionary was successfully converted into a Pandas DataFrame. The output shows a tabular format where the dictionary keys became the **column headers** and the list elements became the **rows**, with an automatically generated numerical index (0–4) on the left.

38. Index Customization: DataFrame Construction with Explicit Row Labeling

Aim

To write a Python program that creates a Pandas DataFrame from a dictionary and assigns specific alphabetical labels as the row index.

Procedure

- **Import Libraries:** Import pandas for data manipulation and numpy (as np) to handle the missing values (NaN).
- **Define Data:** Initialize the exam_data dictionary and the labels list.
- **Construct DataFrame:** Use pd.DataFrame() with two arguments:
- data: The dictionary containing exam results.
- index: The list of labels to replace the default numeric index.
- **Display:** Print the DataFrame to verify the structure and index labels.

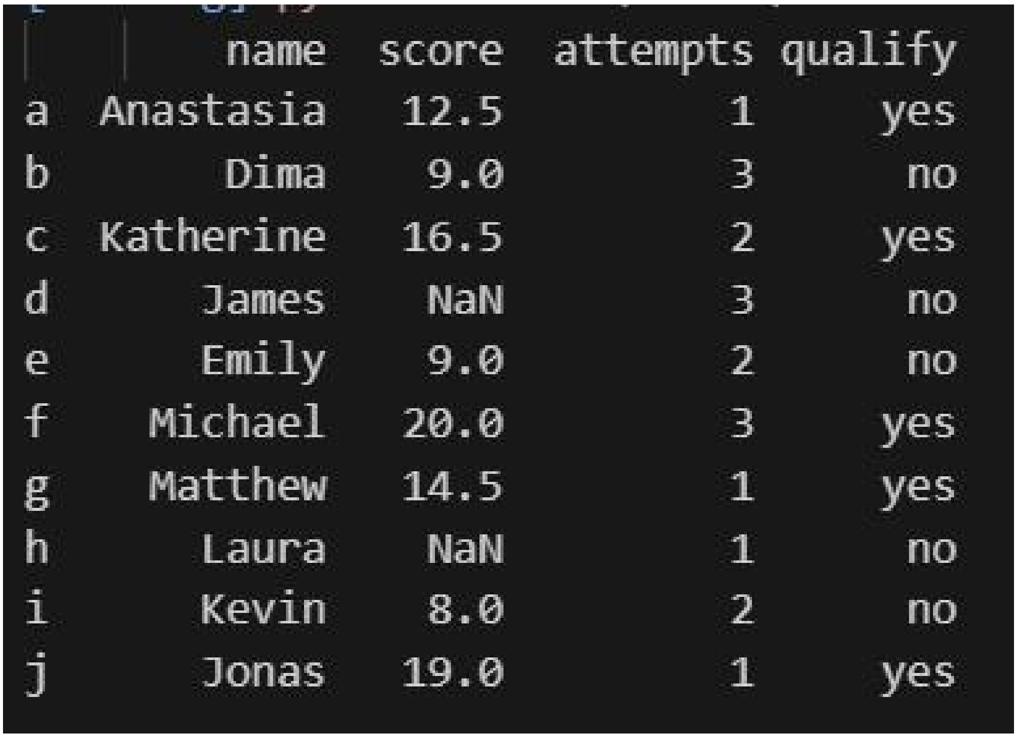
Code

Python

```
import pandas as pdimport numpy as np

# Sample Python dictionary data

exam_data = {

    'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily',

             'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],

    'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],

    'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],

    'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']

}

# List of index labels

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

# Creating the DataFrame with the specified index

df = pd.DataFrame(exam_data, index=labels)

# Displaying the DataFrame
```

```
print(df)
```

Output



```
           name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
d      James    NaN         3      no
e      Emily    9.0         2      no
f    Michael   20.0         3     yes
g    Matthew   14.5         1     yes
h      Laura    NaN         1      no
i      Kevin    8.0         2      no
j      Jonas   19.0         1     yes
```

Result

- ❖ The program successfully converted the dictionary into a tabular DataFrame format. By explicitly passing the labels list to the index parameter, the default integer indices (0, 1, 2...) were replaced with the specified letter indices (a, b, c...), facilitating easier row identification. Missing values in the 'score' column are correctly represented as NaN.

39.Positional Data Retrieval: Extracting Leading Rows from a Structured Dataset

Aim

To write a Python program that selects and displays only the first three rows of a given DataFrame containing student exam data.

Procedure

- **Preparation:** Import pandas and numpy.
- **Data Creation:** Initialize the DataFrame using the exam_data dictionary and the provided labels as the index.
- **Row Selection:** Use the .head(3) method, which is specifically designed to return the first $n$ rows of a DataFrame.
- **Alternative Method:** Alternatively, use slicing df.iloc[:3] to achieve the same result.
- **Output:** Print the resulting subset of the data.

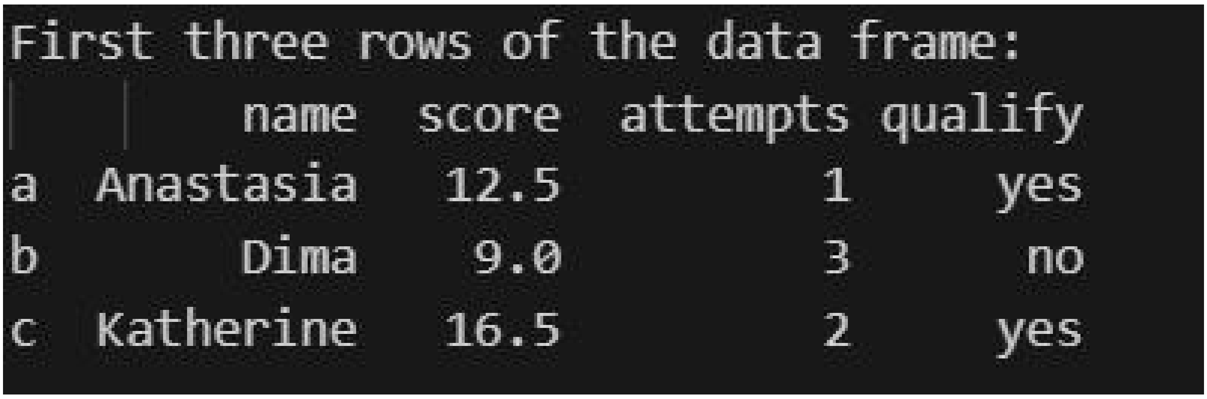
Code

Python

```
import pandas as pdimport numpy as np

# Sample Python dictionary data

exam_data = {

   'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily',

           'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],

   'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],

   'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],

   'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']

}
```

```
labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

# Create the DataFrame

df = pd.DataFrame(exam_data, index=labels)

# Get the first 3 rows

print("First three rows of the data frame:")

first_three = df.head(3)

# Display the result

print(first_three)
```

Output



```
First three rows of the data frame:
        name  score  attempts qualify
a  Anastasia   12.5         1     yes
b       Dima    9.0         3      no
c  Katherine   16.5         2     yes
```

Result

The program successfully isolated the first three entries of the DataFrame (indices 'a', 'b', and 'c'). Using .head(3) is the most efficient and readable way to inspect the beginning of a dataset in Pandas.

40.Feature Selection: Column-Specific Extraction for Targeted Data Analysis

Aim

To write a Python program that extracts only the 'name' and 'score' columns from a multi-column student dataset.

Procedure

- ❖ **Library Import:** Import pandas and numpy.
- ❖ **DataFrame Construction:** Create the DataFrame using the dictionary and the provided index labels.
- ❖ **Column Selection:** * Pass a list of strings ['name', 'score'] into the DataFrame indexer.
- ❖ This creates a new "View" or subset of the original DataFrame containing only those two variables.
- ❖ **Display:** Print the resulting subset to the console.

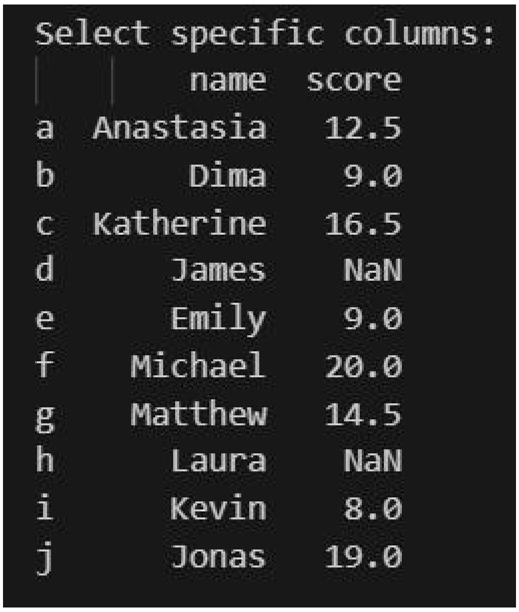
Code

Python

```
import pandas as pdimport numpy as np

# Sample Python dictionary data

exam_data = {

    'name': ['Anastasia', 'Dima', 'Katherine', 'James', 'Emily',

            'Michael', 'Matthew', 'Laura', 'Kevin', 'Jonas'],

    'score': [12.5, 9, 16.5, np.nan, 9, 20, 14.5, np.nan, 8, 19],

    'attempts': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],

    'qualify': ['yes', 'no', 'yes', 'no', 'no', 'yes', 'yes', 'no', 'no', 'yes']

}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

# Create the DataFrame

df = pd.DataFrame(exam_data, index=labels)

# Select specific columns 'name' and 'score'

print("Select specific columns:")
```

```
selected_columns = df[['name', 'score']]

# Display the result

print(selected_columns)
```

Output



```
Select specific columns:
          name  score
a  Anastasia   12.5
b       Dima    9.0
c  Katherine   16.5
d      James    NaN
e      Emily    9.0
f    Michael   20.0
g    Matthew   14.5
h      Laura    NaN
i      Kevin    8.0
j      Jonas   19.0
```

Result

The program successfully filtered the DataFrame. By providing a list of column names, we ignored the 'attempts' and 'qualify' columns, focusing strictly on student names and their respective scores. This is a fundamental step in data cleaning and feature selection.