Qp

Open In Colab

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

AIM: to write a pandas program to select distinct department id

PROCEDURE:

Algorithm: Select Distinct Department IDs using Pandas

Import the Pandas library.

Define or load your dataset into a Pandas DataFrame.

If the data is not already in a DataFrame, you can use Pandas to read it from a file or create one manually.

Use the DataFrame and Pandas functions to extract distinct department IDs:

Use the 'unique' method on the specific column containing department IDs to obtain an array of unique values.

Display or use the resulting distinct department IDs as needed.

Close any open resources, if necessary.

```
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'
   ],
   'MANAGER_ID': [200, 201, 114, 203, 121, 103, 204, 145, 100, 108, 205, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0],
   'LOCATION_ID': [1700, 1800, 1700, 2400, 1500, 1400, 2700, 2500, 1700, 1700, 1700,
1700, 1700, 1700, 1700, 1700, 1700, 1700, 1700, 1700, 1700, 1700, 1700, 1700, 1700,
1700, 1700],
}

df = pd.DataFrame(data)
distinct_department_ids = df['DEPARTMENT_ID'].unique()
print("Distinct Department IDs:")
print(distinct_department_ids)
```

```
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: Successfully implemented.

2. Write a Pandas program to display the ID for those employees who did two or more jobs in the past.
AIM: to write a Pandas program to display the ID for those employees who did two or more jobs in the past.
PROCEDURE: Algorithm: Identify Employees with Multiple Job Positions using Pandas

Import the Pandas library.
Define or load your employee data into a Pandas DataFrame.
If the data is not already in a DataFrame, you can use Pandas to read it from a file or create one manually.
Use the groupby method to group the data by 'EMPLOYEE_ID'.
Use the nunique method to count the number of unique 'JOB_ID' values for each employee.
Filter the results to select employees with two or more distinct job positions.
Display or use the resulting employee IDs with multiple job positions as needed.
Close any open resources, if necessary.

```
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_counts = df.groupby('EMPLOYEE_ID')['JOB_ID'].nunique()
employees_with_multiple_jobs = job_counts[job_counts >= 2]
print("Employee IDs with two or more jobs:")
print(employees_with_multiple_jobs)
```

```
Employee IDs with two or more jobs:
EMPLOYEE_ID
101    2
176    2
200    2
Name: JOB_ID, dtype: int64
```

RESULT: Successfully implemented.

3. Write a Pandas program to display the details of jobs in descending sequence on job title.
AIM:to display the details of jobs in descending sequence on job title
PROCEDURE:Algorithm: Create and Sort a DataFrame by a Specific Column

Import the Pandas library as pd.
Define your dataset as a dictionary or load your data from a source.
Create a DataFrame using the dataset with pd.DataFrame.
Specify the column by which you want to sort the DataFrame, and set the ascending parameter to control the sorting order (True for ascending, False for descending).
Use the sort_values function on the DataFrame to sort it based on the chosen column and order.
Print the sorted DataFrame to view the results.

```
import pandas as pd

# Sample data for the DataFrame
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)
df_sorted = df.sort_values(by='JOB_TITLE', ascending=False)
print(df_sorted)
```

```
      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: Successfully implemented.

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

PROCEDURE:Algorithm: Create a Line Chart for Alphabet Inc. Stock Prices

Import the Pandas and Matplotlib libraries.
Define or load your dataset into a Pandas DataFrame.
Convert the 'Date' column to datetime format using pd.to_datetime.
Define the start_date and end_date for the date range you want to analyze.
Filter the DataFrame to include only the rows that fall within the specified date range using boolean indexing.
Create a line chart by plotting the 'Date' column on the x-axis and the 'Close' column on the y-axis using plt.plot.
Customize the chart by setting the title, xlabel, ylabel, and legend.
Display the line chart using plt.show().

```
import pandas as pd

data = {
   'Date': [
      '01-04-2020', '02-04-2020', '03-04-2020', '06-04-2020', '07-04-2020','08-04-2020',
'09-04-2020', '13-04-2020', '14-04-2020', '15-04-2020','16-04-2020', '17-04-2020',
'20-04-2020', '21-04-2020', '22-04-2020','23-04-2020', '24-04-2020', '27-04-2020',
'28-04-2020'  ],
   'Open': [
      1122.0, 1098.26001, 1119.015015, 1138.0, 1221.0, 1206.5, 1224.079956,1209.180054,
1245.089966, 1245.609985, 1274.099976, 1284.849976,1271.0, 1247.0, 1245.540039,
1271.550049, 1261.170044, 1296.0, 1287.930054 ],
   'High': [1129.689941, 1126.859985, 1123.540039, 1194.660034, 1225.0,
1219.069946,1225.569946, 1220.51001, 1282.069946, 1280.459961, 1279.0,
1294.430054,1281.599976, 1254.27002, 1285.613037, 1293.310059, 1280.400024,
1296.150024,1288.050049],
  'Low': [
    1097.449951, 1096.400024, 1079.810059, 1130.939941, 1182.22998,
1188.160034,1196.734985, 1187.598022, 1236.930054, 1240.400024, 1242.619995,
1271.22998,1261.369995, 1209.709961, 1242.0, 1265.670044, 1249.449951, 1269.0,
1232.199951],
  'Close': [
    1105.619995, 1120.839966, 1097.880005, 1186.920044, 1186.51001,
1210.280029,1211.449951, 1217.560059, 1269.22998, 1262.469971, 1263.469971,
1283.25,1266.609985, 1216.339966, 1263.209961, 1276.310059, 1279.310059,
1275.880005,1233.670044],
  'Adj Close': [
    1105.619995, 1120.839966, 1097.880005, 1186.920044, 1186.51001,
1210.280029,1211.449951, 1217.560059, 1269.22998, 1262.469971, 1263.469971,
1283.25,1266.609985, 1216.339966, 1263.209961, 1276.310059, 1279.310059,
1275.880005,1233.670044],
  'Volume': [
    2343100, 1964900, 2313400, 2664700, 2387300, 1975100, 2175400, 1739800,
2470400,1671700, 2518100, 1949000, 1695500, 2153000, 2093100, 1566200, 1640400,
1600600,2951300 ]
}

df = pd.DataFrame(data)
df['Date'] = pd.to_datetime(df['Date'])
start_date = '2020-04-01'
end_date = '2020-05-01'
filtered_df = df[(df['Date'] >= start_date) & (df['Date'] <= end_date)]
plt.figure(figsize=(12, 6))
plt.plot(filtered_df['Date'], filtered_df['Close'], label='Closing Price')
plt.title('Alphabet Inc. Stock Prices (2022)')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.grid(True)
plt.show()
```

```
<ipython-input-10-e780b659facf>:20: UserWarning: Parsing dates in DD/MM/YYYY format
when dayfirst=False (the default) was specified. This may lead to inconsistently parsed
dates! Specify a format to ensure consistent parsing.
  df['Date'] = pd.to_datetime(df['Date'])
```

RESULT: Successfully implemented.

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

PROCEDURE:Algorithm: Create a Bar Chart in Pandas and Matplotlib

Import the Pandas and Matplotlib libraries.
Define or load your dataset into a Pandas DataFrame.
Convert the 'Date' column to datetime format using pd.to_datetime.
Define the start_date and end_date for the date range you want to analyze.
Create a new DataFrame by filtering the data between the specified start_date and end_date using boolean indexing.
Set the 'Date' column as the index of the new DataFrame for time-series data analysis.
Create a figure and set the title, xlabel, and ylabel for the bar chart.
Plot the trading volume data as a bar chart using df2['Volume'].plot(kind='bar').
Display the bar chart using plt.show().

```
import pandas as pd
import matplotlib.pyplot as plt


data = {
    'Date': [
        '01-04-2020', '02-04-2020', '03-04-2020', '06-04-2020', '07-04-2020','08-04-2020',
'09-04-2020', '13-04-2020', '14-04-2020', '15-04-2020','16-04-2020', '17-04-2020',
'20-04-2020', '21-04-2020', '22-04-2020','23-04-2020', '24-04-2020', '27-04-2020',
'28-04-2020'  ],
    'Open': [
        1122.0, 1098.26001, 1119.015015, 1138.0, 1221.0, 1206.5, 1224.079956,1209.180054,
1245.089966, 1245.609985, 1274.099976, 1284.849976,1271.0, 1247.0, 1245.540039,
1271.550049, 1261.170044, 1296.0, 1287.930054 ],
    'High': [1129.689941, 1126.859985, 1123.540039, 1194.660034, 1225.0,
1219.069946,1225.569946, 1220.51001, 1282.069946, 1280.459961, 1279.0,
1294.430054,1281.599976, 1254.27002, 1285.613037, 1293.310059, 1280.400024,
1296.150024,1288.050049],
    'Low': [
        1097.449951, 1096.400024, 1079.810059, 1130.939941, 1182.22998,
1188.160034,1196.734985, 1187.598022, 1236.930054, 1240.400024, 1242.619995,
1271.22998,1261.369995, 1209.709961, 1242.0, 1265.670044, 1249.449951, 1269.0,
1232.199951],
    'Close': [
        1105.619995, 1120.839966, 1097.880005, 1186.920044, 1186.51001,
1210.280029,1211.449951, 1217.560059, 1269.22998, 1262.469971, 1263.469971,
1283.25,1266.609985, 1216.339966, 1263.209961, 1276.310059, 1279.310059,
1275.880005,1233.670044],
    'Adj Close': [
        1105.619995, 1120.839966, 1097.880005, 1186.920044, 1186.51001,
1210.280029,1211.449951, 1217.560059, 1269.22998, 1262.469971, 1263.469971,
1283.25,1266.609985, 1216.339966, 1263.209961, 1276.310059, 1279.310059,
1275.880005,1233.670044],
    'Volume': [
    2343100, 1964900, 2313400, 2664700, 2387300, 1975100, 2175400, 1739800,
2470400,1671700, 2518100, 1949000, 1695500, 2153000, 2093100, 1566200, 1640400,
1600600,2951300 ]
}

df = pd.DataFrame(data)
start_date = pd.to_datetime('2020-4-1')
end_date = pd.to_datetime('2020-4-30')
df['Date'] = pd.to_datetime(df['Date'])
new_df = (df['Date']>= start_date) & (df['Date']<= end_date)
df1 = df.loc[new_df]
df2 = df1.set_index('Date')
plt.figure(figsize=(6,6))
plt.suptitle('Trading Volume of Alphabet Inc. stock,\n01-04-2020 to 30-04-2020', fontsize=16,
color='black')
plt.xlabel("Date",fontsize=12, color='black')
plt.ylabel("Trading Volume", fontsize=12, color='black')
df2['Volume'].plot(kind='bar');
plt.show()
```

```
<ipython-input-13-273e3dea0438>:24: UserWarning: Parsing dates in DD/MM/YYYY format
when dayfirst=False (the default) was specified. This may lead to inconsistently parsed
dates! Specify a format to ensure consistent parsing.
  df['Date'] = pd.to_datetime(df['Date'])
```

RESULT: Successfully implemented.

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 create a scatter plot

PROCEDURE:Algorithm: Create a Scatter Plot in Pandas and Matplotlib

Import the Pandas and Matplotlib libraries.
Define or load your dataset into a Pandas DataFrame.
Convert the 'Date' column to datetime format using pd.to_datetime.
Define the start_date and end_date for the date range you want to analyze.
Create a new DataFrame by filtering the data between the specified start_date and end_date using boolean indexing.
Set the 'Date' column as the index of the new DataFrame for time-series data analysis.
Define the data for the x-axis (stock price) and the y-axis (trading volume).
Create a scatter plot using Matplotlib, specifying the x and y data, marker size (s), and other plot settings.
Add gridlines, a title, and axis labels to the plot for clarity.
Display the scatter plot using plt.show().

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
data = {
   'Date': [
       '01-04-2020', '02-04-2020', '03-04-2020', '06-04-2020', '07-04-2020','08-04-2020',
'09-04-2020', '13-04-2020', '14-04-2020', '15-04-2020','16-04-2020', '17-04-2020',
'20-04-2020', '21-04-2020', '22-04-2020','23-04-2020', '24-04-2020', '27-04-2020',
'28-04-2020'  ],
   'Open': [
       1122.0, 1098.26001, 1119.015015, 1138.0, 1221.0, 1206.5, 1224.079956,1209.180054,
1245.089966, 1245.609985, 1274.099976, 1284.849976,1271.0, 1247.0, 1245.540039,
1271.550049, 1261.170044, 1296.0, 1287.930054 ],
   'High': [1129.689941, 1126.859985, 1123.540039, 1194.660034, 1225.0,
1219.069946,1225.569946, 1220.51001, 1282.069946, 1280.459961, 1279.0,
1294.430054,1281.599976, 1254.27002, 1285.613037, 1293.310059, 1280.400024,
1296.150024,1288.050049],
   'Low': [
       1097.449951, 1096.400024, 1079.810059, 1130.939941, 1182.22998,
1188.160034,1196.734985, 1187.598022, 1236.930054, 1240.400024, 1242.619995,
1271.22998,1261.369995, 1209.709961, 1242.0, 1265.670044, 1249.449951, 1269.0,
1232.199951],
   'Close': [
       1105.619995, 1120.839966, 1097.880005, 1186.920044, 1186.51001,
1210.280029,1211.449951, 1217.560059, 1269.22998, 1262.469971, 1263.469971,
1283.25,1266.609985, 1216.339966, 1263.209961, 1276.310059, 1279.310059,
1275.880005,1233.670044],
   'Adj Close': [
       1105.619995, 1120.839966, 1097.880005, 1186.920044, 1186.51001,
1210.280029,1211.449951, 1217.560059, 1269.22998, 1262.469971, 1263.469971,
1283.25,1266.609985, 1216.339966, 1263.209961, 1276.310059, 1279.310059,
1275.880005,1233.670044],
   'Volume': [
       2343100, 1964900, 2313400, 2664700, 2387300, 1975100, 2175400, 1739800,
2470400,1671700, 2518100, 1949000, 1695500, 2153000, 2093100, 1566200, 1640400,
1600600,2951300 ]
}

df = pd.DataFrame(data)
start_date = pd.to_datetime('2020-4-1')
end_date = pd.to_datetime('2020-9-30')
df['Date'] = pd.to_datetime(df['Date'])
new_df = (df['Date']>= start_date) & (df['Date']<= end_date)
df1 = df.loc[new_df]
df2 = df1.set_index('Date')
x= ['Close']; y = ['Volume']
plt.figure(figsize=[15,10])
df2.plot.scatter(x, y, s=50);
plt.grid(True)
plt.title('Trading Volume/Price of Alphabet Inc. stock,\n01-04-2020 to 30-09-2020',
fontsize=14, color='black')
plt.xlabel("Stock Price",fontsize=12, color='black')
plt.ylabel("Trading Volume", fontsize=12, color='black')
plt.show()
```

```
<ipython-input-14-12e6937ad076>:24: UserWarning: Parsing dates in DD/MM/YYYY format
when dayfirst=False (the default) was specified. This may lead to inconsistently parsed
dates! Specify a format to ensure consistent parsing.
  df['Date'] = pd.to_datetime(df['Date'])
<Figure size 1500x1000 with 0 Axes>
```

RESULT: Successfully implemented.

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 create a Pivot table and find the maximum and minimum sale value of the items
PROCEDURE:Algorithm: Create a Pivot Table in Pandas

Import the Pandas and NumPy libraries.
Define or load your dataset into a Pandas DataFrame.
Use the pivot_table method on the DataFrame to create a pivot table. Specify the following parameters: a. index: The column(s) to use as the index for the pivot table, which represents the rows. b. values: The column(s) to aggregate or perform operations on, which represents the data to analyze. c. aggfunc: The aggregation function(s) to apply to the specified values.
.
Display or print the resulting pivot table to see the aggregated data.

```
import pandas as pd
import numpy as np
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'],

    'OrderDate': ['1-6-18', '1-23-18', '2-9-18', '2-26-18', '3-15-18', '4-1-18', '4-18-18', '5-5-18',
'5-22-18', '6-8-18', '6-25-18', '7-12-18', '7-29-18', '8-15-18', '9-1-18', '9-18-18', '10-5-18',
'10-22-18'],
    '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', 'Hermann',
'Martha', 'Martha'],
    'SalesMan': ['Alexander', 'Shelli', 'Luis', 'David', 'Stephen', 'Alexander', 'Steven', 'Luis',
'Michael', 'Alexander', 'Sigal', 'Diana', 'Karen', 'Alexander', 'John', 'Alexander', 'Sigal',
'Alexander'],
    'Units': [95, 50, 36, 27, 56, 60, 75, 90, 32, 60, 90, 29, 81, 35, 2, 16, 28, 64],
   'Unit_price': [1198.00, 500.00, 1198.00, 225.00, 1198.00, 500.00, 1198.00, 1198.00,
1198.00, 500.00, 1198.00, 500.00, 500.00, 1198.00, 125.00, 58.50, 500.00, 225.00],
   'Sale_amt': [113810.00, 25000.00, 43128.00, 6075.00, 67088.00, 30000.00, 89850.00,
107820.00, 38336.00, 30000.00, 107820.00, 14500.00, 40500.00, 41930.00, 250.00,
936.00, 14000.00, 14400.00]
}
sales_data = pd.DataFrame(data)
print(sales_data)

pivot_table = sales_data.pivot_table(index='Item', values='Sale_amt', aggfunc=['max', 'min'])
print(pivot_table)
```

```
          Item OrderDate   Region  Manager   SalesMan  Units  Unit_price  \
0     Television    1-6-18     East   Martha  Alexander     95      1198.0
1   Home Theater   1-23-18  Central  Hermann     Shelli     50       500.0
2     Television    2-9-18  Central  Hermann       Luis     36      1198.0
3     Cell Phone   2-26-18  Central  Timothy      David     27       225.0
4     Television   3-15-18     West  Timothy    Stephen     56      1198.0
5   Home Theater    4-1-18     East   Martha  Alexander     60       500.0
6     Television   4-18-18  Central   Martha     Steven     75      1198.0
7     Television    5-5-18  Central  Hermann       Luis     90      1198.0
8     Television   5-22-18     West  Douglas    Michael     32      1198.0
9   Home Theater    6-8-18     East   Martha  Alexander     60       500.0
10    Television   6-25-18  Central  Hermann      Sigal     90      1198.0
11  Home Theater   7-12-18     East   Martha      Diana     29       500.0
12  Home Theater   7-29-18     East  Douglas      Karen     81       500.0
13    Television   8-15-18     East   Martha  Alexander     35      1198.0
14          Desk    9-1-18  Central  Douglas       John      2       125.0
15   Video Games   9-18-18     East  Hermann  Alexander     16        58.5
16  Home Theater   10-5-18  Central   Martha      Sigal     28       500.0
17    Cell Phone  10-22-18     East   Martha  Alexander     64       225.0

   Sale_amt
0  113810.0
1   25000.0
2   43128.0
3    6075.0
4   67088.0
5   30000.0
6   89850.0
7  107820.0
8   38336.0
9   30000.0
10 107820.0
11  14500.0
12  40500.0
13  41930.0
14    250.0
```

```
15    936.0
16   14000.0
17   14400.0
                 max      min
            Sale_amt Sale_amt
Item
Cell Phone     14400.0   6075.0
Desk            250.0    250.0
Home Theater   40500.0  14000.0
Television    113810.0  38336.0
Video Games      936.0    936.0
```

RESULT: Successfully implemented.

8. Write a Pandas program to create a Pivot table and find the item wise unit sold. .(refer sales_data table)
AIM:program to create a Pivot table and find the item wise unit sold
PROCEDURE:Algorithm: Create Pivot Tables in Pandas

Import the Pandas and NumPy libraries.
Define or load your dataset into a Pandas DataFrame.
Use the pivot_table method on the DataFrame to create pivot tables. Specify the following parameters: a. index: The column(s) to use as the index for the pivot table. You can specify one or multiple columns. b. values: The column(s) to aggregate or perform operations on. c. aggfunc: The aggregation function(s) to apply to the specified values.
Display or print the resulting pivot tables to see the aggregated data.

```
import pandas as pd
import numpy as np
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'],

    'OrderDate': ['1-6-18', '1-23-18', '2-9-18', '2-26-18', '3-15-18', '4-1-18', '4-18-18', '5-5-18', '5-22-18', '6-8-18', '6-25-18', '7-12-18', '7-29-18', '8-15-18', '9-1-18', '9-18-18', '10-5-18', '10-22-18'],
    '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', 'Hermann', 'Martha', 'Martha'],
    'SalesMan': ['Alexander', 'Shelli', 'Luis', 'David', 'Stephen', 'Alexander', 'Steven', 'Luis', 'Michael', 'Alexander', 'Sigal', 'Diana', 'Karen', 'Alexander', 'John', 'Alexander', 'Sigal', 'Alexander'],
    'Units': [95, 50, 36, 27, 56, 60, 75, 90, 32, 60, 90, 29, 81, 35, 2, 16, 28, 64],
    'Unit_price': [1198.00, 500.00, 1198.00, 225.00, 1198.00, 500.00, 1198.00, 1198.00, 1198.00, 500.00, 1198.00, 500.00, 500.00, 1198.00, 125.00, 58.50, 500.00, 225.00],
    'Sale_amt': [113810.00, 25000.00, 43128.00, 6075.00, 67088.00, 30000.00, 89850.00,
107820.00, 38336.00, 30000.00, 107820.00, 14500.00, 40500.00, 41930.00, 250.00,
936.00, 14000.00, 14400.00]
}
sales_data = pd.DataFrame(data)
print(sales_data.pivot_table(index=["Region", "Item"], values="Units", aggfunc=np.sum))
pivot_table = sales_data.pivot_table(index='Item', values='Units', aggfunc='sum')
print(pivot_table)
```

```
                  Units
Region  Item
Central Cell Phone       27
        Desk              2
        Home Theater     78
        Television      291
East    Cell Phone       64
        Home Theater    230
        Television      130
        Video Games      16
West    Television       88
            Units
Item
Cell Phone       91
Desk              2
Home Theater    308
Television      509
Video Games      16
```

RESULT: Successfully implemented.

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_data table)

AIM:and find the total sale amount region wise, manager wise, sales man wise.

PROCEDURE:Algorithm: Create a Pivot Table in Pandas

Import the Pandas and NumPy libraries.
Define or load your dataset into a Pandas DataFrame.
Use the pivot_table method on the DataFrame to create a pivot table. Specify the following parameters:
index: The column(s) to use as the index for the pivot table.
values: The column(s) to aggregate or perform operations on.
aggfunc: The aggregation function(s) to apply to the specified values.
Display or print the resulting pivot table to see the aggregated data.

```
import pandas as pd
import numpy as np
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'],

    'OrderDate': ['1-6-18', '1-23-18', '2-9-18', '2-26-18', '3-15-18', '4-1-18', '4-18-18', '5-5-18',
'5-22-18', '6-8-18', '6-25-18', '7-12-18', '7-29-18', '8-15-18', '9-1-18', '9-18-18', '10-5-18',
'10-22-18'],
    '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', 'Hermann',
'Martha', 'Martha'],
    'SalesMan': ['Alexander', 'Shelli', 'Luis', 'David', 'Stephen', 'Alexander', 'Steven', 'Luis',
'Michael', 'Alexander', 'Sigal', 'Diana', 'Karen', 'Alexander', 'John', 'Alexander', 'Sigal',
'Alexander'],
    'Units': [95, 50, 36, 27, 56, 60, 75, 90, 32, 60, 90, 29, 81, 35, 2, 16, 28, 64],
    'Unit_price': [1198.00, 500.00, 1198.00, 225.00, 1198.00, 500.00, 1198.00, 1198.00,
1198.00, 500.00, 1198.00, 500.00, 500.00, 1198.00, 125.00, 58.50, 500.00, 225.00],
    'Sale_amt': [113810.00, 25000.00, 43128.00, 6075.00, 67088.00, 30000.00, 89850.00,
107820.00, 38336.00, 30000.00, 107820.00, 14500.00, 40500.00, 41930.00, 250.00,
936.00, 14000.00, 14400.00]
}
sales_data = pd.DataFrame(data)
print(sales_data.pivot_table(index=["Manager"],values=["Sale_amt"],aggfunc=[np.mean,len])
)
```

```
           mean      len
        Sale_amt Sale_amt
Manager
Douglas  26362.00        3
Hermann  56940.80        5
Martha   43561.25        8
Timothy  36581.50        2
```

RESULT: Successfully implemented.

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 highlight the negative numbers red and positive numbers black.
PROCEDURE:Algorithm: Apply Custom Styling to Pandas DataFrame

Import the Pandas and NumPy libraries.
Optionally, set a random seed with np.random.seed() for reproducibility.
Create a Pandas DataFrame with the desired data.
Define a custom styling function, such as color_negative_red(val), that takes a single value as input and returns a CSS string specifying the text color.
Use the style attribute of the DataFrame to apply the custom styling using the applymap() method. The styling function will be applied to each cell in the DataFrame.
Optionally, assign the styled DataFrame to a variable to capture the styling effect.

Display or print the original and styled DataFrames to see the styling effect.

```
import pandas as pd
import numpy as np
np.random.seed(24)
df = pd.DataFrame({'A': np.linspace(1, 10, 10)})
df = pd.concat([df, pd.DataFrame(np.random.randn(10, 4), columns=list('BCDE'))],
          axis=1)
print("Original array:")
print(df)
def color_negative_red(val):
   color = 'red' if val < 0 else 'black'
   return 'color: %s' % color
print("\nNegative numbers red and positive numbers black:")
df.style.applymap(color_negative_red)
```

```
Original array:
      A         B         C         D         E
0   1.0  1.329212 -0.770033 -0.316280 -0.990810
1   2.0 -1.070816 -1.438713  0.564417  0.295722
2   3.0 -1.626404  0.219565  0.678805  1.889273
3   4.0  0.961538  0.104011 -0.481165  0.850229
4   5.0  1.453425  1.057737  0.165562  0.515018
5   6.0 -1.336936  0.562861  1.392855 -0.063328
6   7.0  0.121668  1.207603 -0.002040  1.627796
7   8.0  0.354493  1.037528 -0.385684  0.519818
8   9.0  1.686583 -1.325963  1.428984 -2.089354
9  10.0 -0.129820  0.631523 -0.586538  0.290720
```

```
Negative numbers red and positive numbers black:
        A       B       C       D       E
0       1.000000        1.329212        -0.770033       -0.316280       -0.990810
1       2.000000        -1.070816       -1.438713       0.564417        0.295722
2       3.000000        -1.626404       0.219565        0.678805        1.889273
3       4.000000        0.961538        0.104011        -0.481165       0.850229
4       5.000000        1.453425        1.057737        0.165562        0.515018
5       6.000000        -1.336936       0.562861        1.392855        -0.063328
6       7.000000        0.121668        1.207603        -0.002040       1.627796
7       8.000000        0.354493        1.037528        -0.385684       0.519818
8       9.000000        1.686583        -1.325963       1.428984        -2.089354
9       10.000000       -0.129820       0.631523        -0.586538       0.290720
```

image.png

RESULT: Successfully implemented.

Open In Colab

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 Write a Pandas program which will highlight the nan values.
PROCEDURE:

Import the necessary libraries, including pandas and numpy.
Set the random seed to ensure reproducibility of random values.
Create a Pandas DataFrame (df) with one column ('A') containing numbers from 1 to 10.
Concatenate the DataFrame with another DataFrame containing random values from a standard normal distribution for columns 'B', 'C', 'D', and 'E', creating a larger DataFrame.
Introduce missing values (NaN) into specific cells by using the iloc method. For example, set the value at row 0 and column 2, row 3 and column 3, row 4 and column 1, and row 9 and column 4 to NaN.
Print the original DataFrame to display its contents.
Define a custom styling function, color_negative_red(val), which takes a value as input and returns a string specifying the font color as 'red' for negative values or 'black' for non-negative values.
Use the style.highlight_null(null_color='red') method to apply custom styling to the DataFrame. This method highlights missing values (NaN) in red.
Print the styled DataFrame, which displays the DataFrame with the specified custom styling. Missing values appear in red, and the font color of negative numbers is set to red as well.

```
import pandas as pd
import numpy as np
np.random.seed(24)
df = pd.DataFrame({'A': np.linspace(1, 10, 10)})
df = pd.concat([df, pd.DataFrame(np.random.randn(10, 4), columns=list('BCDE'))],
               axis=1)
df.iloc[0, 2] = np.nan
df.iloc[3, 3] = np.nan
df.iloc[4, 1] = np.nan
df.iloc[9, 4] = np.nan
print("Original array:")
print(df)
def color_negative_red(val):
   color = 'red' if val < 0 else 'black'
   return 'color: %s' % color
print("\nNegative numbers red and positive numbers black:")
df.style.highlight_null(null_color='red')
```

```
Original array:
      A         B         C         D         E
0   1.0  1.329212       NaN -0.316280 -0.990810
1   2.0 -1.070816 -1.438713  0.564417  0.295722
2   3.0 -1.626404  0.219565  0.678805  1.889273
3   4.0  0.961538  0.104011       NaN  0.850229
4   5.0       NaN  1.057737  0.165562  0.515018
5   6.0 -1.336936  0.562861  1.392855 -0.063328
6   7.0  0.121668  1.207603 -0.002040  1.627796
```

```
7   8.0  0.354493  1.037528 -0.385684  0.519818
8   9.0  1.686583 -1.325963  1.428984 -2.089354
9  10.0 -0.129820  0.631523 -0.586538       NaN
```

Negative numbers red and positive numbers black:

```
<ipython-input-1-29d4b3fe49dc>:17: FutureWarning: `null_color` is deprecated: use `color` instead
  df.style.highlight_null(null_color='red')
        A       B       C       D       E
0       1.000000        1.329212        nan     -0.316280       -0.990810
1       2.000000        -1.070816       -1.438713       0.564417        0.295722
2       3.000000        -1.626404       0.219565        0.678805        1.889273
3       4.000000        0.961538        0.104011        nan     0.850229
4       5.000000        nan     1.057737        0.165562        0.515018
5       6.000000        -1.336936       0.562861        1.392855        -0.063328
6       7.000000        0.121668        1.207603        -0.002040       1.627796
7       8.000000        0.354493        1.037528        -0.385684       0.519818
8       9.000000        1.686583        -1.325963       1.428984        -2.089354
9       10.000000       -0.129820       0.631523        -0.586538       nan
```

Screenshot 2023-10-17 083828.png

RESULT: Successfully implemented.

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 write a Pandas program to set dataframe background Color black and font color yellow.
PROCEDURE:

Import the necessary libraries, including pandas and numpy.
Set the random seed to ensure reproducibility of random values.
Create a Pandas DataFrame (df) with one column ('A') containing numbers from 1 to 10.
Concatenate the DataFrame with another DataFrame containing random values from a standard normal distribution for columns 'B', 'C', 'D', and 'E', creating a larger DataFrame.
Introduce missing values (NaN) into specific cells by using the iloc method. For example, set the value at row 0 and column 2, row 3 and column 3, row 4 and column 1, and row 9 and column 4 to NaN.
Print the original DataFrame to display its contents.
Apply custom styling to the DataFrame using the style.set_properties() method. Set the background color to black and the font color to yellow. This styling is applied to the entire DataFrame, making the background black and the text yellow.
Print the styled DataFrame, which displays the DataFrame with the specified custom styling.

```
import pandas as pd
import numpy as np
np.random.seed(24)
df = pd.DataFrame({'A': np.linspace(1, 10, 10)})
df = pd.concat([df, pd.DataFrame(np.random.randn(10, 4), columns=list('BCDE'))],
            axis=1)
df.iloc[0, 2] = np.nan
df.iloc[3, 3] = np.nan
df.iloc[4, 1] = np.nan
df.iloc[9, 4] = np.nan
print("Original array:")
print(df)
print("\nBackground:black - fontcolor:yelow")
df.style.set_properties(**{'background-color': 'black',
                    'color': 'yellow'})
```

```
Original array:
      A         B         C         D         E
0   1.0  1.329212       NaN -0.316280 -0.990810
1   2.0 -1.070816 -1.438713  0.564417  0.295722
2   3.0 -1.626404  0.219565  0.678805  1.889273
3   4.0  0.961538  0.104011       NaN  0.850229
4   5.0       NaN  1.057737  0.165562  0.515018
5   6.0 -1.336936  0.562861  1.392855 -0.063328
6   7.0  0.121668  1.207603 -0.002040  1.627796
7   8.0  0.354493  1.037528 -0.385684  0.519818
8   9.0  1.686583 -1.325963  1.428984 -2.089354
9  10.0 -0.129820  0.631523 -0.586538       NaN

Background:black - fontcolor:yelow
	A	B	C	D	E
0	1.000000	1.329212	nan	-0.316280	-0.990810
1	2.000000	-1.070816	-1.438713	0.564417	0.295722
2	3.000000	-1.626404	0.219565	0.678805	1.889273
3	4.000000	0.961538	0.104011	nan	0.850229
4	5.000000	nan	1.057737	0.165562	0.515018
5	6.000000	-1.336936	0.562861	1.392855	-0.063328
6	7.000000	0.121668	1.207603	-0.002040	1.627796
7	8.000000	0.354493	1.037528	-0.385684	0.519818
8	9.000000	1.686583	-1.325963	1.428984	-2.089354
9	10.000000	-0.129820	0.631523	-0.586538	nan
image.png
```

RESULT: Successfully implemented.

13.Write a Pandas program to detect missing values of a given DataFrame. Display True or False.
AIM: to detect missing values of a given DataFrame and display True or False.
PROCEDURE:

Import the necessary libraries, including pandas and numpy.

Set Pandas options to display all rows in the DataFrame (pd.set_option('display.max_rows', None)).
Create a Pandas DataFrame (df) with several columns ('ord_no', 'purch_amt', 'ord_date', 'customer_id', 'salesman_id'), which contains both numerical and datetime data.
Print the original DataFrame to display its contents.
Use the isna() method on the DataFrame to create a new Boolean DataFrame that has the same shape as the original one. This Boolean DataFrame indicates where the missing values (NaN) are located.
Print the Boolean DataFrame, which shows True for missing values and False for non-missing values.

```
import pandas as pd
import numpy as np
pd.set_option('display.max_rows', None)
#pd.set_option('display.max_columns', None)
df = pd.DataFrame({
'ord_no':[70001,np.nan,70002,70004,np.nan,70005,np.nan,70010,70003,70012,np.nan,7001
3],
'purch_amt':[150.5,270.65,65.26,110.5,948.5,2400.6,5760,1983.43,2480.4,250.45,
75.29,3045.6],
'ord_date':
['2012-10-05','2012-09-10',np.nan,'2012-08-17','2012-09-10','2012-07-27','2012-09-10','2012-
10-10','2012-10-10','2012-06-27','2012-08-17','2012-04-25'],
'customer_id':[3002,3001,3001,3003,3002,3001,3001,3004,3003,3002,3001,3001],
'salesman_id':[5002,5003,5001,np.nan,5002,5001,5001,np.nan,5003,5002,5003,np.nan]})
print("Original Orders DataFrame:")
print(df)
print("\nMissing values of the said dataframe:")
print(df.isna())
```

```
Original Orders DataFrame:
     ord_no  purch_amt    ord_date  customer_id  salesman_id
0   70001.0     150.50  2012-10-05         3002       5002.0
1       NaN     270.65  2012-09-10         3001       5003.0
2   70002.0      65.26         NaN         3001       5001.0
3   70004.0     110.50  2012-08-17         3003          NaN
4       NaN     948.50  2012-09-10         3002       5002.0
5   70005.0    2400.60  2012-07-27         3001       5001.0
6       NaN    5760.00  2012-09-10         3001       5001.0
7   70010.0    1983.43  2012-10-10         3004          NaN
8   70003.0    2480.40  2012-10-10         3003       5003.0
9   70012.0     250.45  2012-06-27         3002       5002.0
10      NaN      75.29  2012-08-17         3001       5003.0
11  70013.0    3045.60  2012-04-25         3001          NaN

Missing values of the said dataframe:
    ord_no  purch_amt  ord_date  customer_id  salesman_id
```

```
0    False      False     False       False       False
1     True      False     False       False       False
2    False      False      True       False       False
3    False      False     False       False        True
4     True      False     False       False       False
5    False      False     False       False       False
6     True      False     False       False       False
7    False      False     False       False        True
8    False      False     False       False       False
9    False      False     False       False       False
10    True      False     False       False       False
11   False      False     False       False        True
```

RESULT: Successfully implemented.

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 find and replace the missing values in a given DataFrame which do not have any valuable information

PROCEDURE:

Import the necessary libraries, including pandas and numpy.

Set Pandas options to display all rows in the DataFrame (pd.set_option('display.max_rows', None)).

Create a Pandas DataFrame (df) with several columns ('ord_no', 'purch_amt', 'ord_date', 'customer_id', 'salesman_id'), where some missing values are represented as '?' and '--'.

Print the original DataFrame to display its contents.

Use the replace() method to replace the special characters ('?' and '--') with np.nan (representing missing values). This step helps to clean the data and standardize missing values.

Print the resulting DataFrame (result) after replacing the special characters with NaN to display the cleaned data.

```
import pandas as pd
import numpy as np
pd.set_option('display.max_rows', None)
#pd.set_option('display.max_columns', None)
df = pd.DataFrame({
'ord_no':[70001,np.nan,70002,70004,np.nan,70005,"--",70010,70003,70012,np.nan,70013],
'purch_amt':[150.5,270.65,65.26,110.5,948.5,2400.6,5760,"?",12.43,2480.4,250.45, 3045.6],
'ord_date':
['?','2012-09-10',np.nan,'2012-08-17','2012-09-10','2012-07-27','2012-09-10','2012-10-10','2012-10-10','2012-06-27','2012-08-17','2012-04-25'],
'customer_id':[3002,3001,3001,3003,3002,3001,3001,3004,"--",3002,3001,3001],
'salesman_id':[5002,5003,"?",5001,np.nan,5002,5001,"?",5003,5002,5003,"--"]})
print("Original Orders DataFrame:")
print(df)
print("\nReplace the missing values with NaN:")
result = df.replace({"?": np.nan, "--": np.nan})
```

```
print(result)
```

```
Original Orders DataFrame:
   ord_no purch_amt    ord_date customer_id salesman_id
0   70001     150.5           ?        3002        5002
1     NaN    270.65  2012-09-10        3001        5003
2   70002     65.26         NaN        3001           ?
3   70004     110.5  2012-08-17        3003        5001
4     NaN     948.5  2012-09-10        3002         NaN
5   70005    2400.6  2012-07-27        3001        5002
6      --      5760  2012-09-10        3001        5001
7   70010         ?  2012-10-10        3004           ?
8   70003     12.43  2012-10-10          --        5003
9   70012    2480.4  2012-06-27        3002        5002
10    NaN    250.45  2012-08-17        3001        5003
11  70013    3045.6  2012-04-25        3001          --

Replace the missing values with NaN:
     ord_no  purch_amt    ord_date  customer_id  salesman_id
0   70001.0     150.50         NaN       3002.0       5002.0
1       NaN     270.65  2012-09-10       3001.0       5003.0
2   70002.0      65.26         NaN       3001.0          NaN
3   70004.0     110.50  2012-08-17       3003.0       5001.0
4       NaN     948.50  2012-09-10       3002.0          NaN
5   70005.0    2400.60  2012-07-27       3001.0       5002.0
6       NaN    5760.00  2012-09-10       3001.0       5001.0
7   70010.0        NaN  2012-10-10       3004.0          NaN
8   70003.0      12.43  2012-10-10          NaN       5003.0
9   70012.0    2480.40  2012-06-27       3002.0       5002.0
10      NaN     250.45  2012-08-17       3001.0       5003.0
11  70013.0    3045.60  2012-04-25       3001.0          NaN
```

RESULT: Successfully implemented.

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

PROCEDURE:

Import the necessary libraries, including pandas and numpy.
Create a Pandas DataFrame (df) with several columns ('ord_no', 'purch_amt', 'ord_date', 'customer_id') and some missing values represented as NaN.
Use the dropna(thresh=2) method to filter the DataFrame. This method keeps rows that have at least 2 non-null (NaN) values.
Print the resulting filtered DataFrame (filtered_df) to display the rows that meet the filtering criteria.

```
import pandas as pd
import numpy as np
df = pd.DataFrame({

'ord_no':[np.nan,np.nan,70002,np.nan,np.nan,70005,np.nan,70010,70003,70012,np.nan,np.
nan],
   'purch_amt':[np.nan,270.65,65.26,np.nan,948.5,2400.6,5760,1983.43,2480.4,250.45,
75.29,np.nan],
   'ord_date':
[np.nan,'2012-09-10',np.nan,np.nan,'2012-09-10','2012-07-27','2012-09-10','2012-10-10','201
2-10-10','2012-06-27','2012-08-17',np.nan],
   'customer_id':[np.nan,3001,3001,np.nan,3002,3001,3001,3004,3003,3002,3001,np.nan]
})
filtered_df = df.dropna(thresh=2)
print(filtered_df)
```

```
    ord_no  purch_amt    ord_date  customer_id
1      NaN     270.65  2012-09-10       3001.0
2  70002.0      65.26         NaN       3001.0
4      NaN     948.50  2012-09-10       3002.0
5  70005.0    2400.60  2012-07-27       3001.0
6      NaN    5760.00  2012-09-10       3001.0
7  70010.0    1983.43  2012-10-10       3004.0
8  70003.0    2480.40  2012-10-10       3003.0
9  70012.0     250.45  2012-06-27       3002.0
10     NaN      75.29  2012-08-17       3001.0
```

RESULT: Successfully implemented.

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

Import the necessary library, pandas.
Set Pandas options to display all rows in the DataFrame (pd.set_option('display.max_rows', None)).
Create a Pandas DataFrame (student_data) containing student information, including columns like 'school_code,' 'class,' 'name,' 'date_of_birth,' 'age,' 'height,' 'weight,' and 'address.' Also, specify an index for the DataFrame.
Print the original DataFrame to display its contents.
Split the DataFrame into groups based on the 'school_code' column using the groupby() method. This creates a grouped object (result) with groups based on unique 'school_code' values.
Iterate through the groups using a for loop. For each group, display the name of the group (the 'school_code') and the associated data for that group.
Print the type of the result object, which should be of type pandas.core.groupby.generic.DataFrameGroupBy.

```
import pandas as pd
pd.set_option('display.max_rows', None)
#pd.set_option('display.max_columns', None)
student_data = pd.DataFrame({
   'school_code': ['s001','s002','s003','s001','s002','s004'],
   'class': ['V', 'V', 'VI', 'VI', 'V', 'VI'],
   'name': ['Alberto Franco','Gino Mcneill','Ryan Parkes', 'Eesha Hinton', 'Gino Mcneill', 'David
Parkes'],
   'date_Of_Birth ':
['15/05/2002','17/05/2002','16/02/1999','25/09/1998','11/05/2002','15/09/1997'],
   'age': [12, 12, 13, 13, 14, 12],
   'height': [173, 192, 186, 167, 151, 159],
   'weight': [35, 32, 33, 30, 31, 32],
   'address': ['street1', 'street2', 'street3', 'street1', 'street2', 'street4']},
   index=['S1', 'S2', 'S3', 'S4', 'S5', 'S6'])

print("Original DataFrame:")
print(student_data)
print('\nSplit the said data on school_code wise:')
result = student_data.groupby(['school_code'])
for name,group in result:
   print("\nGroup:")
   print(name)
   print(group)
print("\nType of the object:")
print(type(result))
```

```
Original DataFrame:
  school_code class           name date_Of_Birth   age  height  weight  \
S1       s001     V  Alberto Franco     15/05/2002   12     173      35
S2       s002     V    Gino Mcneill    17/05/2002   12     192      32
S3       s003    VI     Ryan Parkes     16/02/1999   13     186      33
S4       s001    VI    Eesha Hinton     25/09/1998   13     167      30
S5       s002     V    Gino Mcneill    11/05/2002   14     151      31
S6       s004    VI    David Parkes     15/09/1997   12     159      32

   address
S1  street1
S2  street2
S3  street3
S4  street1
S5  street2
S6  street4

Split the said data on school_code wise:

Group:
s001
```

```
  school_code class           name date_Of_Birth   age  height  weight  \
S1       s001     V  Alberto Franco     15/05/2002   12     173      35
S4       s001    VI    Eesha Hinton     25/09/1998   13     167      30

   address
S1  street1
S4  street1

Group:
s002
  school_code class          name date_Of_Birth   age  height  weight  \
S2       s002     V  Gino Mcneill     17/05/2002   12     192      32
S5       s002     V  Gino Mcneill     11/05/2002   14     151      31

   address
S2  street2
S5  street2

Group:
s003
  school_code class         name date_Of_Birth   age  height  weight  address
S3       s003    VI  Ryan Parkes     16/02/1999   13     186      33  street3

Group:
s004
  school_code class          name date_Of_Birth   age  height  weight  \
S6       s004    VI  David Parkes     15/09/1997   12     159      32

   address
S6  street4

Type of the object:
<class 'pandas.core.groupby.generic.DataFrameGroupBy'>
<ipython-input-9-45205576c1c9>:19: FutureWarning: In a future version of pandas, a length 1 tuple will be returned when iterating over a groupby with a grouper equal to a list of length 1. Don't supply a list with a single grouper to avoid this warning.
  for name,group in result:
```

RESULT: Successfully implemented.

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 split the following dataframe by school code and get mean, min, and max value of age for each school.

PROCEDURE:

Import the necessary library, pandas.

Set Pandas options to display all rows in the DataFrame (pd.set_option('display.max_rows', None)).

Create a Pandas DataFrame (student_data) containing student information, including columns like 'school_code,' 'class,' 'name,' 'date_of_birth,' 'age,' 'height,' 'weight,' and 'address.' Also, specify an index for the DataFrame.
Print the original DataFrame to display its contents.
Group the DataFrame by the 'school_code' column using the groupby() method. This creates a grouped object.
Use the agg() method to calculate the mean, minimum, and maximum age for each school ('school_code'). The result is a new DataFrame (grouped_single) that shows these statistics.
Print the DataFrame grouped_single to display the calculated statistics for each school.

```
import pandas as pd
pd.set_option('display.max_rows', None)
#pd.set_option('display.max_columns', None)
student_data = pd.DataFrame({
   'school_code': ['s001','s002','s003','s001','s002','s004'],
   'class': ['V', 'V', 'VI', 'VI', 'V', 'VI'],
   'name': ['Alberto Franco','Gino Mcneill','Ryan Parkes', 'Eesha Hinton', 'Gino Mcneill', 'David
Parkes'],
   'date_Of_Birth ':
['15/05/2002','17/05/2002','16/02/1999','25/09/1998','11/05/2002','15/09/1997'],
   'age': [12, 12, 13, 13, 14, 12],
   'height': [173, 192, 186, 167, 151, 159],
   'weight': [35, 32, 33, 30, 31, 32],
   'address': ['street1', 'street2', 'street3', 'street1', 'street2', 'street4']},
   index=['S1', 'S2', 'S3', 'S4', 'S5', 'S6'])

print("Original DataFrame:")
print(student_data)
print('\nMean, min, and max value of age for each value of the school:')
grouped_single = student_data.groupby('school_code').agg({'age': ['mean', 'min', 'max']})
print(grouped_single)
```

```
Original DataFrame:
  school_code class           name date_Of_Birth   age  height  weight  \
S1       s001     V  Alberto Franco     15/05/2002   12     173      35
S2       s002     V    Gino Mcneill     17/05/2002   12     192      32
S3       s003    VI     Ryan Parkes     16/02/1999   13     186      33
S4       s001    VI    Eesha Hinton     25/09/1998   13     167      30
S5       s002     V    Gino Mcneill     11/05/2002   14     151      31
S6       s004    VI    David Parkes     15/09/1997   12     159      32

   address
S1  street1
S2  street2
S3  street3
S4  street1
S5  street2
S6  street4
```

```
Mean, min, and max value of age for each value of the school:
             age
            mean min max
school_code
s001        12.5  12  13
s002        13.0  12  14
s003        13.0  13  13
s004        12.0  12  12
```

RESULT: Successfully implemented.

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

PROCEDURE:

Import the necessary library, pandas.
Set Pandas options to display all rows in the DataFrame (pd.set_option('display.max_rows', None)).
Create a Pandas DataFrame (student_data) containing student information, including columns like 'school_code,' 'class,' 'name,' 'date_of_birth,' 'age,' 'height,' 'weight,' and 'address.' Also, specify an index for the DataFrame.
Print the original DataFrame to display its contents.
Split the DataFrame into groups based on the 'school_code' column using the groupby() method. This creates a grouped object (result) with groups based on unique 'school_code' values.
Iterate through the groups using a for loop, displaying the name of each group (the 'school_code') and the associated data for that group.
Print the type of the result object, which should be of type pandas.core.groupby.generic.DataFrameGroupBy.

```
import pandas as pd
pd.set_option('display.max_rows', None)
#pd.set_option('display.max_columns', None)
student_data = pd.DataFrame({
    'school_code': ['s001','s002','s003','s001','s002','s004'],
    'class': ['V', 'V', 'VI', 'VI', 'V', 'VI'],
    'name': ['Alberto Franco','Gino Mcneill','Ryan Parkes', 'Eesha Hinton', 'Gino Mcneill', 'David Parkes'],
    'date_Of_Birth ': ['15/05/2002','17/05/2002','16/02/1999','25/09/1998','11/05/2002','15/09/1997'],
    'age': [12, 12, 13, 13, 14, 12],
    'height': [173, 192, 186, 167, 151, 159],
    'weight': [35, 32, 33, 30, 31, 32],
    'address': ['street1', 'street2', 'street3', 'street1', 'street2', 'street4']},
    index=['S1', 'S2', 'S3', 'S4', 'S5', 'S6'])
```

```
print("Original DataFrame:")
print(student_data)
print('\nSplit the said data on school_code wise:')
result = student_data.groupby(['school_code'])
for name,group in result:
    print("\nGroup:")
    print(name)
    print(group)
print("\nType of the object:")
print(type(result))
```

```
Original DataFrame:
   school_code class            name date_Of_Birth   age  height  weight  \
S1        s001     V  Alberto Franco     15/05/2002   12     173      35
S2        s002     V    Gino Mcneill    17/05/2002   12     192      32
S3        s003    VI     Ryan Parkes     16/02/1999   13     186      33
S4        s001    VI    Eesha Hinton     25/09/1998   13     167      30
S5        s002     V    Gino Mcneill    11/05/2002   14     151      31
S6        s004    VI    David Parkes     15/09/1997   12     159      32

    address
S1  street1
S2  street2
S3  street3
S4  street1
S5  street2
S6  street4

Split the said data on school_code wise:

Group:
s001
   school_code class            name date_Of_Birth   age  height  weight  \
S1        s001     V  Alberto Franco     15/05/2002   12     173      35
S4        s001    VI    Eesha Hinton     25/09/1998   13     167      30

    address
S1  street1
S4  street1

Group:
s002
   school_code class          name date_Of_Birth   age  height  weight  \
S2        s002     V  Gino Mcneill     17/05/2002   12     192      32
S5        s002     V  Gino Mcneill     11/05/2002   14     151      31

    address
S2  street2
```

```
S5  street2

Group:
s003
  school_code class        name date_Of_Birth   age  height  weight  address
S3       s003    VI  Ryan Parkes     16/02/1999   13     186      33  street3

Group:
s004
  school_code class         name date_Of_Birth   age  height  weight  \
S6       s004    VI  David Parkes     15/09/1997   12     159      32

   address
S6  street4

Type of the object:
<class 'pandas.core.groupby.generic.DataFrameGroupBy'>
<ipython-input-11-45205576c1c9>:19: FutureWarning: In a future version of pandas, a length 1 tuple will be returned when iterating over a groupby with a grouper equal to a list of length 1. Don't supply a list with a single grouper to avoid this warning.
  for name,group in result:
```

RESULT: Successfully implemented.

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 display the dimensions or shape of the World alcohol consumption dataset.

PROCEDURE:

Import the necessary library, pandas.
Create a dictionary (data) containing data for the DataFrame. The dictionary has keys representing column names and lists representing the data for each column.
Use the pd.DataFrame constructor to create a DataFrame (df) from the provided data.
Display the first few rows of the DataFrame using df.head() to get an initial view of the data.
Print the shape of the DataFrame using df.shape to display the number of rows and columns.
Extract the number of rows and columns from the shape of the DataFrame (df.shape[0] for rows and df.shape[1] for columns).
Print the column names using df.columns to show the names of all the columns in the DataFrame.

```
import pandas as pd

data = {
   "Year": [1986, 1986, 1985, 1986, 1987],
   "WHO region": ["Western Pacific", "Americas", "Africa", "Americas", "Americas"],
   "Country": ["Viet Nam", "Uruguay", "Cte d'Ivoire", "Colombia", "Saint Kitts and Nevis"],
   "Beverage Types": ["Wine", "Other", "Wine", "Beer", "Beer"],
    "Display Value": [0.00, 0.50, 1.62, 4.27, 1.98]
}

df = pd.DataFrame(data)
print(df.head())
print('\nShape of the dataframe: ',df.shape)
print('\nNumber of rows: ',df.shape[0])
print('\nNumber of column: ',df.shape[1])
print("\nExtract Column Names:")
print(df.columns)
```

```
   Year      WHO region               Country Beverage Types  Display Value
0  1986  Western Pacific              Viet Nam           Wine           0.00
1  1986        Americas               Uruguay          Other           0.50
2  1985          Africa         Cte d'Ivoire           Wine           1.62
3  1986        Americas              Colombia           Beer           4.27
4  1987        Americas  Saint Kitts and Nevis           Beer           1.98

Shape of the dataframe:  (5, 5)

Number of rows:  5

Number of column:  5

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

RESULT: Successfully implemented.

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

PROCEDURE:

Import the required libraries, including pandas.
Create a Pandas DataFrame (df) with the given data, containing three columns: 'name_code', 'date_of_birth', and 'age'.
Print the original DataFrame to display its contents.
Create a new column called 'Index' in the DataFrame to store the indices of the character 'c' in the 'name_code' column.
Use the str.find() method to find the index of the first occurrence of the character 'c' in each 'name_code' cell. The search is limited to the first 5 characters (0 to 5) of each cell.
Print the modified DataFrame, which now includes the 'Index' column containing the found indices.

```
import pandas as pd
df = pd.DataFrame({
    'name_code': ['c0001','1000c','b00c2', 'b2c02', 'c2222'],
    'date_of_birth ': ['12/05/2002','16/02/1999','25/09/1998','12/02/2022','15/09/1997'],
    'age': [18.5, 21.2, 22.5, 22, 23]
})
print("Original DataFrame:")
print(df)
print("\nIndex of a substring in a specified column of a dataframe:")
df['Index'] = df['name_code'].str.find('c', 0, 5)
print(df)
```

```
Original DataFrame:
  name_code date_of_birth    age
0     c0001     12/05/2002  18.5
1     1000c     16/02/1999  21.2
2     b00c2     25/09/1998  22.5
3     b2c02     12/02/2022  22.0
4     c2222     15/09/1997  23.0

Index of a substring in a specified column of a dataframe:
  name_code date_of_birth    age  Index
0     c0001     12/05/2002  18.5      0
1     1000c     16/02/1999  21.2      4
2     b00c2     25/09/1998  22.5      3
3     b2c02     12/02/2022  22.0      2
4     c2222     15/09/1997  23.0      0
```

RESULT: Successfully implemented.

Open In Colab

21.Write a Pandas program to swap the cases of a specified character column in a given DataFrame.

AIM:to swap the cases of a specified character column in a given DataFrame.

PROCEDURE:

Import the necessary library, pandas.

Create a Pandas DataFrame (df) with three columns: 'company_code,' 'date_of_sale,' and 'sale_amount.' The DataFrame contains data related to company codes, sale dates, and sale amounts.

Print the original DataFrame to display its contents.

Create a new column called 'swapped_company_code' in the DataFrame (df) to store the swapped cases of the 'company_code' column.

Use the map() function along with a lambda function to swap the letter case (convert lowercase letters to uppercase and vice versa) for each value in the 'company_code' column. This operation is applied to each cell in the column.

Print the modified DataFrame (df) with the 'swapped_company_code' column added. This column contains the company codes with swapped cases.

```
import pandas as pd
df = pd.DataFrame({
    'company_code': ['Abcd','EFGF', 'zefsalf', 'sdfslew', 'zekfsdf'],
    'date_of_sale': ['12/05/2002','16/02/1999','25/09/1998','12/02/2022','15/09/1997'],
    'sale_amount': [12348.5, 233331.2, 22.5, 2566552.0, 23.0]
})
print("Original DataFrame:")
print(df)
print("\nSwapp cases in comapny_code:")
df['swapped_company_code'] = list(map(lambda x: x.swapcase(), df['company_code']))
print(df)
```

```
Original DataFrame:
  company_code date_of_sale  sale_amount
0         Abcd   12/05/2002      12348.5
1         EFGF   16/02/1999     233331.2
2      zefsalf   25/09/1998         22.5
3      sdfslew   12/02/2022    2566552.0
4      zekfsdf   15/09/1997         23.0

Swapp cases in comapny_code:
  company_code date_of_sale  sale_amount swapped_company_code
0         Abcd   12/05/2002      12348.5                 aBCD
1         EFGF   16/02/1999     233331.2                 efgf
2      zefsalf   25/09/1998         22.5              ZEFSALF
3      sdfslew   12/02/2022    2566552.0              SDFSLEW
4      zekfsdf   15/09/1997         23.0              ZEKFSDF
```

RESULT: Successfully implemented.

22.Write a Python program to draw a line with suitable label in the x axis, y axis and a title.
AIM: to draw a line plot
PROCEDURE:

Import the necessary library, matplotlib.pyplot, for creating plots and charts.
Create two lists, X and Y. X contains values from 1 to 49 (using the range function), and Y is generated by multiplying each value in X by 3 (using a list comprehension).
Print the values of X and Y to display the lists.
Use plt.plot(X, Y) to create a line plot. This function connects the points defined by the X and Y coordinates with lines.
Set the x-axis label using plt.xlabel('x - axis') to provide a description for the x-axis.
Set the y-axis label using plt.ylabel('y - axis') to provide a description for the y-axis.
Set a title for the plot using plt.title('Draw a line.').
Display the figure by calling plt.show(), which shows the generated line plot with the provided labels and title.

```
import matplotlib.pyplot as plt
X = range(1, 50)
Y = [value * 3 for value in X]
print("Values of X:")
print(*range(1,50))
print("Values of Y (thrice of X):")
print(Y)
plt.plot(X, Y)
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.title('Draw a line.')
plt.show()
```

```
Values of X:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33
34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
Values of Y (thrice of X):
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 63, 66, 69, 72, 75,
78, 81, 84, 87, 90, 93, 96, 99, 102, 105, 108, 111, 114, 117, 120, 123, 126, 129, 132, 135,
138, 141, 144, 147]
```

RESULT: Successfully implemented.

23.Write a Python program to draw a line using given axis values taken from a text file, with suitable label in the x axis, y axis and a title.
AIM: to draw a line using given axis values taken from a text file PROCEDURE:

Import the necessary library, matplotlib.pyplot, for creating plots and charts.
Open a text file named "test.txt" in read mode using a with statement and assign it to the variable f.
Read the contents of the text file using f.read() and store it in the variable data. This step reads the entire file as a single string.
Split the data string into individual lines by using data.split('\n'), which creates a list of lines.
Create two empty lists, x and y, to store the x and y coordinates of the data points.
Iterate through the lines in the data list using a for loop:
Split each line into two parts by space (e.g., "x y").
Append the first part (x-coordinate) to the x list, and the second part (y-coordinate) to the y list.
Use plt.plot(x, y) to create a line plot, where x and y are the lists of data points.
Set the x-axis label using plt.xlabel('x - axis') to provide a description for the x-axis.
Set the y-axis label using plt.ylabel('y - axis') to provide a description for the y-axis.
Set a title for the plot using plt.title('Sample graph!').
Display the figure by calling plt.show(), which shows the generated line plot with the provided labels and title.

```
import matplotlib.pyplot as plt
# x axis values
x = [1,2,3]
# y axis values
y = [2,4,1]
# Plot lines and/or markers to the Axes.
```

```
plt.plot(x, y)
# Set the x axis label of the current axis.
plt.xlabel('x - axis')
# Set the y axis label of the current axis.
plt.ylabel('y - axis')
# Set a title
plt.title('Line Plot')
# Display a figure.
plt.show()
```

image.png

RESULT: Successfully implemented.

24.Write a Python program to draw line charts of the financial data of Alphabet Inc. between October 3, 2016 to October 7, 2016.
AIM:to draw line charts of the financial data of Alphabet Inc.
PROCEDURE:

Import the necessary libraries: pandas for data manipulation and matplotlib.pyplot for creating plots.
Create a dictionary data containing the financial data for the specified date range. The dictionary includes the Date, Open, High, Low, and Close prices for each day.
Use the pd.DataFrame(data) function to create a Pandas DataFrame from the dictionary. This DataFrame, named df, will hold the financial data.
Print the DataFrame df to the console, showing the structure of the data.
Use df.plot() to create line charts for the data in the DataFrame. By default, this function uses the Date column as the x-axis and plots all numeric columns as lines.
Use plt.show() to display the generated line chart.

```
import pandas as pd
import matplotlib.pyplot as plt

data = {
   'Date': ['10-03-16', '10-04-16', '10-05-16', '10-06-16', '10-07-16'],
   'Open': [774.25, 776.030029, 779.309998, 779.0, 779.659973],
   'High': [776.065002, 778.710022, 782.070007, 780.47998, 779.659973],
   'Low': [769.5, 772.890015, 775.650024, 775.539978, 770.75],
   'Close': [772.559998, 776.429993, 776.469971, 776.859985, 775.080017]
}
df = pd.DataFrame(data)
print(df)
df.plot()
plt.show()
```

```
       Date       Open        High        Low       Close
0  10-03-16  774.250000  776.065002  769.500000  772.559998
1  10-04-16  776.030029  778.710022  772.890015  776.429993
2  10-05-16  779.309998  782.070007  775.650024  776.469971
3  10-06-16  779.000000  780.479980  775.539978  776.859985
4  10-07-16  779.659973  779.659973  770.750000  775.080017
```

RESULT: Successfully implemented.

25.Write a Python program to plot two or more lines with legends, different widths and colors.
AIM:to plot two or more lines with legends, different widths and colors

PROCEDURE:

Import the matplotlib.pyplot library for creating plots.
Define two sets of data points x1, y1, x2, and y2, which represent the x and y coordinates of two lines.
Set labels for the x-axis and y-axis using plt.xlabel('x - axis') and plt.ylabel('y - axis').
Set a title for the plot using plt.title('Two or more lines with different widths and colors with suitable legends').
Create the first line plot using plt.plot(x1, y1, color='blue', linewidth=3, label='line1-width-3'). This line will be blue, have a width of 3, and be labeled as 'line1-width-3'.
Create the second line plot using plt.plot(x2, y2, color='red', linewidth=5, label='line2-width-5'). This line will be red, have a width of 5, and be labeled as 'line2-width-5'.
Add legends to distinguish between the two lines using plt.legend().
Display the plot using plt.show().

```
import matplotlib.pyplot as plt
x1 = [10,20,30]
y1 = [20,40,10]
x2 = [10,20,30]
y2 = [40,10,30]
plt.xlabel('x - axis')
plt.ylabel('y - axis')
plt.title('Two or more lines with different widths and colors with suitable legends ')
plt.plot(x1,y1, color='blue', linewidth = 3,  label = 'line1-width-3')
plt.plot(x2,y2, color='red', linewidth = 5,  label = 'line2-width-5')
plt.legend()
plt.show()
```

RESULT: Successfully implemented.

26.Write a Python program to create multiple plots.
AIM:to create multiple plots.

PROCEDURE:

Import the matplotlib.pyplot library to create plots.
Create a figure using plt.figure(). This figure will contain multiple subplots.
Use fig.subplots_adjust() to adjust the layout of the subplots within the figure. The arguments bottom, left, top, and right control the spacing around the subplots.
Create the first subplot using plt.subplot(2, 1, 1). The arguments 2, 1, and 1 indicate that this subplot is part of a 2x1 grid of subplots, and it's the first subplot. The plt.xticks(()) and plt.yticks(()) functions remove the tick marks and labels from the x and y axes.
Create the second subplot using plt.subplot(2, 3, 4). This subplot is part of a 2x3 grid of subplots and is positioned in the fourth location. The plt.xticks(()) and plt.yticks(()) functions remove the tick marks and labels from the x and y axes.
Create the third subplot using plt.subplot(2, 3, 5) in a similar manner as the second subplot. This is positioned in the fifth location of the 2x3 grid.
Create the fourth subplot using plt.subplot(2, 3, 6). This is positioned in the sixth location of the 2x3 grid.
Use plt.xticks(()) and plt.yticks(()) for each subplot to remove tick marks and labels from the x and y axes.
Finally, display the figure with all the subplots using plt.show().

```
import matplotlib.pyplot as plt
fig = plt.figure()
fig.subplots_adjust(bottom=0.020, left=0.020, top = 0.900, right=0.800)
plt.subplot(2, 1, 1)
plt.xticks(()), plt.yticks(())
plt.subplot(2, 3, 4)
plt.xticks(())
plt.yticks(())
plt.subplot(2, 3, 5)
plt.xticks(())
plt.yticks(())
plt.subplot(2, 3, 6)
plt.xticks(())
plt.yticks(())
plt.show()
```

RESULT: Successfully implemented.

27.Write a Python programming to display a bar chart of the popularity of programming Languages.
AIM: to display a bar chart of the popularity of programming Languages.

PROCEDURE:

Import the matplotlib.pyplot library for creating plots.
Define the programming languages (x) and their corresponding popularity percentages (popularity) in two separate lists.
Create a list x_pos using a list comprehension to generate an index for each programming language. This index will be used for positioning the bars on the x-axis.
Use plt.bar(x_pos, popularity, color=(0.4, 0.6, 0.8, 1.0), edgecolor='blue') to create a bar chart. The x_pos list is used for the x-axis, popularity for the y-axis, and color to set the color of the bars. The edgecolor parameter sets the color of the bar edges.
Set the x-axis label using plt.xlabel("Languages") and the y-axis label using plt.ylabel("Popularity").
Set the title of the plot using plt.title("Popularity of Programming Language\nWorldwide, Oct 2017 compared to a year ago").
Customize the x-axis tick labels to display the programming languages by using plt.xticks(x_pos, x).
Turn on the grid lines using plt.minorticks_on() and customize the major grid lines' style, width, and color using plt.grid(which='major', linestyle='-', linewidth='0.5', color='red').
Customize the minor grid lines' style, width, and color using plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black').
Finally, display the bar chart using plt.show().

```
import matplotlib.pyplot as plt
x = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
x_pos = [i for i, _ in enumerate(x)]

plt.bar(x_pos, popularity, color=(0.4, 0.6, 0.8, 1.0), edgecolor='blue')

plt.xlabel("Languages")
plt.ylabel("Popularity")
plt.title("PopularitY of Programming Language\n" + "Worldwide, Oct 2017 compared to a year ago")
plt.xticks(x_pos, x)
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='red')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```

RESULT: Successfully implemented.

28.Write a Python programming to display a horizontal bar chart of the popularity of programming Languages.
AIM: to display a horizontal bar chart of the popularity of programming Languages.
PROCEDURE:

Import the matplotlib.pyplot library for creating plots.

Define the programming languages (x) and their corresponding popularity percentages (popularity) in two separate lists.
Create a list x_pos using a list comprehension to generate an index for each programming language. This index will be used for positioning the bars on the y-axis.
Use plt.barh(x_pos, popularity, color='green') to create a horizontal bar chart. The x_pos list is used for the y-axis, popularity for the x-axis, and color to set the color of the bars.
Set the x-axis label using plt.xlabel("Popularity") and the y-axis label using plt.ylabel("Languages").
Set the title of the plot using plt.title("Popularity of Programming Language\nWorldwide, Oct 2017 compared to a year ago").
Customize the y-axis tick labels to display the programming languages by using plt.yticks(x_pos, x).
Turn on the grid lines using plt.minorticks_on() and customize the major grid lines' style, width, and color using plt.grid(which='major', linestyle='-', linewidth='0.5', color='red').
Customize the minor grid lines' style, width, and color using plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black').
Finally, display the horizontal bar chart using plt.show().

```
import matplotlib.pyplot as plt
x = ['Java', 'Python', 'PHP', 'JS', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
x_pos = [i for i, _ in enumerate(x)]
plt.barh(x_pos, popularity, color='green')
plt.xlabel("Popularity")
plt.ylabel("Languages")
plt.title("PopularitY of Programming Language\n" + "Worldwide, Oct 2017 compared to a year ago")
plt.yticks(x_pos, x)
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='red')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```

RESULT: Successfully implemented.

29.Write a Python programming to display a bar chart of the popularity of programming Languages. Use different color for each bar.
AIM: to display a bar chart of the popularity of programming Languages. Use different color for each bar.

PROCEDURE:

Import the matplotlib.pyplot library for creating plots.
Define the programming languages (x) and their corresponding popularity percentages (popularity) in two separate lists.

Create a list x_pos using a list comprehension to generate an index for each programming language. This index will be used for positioning the bars on the x-axis.
Use plt.bar(x_pos, popularity, color=['red', 'black', 'green', 'blue', 'yellow', 'cyan']) to create a bar chart. The x_pos list is used for the x-axis, popularity for the y-axis, and the color parameter is set to a list of custom colors for each bar.
Set the x-axis label using plt.xlabel("Languages") and the y-axis label using plt.ylabel("Popularity").
Set the title of the plot using plt.title("Popularity of Programming Language\nWorldwide, Oct 2017 compared to a year ago").
Customize the x-axis tick labels to display the programming languages by using plt.xticks(x_pos, x).
Turn on the grid lines using plt.minorticks_on() and customize the major grid lines' style, width, and color using plt.grid(which='major', linestyle='-', linewidth='0.5', color='red').
Customize the minor grid lines' style, width, and color using plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black').
Finally, display the bar chart with custom colors for each bar using plt.show().

```
import matplotlib.pyplot as plt
x = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
x_pos = [i for i, _ in enumerate(x)]
plt.bar(x_pos, popularity, color=['red', 'black', 'green', 'blue', 'yellow', 'cyan'])
plt.xlabel("Languages")
plt.ylabel("Popularity")
plt.title("PopularitY of Programming Language\n" + "Worldwide, Oct 2017 compared to a
year ago")
plt.xticks(x_pos, x)
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='red')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```

RESULT: Successfully implemented.

30.Write a Python program to create bar plot of scores by group and gender. Use multiple X values on the same chart for men and women.
AIM:to create bar plot of scores by group and gender. Use multiple X values on the same chart for men and women.

PROCEDURE:

Import the necessary libraries, numpy as np, and matplotlib.pyplot as plt.
Define the number of groups (n_groups) and the scores for Men (men_means) and Women (women_means). In this example, there are 5 groups.
Create the figure and axes for the plot using fig, ax = plt.subplots().
Define the x-axis positions for each group using index = np.arange(n_groups).

Set the bar width using bar_width = 0.35.
Define the opacity of the bars using opacity = 0.8.
Create the bars for Men using rects1 = plt.bar(index, men_means, bar_width, alpha=opacity, color='g', label='Men'). This line creates the bars for Men and specifies their position, width, opacity, color, and label.
Create the bars for Women using rects2 = plt.bar(index + bar_width, women_means, bar_width, alpha=opacity, color='r', label='Women'). This line creates the bars for Women, positioning them next to the Men's bars.
Set the x-axis label using plt.xlabel('Person'), the y-axis label using plt.ylabel('Scores'), and the plot title using plt.title('Scores by person').
Customize the x-axis tick labels to display group labels using plt.xticks(index + bar_width, ('G1', 'G2', 'G3', 'G4', 'G5')).
Add a legend to distinguish Men and Women using plt.legend().
Adjust the layout for a better display with plt.tight_layout().
Finally, display the grouped bar chart using plt.show().

```
import numpy as np
import matplotlib.pyplot as plt
n_groups = 5
men_means = (22, 30, 33, 30, 26)
women_means = (25, 32, 30, 35, 29)
fig, ax = plt.subplots()
index = np.arange(n_groups)
bar_width = 0.35
opacity = 0.8
rects1 = plt.bar(index, men_means, bar_width,alpha=opacity,color='g',label='Men')
rects2 = plt.bar(index + bar_width, women_means,
bar_width,alpha=opacity,color='r',label='Women')
plt.xlabel('Person')
plt.ylabel('Scores')
plt.title('Scores by person')
plt.xticks(index + bar_width, ('G1', 'G2', 'G3', 'G4', 'G5'))
plt.legend()
plt.tight_layout()
plt.show()
```

RESULT: Successfully implemented.

Open In Colab
31.Write a Python program to create a stacked bar plot with error bars.
AIM:to create a stacked bar plot with error bars.

PROCEDURE:

Import the necessary libraries, numpy as np, and matplotlib.pyplot as plt.

Define the number of groups N, and the means and standard deviations for Men (menMeans and menStd) and Women (womenMeans and womenStd). In this example, there are 5 groups.
Create an array ind using np.arange(N) to specify the x-positions for each group.
Define the width of the bars using width = 0.35.
Create the bars for Men with error bars using p1 = plt.bar(ind, menMeans, width, yerr=menStd, color='red'). This line creates the bars for Men, specifies their positions, width, and error bars.
Create the bars for Women with error bars using p2 = plt.bar(ind, womenMeans, width, bottom=menMeans, yerr=womenStd, color='green'). This line creates the bars for Women, positions them above the Men's bars, and specifies error bars.
Set the y-axis label using plt.ylabel('Scores'), the x-axis label using plt.xlabel('Groups'), and the plot title using plt.title('Scores by group and gender').
Customize the x-axis tick labels to display group labels using plt.xticks(ind, ('Group1', 'Group2', 'Group3', 'Group4', 'Group5')).
Set the y-axis tick intervals to display ticks from 0 to 80 with intervals of 10 using plt.yticks(np.arange(0, 81, 10)).
Add a legend to distinguish Men and Women using plt.legend((p1[0], p2[0]), ('Men', 'Women')).
Finally, display the grouped bar chart with error bars using plt.show().

```
import numpy as np
import matplotlib.pyplot as plt
N = 5
menMeans = (22, 30, 35, 35, 26)
womenMeans = (25, 32, 30, 35, 29)
menStd = (4, 3, 4, 1, 5)
womenStd = (3, 5, 2, 3, 3)
ind = np.arange(N)
width = 0.35
p1 = plt.bar(ind, menMeans, width, yerr=menStd, color='red')
p2 = plt.bar(ind, womenMeans, width,
bottom=menMeans, yerr=womenStd, color='green')
plt.ylabel('Scores')
plt.xlabel('Groups')
plt.title('Scores by group\n' + 'and gender')
plt.xticks(ind, ('Group1', 'Group2', 'Group3', 'Group4', 'Group5'))
plt.yticks(np.arange(0, 81, 10))
plt.legend((p1[0], p2[0]), ('Men', 'Women'))
plt.show()
```

RESULT: Successfully implemented.

32.Write a Python program to draw a scatter graph taking a random distribution in X and Y and plotted against each other.
AIM:to draw a scatter graph taking a random distribution in X and Y

PROCEDURE:

Import the necessary library, matplotlib.pyplot as plt.
Generate random data points for the X and Y coordinates. In this example, the data points are generated using randn(200) for both X and Y, which creates 200 random values for each coordinate.
Create a scatter plot of the generated data points using plt.scatter(X, Y, color='r'). This line creates the scatter plot and specifies the X and Y coordinates. The color parameter is set to 'r', which represents the color red.
Set the x-axis label using plt.xlabel("X"), and the y-axis label using plt.ylabel("Y").
Finally, display the scatter plot using plt.show().

```
import matplotlib.pyplot as plt
from pylab import randn
X = randn(200)
Y = randn(200)
plt.scatter(X,Y, color='r')
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

RESULT: Successfully implemented.

33.Write a Python program to draw a scatter plot with empty circles taking a random distribution in X and Y and plotted against each other.
AIM: to draw a scatter plot with empty circles taking a random distribution
PROCEDURE:

Import the necessary libraries, matplotlib.pyplot as plt and numpy as np.
Generate random data points for the X and Y coordinates. In this example, np.random.randn(50) is used to generate 50 random values for both X and Y.
Create a scatter plot of the generated data points using plt.scatter(x, y, s=70, facecolors='none', edgecolors='g'). This line creates the scatter plot and specifies the X and Y coordinates. The s parameter sets the size of the markers, facecolors is set to 'none' to make the markers unfilled, and edgecolors is set to 'g' for green edge colors.
Set the x-axis label using plt.xlabel("X") and the y-axis label using plt.ylabel("Y").
Finally, display the scatter plot using plt.show().

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randn(50)
y = np.random.randn(50)
plt.scatter(x, y, s=70, facecolors='none', edgecolors='g')
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

RESULT: Successfully implemented.

34.Write a Python program to draw a scatter plot using random distributions to generate balls of different sizes.
AIM: to draw a scatter plot using random distributions to generate balls of different sizes.

PROCEDURE:

Import the necessary libraries: math, random, and matplotlib.pyplot as plt.
Set the number of balls to be created as no_of_balls = 25.
Generate random X and Y coordinates for the positions of the balls:
For the X-coordinate, generate 25 random values using random.triangular().
For the Y-coordinate, generate 25 random values using random.gauss(0.5, 0.25). The random.gauss function generates values with a mean of 0.5 and a standard deviation of 0.25, creating a distribution around the center.
Generate random colors for the balls using random.randint(1, 4). This assigns each ball a random integer color code between 1 and 4.
Generate random areas for the balls using math.pi * random.randint(5, 15)**2. The areas are calculated as the square of a random integer value between 5 and 15, multiplied by pi.
Create a new figure using plt.figure() to prepare for plotting.
Create a scatter plot of the balls using plt.scatter(x, y, s=areas, c=colors, alpha=0.85). This line specifies the X and Y coordinates, marker areas, colors, and marker transparency. The s parameter sets the marker areas, c specifies the marker colors, and alpha controls the marker transparency.
Set the axis limits for the plot using plt.axis([0.0, 1.0, 0.0, 1.0]) to ensure the plot is within the range of [0, 1] for both X and Y axes.
Label the x-axis as "X" using plt.xlabel("X") and the y-axis as "Y" using plt.ylabel("Y").
Display the scatter plot using plt.show().

```
import math
import random
import matplotlib.pyplot as plt
# create random data
no_of_balls = 25
x = [random.triangular() for i in range(no_of_balls)]
y = [random.gauss(0.5, 0.25) for i in range(no_of_balls)]
colors = [random.randint(1, 4) for i in range(no_of_balls)]
areas = [math.pi * random.randint(5, 15)**2 for i in range(no_of_balls)]
# draw the plot
plt.figure()
plt.scatter(x, y, s=areas, c=colors, alpha=0.85)
plt.axis([0.0, 1.0, 0.0, 1.0])
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```

RESULT: Successfully implemented.

35.Write a Python program to draw a scatter plot comparing two subject marks of Mathematics and Science. Use marks of 10 students.
AIM:to draw a scatter plot comparing two subject marks of Mathematics and Science.
PROCEDURE:

Import the necessary libraries: matplotlib.pyplot as plt and pandas as pd.
Create two lists, math_marks and science_marks, to store the marks obtained by students in the Math and Science subjects, respectively. The values in these lists represent the marks of individual students.
Create a marks_range list to represent the different marks ranges. This list contains values from 10 to 100 in increments of 10, indicating the possible mark ranges.
Use plt.scatter(marks_range, math_marks, label='Math marks', color='r') to create a scatter plot for Math marks. The marks_range represents the X-axis (marks range), and math_marks represent the Y-axis (Math marks). The label parameter provides a label for the Math marks plot, and color='r' specifies the color of the data points as red.
Use plt.scatter(marks_range, science_marks, label='Science marks', color='g') to create a scatter plot for Science marks. Similarly, the marks_range represents the X-axis (marks range), and science_marks represent the Y-axis (Science marks). The label parameter provides a label for the Science marks plot, and color='g' specifies the color of the data points as green.
Set the title of the scatter plot using plt.title('Scatter Plot').
Label the X-axis as "Marks Range" using plt.xlabel('Marks Range').
Label the Y-axis as "Marks Scored" using plt.ylabel('Marks Scored').
Add a legend to the plot using plt.legend() to distinguish between the Math and Science marks.
Display the scatter plot using plt.show().

```
import matplotlib.pyplot as plt
import pandas as pd
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]
plt.scatter(marks_range, math_marks, label='Math marks', color='r')
plt.scatter(marks_range, science_marks, label='Science marks', color='g')
plt.title('Scatter Plot')
plt.xlabel('Marks Range')
plt.ylabel('Marks Scored')
plt.legend()
plt.show()
```

RESULT: Successfully implemented.

36.Write a Python program to draw a scatter plot for three different groups comparing weights and heights.
AIM: to draw a scatter plot for three different groups comparing weights and heights.
PROCEDURE:

Import the necessary libraries: matplotlib.pyplot as plt and numpy as np.
Create three lists for each group: weight1, height1, weight2, height2, weight3, and height3. Each list contains the weight and height data of individuals in the respective group.
Concatenate the weight and height data from all three groups into two new arrays, weight and height. This is done using np.concatenate().
Create a scatter plot using plt.scatter(weight, height, marker='*', color=['blue']). The weight array represents the X-axis (weight), and the height array represents the Y-axis (height). The marker='*' specifies that asterisks should be used as markers, and color=['blue'] sets the color of the data points to blue.
Label the X-axis as 'Weight' with plt.xlabel('weight', fontsize=16).
Label the Y-axis as 'Height' with plt.ylabel('height', fontsize=16).
Set the title of the scatter plot to 'Group wise Weight vs Height scatter plot' with plt.title('Group wise Weight vs Height scatter plot', fontsize=20).
Display the scatter plot using plt.show().

```
import matplotlib.pyplot as plt
import numpy as np
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]
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]
weight3=[68.2,67.2,68.4,68.7,71,71.3,70.8,70,71.1,71.7]
height3=[165.8,170.9,192.8,135.4,161.4,136.1,167.1,235.1,181.1,177.3]
weight=np.concatenate((weight1,weight2,weight3))
height=np.concatenate((height1,height2,height3))
plt.scatter(weight, height, marker='*', color=['blue'])
plt.xlabel('weight', fontsize=16)
plt.ylabel('height', fontsize=16)
plt.title('Group wise Weight vs Height scatter plot',fontsize=20)
plt.show()
```

RESULT: Successfully implemented.

37.Write a Pandas program to create a dataframe from a dictionary and display it.
AIM:to create a dataframe from a dictionary and display it.
PROCEDURE:

Import the pandas library as pd.
Create a dictionary-like data structure with column labels ('X', 'Y', and 'Z') as keys and lists of values as their corresponding values. The lists represent the values for each column:

'X': [78, 85, 96, 80, 86]
'Y': [84, 94, 89, 83, 86]
'Z': [86, 97, 96, 72, 83]
Use pd.DataFrame() to create a pandas DataFrame from the dictionary-like data structure.
This DataFrame has three columns: 'X', 'Y', and 'Z', and the provided values are used as the data for each column.
Print the resulting DataFrame, which displays the values in a tabular format with labeled columns.

```
import pandas as pd
df = pd.DataFrame({'X':[78,85,96,80,86], 'Y':[84,94,89,83,86],'Z':[86,97,96,72,83]});
print(df)
```

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

RESULT: Successfully implemented.

38.Write a Pandas program to create and display a DataFrame from a specified dictionary data which has the index labels.
AIM:to create and display a DataFrame from a specified dictionary data which has the index labels.
PROCEDURE:

Import the pandas library as pd.
Import the numpy library as np.
Define a dictionary named exam_data containing four key-value pairs:
'name': A list of student names.
'score': A list of exam scores. It includes some missing values represented by np.nan.
'attempts': A list of the number of attempts.
'qualify': A list indicating whether the students qualify, with values 'yes' or 'no'.
Define a list named labels containing labels for the index of the DataFrame. These labels will be used to uniquely identify each row in the DataFrame.
Create a pandas DataFrame named df using the pd.DataFrame() constructor, passing exam_data as the data and specifying labels as the custom index labels.
Print the resulting DataFrame, which displays the student data in a tabular format with labeled columns and a custom index.

```
import pandas as pd
import numpy as np

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']

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

```
        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: Successfully implemented.

39.Write a Pandas program to get the first 3 rows of a given DataFrame.
AIM: to get the first 3 rows of a given DataFrame.
PROCEDURE:

Import the pandas library as pd.
Import the numpy library as np.
Define a dictionary named exam_data containing four key-value pairs:
'name': A list of student names.
'score': A list of exam scores. It includes some missing values represented by np.nan.
'attempts': A list of the number of attempts.
'qualify': A list indicating whether the students qualify, with values 'yes' or 'no'.
Define a list named labels containing labels for the index of the DataFrame. These labels will be used to uniquely identify each row in the DataFrame.
Create a pandas DataFrame named df using the pd.DataFrame() constructor, passing exam_data as the data and specifying labels as the custom index labels.
Print the first three rows of the DataFrame using the .iloc[:3] indexing, which selects the first three rows.

```
import pandas as pd
import numpy as np

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']
```

```
df = pd.DataFrame(exam_data , index=labels)
print("First three rows of the data frame:")
print(df.iloc[:3])
```

```
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: Successfully implemented.

40. Write a Pandas program to select the 'name' and 'score' columns from the following DataFrame.
AIM:to select the 'name' and 'score' columns from the following DataFrame.
PROCEDURE:

Import the pandas library as pd.
Import the numpy library as np.
Define a dictionary named exam_data containing four key-value pairs:
'name': A list of student names.
'score': A list of exam scores. It includes some missing values represented by np.nan.
'attempts': A list of the number of attempts.
'qualify': A list indicating whether the students qualify, with values 'yes' or 'no'.
Define a list named labels containing labels for the index of the DataFrame. These labels will be used to uniquely identify each row in the DataFrame.
Create a pandas DataFrame named df using the pd.DataFrame() constructor, passing exam_data as the data and specifying labels as the custom index labels.
Select specific columns from the DataFrame and print the result. In this case, the code selects the 'name' and 'score' columns using the following syntax: df[['name', 'score']].

```
import pandas as pd
import numpy as np

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']

df = pd.DataFrame(exam_data , index=labels)
print("Select specific columns:")
print(df[['name', 'score']])
```

```
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: Successfully implemented.
```