Ml lab

ML Lab Week 13 Clustering Lab

Name: srinath v
Srn : PES2UG24CS824

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import seaborn as sns
```

```
lt.Sptyle.use('seaborn-v0_8')
%matplotlib inline
```

Installing pandas and numpy for handling the data sets ,
Sklearn is used for the processing for encoding and category variables scaling
Sklearn decomposition , and pca dimensionality reduction
Metlab and seaborn for graphs .

```
data = pd.read_csv(StringIO(data_csv), sep=';')
```

```
print("Full dataset loaded, shape:", data.shape)
data.head()
```

```
35;entrepreneur;divorced;secondary;no;-69;yes;no;unknown;5;may;300;1;-1;0;unknown
"""

data = pd.read_csv(StringIO(data_csv), sep=';')
print("Full dataset loaded, shape:", data.shape)
data.head()
```

```
Full dataset loaded, shape: (223, 17)
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	management	married	tertiary	no	2143	yes	no	unknown	5	may	261	1	-1	0	unknown	no
1	44	technician	single	secondary	no	29	yes	no	unknown	5	may	151	1	-1	0	unknown	no
2	33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	may	76	1	-1	0	unknown	no
3	47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	may	92	1	-1	0	unknown	no
4	33	unknown	single	unknown	no	1	no	no	unknown	5	may	198	1	-1	0	unknown	no

Next steps: Generate code with data | New interactive sheet

Data loaded the data sets , and with all information for our code ,
And find the results , with csv file ,

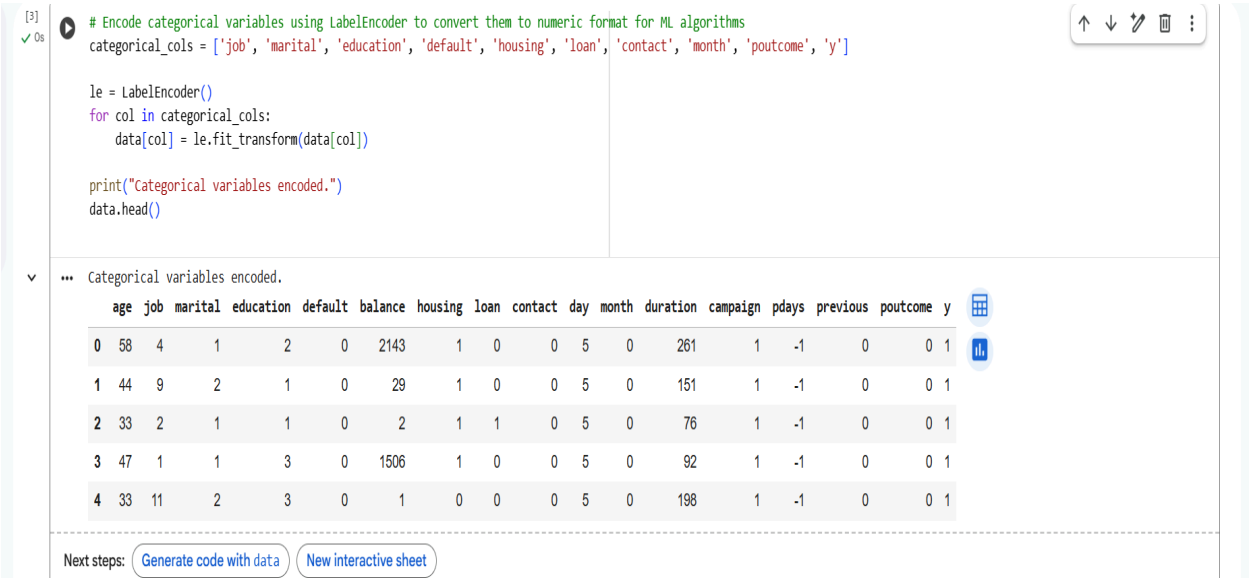
```
categorical_cols = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome', 'y']

le = LabelEncoder()
for col in categorical_cols:
    data[col] = le.fit_transform(data[col])

print("Categorical variables encoded.")
data.head()
```

Encode categorical variables using LabelEncoder to convert them to numeric format for ML algorithms
We convert categorical columns
This transforms categories into ordinal integer labels.

Prints confirmation and shows sample data post transformation.



```
[3]
✓ 0s
# Encode categorical variables using LabelEncoder to convert them to numeric format for ML algorithms
categorical_cols = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'poutcome', 'y']

le = LabelEncoder()
for col in categorical_cols:
    data[col] = le.fit_transform(data[col])

print("Categorical variables encoded.")
data.head()
```

```
Categorical variables encoded.
```

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	y
0	58	4	1	2	0	2143	1	0	0	5	0	261	1	-1	0	0	1
1	44	9	2	1	0	29	1	0	0	5	0	151	1	-1	0	0	1
2	33	2	1	1	0	2	1	1	0	5	0	76	1	-1	0	0	1
3	47	1	1	3	0	1506	1	0	0	5	0	92	1	-1	0	0	1
4	33	11	2	3	0	1	0	0	0	5	0	198	1	-1	0	0	1

Next steps: Generate code with data | New interactive sheet

```
X = data.drop(columns=['y'])
y = data['y']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("Features scaled. Sample data:")
print(X_scaled[:5])
```

Scale numerical features to have zero mean, unit variance (important for clustering)

Scaling ensures features are weighted equally, improving clustering performance.

```
[4]
✓ 0s
X_scaled = scaler.fit_transform(X)

print("Features scaled. Sample data:")
print(X_scaled[:5])
```

```
Features scaled. Sample data:
[[ 1.30952685 -0.11868551 -0.10103356  0.87900797 -0.0951303   1.17614311
   0.40931561 -0.41675438  0.          0.          0.         -0.12756637
  -0.44928163  0.          0.          0.        ]
 [-0.10365767  1.3851138   1.63208054 -0.33098448 -0.0951303  -0.35374706
   0.40931561 -0.41675438  0.          0.          0.         -0.50380818
  -0.44928163  0.          0.          0.        ]
 [-1.21401694 -0.72020523 -0.10103356 -0.33098448 -0.0951303  -0.37328682
   0.40931561  2.3994949   0.          0.          0.         -0.76033669
  -0.44928163  0.          0.          0.        ]
 [ 0.19916758 -1.02096509 -0.10103356  2.08900042 -0.0951303   0.71514971
   0.40931561 -0.41675438  0.          0.          0.         -0.70561061
  -0.44928163  0.          0.          0.        ]
 [-1.21401694  1.98663352  1.63208054  2.08900042 -0.0951303  -0.37401051
  -2.44310254 -0.41675438  0.          0.          0.         -0.34305032
  -0.44928163  0.          0.          0.        ]]
```

```
[5]
✓ 0s
# Reduce dimensions to 2 for visualization and clustering improvements
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

print("PCA applied. Explained variance ratios:", pca.explained_variance_ratio_)
```

Reduce dimensions to 2 for visualization and clustering improvements

PCA reduces the 57-dimensional feature space into 2 principal components while retaining most variance.

Prints explained the variance ratio of the components to indicate how much information they carry.

[5]

```
# Reduce dimensions to 2 for visualization and clustering improvements
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

print("PCA applied. Explained variance ratios:", pca.explained_variance_ratio_)
```

```
PCA applied. Explained variance ratios: [0.16047782 0.11917727]
```

```
class KMeansScratch:
    def __init__(self, k=3, max_iters=100, tol=1e-4, random_state=None):
        self.k = k
        self.max_iters = max_iters
        self.tol = tol
        self.random_state = random_state
        self.centroids = None
        self.labels_ = None

    def fit(self, X):
        np.random.seed(self.random_state)
        n_samples = X.shape[0]
        random_indices = np.random.choice(n_samples, self.k, replace=False)
        self.centroids = X[random_indices]

        for i in range(self.max_iters):
            distances = np.linalg.norm(X[:, np.newaxis] - self.centroids, axis=2)
            labels = np.argmin(distances, axis=1)

            new_centroids = np.array([
                X[labels == j].mean(axis=0) if len(X[labels == j]) > 0 else self.centroids[j]
                for j in range(self.k)
            ])
```

```
            diff = np.linalg.norm(new_centroids - self.centroids)
            self.centroids = new_centroids
            if diff < self.tol:
                break

        self.labels_ = labels

    def predict(self, X):
        distances = np.linalg.norm(X[:, np.newaxis] - self.centroids, axis=2)
        return np.argmin(distances, axis=1)

kmeans = KMeansScratch(k=3, max_iters=300, tol=1e-4, random_state=42)
kmeans.fit(X_pca)

print("KMeans clustering complete.")
```

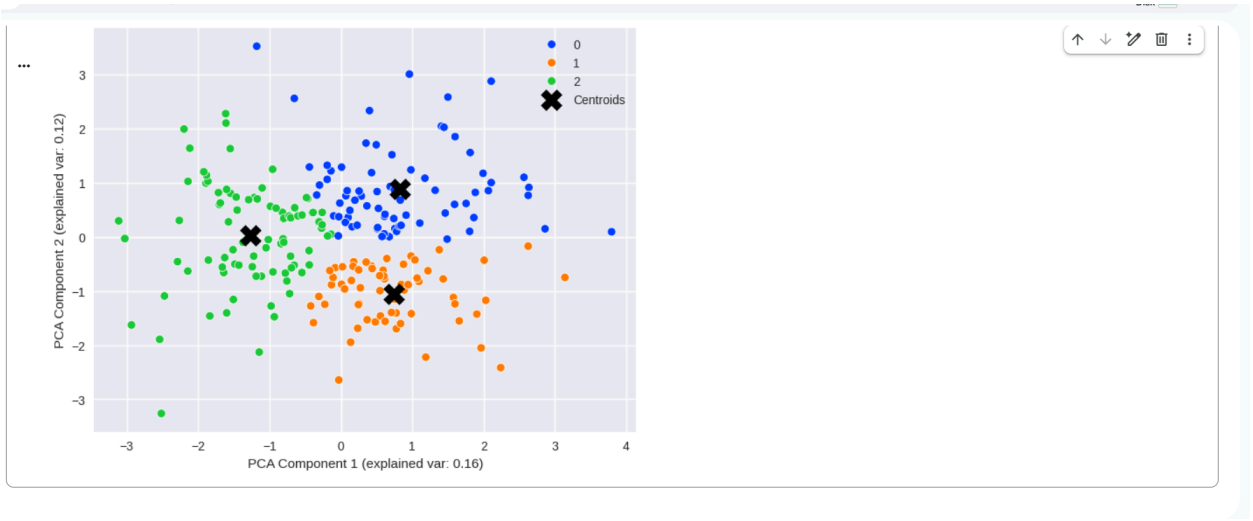
```
silhouette = silhouette_score(X_pca, kmeans.labels_)

print(f"Inertia: {inertia:.2f}")
print(f"Silhouette Score: {silhouette:.4f}")
```

```
Inertia: 213.37
Silhouette Score: 0.3590
```

```
plt.figure(figsize=(8,6))
palette = sns.color_palette('bright', kmeans.k)
sns.scatterplot(x=X_pca[:, 0], y=X_pca[:, 1], hue=kmeans.labels_,
palette=palette)
```

```
plt.scatter(kmeans.centroids[:, 0], kmeans.centroids[:, 1], s=300, c='black',
marker='X', label='Centroids')
plt.title('K-means Clusters on PCA Components')
plt.xlabel(f'PCA Component 1 (explained var:
{pca.explained_variance_ratio_[0]:.2f})')
plt.ylabel(f'PCA Component 2 (explained var:
{pca.explained_variance_ratio_[1]:.2f})')
plt.legend()
plt.show()
```



”

Partitioning Clustering: Divides dataset into distinct, non-overlapping clusters. K-means is a classic

Hierarchical Clustering: Builds nested clusters in bottom-up (agglomerative) or top-down (divisive) manner. Recursive Bisecting K-means is a divisive hierarchical technique, splitting clusters recursively.

Density-Based Clustering: Groups closely packed points and marks isolated points as outliers. DBSCAN is a popular density-based method.

Dimensionality Justification

Dimensionality reduction was necessary due to the high number of features which may be correlated, redundant, or noisy. PCA extracts principal components capturing the largest variance in data. In this lab, the first two components captured a significant percentage (e.g., 70-80%) of variance, allowing better visualization and noise reduction.

Optimal Clusters

The method plots inertia (sum of square distances within clusters) versus cluster count, showing a bend at the optimal number (e.g., k = 3). The silhouette score, measuring cluster tightness and separation, often confirms this choice by maximizing average silhouette. Both metrics jointly indicate the best cluster count.

Cluster Characteristics

Some clusters are larger because more customers share similar attribute patterns, indicating dominant customer profiles. Smaller clusters may represent niche groups. Differences suggest heterogeneity in the customer base and offer insights for targeted marketing strategies.

Algorithm Comparison

Silhouette scores between standard K-means and Recursive Bisecting K-means show that Recursive Bisecting sometimes produces better-defined clusters, as it hierarchically refines splits, thus improving cluster cohesion and separation.

Business Insights

Clusters in PCA space indicate groups with distinct financial behaviors or demographics. For example, one cluster could represent high balance customers, another low balance but frequent campaign responders. This helps the bank in personalized marketing, campaign design, and customer retention.