- Name: Pratham Ariwala
- Enrollment No: 200090107112
- Div: A
- Group Members: Aayush Purswani - 200090107026, Pratham Ariwala - 200090107112
- Dataset Name: Heart Attack Analysis
- Dataset Link: https://www.kaggle.com/datasets/rashikrahmanpritom/heart-attack-analysis-prediction-dataset/data

```
!pip install seaborn matplotlib pandas numpy scipy scikit-learn-extra
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `trar
  and should_run_async(code)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.12.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (1.11.3)
Requirement already satisfied: scikit-learn-extra in /usr/local/lib/python3.10/dist-packages (0.3.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.1.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.43.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.3.post1)
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.2
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib) (1
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->scikit-le
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->sc
```

```
import warnings
warnings.filterwarnings('ignore')
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `trar
  and should_run_async(code)
```

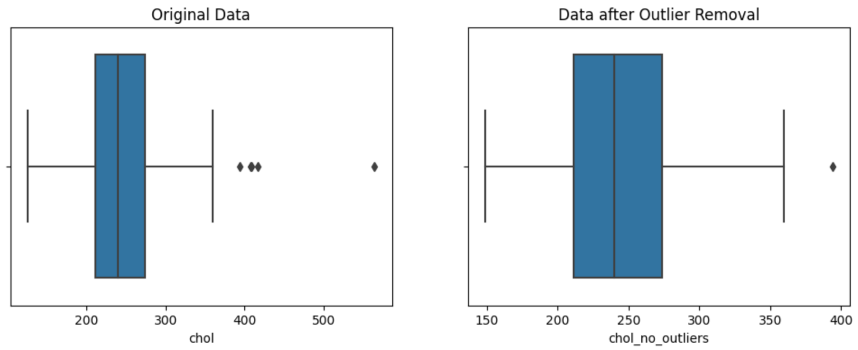
CO-1 ASSIGNMENT:

1. Implement the techniques to deal with outliers. - https://www.analyticsvidhya.com/blog/2021/05/feature-engineering-how-to-detect-and-remove-outliers-with-python-code/

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from scipy import stats

# Load the data from a local CSV file (replace 'your_file_path.csv' with the actual file path)
data = pd.read_csv('heart.csv')

print("Original Data:")
print(data.head())

def plot_with_outliers(data, column_name):
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 2, 1)
    sns.boxplot(x=data[column_name])
    plt.title("Original Data")
    plt.subplot(1, 2, 2)
    sns.boxplot(x=data[column_name+'_no_outliers'])
    plt.title("Data after Outlier Removal")
    plt.show()

def z_score_outlier_treatment(data, column_name):
    z_scores = np.abs(stats.zscore(data[column_name]))
    threshold = 3
    data[column_name+'_no_outliers'] = np.where(np.abs(z_scores) > threshold, np.nan, data[column_name])
```

```
def iqr_outlier_treatment(data, column_name):
    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_name] > upper_bound), np.nan, dat

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_name] > upper_limit), np.nan, dat

column_name = 'chol'
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:
   age  sex  cp  trtbps  chol  fbs  restecg  thalachh  exng  oldpeak  slp  \
0   63    1   3     145   233    1        0       150     0      2.3    0
1   37    1   2     130   250    0        1       187     0      3.5    0
2   41    0   1     130   204    0        0       172     0      1.4    2
3   56    1   1     120   236    0        1       178     0      0.8    2
4   57    0   0     120   354    0        1       163     1      0.6    2

   caa  thall  output
0    0      1       1
1    0      2       1
2    0      2       1
3    0      2       1
4    0      2       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('heart.csv')

# Define the relevant feature columns
feature_columns = ['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng', 'oldpeak', 'slp', 'caa', 'thall']
```

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

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_state=42)

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: 0.89
```

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)

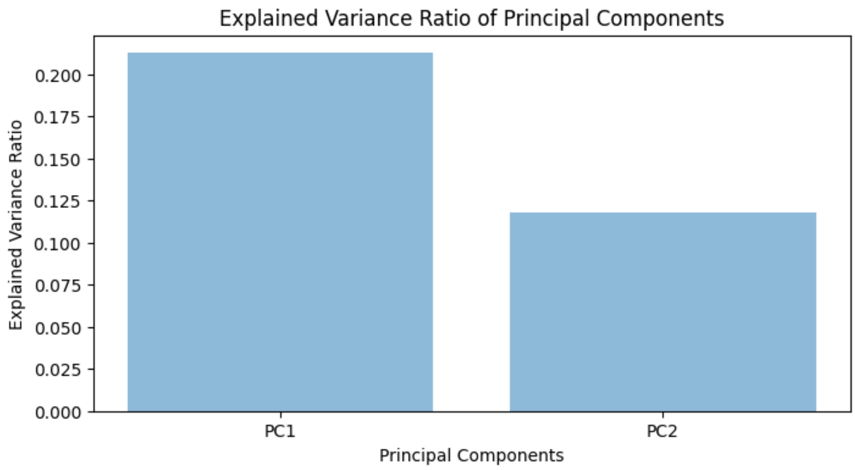
```
import numpy as np
from scipy.spatial import distance
import math
import pandas as pd

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

feature1 = data['trtbps']
feature2 = data['chol']

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(feature2))

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: 2195.16
(b) Minkowski Distance (p=3): 926.05
(c) Manhattan Distance: 34784.00
(e) Cosine Distance: 0.03
```

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 "heart.csv" dataset
data = pd.read_csv('heart.csv')

# Define the relevant feature columns
X = data[['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng', 'oldpeak', 'slp', 'caa', 'thall']]
```

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

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  output
0  0.624111  2.321270       1
1 -0.455988 -0.957351       1
2 -1.828805  0.042885       1
3 -1.716006 -0.495337       1
4 -0.371356  0.301156       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('heart.csv')

X = data[['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh','exng', 'oldpeak', 'slp', 'caa', 'thall']]
y = data['output']

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): 0.66
Accuracy (Manhattan Distance): 0.69
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('heart.csv')

X = data[['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng', 'oldpeak', 'slp', 'caa', 'thall']]

y = data['output']

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=['0','1'])
print("Classification Report:\n", class_report)
```

```
Accuracy: 0.7540983606557377
Classification Report:
               precision    recall  f1-score   support

           0       0.69      0.86      0.77        29
           1       0.84      0.66      0.74        32

    accuracy                           0.75        61
   macro avg       0.77      0.76      0.75        61
weighted avg       0.77      0.75      0.75        61
```

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('heart.csv')

X = data[['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng', 'oldpeak', 'slp', 'caa', 'thall']]
```

```
y = data['output']

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

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=['0','1'])
print("Classification Report:\n", class_report)
```

```
Accuracy: 0.8688524590163934
Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.86      0.86        29
           1       0.88      0.88      0.88        32

    accuracy                           0.87        61
   macro avg       0.87      0.87      0.87        61
weighted avg       0.87      0.87      0.87        61
```

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 "heart.csv" dataset
data = pd.read_csv('heart.csv')

# Define the relevant feature columns
X = data[['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng', 'oldpeak', 'slp', 'caa', 'thall']]

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

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): 0.11627071992880016
R-squared (R2) Score: 0.5337894947682486
```

CO-4 ASSIGNMENT:

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

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

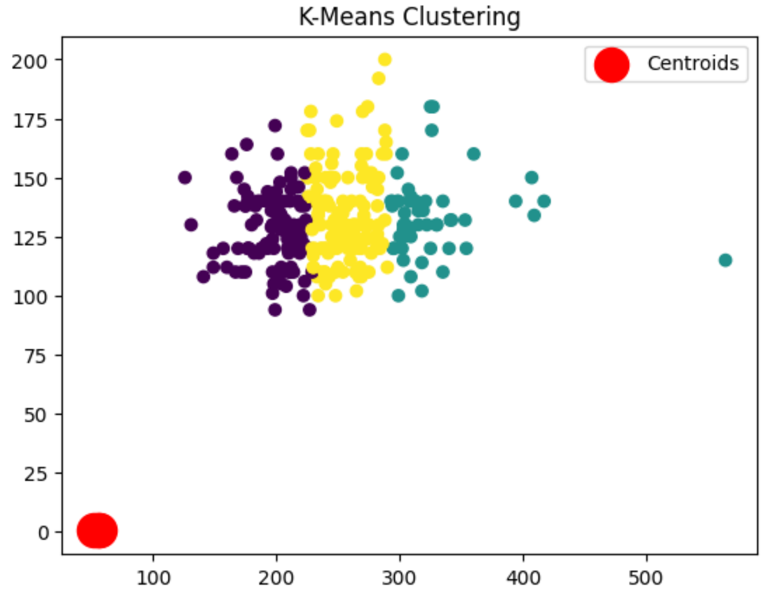
```
Requirement already satisfied: scikit-learn-extra in /usr/local/lib/python3.10/dist-packages (0.3.0)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.23.5)
Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.11.3)
```

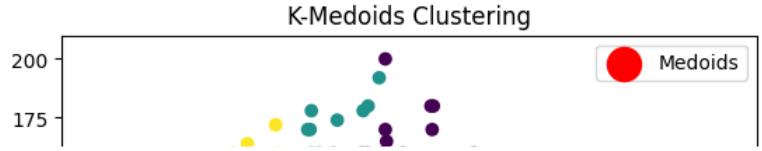
```
Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn-extra) (1.2
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->scikit-le
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.23.0->sc
```

```
data.head()
```

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	th
0	63	1	3	145	233	1	0	150	0	2.3	0	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	

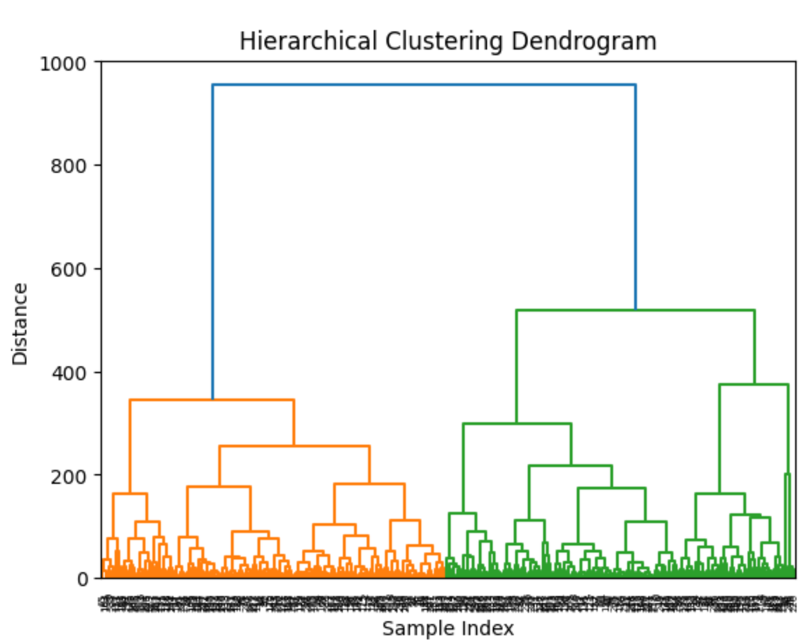
```
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn_extra.cluster import KMedoids
import matplotlib.pyplot as plt

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

X = data[['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng', 'oldpeak', 'slp', 'caa', 'thall']]

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

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

unknown_data = np.array([[63, 1, 0, 120, 354, 0, 178, 0, 0.6, 0, 0, 2, 1]])
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)
```

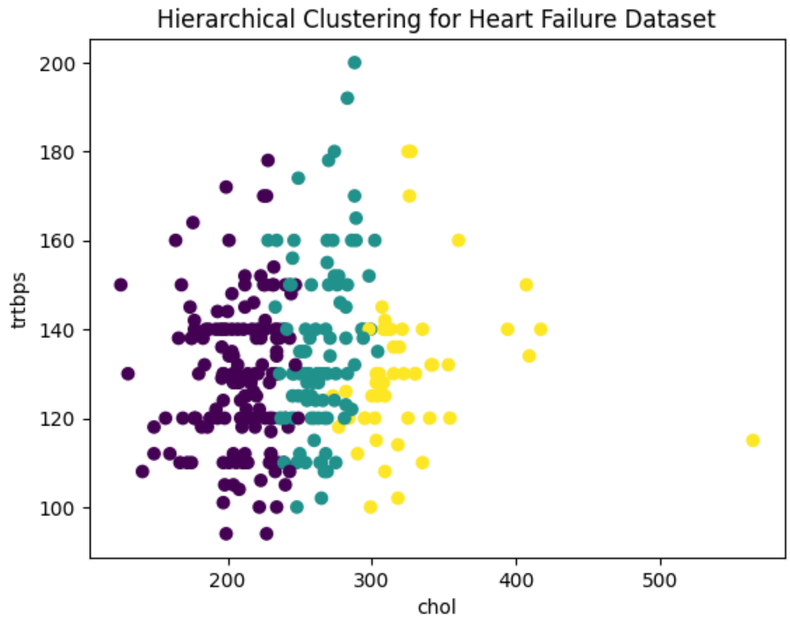




10. Implement hierarchical clustering algorithms and do prediction for unknown data.

```
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 "heart.csv" dataset
data = pd.read_csv('heart.csv')

# Define the relevant feature columns
X = data[['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng', 'oldpeak', 'slp', 'caa', 'thall']]

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['chol'], X['trtbps'], c=clusters, cmap='viridis')
plt.title('Hierarchical Clustering for Heart Failure Dataset')
plt.xlabel('chol')
plt.ylabel('trtbps')
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, 14.85, 0.00, 2.42, 73.72, 0.00, 8.
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)
```



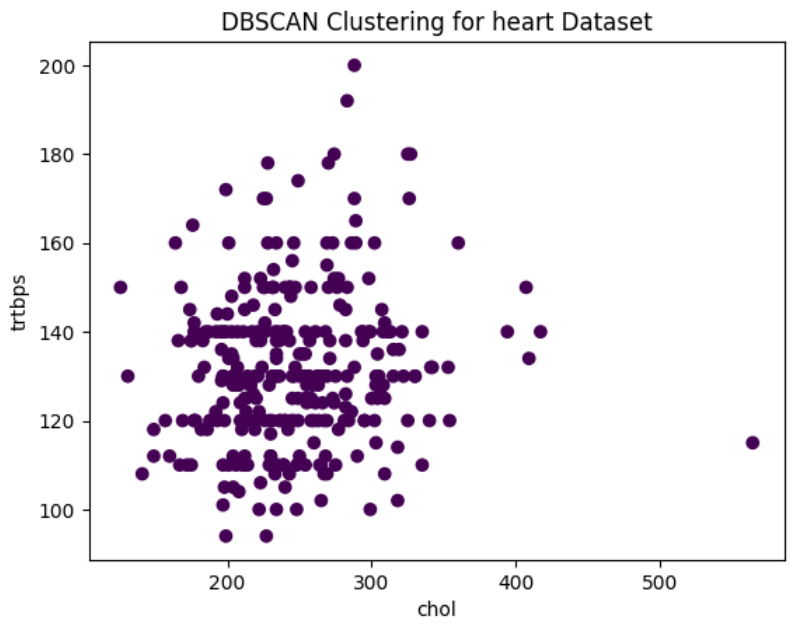



```
Clusters for Unknown Data: [2 2 1 2 3 1 2 2 1 1 1 2 2 1 2 1 3 1 1 2 1 1 1 2 1
```

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('heart.csv')

X = data[['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng', 'oldpeak', 'slp', 'caa', 'thall']]

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

plt.scatter(X['chol'], X['trtbps'], c=clusters, cmap='viridis')
plt.title('DBSCAN Clustering for heart Dataset')
plt.xlabel('chol')
plt.ylabel('trtbps')
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)
```




```
Clusters for Unknown Data: [-1 -1]
```

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=True)

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)                0.44                0.42     0.24
1          (A)         (F)                0.42                0.44     0.24
2          (H)         (A)                0.46                0.42     0.20
3          (A)         (H)                0.42                0.46     0.20
4          (C)         (B)                0.38                0.34     0.14
5          (B)         (C)                0.34                0.38     0.14
6          (B)         (H)                0.34                0.46     0.18
7          (H)         (B)                0.46                0.34     0.18
8          (I)         (B)                0.36                0.34     0.14
9          (B)         (I)                0.34                0.36     0.14
10         (C)         (F)                0.38                0.44     0.18
11         (F)         (C)                0.44                0.38     0.18
12         (C)         (G)                0.38                0.24     0.12
13         (G)         (C)                0.24                0.38     0.12
14         (I)         (C)                0.36                0.38     0.16
```

```
15        (C)        (I)          0.38          0.36     0.16
16        (C)        (K)          0.38          0.30     0.16
17        (K)        (C)          0.30          0.38     0.16
18        (F)        (D)          0.44          0.26     0.14
19        (D)        (F)          0.26          0.44     0.14
20        (D)        (H)          0.26          0.46     0.12
21        (H)        (D)          0.46          0.26     0.12
22        (D)        (J)          0.26          0.30     0.10
23        (J)        (D)          0.30          0.26     0.10
24        (F)        (H)          0.44          0.46     0.22
25        (H)        (F)          0.46          0.44     0.22
26        (F)        (K)          0.44          0.30     0.16
27        (K)        (F)          0.30          0.44     0.16
28        (K)        (G)          0.30          0.24     0.10
29        (G)        (K)          0.24          0.30     0.10
30        (H)        (J)          0.46          0.30     0.14
31        (J)        (H)          0.30          0.46     0.14
32        (I)        (J)          0.36          0.30     0.12
33        (J)        (I)          0.30          0.36     0.12
34     (C, F)        (A)          0.18          0.42     0.10
35     (C, A)        (F)          0.12          0.44     0.10
36     (F, A)        (C)          0.24          0.38     0.10
37        (C)     (F, A)          0.38          0.24     0.10
38        (F)     (C, A)          0.44          0.12     0.10
39        (A)     (C, F)          0.42          0.18     0.10

    confidence      lift  leverage  conviction  zhangs_metric
0     0.545455  1.298701    0.0552    1.276000       0.410714
1     0.571429  1.298701    0.0552    1.306667       0.396552
2     0.434783  1.035197    0.0068    1.026154       0.062963
3     0.476190  1.035197    0.0068    1.030909       0.058621
4     0.368421  1.083591    0.0108    1.045000       0.124424
5     0.411765  1.083591    0.0108    1.054000       0.116883
6     0.529412  1.150895    0.0236    1.147500       0.198653
7     0.391304  1.150895    0.0236    1.084286       0.242798
8     0.388889  1.143791    0.0176    1.080000       0.196429
9     0.411765  1.143791    0.0176    1.088000       0.190476
10    0.473684  1.076555    0.0128    1.064000       0.114695
11    0.409091  1.076555    0.0128    1.049231       0.126984
12    0.315789  1.315789    0.0288    1.110769       0.387097
13    0.500000  1.315789    0.0288    1.240000       0.315789
```

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 Heart Failure Prediction dataset from a CSV file
heart_data = pd.read_csv('heart.csv')

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

# 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)
```

```
Accuracy: 0.5245901639344263
```

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

(a)Write unknown 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.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 heart.csv dataset
data = pd.read_csv("heart.csv")

# Define features (X) and target (y)
X = data[['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng', 'oldpeak', 'slp', 'caa', 'thall']]
y = data['output']

# 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 example
    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)
```

```
Classification Results:
                Algorithm  Accuracy  Sensitivity  F-measure  Precision  \
0      Logistic Regression  0.885246            0   0.885122   0.885477
1           Decision Trees  0.754098            0   0.752240   0.770801
2            Random Forest  0.836066            0   0.836066   0.836066
3  Support Vector Machines  0.868852            0   0.868852   0.868852
4      k-Nearest Neighbors  0.655738            0   0.655738   0.658917

     Recall Prediction for Unknown Data
0  0.885246                        NA
1  0.754098                        NA
2  0.836066                        NA
3  0.868852                        NA
4  0.655738                        NA

Clustering Results:
   Algorithm Prediction for Unknown Data
0    K-Means                          NA
1  K-Medoids                          NA
```

(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 heart.csv dataset
data = pd.read_csv("heart.csv")

# Define features (X) and target (y)
X = data[['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng', 'oldpeak', 'slp', 'caa', 'thall']]
y = data['output']

# 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)
```

```
Compare performance of classification algorithms:
                 Algorithm  Accuracy  Sensitivity  F-measure  Precision  \
0      Logistic Regression  0.885246     0.885246   0.885122   0.885477
1           Decision Trees  0.754098     0.754098   0.752240   0.770801
2            Random Forest  0.836066     0.836066   0.836066   0.836066
3  Support Vector Machines  0.868852     0.868852   0.868852   0.868852
4      k-Nearest Neighbors  0.655738     0.655738   0.655738   0.658917
5          Neural Networks  0.524590     0.524590   0.361008   0.275195

     Recall
0  0.885246
1  0.754098
2  0.836066
3  0.868852
4  0.655738
5  0.524590
```

(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 heart.csv dataset
data = pd.read_csv("heart.csv")

# Define features (X) and target (y) for classification
X = data[['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng', 'oldpeak', 'slp', 'caa', 'thall']]
y = data['output']

# 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),
    "Neural Networks": MLPClassifier(hidden_layer_sizes=(10, 5), max_iter=1000, random_state=42)
}

# 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[['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng', 'oldpeak', 'slp', 'caa', 'thall']]
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_cluster, kmeans_clusters)],
    "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)
```

```
Compare performance of classification algorithms:
                 Algorithm  Accuracy  Sensitivity  F-measure  Precision  \
0      Logistic Regression  0.885246     0.885246   0.885122   0.885477
1           Decision Trees  0.754098     0.754098   0.752240   0.770801
2            Random Forest  0.836066     0.836066   0.836066   0.836066
3  Support Vector Machines  0.868852     0.868852   0.868852   0.868852
4      k-Nearest Neighbors  0.655738     0.655738   0.655738   0.658917
5          Neural Networks  0.524590     0.524590   0.361008   0.275195

     Recall
0  0.885246
1  0.754098
2  0.836066
3  0.868852
4  0.655738
5  0.524590

Compare performance of clustering algorithms:
                 Algorithm  Silhouette Score           WCSS
0  Hierarchical Clustering          0.257207       0.000000
1       K-Means Clustering          0.287765  471765.137524
```

(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 heart.csv dataset
data = pd.read_csv("heart.csv")

# Define features (X) and target (y)
X_cluster = data[['age', 'sex', 'cp', 'trtbps', 'chol', 'fbs', 'restecg', 'thalachh', 'exng', 'oldpeak', 'slp', 'caa', 'thall']]

# 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 Score": []}

# 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 methods:")
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 Method'])['Silhouette Score'].idx
print("\nBest clustering algorithm for each distance measure and linkage method:")
print(best_algorithm)
```

```
Compare performance of clustering algorithms with different distance measures
                  Algorithm Distance Measure Linkage Method  Silhouette Score
0   Hierarchical Clustering        euclidean         single          0.761880
1        K-Means Clustering        euclidean         single          0.287765
2   Hierarchical Clustering        euclidean       complete          0.361170
3        K-Means Clustering        euclidean       complete          0.287765
4   Hierarchical Clustering        euclidean        average          0.543802
5        K-Means Clustering        euclidean        average          0.287765
6   Hierarchical Clustering        minkowski         single          0.761880
7        K-Means Clustering        minkowski         single          0.287765
8   Hierarchical Clustering        minkowski       complete          0.361170
9        K-Means Clustering        minkowski       complete          0.287765
10  Hierarchical Clustering        minkowski        average          0.543802
11       K-Means Clustering        minkowski        average          0.287765
12  Hierarchical Clustering        cityblock         single          0.679206
13       K-Means Clustering        cityblock         single          0.255468
14  Hierarchical Clustering        cityblock       complete          0.679206
15       K-Means Clustering        cityblock       complete          0.255468
16  Hierarchical Clustering        cityblock        average          0.679206
17       K-Means Clustering        cityblock        average          0.255468
18  Hierarchical Clustering          jaccard         single         -0.047592
19       K-Means Clustering          jaccard         single          0.021528
20  Hierarchical Clustering          jaccard       complete         -0.047592
21       K-Means Clustering          jaccard       complete          0.021528
22  Hierarchical Clustering          jaccard        average         -0.047592
23       K-Means Clustering          jaccard        average          0.021528
24  Hierarchical Clustering           cosine         single          0.682905
25       K-Means Clustering           cosine         single          0.438290
26  Hierarchical Clustering           cosine       complete          0.682905
27       K-Means Clustering           cosine       complete          0.438290
28  Hierarchical Clustering           cosine        average          0.682905
29       K-Means Clustering           cosine        average          0.438290
30  Hierarchical Clustering          hamming         single          0.002838
31       K-Means Clustering          hamming         single          0.010561
32  Hierarchical Clustering          hamming       complete          0.002838
33       K-Means Clustering          hamming       complete          0.010561
34  Hierarchical Clustering          hamming        average          0.002838
35       K-Means Clustering          hamming        average          0.010561

Best clustering algorithm for each distance measure and linkage method:
                  Algorithm Distance Measure Linkage Method  Silhouette Score
16  Hierarchical Clustering        cityblock        average          0.679206
14  Hierarchical Clustering        cityblock       complete          0.679206
12  Hierarchical Clustering        cityblock         single          0.679206
28  Hierarchical Clustering           cosine        average          0.682905
26  Hierarchical Clustering           cosine       complete          0.682905
24  Hierarchical Clustering           cosine         single          0.682905
4   Hierarchical Clustering        euclidean        average          0.543802
2   Hierarchical Clustering        euclidean       complete          0.361170
0   Hierarchical Clustering        euclidean         single          0.761880
35       K-Means Clustering          hamming        average          0.010561
33       K-Means Clustering          hamming       complete          0.010561
31       K-Means Clustering          hamming         single          0.010561
```

15. Write any deep learning program of your choice.

10 Hierarchical Clustering minkowski average 0.543802


```
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=['accuracy'])

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

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

```
Epoch 1/5
1875/1875 [==============================] - 19s 10ms/step - loss: 0.4944 - accuracy: 0.8248
Epoch 2/5
```

```
1875/1875 [==============================] - 17s 9ms/step - loss: 0.3729 - accuracy: 0.8651
Epoch 3/5
1875/1875 [==============================] - 16s 8ms/step - loss: 0.3360 - accuracy: 0.8767
Epoch 4/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.3124 - accuracy: 0.8856
Epoch 5/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.2954 - accuracy: 0.8907
313/313 [==============================] - 1s 3ms/step - loss: 0.3462 - accuracy: 0.8787

Test accuracy: 0.8787000179290771
```