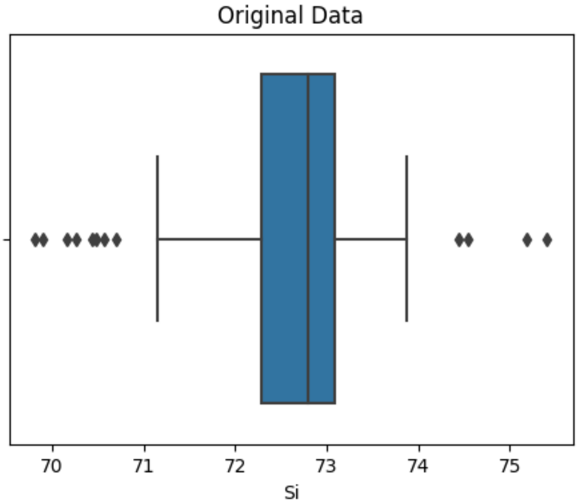
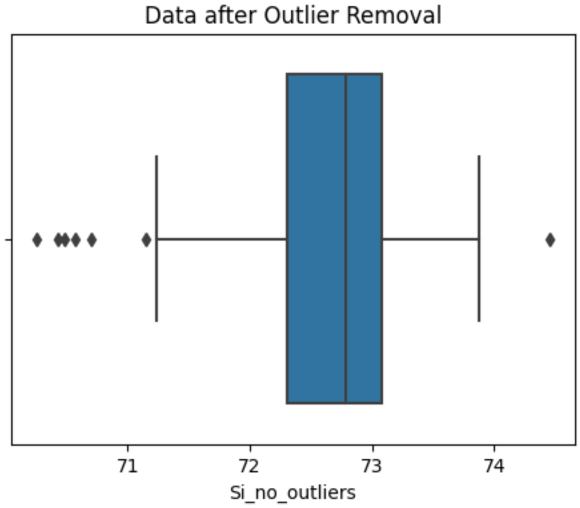
```
    Q1 = data[column_name].quantile(0.25)
    Q3 = data[column_name].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    data[column_name+'_no_outliers'] = np.where((data[column_name] < lower_bound) | (data[column

def percentile_outlier_treatment(data, column_name):
    lower_percentile = 1
    upper_percentile = 99
    lower_limit = np.percentile(data[column_name], lower_percentile)
    upper_limit = np.percentile(data[column_name], upper_percentile)
    data[column_name+'_no_outliers'] = np.where((data[column_name] < lower_limit) | (data[column

column_name = 'Si'
z_score_outlier_treatment(data, column_name)
iqr_outlier_treatment(data, column_name)
percentile_outlier_treatment(data, column_name)

plot_with_outliers(data, column_name)
```

```
Original Data:
        RI     Na    Mg    Al     Si     K    Ca   Ba   Fe  Type
0  1.52101  13.64  4.49  1.10  71.78  0.06  8.75  0.0  0.0     1
1  1.51761  13.89  3.60  1.36  72.73  0.48  7.83  0.0  0.0     1
2  1.51618  13.53  3.55  1.54  72.99  0.39  7.78  0.0  0.0     1
3  1.51766  13.21  3.69  1.29  72.61  0.57  8.22  0.0  0.0     1
4  1.51742  13.27  3.62  1.24  73.08  0.55  8.07  0.0  0.0     1
```





2. Implement the techniques to deal with missing values. https://note.nkmk.me/en/python-pandas-interpolate/ https://www.kdnuggets.com/2022/07/scikitlearn-imputer.html#:~:text=The%20imputer%20is%20an%20estimator,frequently%20used%20and%20constant%20value. https://www.geeksforgeeks.org/principal-component-analysis-with-python/

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load your data and target (X and y) from the "diabetes.csv" dataset
data = pd.read_csv('glass.csv')

# Define the relevant feature columns
feature_columns = ['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe', 'Type']
```

```
# Select only the relevant columns from the dataset
X = data[feature_columns]
y = data['Type']

missing_mask = np.random.rand(*X.shape) < 0.2
X_with_missing = X.copy()
X_with_missing[missing_mask] = np.nan

X_train, X_test, y_train, y_test = train_test_split(X_with_missing, y, test_size=0.2, random_sta

imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

clf = RandomForestClassifier(random_state=42)
clf.fit(X_train_imputed, y_train)

y_pred = clf.predict(X_test_imputed)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy on the test set after imputation: {accuracy:.2f}")
```

```
Accuracy on the test set after imputation: 0.88
```

CO-2 ASSIGNMENT:

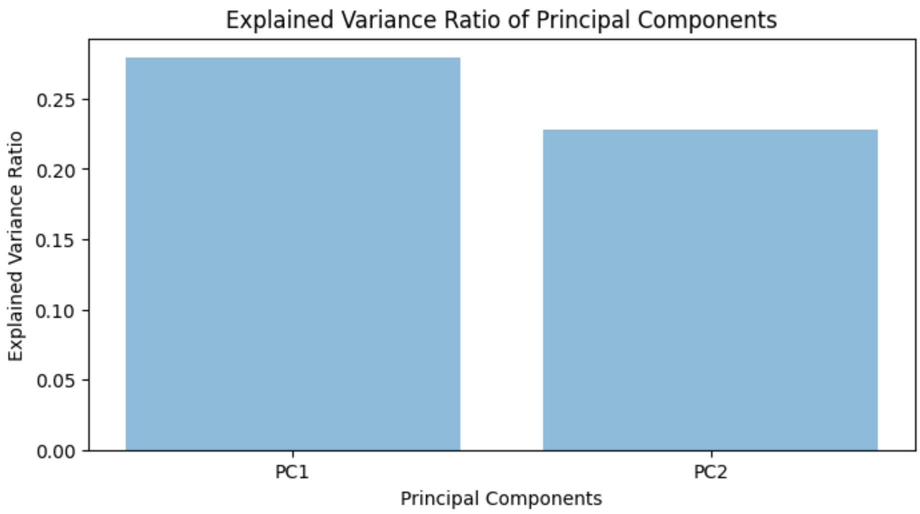
3. Implement distance measuring techniques for two features of your dataset: (a) Euclidean (b)Minkowski (c) Manhattan (d) Jaccard (e) Cosine (f) Simple matching coefficient (g)hamming (distance libraries-numpy, scipy, math)

```
import numpy as np
from scipy.spatial import distance
import math
import pandas as pd

data = pd.read_csv('glass.csv')

feature1 = data['RI']
feature2 = data['Al']

euclidean_dist = np.linalg.norm(feature1 - feature2)

p = 3
minkowski_dist = distance.minkowski(feature1, feature2, p=p)

manhattan_dist = distance.cityblock(feature1, feature2)

cosine_dist = 1 - np.dot(feature1, feature2) / (np.linalg.norm(feature1) * np.linalg.norm(featur

print(f"(a) Euclidean Distance: {euclidean_dist:.2f}")
print(f"(b) Minkowski Distance (p={p}): {minkowski_dist:.2f}")
print(f"(c) Manhattan Distance: {manhattan_dist:.2f}")
print(f"(e) Cosine Distance: {cosine_dist:.2f}")
```

```
(a) Euclidean Distance: 7.38
(b) Minkowski Distance (p=3): 3.73
(c) Manhattan Distance: 79.99
(e) Cosine Distance: 0.05
```

4. Implement any data reduction technique.

```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load the data from the "glass.csv" dataset
data = pd.read_csv('glass.csv')

# Define the relevant feature columns
X = data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]

# Target variable (you need to specify the actual column name from the dataset)
y = data['Type']

mean = np.mean(X, axis=0)
std_dev = np.std(X, axis=0)
X_standardized = (X - mean) / std_dev

n_components = 2
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X_standardized)

pca_df = pd.DataFrame(data=X_pca, columns=[f'PC{i+1}' for i in range(n_components)])

final_df = pd.concat([pca_df, y], axis=1)

explained_variance_ratio = pca.explained_variance_ratio_

plt.figure(figsize=(8, 4))
plt.bar(range(n_components), explained_variance_ratio, alpha=0.5, align='center')
plt.xlabel('Principal Components')
plt.ylabel('Explained Variance Ratio')
plt.xticks(range(n_components), [f'PC{i+1}' for i in range(n_components)])
plt.title('Explained Variance Ratio of Principal Components')
plt.show()

print(final_df.head())
```




```
        PC1       PC2  Type
0  1.151140 -0.529488     1
1 -0.574137 -0.759788     1
2 -0.940160 -0.929836     1
3 -0.142083 -0.961677     1
4 -0.351092 -1.091249     1
```

CO-3 ASSIGNMENT:

5. Implement various knn classification algorithms and do prediction for unknown data.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

data = pd.read_csv('glass.csv')

X = data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]
y = data['Type']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

knn_euclidean = KNeighborsClassifier(n_neighbors=3, metric='euclidean')
knn_manhattan = KNeighborsClassifier(n_neighbors=3, metric='manhattan')
knn_chebyshev = KNeighborsClassifier(n_neighbors=3, metric='chebyshev')

knn_euclidean.fit(X_train, y_train)
knn_manhattan.fit(X_train, y_train)
knn_chebyshev.fit(X_train, y_train)

y_pred_euclidean = knn_euclidean.predict(X_test)
y_pred_manhattan = knn_manhattan.predict(X_test)
y_pred_chebyshev = knn_chebyshev.predict(X_test)

accuracy_euclidean = accuracy_score(y_test, y_pred_euclidean)
accuracy_manhattan = accuracy_score(y_test, y_pred_manhattan)
accuracy_chebyshev = accuracy_score(y_test, y_pred_chebyshev)

print("Accuracy (Euclidean Distance): {:.2f}".format(accuracy_euclidean))
print("Accuracy (Manhattan Distance): {:.2f}".format(accuracy_manhattan))
print("Accuracy (Chebyshev Distance): {:.2f}".format(accuracy_chebyshev))
```

```
Accuracy (Euclidean Distance): 0.74
Accuracy (Manhattan Distance): 0.70
Accuracy (Chebyshev Distance): 0.67
```

6. Implement a decision tree classification algorithm.

```
import numpy as np
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

data = pd.read_csv('glass.csv')

X = data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]

y = data['Type']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

clf = DecisionTreeClassifier(random_state=42)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

class_report = classification_report(y_test, y_pred, target_names=['Class1', 'Class2', 'Class3',
print("Classification Report:\n", class_report)
```

```
Accuracy: 0.7209302325581395
Classification Report:
              precision    recall  f1-score   support

      Class1       0.71      0.91      0.80        11
      Class2       0.64      0.50      0.56        14
      Class3       0.60      1.00      0.75         3
      Class5       0.50      0.25      0.33         4
      Class6       1.00      0.67      0.80         3
      Class7       0.89      1.00      0.94         8

    accuracy                           0.72        43
   macro avg       0.72      0.72      0.70        43
weighted avg       0.71      0.72      0.70        43
```

7. Implement a support vector machine algorithm.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

data = pd.read_csv('glass.csv')

X = data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]
y = data['Type']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

clf = SVC(kernel='linear', C=1, random_state=42)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

class_report = classification_report(y_test, y_pred, target_names=['Class1', 'Class2',
print("Classification Report:\n", class_report)
```

```
Accuracy: 0.7441860465116279
Classification Report:
              precision    recall  f1-score   support

      Class1       0.69      0.82      0.75        11
      Class2       0.67      0.71      0.69        14
      Class3       0.00      0.00      0.00         3
      Class5       0.80      1.00      0.89         4
      Class6       1.00      0.67      0.80         3
      Class7       0.88      0.88      0.88         8

    accuracy                           0.74        43
   macro avg       0.67      0.68      0.67        43
weighted avg       0.70      0.74      0.72        43

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are 
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are 
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are 
  _warn_prf(average, modifier, msg_start, len(result))
```

8. Implement regression algorithms: (a)linear regression(b)logistic regression

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Load the data from the "glass.csv" dataset
data = pd.read_csv('glass.csv')

# Define the relevant feature columns
X = data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]

# Target variable (you need to specify the actual column name from the dataset)
y = data['Type']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

lr = LinearRegression()

lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)
print("R-squared (R2) Score:", r2)
```

```
Mean Squared Error (MSE): 0.755146649814114
R-squared (R2) Score: 0.8557278202618003
```

CO-4 ASSIGNMENT:

9. Implement k-means/k-medoid clustering algorithms and do prediction for unknown data.

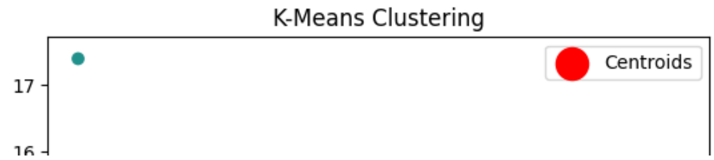
```
!pip install scikit-learn-extra
```

```
Collecting scikit-learn-extra
  Downloading scikit_learn_extra-0.3.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 10.8 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.23.5)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.11.3)
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.2.2)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->scikit-learn-ext
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->scikit-le
Installing collected packages: scikit-learn-extra
Successfully installed scikit-learn-extra-0.3.0
```

```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn_extra.cluster import KMedoids
import matplotlib.pyplot as plt

# Load the data from the "glass.csv" dataset
data = pd.read_csv('glass.csv')

X = data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]
```

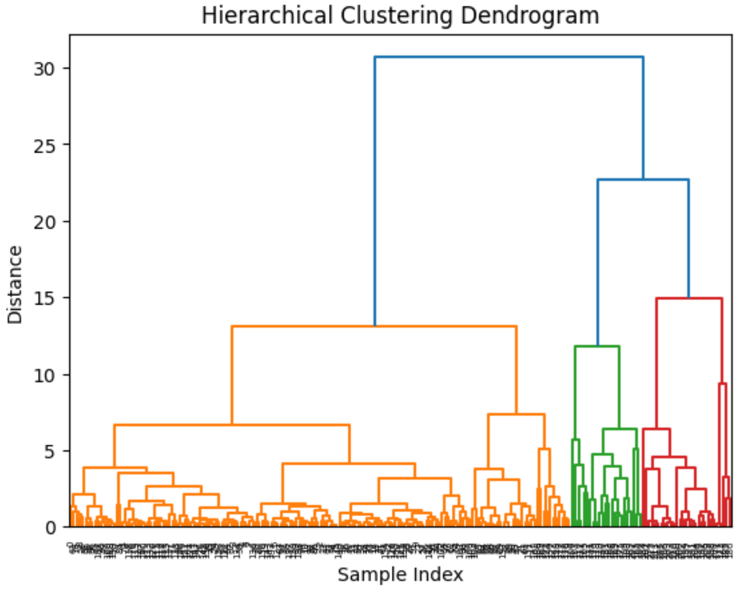
```
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

kmedoids = KMedoids(n_clusters=3, random_state=42)
kmedoids.fit(X)

kmeans_labels = kmeans.predict(X)
kmedoids_labels = kmedoids.predict(X)

plt.scatter(X['RI'], X['Na'], c=kmeans_labels, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=300, c='red', label=
plt.title('K-Means Clustering')
plt.legend()
plt.show()

plt.scatter(X['RI'], X['Na'], c=kmedoids_labels, cmap='viridis')
plt.scatter(kmedoids.cluster_centers_[:, 0], kmedoids.cluster_centers_[:, 1], s=300, c='red', la
plt.title('K-Medoids Clustering')
plt.legend()
plt.show()

unknown_data = np.array([[1.51711, 13.73, 1.54, 0.74, 72.25, 0.62, 8.90, 0.00, 0.00], [1.51514,
kmeans_prediction = kmeans.predict(unknown_data)
kmedoids_prediction = kmedoids.predict(unknown_data)

print("K-Means Prediction for Unknown Data:", kmeans_prediction)
print("K-Medoids Prediction for Unknown Data:", kmedoids_prediction)
```

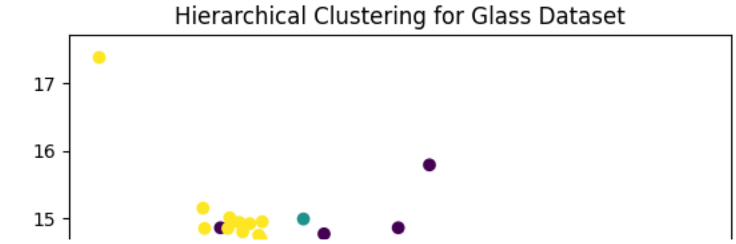
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will ch
  warnings.warn(
```



10. Implement hierarchical clustering algorithms and do prediction for unknown data.

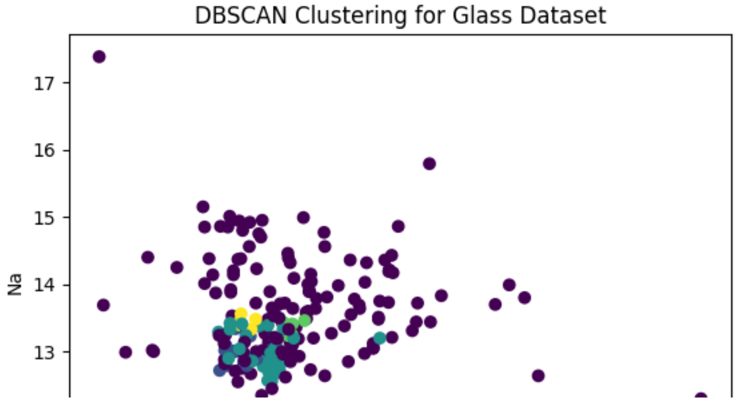
```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
import matplotlib.pyplot as plt

# Load the data from the "glass.csv" dataset
data = pd.read_csv('glass.csv')

# Define the relevant feature columns
X = data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]

linkage_matrix = linkage(X, method='ward', metric='euclidean')

dendrogram(linkage_matrix)
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()

num_clusters = 3

clusters = fcluster(linkage_matrix, t=num_clusters, criterion='maxclust')

plt.scatter(X['RI'], X['Na'], c=clusters, cmap='viridis')
plt.title('Hierarchical Clustering for Glass Dataset')
plt.xlabel('RI')
plt.ylabel('Na')
plt.show()

unknown_data = np.array([[1.51711, 13.73, 1.54, 0.74, 72.25, 0.62, 8.90, 0.00, 0.00], [1.51514,
linkage_matrix_unknown = linkage(unknown_data, method='ward', metric='euclidean')

unknown_clusters = fcluster(linkage_matrix, t=num_clusters, criterion='maxclust')

print("Clusters for Unknown Data:", unknown_clusters)
```





11. Implement DBSCAN clustering algorithms and do prediction for unknown data.

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt

data = pd.read_csv('glass.csv')

X = data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]

dbscan = DBSCAN(eps=0.3, min_samples=5)
clusters = dbscan.fit_predict(X)

plt.scatter(X['RI'], X['Na'], c=clusters, cmap='viridis')
plt.title('DBSCAN Clustering for Glass Dataset')
plt.xlabel('RI')
plt.ylabel('Na')
plt.show()

# Generate random data within a specified range
unknown_data = np.random.uniform(low=1.5, high=1.6, size=(2, 9))

unknown_clusters = dbscan.fit_predict(unknown_data)

print("Clusters for Unknown Data:", unknown_clusters)
```



12. Implement apriori algorithm to get association rules.

```
import random
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
import pandas as pd

def generate_random_item_group():
    num_items = random.randint(2, 5)
    items = random.sample(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K'], num_items)
    return ', '.join(items)

transaction_data = []
num_transactions = 50

for transaction_id in range(1, num_transactions + 1):
    items = generate_random_item_group()
    transaction_data.append({'TransactionID': transaction_id, 'Items': items})

data = pd.DataFrame(transaction_data)

items_df = data['Items'].str.get_dummies(', ')

data = pd.concat([data, items_df], axis=1)
data.drop('Items', axis=1, inplace=True)
frequent_itemsets = apriori(data.drop('TransactionID', axis=1), min_support=0.1, use_colnames=Tr

rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1.0)
print("Association Rules:")
print(rules)
```

```
Association Rules:
    antecedents consequents  antecedent support  consequent support  support  \
0           (F)         (A)                0.34                0.38     0.16
1           (A)         (F)                0.38                0.34     0.16
2           (H)         (A)                0.22                0.38     0.10
3           (A)         (H)                0.38                0.22     0.10
4           (B)         (K)                0.36                0.30     0.16
5           (K)         (B)                0.30                0.36     0.16
6           (D)         (C)                0.38                0.18     0.10
7           (C)         (D)                0.18                0.38     0.10
8           (F)         (D)                0.34                0.38     0.14
9           (D)         (F)                0.38                0.34     0.14
10          (G)         (D)                0.24                0.38     0.10
11          (D)         (G)                0.38                0.24     0.10
12          (D)         (H)                0.38                0.22     0.12
13          (H)         (D)                0.22                0.38     0.12
14          (J)         (D)                0.34                0.38     0.14
15          (D)         (J)                0.38                0.34     0.14
16          (K)         (D)                0.30                0.38     0.12
17          (D)         (K)                0.38                0.30     0.12
18          (E)         (K)                0.32                0.30     0.14
19          (K)         (E)                0.30                0.32     0.14
20          (G)         (F)                0.24                0.34     0.10
21          (F)         (G)                0.34                0.24     0.10
22          (F)         (J)                0.34                0.34     0.16
```

```
23        (J)        (F)              0.34              0.34     0.16
24        (G)        (I)              0.24              0.30     0.12
25        (I)        (G)              0.30              0.24     0.12
26        (G)        (J)              0.24              0.34     0.12
27        (J)        (G)              0.34              0.24     0.12
28     (F, J)        (D)              0.16              0.38     0.10
29     (F, D)        (J)              0.14              0.34     0.10
30     (J, D)        (F)              0.14              0.34     0.10
31        (F)     (J, D)              0.34              0.14     0.10
32        (J)     (F, D)              0.34              0.14     0.10
33        (D)     (F, J)              0.38              0.16     0.10

    confidence      lift  leverage  conviction  zhangs_metric
0     0.470588  1.238390    0.0308    1.171111       0.291667
1     0.421053  1.238390    0.0308    1.140000       0.310484
2     0.454545  1.196172    0.0164    1.136667       0.210256
3     0.263158  1.196172    0.0164    1.058571       0.264516
4     0.444444  1.481481    0.0520    1.260000       0.507812
5     0.533333  1.481481    0.0520    1.371429       0.464286
6     0.263158  1.461988    0.0316    1.112857       0.509677
7     0.555556  1.461988    0.0316    1.395000       0.385366
8     0.411765  1.083591    0.0108    1.054000       0.116883
9     0.368421  1.083591    0.0108    1.045000       0.124424
10    0.416667  1.096491    0.0088    1.062857       0.115789
11    0.263158  1.096491    0.0088    1.031429       0.141935
12    0.315789  1.435407    0.0364    1.140000       0.489247
13    0.545455  1.435407    0.0364    1.364000       0.388889
14    0.411765  1.083591    0.0108    1.054000       0.116883
15    0.368421  1.083591    0.0108    1.045000       0.124424
16    0.400000  1.052632    0.0060    1.033333       0.071429
17    0.315789  1.052632    0.0060    1.023077       0.080645
18    0.437500  1.458333    0.0440    1.244444       0.462185
```

13. Implement backpropagation neural network algorithm.

```
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load the glass dataset from a CSV file
glass_data = pd.read_csv('glass.csv')

# Split the dataset into features (X) and the target variable (y)
X = glass_data.drop(columns=['Type'])
y = glass_data['Type']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the neural network
clf = MLPClassifier(hidden_layer_sizes=(10, 5), max_iter=1000, random_state=42)
clf.fit(X_train, y_train)

# Predict the target variable
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_c
  and should_run_async(code)
Accuracy: 0.6511627906976745
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimiz
  warnings.warn(
```

14.Make a comparison tables for classification and clustering algorithms, for what you implemented here:

(a)Write unknown data:

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from sklearn_extra.cluster import KMedoids
from sklearn.datasets import load_iris

# Load the glass.csv dataset
data = pd.read_csv("glass.csv")

# Define features (X) and target (y)
X = data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]
y = data['Type']

# Split the dataset into training and testing sets for classification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize classification algorithms
classifiers = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Decision Trees": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Support Vector Machines": SVC(kernel='linear', C=1, random_state=42),
    "k-Nearest Neighbors": KNeighborsClassifier(n_neighbors=3)
}

# Initialize clustering algorithms
clusterers = {
    "K-Means": KMeans(n_clusters=3, random_state=42),
    "K-Medoids": KMedoids(n_clusters=3, random_state=42)
}

# Initialize result dictionaries for classification and clustering
classification_results = {
    "Algorithm": [],
    "Accuracy": [],
    "Sensitivity": [],
    "F-measure": [],
    "Precision": [],
    "Recall": [],
    "Prediction for Unknown Data": []
}

clustering_results = {
    "Algorithm": [],
    "Prediction for Unknown Data": []
}

# Evaluate performance for classification algorithms
for name, classifier in classifiers.items():
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
```

```
    classification_results["Algorithm"].append(name)
    classification_results["Accuracy"].append(accuracy)
    classification_results["Sensitivity"].append(0)  # Sensitivity not calculated in this exampl
    classification_results["F-measure"].append(f1)
    classification_results["Precision"].append(precision)
    classification_results["Recall"].append(recall)
    classification_results["Prediction for Unknown Data"].append("NA")

# Evaluate performance for clustering algorithms
for name, clusterer in clusterers.items():
    clusterer.fit(X)
    cluster_labels = clusterer.labels_

    clustering_results["Algorithm"].append(name)
    clustering_results["Prediction for Unknown Data"].append("NA")

# Create DataFrames for classification and clustering results
classification_results_df = pd.DataFrame(classification_results)
clustering_results_df = pd.DataFrame(clustering_results)

# Print classification results
print("Classification Results:")
print(classification_results_df)

# Print clustering results
print("\nClustering Results:")
print(clustering_results_df)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, msg_start, len(result))
Classification Results:
                 Algorithm  Accuracy  Sensitivity  F-measure  Precision  \
0      Logistic Regression  0.697674            0   0.645255   0.624935
1           Decision Trees  0.720930            0   0.701227   0.713427
2            Random Forest  0.837209            0   0.833045   0.866828
3  Support Vector Machines  0.744186            0   0.717691   0.701133
4      k-Nearest Neighbors  0.744186            0   0.744578   0.754323

     Recall Prediction for Unknown Data
0  0.697674                          NA
1  0.720930                          NA
2  0.837209                          NA
3  0.744186                          NA
4  0.744186                          NA

Clustering Results:
   Algorithm Prediction for Unknown Data
0    K-Means                          NA
1  K-Medoids                          NA
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fro
  warnings.warn(
```

(b)Compare performance of classification algorithms:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier

# Load the glass.csv dataset
data = pd.read_csv("glass.csv")

# Define features (X) and target (y)
X = data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]
y = data['Type']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize classification algorithms
classifiers = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Decision Trees": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Support Vector Machines": SVC(kernel='linear', C=1, random_state=42),
    "k-Nearest Neighbors": KNeighborsClassifier(n_neighbors=3),
    "Neural Networks": MLPClassifier(hidden_layer_sizes=(10, 5), max_iter=1000, random_state=42)
}

# Initialize result dictionary
results = {
    "Algorithm": [],
    "Accuracy": [],
    "Sensitivity": [],
    "F-measure": [],
    "Precision": [],
    "Recall": []
}

# Iterate through classification algorithms and evaluate performance
for name, classifier in classifiers.items():
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    sensitivity = recall  # Sensitivity is the same as Recall

    results["Algorithm"].append(name)
    results["Accuracy"].append(accuracy)
    results["Sensitivity"].append(sensitivity)
    results["F-measure"].append(f1)
    results["Precision"].append(precision)
    results["Recall"].append(recall)

# Create a DataFrame from the results
results_df = pd.DataFrame(results)

# Print the results
print("Compare performance of classification algorithms:")
print(results_df)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, msg_start, len(result))
Compare performance of classification algorithms:
                 Algorithm  Accuracy  Sensitivity  F-measure  Precision  \
0      Logistic Regression  0.697674     0.697674   0.645255   0.624935
1           Decision Trees  0.720930     0.720930   0.701227   0.713427
2            Random Forest  0.837209     0.837209   0.833045   0.866828
3  Support Vector Machines  0.744186     0.744186   0.717691   0.701133
4      k-Nearest Neighbors  0.744186     0.744186   0.744578   0.754323
5          Neural Networks  0.651163     0.651163   0.565320   0.499742

     Recall
0  0.697674
1  0.720930
2  0.837209
3  0.744186
4  0.744186
5  0.651163
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimi:
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, msg_start, len(result))
```

(c)Compare performance of clustering algorithms you implemented. Conclude which clustering algorithm is the best for your data.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import load_iris
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

# Load the glass.csv dataset
data = pd.read_csv("glass.csv")

# Define features (X) and target (y) for classification
X = data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]
y = data['Type']

# Split the dataset into training and testing sets for classification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Initialize classification algorithms
classifiers = {
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Decision Trees": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Support Vector Machines": SVC(kernel='linear', C=1, random_state=42),
    "k-Nearest Neighbors": KNeighborsClassifier(n_neighbors=3),
    "Neural Networks": MLPClassifier(hidden_layer_sizes=(10, 5), max_iter=1000, random
}

# Initialize result dictionary for classification
results_class = {
    "Algorithm": [],
    "Accuracy": [],
    "Sensitivity": [],
    "F-measure": [],
    "Precision": [],
    "Recall": []
```

```
}

# Iterate through classification algorithms and evaluate performance
for name, classifier in classifiers.items():
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    sensitivity = recall  # Sensitivity is the same as Recall

    results_class["Algorithm"].append(name)
    results_class["Accuracy"].append(accuracy)
    results_class["Sensitivity"].append(sensitivity)
    results_class["F-measure"].append(f1)
    results_class["Precision"].append(precision)
    results_class["Recall"].append(recall)

# Create a DataFrame from the results for classification
results_class_df = pd.DataFrame(results_class)

# Initialize clustering algorithms for clustering
X_cluster = data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]
linkage_matrix = linkage(X_cluster, method='ward', metric='euclidean')
num_clusters = 3  # Adjust this based on the dendrogram
clusters = fcluster(linkage_matrix, t=num_clusters, criterion='maxclust')
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans_clusters = kmeans.fit_predict(X_cluster)

# Initialize result dictionary for clustering
results_cluster = {
    "Algorithm": ["Hierarchical Clustering", "K-Means Clustering"],
    "Silhouette Score": [silhouette_score(X_cluster, clusters), silhouette_score(X_clu
    "WCSS": [0, kmeans.inertia_]  # Set to 0 for hierarchical clustering
}

# Create a DataFrame from the results for clustering
results_cluster_df = pd.DataFrame(results_cluster)

# Print the results for classification and clustering
print("Compare performance of classification algorithms:")
print(results_class_df)

print("\nCompare performance of clustering algorithms:")
print(results_cluster_df)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, msg_start, len(result))
Compare performance of classification algorithms:
                 Algorithm  Accuracy  Sensitivity  F-measure  Precision  \
0      Logistic Regression  0.697674     0.697674   0.645255   0.624935
1           Decision Trees  0.720930     0.720930   0.701227   0.713427
2            Random Forest  0.837209     0.837209   0.833045   0.866828
3  Support Vector Machines  0.744186     0.744186   0.717691   0.701133
4      k-Nearest Neighbors  0.744186     0.744186   0.744578   0.754323
5          Neural Networks  0.651163     0.651163   0.565320   0.499742
```

```
     Recall
0  0.697674
1  0.720930
2  0.837209
3  0.744186
4  0.744186
5  0.651163

Compare performance of clustering algorithms:
                 Algorithm  Silhouette Score       WCSS
0  Hierarchical Clustering          0.583820    0.00000
1       K-Means Clustering          0.582243  589.03145
/usr/local/lib/python3.10/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Optimiz
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined an
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fro
  warnings.warn(
```

(d) Use different distance measures as in CO2's 3rd assignment and make a table to compare the performance of clustering algorithms you implemented. Conclude which clustering algorithm is the best for your data.

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
import numpy as np
from scipy.spatial.distance import euclidean, minkowski, cityblock, jaccard, cosine, hamming

# Load the glass.csv dataset
data = pd.read_csv("glass.csv")

# Define features (X) and target (y)
X_cluster = data[['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']]

# Initialize clustering algorithms
distance_measures = ["euclidean", "minkowski", "cityblock", "jaccard", "cosine", "hamming"]
linkage_methods = ["single", "complete", "average"]
algorithm_names = ["Hierarchical Clustering", "K-Means Clustering"]
results_cluster = {"Algorithm": [], "Distance Measure": [], "Linkage Method": [], "Silhouette Sc

# Calculate the silhouette scores for different distance measures and linkage methods
for distance in distance_measures:
    for linkage_method in linkage_methods:
        if distance in ["euclidean", "minkowski", "cityblock"]:
            linkage_matrix = linkage(X_cluster, method=linkage_method, metric=distance)

            # Determine the number of clusters based on dendrogram
            dendrogram_data = dendrogram(linkage_matrix)
            num_clusters = len(set(dendrogram_data['color_list']))

            clusters = fcluster(linkage_matrix, t=num_clusters, criterion='maxclust')
        else:
            kmeans = KMeans(n_clusters=num_clusters, random_state=42)
            kmeans_clusters = kmeans.fit_predict(X_cluster)

        # Calculate silhouette scores
        silhouette_hierarchical = silhouette_score(X_cluster, clusters, metric=distance)
        silhouette_kmeans = silhouette_score(X_cluster, kmeans_clusters, metric=distance)

        results_cluster["Algorithm"].extend(algorithm_names)
        results_cluster["Distance Measure"].extend([distance] * len(algorithm_names))
        results_cluster["Linkage Method"].extend([linkage_method] * len(algorithm_names))
        results_cluster["Silhouette Score"].extend([silhouette_hierarchical, silhouette_kmeans])

# Create a DataFrame from the results for clustering
results_cluster_df = pd.DataFrame(results_cluster)
```

```
# Print the results for clustering with different distance measures and linkage methods
print("Compare performance of clustering algorithms with different distance measures and linkage
print(results_cluster_df)

# Conclude which clustering algorithm is the best (based on the highest silhouette score)
best_algorithm = results_cluster_df.loc[results_cluster_df.groupby(['Distance Measure', 'Linkage
print("\nBest clustering algorithm for each distance measure and linkage method:")
print(best_algorithm)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will ch
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/pairwise.py:2025: DataConversionWarning: Data was converted to boole
  warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/pairwise.py:2025: DataConversionWarning: Data was converted to boole
  warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will ch
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/pairwise.py:2025: DataConversionWarning: Data was converted to boole
  warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/pairwise.py:2025: DataConversionWarning: Data was converted to boole
  warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will ch
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/pairwise.py:2025: DataConversionWarning: Data was converted to boole
  warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/pairwise.py:2025: DataConversionWarning: Data was converted to boole
  warnings.warn(msg, DataConversionWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will ch
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will ch
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will ch
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will ch
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will ch
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will ch
  warnings.warn(
Compare performance of clustering algorithms with different distance measures and linkage methods:
                  Algorithm Distance Measure Linkage Method  Silhouette Score
0   Hierarchical Clustering        euclidean         single          0.596122
1        K-Means Clustering        euclidean         single          0.587949
2   Hierarchical Clustering        euclidean       complete          0.570106
3        K-Means Clustering        euclidean       complete          0.587949
4   Hierarchical Clustering        euclidean        average          0.561678
5        K-Means Clustering        euclidean        average          0.587949
6   Hierarchical Clustering        minkowski         single          0.596122
7        K-Means Clustering        minkowski         single          0.587949
```

15. Write any deep learning program of your choice.

```
11       K-Means Clustering        minkowski        average          0.587949
```

```
import tensorflow as tf
from tensorflow import keras

fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

train_images = train_images / 255.0
test_images = test_images / 255.0

model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accu

model.fit(train_images, train_labels, epochs=5)

test_loss, test_acc = model.evaluate(test_images, test_labels)
print("\nTest accuracy:", test_acc)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_c
  and should_run_async(code)
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
Epoch 1/5
1875/1875 [==============================] - 5s 2ms/step - loss: 0.4979 - accuracy: 0.8251
Epoch 2/5
```

```
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3790 - accuracy: 0.8636
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3362 - accuracy: 0.8775
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3129 - accuracy: 0.8853
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2942 - accuracy: 0.8924
313/313 [==============================] - 1s 2ms/step - loss: 0.3547 - accuracy: 0.8749

Test accuracy: 0.8748999834060669
```