Lab 1: Data Pre-processing

Introduction: This lab report details the implementation and results of various data pre-processing techniques, including data cleaning, normalization, data binning, discretization, and feature selection.

1.1 Data Cleaning:

Datasets

ID	Name	Age	Department	Salary
1	John	28	HR	50000
2	Jane	35	Finance	60000
3	Emily		HR	55000
4	Michael	40	Human Resources	
5	Sarah	29	IT	52000
6	David	50	Finance	75000
7	Laura	38	H.R.	68000
8	Robert	32	HR	57000
9	Linda	45	IT	62000
10	James	30	HR	51000

Implementation Code:

```
import pandas as pd
print("Sudha Lab")
df = pd.read_csv('employee_data.csv')
print("Initial Data:\n", df.head())
df['Age'] = df['Age'].fillna(df['Age'].mean())
df['Salary'] = df['Salary'].fillna(df['Salary'].mean())
df['Department'] = df['Department'].replace({
    'Human Resources': 'HR',
    'H.R.': 'HR',
    'hr': 'HR'
})
df.drop_duplicates(subset='ID', keep='first', inplace=True)
print("\nCleaned Data:\n", df.head())
```

Output SnapShot:

```
Sudha Lab
Initial Data:
    ID     Name   Age       Department   Salary
0   1     John  28.0               HR  50000.0
1   2     Jane  35.0          Finance  60000.0
2   3    Emily   NaN               HR  55000.0
3   4  Michael  40.0  Human Resources      NaN
4   5    Sarah  29.0               IT  52000.0

Cleaned Data:
    ID     Name   Age Department   Salary
0   1     John  28.0         HR  50000.0
1   2     Jane  35.0    Finance  60000.0
2   3    Emily  35.7         HR  55000.0
3   4  Michael  40.0         HR  58100.0
4   5    Sarah  29.0         IT  52000.0
```

1.2 Normalization

Datasets

StudentID	Math	Science	English
1	78	65	80
2	88	75	85
3	60	50	55
4	90	78	92
5	55	48	58
6	83	72	88
7	71	66	79
8	64	52	70
9	88	80	90
10	76	68	82

Implementation Code

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

print("Sudha Lab")
df = pd.read_csv('/content/student_scores.csv')
print("Initial Data:\n", df.head())

scaler = MinMaxScaler()
df[['Math', 'Science', 'English']] = scaler.fit_transform(df[['Math', 'Science', 'English']])
print("\nNormalized Scores:\n", df.head())
```

Output Snapshot

```
Sudha Lab
Initial Data:
    StudentID  Math  Science  English
0          1    78       65       80
1          2    88       75       85
2          3    60       50       55
3          4    90       78       92
4          5    55       48       58

Normalized Scores:
    StudentID      Math  Science   English
0          1  0.657143  0.53125  0.675676
1          2  0.942857  0.84375  0.810811
2          3  0.142857  0.06250  0.000000
3          4  1.000000  0.93750  1.000000
4          5  0.000000  0.00000  0.081081
```

1.3 Data Binning

DataSets

CustomerID	Age
1	25
2	42
3	36
4	53
5	28
6	47
7	31
8	50
9	22
10	60

Implementation Code:

```
import pandas as pd
print("Sudha Lab")
df = pd.read_csv('customer_ages.csv')
print("Initial Data:\n", df.head())
bins = [18, 30, 50, 100]
labels = ['Young', 'Middle-aged', 'Senior']
df['AgeGroup'] = pd.cut(df['Age'], bins=bins, labels=labels, right=False)
print("\nData after Binning:\n", df.head())
age_group_distribution = df['AgeGroup'].value_counts()
print("\nAge Group Distribution:\n", age_group_distribution)
```

Output SnapShot:

```
Sudha Lab
Initial Data:
    CustomerID  Age
0           1   25
1           2   42
2           3   36
3           4   53
4           5   28

Data after Binning:
    CustomerID  Age     AgeGroup
0           1   25        Young
1           2   42  Middle-aged
2           3   36  Middle-aged
3           4   53       Senior
4           5   28        Young

Age Group Distribution:
 AgeGroup
Middle-aged    7
Young          5
Senior         3
Name: count, dtype: int64
```

1.4 Discretization

DataSets

Month	Sales
January	15000
February	18000
March	12000
April	30000
May	22000
June	5000
July	8000
August	25000
September	10000
October	20000

Implementation Code

```
import pandas as pd
print("Sudha Lab")
df = pd.read_csv('sales_data.csv')
print("Initial Data:\n", df.head())
bins = [0, 5000, 20000, float('inf')]
labels = ['Low', 'Medium', 'High']
df['SalesCategory'] = pd.cut(df['Sales'], bins=bins, labels=labels)
print("\nData after Discretization:\n", df.head())
sales_category_distribution = df['SalesCategory'].value_counts()
print("\nSales Category Distribution:\n", sales_category_distribution)
```

Output SnapShot

```
Sudha Lab
Initial Data:
        Month  Sales
0    January  15000
1   February  18000
2      March  12000
3      April  30000
4        May  22000

Data after Discretization:
        Month  Sales SalesCategory
0    January  15000        Medium
1   February  18000        Medium
2      March  12000        Medium
3      April  30000          High
4        May  22000          High

Sales Category Distribution:
 SalesCategory
Medium    7
High      4
Low       1
Name: count, dtype: int64
```

1.5 Feature Selection

DataSets

PatientID	Age	BloodPressure	Cholesterol	Glucose	HeartRate	Disease
1	45	130	180	95	70	1
2	50	140	200	105	75	1
3	60	150	240	120	80	1
4	40	120	170	90	65	0
5	35	110	160	85	60	0
6	55	145	210	115	78	1
7	42	135	190	100	72	0
8	38	115	150	80	68	0
9	47	125	170	95	70	1
10	53	140	210	110	76	1

Implementation Code

```
import pandas as pd
from sklearn.feature_selection import SelectKBest, chi2
print("Sudha lab")

df = pd.read_csv('/content/medical_data.csv')
print("Initial Data:\n", df.head())

X = df.drop(columns=['Disease'])
y = df['Disease']

selector = SelectKBest(score_func=chi2, k=3)
selector.fit(X, y)

top_features = X.columns[selector.get_support()]
print("\nTop 3 Features for Predicting Disease:\n", top_features)
```

Output SnapShot

```
Sudha lab
Initial Data:
    PatientID  Age  BloodPressure  Cholesterol  Glucose  HeartRate  Disease
0          1   45            130          180       95         70        1
1          2   50            140          200      105         75        1
2          3   60            150          240      120         80        1
3          4   40            120          170       90         65        0
4          5   35            110          160       85         60        0

Top 3 Features for Predicting Disease:
 Index(['Age', 'Cholesterol', 'Glucose'], dtype='object')
```

Lab-2 Association Mining

Introduction:

Association mining is a data mining technique used to discover interesting relationships, patterns, or associations among items in large datasets. In this lab, association rule mining is performed using the Apriori algorithm on a small transactional dataset. The process includes transforming the data into one-hot encoded format, identifying frequent itemsets with minimum support, and generating strong association rules based on confidence and lift.

Implementation Code:

```python
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
data = {
'TransactionID': [1, 2, 3, 4, 5],
'Items': [
['Bread', 'Milk'],
['Bread', 'Diaper', 'Beer', 'Eggs'],
['Milk', 'Diaper', 'Beer', 'Coke'],
['Bread', 'Milk', 'Diaper', 'Beer'],
['Bread', 'Milk', 'Diaper', 'Coke']
]
}
df = pd.DataFrame(data)
print("Sudha Lab")
print("Initial Data:\n", df)
df_items = df['Items'].apply(lambda x: pd.Series(1, index=x)).fillna(0)
print("\nOne-Hot Encoded Data:\n", df_items)
frequent_itemsets = apriori(df_items, min_support=0.6, use_colnames=True)
print("\nFrequent Itemsets:\n", frequent_itemsets)
rules = association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.7)
print("\nAssociation Rules:\n", rules)
for _, row in rules.iterrows():
    print(f"\nRule: {set(row['antecedents'])} -> {set(row['consequents'])}")
    print(f"Support: {row['support']:.2f}")
    print(f"Confidence: {row['confidence']:.2f}")
    print(f"Lift: {row['lift']:.2f}")
```

Output SnapShot:

```
Sudha Lab
Initial Data:
    TransactionID                         Items
0              1                 [Bread, Milk]
1              2  [Bread, Diaper, Beer, Eggs]
2              3   [Milk, Diaper, Beer, Coke]
3              4  [Bread, Milk, Diaper, Beer]
4              5  [Bread, Milk, Diaper, Coke]

One-Hot Encoded Data:
    Bread  Milk  Diaper  Beer  Eggs  Coke
0    1.0   1.0     0.0   0.0   0.0   0.0
1    1.0   0.0     1.0   1.0   1.0   0.0
2    0.0   1.0     1.0   1.0   0.0   1.0
3    1.0   1.0     1.0   1.0   0.0   0.0
4    1.0   1.0     1.0   0.0   0.0   1.0

Frequent Itemsets:
    support          itemsets
0      0.8           (Bread)
1      0.8            (Milk)
2      0.8          (Diaper)
3      0.6            (Beer)
4      0.6     (Milk, Bread)
5      0.6   (Diaper, Bread)
6      0.6    (Diaper, Milk)
7      0.6    (Diaper, Beer)
```

```
Association Rules:
  antecedents consequents  antecedent support  consequent support  support  \
0      (Milk)     (Bread)                 0.8                 0.8      0.6
1     (Bread)      (Milk)                 0.8                 0.8      0.6
2    (Diaper)     (Bread)                 0.8                 0.8      0.6
3     (Bread)    (Diaper)                 0.8                 0.8      0.6
4    (Diaper)      (Milk)                 0.8                 0.8      0.6
5      (Milk)    (Diaper)                 0.8                 0.8      0.6
6    (Diaper)      (Beer)                 0.8                 0.6      0.6
7      (Beer)    (Diaper)                 0.6                 0.8      0.6

   confidence    lift  representativity  leverage  conviction  zhangs_metric  \
0        0.75  0.9375               1.0     -0.04         0.8          -0.25
1        0.75  0.9375               1.0     -0.04         0.8          -0.25
2        0.75  0.9375               1.0     -0.04         0.8          -0.25
3        0.75  0.9375               1.0     -0.04         0.8          -0.25
4        0.75  0.9375               1.0     -0.04         0.8          -0.25
5        0.75  0.9375               1.0     -0.04         0.8          -0.25
6        0.75  1.2500               1.0      0.12         1.6           1.00
7        1.00  1.2500               1.0      0.12         inf           0.50

   jaccard  certainty  kulczynski
0     0.60     -0.250       0.750
1     0.60     -0.250       0.750
2     0.60     -0.250       0.750
3     0.60     -0.250       0.750
4     0.60     -0.250       0.750
5     0.60     -0.250       0.750
6     0.75      0.375       0.875
7     0.75      1.000       0.875
```

```
Rule: {'Milk'} -> {'Bread'}
Support: 0.60
Confidence: 0.75
Lift: 0.94

Rule: {'Bread'} -> {'Milk'}
Support: 0.60
Confidence: 0.75
Lift: 0.94

Rule: {'Diaper'} -> {'Bread'}
Support: 0.60
Confidence: 0.75
Lift: 0.94

Rule: {'Bread'} -> {'Diaper'}
Support: 0.60
Confidence: 0.75
Lift: 0.94

Rule: {'Diaper'} -> {'Milk'}
Support: 0.60
Confidence: 0.75
Lift: 0.94

Rule: {'Milk'} -> {'Diaper'}
Support: 0.60
Confidence: 0.75
Lift: 0.94

Rule: {'Diaper'} -> {'Beer'}
Support: 0.60
Confidence: 0.75
Lift: 1.25

Rule: {'Beer'} -> {'Diaper'}
Support: 0.60
Confidence: 1.00
Lift: 1.25
```

Lab-3 Classification

Introduction:

Classification is a supervised machine learning technique used to predict a categorical label or class based on input data. In this lab, we applied classification techniques to predict whether a patient has diabetes based on health-related attributes using the Naive Bayes and Decision Tree algorithms. The model was trained on a dataset (`diabetes_data.csv`) and evaluated using accuracy, ROC AUC score, and confusion matrix to compare performance.

DataSets

diabetes_data.csv ×

1 to 50 of 768 entries Filter

Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31	0.248	26	1
10	115	0	0	0	35.3	0.134	29	0
2	197	70	45	543	30.5	0.158	53	1
8	125	96	0	0	0	0.232	54	1
4	110	92	0	0	37.6	0.191	30	0
10	168	74	0	0	38	0.537	34	1
10	139	80	0	0	27.1	1.441	57	0
1	189	60	23	846	30.1	0.398	59	1
5	166	72	19	175	25.8	0.587	51	1
7	100	0	0	0	30	0.484	32	1
0	118	84	47	230	45.8	0.551	31	1
7	107	74	0	0	29.6	0.254	31	1
1	103	30	38	83	43.3	0.183	33	0
1	115	70	30	96	34.6	0.529	32	1
3	126	88	41	235	39.3	0.704	27	0
8	99	84	0	0	35.4	0.388	50	0
7	196	90	0	0	39.8	0.451	41	1
9	119	80	35	0	29	0.263	29	1
2	90	68	42	0	38.2	0.503	27	1
4	111	72	47	207	37.1	1.39	56	1
3	180	64	25	70	34	0.271	26	0
7	133	84	0	0	40.2	0.696	37	0
7	106	92	18	0	22.7	0.235	48	0
9	171	110	24	240	45.4	0.721	54	1
7	159	64	0	0	27.4	0.294	40	0
0	180	66	39	0	42	1.893	25	1
1	146	56	0	0	29.7	0.564	29	0
2	71	70	27	0	28	0.586	22	0
7	103	66	32	0	39.1	0.344	31	1

3.1 Naive Bayes Classification

Implementation Code:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report

# Step 1: Load the dataset
df = pd.read_csv('diabetes.csv')
# Step 2: Split the data into features and target
X = df.drop(columns='Outcome')
y = df['Outcome']
# Step 3: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Step 4: Initialize the Naive Bayes classifier
nb_classifier = GaussianNB()
# Step 5: Train the model
nb_classifier.fit(X_train, y_train)
# Step 6: Make predictions
y_pred_nb = nb_classifier.predict(X_test)
# Step 7: Evaluate the model
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print("Sudha Lab")
print(f"Naive Bayes Accuracy: {accuracy_nb:.2f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred_nb))
```

Output SnapShot:

```
Sudha Lab
Naive Bayes Accuracy: 0.74

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.79      0.80       151
           1       0.62      0.66      0.64        80

    accuracy                           0.74       231
   macro avg       0.72      0.73      0.72       231
weighted avg       0.75      0.74      0.75       231
```

3.2 Decision Tree Classification

Implementation Code and Output

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
df = pd.read_csv('diabetes.csv')
X = df.drop(columns='Outcome')
y = df['Outcome']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
dt_classifier = DecisionTreeClassifier(criterion='entropy',random_state=42)

dt_classifier.fit(X_train, y_train)

y_pred_dt = dt_classifier.predict(X_test)

accuracy_dt = accuracy_score(y_test, y_pred_dt)
print("Sudha Lab")
print(f"Decision Tree Accuracy: {accuracy_dt:.2f}")
print("\nClassification Report:\n", classification_report(y_test,y_pred_dt))
```

```
Sudha Lab
Decision Tree Accuracy: 0.73

Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.78      0.79       151
           1       0.60      0.62      0.61        80

    accuracy                           0.73       231
   macro avg       0.70      0.70      0.70       231
weighted avg       0.73      0.73      0.73       231
```

3.3 Comparing Accuracy

Implementation Code

```python
from sklearn.metrics import confusion_matrix, roc_auc_score, accuracy_score
from sklearn.naive_bayes import GaussianNB

# Initialize and train the Naive Bayes classifier
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
# Make predictions with Naive Bayes
y_pred_nb = nb_classifier.predict(X_test)
# Calculate accuracy
accuracy_nb = accuracy_score(y_test, y_pred_nb)
# Calculate confusion matrices
conf_matrix_nb = confusion_matrix(y_test, y_pred_nb)
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
# Calculate ROC AUC scores
roc_auc_nb = roc_auc_score(y_test, y_pred_nb)
roc_auc_dt = roc_auc_score(y_test, y_pred_dt)

# Print comparison results
print("Sudha Lab")
print("\nNaive Bayes vs Decision Tree Classifier Performance:\n")
print(f"Naive Bayes Accuracy: {accuracy_nb:.2f}")
print(f"Decision Tree Accuracy: {accuracy_dt:.2f}")
print(f"Naive Bayes ROC AUC: {roc_auc_nb:.2f}")
print(f"Decision Tree ROC AUC: {roc_auc_dt:.2f}")
print("\nConfusion Matrix - Naive Bayes:\n", conf_matrix_nb)
print("\nConfusion Matrix - Decision Tree:\n", conf_matrix_dt)
```

Output SnapShot:

```
Sudha Lab

Naive Bayes vs Decision Tree Classifier Performance:

Naive Bayes Accuracy: 0.74
Decision Tree Accuracy: 0.73
Naive Bayes ROC AUC: 0.73
Decision Tree ROC AUC: 0.70

Confusion Matrix - Naive Bayes:
 [[119  32]
 [ 27  53]]

Confusion Matrix - Decision Tree:
 [[118  33]
 [ 30  50]]
```

Lab-4: Clustering

Introduction:

This lab focuses on the implementation and analysis of clustering algorithms, a key area of unsupervised machine learning. Clustering involves grouping data points so that those within the same cluster exhibit high similarity, while being distinct from those in other clusters. The algorithms explored include K-means, Mini-batch K-means, K-means++, K-Medoids, and Agglomerative Clustering.

DataSets

- Synthetic 2-D data points: Randomly generated 2-D data points within specified ranges (0-100 or 0-200) for K-means, Mini-batch K-means, and K-means++.
- Iris Dataset: A well-known multivariate dataset used for KMedoids and Agglomerative Clustering.

4.1 K-Means Clustering

Implementation Code and Output

```
import numpy as np
import time
from sklearn.cluster import KMeans
# Generate 10,000 2-D data points in the range 0-100
data = np.random.rand(10000, 2) * 100
# Initialize K-means algorithm with 5 clusters
kmeans = KMeans(n_clusters=5, random_state=42)
# Measure the time taken by the algorithm
start_time = time.time()
kmeans.fit(data)
end_time = time.time()
# Calculate the time taken
time_taken = end_time - start_time
print("Sudha Lab")
print(f"Time taken by K-means to find clusters: {time_taken:.4f} seconds")
# Get the cluster centers
print("Cluster Centers:\n", kmeans.cluster_centers_)
```

```
Sudha Lab
Time taken by K-means to find clusters: 0.0168 seconds
Cluster Centers:
 [[21.9602548  23.04318814]
 [78.08291821 22.44190359]
 [50.48467099 50.42790827]
 [78.06755446 78.35331707]
 [22.05517192 78.58995496]]
```

4.2 Mini-Batch K-means

Implementation Code and Output

```
import numpy as np
import time
from sklearn.cluster import MiniBatchKMeans
# Generate 10,000 2-D data points in the range 0-100
data = np.random.rand(10000, 2) * 100
batch_sizes = [100, 300, 500, 1000, 1500]
times = []
for batch_size in batch_sizes:
    # Initialize Mini-batch K-means algorithm
    minibatch_kmeans = MiniBatchKMeans(n_clusters=5, batch_size=batch_size, random_state=42)
    # Measure the time taken by the algorithm
    start_time = time.time()
    minibatch_kmeans.fit(data)
    end_time = time.time()
    # Calculate the time taken
    time_taken = end_time - start_time
    times.append(time_taken)
    print(f"Time taken with batch size {batch_size}: {time_taken:.4f} seconds")
# Determine the best batch size
best_batch_size = batch_sizes[np.argmin(times)]
print(f"\nBest batch size: {best_batch_size}")
```

```
Time taken with batch size 100: 0.0605 seconds
Time taken with batch size 300: 0.0222 seconds
Time taken with batch size 500: 0.0119 seconds
Time taken with batch size 1000: 0.0218 seconds
Time taken with batch size 1500: 0.0298 seconds

Best batch size: 500
```

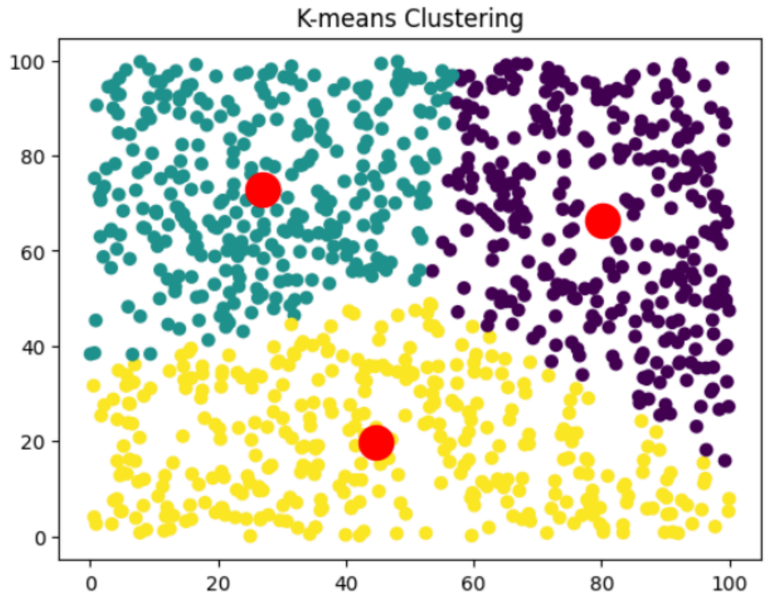
4.3 K-Means Clustering Algorithm

Implementation Code

```
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
print("Sudha lab")
# Generate 1,000 2-D data points in the range 0-100
data = np.random.rand(1000, 2) * 100
kmeans = KMeans(n_clusters=3, random_state=42)
# Fit the model
kmeans.fit(data)
# Plot the data points and cluster centers
plt.scatter(data[:, 0], data[:, 1], c=kmeans.labels_, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red')
plt.title("K-means Clustering")
plt.show()
```

Output SnapShot

Sudha lab



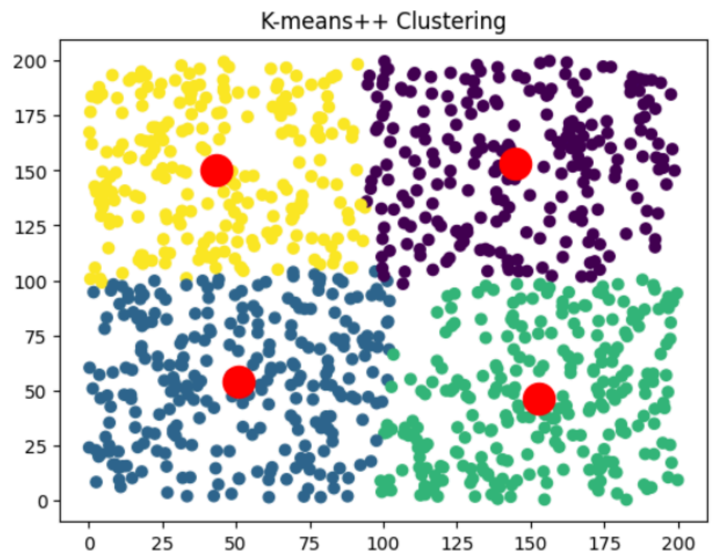
4.4 K-Means++ Algorithm

Implementation Code

```
import numpy as np
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
print("Sudha lab")
# Generate 1,000 2-D data points in the range 0-200
data = np.random.rand(1000, 2) * 200# Initialize K-means++ algorithm with 4 clusters
kmeans_plus = KMeans(n_clusters=4, init='k-means++', random_state=42)
# Fit the model
kmeans_plus.fit(data)
# Plot the data points and cluster centers
plt.scatter(data[:, 0], data[:, 1], c=kmeans_plus.labels_, cmap='viridis')
plt.scatter(kmeans_plus.cluster_centers_[:, 0], kmeans_plus.cluster_centers_[:, 1], s=300, c='red')
plt.title("K-means++ Clustering")
plt.show()
```

Output SnapShot

Sudha lab



4.5 Agglomerative Clustering Algorithm

Implementation Code

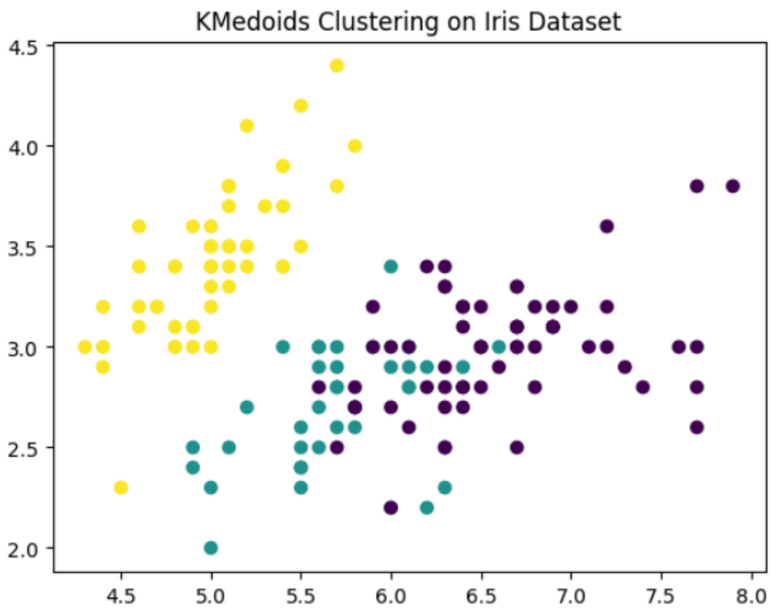
```
! pip install scikit-learn-extra
import pandas as pd
from sklearn_extra.cluster import KMedoids
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
print("Sudha Lab")
# Load the Iris dataset
iris = load_iris()
data = iris.data

# Initialize KMedoids algorithm with 3 clusters
kmedoids = KMedoids(n_clusters=3, random_state=42)

# Fit the model
kmedoids.fit(data)

# Plot the clusters
plt.scatter(data[:, 0], data[:, 1], c=kmedoids.labels_, cmap='viridis')
plt.title("KMedoids Clustering on Iris Dataset")
plt.show()
```

Output SnapShot



4.6 KMedoids Algorithm

Implementation Code

```
import pandas as pd
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
print("Sudha Lab")
# Load the Iris dataset
iris = load_iris()
data = iris.data

# Initialize Agglomerative Clustering algorithm with 3 clusters
agg_clustering = AgglomerativeClustering(n_clusters=3)

# Fit the model
labels = agg_clustering.fit_predict(data)

# Plot the clusters
plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')
plt.title("Agglomerative Clustering on Iris Dataset")
plt.show()
```

Output SnapShot

Sudha Lab