Experiment Number : 7 - Exploration of data normalization and data discretization techniques

---

Batch: FDS-2 **Roll Number: 1601042223** **Name: Chandana Ramesh Galgali**

Aim of the Experiment: Exploration of data normalization and data discretization techniques

---

Program/ Steps:

1. Identify the suitable attributes to apply the following normalization techniques and display the consolidated output of all 3 normalizations.
    a. Min-Max Normalization
    b. Z-score Normalization
    c. Decimal Scaling
2. Identify the suitable attributes to apply the K-means discretisation techniques and display the output of discretized data

---

Code with Output/Result:

Decimal Scaling:

```
import numpy as np
def decimal_scaling(data):
  max_value=np.max(np.abs(data))
  scaled_data=data/(10**(len(str(int(max_value))))-1)
  return scaled_data
data=np.array([25,100,1000,5000])
scaled_data=decimal_scaling(data)
print(scaled_data)
```

```
[0.00250025 0.010001   0.10001    0.50005001]
```

Z-Score (Standardization):

```
import numpy as np
def z_score_scaling(data):
  mean = np.mean(data)
  std_dev = np.std(data)
  standardized_data = (data - mean) / std_dev
  return standardized_data
data = np.array([25, 100, 1000, 5000])
standardized_data = z_score_scaling(data)
print(standardized_data)
```

```
[-0.73868366 -0.70190273 -0.26053158  1.70111796]
```

Min-Max Scaling:

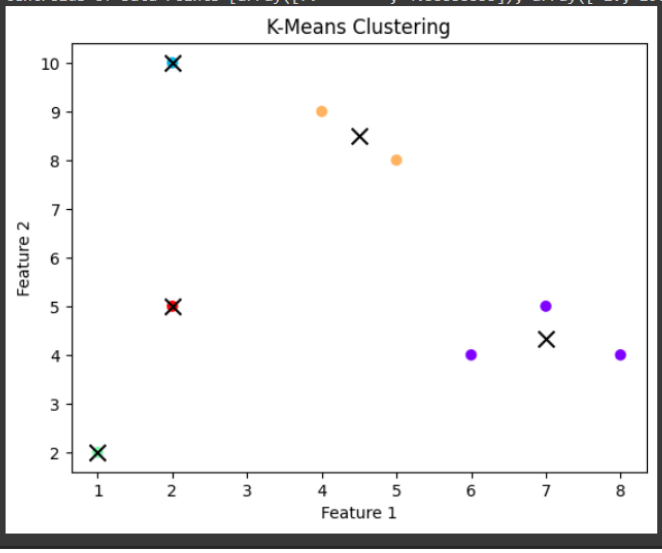
```
import numpy as np
def min_max_scaling(data, min_value=0, max_value=1):
  min_data = np.min(data)
  max_data = np.max(data)
  scaled_data = (data - min_data) / (max_data - min_data) * (max_value -min_value) + min_value
  return scaled_data
data = np.array([25, 100, 1000, 5000])
scaled_data = min_max_scaling(data)
print(scaled_data)
```

```
[0.         0.01507538 0.1959799  1.        ]
```

Clustering of Data(K-means Method):

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
X = np.array([[2, 10], [2, 5], [8, 4], [5, 8], [7, 5], [6, 4],[1,2],[4,9]])
K = 5
max_iterations = 100
centroids = X[np.random.choice(range(len(X)), K, replace=False)]
def euclidean_distance(x1, x2):
  return np.sqrt(np.sum((x1 - x2) ** 2))
for _ in range(max_iterations):
  cluster_assignments = []
  for x in X:
    distances = [euclidean_distance(x, centroid) for centroid in centroids]
    closest_cluster = np.argmin(distances)
    cluster_assignments.append(closest_cluster)
  new_centroids = []
  for cluster_id in range(K):
    cluster_points = [X[i] for i, cluster in enumerate(cluster_assignments) if cluster == cluster_id]
    if len(cluster_points) == 0:
      new_centroids.append(centroids[cluster_id])
    else:
      new_centroid = np.mean(cluster_points, axis=0)
      new_centroids.append(new_centroid)
  if np.all(np.array(centroids) == np.array(new_centroids)):
    break
  centroids = new_centroids
for i in range(3):
    print("Centroids of Data Points",centroids)
cluster_assignments = np.array(cluster_assignments)
plt.scatter(X[:, 0], X[:, 1], c=cluster_assignments, cmap='rainbow')
plt.scatter(np.array(centroids)[:, 0], np.array(centroids)[:, 1], c='black',
marker='x', s=100)
plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

```
Centroids of Data Points [array([7.        , 4.33333333]), array([ 2., 10.]), array([1., 2.]), array([4.5, 8.5]), array([2., 5.])]
Centroids of Data Points [array([7.        , 4.33333333]), array([ 2., 10.]), array([1., 2.]), array([4.5, 8.5]), array([2., 5.])]
Centroids of Data Points [array([7.        , 4.33333333]), array([ 2., 10.]), array([1., 2.]), array([4.5, 8.5]), array([2., 5.])]
```



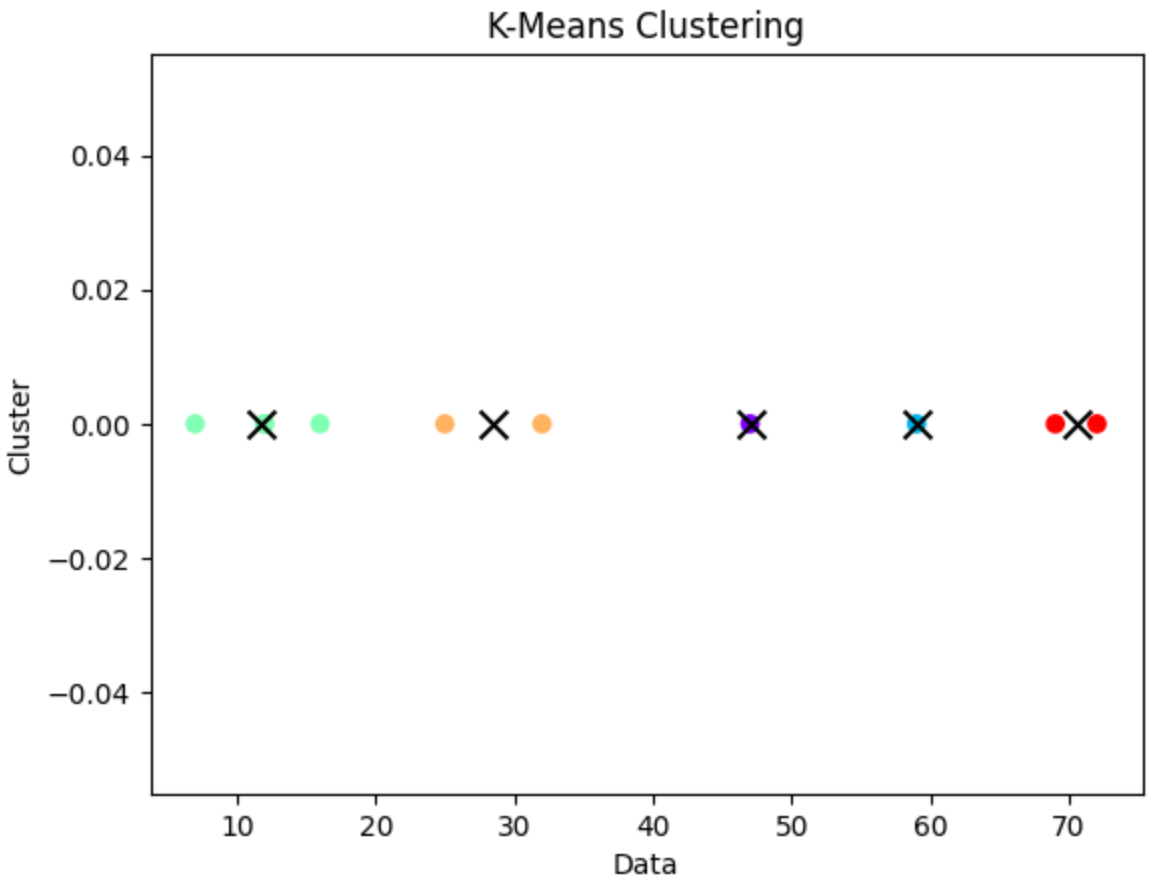
Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
def decimal_scaling(data):
    max_value = np.max(np.abs(data))
    scaled_data = data / (10 ** (len(str(int(max_value)))) - 1)
    return scaled_data
def z_score_scaling(data):
    mean = np.mean(data)
    std_dev = np.std(data)
    standardized_data = (data - mean) / std_dev
    return standardized_data
def min_max_scaling(data, min_value=0, max_value=1):
    min_data = np.min(data)
    max_data = np.max(data)
      scaled_data = (data - min_data) / (max_data - min_data) * (max_value - min_value) +
min_value
    return scaled_data
dataframe = pd.read_csv(r'/content/Flight_delay.csv')
data_array = dataframe.to_numpy()
data = data_array[:10, 28]
scaled_data = decimal_scaling(data)
print("Decimal Scaled:", scaled_data)
standardized_data = z_score_scaling(data)
print("Z-score Scaled:", standardized_data)
scaled_data = min_max_scaling(data)
print("Min-max Scaled:", scaled_data)
K = 5
max_iterations = 100
centroids = data[np.random.choice(range(len(data)), K, replace=False)]
def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))
for _ in range(max_iterations):
    cluster_assignments = []
    for x in data:
        distances = [euclidean_distance(x, centroid) for centroid in centroids]
        closest_cluster = np.argmin(distances)
```

```
        cluster_assignments.append(closest_cluster)
    new_centroids = []
    for cluster_id in range(K):
            cluster_points = [data[i] for i, cluster in enumerate(cluster_assignments) if cluster ==
cluster_id]
        if len(cluster_points) == 0:
            new_centroids.append(centroids[cluster_id])
        else:
            new_centroid = np.mean(cluster_points, axis=0)
            new_centroids.append(new_centroid)
    if np.all(np.array(centroids) == np.array(new_centroids)):
        break
    centroids = new_centroids
for i in range(3):
    print("Centroids of Data Points", centroids)
cluster_assignments = np.array(cluster_assignments)
plt.scatter(data, np.zeros_like(data), c=cluster_assignments, cmap='rainbow')
plt.scatter(centroids, np.zeros_like(centroids), c='black', marker='x', s=100)
plt.title('K-Means Clustering')
plt.xlabel('Data')
plt.ylabel('Cluster')
plt.show()
```

Result:

```
Decimal Scaled: [0.32323232323232326 0.47474747474747475 0.7272727272727273
 0.12121212121212122 0.16161616161616163 0.25252525252525254
 0.12121212121212122 0.0707070707070707 0.5959595959595959
 0.696969696969697]
Z-score Scaled: [-0.1315059500433155 0.5048131630695011 1.5653450182575288
 -0.9799314341937376 -0.8102463373636531 -0.42845486949596323
 -0.9799314341937376 -1.1920378052313432 1.0138684535597544
 1.4380811956349655]
Min-max Scaled: [0.38461538461538464 0.6153846153846154 1.0 0.07692307692307693
 0.13846153846153847 0.27692307692307694 0.07692307692307693 0.0 0.8
 0.9538461538461539]
Centroids of Data Points [47.0, 59.0, 11.75, 28.5, 70.5]
Centroids of Data Points [47.0, 59.0, 11.75, 28.5, 70.5]
Centroids of Data Points [47.0, 59.0, 11.75, 28.5, 70.5]
```



Post Lab Question-Answers:

1. What happens if data is not normalized?

Ans: If data is not normalized, it can lead to several issues and challenges in data analysis and machine learning tasks. Here are some potential consequences:

1. Inaccurate Comparisons: Without normalization, data from different scales and units can't be compared accurately. This can result in misleading conclusions and incorrect interpretations. For example, if you have a dataset with features like age (ranging from 0 to 100) and income (ranging from 0 to 1,000,000), the income feature will dominate the analysis due to its larger scale.

2. Biased Model Training: Machine learning algorithms often rely on numerical optimization techniques that assume the data is normalized. Failure to normalize the data can lead to biased model training, where certain features with larger scales or variances disproportionately influence the model's learning process. This can result in suboptimal or unreliable models.

3. Slow Convergence: Normalization helps algorithms converge faster during training. When data is not normalized, the optimization process may take longer to reach an optimal solution. This can increase the training time and computational resources required.

4. Sensitivity to Outliers: Outliers, which are extreme values in the dataset, can have a significant impact on models that are not robust to them. Normalization can help mitigate the influence of outliers by scaling the data appropriately. Without normalization, outliers can distort the overall distribution and affect the model's performance.

5. Interpretability Issues: Normalization can improve the interpretability of the data by making it easier to understand and visualize. When data is not normalized, it becomes challenging to compare and interpret the values accurately, especially when dealing with multiple features or dimensions.

In summary, data normalization is crucial for ensuring fair comparisons, avoiding biased models, improving convergence speed, handling outliers, and enhancing interpretability in data analysis and machine learning tasks.

2. Is there any criteria for selecting attributes for the K-means algorithm. If so, discuss.

Ans: Yes, there are criteria for selecting attributes (also known as features or variables) when using the K-means algorithm. The choice of attributes can significantly impact the performance and effectiveness of the algorithm. Here are some criteria to consider:

1. Relevance: Select attributes that are relevant to the problem you are trying to solve. Irrelevant or redundant attributes can introduce noise and unnecessary complexity to the clustering process. It's important to choose attributes that capture meaningful information about the data.

2. Scale and Units: Ensure that the attributes have similar scales and units. K-means is a distance-based algorithm, and the distance between data points is calculated using the attributes. If the attributes have different scales or units, it can lead to biased clustering results. Therefore, it is often recommended to normalize or standardize the attributes before applying K-means.

3. Independence: Choose attributes that are as independent as possible. Highly correlated attributes can introduce bias and redundancy in the clustering process. If two attributes are strongly correlated, they may provide similar information, and using both of them may not add much value. Consider using techniques like feature selection or dimensionality reduction to identify and remove redundant attributes.

4. Interpretability: Select attributes that are interpretable and meaningful to the problem domain. K-means clustering aims to find natural groupings in the data, and using attributes that are easily interpretable can help in understanding and interpreting the resulting clusters. This can be particularly important when using the clustering results for decision-making or further analysis.

5. Computational Efficiency: Consider the computational cost of using certain attributes. Some attributes may require more computational resources or time to process, especially if they involve complex calculations or large amounts of data. It's important to strike a balance between the usefulness of the attributes and the computational efficiency of the algorithm.

6. Domain Knowledge: Leverage domain knowledge and expertise to guide the selection of attributes. Domain experts often have valuable insights into the problem and can provide guidance on which attributes are likely to be relevant and informative for clustering.

It's worth noting that the criteria for attribute selection may vary depending on the specific problem, dataset, and goals of the analysis. It's always a good practice to experiment with different attribute combinations and evaluate the clustering results to determine the most suitable attributes for the K-means algorithm in a given context.

Outcomes: Apply the transformations required on data to make it suitable for Mining

---

Conclusion (based on the Results and outcomes achieved):

The experiment on data normalization and data discretization techniques emphasized their significance in data analysis. Normalizing the data and selecting relevant attributes improved the accuracy, interpretability, and performance of the analysis. Additionally, data discretization techniques proved useful in simplifying complex data and enhancing the interpretability of the results.

References:

Books/ Journals/ Websites

1. Han, Kamber, "Data Mining Concepts and Techniques", Morgan Kaufmann 3nd Edition

---