Date : 18/04/2024

EXPERIMENT NO. 9

AIM: To implement association rule, DBSCAN and PCA using python programming.

SOFTWARE USED: Jupyter Notebook.

THEORY:

Association rule mining, DBSCAN (Density-Based Spatial Clustering of Applications with Noise), and PCA (Principal Component Analysis) are three distinct machine learning and data analysis techniques used in different contexts. Here's an overview of the theory behind each of these methods:

Association Rule Mining:

- Association rule mining is a data mining technique used to find interesting relationships (or associations) between different sets of items in large datasets.
- Support: Measures how frequently an itemset appears in a dataset. High support indicates that an itemset is common.
- Confidence: Measures the likelihood that an association rule is true given that the antecedent is true. High confidence indicates a strong relationship.
- Lift: Measures how much more likely the consequent is given the antecedent compared to its overall frequency in the dataset. Lift greater than 1 indicates a positive association.
- Apriori Algorithm: A common algorithm used in association rule mining. It generates candidate itemsets and prunes them based on support thresholds, then uses these frequent itemsets to generate association rules.

DBSCAN:

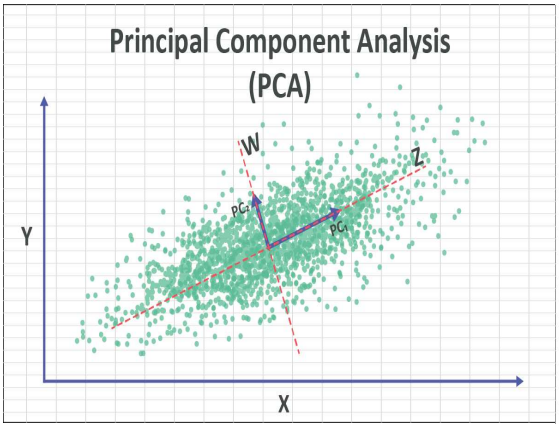
- DBSCAN is a clustering algorithm that groups data points based on density.
- Core Point: A point is a core point if it has a specified minimum number of points (minPts) within a specified distance (epsilon or eps).
- Density Reachable: Points within the neighborhood of a core point are density-reachable, meaning they belong to the same cluster.
- Density Connected: Points are density connected if they are both density reachable from a common point.
- Noise Points: Points that are not part of any cluster (neither core points nor density reachable) are labeled as noise.
- Advantages: DBSCAN can find clusters of arbitrary shapes and is robust to outliers.

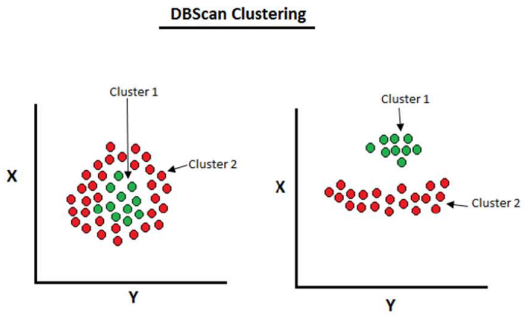
PCA:

- PCA is a dimensionality reduction technique used to reduce the number of variables in a dataset while preserving the most important information.

- Covariance Matrix: The covariance matrix represents the relationships between variables in a dataset. PCA calculates the eigenvalues and eigenvectors of this matrix.
- Principal Components: The eigenvectors with the largest eigenvalues represent the directions of maximum variance in the data and are called principal components.
- Projection: The original data is projected onto the principal components to reduce its dimensionality.
- Variance Explained: PCA aims to maximize the variance explained by the principal components. This helps to capture the most important information in fewer dimensions.
- Applications: PCA is used for visualization, noise reduction, and feature extraction.

Together, these methods provide powerful tools for data analysis, clustering, and dimensionality reduction in various machine learning and data mining applications.





OUTPUT CODE:

DSA Experiment 9
DBSCAN
18th April 2024

DBSCAN

In [1]:
```
import numpy as mp
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch
from sklearn.cluster import DBSCANb
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
df=pd.read_csv("Wholesale customers data.csv")
df.head()
```

Out[2]:

	Channel	Region	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen
0	2	3	12669	9656	7561	214	2674	1338
1	2	3	7057	9810	9568	1762	3293	1776
2	2	3	6353	8808	7684	2405	3516	7844
3	1	3	13265	1196	4221	6404	507	1788
4	2	3	22615	5410	7198	3915	1777	5185

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 440 entries, 0 to 439
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Channel           440 non-null    int64
 1   Region            440 non-null    int64
 2   Fresh             440 non-null    int64
 3   Milk              440 non-null    int64
 4   Grocery           440 non-null    int64
 5   Frozen            440 non-null    int64
 6   Detergents_Paper  440 non-null    int64
 7   Delicassen        440 non-null    int64
dtypes: int64(8)
memory usage: 27.6 KB
```

In [4]:
```
df.drop(['Channel','Region'],axis=1,inplace=True)
```

In [5]:
```
array=df.values
```

In [6]:
```
array
```

Out[6]:
```
array([[12669,  9656,  7561,   214,  2674,  1338],
       [ 7057,  9810,  9568,  1762,  3293,  1776],
       [ 6353,  8808,  7684,  2405,  3516,  7844],
       ...,
       [14531, 15488, 30243,   437, 14841,  1867],
       [10290,  1981,  2232,  1038,   168,  2125],
       [ 2787,  1698,  2510,    65,   477,    52]], dtype=int64)
```

In [7]:
```
stscaler=StandardScaler().fit(array)
X = stscaler.transform(array)
```

In [8]:
```
X
```

Out[8]:
```
array([[ 0.05293319,  0.52356777, -0.04111489, -0.58936716, -0.04356873,
        -0.06633906],
       [-0.39130197,  0.54445767,  0.17031835, -0.27013618,  0.08640684,
         0.08915105],
       [-0.44702926,  0.40853771, -0.0281571 , -0.13753572,  0.13323164,
         2.24329255],
       ...,
       [ 0.20032554,  1.31467078,  2.34838631, -0.54337975,  2.51121768,
         0.12145607],
       [-0.13538389, -0.51753572, -0.60251388, -0.41944059, -0.56977032,
         0.21304614],
       [-0.72930698, -0.5559243 , -0.57322717, -0.62009417, -0.50488752,
        -0.52286938]])
```

In [10]:
```
dbscan = DBSCAN(eps=0.8,min_samples=6)
dbscan.fit(X)
```

Out[10]:
```
DBSCAN(eps=0.8, min_samples=6)
```

In [11]:
```
dbscan.labels_
```

Out[11]:
```
array([ 0,  0, -1,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
       -1,  0,  0,  0,  0, -1, -1, -1,  0,  0,  0, -1,  0,  0,  0,  0, -1,
        0,  0,  0,  0,  0, -1, -1,  0,  0,  0,  0, -1,  0, -1,  0, -1,  0,
        0,  0,  0,  0,  0, -1,  0,  0,  0,  0, -1,  0,  0,  0, -1,  0,  0,
        0,  0,  0, -1,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,
       -1, -1, -1,  0,  0,  0,  0, -1, -1,  0,  0,  0,  0,  0,  0, -1,  0,
        0, -1,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0, -1, -1,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0, -1,  0,  0,  0,  0,
        0, -1,  0,  0,  0,  0, -1,  0,  0,  0,  0, -1,  0, -1,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0, -1, -1,  0,
        0,  0,  0,  0,  0, -1,  0, -1,  0,  0,  0,  0, -1,  0, -1,  0,  0,
        0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0, -1, -1,
        0,  0,  0, -1, -1,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0, -1,  0,  0,  0,  0, -1,  0, -1,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0,
        0,  0,  0, -1,  0,  0, -1,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,
        0,  0, -1,  0,  0,  0,  0,  0, -1,  0, -1,  0,  0,  0,  0, -1,  0,
        0,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0,
       -1, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0,
        0,  0, -1,  0,  0,  0,  0,  0, -1,  0, -1,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,
        0, -1,  0, -1,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
       -1,  0, -1,  0,  0, -1, -1,  0,  0,  0, -1,  0, -1,  0,  0],
      dtype=int64)
```

In [12]:
```
cl=pd.DataFrame(dbscan.labels_,columns=['cluster'])
```

In [13]:
```
cl
```

Out[13]:

	cluster
0	0
1	0
2	-1
3	0
4	-1
...	...
435	-1
436	0
437	-1
438	0
439	0

440 rows × 1 columns

In [14]:
```
pd.concat([df,cl],axis=1)
```

Out[14]:

	Fresh	Milk	Grocery	Frozen	Detergents_Paper	Delicassen	cluster
0	12669	9656	7561	214	2674	1338	0
1	7057	9810	9568	1762	3293	1776	0
2	6353	8808	7684	2405	3516	7844	-1
3	13265	1196	4221	6404	507	1788	0
4	22615	5410	7198	3915	1777	5185	-1
...	...	...	...	...	...	...	...
435	29703	12051	16027	13135	182	2204	-1
436	39228	1431	764	4510	93	2346	0
437	14531	15488	30243	437	14841	1867	-1
438	10290	1981	2232	1038	168	2125	0
439	2787	1698	2510	65	477	52	0

440 rows × 7 columns

Association Rule

In [18]:
```
from mlxtend.frequent_patterns import apriori,association_rules
```

In [16]:
```
titanic = pd.read_csv("Titanic.csv")
titanic.head()
```

Out[16]:

	Class	Gender	Age	Survived
0	3rd	Male	Child	No
1	3rd	Male	Child	No
2	3rd	Male	Child	No
3	3rd	Male	Child	No
4	3rd	Male	Child	No

In [17]:
```
df=pd.get_dummies(titanic)
df.head()
```

Out[17]:

	Class_1st	Class_2nd	Class_3rd	Class_Crew	Gender_Female	Gender_Male	Age_Adult	Age_Child	Survived_No	Survived_Yes
0	0	0	1	0	0	1	0	1	1	0
1	0	0	1	0	0	1	0	1	1	0
2	0	0	1	0	0	1	0	1	1	0
3	0	0	1	0	0	1	0	1	1	0
4	0	0	1	0	0	1	0	1	1	0

In [22]:
```
# apropri algo
frequent_itemsets = apriori(df,min_support=0.1,use_colnames=True)
frequent_itemsets
```

```
C:\ProgramData\Anaconda3\lib\site-packages\mlxtend\frequent_patterns\fpcommon.py:109: De
s with non-bool types result in worse computationalperformance and their support might b
e.Please use a DataFrame with bool type
  warnings.warn(
```

Out[22]:

	support	itemsets
0	0.147660	(Class_1st)
1	0.129487	(Class_2nd)
2	0.320763	(Class_3rd)
3	0.402090	(Class_Crew)
4	0.213539	(Gender_Female)
5	0.786461	(Gender_Male)
6	0.950477	(Age_Adult)
7	0.676965	(Survived_No)
8	0.323035	(Survived_Yes)
9	0.144934	(Class_1st, Age_Adult)
10	0.118582	(Age_Adult, Class_2nd)
11	0.231713	(Gender_Male, Class_3rd)
12	0.284871	(Age_Adult, Class_3rd)
13	0.239891	(Survived_No, Class_3rd)

34	0.304407	(Age_Adult, Class_Crew, Survived_No, Gender_Male)

In [23]:
```
rules= association_rules(frequent_itemsets, metric="lift",min_threshold=0.7)
rules
```

Out[23]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(Class_1st)	(Age_Adult)	0.147660	0.950477	0.144934	0.981538	1.032680	0.004587	2.682493	0.037128
1	(Age_Adult)	(Class_1st)	0.950477	0.147660	0.144934	0.152486	1.032680	0.004587	1.005694	0.639010
2	(Age_Adult)	(Class_2nd)	0.950477	0.129487	0.118582	0.124761	0.963505	-0.004492	0.994601	-0.433377
3	(Class_2nd)	(Age_Adult)	0.129487	0.950477	0.118582	0.915789	0.963505	-0.004492	0.588085	-0.041697
4	(Gender_Male)	(Class_3rd)	0.786461	0.320763	0.231713	0.294627	0.918520	-0.020555	0.962947	-0.293496
...	...	...	...	...	...	...	...	...	...	...
101	(Survived_No, Gender_Male)	(Age_Adult, Class_Crew)	0.619718	0.402090	0.304407	0.491202	1.221623	0.055225	1.175143	0.477059
102	(Age_Adult)	(Class_Crew, Survived_No, Gender_Male)	0.950477	0.304407	0.304407	0.320268	1.052103	0.015075	1.023334	1.000000
103	(Class_Crew)	(Age_Adult, Survived_No, Gender_Male)	0.402090	0.603816	0.304407	0.757062	1.253795	0.061619	1.630802	0.338549
104	(Survived_No)	(Age_Adult, Class_Crew, Gender_Male)	0.676965	0.391640	0.304407	0.449664	1.148157	0.039280	1.105434	0.399458
105	(Gender_Male)	(Age_Adult, Class_Crew, Survived_No)	0.786461	0.305770	0.304407	0.387060	1.265851	0.063931	1.132622	0.983509

106 rows × 10 columns

In [24]:
```
rules.sort_values('lift',ascending=False)[0:20]
```

Out[24]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
65	(Gender_Female, Age_Adult)	(Survived_Yes)	0.193094	0.323035	0.143571	0.743529	2.301699	0.081195	2.639542	0.700873
68	(Survived_Yes)	(Gender_Female, Age_Adult)	0.323035	0.193094	0.143571	0.444444	2.301699	0.081195	1.452431	0.835403
19	(Survived_Yes)	(Gender_Female)	0.323035	0.213539	0.156293	0.483826	2.265745	0.087312	1.523634	0.825219
18	(Gender_Female)	(Survived_Yes)	0.213539	0.323035	0.156293	0.731915	2.265745	0.087312	2.525187	0.710327
66	(Survived_Yes, Age_Adult)	(Gender_Female)	0.297138	0.213539	0.143571	0.483180	2.262724	0.080121	1.521732	0.793974
67	(Gender_Female)	(Survived_Yes, Age_Adult)	0.213539	0.297138	0.143571	0.672340	2.262724	0.080121	2.145099	0.709577
99	(Class_Crew, Survived_No)	(Age_Adult, Gender_Male)	0.305770	0.757383	0.304407	0.995542	1.314450	0.072822	54.427079	0.344592
98	(Age_Adult, Gender_Male)	(Class_Crew, Survived_No)	0.757383	0.305770	0.304407	0.401920	1.314450	0.072822	1.160764	0.986022
47	(Age_Adult, Gender_Male)	(Class_Crew)	0.757383	0.402090	0.391640	0.517097	1.286022	0.087104	1.238157	0.916706
50	(Class_Crew)	(Age_Adult, Gender_Male)	0.402090	0.757383	0.391640	0.974011	1.286022	0.087104	9.335480	0.371976
92	(Age_Adult, Class_Crew, Survived_No)	(Gender_Male)	0.305770	0.786461	0.304407	0.995542	1.265851	0.063931	47.903983	0.302519
52	(Class_Crew, Survived_No)	(Gender_Male)	0.305770	0.786461	0.304407	0.995542	1.265851	0.063931	47.903983	0.302519

50	(Class_Crew)	(Age_Adult, Gender_Male)	0.402090	0.757383	0.391640	0.974011	1.286022	0.087104	9.335480	0.371976
92	(Age_Adult, Class_Crew, Survived_No)	(Gender_Male)	0.305770	0.786461	0.304407	0.995542	1.265851	0.063931	47.903983	0.302519
52	(Class_Crew, Survived_No)	(Gender_Male)	0.305770	0.786461	0.304407	0.995542	1.265851	0.063931	47.903983	0.302519
57	(Gender_Male)	(Class_Crew, Survived_No)	0.786461	0.305770	0.304407	0.387060	1.265851	0.063931	1.132622	0.983509
105	(Gender_Male)	(Age_Adult, Class_Crew, Survived_No)	0.786461	0.305770	0.304407	0.387060	1.265851	0.063931	1.132622	0.983509
103	(Class_Crew)	(Age_Adult, Survived_No, Gender_Male)	0.402090	0.603816	0.304407	0.757062	1.253795	0.061619	1.630802	0.338549
94	(Age_Adult, Survived_No, Gender_Male)	(Class_Crew)	0.603816	0.402090	0.304407	0.504138	1.253795	0.061619	1.205800	0.510929
51	(Gender_Male)	(Age_Adult, Class_Crew)	0.786461	0.402090	0.391640	0.497978	1.238474	0.075412	1.191004	0.901730
11	(Gender_Male)	(Class_Crew)	0.786461	0.402090	0.391640	0.497978	1.238474	0.075412	1.191004	0.901730
46	(Age_Adult, Class_Crew)	(Gender_Male)	0.402090	0.786461	0.391640	0.974011	1.238474	0.075412	8.216621	0.322047
10	(Class_Crew)	(Gender_Male)	0.402090	0.786461	0.391640	0.974011	1.238474	0.075412	8.216621	0.322047

In [25]:
```
rules[rules.lift>1]
```

Out[25]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(Class_1st)	(Age_Adult)	0.147660	0.950477	0.144934	0.981538	1.032680	0.004587	2.682493	0.037128
1	(Age_Adult)	(Class_1st)	0.950477	0.147660	0.144934	0.152486	1.032680	0.004587	1.005694	0.639010
8	(Survived_No)	(Class_3rd)	0.676965	0.320763	0.239891	0.354362	1.104747	0.022745	1.052040	0.293515
9	(Class_3rd)	(Survived_No)	0.320763	0.676965	0.239891	0.747875	1.104747	0.022745	1.281251	0.139592
10	(Class_Crew)	(Gender_Male)	0.402090	0.786461	0.391640	0.974011	1.238474	0.075412	8.216621	0.322047
...	...	...	...	...	...	...	...	...	...	...
101	(Survived_No, Gender_Male)	(Age_Adult, Class_Crew)	0.619718	0.402090	0.304407	0.491202	1.221623	0.055225	1.175143	0.477059
102	(Age_Adult)	(Class_Crew, Survived_No, Gender_Male)	0.950477	0.304407	0.304407	0.320268	1.052103	0.015075	1.023334	1.000000
103	(Class_Crew)	(Age_Adult, Survived_No, Gender_Male)	0.402090	0.603816	0.304407	0.757062	1.253795	0.061619	1.630802	0.338549
104	(Survived_No)	(Age_Adult, Class_Crew, Gender_Male)	0.676965	0.391640	0.304407	0.449664	1.148157	0.039280	1.105434	0.399458
105	(Gender_Male)	(Age_Adult, Class_Crew, Survived_No)	0.786461	0.305770	0.304407	0.387060	1.265851	0.063931	1.132622	0.983509

74 rows × 10 columns

Principal Component Analysis

In [27]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
```

In [28]:
```
uni = pd.read_csv("Universities (1).csv")
uni.head()
```

Out[28]:

	Univ	SAT	Top10	Accept	SFRatio	Expenses	GradRate
0	Brown	1310	89	22	13	22704	94
1	CalTech	1415	100	25	6	63575	81
2	CMU	1260	62	59	9	25026	72
3	Columbia	1310	76	24	12	31510	88
4	Cornell	1280	83	33	13	21864	90

In [29]:
```
uni.describe()
```

Out[29]:

	SAT	Top10	Accept	SFRatio	Expenses	GradRate
count	25.000000	25.000000	25.000000	25.00000	25.000000	25.000000
mean	1266.440000	76.480000	39.200000	12.72000	27388.000000	86.720000
std	108.359771	19.433905	19.727308	4.06735	14424.883165	9.057778
min	1005.000000	28.000000	14.000000	6.00000	8704.000000	67.000000
25%	1240.000000	74.000000	24.000000	11.00000	15140.000000	81.000000
50%	1285.000000	81.000000	36.000000	12.00000	27553.000000	90.000000
75%	1340.000000	90.000000	50.000000	14.00000	34870.000000	94.000000
max	1415.000000	100.000000	90.000000	25.00000	63575.000000	97.000000

In [30]:
```
uni.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25 entries, 0 to 24
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Univ      25 non-null     object
 1   SAT       25 non-null     int64
 2   Top10     25 non-null     int64
 3   Accept    25 non-null     int64
 4   SFRatio   25 non-null     int64
 5   Expenses  25 non-null     int64
 6   GradRate  25 non-null     int64
dtypes: int64(6), object(1)
memory usage: 1.5+ KB
```

In [31]:
```
uni.data=uni.iloc[:,1:]
uni.data.head()
```

```
C:\Users\Chetana\AppData\Local\Temp\ipykernel_22312\329992
e created via a new attribute name - see https://pandas.py
s
  uni.data=uni.iloc[:,1:]
```

Out[31]:

	SAT	Top10	Accept	SFRatio	Expenses	GradRate
0	1310	89	22	13	22704	94
1	1415	100	25	6	63575	81
2	1260	62	59	9	25026	72
3	1310	76	24	12	31510	88
4	1280	83	33	13	21864	90

In [32]:
```
UNI =uni.data.values
UNI
```

Out[32]:
```
array([[ 1310,    89,    22,    13, 22704,    94],
       [ 1415,   100,    25,     6, 63575,    81],
       [ 1260,    62,    59,     9, 25026,    72],
       [ 1310,    76,    24,    12, 31510,    88],
       [ 1280,    83,    33,    13, 21864,    90],
       [ 1340,    89,    23,    10, 32162,    95],
       [ 1315,    90,    30,    12, 31585,    95],
       [ 1255,    74,    24,    12, 20126,    92],
       [ 1400,    91,    14,    11, 39525,    97],
       [ 1305,    75,    44,     7, 58691,    87],
       [ 1380,    94,    30,    10, 34870,    91],
       [ 1260,    85,    39,    11, 28052,    89],
       [ 1255,    81,    42,    13, 15122,    94],
       [ 1081,    38,    54,    18, 10185,    80],
       [ 1375,    91,    14,     8, 30220,    95],
       [ 1005,    28,    90,    19,  9066,    69],
       [ 1360,    90,    20,    12, 36450,    93],
       [ 1075,    49,    67,    25,  8704,    67],
       [ 1240,    95,    40,    17, 15140,    78],
       [ 1290,    75,    50,    13, 38380,    87],
       [ 1180,    65,    68,    16, 15470,    85],
       [ 1285,    80,    36,    11, 27553,    90],
       [ 1225,    77,    44,    14, 13349,    92],
       [ 1085,    40,    69,    15, 11857,    71],
       [ 1375,    95,    19,    11, 43514,    96]], dtype=int64)
```

In [33]:
```
uni_normal=scale(UNI)
uni_normal
```

Out[33]:
```
array([[ 0.41028362,  0.6575195 , -0.88986682,  0.07026045, -0.33141256,
         0.82030265],
       [ 1.39925928,  1.23521235, -0.73465749, -1.68625071,  2.56038138,
        -0.64452351],
       [-0.06065717, -0.76045386,  1.02438157, -0.93346022, -0.16712136,
        -1.65863393],
       [ 0.41028362, -0.02520842, -0.78639393, -0.18066972,  0.29164871,
         0.14422904],
       [ 0.12771914,  0.34241431, -0.32076595,  0.07026045, -0.39084607,
         0.36958691],
       [ 0.69284809,  0.6575195 , -0.83813038, -0.68253005,  0.33778044,
         0.93298158],
       [ 0.4573777 ,  0.71003703, -0.47597528, -0.18066972,  0.29695528,
         0.93298158],
       [-0.10775125, -0.13024348, -0.78639393, -0.18066972, -0.51381683,
         0.59494478],
       [ 1.25797704,  0.76255456, -1.30375836, -0.43159988,  0.85874344,
         1.15833946],
       [ 0.36318954, -0.07772595,  0.24833493, -1.43532055,  2.21481798,
         0.0315501 ],
       [ 1.06960072,  0.92010716, -0.47597528, -0.68253005,  0.52938275,
         0.48226584],
       [-0.06065717,  0.44744937, -0.01034729, -0.43159988,  0.04698077,
         0.25690797],
       [-0.10775125,  0.23737924,  0.14486204,  0.07026045, -0.86787073,
         0.82030265],
       [-1.7466252 , -2.02087462,  0.76569936,  1.32491127, -1.21718409,
        -0.75720245],
```

```
[ 0.1077....,  0.2373....,  0.1448....,  0.0702....,  -0.8678....,
  0.82030265],
[-1.7466252 , -2.02087462,  0.76569936,  1.32491127, -1.21718409,
 -0.75720245],
[ 1.02250664,  0.76255456, -1.30375836, -1.18439038,  0.20037583,
  0.93298158],
[-2.46245521, -2.54604994,  2.6282113 ,  1.57584144, -1.29635802,
 -1.99667073],
[ 0.88122441,  0.71003703, -0.9933397 , -0.18066972,  0.64117435,
  0.70762371],
[-1.8031381 , -1.44318177,  1.43827311,  3.08142243, -1.32197103,
 -2.22202861],
[-0.24903349,  0.97262469,  0.04138915,  1.07398111, -0.86659715,
 -0.98256032],
[ 0.2219073 , -0.07772595,  0.55875358,  0.07026045,  0.77772991,
  0.0315501 ],
[-0.81416244, -0.60290126,  1.49000956,  0.82305094, -0.84324827,
 -0.19380777],
[ 0.17481322,  0.18486171, -0.16555662, -0.43159988,  0.01167444,
  0.36958691],
[-0.39031573,  0.02730912,  0.24833493,  0.32119061, -0.99331788,
  0.59494478],
[-1.70894994, -1.91583956,  1.541746  ,  0.57212078, -1.09888311,
 -1.77131286],
[ 1.02250664,  0.97262469, -1.04507615, -0.43159988,  1.14098185,
  1.04566052]])
```

In [35]:
```
pca=PCA()
pca_values=pca.fit_transform(uni_normal)
pca_values
```

Out[35]:
```
array([[-1.00987445e+00, -1.06430962e+00,  8.10663051e-02,
         5.69506350e-02, -1.28754245e-01, -3.46496377e-02],
       [-2.82223781e+00,  2.25904458e+00,  8.36828830e-01,
         1.43844644e-01, -1.25961913e-01, -1.80703168e-01],
       [ 1.11246577e+00,  1.63120889e+00, -2.66786839e-01,
         1.07507502e+00, -1.91814148e-01,  3.45679459e-01],
       [-7.41741217e-01, -4.21874699e-02,  6.05008649e-02,
        -1.57208116e-01, -5.77611392e-01,  1.09163092e-01],
       [-3.11912064e-01, -6.35243572e-01,  1.02405189e-02,
         1.71363672e-01,  1.27261287e-02, -1.69212696e-02],
       [-1.69669089e+00, -3.44363283e-01, -2.53407507e-01,
         1.25643278e-02, -5.26606002e-02, -2.71661600e-02],
       [-1.24682093e+00, -4.90983662e-01, -3.20938196e-02,
        -2.05643780e-01,  2.93505340e-01, -7.80119838e-02],
       [-3.38749784e-01, -7.85168589e-01, -4.93584829e-01,
         3.98563085e-02, -5.44978619e-01, -1.55371653e-01],
       [-2.37415013e+00, -3.86538883e-01,  1.16098392e-01,
        -4.53365617e-01, -2.30108300e-01,  2.66983932e-01],
       [-1.40327739e+00,  2.11951503e+00, -4.42827141e-01,
        -6.32543273e-01,  2.30053526e-01, -2.35615124e-01],
       [-1.72610332e+00,  8.82371161e-02,  1.70403663e-01,
         2.60901913e-01,  2.33318380e-01,  2.38968449e-01],
       [-4.50857480e-01, -1.11329480e-02, -1.75746046e-01,
         2.36165626e-01,  2.63250697e-01, -3.14843521e-01],
       [ 4.02381405e-02, -1.00920438e+00, -4.96517167e-01,
         2.29298758e-01,  4.48031921e-01,  4.93921533e-03],
       [ 3.23373034e+00, -3.74580487e-01, -4.95372816e-01,
```

In [37]:
```
pca=PCA(n_components=5)
pca_values=pca.fit_transform(uni_normal)
pca_values
```

Out[37]:
```
array([[-1.00987445, -1.06430962,  0.08106631,  0.05695064, -0.12875425],
       [-2.82223781,  2.25904458,  0.83682883,  0.14384464, -0.12596191],
       [ 1.11246577,  1.63120889, -0.26678684,  1.07507502, -0.19181415],
       [-0.74174122, -0.04218747,  0.06050086, -0.15720812, -0.57761139],
       [-0.31191206, -0.63524357,  0.01024052,  0.17136367,  0.01272613],
       [-1.69669089, -0.34436328, -0.25340751,  0.01256433, -0.0526606 ],
       [-1.24682093, -0.49098366, -0.03209382, -0.20564378,  0.29350534],
       [-0.33874978, -0.78516859, -0.49358483,  0.03985631, -0.54497862],
       [-2.37415013, -0.38653888,  0.11609839, -0.45336562, -0.2301083 ],
       [-1.40327739,  2.11951503, -0.44282714, -0.63254327,  0.23005353],
       [-1.72610332,  0.08823712,  0.17040366,  0.26090191,  0.23331838],
       [-0.45085748, -0.01113295, -0.17574605,  0.23616563,  0.2632507 ],
       [ 0.04023814, -1.00920438, -0.49651717,  0.22929876,  0.44803192],
       [ 3.23373034, -0.37458049, -0.49537282, -0.52123771, -0.63929481],
       [-2.23626502, -0.37179329, -0.39899365,  0.40696648, -0.41676068],
       [ 5.17299212,  0.77991535, -0.38591233, -0.23221171,  0.17928698],
       [-1.69964377, -0.30559745,  0.31850785, -0.29746268, -0.16342468],
       [ 4.578146  , -0.34759136,  1.49964176, -0.45425171, -0.19114197],
       [ 0.82260312, -0.69890615,  1.42781145,  0.7607788 ,  0.18426033],
       [-0.09776213,  0.65044645,  0.10050844, -0.50009719,  0.48721782],
       [ 1.9631826 , -0.22476756, -0.25588143, -0.0484741 ,  0.82274566],
       [-0.54228894, -0.07958884, -0.30539348,  0.13169876,  0.05273991],
       [ 0.53222092, -1.0171672 , -0.42371636,  0.16953571,  0.35781321],
       [ 3.54869664,  0.77846167, -0.44936332,  0.32367862, -0.35833256],
       [-2.30590032, -0.11770432,  0.25398866, -0.51618337,  0.05589401]])
```

In [38]:
```
pca=PCA(n_components=6)
pca_values=pca.fit_transform(uni_normal)
pca_values
```

Out[38]:
```
array([[-1.00987445e+00, -1.06430962e+00,  8.10663051e-02,
         5.69506350e-02, -1.28754245e-01, -3.46496377e-02],
       [-2.82223781e+00,  2.25904458e+00,  8.36828830e-01,
         1.43844644e-01, -1.25961913e-01, -1.80703168e-01],
       [ 1.11246577e+00,  1.63120889e+00, -2.66786839e-01,
         1.07507502e+00, -1.91814148e-01,  3.45679459e-01],
       [-7.41741217e-01, -4.21874699e-02,  6.05008649e-02,
        -1.57208116e-01, -5.77611392e-01,  1.09163092e-01],
       [-3.11912064e-01, -6.35243572e-01,  1.02405189e-02,
         1.71363672e-01,  1.27261287e-02, -1.69212696e-02],
       [-1.69669089e+00, -3.44363283e-01, -2.53407507e-01,
         1.25643278e-02, -5.26606002e-02, -2.71661600e-02],
       [-1.24682093e+00, -4.90983662e-01, -3.20938196e-02,
        -2.05643780e-01,  2.93505340e-01, -7.80119838e-02],
       [-3.38749784e-01, -7.85168589e-01, -4.93584829e-01,
         3.98563085e-02, -5.44978619e-01, -1.55371653e-01],
       [-2.37415013e+00, -3.86538883e-01,  1.16098392e-01,
        -4.53365617e-01, -2.30108300e-01,  2.66983932e-01],
       [-1.40327739e+00,  2.11951503e+00, -4.42827141e-01,
        -6.32543273e-01,  2.30053526e-01, -2.35615124e-01],
       [-1.72610332e+00,  8.82371161e-02,  1.70403663e-01,
         2.60901913e-01,  2.33318380e-01,  2.38968449e-01],
       [-4.50857480e-01, -1.11329480e-02, -1.75746046e-01,
         2.36165626e-01,  2.63250697e-01, -3.14843521e-01],
       [ 4.02381405e-02, -1.00920438e+00, -4.96517167e-01,
         2.29298758e-01,  4.48031921e-01,  4.93921533e-03],
       [ 3.23373034e+00, -3.74580487e-01, -4.95372816e-01,
        -5.21237711e-01, -6.39294809e-01, -9.00477852e-02],
```

In [39]:
```
var = pca.explained_variance_ratio_
var
```

Out[39]:
```
array([0.76868084, 0.13113602, 0.04776031, 0.02729668, 0.0207177 ,
       0.00440844])
```

In [41]:
```
import numpy as np
var1= np.cumsum(np.round(var,decimals=4)*100)
var1
```

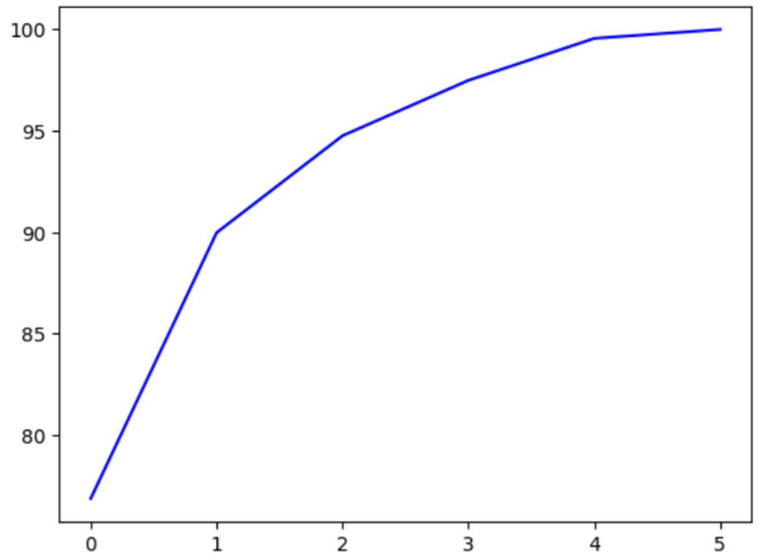
Out[41]:
```
array([ 76.87,  89.98,  94.76,  97.49,  99.56, 100.  ])
```

In [42]:
```
pca.components_
```

Out[42]:
```
array([[-0.45774863, -0.42714437,  0.42430805,  0.39064831, -0.36252316,
        -0.37940403],
       [ 0.03968044, -0.19993153,  0.32089297, -0.43256441,  0.6344864 ,
        -0.51555367],
       [ 0.1870388 ,  0.49780855, -0.15627899,  0.60608085,  0.20474114,
        -0.53247261],
       [ 0.13124033,  0.37489567,  0.0612872 , -0.50739095, -0.62340055,
        -0.43863341],
       [ 0.02064583,  0.4820162 ,  0.8010936 ,  0.07682369,  0.07254775,
         0.33810965],
       [ 0.8580547 , -0.39607492,  0.21693361,  0.1720479 , -0.17376309,
        -0.00353754]])
```

In [45]:
```
plt.plot(var1,color="blue")
```

Out[45]: [<matplotlib.lines.Line2D at 0x26815e71160>]

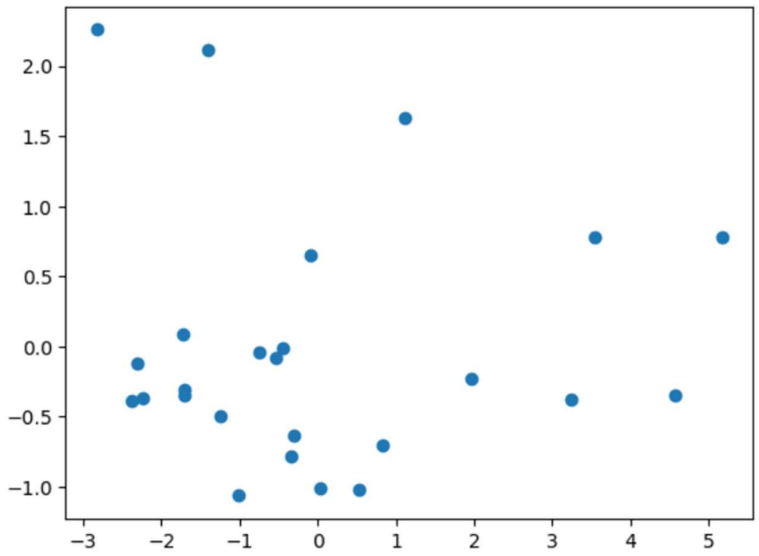


In [46]:
```
pca_values[:,0:1]
```

Out[46]:
```
array([[-1.00987445],
       [-2.82223781],
       [ 1.11246577],
       [-0.74174122],
       [-0.31191206],
       [-1.69669089],
       [-1.24682093],
       [-0.33874978],
       [-2.37415013],
       [-1.40327739],
       [-1.72610332],
       [-0.45085748],
       [ 0.04023814],
       [ 3.23373034],
       [-2.23626502],
       [ 5.17299212],
       [-1.69964377],
       [ 4.578146  ],
       [ 0.82260312],
       [-0.09776213],
       [ 1.9631826 ],
       [-0.54228894],
       [ 0.53222092],
       [ 3.54869664],
       [-2.30590032]])
```

In [49]:
```
x=pca_values[:,0:1]
y=pca_values[:,1:2]
#z=pca_values[:,2:3]
plt.scatter(x,y)
```

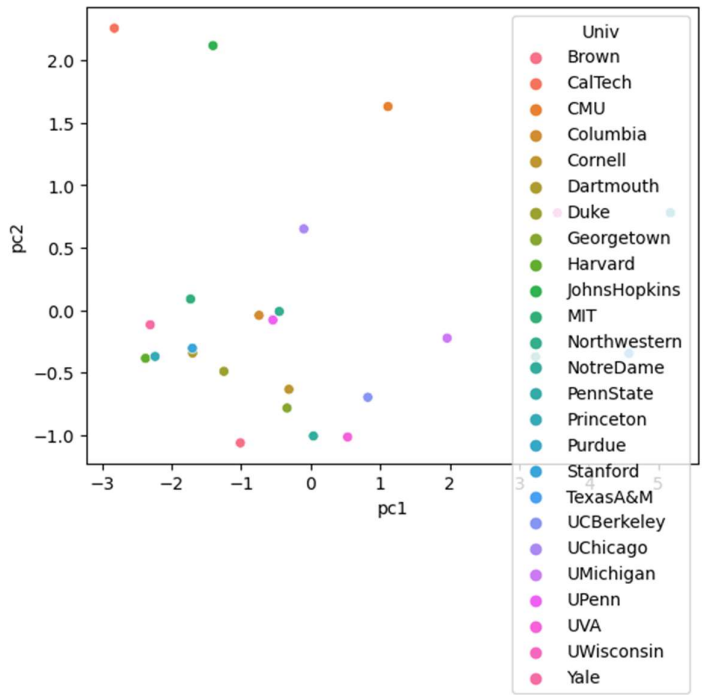
Out[49]: <matplotlib.collections.PathCollection at 0x2681563ab50>



In [51]:
```
finalDf=pd.concat([pd.DataFrame(pca_values[:,0:2],columns=['pc1','pc2']),uni[['Univ']]],axis=1)
```

In [52]:
```
sns.scatterplot(data=finalDf,x='pc1',y='pc2',hue='Univ')
```

Out[52]: <AxesSubplot:xlabel='pc1', ylabel='pc2'>



CONCLUSION:

In conclusion, this experiment is using association rule mining, DBSCAN, and PCA provided a thorough analysis of your dataset. Association rule mining revealed key relationships and patterns, while DBSCAN identified clusters and outliers, offering insights into data groupings and potential anomalies. PCA effectively reduced dimensionality, simplifying the dataset and retaining important information.