WEEK 14

67 Write a python code for Agglomerative clustering with different metrics in Scikit Learn.

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler

# Create sample data using make_blobs
X, y = make_blobs(n_samples=150, centers=3, random_state=42)

# Standardize the features (optional but often helpful)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Function to perform Agglomerative Clustering with different distance metrics
def agglomerative_clustering(X, metric):
    # Perform Agglomerative Clustering
    model = AgglomerativeClustering(n_clusters=3, affinity=metric, linkage='average')
    labels = model.fit_predict(X)

    # Calculate the silhouette score to measure clustering quality
    score = silhouette_score(X, labels)

    # Plot the clusters
```

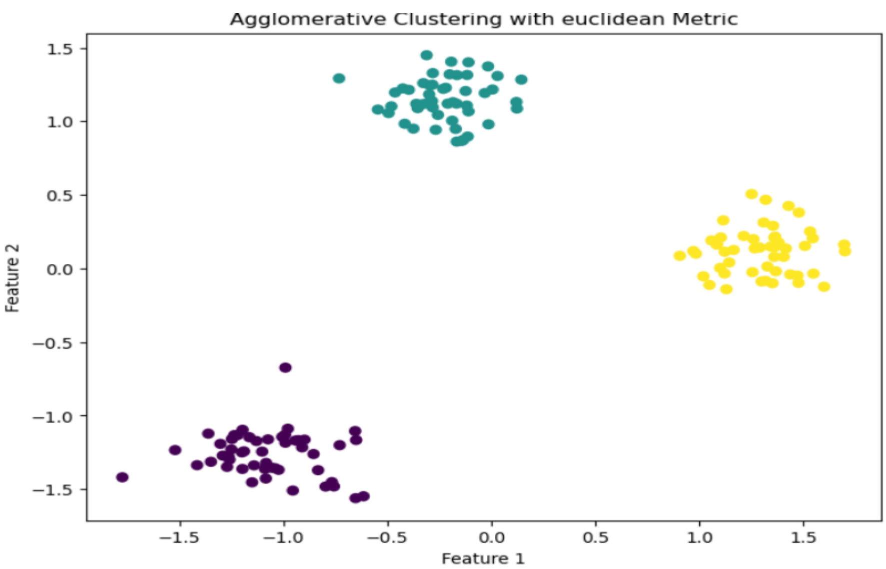
```
    plt.figure(figsize=(8, 6))
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o')
    plt.title(f"Agglomerative Clustering with {metric} Metric")
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.show()

    return labels, score

# List of different metrics to try
metrics = ['euclidean', 'manhattan',s 'cosine', 'l1', 'l2']

# Loop through different metrics
for metric in metrics:
    labels, score = agglomerative_clustering(X_scaled, metric)
    print(f"Silhouette Score for {metric} metric: {score:.4f}")
```

Output




```
Silhouette Score for euclidean metric: 0.8461
```

68. Implement the association rules method for analyse buyer baskets and detect cross-category purchase correlations.

```
import pandas as pd

from mlxtend.frequent_patterns import apriori, association_rules

from mlxtend.preprocessing import TransactionEncoder


# Step 1: Prepare the data (list of transactions with each transaction being a list of items)

transactions = [

    ['milk', 'bread', 'butter'],

    ['bread', 'butter', 'jam'],

    ['milk', 'bread', 'butter', 'jam'],

    ['milk', 'bread', 'butter'],

    ['bread', 'butter', 'jam'],

    ['milk', 'bread'],

]


# Step 2: Convert the transactions to the format required by the Apriori algorithm (one-hot encoded)

te = TransactionEncoder()

te_ary = te.fit_transform(transactions)

df = pd.DataFrame(te_ary, columns=te.columns_)


# Step 3: Apply the Apriori algorithm to mine frequent itemsets
```

```
# Setting a minimum support threshold (e.g., 0.4 means at least 40% of transactions must contain the itemset)
frequent_itemsets = apriori(df, min_support=0.4, use_colnames=True)

# Step 4: Generate the association rules
# Setting a minimum confidence threshold (e.g., 0.7 means the rule should have at least 70% confidence)
rules = association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)

# Step 5: Filter the rules for cross-category purchases (optional)
# Example: Look for rules where antecedents and consequents are in different categories
# Assuming we know that 'milk' and 'bread' are in the 'Dairy' and 'Bakery' categories
rules['antecedent_len'] = rules['antecedents'].apply(lambda x: len(x))

# Filter for rules that involve exactly 1 item in the antecedent and consequent (cross-category detection)
cross_category_rules = rules[(rules['antecedent_len'] == 1) & (rules['lift'] > 1.0)]

# Step 6: Display the filtered association rules
print("Filtered Cross-Category Rules:")
print(cross_category_rules[['antecedents', 'consequents', 'support', 'confidence', 'lift']])
```

Output

```
Filtered Cross-Category Rules:
  antecedents       consequents  support  confidence  lift
4       (jam)          (butter)      0.5         1.0   1.2
8       (jam)  (butter, bread)      0.5         1.0   1.2
```

Note : !pip install mlxtend

69. Implement How frequent itemsets are singled out in the transactions using apriori algorithm.

70. Illustrates the progression of agglomerative clustering on a two dimensional dataset, looking for three clusters.

```
import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import AgglomerativeClustering

from sklearn.datasets import make_blobs

from scipy.cluster.hierarchy import dendrogram, linkage


# Step 1: Generate a 2D dataset with 3 distinct clusters

X, y = make_blobs(n_samples=100, centers=3, random_state=42)


# Step 2: Visualize the dataset

plt.figure(figsize=(6, 6))

plt.scatter(X[:, 0], X[:, 1], c='blue', marker='o')
```

```
plt.title("Original 2D Dataset")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()

# Step 3: Apply Agglomerative Clustering to the data
agg_clust = AgglomerativeClustering(n_clusters=3)
y_pred = agg_clust.fit_predict(X)

# Step 4: Visualize the resulting clusters
plt.figure(figsize=(6, 6))
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap='viridis', marker='o')
plt.title("Agglomerative Clustering (3 Clusters)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()

# Step 5: Plot the Dendrogram to show the progression of the clustering
# Perform hierarchical/agglomerative clustering
Z = linkage(X, 'ward')

# Plot dendrogram
plt.figure(figsize=(10, 6))
dendrogram(Z)
plt.title("Dendrogram (Agglomerative Clustering Progression)")
```

```
plt.xlabel("Sample Index")
plt.ylabel("Distance")
plt.show()
```

Output