Output –

weather.numeric_missing

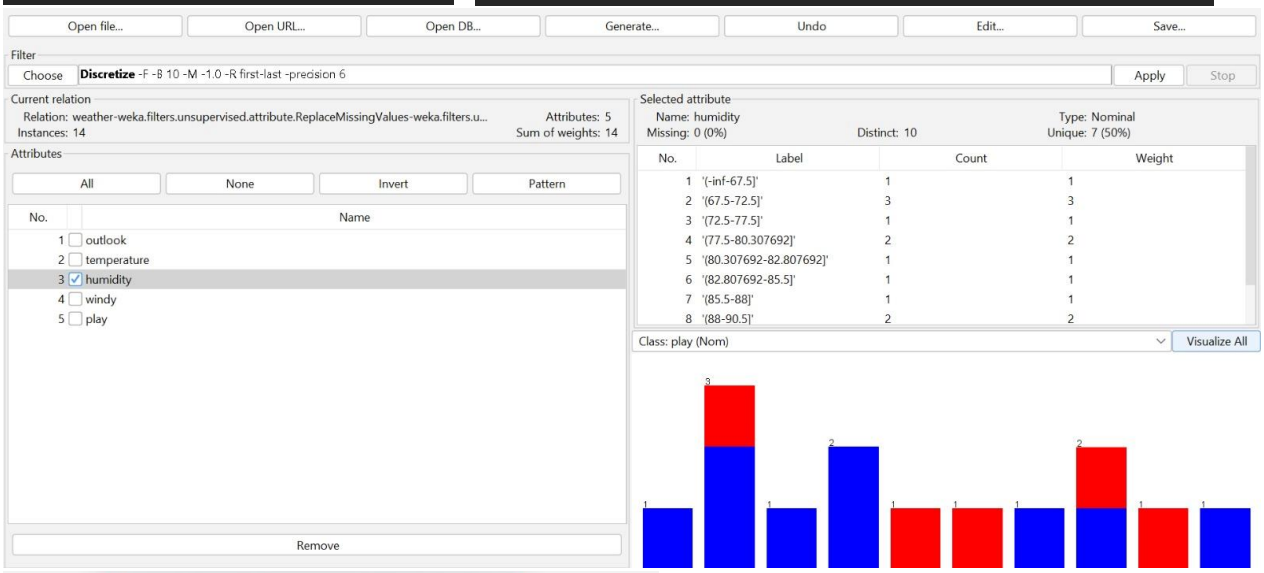
File Edit View

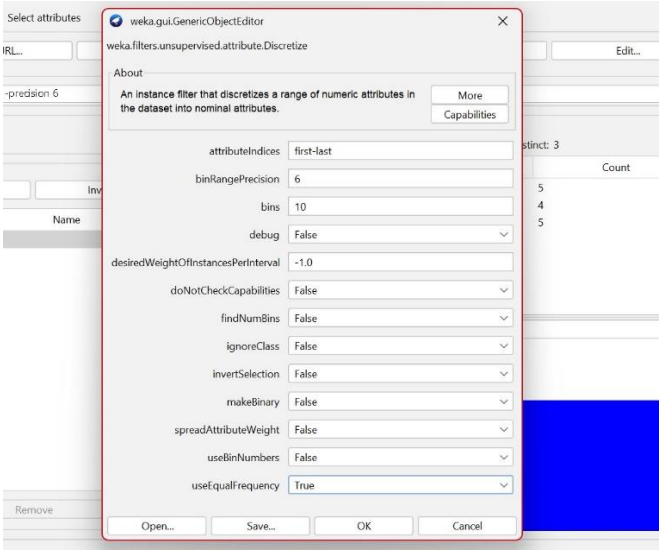
```
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature numeric
@attribute humidity numeric
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,?,85,FALSE,no
sunny,80,90,TRUE,no
overcast,?,86,FALSE,yes
rainy,70,96,FALSE,yes
rainy,68,80,FALSE,yes
rainy,65,70,TRUE,no
overcast,64,65,TRUE,yes
sunny,72,?,FALSE,no
sunny,69,70,FALSE,yes
rainy,75,80,FALSE,yes
sunny,?,70,TRUE,yes
overcast,72,90,TRUE,yes
overcast,81,75,FALSE,yes
rainy,71,91,TRUE,no
```

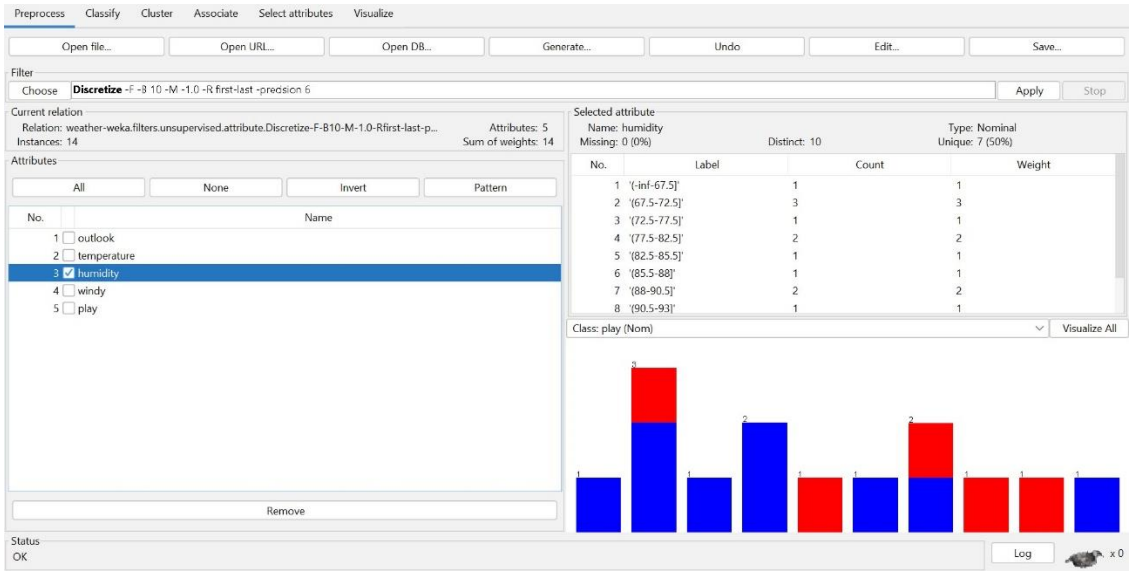
weather.numeric_replaced | weather.numeric_missing

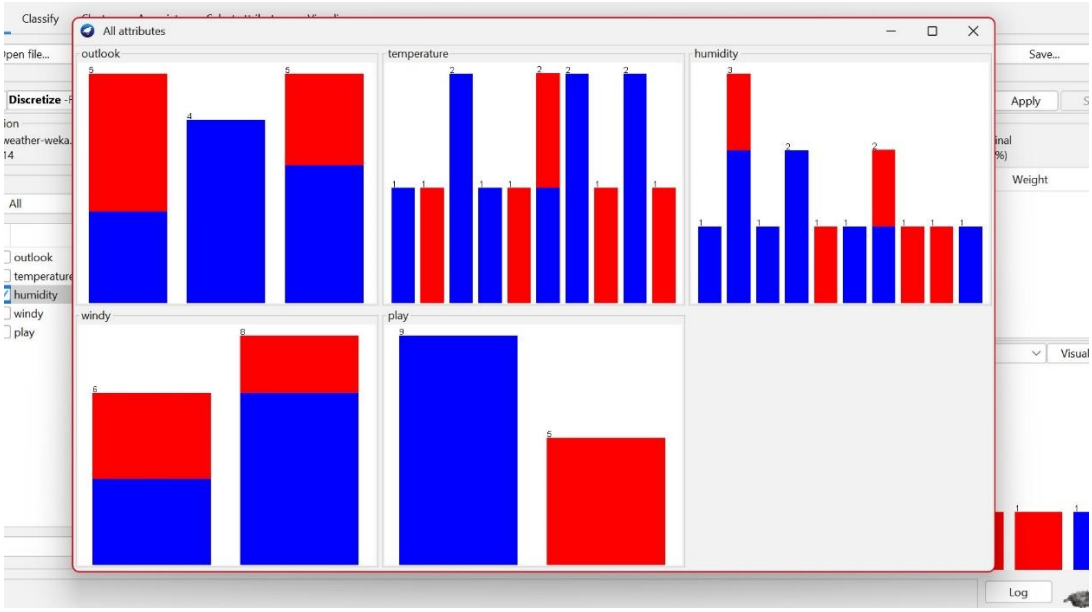
File Edit View

```
@relation weather-weka.filters.unsupervised.attribute.ReplaceMissingValues

@attribute outlook {sunny,overcast,rainy}
@attribute temperature numeric
@attribute humidity numeric
@attribute windy {TRUE,FALSE}
@attribute play {yes,no}

@data
sunny,71.545455,85,FALSE,no
sunny,80,90,TRUE,no
overcast,71.545455,86,FALSE,yes
rainy,70,96,FALSE,yes
rainy,68,80,FALSE,yes
rainy,65,70,TRUE,no
overcast,64,65,TRUE,yes
sunny,72,80.615385,FALSE,no
sunny,69,70,FALSE,yes
rainy,75,80,FALSE,yes
sunny,71.545455,70,TRUE,yes
overcast,72,90,TRUE,yes
overcast,81,75,FALSE,yes
rainy,71,91,TRUE,no
```





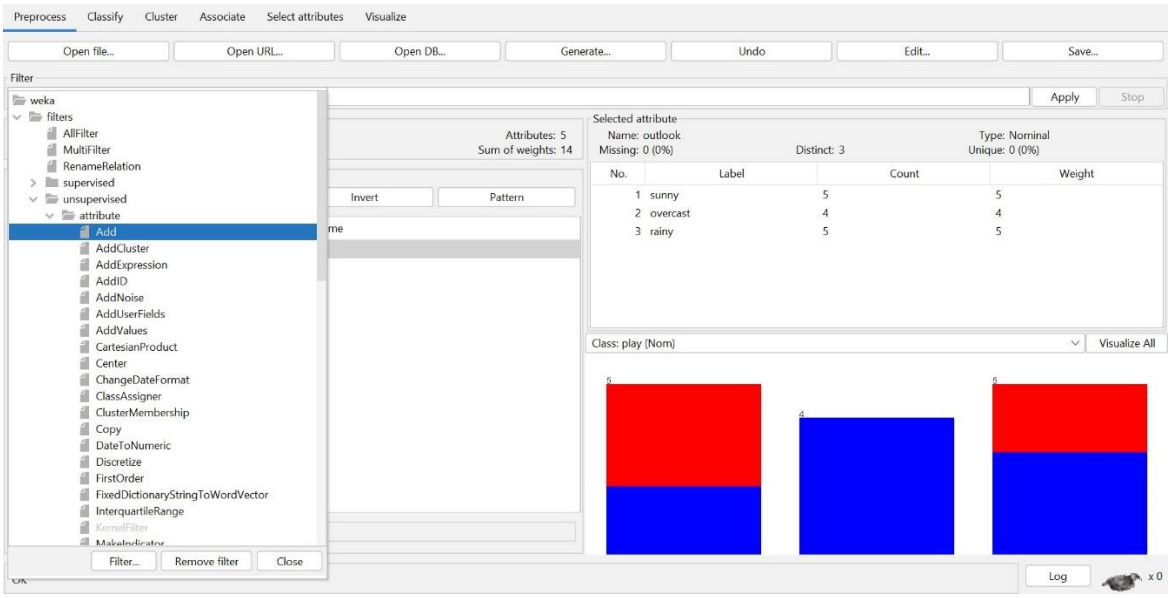


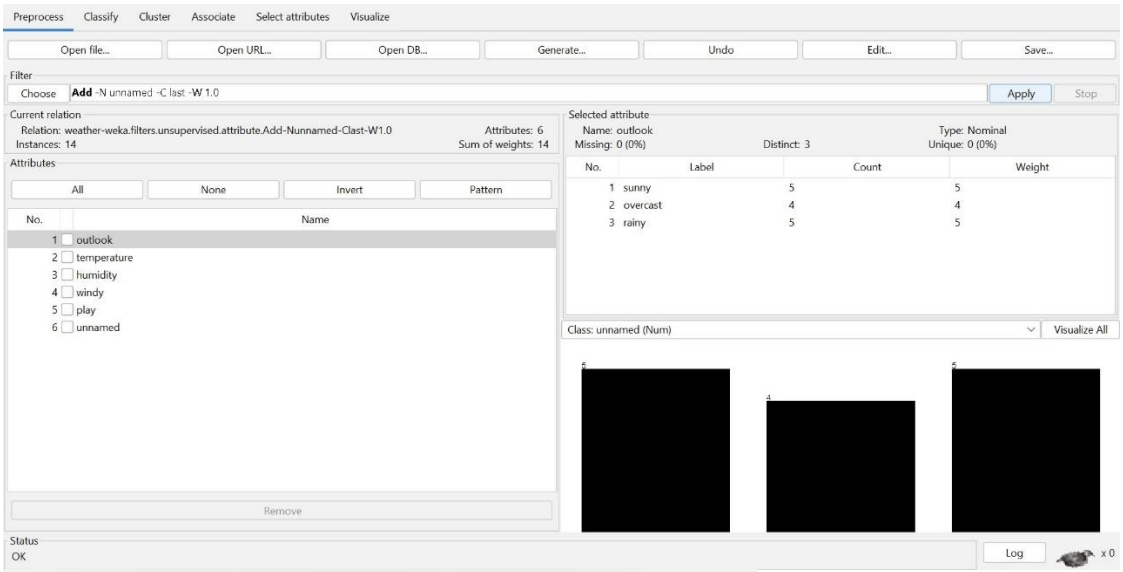


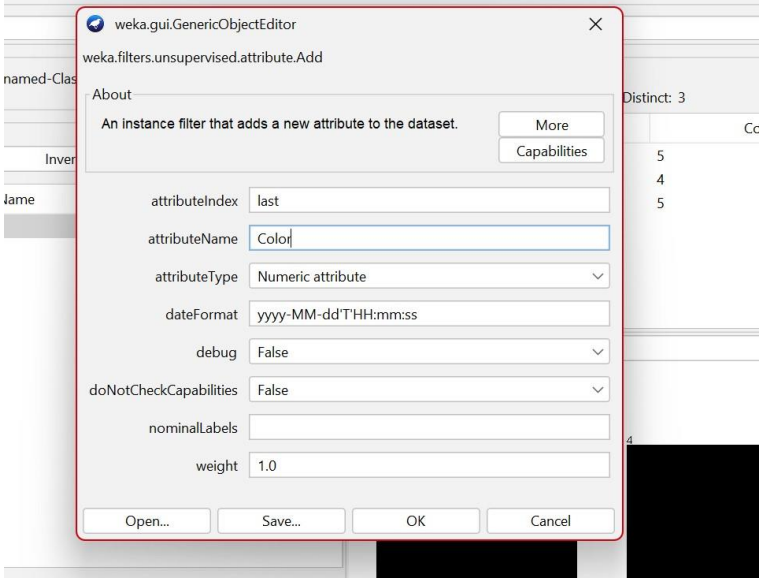
weather.numerical_desc

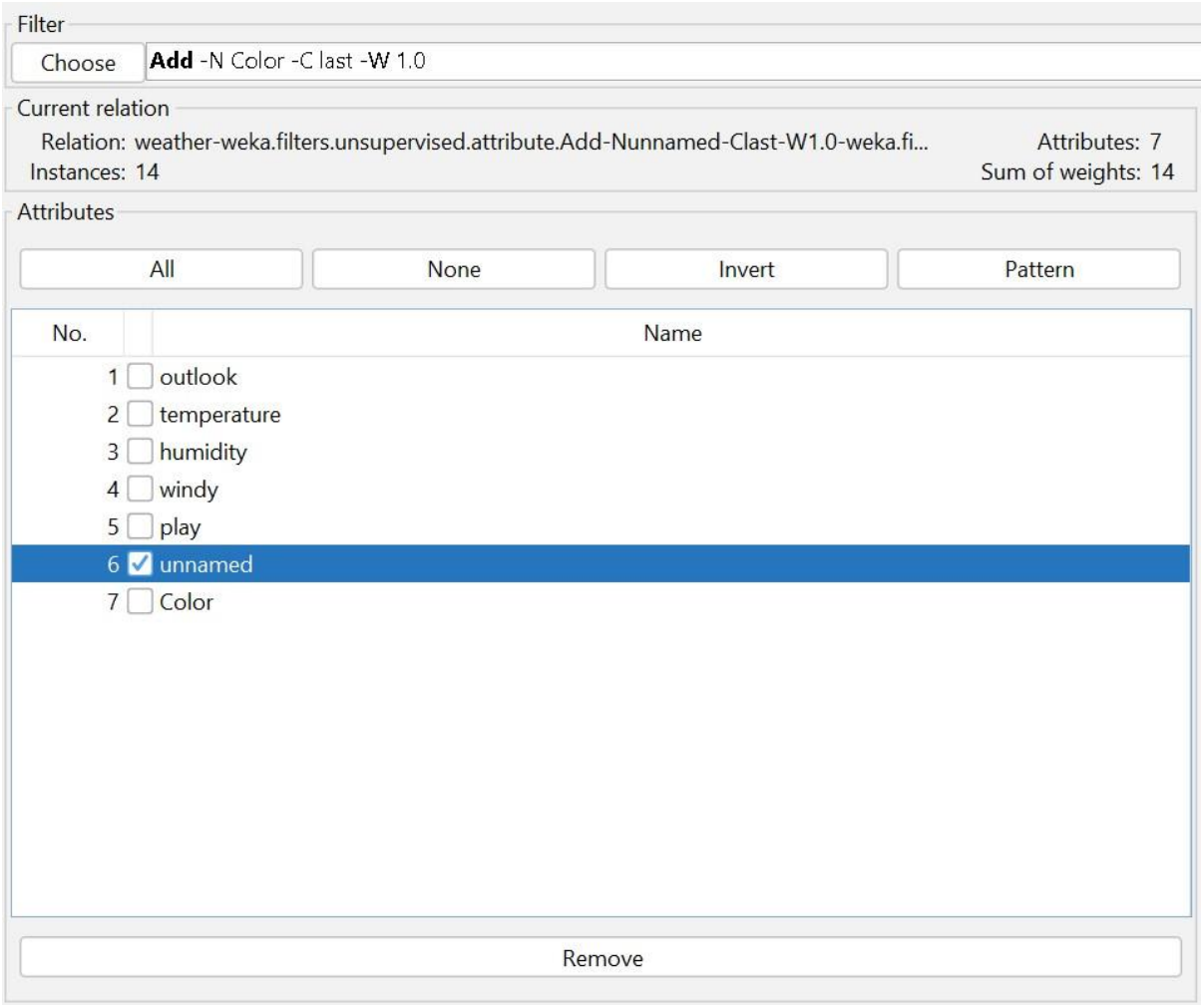
File Edit View

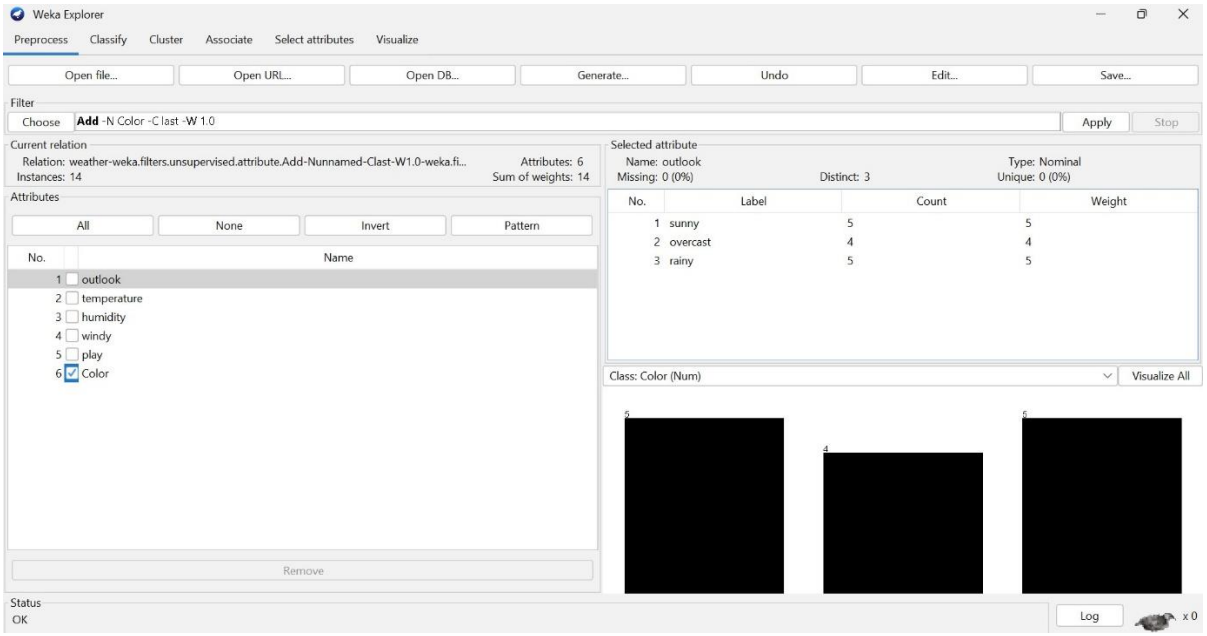
```
@relation weather-weka.filters.unsupervised.attribute.Discretize-F-B10-M-1.0-Rfirst-last-precision6

@attribute outlook {sunny,overcast,rainy}
@attribute temperature {'\'(-
inf-64.5]\'','\'(64.5-66.5]\'','\'(66.5-69.5]\'','\'(69.5-70.5]\'','\'(70.5-71.5]\'','\'(71.5-73.5]\'','\'(73.5-77.5]\'','\'(77.5-80.5]\
'','\'(80.5-84]\'','\'(84-inf)\''}
@attribute humidity {'\'(-
inf-67.5]\'','\'(67.5-72.5]\'','\'(72.5-77.5]\'','\'(77.5-82.5]\'','\'(82.5-85.5]\'','\'(85.5-88]\'','\'(88-90.5]\'','\'(90.5-93]\'','\'
(93-95.5]\'','\'(95.5-inf)\''}
@attribute windy {TRUE,FALSE}
@attribute play {yes,no}

@data
sunny,'\'(84-inf)\'','\'(82.5-85.5]\'',FALSE,no
sunny,'\'(77.5-80.5]\'','\'(88-90.5]\'',TRUE,no
overcast,'\'(80.5-84]\'','\'(85.5-88]\'',FALSE,yes
rainy,'\'(69.5-70.5]\'','\'(95.5-inf)\'',FALSE,yes
rainy,'\'(66.5-69.5]\'','\'(77.5-82.5]\'',FALSE,yes
rainy,'\'(64.5-66.5]\'','\'(67.5-72.5]\'',TRUE,no
overcast,'\'(-inf-64.5]\'','\'(-inf-67.5]\'',TRUE,yes
sunny,'\'(71.5-73.5]\'','\'(93-95.5]\'',FALSE,no
sunny,'\'(66.5-69.5]\'','\'(67.5-72.5]\'',FALSE,yes
rainy,'\'(73.5-77.5]\'','\'(77.5-82.5]\'',FALSE,yes
sunny,'\'(73.5-77.5]\'','\'(67.5-72.5]\'',TRUE,yes
overcast,'\'(71.5-73.5]\'','\'(88-90.5]\'',TRUE,yes
overcast,'\'(80.5-84]\'','\'(72.5-77.5]\'',FALSE,yes
rainy,'\'(70.5-71.5]\'','\'(90.5-93]\'',TRUE,no
```


Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Apply
Stop
weka
filters
AllFilter
MultiFilter
RenameRelation
supervised
unsupervised
attribute
Add
AddCluster
AddExpression
AddID
AddNoise
AddUserFields
AddValues
CartesianProduct
Center
ChangeDateFormat
ClassAssigner
ClusterMembership
Copy
DateToNumeric
Discretize
FirstOrder
FixedDictionaryStringToWordVector
InterquartileRange
Attributes: 5
Sum of weights: 14
Invert
Pattern
Selected attribute
Name: outlook
Missing: 0 (0%)
Distinct: 3
Type: Nominal
Unique: 0 (0%)
No.
Label
Count
Weight
1 sunny 5 5
2 overcast 4 4
3 rainy 5 5
Class: play (Nom)
Visualize All
Filter...
Remove filter
Close
Log
x 0


Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
Add -N unnamed -C last -W 1.0
Apply
Stop
Current relation
Relation: weather-weka.filters.unsupervised.attribute.Add-Nunnamed-Clast-W1.0
Instances: 14
Attributes: 6
Sum of weights: 14
Attributes
All
None
Invert
Pattern
No.
Name
1 outlook
2 temperature
3 humidity
4 windy
5 play
6 unnamed
Remove
Selected attribute
Name: outlook
Missing: 0 (0%)
Distinct: 3
Type: Nominal
Unique: 0 (0%)
No.
Label
Count
Weight
1 sunny 5 5
2 overcast 4 4
3 rainy 5 5
Class: unnamed (Num)
Visualize All
Status
OK
Log
x 0


weka.gui.GenericObjectEditor
weka.filters.unsupervised.attribute.Add
About
An instance filter that adds a new attribute to the dataset.
More
Capabilities
attributeIndex
last
attributeName
Color
attributeType
Numeric attribute
dateFormat
yyyy-MM-dd'T'HH:mm:ss
debug
False
doNotCheckCapabilities
False
nominalLabels
weight
1.0
Open...
Save...
OK
Cancel


Filter
Choose
Add -N Color -C last -W 1.0
Current relation
Relation: weather-weka.filters.unsupervised.attribute.Add-Nunnamed-Clast-W1.0-weka.fi...
Attributes: 7
Instances: 14
Sum of weights: 14
Attributes
All
None
Invert
Pattern
No.
Name
1 outlook
2 temperature
3 humidity
4 windy
5 play
6 unnamed
7 Color
Remove


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
Add -N Color -C last -W 1.0
Apply
Stop
Current relation
Relation: weather-weka.filters.unsupervised.attribute.Add-Nunnamed-Clast-W1.0-weka.fi...
Attributes: 6
Instances: 14
Sum of weights: 14
Attributes
All
None
Invert
Pattern
No.
Name
1 outlook
2 temperature
3 humidity
4 windy
5 play
6 Color
Remove
Selected attribute
Name: outlook
Type: Nominal
Missing: 0 (0%)
Distinct: 3
Unique: 0 (0%)
No.
Label
Count
Weight
1 sunny 5 5
2 overcast 4 4
3 rainy 5 5
Class: Color (Num)
Visualize All
5
4
5
Status
OK
Log
x 0

Program –

```
import pandas as pd
import numpy as np
from mlxtend.frequent_patterns import apriori, association_rules
df = pd.read_csv('GroceryStoreDataSet.csv', names = ['products'], sep = ',')
df.head()
df.shape
data = list(df["products"].apply(lambda x:x.split(",") ))
data
from mlxtend.preprocessing import TransactionEncoder
a = TransactionEncoder()
a_data = a.fit(data).transform(data)
df = pd.DataFrame(a_data,columns=a.columns_)
df = df.replace(False,0)
df
df = apriori(df, min_support = 0.2, use_colnames = True, verbose = 1)
df
df_ar = association_rules(df, metric = "confidence", min_threshold = 0.6)
df_ar
```

Output –

```
Processing 42 combinations | Sampling itemset size 3
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` aut
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result
  warnings.warn(
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
0	(MILK)	(BREAD)	0.25	0.65	0.2	0.800000	1.230769	0.0375	1.75	0.250000
1	(SUGER)	(BREAD)	0.30	0.65	0.2	0.666667	1.025641	0.0050	1.05	0.035714
2	(CORNFLAKES)	(COFFEE)	0.30	0.40	0.2	0.666667	1.666667	0.0800	1.80	0.571429
3	(SUGER)	(COFFEE)	0.30	0.40	0.2	0.666667	1.666667	0.0800	1.80	0.571429
4	(MAGGI)	(TEA)	0.25	0.35	0.2	0.800000	2.285714	0.1125	3.25	0.750000

Program –

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
           ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
           ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
           ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
df
from mlxtend.frequent_patterns import fpgrowth

fpgrowth(df, min_support=0.6)
fpgrowth(df, min_support=0.6, use_colnames=True)
```

Output –

```
/usr/local/lib/python3.10/dist-packages/ipyker
  and should_run_async(code)
```

	support	itemsets
0	1.0	(Kidney Beans)
1	0.8	(Eggs)
2	0.6	(Yogurt)
3	0.6	(Onion)
4	0.6	(Milk)
5	0.8	(Eggs, Kidney Beans)
6	0.6	(Yogurt, Kidney Beans)
7	0.6	(Eggs, Onion)
8	0.6	(Onion, Kidney Beans)
9	0.6	(Eggs, Onion, Kidney Beans)
10	0.6	(Milk, Kidney Beans)

Program –

```
!pip install mmh3
!pip install bitarray
!pip install bloomfilter

import math
import mmh3
from bitarray import bitarray

class BloomFilter(object):

    def __init__(self, items_count, fp_prob):
        self.fp_prob = fp_prob
        self.size = self.get_size(items_count, fp_prob)
        self.hash_count = self.get_hash_count(self.size, items_count)
        self.bit_array = bitarray(self.size)
        self.bit_array.setall(0)

    def add(self, item):

        digests = []
        for i in range(self.hash_count):
            digest = mmh3.hash(item, i) % self.size
            digests.append(digest)

            self.bit_array[digest] = True

    def check(self, item):
        for i in range(self.hash_count):
            digest = mmh3.hash(item, i) % self.size
            if self.bit_array[digest] == False:
                return False
        return True

    @classmethod
    def get_size(self, n, p):
        m = -(n * math.log(p))/(math.log(2)**2)
        return int(m)

    @classmethod
    def get_hash_count(self, m, n):
```

```
                k = (m/n) * math.log(2)
                return int(k)

from random import shuffle

n = 20
p = 0.05

bloomf = BloomFilter(n,p)
print("Size of bit array:{}".format(bloomf.size))
print("False positive Probability:{}".format(bloomf.fp_prob))
print("Number of hash functions:{}".format(bloomf.hash_count))

word_present = ['abound','abounds','abundance','abundant','accessible',
'bloom','blossom','bolster','bonny','bonus','bonuses',
'coherent','cohesive','colorful','comely','comfort',
'gems','generosity','generous','generously','genial']

word_absent = ['bluff','cheater','hate','war','humanity',
'racism','hurt','nuke','gloomy','facebook', 'geeksforgeeks','twitter']

for item in word_present:
        bloomf.add(item)

shuffle(word_present)
shuffle(word_absent)

test_words = word_present[:10] + word_absent
shuffle(test_words)
for word in test_words:
        if bloomf.check(word):
                if word in word_absent:
                        print("'{}' is a false positive!".format(word))
                else:
                        print("'{}' is probably present!".format(word))
        else:
                print("'{}' is definitely not present!".format(word))
```

Output –

```
Size of bit array:124
False positive Probability:0.05
Number of hash functions:4
'hate' is definitely not present!
'facebook' is definitely not present!
'bloom' is probably present!
'abounds' is probably present!
'cheater' is definitely not present!
'nuke' is definitely not present!
'war' is definitely not present!
'comely' is probably present!
'generously' is probably present!
'geeksforgeeks' is definitely not present!
'bluff' is definitely not present!
'gloomy' is definitely not present!
'racism' is definitely not present!
'generosity' is probably present!
'bonuses' is probably present!
'blossom' is probably present!
'humanity' is a false positive!
'twitter' is a false positive!
'hurt' is definitely not present!
'bonny' is probably present!
'colorful' is probably present!
'abundant' is probably present!
```

Program –

```
import numpy
def PageRank(A, d = 0.85, eps = 0.0005, maxIterations = 1000,
             verbose = False):
    # find the size of the "Internet"
    N = A.shape[0]

    # initialize the old and new PageRank vectors
    vOld = numpy.ones([N])
    vNew = numpy.ones([N])/N

    # initialize a counter
    i = 0

    # compute the update matrix
    U = d * A.T + (1 - d) / N

    while numpy.linalg.norm(vOld - vNew) >= eps:
        # if the verbose flag is true, print the progress at each iteration
        if verbose:
            print('At iteration', i, 'the error is',
                  numpy.round(numpy.linalg.norm(vOld - vNew), 3),
                  'with PageRank', numpy.round(vNew, 3))

        # save the current PageRank as the old PageRank
        vOld = vNew

        # update the PageRank vector
        vNew = numpy.dot(U, vOld)

        # increment the counter
        i += 1

        # if it runs too long before converging, stop and notify the user
        if i == maxIterations:
            print('The PageRank algorithm ran for',
                  maxIterations, 'with error',
                  numpy.round(numpy.linalg.norm(vOld - vNew), 3))

            # return the PageRank vectora and the
            return vNew, i
```

```
    # return the steady state PageRank vector and iteration number
    return vNew, i

# transition probability matrix
A = numpy.array([[0, 1/4, 1/4, 1/4, 1/4],
                 [1/2, 0, 0, 1/2, 0],
                 [1/3, 0, 0, 1/3, 1/3],
                 [1, 0, 0, 0, 0],
                 [0, 0, 0, 1, 0]])

# Run the PageRank algorithm with default settings
PageRank(A, verbose = True)
```

Output –

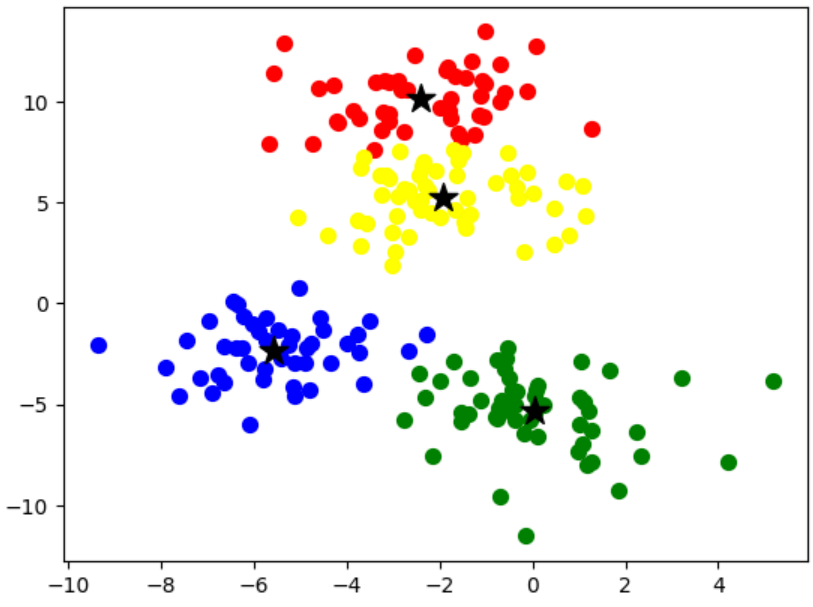
```
PageRank(A, verbose = True)

At iteration 0 the error is 1.789 with PageRank [0.2 0.2 0.2 0.2 0.2]
At iteration 1 the error is 0.303 with PageRank [0.342 0.073 0.073 0.384 0.129]
At iteration 2 the error is 0.144 with PageRank [0.408 0.103 0.103 0.264 0.123]
At iteration 3 the error is 0.092 with PageRank [0.327 0.117 0.117 0.294 0.146]
At iteration 4 the error is 0.047 with PageRank [0.363 0.099 0.099 0.306 0.133]
At iteration 5 the error is 0.019 with PageRank [0.361 0.107 0.107 0.29  0.135]
At iteration 6 the error is 0.011 with PageRank [0.352 0.107 0.107 0.297 0.137]
At iteration 7 the error is 0.007 with PageRank [0.358 0.105 0.105 0.297 0.135]
At iteration 8 the error is 0.003 with PageRank [0.357 0.106 0.106 0.295 0.136]
At iteration 9 the error is 0.001 with PageRank [0.356 0.106 0.106 0.296 0.136]
At iteration 10 the error is 0.001 with PageRank [0.357 0.106 0.106 0.296 0.136]
(array([0.3565286 , 0.10584025, 0.10584025, 0.29600666, 0.13578424]), 11)
```

Program –

```
!pip install -U scikit-learn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
dataset = make_blobs(n_samples=200, centers=4, n_features=2, cluster_std=1.6,
random_state=50)
points = dataset[0]
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=4)
kmeans.fit(points)
plt.scatter(dataset[0][:,0], dataset[0][:,1])
clusters = kmeans.cluster_centers_
print(clusters)
y_km = kmeans.fit_predict(points)
y_km
plt.scatter(points[y_km== 1,0], points[y_km == 1,1], s=50, color= 'blue')
plt.scatter(points[y_km== 0,0], points[y_km == 0,1], s=50, color= 'red')
plt.scatter(points[y_km== 2,0], points[y_km == 2,1], s=50, color= 'green')
plt.scatter(points[y_km== 3,0], points[y_km == 3,1], s=50, color= 'yellow')
plt.scatter(clusters[0][0], clusters[0][1], marker='*', s=200, color= 'black')
plt.scatter(clusters[1][0], clusters[1][1], marker='*', s=200, color= 'black')
plt.scatter(clusters[2][0], clusters[2][1], marker='*', s=200, color= 'black')
plt.scatter(clusters[3][0], clusters[3][1], marker='*', s=200, color= 'black')
plt.show()
```

Output –

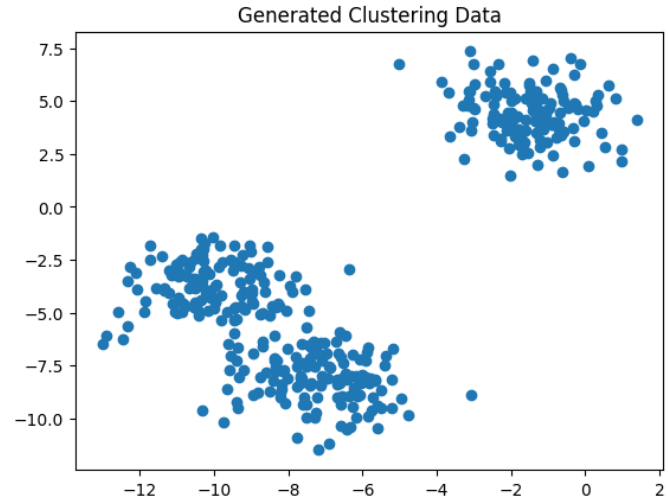
```
plt.show()

[[-2.40167949 10.17352695]
 [ 0.05161133 -5.35489826]
 [-5.56465793 -2.34988939]
 [-1.92101646  5.21673484]]
```

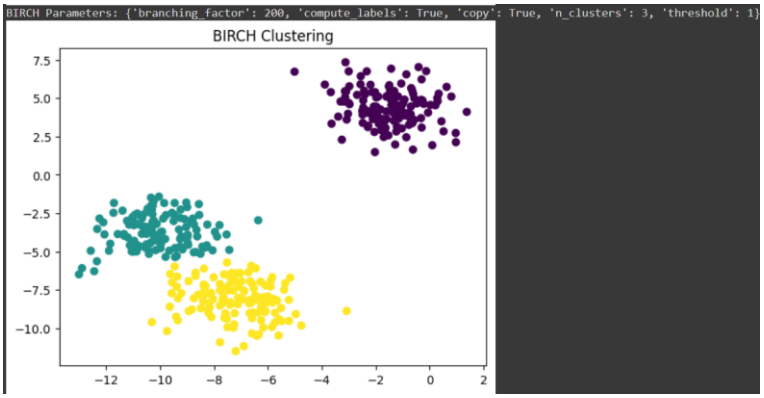


Program –

```
from sklearn.cluster import Birch
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from numpy import random

random.seed(1)
x, _ = make_blobs(n_samples=400, centers=3, cluster_std=1.2)
plt.scatter(x[:,0], x[:,1])
plt.title("Generated Clustering Data")
plt.show()

bclust = Birch(branching_factor=200, threshold=1).fit(x)
print("BIRCH Parameters:", bclust.get_params())
labels = bclust.predict(x)
plt.scatter(x[:,0], x[:,1], c=labels)
plt.title("BIRCH Clustering")
plt.show()
```

Output –

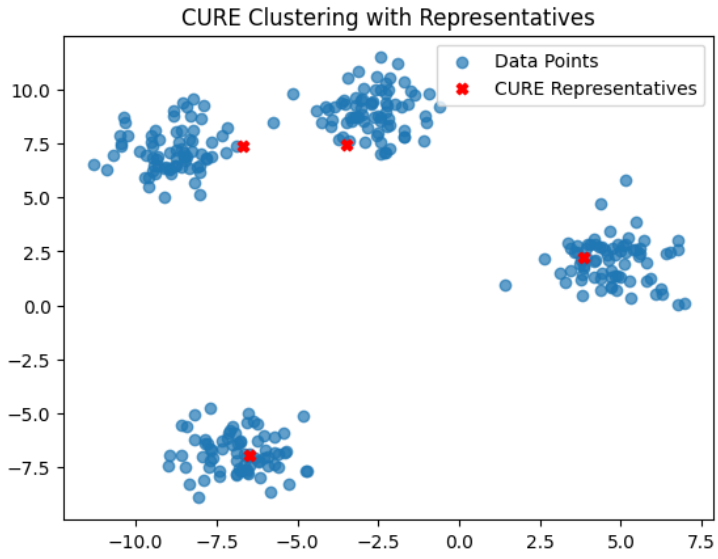




Program –

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import AgglomerativeClustering
from scipy.spatial.distance import euclidean
X, _ = make_blobs(n_samples=300, centers=4, random_state=42)
def cure_algorithm(X, num_clusters, reduction_factor=0.1):
    clustering = AgglomerativeClustering(n_clusters=num_clusters).fit(X)
    reduced_representatives = []
    for cluster_label in range(num_clusters):
        cluster_points = X[clustering.labels_ == cluster_label]
        representative_index = np.random.choice(len(cluster_points))
        representative_point = cluster_points[representative_index]
        reduced_point = representative_point - reduction_factor * np.mean(cluster_points -
representative_point, axis=0)
        reduced_representatives.append(reduced_point)
    return np.array(reduced_representatives)
num_clusters = 4
cure_representatives = cure_algorithm(X, num_clusters)
plt.scatter(X[:, 0], X[:, 1], alpha=0.7, label='Data Points')
plt.scatter(cure_representatives[:, 0], cure_representatives[:, 1], c='red', marker='X',
label='CURE Representatives')
plt.title('CURE Clustering with Representatives')
plt.legend()
plt.show()
```

Output –

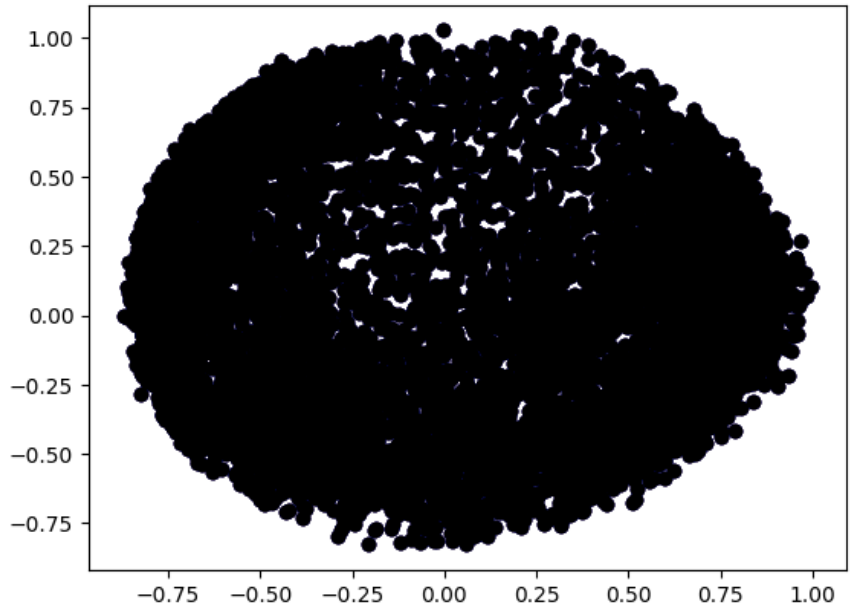


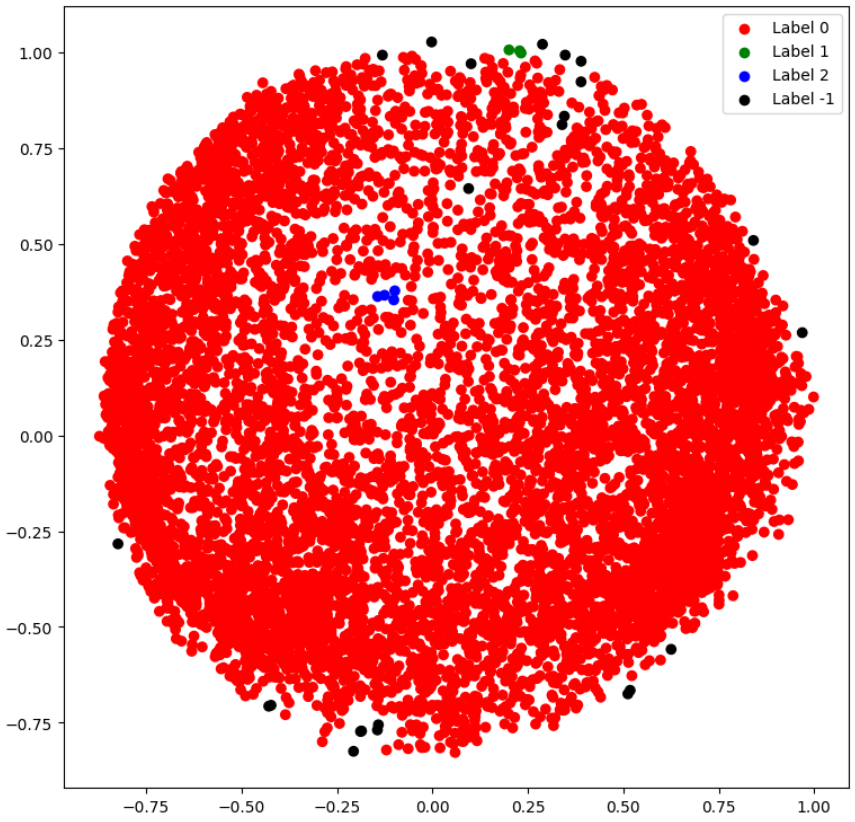
Program –

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
from sklearn.decomposition import PCA

X = pd.read_csv('CC GENERAL.csv')

X = X.drop('CUST_ID', axis = 1)
X.fillna(method ='ffill', inplace = True)

print(X.head())

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_normalized = normalize(X_scaled)
X_normalized = pd.DataFrame(X_normalized)

pca = PCA(n_components = 2)
X_principal = pca.fit_transform(X_normalized)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']
print(X_principal.head())

db_default = DBSCAN(eps = 0.0375, min_samples = 3).fit(X_principal)
labels = db_default.labels_

colours = {}
colours[0] = 'r'
colours[1] = 'g'
colours[2] = 'b'
colours[-1] = 'k'

cvec = [colours[label] for label in labels]

r = plt.scatter(X_principal['P1'], X_principal['P2'], color ='r');
g = plt.scatter(X_principal['P1'], X_principal['P2'], color ='g');
b = plt.scatter(X_principal['P1'], X_principal['P2'], color ='b');
```

```
k = plt.scatter(X_principal['P1'], X_principal['P2'], color ='k');
plt.figure(figsize =(9, 9))
plt.scatter(X_principal['P1'], X_principal['P2'], c = cvec)
plt.legend((r, g, b, k), ('Label 0', 'Label 1', 'Label 2', 'Label -1'))
plt.show()
```

Output –

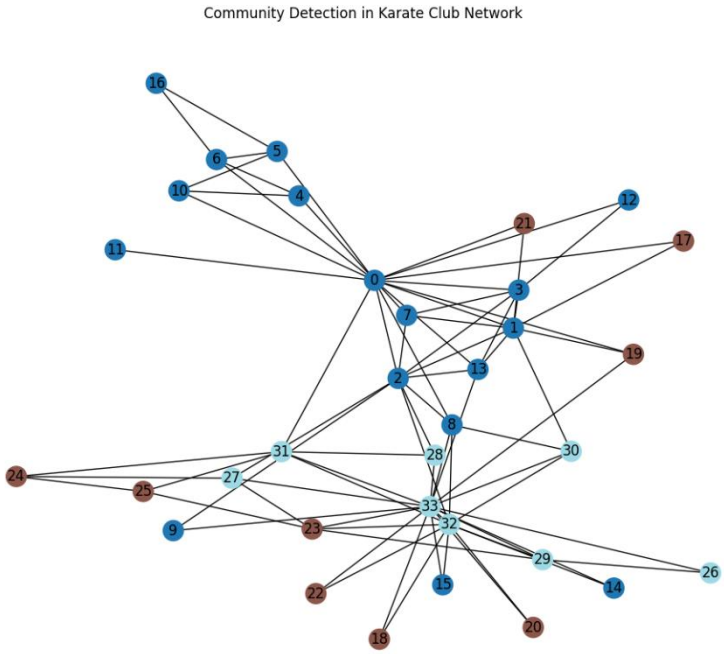




Program –

```
import networkx as nx
import matplotlib.pyplot as plt
from networkx.algorithms import community

G = nx.karate_club_graph()
communities = community.greedy_modularity_communities(G)
pos = nx.spring_layout(G)
plt.figure(figsize=(10, 8))
nx.draw(G, pos, node_color=[i for i, c in enumerate(communities) for _ in c],
cmap=plt.cm.tab20, with_labels=True)
plt.title("Community Detection in Karate Club Network")
plt.show()

print("Communities identified:")
for i, community in enumerate(communities):
    print(f"Community {i + 1}: {community}")
```

Output –




```
Communities identified:
Community 1: frozenset({8, 14, 15, 18, 20, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33})
Community 2: frozenset({1, 2, 3, 7, 9, 12, 13, 17, 21})
Community 3: frozenset({0, 16, 19, 4, 5, 6, 10, 11})
```

Program –

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

ad_data = pd.read_csv('Advertising.csv')
print(ad_data.head())
plt.figure(figsize=(10, 6))
# plt.hist(ad_data['Sales'], bins=20, color='blue', alpha=0.7, label='Sales')
plt.hist(ad_data['TV'], bins=20, color='green', alpha=0.7, label='TV')
plt.hist(ad_data['Radio'], bins=20, color='yellow', alpha=0.7, label='Radio')
plt.hist(ad_data['Newspaper'], bins=20, color='pink', alpha=0.7, label='Newspaper')
plt.xlabel('Engagement Metrics')
plt.ylabel('Frequency')
plt.title('Distribution of Ad Engagement Metrics')
plt.legend()
plt.show()
```

Output –

```
plt.show()
```

	TV	Radio	Newspaper	Sales
0	230.1	37.8	69.2	22.1
1	44.5	39.3	45.1	10.4
2	17.2	45.9	69.3	9.3
3	151.5	41.3	58.5	18.5
4	180.8	10.8	58.4	12.9