TRIBHUVAN UNIVERSITY

INSTITUTE OF SCIENCE AND TECHNOLOGY

Lab Report on

Data Warehousing and Data Mining

Submitted To

____

Yoga Raj Joshi

Department of Computer Science and Information Technology

Nagarjuna College of IT

Submitted By

Bibek Angdembe (23957/076)

Jan 2024

Table of contents

Yog Raj Joshi

1) Lab1 – Perform data cleaning on given data

Source Code

```
import pandas as pd
import numpy as np

data = pd.read_csv('employees.csv')
print("Original Data")
print(data[0:25])

# Removing missing values
data=data.dropna(axis=0)
# Removing duplicate rows
data.drop_duplicates(keep='first',inplace=True)
# Removing column Boonus %
del data['Bonus %']
# Correcting Inconsitencies among values
data['Team']=data['Team'].str.replace('Fin','Finance')
data['Team']=data['Team'].str.replace('Mkt','Marketing')
data['Team']=data['Team'].str.replace('Financeance','Finance')
print("Cleaned Data")
print(data[0:25])
data.to_csv('employees_cleaned.csv', index=False)
print("Successfully Cleaned...")
```

Output

```
PS E:\Bsccsit\7thsem\labs> python 1_Lab_Cleaning_data.py
E:\Bsccsit\7thsem\labs\1_Lab_Cleaning_data.py:1: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better interoperability with other l
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
Uncleaned or Orginal data
   First Name  Gender    Salary  Bonus %                  Team
0     Douglas    Male   97308.0    6.945             Marketing
1     Douglas    Male   97308.0    6.945             Marketing
2     Douglas    Male   97310.0    6.945             Marketing
3      Thomas    Male   61933.0    4.170                   NaN
4       Maria  Female  130590.0   11.858               Finance
5       Jerry    Male  138705.0    9.340                   Fin
6       Larry    Male       NaN    1.389       Client Services
7      Dennis    Male  115163.0   10.125                 Legal
8        Ruby  Female   65476.0   10.012               Product
9         NaN  Female   45906.0   11.598                   Fin
10     Angela  Female   95570.0   18.523           Engineering
11      Maria  Female  130590.0   11.858               Finance
12    Frances  Female  139852.0    7.524  Business Development
13     Louise  Female   63241.0   15.132                   NaN
14      Julie  Female  102508.0   12.637                 Legal
15    Brandon    Male  112807.0   17.492       Human Resources
16       Gary    Male  109831.0    5.831                 Sales
17   Kimberly  Female       NaN   14.543               Finance
18    Lillian  Female   59414.0    1.256               Product
19     Jeremy    Male   90370.0    7.369       Human Resources
20      Shawn    Male  111737.0    6.414               Product
21      Diana  Female  132940.0   19.082       Client Services
22      Donna  Female   81014.0    1.894               Product
23       Lois     NaN   64714.0    4.934                 Legal
24    Matthew    Male  100612.0   13.645                   Mkt
```

```
Data after cleaning
   First Name  Gender    Salary                  Team
0     Douglas    Male   97308.0             Marketing
2     Douglas    Male   97310.0             Marketing
4       Maria  Female  130590.0               Finance
5       Jerry    Male  138705.0               Finance
7      Dennis    Male  115163.0                 Legal
8        Ruby  Female   65476.0               Product
10     Angela  Female   95570.0           Engineering
12    Frances  Female  139852.0  Business Development
14      Julie  Female  102508.0                 Legal
15    Brandon    Male  112807.0       Human Resources
16       Gary    Male  109831.0                 Sales
18    Lillian  Female   59414.0               Product
19     Jeremy    Male   90370.0       Human Resources
20      Shawn    Male  111737.0               Product
21      Diana  Female  132940.0       Client Services
22      Donna  Female   81014.0               Product
24    Matthew    Male  100612.0             Marketing
27       John    Male   97950.0       Client Services
29      Craig    Male   37598.0             Marketing
31      Terry    Male  124008.0       Client Services
32   Benjamin    Male   79529.0                 Legal
33  Christina  Female  118780.0           Engineering
36       Jean  Female  119082.0  Business Development
37      Jerry    Male   95734.0       Client Services
38    Theresa  Female   85182.0                 Sales
Data has been Successfully Cleaned...
```

2) Lab2 – Perform data cleaning on the given data

Source Code

```
import pandas as pd
import numpy as np

data = pd.read_csv('employees.csv')
print("Original Data or Uncleaned data")
print(data[0:20])

# Filling missing values with mean
data['Salary']=data['Salary'].fillna(data['Salary'].mean())
print("Cleaned Data ")
print(data[0:20])

data = pd.read_csv('employees.csv')
print("Original Data")
print(data[0:20])



data['Salary']=data['Salary'].interpolate(method="linear")
print("Cleaned Data")
print(data[0:20])
```

Output

```
 import pandas as pd
Original Data or Uncleaned data
   First Name  Gender    Salary  Bonus %                 Team
0     Douglas    Male   97308.0    6.945            Marketing
1     Douglas    Male   97308.0    6.945            Marketing
2     Douglas    Male   97310.0    6.945            Marketing
3      Thomas    Male   61933.0    4.170                  NaN
4       Maria  Female  130590.0   11.858              Finance
5       Jerry    Male  138705.0    9.340                  Fin
6       Larry    Male       NaN    1.389      Client Services
7      Dennis    Male  115163.0   10.125                Legal
8        Ruby  Female   65476.0   10.012              Product
9         NaN  Female   45906.0   11.598                  Fin
10     Angela  Female   95570.0   18.523          Engineering
11      Maria  Female  130590.0   11.858              Finance
12    Frances  Female  139852.0    7.524 Business Development
13     Louise  Female   63241.0   15.132                  NaN
14      Julie  Female  102508.0   12.637                Legal
15    Brandon    Male  112807.0   17.492      Human Resources
16       Gary    Male  109831.0    5.831                Sales
17   Kimberly  Female       NaN   14.543              Finance
18    Lillian  Female   59414.0    1.256              Product
19     Jeremy    Male   90370.0    7.369      Human Resources
```

```
Cleaned Data
   First Name  Gender         Salary  Bonus %                 Team
0     Douglas    Male   97308.000000    6.945            Marketing
1     Douglas    Male   97308.000000    6.945            Marketing
2     Douglas    Male   97310.000000    6.945            Marketing
3      Thomas    Male   61933.000000    4.170                  NaN
4       Maria  Female  130590.000000   11.858              Finance
5       Jerry    Male  138705.000000    9.340                  Fin
6       Larry    Male   90754.204795    1.389      Client Services
7      Dennis    Male  115163.000000   10.125                Legal
8        Ruby  Female   65476.000000   10.012              Product
9         NaN  Female   45906.000000   11.598                  Fin
10     Angela  Female   95570.000000   18.523          Engineering
11      Maria  Female  130590.000000   11.858              Finance
12    Frances  Female  139852.000000    7.524 Business Development
13     Louise  Female   63241.000000   15.132                  NaN
14      Julie  Female  102508.000000   12.637                Legal
15    Brandon    Male  112807.000000   17.492      Human Resources
16       Gary    Male  109831.000000    5.831                Sales
17   Kimberly  Female   90754.204795   14.543              Finance
18    Lillian  Female   59414.000000    1.256              Product
19     Jeremy    Male   90370.000000    7.369      Human Resources
```

```
Cleaned Data
   First Name  Gender    Salary  Bonus %                 Team
0     Douglas    Male   97308.0    6.945            Marketing
1     Douglas    Male   97308.0    6.945            Marketing
2     Douglas    Male   97310.0    6.945            Marketing
3      Thomas    Male   61933.0    4.170                  NaN
4       Maria  Female  130590.0   11.858              Finance
5       Jerry    Male  138705.0    9.340                  Fin
6       Larry    Male  126934.0    1.389      Client Services
7      Dennis    Male  115163.0   10.125                Legal
8        Ruby  Female   65476.0   10.012              Product
9         NaN  Female   45906.0   11.598                  Fin
10     Angela  Female   95570.0   18.523          Engineering
11      Maria  Female  130590.0   11.858              Finance
12    Frances  Female  139852.0    7.524 Business Development
13     Louise  Female   63241.0   15.132                  NaN
14      Julie  Female  102508.0   12.637                Legal
15    Brandon    Male  112807.0   17.492      Human Resources
16       Gary    Male  109831.0    5.831                Sales
17   Kimberly  Female   84622.5   14.543              Finance
18    Lillian  Female   59414.0    1.256              Product
19     Jeremy    Male   90370.0    7.369      Human Resources
```

3) Simulating Network Stability and Coalition Formation Using Graph Theory

Source Code

```
import networkx as nx
import matplotlib.pyplot as plt
import random
import itertools
def get_signs_of_graph(g, tris_list):
    # eg-['A-B','B-C','C-A']
    all_signs = []
    for i in range(len(tris_list)):
        t = []
        t.append(g[tris_list[i][0]][tris_list[i][1]]['sign'])
        t.append(g[tris_list[i][1]][tris_list[i][2]]['sign'])
        t.append(g[tris_list[i][2]][tris_list[i][0]]['sign'])
        all_signs.append(t)
    return all_signs
def unstablecount(all_signs):
    stable = 0
    unstable = 0
    for i in range(len(all_signs)):
        if (((all_signs[i]).count('+')) == 1 or ((all_signs[i]).count('+')) == 3):
            stable += 1
    unstable = len(all_signs) - stable
    return unstable
def move_graph_to_stable(g, tris_list, all_signs):
    found_unstable = False
    ran = 0
```

```
while (found_unstable == False):
    ran = random.randint(0, len(tris_list) - 1)
    if (all_signs[ran].count('+') % 2 == 0):
        found_unstable = True
    else:
        continue
r = random.randint(1, 3)
if (all_signs[ran].count('+') == 2):
    if (r == 1):
        if (g[tris_list[ran][0]][tris_list[ran][1]]['sign'] == '+'):
            g[tris_list[ran][0]][tris_list[ran][1]]['sign'] = '-'
        else:
            g[tris_list[ran][0]][tris_list[ran][1]]['sign'] = '+'
    elif (r == 2):
        if (g[tris_list[ran][1]][tris_list[ran][2]]['sign'] == '+'):
            g[tris_list[ran][1]][tris_list[ran][2]]['sign'] = '-'
        else:
            g[tris_list[ran][1]][tris_list[ran][2]]['sign'] = '+'
    else:
        if (g[tris_list[ran][0]][tris_list[ran][2]]['sign'] == '+'):
            g[tris_list[ran][0]][tris_list[ran][2]]['sign'] = '-'
        else:
            g[tris_list[ran][0]][tris_list[ran][2]]['sign'] = '+'


else:


    if (r == 1):
        g[tris_list[ran][0]][tris_list[ran][1]]['sign'] = '+'


    elif (r == 2):
```

```
                g[tris_list[ran][1]][tris_list[ran][2]]['sign'] = '+'

            else:
                g[tris_list[ran][0]][tris_list[ran][2]]['sign'] = '+'

        return g



def Coalition(g):

        f = []
        s = []
        nodes = g.nodes()
        r = random.choice(list(nodes))

        f.append(r)
        processed_nodes = []
        to_be_processed = [r]

        for each in to_be_processed:
            if each not in processed_nodes:
                neigh = list(g.neighbors(each))
                for i in range(len(neigh)):
                    if (g[each][neigh[i]]['sign'] == '+'):
                        if (neigh[i] not in f):
                            f.append(neigh[i])
                        if (neigh[i] not in to_be_processed):
                            to_be_processed.append(neigh[i])
                    elif (g[each][neigh[i]]['sign'] == '-'):
                        if (neigh[i] not in s):
```

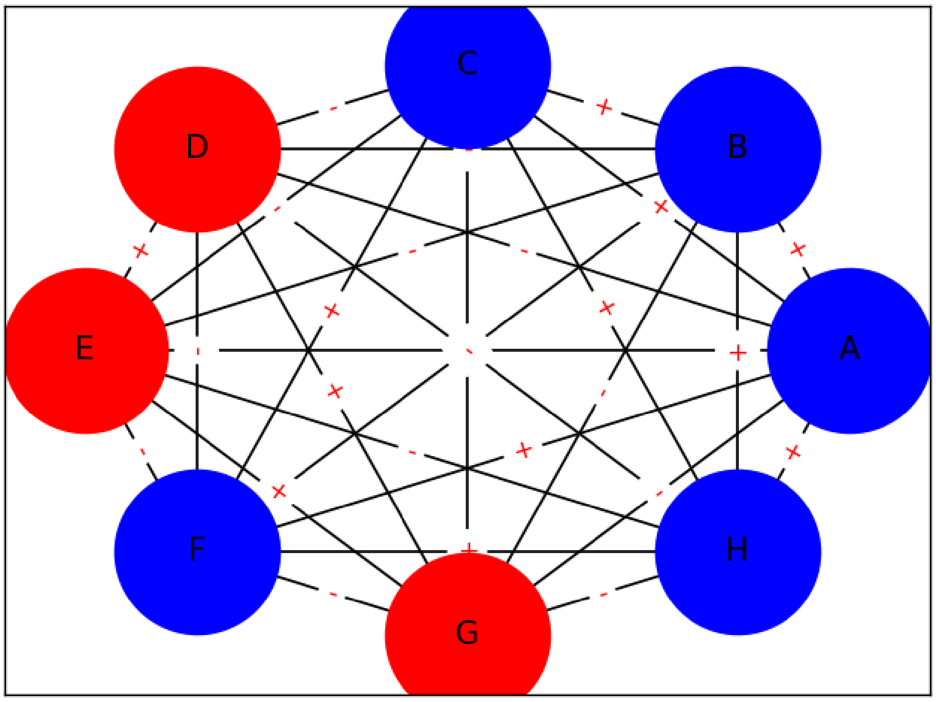
```
                        s.append(neigh[i])
                        processed_nodes.append(neigh[i])

            processed_nodes.append(each)

    return f, s



# 1.Create graph
g = nx.Graph()
n = 8
g.add_nodes_from(range(1, n + 1))
map = {1: "A", 2: "B", 3: "C", 4: "D", 5: "E",
    6: "F", 7: "G", 8: "H", 9: "I", 10: "J"}
signs = ['+', '-']
g = nx.relabel_nodes(g, map)

# 2.Add every possible edge and assign sign
for i in g.nodes():
    for j in g.nodes():
        if (i != j):
            g.add_edge(i, j, sign=random.choice(signs))

# 3.Display graph
edge_attributes = nx.get_edge_attributes(g, 'sign')
pos = nx.circular_layout(g)
nx.draw(g, pos, node_size=3000, with_labels=1)
nx.draw_networkx_edge_labels(
    g, pos, edge_labels=edge_attributes, font_size=20, font_color='blue')
plt.show()
```

```
# 4.1.Get list of all the triangles in network
nodes = g.nodes()
tris_list = [list(x) for x in itertools.combinations(nodes, 3)]

# 4.2.Store the sign details of all the triangles
all_signs = get_signs_of_graph(g, tris_list)

# 4.3.Count total number of unstable triangle
# in the network
unstable = unstablecount(all_signs)

# 5 chose the triangle in the graph that is unstable
# and make the triangle stable
unstable_track = [unstable]

while (unstable != 0):
    g = move_graph_to_stable(g, tris_list, all_signs)
    all_signs = get_signs_of_graph(g, tris_list)
    unstable = unstablecount(all_signs)
    unstable_track.append(unstable)

# 6 Form the coalition
first, second = Coalition(g)
print(first)
print(second)

edge_labels = nx.get_edge_attributes(g, 'sign')
pos = nx.circular_layout(g)
```

```
nx.draw_networkx_nodes(g, pos, nodelist=first,
                       node_color='red', node_size=4000)
nx.draw_networkx_nodes(g, pos, nodelist=second,
                       node_color='blue', node_size=4000)
nx.draw_networkx_labels(g, pos)
nx.draw_networkx_edges(g, pos)
nx.draw_networkx_edge_labels(g, pos, edge_labels=edge_labels, font_color="red")
plt.show()
```

Output

```
PS E:\Bsccsit\7thsem\labs> python 3_Lab_Data_transforming.py
Matplotlib is building the font cache; this may take a moment.
['E', 'D', 'G']
['A', 'B', 'C', 'F', 'H']
```



4) Lab4 – Data tranform

Source Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv("student-data.csv")
print(df[0:10])
#Sort by name
sorted = df.sort_values(by=['name'])
print(sorted[0:10])
#Filter rows
just_students = df.query('is_student==True')
print(just_students[0:10])
#Filter columns
no_birthday = df.filter(['name','is_student','target'])
print(no_birthday[0:10])
#Remove duplicates
print(df.duplicated())
dups_removed = df.drop_duplicates()
print(dups_removed[0:10])
```

Output

```
  import pandas as pd
                          participant_id               name         dob is_student  target   Q1   Q2   Q3   Q4
0  1de9ea66-70d3-4a1f-8735-df5ef7697fb9       Thomas Crosby  1996-07-16      False  809.97  1.0  5.0  0.0  0.0
1  9da618fd-7bf7-4a4d-9f8f-5ffba5f80a0a      Brandon Reeves  1957-11-30      False  866.41  1.0  8.0  1.0  1.0
2  d30aad4b-4503-4e22-8bc4-621b94398520      Maria Castillo  1981-02-03      False  776.92  3.0  8.0  0.0  1.0
3  790b2f7c-b5c3-4ec1-a4ce-01e15560eaba          Carol Jones  1990-09-17       True  929.72  3.0  0.0  1.0  2.0
4  6a8f6926-0a22-4ba8-b442-a1244e2e3761     Robert Thompson  1967-06-09       True  923.66  NaN  0.0  6.0  NaN
5  d103b91f-9536-4d68-977a-296e7ef077e0  Andrew Fitzpatrick  1997-05-27      False  939.97  8.0  4.0  1.0  NaN
6  1e956a7d-c4e1-4589-afa1-9d466a94427d      Ashley Pittman  1982-07-24      False  979.54  3.0  1.0  2.0  4.0
7  b85a4eab-fbd6-4b31-9729-a460d911b0dc          Richard Le  1962-04-05      False  934.08  4.0  3.0  1.0  NaN
8  10f44ac0-75f9-4a5f-b1ee-ddee4e3d6967       Selena Barker  1970-08-29      False  791.69  7.0  3.0  3.0  6.0
9  4d4142f8-3408-467f-95a2-90e3f3d9ff9f    Harold Hernandez  1994-04-13      False  973.24  2.0  9.0  2.0  1.0
                           participant_id               name         dob is_student  target   Q1   Q2   Q3   Q4
93  4962b084-e6be-45d1-8dbf-2fa904faee7b        Adam Odonnell  1994-05-02      False  874.94  1.0  2.0  4.0  NaN
5   d103b91f-9536-4d68-977a-296e7ef077e0  Andrew Fitzpatrick  1997-05-27      False  939.97  8.0  4.0  1.0  NaN
38  652d6d98-3eb0-49f9-825d-bd51db3883e0        Andrew Myers  1984-12-30      False  993.89  1.0  NaN  2.0  NaN
6   1e956a7d-c4e1-4589-afa1-9d466a94427d      Ashley Pittman  1982-07-24      False  979.54  3.0  1.0  2.0  4.0
99  af2fb5fe-b506-4098-bf79-c84e5a2e595e        Barbara Ford  1975-01-23      False  937.97  8.0  4.0  4.0  4.0
1   9da618fd-7bf7-4a4d-9f8f-5ffba5f80a0a      Brandon Reeves  1957-11-30      False  866.41  1.0  8.0  1.0  1.0
69  77c8bcc5-f206-40be-9c05-be867d85a031         Brian Lopez  1964-03-01      False  965.28  7.0  9.0  4.0  1.0
86  462790a1-a950-45d8-9193-832c99bb9f7a       Brian Wilkins  1984-11-20       True  875.56  7.0  4.0  1.0  1.0
39  833cd4a2-1032-4bc3-8dbd-e5aa3ac4ea87     Cameron Leonard  1958-11-17      False  779.00  4.0  NaN  0.0  6.0
3   790b2f7c-b5c3-4ec1-a4ce-01e15560eaba         Carol Jones  1990-09-17       True  929.72  3.0  0.0  1.0  2.0
                           participant_id               name         dob is_student  target   Q1   Q2   Q3   Q4
3   790b2f7c-b5c3-4ec1-a4ce-01e15560eaba         Carol Jones  1990-09-17       True  929.72  3.0  0.0  1.0  2.0
4   6a8f6926-0a22-4ba8-b442-a1244e2e3761     Robert Thompson  1967-06-09       True  923.66  NaN  0.0  6.0  NaN
10  ab11c60e-006f-4424-abc8-35ab5345e427      Margaret Chang  1967-01-04       True  740.54  6.0  NaN  3.0  NaN
14  916a0b9a-edff-4776-bf07-4b2584d44ac4      Michael Keller  1988-10-04       True  761.09  5.0  1.0  9.0  5.0
26  5f7edd34-bc6c-4a3f-8db2-9178bafc1f6b       Michelle Rice  1984-03-30       True  822.34  7.0  1.0  8.0  4.0
33  a8d16ddc-ce3e-46cd-b3b3-097984b40ffb  Elizabeth Carrillo  1991-01-07       True  902.88  6.0  2.0  2.0  1.0
36  f45fcbeb-84b8-4b91-8079-42c9adf9ca6f        Corey Walker  1994-07-29       True  992.89  7.0  0.0  6.0  6.0
48  18543641-b209-4a51-9ae5-38506364b815        Casey Howard  1965-06-28       True  892.49  9.0  0.0  5.0  9.0
58  0ddadb49-dbab-4180-9b8b-45aff9024ada    Charles Anderson  1988-12-25       True  741.85  0.0  3.0  5.0  4.0
60  41bbcbbe-4e1b-417e-bb8a-25617e6be506        Rhonda Martin  1978-07-06       True  921.21  5.0  NaN  5.0  5.0
```

```
                 name  is_student  target
0       Thomas Crosby       False  809.97
1      Brandon Reeves       False  866.41
2      Maria Castillo       False  776.92
3         Carol Jones        True  929.72
4     Robert Thompson        True  923.66
5  Andrew Fitzpatrick       False  939.97
6      Ashley Pittman       False  979.54
7          Richard Le       False  934.08
8       Selena Barker       False  791.69
9    Harold Hernandez       False  973.24
0      False
1      False
2      False
3      False
4      False
       ...
99     False
100     True
101     True
102     True
103     True
Length: 104, dtype: bool
                          participant_id               name         dob is_student  target   Q1   Q2   Q3   Q4
0  1de9ea66-70d3-4a1f-8735-df5ef7697fb9       Thomas Crosby  1996-07-16      False  809.97  1.0  5.0  0.0  0.0
1  9da618fd-7bf7-4a4d-9f8f-5ffba5f80a0a      Brandon Reeves  1957-11-30      False  866.41  1.0  8.0  1.0  1.0
2  d30aad4b-4503-4e22-8bc4-621b94398520      Maria Castillo  1981-02-03      False  776.92  3.0  8.0  0.0  1.0
3  790b2f7c-b5c3-4ec1-a4ce-01e15560eaba          Carol Jones  1990-09-17       True  929.72  3.0  0.0  1.0  2.0
4  6a8f6926-0a22-4ba8-b442-a1244e2e3761     Robert Thompson  1967-06-09       True  923.66  NaN  0.0  6.0  NaN
5  d103b91f-9536-4d68-977a-296e7ef077e0  Andrew Fitzpatrick  1997-05-27      False  939.97  8.0  4.0  1.0  NaN
6  1e956a7d-c4e1-4589-afa1-9d466a94427d      Ashley Pittman  1982-07-24      False  979.54  3.0  1.0  2.0  4.0
7  b85a4eab-fbd6-4b31-9729-a460d911b0dc          Richard Le  1962-04-05      False  934.08  4.0  3.0  1.0  NaN
8  10f44ac0-75f9-4a5f-b1ee-ddee4e3d6967       Selena Barker  1970-08-29      False  791.69  7.0  3.0  3.0  6.0
9  4d4142f8-3408-467f-95a2-90e3f3d9ff9f    Harold Hernandez  1994-04-13      False  973.24  2.0  9.0  2.0  1.0
PS E:\Bsccsit\7thsem\labs>
```

5) Perform data tranformation

Source Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#New Variable
from dateutil.relativedelta import *
from datetime import *

df = pd.read_csv("student-data.csv")
print("Original data")
print(df[0:10])
def get_age(dob):
    now = datetime.now()
    age = relativedelta(now, dob).years
    return age

df['age'] = pd.to_datetime(df['dob']).apply(get_age)

data = df.filter(['name','dob','age','is_student','target'])
print("Trabformed data")
print(data)
```

Output

```
  import pandas as pd
Original data
                         participant_id               name         dob is_student  target   Q1   Q2   Q3   Q4
0  1de9ea66-70d3-4a1f-8735-df5ef7697fb9      Thomas Crosby  1996-07-16      False  809.97  1.0  5.0  0.0  0.0
1  9da618fd-7bf7-4a4d-9f8f-5ffba5f80a0a     Brandon Reeves  1957-11-30      False  866.41  1.0  8.0  1.0  1.0
2  d30aad4b-4503-4e22-8bc4-621b94398520     Maria Castillo  1981-02-03      False  776.92  3.0  8.0  0.0  1.0
3  790b2f7c-b5c3-4ec1-a4ce-01e15560eaba        Carol Jones  1990-09-17       True  929.72  3.0  0.0  1.0  2.0
4  6a8f6926-0a22-4ba8-b442-a1244e2e3761    Robert Thompson  1967-06-09       True  923.66  NaN  0.0  6.0  NaN
5  d103b91f-9536-4d68-977a-296e7ef077e0  Andrew Fitzpatrick  1997-05-27     False  939.97  8.0  4.0  1.0  NaN
6  1e956a7d-c4e1-4589-afa1-9d466a94427d     Ashley Pittman  1982-07-24      False  979.54  3.0  1.0  2.0  4.0
7  b85a4eab-fbd6-4b31-9729-a460d911b0dc         Richard Le  1962-04-05      False  934.08  4.0  3.0  1.0  NaN
8  10f44ac0-75f9-4a5f-b1ee-ddee4e3d6967      Selena Barker  1970-08-29      False  791.69  7.0  3.0  3.0  6.0
9  4d4142f8-3408-467f-95a2-90e3f3d9ff9f   Harold Hernandez  1994-04-13      False  973.24  2.0  9.0  2.0  1.0
Trabformed data
                name         dob  age is_student  target
0      Thomas Crosby  1996-07-16   27      False  809.97
1     Brandon Reeves  1957-11-30   66      False  866.41
2     Maria Castillo  1981-02-03   43      False  776.92
3        Carol Jones  1990-09-17   33       True  929.72
4    Robert Thompson  1967-06-09   56       True  923.66
..               ...         ...  ...        ...     ...
99      Barbara Ford  1975-01-23   49      False  937.97
100   Maria Castillo  1981-02-03   43      False  776.92
101      Edward Hood  1984-01-10   40      False  975.77
102    Jeffrey Smith  1996-09-29   27      False  996.08
103   Hayley Hoffman  1958-06-23   65      False  825.04

[104 rows x 5 columns]
PS E:\Bsccsit\7thsem\labs>
                                                                                                   Ln 19, Col 23
```

6) Association Rule Mining Using the Apriori Algorithm on Store Transaction Data

Source Code

```
import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from apyori import apriori

store_data = pd.read_csv('store_data.csv',header=None)

store_data.head()

records = []

for i in range(0, 7501):

   records.append([str(store_data.values[i,j]) for j in range(0, 20)])

association_rules = apriori(records, min_support=0.0045, min_confidence=0.2, min_lift=3, min_length=2)

association_results = list(association_rules)
```

```
for item in association_results:
    pair = item[0]
    items = [x for x in pair]
    print("Rule: " + items[0] + " -> " + items[1])
    print("Support: " + str(item[1]))
    print("Confidence: " + str(item[2][0][2]))
    print("Lift: " + str(item[2][0][3]))
    print("=====================================")
```

Output

```
 import pandas as pd
Rule: light cream -> chicken
Support: 0.004532728969470737
Confidence: 0.29059829059829057
Lift: 4.84395061728395
=====================================
Rule: mushroom cream sauce -> escalope
Support: 0.005732568990801226
Confidence: 0.3006993006993007
Lift: 3.790832696715049
=====================================
Rule: escalope -> pasta
Support: 0.005865884548726837
Confidence: 0.3728813559322034
Lift: 4.700811850163794
=====================================
Rule: ground beef -> herb & pepper
Support: 0.015997866951073192
Confidence: 0.3234501347708895
Lift: 3.2919938411349285
=====================================
Rule: tomato sauce -> ground beef
Support: 0.005332622317024397
Confidence: 0.3773584905660377
Lift: 3.840659481324083
=====================================
Rule: whole wheat pasta -> olive oil
Support: 0.007998933475536596
Confidence: 0.2714932126696833
Lift: 4.122410097642296
=====================================
Rule: shrimp -> pasta
Support: 0.005065991201173177
Confidence: 0.3220338983050847
Lift: 4.506672147735896
=====================================
Rule: nan -> light cream
Support: 0.004532728969470737
Confidence: 0.29059829059829057
Lift: 4.84395061728395
```

```
Rule: nan -> mineral water
Support: 0.006665777896280496
Confidence: 0.39062500000000006
Lift: 3.975682666214383
=====================================
Rule: spaghetti -> nan
Support: 0.006399146780429276
Confidence: 0.3934426229508197
Lift: 4.004359721511667
=====================================
Rule: nan -> milk
Support: 0.004932675643247567
Confidence: 0.22424242424242427
Lift: 3.4118507591124225
=====================================
Rule: spaghetti -> nan
Support: 0.005999200106652446
Confidence: 0.5232558139534884
Lift: 3.005315360233627
=====================================
Rule: spaghetti -> nan
Support: 0.007199040127982935
Confidence: 0.20300751879699247
Lift: 3.088761457396025
=====================================
Rule: nan -> olive oil
Support: 0.005199306759098787
Confidence: 0.22543352601156072
Lift: 3.429973384609973
=====================================
Rule: pancakes -> nan
Support: 0.005065991201173177
Confidence: 0.20105820105820105
Lift: 3.0591025682303568
=====================================
Rule: nan -> milk
Support: 0.004532728969470737
Confidence: 0.28813559322033894
Lift: 3.0228043143297376
```

7) FP-Growth Algorithm for Mining Frequent Patterns and Generating Association Rules in Python

Source Code

```
import pyfpgrowth
transactions = [
    ['Apple', 'Banana', 'Orange'],
    ['Banana', 'Grapes'],
    ['Apple', 'Banana', 'Grapes'],
    ['Banana', 'Orange', 'Grapes'],
    ['Apple', 'Orange'],
    ['Apple', 'Banana', 'Orange', 'Grapes'],
    ['Apple', 'Grapes'],
    ['Banana', 'Grapes'],
    ['Apple', 'Banana', 'Orange']
]
FrequentPatterns=pyfpgrowth.find_frequent_patterns(transactions=transactions,support_threshold=0.5)
print(FrequentPatterns)
# Generating rules with min confidence threshold=0.5
print("Generating rules with min confidence threshold=0.5")
Rules=pyfpgrowth.generate_association_rules(patterns=FrequentPatterns,confidence_threshold=0.5)
print(Rules)
```

Output

```
PS E:\Bsccsit\7thsem\labs> python 6_FP_Growth_algorittm.py
{('Grapes', 'Orange'): 2, ('Banana', 'Grapes', 'Orange'): 2, ('Apple', 'Grapes', 'Orange'): 1, ('Apple', 'B
anana', 'Grapes', 'Orange'): 1, ('Apple', 'Orange'): 4, ('Banana', 'Orange'): 4, ('Apple', 'Banana', 'Orang
e'): 3, ('Apple',): 6, ('Apple', 'Banana'): 4, ('Apple', 'Grapes'): 3, ('Apple', 'Banana', 'Grapes'): 2, ('
Banana', 'Grapes'): 5, ('Banana',): 7}
Generating rules with min confidence threshold=0.5
{('Banana', 'Orange'): (('Apple',), 0.75), ('Grapes', 'Orange'): (('Apple', 'Banana'), 0.5), ('Apple', 'Ban
ana', 'Grapes'): (('Orange',), 0.5), ('Apple', 'Grapes', 'Orange'): (('Banana',), 1.0), ('Banana', 'Grapes'
, 'Orange'): (('Apple',), 0.5), ('Apple',): (('Grapes',), 0.5), ('Banana',): (('Grapes',), 0.7142857142857143), ('Apple', 'Banana'): (('Grapes',), 0.5), ('Apple', 'Orange'): (('Banana',), 0.75), ('Apple', 'Grapes')
: (('Banana',), 0.6666666666666666)}
PS E:\Bsccsit\7thsem\labs>
```

8) Diabetes Prediction Using Naive Bayes Classifier

Source Code

```
#Diabetes Prediction Using Naive Bayes Classifier
import pandas as pd
from sklearn import metrics
from sklearn.naive_bayes import GaussianNB
dataset = pd.read_csv('Diabetes.csv')
split = int(len(dataset)*0.7)
train, test = dataset[:split], dataset[split:]
p = train['Pragnency'].values
g = train['Glucose'].values
bp= train['Blod Pressure'].values
st= train['Skin Thikness'].values
ins= train['Insulin'].values
bmi= train['BMI'].values
dfp= train['DFP'].values
a= train['Age'].values
d= train['Diabetes'].values
trainfeatures=zip(p,g,bp,st,ins,bmi,dfp,a)
traininput=list(trainfeatures)
model = GaussianNB()
model.fit(traininput,d)
p = test['Pragnency'].values
g = test['Glucose'].values
bp= test['Blod Pressure'].values
st= test['Skin Thikness'].values
ins= test['Insulin'].values
bmi= test['BMI'].values
dpf= test['DFP'].values
a= test['Age'].values
```

```
d= test['Diabetes'].values
testfeatures=zip(p,g,bp,st,ins,bmi,dpf,a)
testinput=list(testfeatures)
predicted= model.predict(testinput)
print("Actual Class:   ", *d)
print("Predicted Class:", *predicted)

print("Confusion Matrix")
print(metrics.confusion_matrix(d, predicted))
print("********Classifiaction Measures*********")
print("Accuracy:",metrics.accuracy_score(d,predicted))
print("Recall:",metrics.recall_score(d,predicted))
print("Precision:",metrics.precision_score(d,predicted))
print("F1-Score:",metrics.f1_score(d,predicted))
```

Output

```
import pandas as pd
Actual Class:    0 0 1 1 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 1 0
0 1 0 0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 1 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 1 0
 1 1 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 1 0 0 1 0 0 1 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 1 1 0
0 0 0 0 0 1 1 0 0 1 0 0 1 0 1 1 1 0 0 1 1 1 0 1 0 1 0 1 0 0 0 0 1 0
Predicted Class: 0 0 1 1 0 1 0 0 1 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1 0 1 0 1 0 1 0 1 0 0 1 1 1 0
0 1 0 0 0 0 1 1 0 1 0 1 0 0 1 1 0 1 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 1 1 0 1 1 1 1 0 0
 0 0 0 1 1 0 1 1 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 1 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 0 0 1 1 1 1 0 1 0 0 0 0 1 1 1 1 0 0 1 0 1 0 1 0 0 0 0
Confusion Matrix
[[128  24]
 [ 30  49]]
********Classifiaction Measures*********
Accuracy: 0.7662337662337663
Recall: 0.620253164556962
Precision: 0.6712328767123288
F1-Score: 0.6447368421052632
PS E:\Bsccsit\7thsem\labs>
```

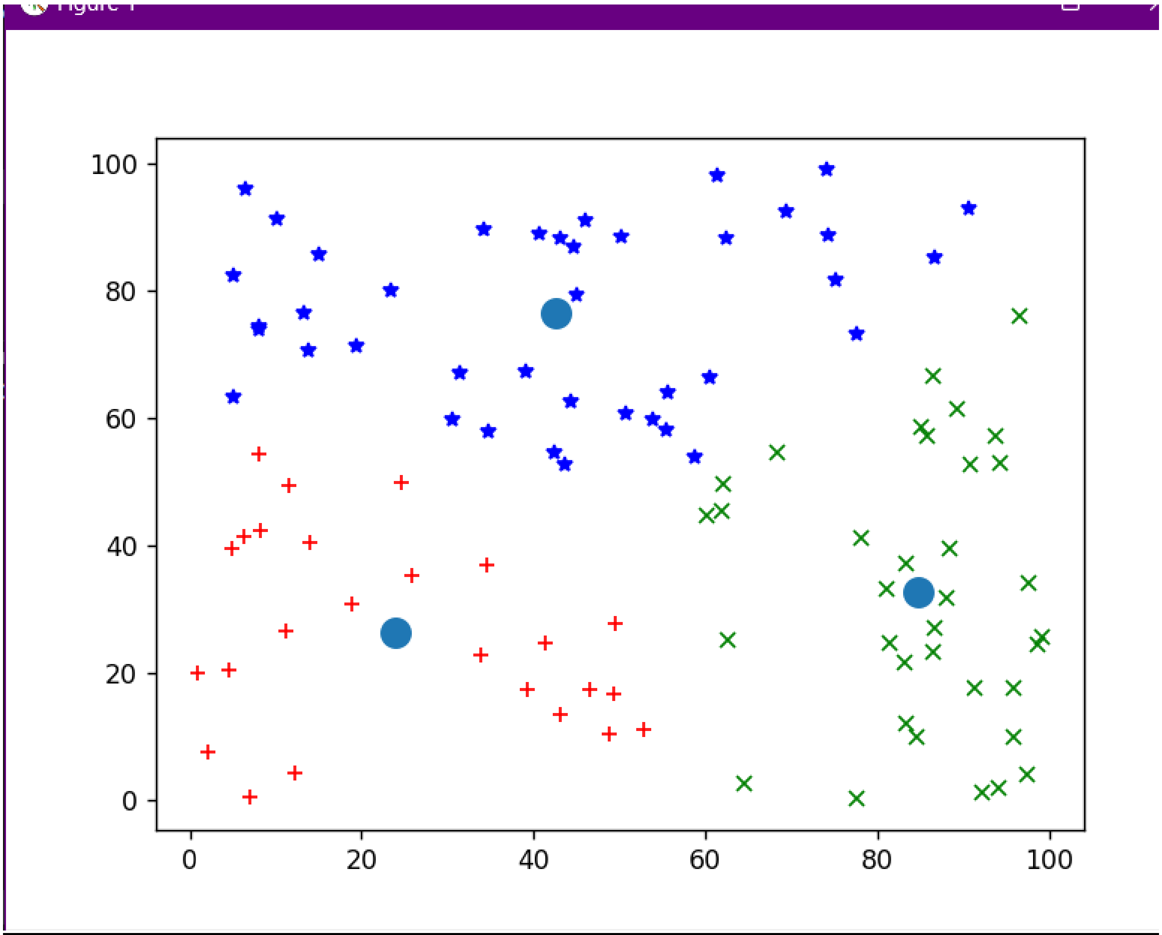
9) Visualization of Clustering with K-Means Algorithm.

Source Code

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

data=100*np.random.rand(100,2)
print(*data)
km=KMeans(n_clusters=3,init='random')
km.fit(data)
centers = km.cluster_centers_
labels = km.labels_
print("Cluster Centers:",*centers)
print("Cluster Labels:",*labels)

colors = ["r","g","b"]
markers=["+","x","*"]
for i in range(len(data)):
    plt.plot(data[i][0], data[i][1], color=colors[labels[i]], marker=markers[labels[i]])
plt.scatter(centers[:, 0],centers[:, 1], marker = "o", s=50, linewidths = 5)
plt.show()
```

Output

```
PS E:\Bsccsit\7thsem\labs> python lab9.py
E:\Bsccsit\7thsem\labs\lab9.py:2: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better interoperability with other libraries)
but was not found to be installed on your system.
.12866462] [91.31990594 17.5716752 ] [13.98410153 40.38501664] [48.86322776 10.36205168] [81.42897722 24.63290359] [60.43298714 66.21064439] [77.6206182
2 73.09739914] [78.11651542 41.04132734] [46.56120406 17.44241751] [75.08209493 81.47263995] [13.85917747 70.54039373] [87.98903165 31.80569304] [43.021
25336 88.21507064] [44.72421054 86.76679665] [62.6395942  25.10409603] [31.36068617 67.0909134 ] [97.38064686  3.91571571] [74.26115618 88.69574876] [55
.70389811 63.83517683] [25.83785244 35.26054001] [ 8.03514927 74.36784931] [30.52190572 59.64272728] [11.22971258 26.60867949] [19.45634793 71.18409758]
 [68.33759311 54.39912741] [96.59403146 75.8940226 ] [33.7859615  22.86859554] [90.78503695 52.65632508] [85.74604813 57.15853036] [77.61668218  0.18934
944] [86.41359283 66.55351919] [55.52560219 57.97347282] [86.54228859 26.96195254] [43.5272206  52.61650115] [94.35342982 52.75837259] [52.77348532 10.9
0042226] [ 0.96551421 19.99310284] [83.2283731 37.0150487] [24.58835625 49.81424384] [11.48952433 49.4216508 ] [23.33082502 79.95218092] [64.51139203  2
.45737551] [2.18185815 7.48317624] [41.2906258  24.72598193] [53.82142754 59.65694742] [81.12236214 33.0749988 ] [62.03492332 49.68632568] [ 4.89413026
39.36815557] [44.37800974 62.5330375 ] [83.15145909 21.50023476] [89.16553222 61.37821708] [ 5.13405566 63.21829366] [83.36126676 11.99468097] [98.61491
214 24.39881794] [12.19900623  4.14095616] [34.57891959 36.76361283] [ 4.6489168  20.35776538] [95.86709583  9.76831821] [93.76641882 57.0619156 ] [39.2
3030238 17.29477231] [15.08893894 85.61664759] [40.65360172 88.90910013] [ 6.39247329 41.36870475] [44.93168256 79.21560799] [34.77158943 57.77432826] [
42.4151951  54.40203503] [7.0993647  0.46717727] [13.36691858 76.44808938] [74.11997465 98.95998996] [49.34839097 16.70510019] [34.15421769 89.64208754]
 [97.6306026  33.93834501] [ 8.32343073 42.37007948] [ 5.14398483 82.29455295] [69.3066309  92.31918099] [84.58885229  9.83375167] [90.61876504 92.81106
649] [10.21660587 91.22553751] [99.12963088 25.66662307] [50.78872826 60.66324083] [49.49657601 27.7221447 ] [43.0931883  13.35000451] [85.05941449 58.4
6682685] [95.88748718 17.53457078] [61.31460099 98.09538317] [ 6.49101809 95.80931827] [86.56828401 85.08042264] [88.41383091 39.52894699] [ 8.02510592
73.83830164] [46.07478946 91.07723627] [ 8.05044218 54.35666355] [18.81111954 30.65608239] [39.12225252 67.20185172] [50.13944718 88.45171543] [92.25818
597  1.1373014 ] [62.38085499 88.21570235]
Cluster Centers: [23.9887074  26.40748392] [84.72977048 32.57040763] [42.60763767 76.47513391]
Cluster Labels: 1 1 1 2 1 1 0 0 1 2 2 1 0 2 2 1 2 2 1 2 1 2 2 0 2 2 0 2 1 1 0 1 1 1 1 2 1 2 1 0 0 1 0 0 2 1 0 0 2 1 1 0 2 1 1 2 1 1 0 0 0 1 1 0 2 2 0 2
2 2 0 2 2 0 2 1 0 2 2 1 2 2 1 2 0 0 1 1 2 2 2 1 2 2 0 0 2 2 1 2
PS E:\Bsccsit\7thsem\labs>
```



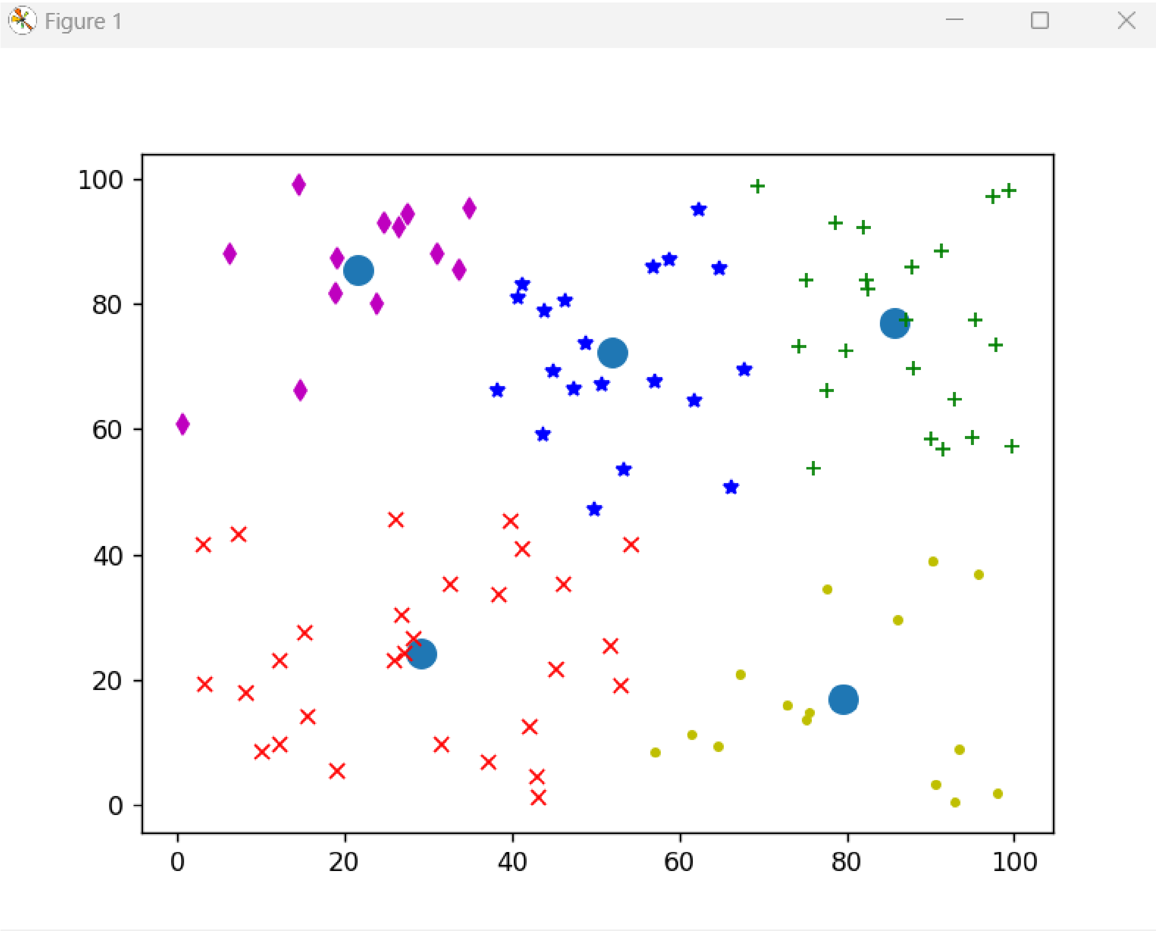
10) Mini-Batch K-Means Clustering Visualization and Performance Measurement

Source Code

```
import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MiniBatchKMeans
data=100*np.random.rand(100,2)
print(*data)
mbk=MiniBatchKMeans(n_clusters=5,init='random', batch_size=3000)
t0= time.time()
mbk.fit(data)
t1= time.time()
tt=t1-t0
print("Total Time:",tt)
cents = mbk.cluster_centers_
labels = mbk.labels_
print("Cluster Centers:",*cents)
print("Labels:",*labels)
colors = ["g","r","b",'y','m']
markers=["+","x","*",'.','d']
for i in range(len(data)):
    plt.plot(data[i][0], data[i][1], color=colors[labels[i]], marker=markers[labels[i]])
plt.scatter(cents[:, 0],cents[:, 1], marker = "o", s=50, linewidths = 5)
plt.show()
```

Output

```
  import pandas as pd
[48.77318459 73.57745029] [ 3.28151047 19.26611623] [43.01753122  4.53462209] [10.16671174  8.57946664] [75.17413401 13.60162145] [37.12298805  6.805115
52] [64.84539984 85.56028588] [50.77360398 67.14709485] [38.36046117 33.46929435] [34.9072699  95.23609919] [91.5364231 56.6917198] [46.30996615 80.5624
4356] [77.67690033 34.40635805] [24.8097367  92.85168438] [57.14758269 67.45244836] [27.62332385 94.33384801] [15.60421554 14.10887185] [ 7.39842086 43.
16307576] [ 3.10540876 41.56624202] [44.94974126 69.08585145] [87.04936696 77.42375168] [ 6.32491608 87.95113599] [53.38178312 53.46569768] [99.72227504
 57.14802604] [32.57574762 35.28024101] [14.5607391 98.9734044] [79.88366568 72.37386536] [77.65145009 66.18035538] [49.7918912  47.04822537] [26.496352
83 92.26434492] [92.90707385 64.78657369] [76.02343684 53.7134551 ] [56.93546124 85.94101645] [46.09975164 35.29692896] [51.80416103 25.29437571] [12.32
95756  9.6454934] [97.76240162 73.49576412] [38.25976695 66.18016671] [ 0.69555251 60.66243014] [27.21602916 24.20794205] [92.92451103  0.49353054] [67.
79346171 69.5183356 ] [87.81812874 85.7732784 ] [31.04432428 87.91177871] [58.83838011 86.93121946] [45.27029223 21.6062187 ] [57.08441887  8.36126614]
[15.30430543 27.41750929] [78.73756243 92.98122819] [95.62821414 36.88342512] [42.17479087 12.49539948] [85.98023154 29.68027861] [41.22238714 83.049919
73] [43.88067103 78.83059667] [82.00363106 92.13389898] [90.03450226 58.31983086] [26.77713311 30.37027654] [82.4467934 83.6760045] [19.09707634 87.1582
5742] [43.21880756  1.22132467] [87.92559318 69.68148144] [64.6212775   9.37897669] [31.5519907   9.69406917] [98.02618904  1.82797624] [90.61297957  3.
27163044] [54.18158313 41.42520611] [90.31446774 38.99430634] [28.24317565 26.4083402 ] [ 8.27212893 17.87522981] [12.34521353 23.10384239] [41.32905652
 40.93259543] [67.30969127 20.92624056] [39.82643398 45.20143091] [53.07236968 19.06975041] [61.4472625  11.20491134] [40.71109426 80.90037547] [18.9983
7908 81.54458374] [93.51500422  8.83303773] [99.38380265 98.0879168 ] [19.09956716  5.39973389] [75.52863531 14.90636305] [26.09169674 45.5061833 ] [75.
13867281 83.75931606] [69.44444772 98.84141142] [82.60442343 82.35121006] [97.43341689 97.02074164] [47.31873189 66.47007976] [33.63749409 85.31340688]
[91.367123   88.35239043] [61.76741922 64.3905046 ] [72.90814922 15.87986057] [66.18265944 50.57389584] [23.83701561 80.09177236] [95.42530821 77.453079
92] [95.04181977 58.59980504] [43.73341799 59.21226144] [14.69151199 66.12310577] [74.36389938 73.07715717] [62.29165061 94.93263989] [26.03507812 23.10
888011]
Total Time: 0.7394924163818359
Cluster Centers: [85.77892859 76.83593627] [29.1250Close24.18943461] [51.94503161 72.27789456] [79.54050269 17.01698965] [21.54766764 85.27913741]
Labels: 2 1 1 1 3 1 2 2 1 4 0 2 3 4 2 4 1 1 1 2 0 4 2 0 1 4 0 0 2 4 0 0 2 1 1 1 0 2 4 1 3 2 0 4 2 1 3 1 0 3 1 3 2 2 0 0 1 0 4 1 0 3 1 3 3 1 3 1 1 1 1 3
1 1 3 2 4 3 0 1 3 1 0 0 0 0 2 4 0 2 3 2 4 0 0 2 4 0 2 1
PS E:\Bsccsit\7thsem\labs>
```



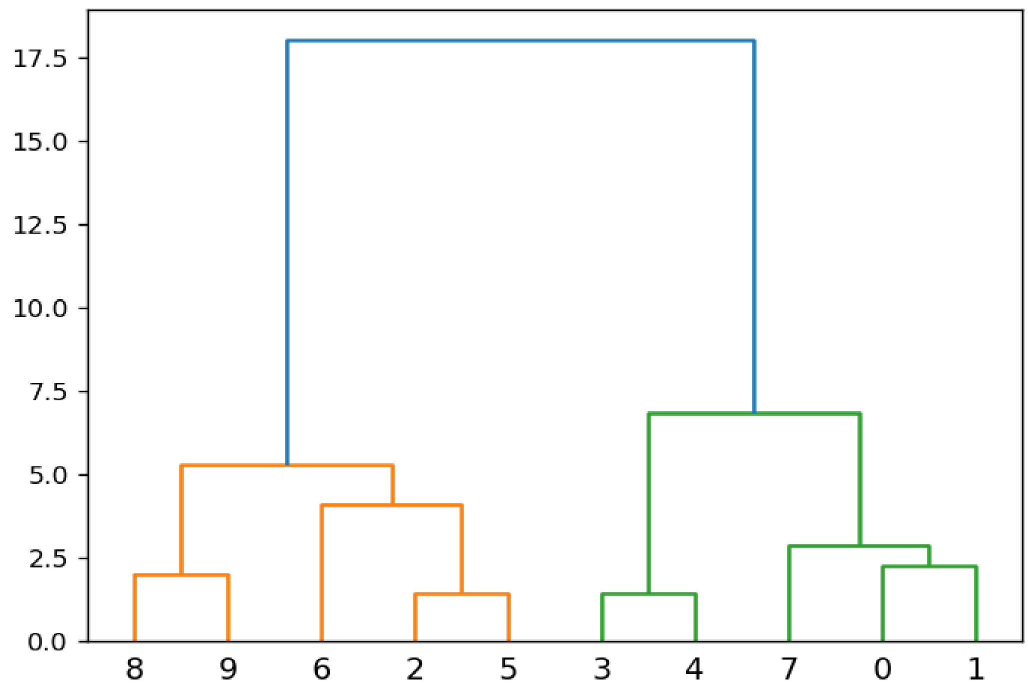
11) Hierarchical Clustering and Agglomerative Clustering for Data Visualization.

Source Code

#Hierarchical Clustering

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]
data = list(zip(x, y))
linkage_data = linkage(data, method='ward', metric='euclidean')
dendrogram(linkage_data)
plt.show()
```

Output



Source Code

```
#Agglomerative Clustering
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering

x = [4, 5, 10, 4, 3, 11, 14 , 6, 10, 12]
y = [21, 19, 24, 17, 16, 25, 24, 22, 21, 21]

data = list(zip(x, y))

hierarchical_cluster = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')
labels = hierarchical_cluster.fit_predict(data)
print(labels)
plt.scatter(x, y, c=labels)
plt.show()
```

Output