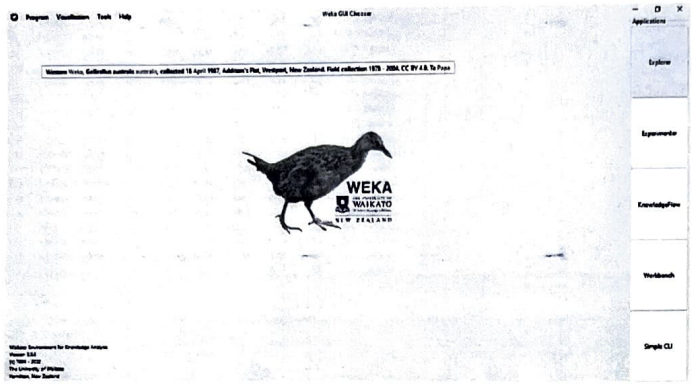
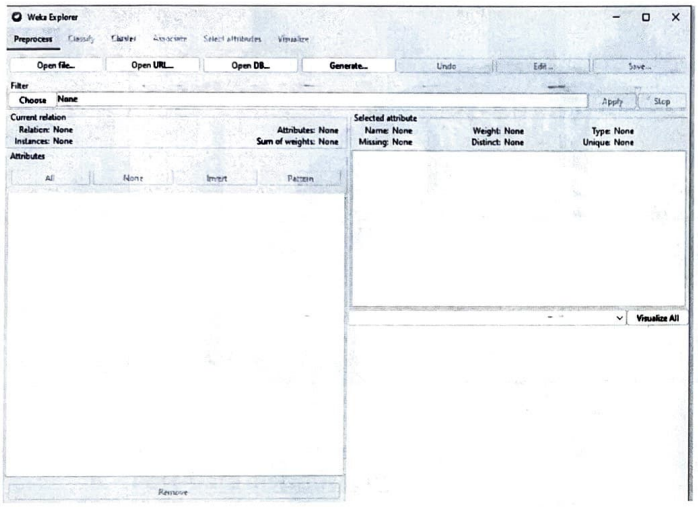
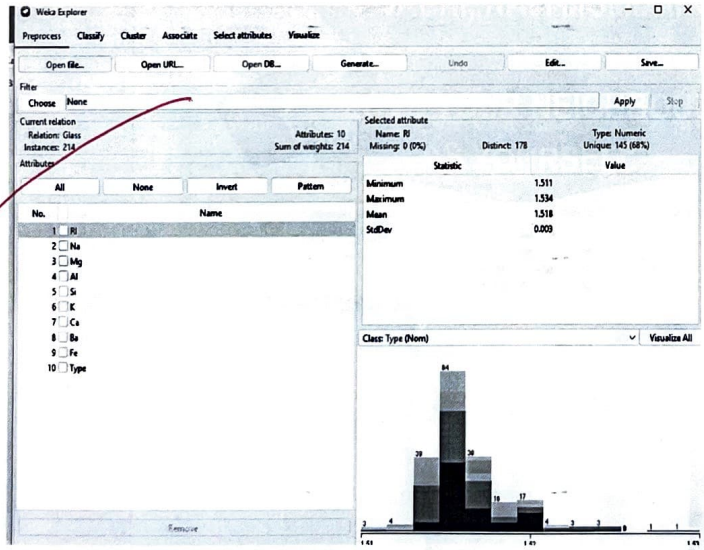
Practical 1: Data Visualization and Descretization

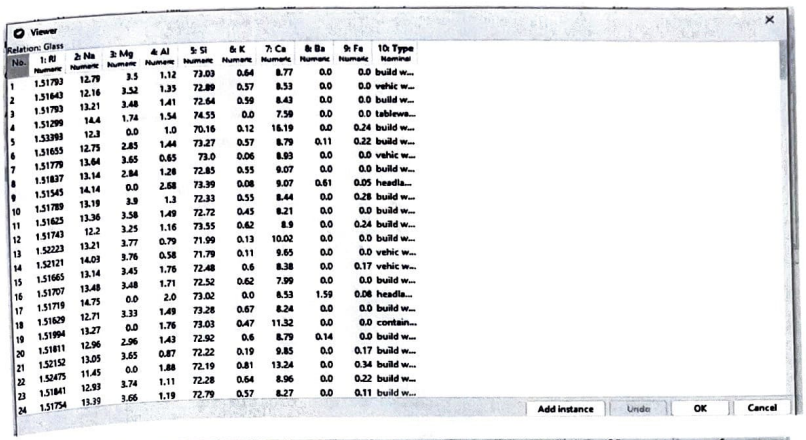
1) Data Discretization

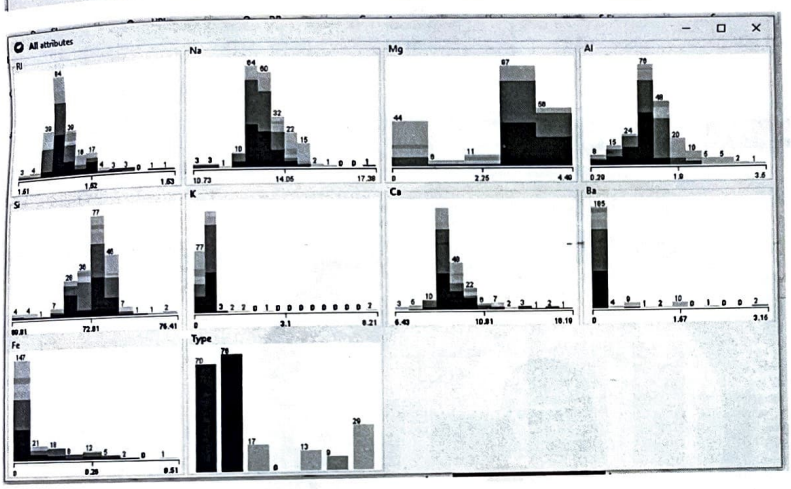
a. Pre-Data discretiztion



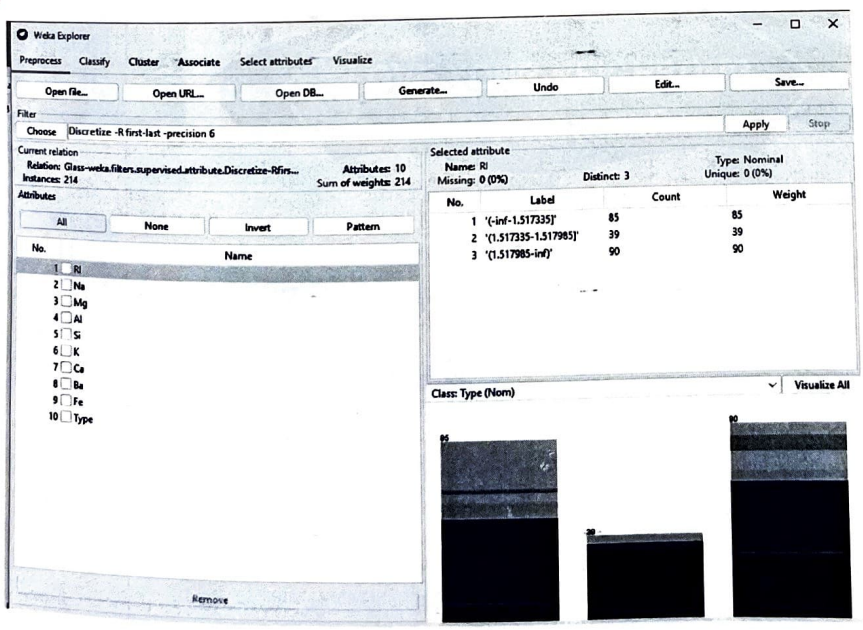


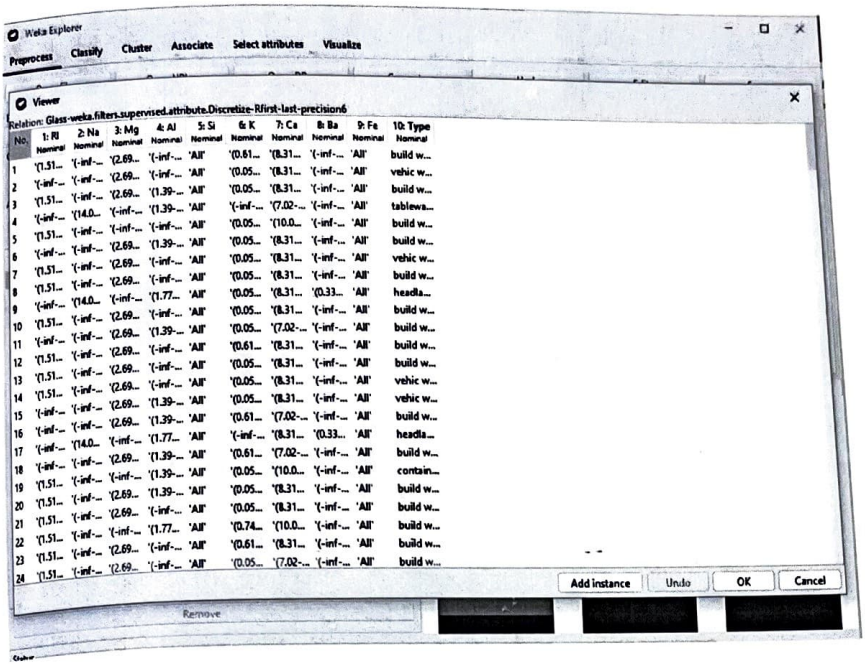


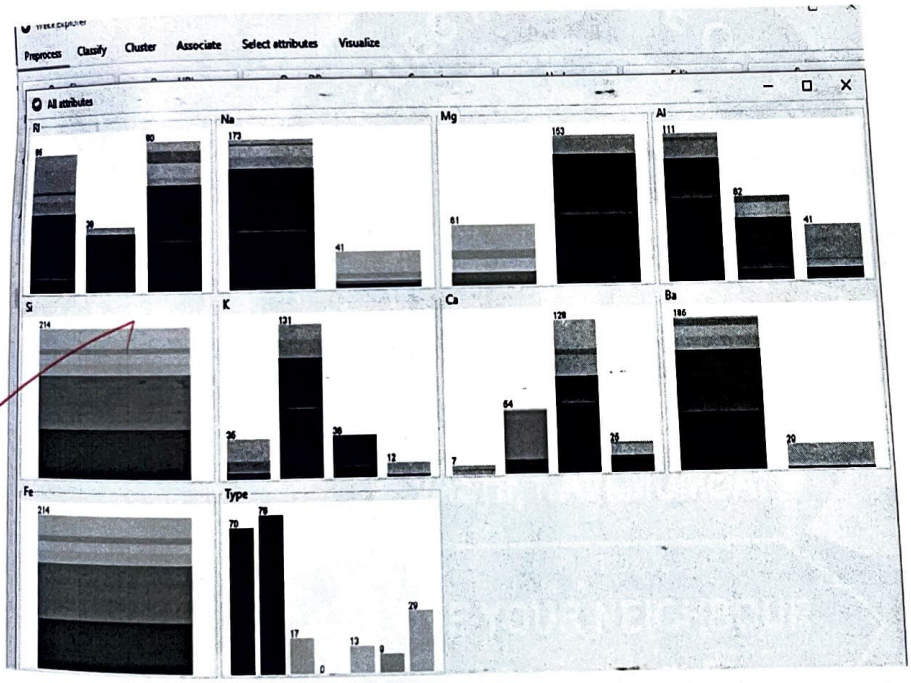




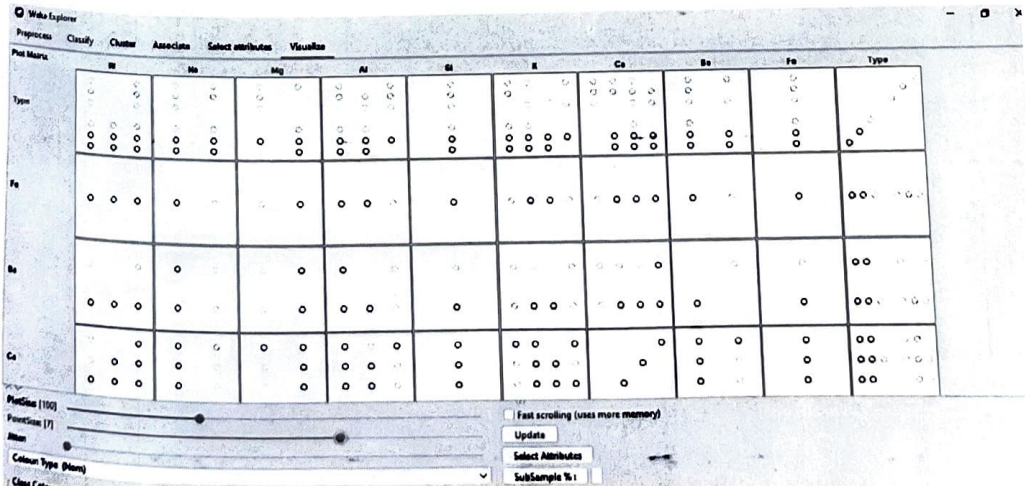
b. Post-Data discretization







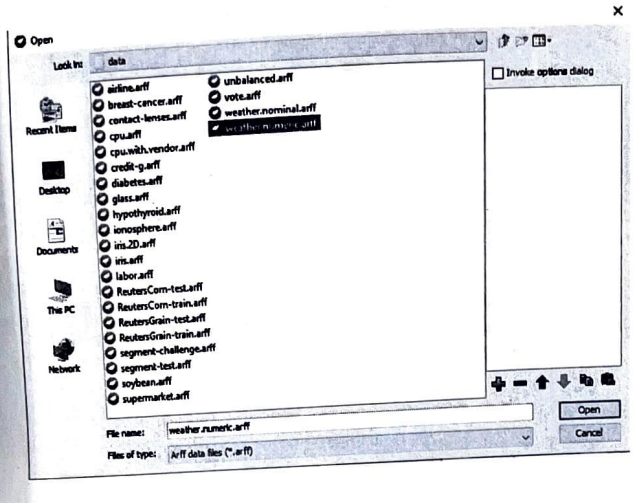
2) Data Visualization

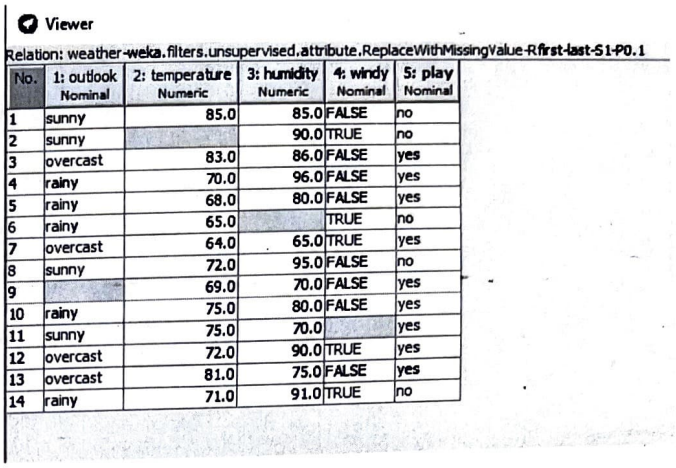


Practical 2: Data Preprocessing

Data Cleaning on – weather numeric .arff

- Replace with missing and replace missing

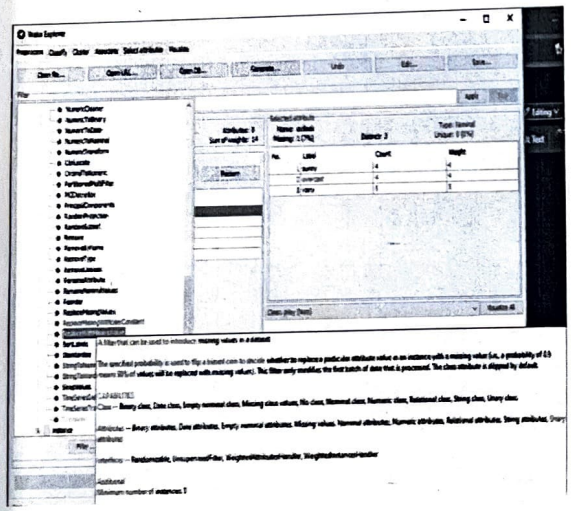


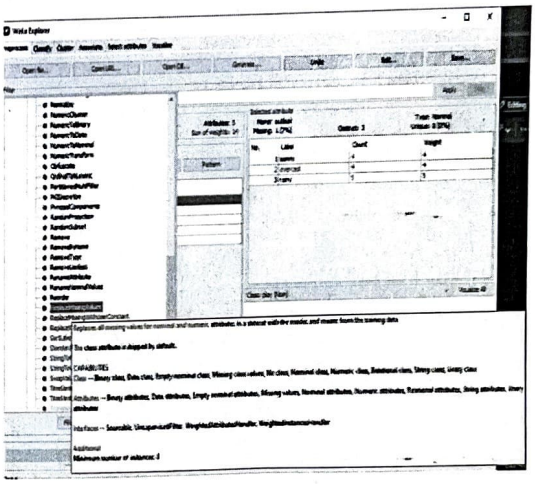


Viewer

Relation: weather-weka.filters.unsupervised.attribute.ReplaceWithMissingValue-Rfirst-last-S1-P0.1

No.	1: outlook Nominal	2: temperature Numeric	3: humidity Numeric	4: windy Nominal	5: play Nominal
1	sunny	85.0	85.0	FALSE	no
2	sunny		90.0	TRUE	no
3	overcast	83.0	86.0	FALSE	yes
4	rainy	70.0	96.0	FALSE	yes
5	rainy	68.0	80.0	FALSE	yes
6	rainy	65.0		TRUE	no
7	overcast	64.0	65.0	TRUE	yes
8	sunny	72.0	95.0	FALSE	no
9		69.0	70.0	FALSE	yes
10	rainy	75.0	80.0	FALSE	yes
11	sunny	75.0	70.0		yes
12	overcast	72.0	90.0	TRUE	yes
13	overcast	81.0	75.0	FALSE	yes
14	rainy	71.0	91.0	TRUE	no



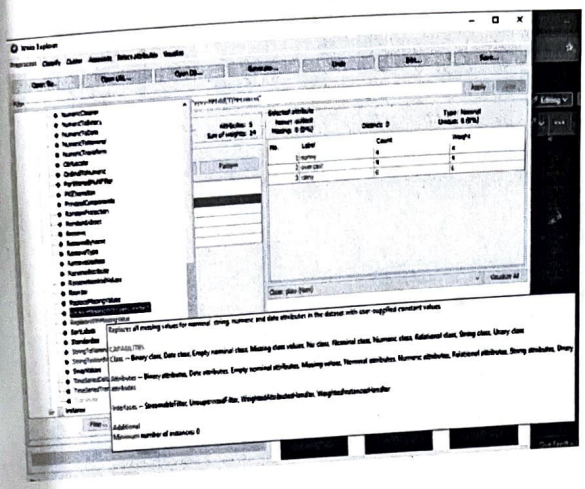


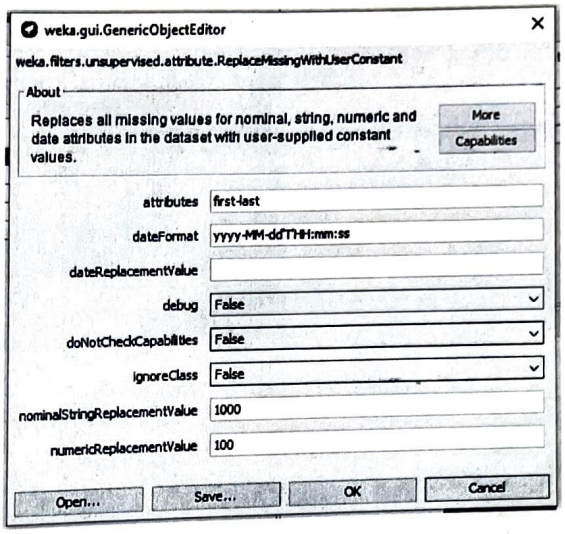
Viewer

Relation: weather-weka.filters.unsupervised.attribute.ReplaceWithMissingValue-F

No.	1: outlook Nominal	2: temperature Numeric	3: humidity Numeric	4: windy Nominal	5: play Nominal
1	sunny	85.0	85.0	FALSE	no
2	sunny	73.07692307...	90.0	TRUE	no
3	overcast	83.0	86.0	FALSE	yes
4	rainy	70.0	96.0	FALSE	yes
5	rainy	68.0	80.0	FALSE	yes
6	rainy	65.0	82.53846...	TRUE	no
7	overcast	64.0	65.0	TRUE	yes
8	sunny	72.0	95.0	FALSE	no
9	rainy	69.0	70.0	FALSE	yes
10	rainy	75.0	80.0	FALSE	yes
11	sunny	75.0	70.0	FALSE	yes
12	overcast	72.0	90.0	TRUE	yes
13	overcast	81.0	75.0	FALSE	yes
14	rainy	71.0	91.0	TRUE	no

- Replace with user constant





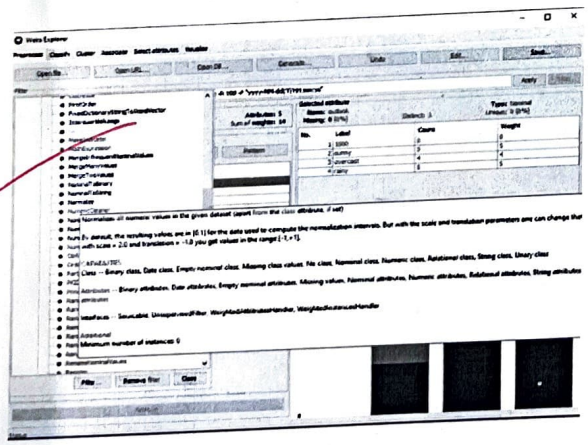
Viewer

Relation: weather-weka.filters.unsupervised.attribute.ReplaceMissingW

No.	1: outlook Nominal	2: temperature Numeric	3: humidity Numeric	4: windy Nominal	5: play Nominal
1	sunny	85.0	85.0	FALSE	no
2	sunny	100.0	90.0	TRUE	no
3	overcast	83.0	86.0	FALSE	yes
4	rainy	70.0	96.0	FALSE	yes
5	rainy	68.0	80.0	FALSE	yes
6	rainy	65.0	100.0	TRUE	no
7	overcast	64.0	65.0	TRUE	yes
8	sunny	72.0	95.0	FALSE	no
9	1000	69.0	70.0	FALSE	yes
10	rainy	75.0	80.0	FALSE	yes
11	sunny	75.0	70.0	1000	yes
12	overcast	72.0	90.0	TRUE	yes
13	overcast	81.0	75.0	FALSE	yes
14	rainy	71.0	91.0	TRUE	no

Transformation

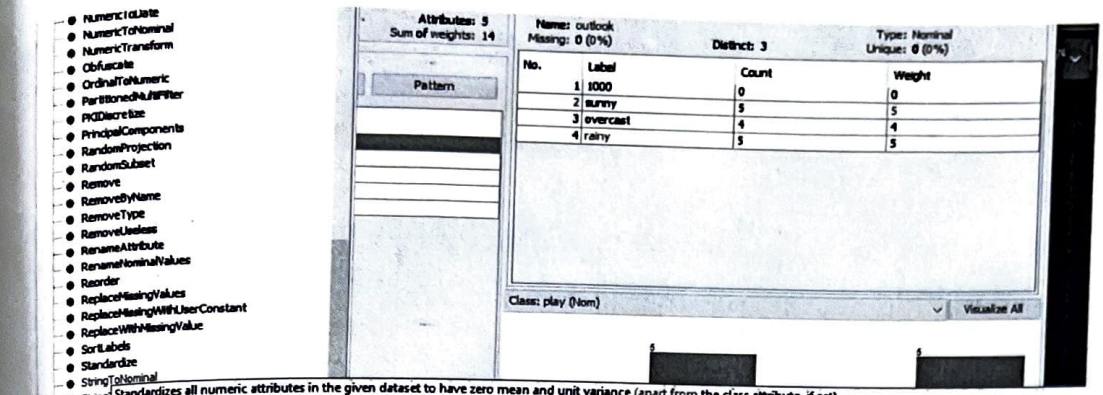
- Normalize and min-max, Z Score



Viewer

Relation: weather-weka.filters.unsupervised.attribute.ReplaceMissingWithUserConsta

No.	1: outlook Nominal	2: temperature Numeric	3: humidity Numeric	4: windy Nominal	5: play Nominal
1	sunny	1.0	0.645161...	FALSE	no
2	sunny	0.761904761...	0.806451...	TRUE	no
3	overcast	0.904761904...	0.677419...	FALSE	yes
4	rainy	0.285714285...	1.0	FALSE	yes
5	rainy	0.190476190...	0.483870...	FALSE	yes
6	rainy	0.047619047...	0.161290...	TRUE	no
7	overcast	0.0	0.0	TRUE	yes
8	sunny	0.380952380...	0.967741...	FALSE	no
9	sunny	0.238095238...	0.161290...	FALSE	yes
10	rainy	0.523809523...	0.483870...	FALSE	yes
11	sunny	0.523809523...	0.161290...	TRUE	yes
12	overcast	0.380952380...	0.806451...	TRUE	yes
13	overcast	0.809523809...	0.322580...	FALSE	yes
14	rainy	0.333333333...	0.838709...	TRUE	no



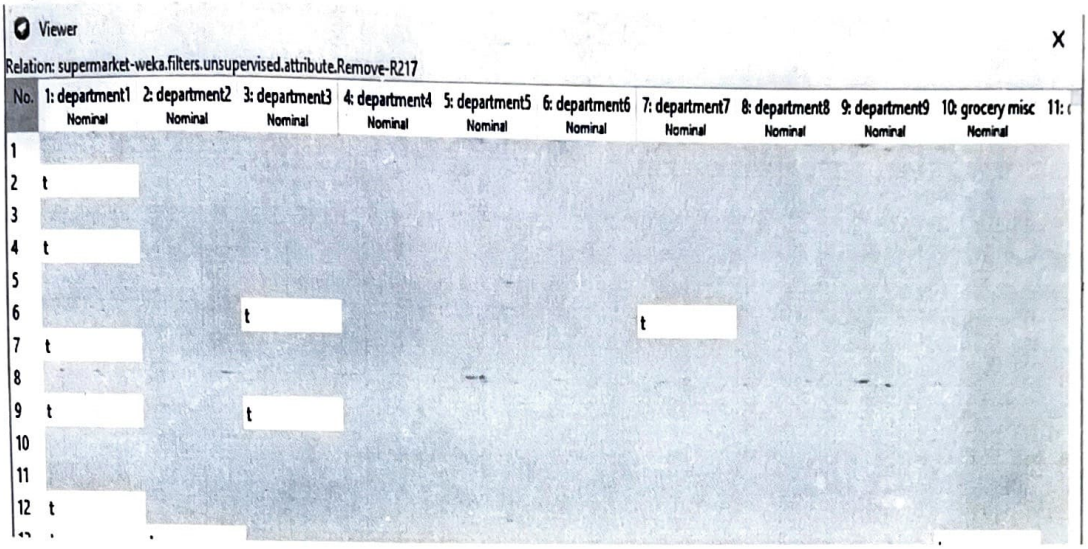
Viewer

Relation: weather-weka.filters.unsupervised.attribute.ReplaceMissingWithUserConst

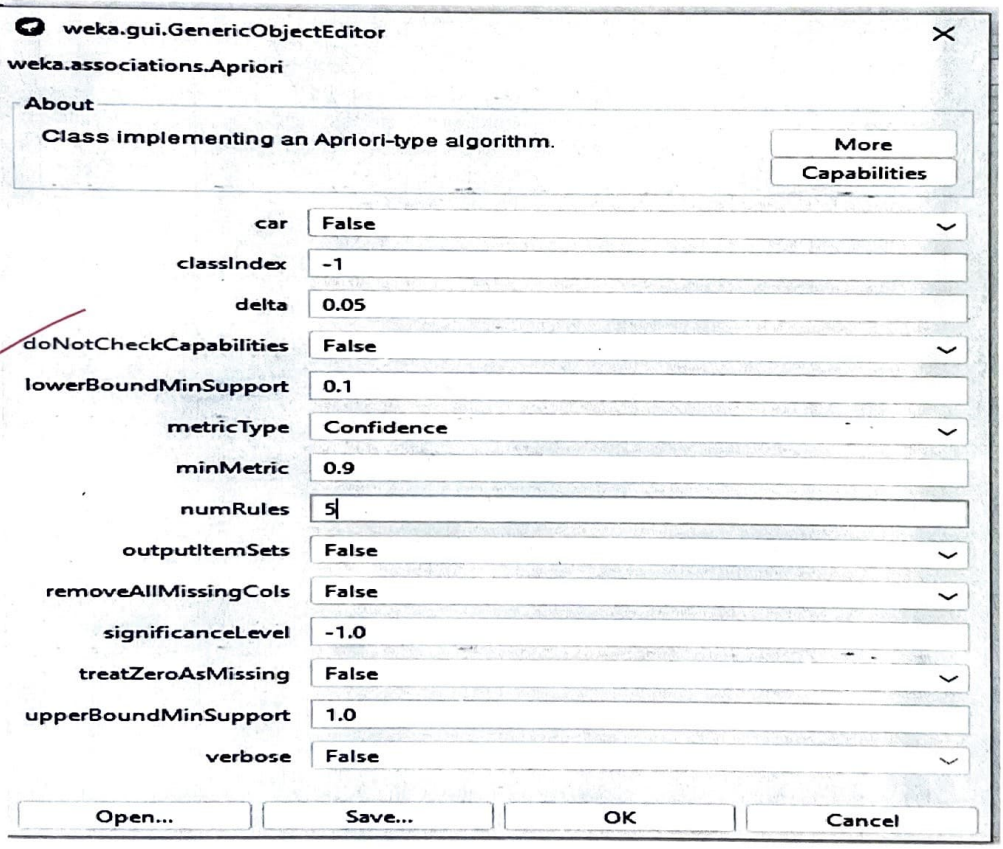
No.	1: outlook Nominal	2: temperature Numeric	3: humidity Numeric	4: windy Nominal	5: play Nominal
1	sunny	1.739067215...	0.326404...	FALSE	no
2	sunny	0.978225308...	0.812539...	TRUE	no
3	overcast	1.434730452...	0.423631...	FALSE	yes
4	rainy	-0.543458504...	1.395900...	FALSE	yes
5	rainy	-0.847795267...	-0.15972...	FALSE	yes
6	rainy	-1.304300411...	-1.13199...	TRUE	no
7	overcast	-1.456468792...	-1.61813...	TRUE	yes
8	sunny	-0.239121742...	1.298673...	FALSE	no
9	sunny	-0.695626886...	-1.13199...	FALSE	yes
10	rainy	0.217383401...	-0.15972...	FALSE	yes
11	sunny	0.217383401...	-1.13199...	TRUE	yes
12	overcast	-0.239121742...	0.812539...	TRUE	yes
13	overcast	1.130393690...	-0.64586...	FALSE	yes
14	rainy	-0.391290123...	0.909766...	TRUE	no

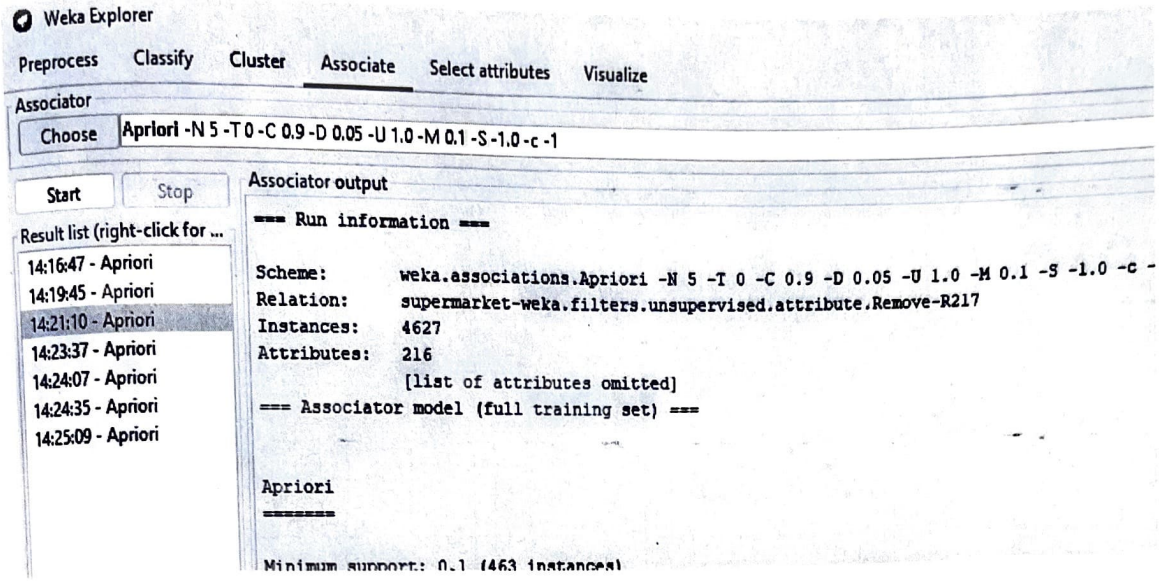
PRACTICAL 3

1. Supermarket Dataset



2. Apriori output





Best rules found:

1. biscuits=t frozen foods=t pet foods=t milk-cream=t vegetables=t 516 ==> bread and cake=t 475 <conf:(0.92)> lift:(1.28) lev:(0.02) [103] conv:(3.44)
2. baking needs=t biscuits=t milk-cream=t margarine=t fruit=t vegetables=t 505 ==> bread and cake=t 464 <conf:(0.92)> lift:(1.28) lev:(0.02) [100] conv:(3.37)
3. biscuits=t frozen foods=t milk-cream=t margarine=t vegetables=t 585 ==> bread and cake=t 537 <conf:(0.92)> lift:(1.28) lev:(0.03) [115] conv:(3.35)
4. biscuits=t canned vegetables=t frozen foods=t fruit=t vegetables=t 536 ==> bread and cake=t 492 <conf:(0.92)> lift:(1.28) lev:(0.02) [106] conv:(3.34)
5. baking needs=t frozen foods=t milk-cream=t margarine=t fruit=t vegetables=t 517 ==> bread and cake=t 474 <conf:(0.92)> lift:(1.27) lev:(0.02) [101] conv:(3.29)

Prathmesh R. Deokar
1022133
COMP A (SEM-5)

EXP. 4: Implementation of Naïve Bayes using data mining tool.

Viewer

Relation: weather.symbolic

No.	1: outlook Nominal	2: temperature Nominal	3: humidity Nominal	4: windy Nominal	5: play Nominal
1	sunny	hot	high	FALSE	no
2	sunny	hot	high	TRUE	no
3	overcast	hot	high	FALSE	yes
4	rainy	mild	high	FALSE	yes
5	rainy	cool	normal	FALSE	yes
6	rainy	cool	normal	TRUE	no
7	overcast	cool	normal	TRUE	yes
8	sunny	mild	high	FALSE	no
9	sunny	cool	normal	FALSE	yes
10	rainy	mild	normal	FALSE	yes
11	sunny	mild	normal	TRUE	yes
12	overcast	mild	high	TRUE	yes
13	overcast	hot	normal	FALSE	yes
14	rainy	mild	high	TRUE	no

```
=== Run information ===

Scheme:     weka.classifiers.bayes.NaiveBayes
Relation:   weather.symbolic
Instances:  14
Attributes: 5
            outlook
            temperature
            humidity
            windy
            play
Test mode:  10-fold cross-validation
=== Classifier model (full training set) ===
Naive Bayes Classifier

                Class
Attribute       yes    no
            (0.63) (0.38)
============================
outlook
```

Experiment 5

Association Rule Mining using Apriori Algorithm

(Programming Language :Python)

Code :

```
from itertools import combinations

def generate_candidates(frequent_itemsets, k):
    candidates = []
    items = sorted(set([item for itemset in frequent_itemsets for item in itemset]))
    for combination in combinations(items, k):
        candidates.append(combination)
    return candidates

def calculate_support(transactions, candidates):
    support_count = {candidate: 0 for candidate in candidates}
    for transaction in transactions:
        for candidate in candidates:
            if set(candidate).issubset(transaction):
                support_count[candidate] += 1
    return support_count

def prune_itemsets(support_count, min_support, total_transactions):
    frequent_itemsets = {}
    for itemset, count in support_count.items():
        support = count / total_transactions
        if support >= min_support:
            frequent_itemsets[itemset] = count
    return frequent_itemsets

def generate_association_rules(frequent_itemsets, min_confidence, total_transactions):
    rules = []
    for itemset, itemset_count in frequent_itemsets.items():
        length = len(itemset)
        if length > 1:
            for i in range(1, length):
                for antecedent in combinations(itemset, i):
                    consequent = tuple(set(itemset) - set(antecedent))
                    antecedent_count = sum(
                        1 for transaction in transactions if set(antecedent).issubset(transaction)
                    )
                    confidence = itemset_count / antecedent_count
                    if confidence >= min_confidence:
                        rules.append((antecedent, consequent, confidence))
    return rules

def input_transactions():
    transactions = []
    print("Enter transactions (enter 'done' when finished):")
    while True:
        transaction = input("Transaction (space-separated items): ").strip()
        if transaction.lower() == 'done':
```

```
            break
        transactions.append(set(map(int, transaction.split())))
    return transactions

transactions = input_transactions()
min_support = float(input("Enter minimum support (as a fraction, e.g., 0.5): "))
min_confidence = float(input("Enter minimum confidence (as a fraction, e.g., 0.7): "))
total_transactions = len(transactions)
items = sorted(set(item for transaction in transactions for item in transaction))
candidates = [(item,) for item in items]
support_count = calculate_support(transactions, candidates)
frequent_itemsets = prune_itemsets(support_count, min_support, total_transactions)
k = 2
while True:
    candidates = generate_candidates(frequent_itemsets, k)
    if not candidates:
        break
    support_count = calculate_support(transactions, candidates)
    new_frequent_itemsets = prune_itemsets(support_count, min_support, total_transactions)
    if not new_frequent_itemsets:
        break
    frequent_itemsets.update(new_frequent_itemsets)
    k += 1
frequent_itemsets_3 = {itemset: count for itemset, count in frequent_itemsets.items() if
len(itemset) == 3}
rules = generate_association_rules(frequent_itemsets_3, min_confidence, total_transactions)
print("\nFrequent Itemsets:")
if not frequent_itemsets_3:
    print("No frequent itemsets of size 3 meet the minimum support threshold.")
else:
    for itemset, count in frequent_itemsets_3.items():
        print(f"{itemset}: {count}")
print("\nAssociation Rules:")
if not rules:
    print("No rules meet the minimum confidence threshold.")
else:
    for antecedent, consequent, confidence in rules:
        print(f"{antecedent} -> {consequent}, confidence: {confidence:.2f}")
```

Output :

```
C:\Users\AARYA\Desktop\Python>python Apriori2.py
Enter transactions (enter 'done' when finished):
Transaction (space-separated items): 1 3 4 6
Transaction (space-separated items): 2 3 5 7
Transaction (space-separated items): 1 2 3 5 8
Transaction (space-separated items): 2 5 9 10
Transaction (space-separated items): 1 4
Transaction (space-separated items): done
Enter minimum support (as a fraction, e.g., 0.5): .4
Enter minimum confidence (as a fraction, e.g., 0.7): .6

Frequent Itemsets:
(2, 3, 5): 2

Association Rules:
(2,) -> (3, 5), confidence: 0.67
(3,) -> (2, 5), confidence: 0.67
(5,) -> (2, 3), confidence: 0.67
(2, 3) -> (5,), confidence: 1.00
(2, 5) -> (3,), confidence: 0.67
(3, 5) -> (2,), confidence: 1.00
```

PRACTICAL –7

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

def euclidean_distance(point1, point2):
  return np.sqrt(np.sum((point1 - point2)
** 2))

def initialize_centroids(data, k):
  return
data[np.random.choice(data.shape[0], k,
replace=False)]

def assign_clusters(data, centroids):
  distances = np.linalg.norm(data[:,
np.newaxis] - centroids, axis=2)
  return np.argmin(distances, axis=1)

def update_centroids(data, labels, k):
  return np.array([data[labels ==
i].mean(axis=0) for i in range(k)])

def k_means(data, k,
max_iterations=100):
  centroids = initialize_centroids(data, k)
  for _ in range(max_iterations):
    labels = assign_clusters(data,
centroids)
    new_centroids =
update_centroids(data, labels, k)
    if
np.all(np.linalg.norm(new_centroids -
centroids, axis=1) < 1e-6):
      break
    centroids = new_centroids
  return centroids, labels

def plot_clusters(data, centroids, labels):
  plt.figure(figsize=(8, 6))
  plt.scatter(data[:, 0], data[:, 1],
c=labels, cmap='viridis', marker='o',
edgecolor='k')
  plt.scatter(centroids[:, 0], centroids[:,
1], c='red', s=200, marker='X',
label='Centroids')
  plt.title('K-means Clustering')
  plt.xlabel('Feature 1')
  plt.ylabel('Feature 2')
  plt.legend()
  plt.grid(True)
  plt.show()

# Example usage:
np.random.seed(0) # For reproducibility
data = np.array([
  [1.0, 2.0],
  [1.5, 1.8],
  [5.0, 8.0],
  [8.0, 8.0],
  [1.0, 0.6],
  [9.0, 11.0],
  [8.0, 2.0],
  [10.0, 2.0],
  [9.0, 3.0]
])

k = 3
centroids, labels = k_means(data, k)
plot_clusters(data, centroids, labels)
```

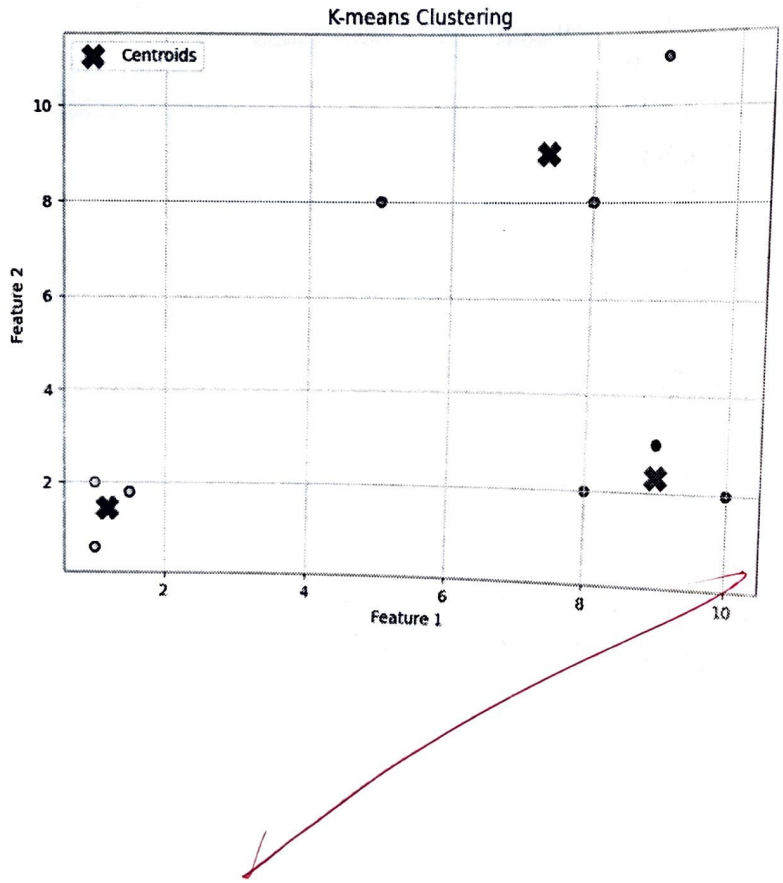
OUTPUT:

Centroids:

[(9.0,2.3333333333333335),
(1.1666666666666667,
1.4666666666666666),(7.333333333333
333, 9.0)]

Clusters:

[[(8.0, 2.0), (10.0, 2.0), (9.0, 3.0)], [(1.0,
2.0), (1.5, 1.8), (1.0, 0.6)], [(5.0, 8.0), (8.0,
8.0), (9.0, 11.0)]]



PRACTICAL - 08

=== Run information ===

Scheme: weka.clusterers.HierarchicalClusterer -N 4 -L AVERAGE -P -A "weka.core.EuclideanDistance -R first-last"
Relation: contact-lenses
Instances: 24
Attributes: 5
age
spectacle-prescrip
astigmatism
tear-prod-rate
contact-lenses
Test mode: evaluate on training data

=== Clustering model (full training set) ===

Cluster 0
((((2.0:1,2.0:1):0.20711,(2.0:1,2.0:1):0.20711):0.18301,(2.0:1,2.0:1):0.39012):0.061,((2.0:1,2.0:1):0,2.0:1):0.45112)

Cluster 1
((0.0:1,0.0:1):0.20711,((0.0:1,0.0:1):0.20711,0.0:1.20711):0)

Cluster 2
((1.0:1,1.0:1):0.20711,(1.0:1,1.0:1):0.20711)

Cluster 3
(((2.0:1,2.0:1):0.20711,(2.0:1,2.0:1):0.20711):0.14328,(2.0:1,2.0:1):0.35039)

Time taken to build model (full training data) : 0 seconds

=== Model and evaluation on training set ===

Clustered Instances

0 9 (38%)
1 5 (21%)
2 4 (17%)
3 6 (25%)

PRACTICAL 9

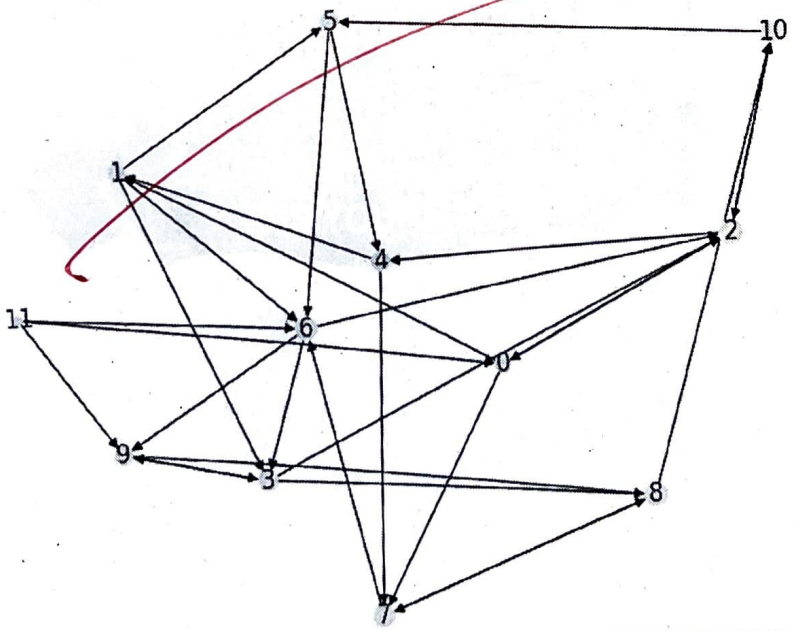
CODE:

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
def pagerank(adjacency_matrix, num_iterations:
int = 100, damping_factor: float = 0.85):
    n = adjacency_matrix.shape[0]
    row_sums = adjacency_matrix.sum(axis=1,
keepdims=True)
    stochastic_matrix = adjacency_matrix /
row_sums
    pagerank = np.ones(n) / n
    for _ in range(num_iterations):
        pagerank = damping_factor *
stochastic_matrix.T @ pagerank + (1 -
damping_factor) / n
    return pagerank
def create_graph(adjacency_matrix):
    G = nx.DiGraph(adjacency_matrix)
    return G
def draw_graph(G, pagerank):
    """ Draw the directed graph with PageRank
values as node sizes. """
    pos = nx.spring_layout(G, seed=42, k=0.5)  #
Adjust k for more spacing
    sizes = 1000 * pagerank  # Scale PageRank
values for visualization
    nx.draw(G, pos, with_labels=True,
node_size=sizes, node_color='lightblue',
font_size=12, arrows=True)
    plt.title("Directed Graph with PageRank
Values")
    plt.show()
if __name__ == "__main__":
    adjacency_matrix = np.array([[0, 1, 1, 0, 0, 0,
0, 1, 0, 0, 0, 0],
                                 [0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0],
                                 [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0],
                                 [0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0],
                                 [0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
                                 [0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0],
                                 [0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0],
                                 [0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
                                 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0],
                                 [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0],
                                 [0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                                 [1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
0]])

    pagerank_values =
pagerank(adjacency_matrix)
    print("PageRank values:", pagerank_values)

    # Create and draw the graph
    G = create_graph(adjacency_matrix)
    draw_graph(G, pagerank_values)
```

OUTPUT:




```
PageRank values: [0.05173261 0.06049222 0.12596804 0.09202097 0.07843447 0.07116123
```

PRACTICAL 9

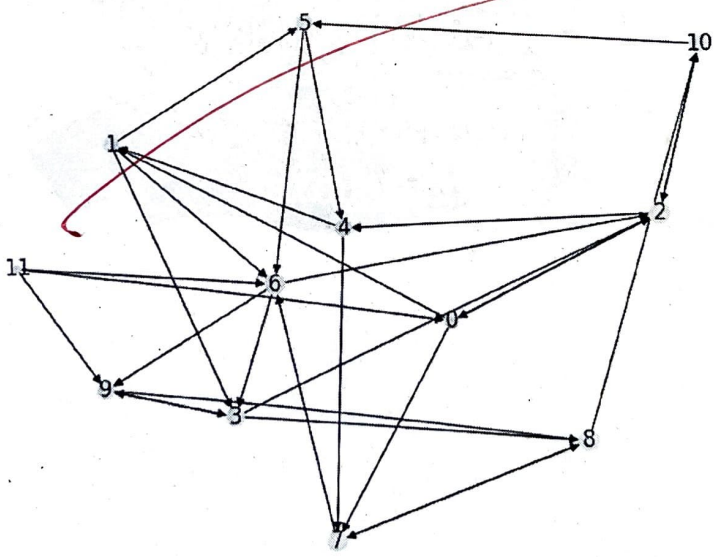
CODE:

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
def pagerank(adjacency_matrix, num_iterations: int = 100, damping_factor: float = 0.85):
    n = adjacency_matrix.shape[0]
    row_sums = adjacency_matrix.sum(axis=1, keepdims=True)
    stochastic_matrix = adjacency_matrix / row_sums
    pagerank = np.ones(n) / n
    for _ in range(num_iterations):
        pagerank = damping_factor * stochastic_matrix.T @ pagerank + (1 - damping_factor) / n
    return pagerank
def create_graph(adjacency_matrix):
    G = nx.DiGraph(adjacency_matrix)
    return G
def draw_graph(G, pagerank):
    """ Draw the directed graph with PageRank values as node sizes. """
    pos = nx.spring_layout(G, seed=42, k=0.5)  # Adjust k for more spacing
    sizes = 1000 * pagerank  # Scale PageRank values for visualization
    nx.draw(G, pos, with_labels=True, node_size=sizes, node_color='lightblue', font_size=12, arrows=True)
    plt.title("Directed Graph with PageRank Values")
    plt.show()
if __name__ == "__main__":
    adjacency_matrix = np.array([[0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0],
                                 [0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0],
                                 [1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0],
                                 [0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0],
                                 [0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
                                 [0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0],
                                 [0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0],
                                 [0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
                                 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0],
                                 [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0],
                                 [0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0],
                                 [1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0]])

    pagerank_values = pagerank(adjacency_matrix)
    print("PageRank values:", pagerank_values)

    # Create and draw the graph
    G = create_graph(adjacency_matrix)
    draw_graph(G, pagerank_values)
```

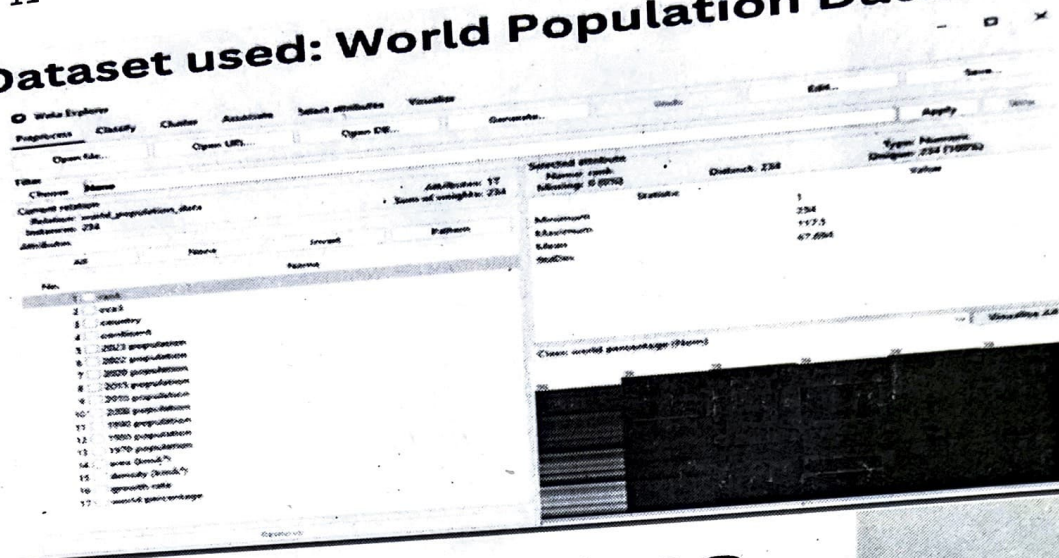
OUTPUT:




```
PageRank values: [0.05173261 0.06049222 0.12596804 0.09202097 0.07843447 0.07116123
 0.11017446 0.10999955 0.11648782 0.07333037 0.09769827 0.0125    ]
```

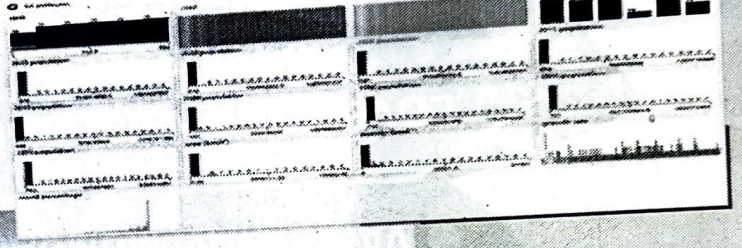
Experiment : 11

Dataset used: World Population Dataset

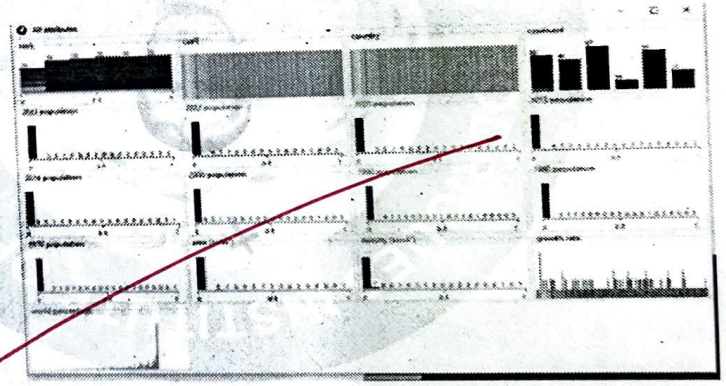


PREPROCESSING

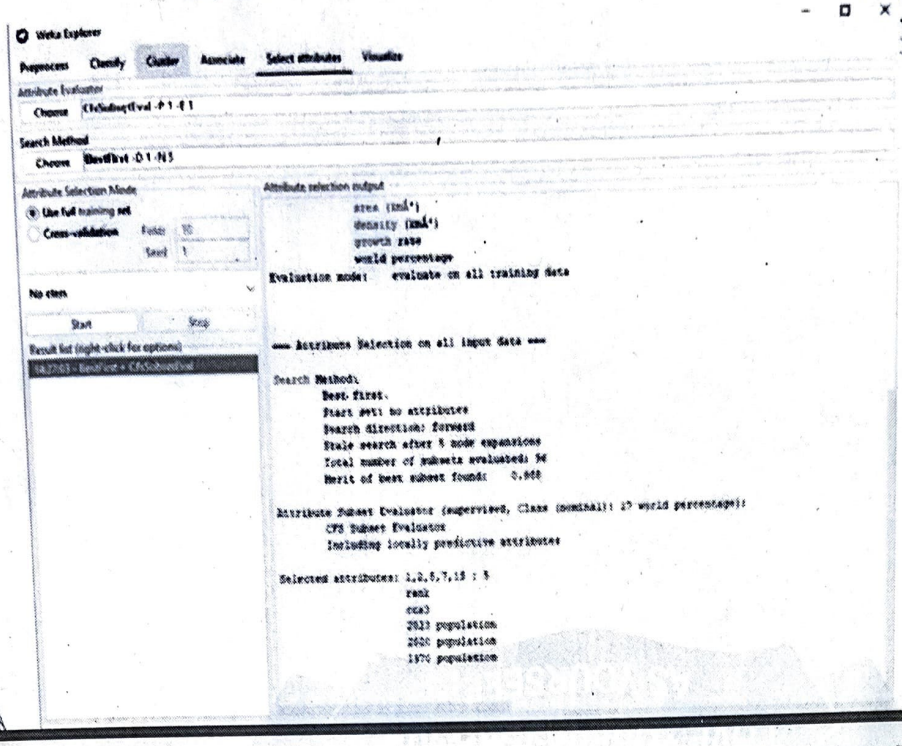
Replacing Missing values



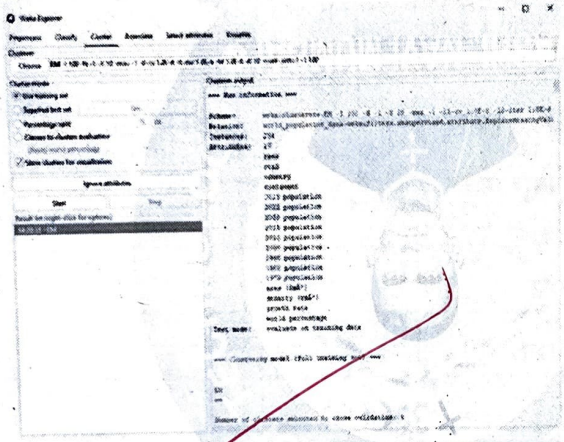
Normalizing



FEATURE SELECTION



Clustering using EM [Expectation -Maximization]



Clustering EM