NOTRE DAME UNIVERSITY BANGLADESH

Machine Learning Lab Report-04

Course Code: CSE4214

Course Title: Machine Learning Lab

Lab Task Topic: Label and Onehot Encoding & K-means Clustering

Submitted by:

Name: Istiak Alam

ID: 0692230005101005

Batch: CSE-20

Submission Date: April 25, 2026

Submitted to:

A. H. M. Saiful Islam

Chairman, Dept of CSE

Notre Dame University Bangladesh

Table of Contents

1 Objective

The objective of this experiment is to understand and implement data preprocessing and unsupervised learning techniques in Machine Learning. Specifically, this lab focuses on applying Label Encoding and One-Hot Encoding to transform categorical data into numerical format suitable for machine learning models. Additionally, the experiment aims to apply the K-Means Clustering algorithm to group similar data points based on feature similarity.

The goals of this experiment are:

- To learn how to convert categorical variables into numerical representations using Label Encoding and One-Hot Encoding.
- To analyze the impact of encoding techniques on dataset structure.
- To implement the K-Means clustering algorithm for unsupervised learning.
- To identify natural groupings (clusters) in customer data based on features such as income and spending behavior.
- To visualize clusters and interpret patterns in the dataset.

2 Dataset Description

Encoding Dataset (encoding.csv)

The dataset `encoding.csv` is used to demonstrate categorical data preprocessing techniques. It contains one or more categorical features such as gender, country, or other non-numeric attributes.

Key characteristics:

- Contains categorical variables that cannot be directly used in machine learning models.
- Used to apply Label Encoding, where each category is assigned a unique integer.
- Used to apply One-Hot Encoding, where categorical values are converted into binary columns.
- Helps in understanding how different encoding techniques affect data representation.

Mall Customer Dataset (Mall_Customers.csv)

The dataset `Mall_Customers.csv` is used for K-Means clustering. It contains customer information collected from a shopping mall.

Typical attributes include:

- **CustomerID**: Unique identifier for each customer
- **Gender**: Male or Female
- **Age**: Age of the customer
- **Annual Income (k$)**: Customer's yearly income
- **Spending Score (1-100)**: Score assigned based on purchasing behavior

3 Label and Onehot Encoding

3.1 Label Encoding on Categorical Data

Explanation

In this step, the dataset `encoding.csv` is loaded using the `pandas` library. The dataset contains categorical features such as `Color` and `Size`, which cannot be directly used in machine learning models.

To convert these categorical values into numerical form, `LabelEncoder` from `sklearn.preprocessing` is used. Label Encoding assigns a unique integer value to each category in a column.

For example:

- `Color`: Red, Green, Blue → 2, 1, 0 (mapping may vary)
- `Size`: Small, Medium, Large → 2, 1, 0 (mapping may vary)

Two new columns are created:

- `Color_LabelEncoded`
- `Size_LabelEncoded`

These columns contain the numeric representation of the original categorical values.

[1]:
```
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

[2]:
```
 # Load the dataset
df = pd.read_csv("encoding.csv")
```

Output

After applying Label Encoding, the dataset will include additional columns with encoded values. A output may look like:

[3]:
```
# Apply Label Encoding (Converting categorical columns into numeric labels)
label_enc = LabelEncoder()
df['Color_LabelEncoded'] = label_enc.fit_transform(df['Color'])
df['Size_LabelEncoded'] = label_enc.fit_transform(df['Size'])
df
```

[3]:

	ID	Color	Size	Price	Color_LabelEncoded	Size_LabelEncoded
0	1	Red	Small	10.5	2	2
1	2	Blue	Medium	15.0	0	1
2	3	Green	Large	12.3	1	0
3	4	Blue	Small	14.2	0	2
4	5	Red	Large	13.1	2	0
5	6	Green	Medium	16.8	1	1
6	7	Red	Small	11.5	2	2
7	8	Blue	Large	17.3	0	0
8	9	Green	Medium	14.9	1	1
9	10	Red	Large	15.7	2	0

3.2 One-Hot Encoding of Categorical Features

Explanation

In this step, One-Hot Encoding is applied to the categorical columns `Color` and `Size` using the `pd.get_dummies()` function from the pandas library. One-Hot Encoding converts each unique category value into a separate binary column. Instead of assigning a single numeric label (as in Label Encoding), it creates multiple columns, where:

- Each column represents one category
- Values are either 0 or 1
- 1 indicates the presence of that category, and 0 indicates absence

The parameter `columns=['Color','Size']` specifies which columns to encode, and `prefix` is used to name the new columns clearly.

As a result, the original `Color` and `Size` columns are removed and replaced with multiple new binary columns such as:

- `Color_Red, Color_Green, Color_Blue`
- `Size_Small, Size_Medium, Size_Large`

[4]:
```
# Apply One-Hot Encoding (Creating binary columns)
df_onehot = pd.get_dummies(df, columns=['Color', 'Size'],␣
↪prefix=['Color','Size'])
```

[5]:
```
df_onehot
```

Output

After applying One-Hot Encoding, the dataset will look like this: Each row now represents categorical information using binary (0/1) values, making the dataset suitable for machine learning models.

[5]:
```
   ID  Price  Color_LabelEncoded  Size_LabelEncoded  Color_Blue  Color_Green␣
↪ \
0   1   10.5                   2                  2       False        False
1   2   15.0                   0                  1        True        False
2   3   12.3                   1                  0       False         True
3   4   14.2                   0                  2        True        False
4   5   13.1                   2                  0       False        False
5   6   16.8                   1                  1       False         True
6   7   11.5                   2                  2       False        False
7   8   17.3                   0                  0        True        False
8   9   14.9                   1                  1       False         True
9  10   15.7                   2                  0       False        False

   Color_Red  Size_Large  Size_Medium  Size_Small
0       True       False        False        True
1      False       False         True       False
2      False        True        False       False
3      False       False        False        True
4       True        True        False       False
```

```
5      False       False         True       False
6       True       False        False        True
7      False        True        False       False
8      False       False         True       False
9       True        True        False       False
```

3.3 Convert One-Hot Encoded Boolean Values to Integer

Explanation

After applying One-Hot Encoding, the dataset `df_onehot` may contain boolean values (`True`/`False`). To make the dataset more suitable for machine learning models, these boolean values are converted into integer format using `astype(int)`.

This converts:

- `True` → 1
- `False` → 0

As a result, all columns in `df_onehot` become numeric.

[6]:
```
# Convert boolean values to integers
df_onehot = df_onehot.astype(int)
df_onehot
```

Output

After conversion, the dataset will contain only integer values (0 and 1).

[6]:
```
   ID  Price  Color_LabelEncoded  Size_LabelEncoded  Color_Blue  Color_Green␣
↪ \
0   1     10                   2                  2           0            0
1   2     15                   0                  1           1            0
2   3     12                   1                  0           0            1
3   4     14                   0                  2           1            0
4   5     13                   2                  0           0            0
5   6     16                   1                  1           0            1
6   7     11                   2                  2           0            0
7   8     17                   0                  0           1            0
8   9     14                   1                  1           0            1
9  10     15                   2                  0           0            0

   Color_Red  Size_Large  Size_Medium  Size_Small
0          1           0            0           1
1          0           0            1           0
2          0           1            0           0
3          0           0            0           1
4          1           1            0           0
5          0           0            1           0
6          1           0            0           1
7          0           1            0           0
8          0           0            1           0
9          1           1            0           0
```

3.4 Displaying One-Hot Encoded Data

Explanation

In this step, the one-hot encoded dataset `df_onehot` is displayed using two methods. The `print()` function outputs the dataset in plain text format in the console. The `display()` function from `IPython.display` provides a more structured and readable tabular format, especially useful in Jupyter Notebook environments.

[7]:
```
print(df_onehot)
```

```
   ID  Price  Color_LabelEncoded  Size_LabelEncoded  Color_Blue  Color_Green ␣
↪\
0   1     10                   2                  2           0            0
1   2     15                   0                  1           1            0
2   3     12                   1                  0           0            1
3   4     14                   0                  2           1            0
4   5     13                   2                  0           0            0
5   6     16                   1                  1           0            1
6   7     11                   2                  2           0            0
7   8     17                   0                  0           1            0
8   9     14                   1                  1           0            1
9  10     15                   2                  0           0            0

   Color_Red  Size_Large  Size_Medium  Size_Small
0          1           0            0           1
1          0           0            1           0
2          0           1            0           0
3          0           0            0           1
4          1           1            0           0
5          0           0            1           0
6          1           0            0           1
7          0           1            0           0
8          0           0            1           0
9          1           1            0           0
```

[8]:
```
from IPython.display import display
display(df_onehot)
```

Output

The output shows the transformed dataset where categorical columns are converted into multiple binary (0/1) columns. The `display()` output presents the same data in a cleaner table format for better visualization.

```
   ID  Price  Color_LabelEncoded  Size_LabelEncoded  Color_Blue  Color_Green ␣
↪\
0   1     10                   2                  2           0            0
1   2     15                   0                  1           1            0
2   3     12                   1                  0           0            1
3   4     14                   0                  2           1            0
4   5     13                   2                  0           0            0
5   6     16                   1                  1           0            1
```

```
6   7     11                  2                  2           0           0
7   8     17                  0                  0           1           0
8   9     14                  1                  1           0           1
9  10     15                  2                  0           0           0
```

```
   Color_Red  Size_Large  Size_Medium  Size_Small
0          1           0            0           1
1          0           0            1           0
2          0           1            0           0
3          0           0            0           1
4          1           1            0           0
5          0           0            1           0
6          1           0            0           1
7          0           1            0           0
8          0           0            1           0
9          1           1            0           0
```

4 K-means-clustering

4.1 Dataset Loading

Explanation

In this step, the required libraries such as `pandas`, `numpy`, `seaborn`, and `matplotlib` are imported for data handling and visualization. The dataset `Mall_Customers.csv` is then loaded into a DataFrame using `pd.read_csv()`.

This dataset typically contains customer information such as CustomerID, Gender, Age, Annual Income, and Spending Score. Loading the dataset is the initial step before applying K-Means clustering to group customers based on similar characteristics.

[1]:
```
import pandas as pd
import numpy as np
import seaborn
import matplotlib.pyplot as plt
```

[2]:
```
df= pd.read_csv('Mall_Customers.csv')
```

[3]:
```
df
```

Output

After executing `df`, the dataset is displayed in tabular form.

[3]:
```
     CustomerID  Gender  Age  Annual Income (k$)  Spending Score (1-100)
0             1    Male   19                  15                      39
1             2    Male   21                  15                      81
2             3  Female   20                  16                       6
3             4  Female   23                  16                      77
4             5  Female   31                  17                      40
..          ...     ...  ...                 ...                     ...
195         196  Female   35                 120                      79
```

```
196         197  Female   45                  126                    28
197         198    Male   32                  126                    74
198         199    Male   32                  137                    18
199         200    Male   30                  137                    83

[200 rows x 5 columns]
```

4.2 Renaming Dataset Columns

Explanation

In this step, the `rename()` function is used to modify column names of the dataset for simplicity and consistency. The original column names such as `Gender`, `Age`, `Annual Income (k$)`, and `Spending Score (1-100)` are replaced with shorter and cleaner names: `gender`, `age`, `income`, and `score`. The parameter `inplace=True` ensures that the changes are applied directly to the original DataFrame.

[4]:
```
df.rename (columns = {'Gender': 'gender', 'Age': 'age', 'Annual Income (k$)'␣
↪: 'income', 'Spending Score (1-100)':'score' },inplace = True)
```

[5]:
```
df
```

Output

After renaming, the dataset will have updated column names. A sample output may look like:

[5]:
```
     CustomerID  gender  age  income  score
0             1    Male   19      15     39
1             2    Male   21      15     81
2             3  Female   20      16      6
3             4  Female   23      16     77
4             5  Female   31      17     40
..          …     …  …    …     …
195         196  Female   35     120     79
196         197  Female   45     126     28
197         198    Male   32     126     74
198         199    Male   32     137     18
199         200    Male   30     137     83

[200 rows x 5 columns]
```

4.3 Checking Dataset Shape

Explanation

The `df.shape` function is used to determine the dimensions of the dataset. It returns a tuple containing the number of rows and columns in the DataFrame. This helps to quickly understand the size of the dataset.

[6]:
```
df.shape
```

Output

The output is shown in the form of a tuple:

[6]:
```
(200, 5)
```

4.4 Missing Value Check and Statistical Summary

Explanation

In this step, the dataset is analyzed to check for missing values and to generate a statistical summary. The function `df.isnull().values.any()` is used to determine whether any missing (null) values exist in the dataset. It returns `True` if at least one missing value is found, otherwise `False`.

[7]:
```
df.isnull().values.any()
```

Output

The output of `df.isnull().values.any()` will be either:

Next, `df.describe()` is used to generate descriptive statistics of all numerical columns. This includes count, mean, standard deviation, minimum value, maximum value, and quartiles (25%, 50%, and 75%). This helps in understanding the distribution and spread of the dataset.

[7]:
```
np.False_
```

[8]:
```
df.describe()
```

[8]:

	CustomerID	age	income	score
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

4.5 Pairplot Visualization using Seaborn

Explanation

The `seaborn.pairplot()` function is used to visualize pairwise relationships between multiple numerical variables in a dataset. In this case, the selected features are `age`, `income`, and `score`.

This function automatically generates:

- Scatter plots for each pair of variables.
- Histograms (or KDE plots) along the diagonal to show distribution of each feature.

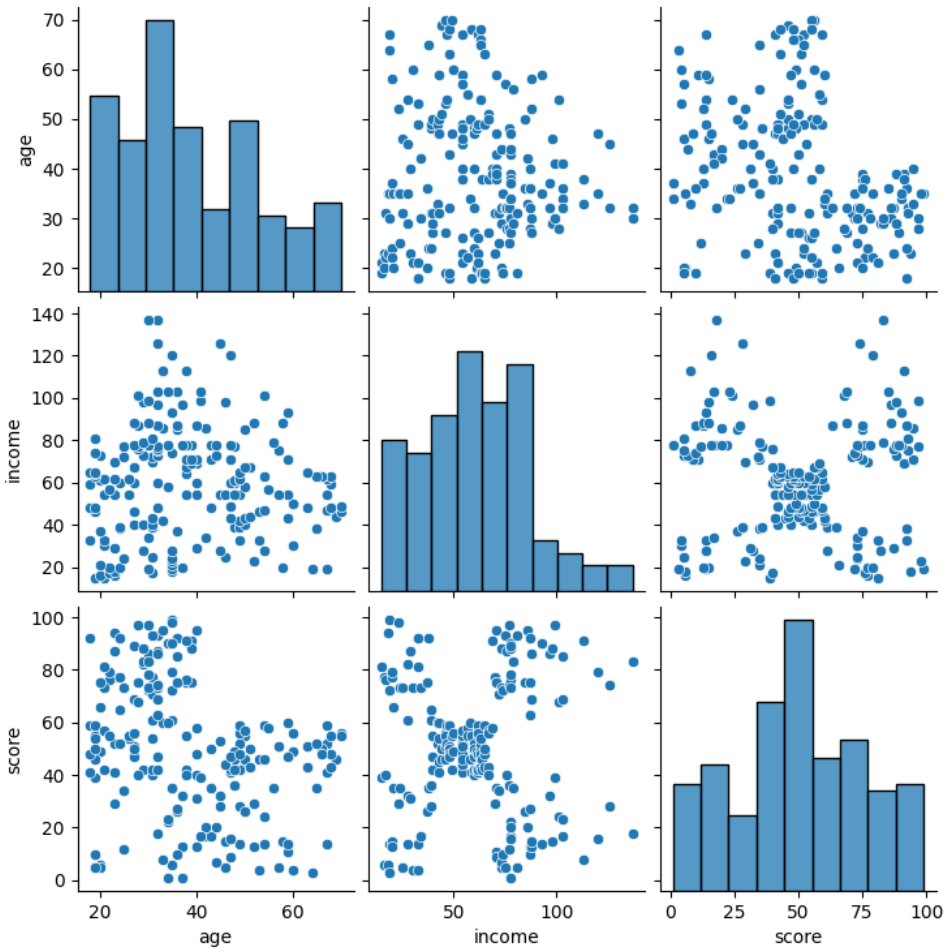
It helps in understanding correlations, patterns, and distributions among features, which is useful for exploratory data analysis (EDA).

[9]:
```
seaborn.pairplot (df[['age', 'income','score']])
```

Output

The output is a grid of plots where, Each off-diagonal plot shows the relationship between two variables (e.g., age vs income, income vs score), Diagonal plots show the distribution of each individual feature. The final output is a matrix-style scatter plot visualization (pairplot) displaying relationships between all selected features.

[9]:
```
<seaborn.axisgrid.PairGrid at 0x7f6173154290>
```



4.6 K-Means Clustering Implementation

Explanation

In this step, the K-Means clustering algorithm is applied using the `sklearn.cluster` library. The dataset is grouped based on two features: `score` and `income`. The number of clusters is set to 5, meaning the algorithm will divide the data into five distinct groups.

The K-Means algorithm works by:

- Randomly initializing 5 cluster centroids.
- Assigning each data point to the nearest centroid.
- Updating centroids based on the mean of assigned points.
- Repeating the process until convergence.

After fitting the model, the cluster centroids are extracted using `cluster_centers_`, which represent the central point of each cluster.

[10]:
```
import sklearn.cluster as cluster
kmeans = cluster.KMeans(n_clusters = 5)
```

[11]:
```
kmeans = kmeans.fit(df[['score','income']])
```

[12]:
```
#Finding out the centroids
kmeans.cluster_centers_
```

Output

The output of this step is the coordinates of the 5 cluster centroids in the feature space `(score, income)`. Each row represents the center of a cluster, showing the average `score` and `income` values for that group.

[12]:
```
array([[82.12820513, 86.53846154],
       [49.51851852, 55.2962963 ],
       [79.36363636, 25.72727273],
       [17.11428571, 88.2       ],
       [20.91304348, 26.30434783]])
```

4.7 K-Means Model Training and Cluster Centers Extraction

Explanation

In this step, the K-Means clustering algorithm is applied to the dataset using two selected features: `income` and `score`. The model is trained using the `fit()` function, which groups the data into clusters based on similarity between data points.

After training the model, the `cluster_centers_` attribute is used to retrieve the centroid (center point) of each cluster. These centroids represent the average position of all data points within each cluster and help in understanding the grouping pattern of the dataset.

[13]:
```
kmeans = kmeans.fit(df[['income','score']])
```

[14]:
```
kmeans.cluster_centers_
```

Output

The output of `kmeans.cluster_centers_` is a set of numerical values representing the center of each cluster in the feature space:

[14]:
```
array([[26.30434783, 20.91304348],
       [86.53846154, 82.12820513],
       [88.2       , 17.11428571],
       [25.72727273, 79.36363636],
       [55.2962963 , 49.51851852]])
```

4.8 Assigning K-Means Cluster Labels to Dataset

Explanation

In this step, the cluster labels generated by the K-Means algorithm are assigned to the original dataset. The attribute `kmeans.labels_` contains the cluster index for each data point after training the model.

These labels represent the cluster group to which each record belongs. By storing them in a new column named `income_clusters`, we can easily analyze and visualize the segmented groups within the dataset.

This step is essential for interpreting the clustering results in a structured tabular format.

[15]:
```
df['income_clusters'] = kmeans.labels_
```

[16]:
```
df
```

Output

After executing the code, a new column `income_clusters` is added to the dataset. Each row now includes a cluster label (e.g., 0, 1, 2, etc.), indicating the assigned cluster group.

The updated dataset will look like:

[16]:

	CustomerID	gender	age	income	score	income_clusters
0	1	Male	19	15	39	0
1	2	Male	21	15	81	3
2	3	Female	20	16	6	0
3	4	Female	23	16	77	3
4	5	Female	31	17	40	0
..	…	…	…	…	…	…
195	196	Female	35	120	79	1
196	197	Female	45	126	28	2
197	198	Male	32	126	74	1
198	199	Male	32	137	18	2
199	200	Male	30	137	83	1

[200 rows x 6 columns]

4.9 Cluster Value Count using value_counts()

Explanation

This step is used after applying clustering (e.g., K-Means) on the dataset. The column `income_clusters` contains the cluster labels assigned to each data point.

The function `value_counts()` is used to count how many data points belong to each cluster. This helps in understanding the distribution of customers across different clusters and checking whether the clustering is balanced or not.

[17]:
```
# counting in which cluster how many values
df['income_clusters'].value_counts()
```

Output

The output shows the number of records in each cluster.

[17]:
```
income_clusters
4    81
1    39
2    35
0    23
3    22
Name: count, dtype: int64
```

4.10 K-Means Cluster Visualization using Scatter Plot

Explanation

This code uses Seaborn's `scatterplot` function to visualize the results of K-Means clustering. The dataset (`df`) contains customer information where `score` and `income` are used as feature variables.

The parameter `hue='income_clusters'` is used to color the data points based on the cluster labels generated by the K-Means algorithm. This helps in visually distinguishing different customer groups based on their income and spending behavior.

[18]:
```
seaborn.scatterplot(x= 'score', y = 'income', hue= 'income_clusters', data=␣
↪df)
```

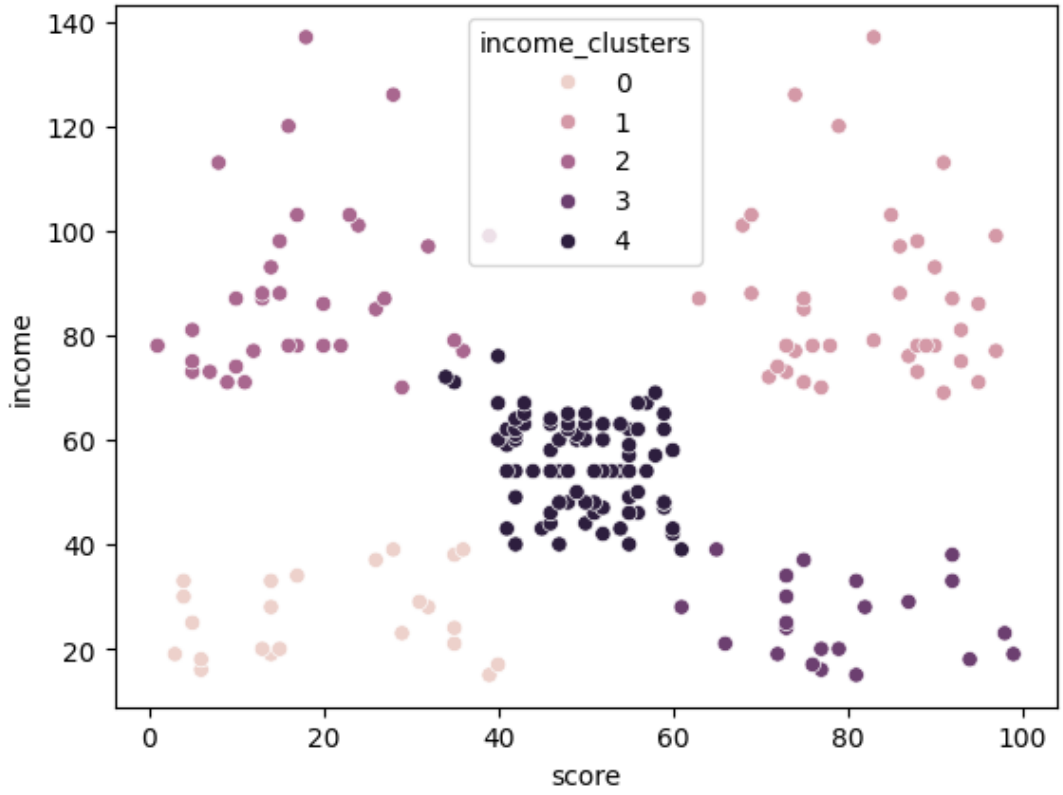
Output

The output is a scatter plot where:

- X-axis represents `score` (customer spending score)
- Y-axis represents `income` (customer income)
- Different colors represent different clusters formed by K-Means

The plot visually shows how customers are grouped into distinct clusters. Customers with similar income and spending patterns appear close together in the same colored group.

[18]:
```
<Axes: xlabel='score', ylabel='income'>
```



4.11 Scatter Plot for Cluster Visualization

Explanation

This code uses the Seaborn library to visualize clustering results using a scatter plot. The plot displays the relationship between `income` and `score` from the dataset, while also differentiating data points based on their assigned cluster labels stored in the column `income_clusters`.

Each cluster is represented by a different color, allowing clear visualization of how customers are grouped based on similarity in income and spending behavior. This helps in understanding the distribution and separation of clusters generated by the K-Means algorithm.

[19]:
```
seaborn.scatterplot(x= 'income', y = 'score', hue= 'income_clusters', data=␣
↪df)
```

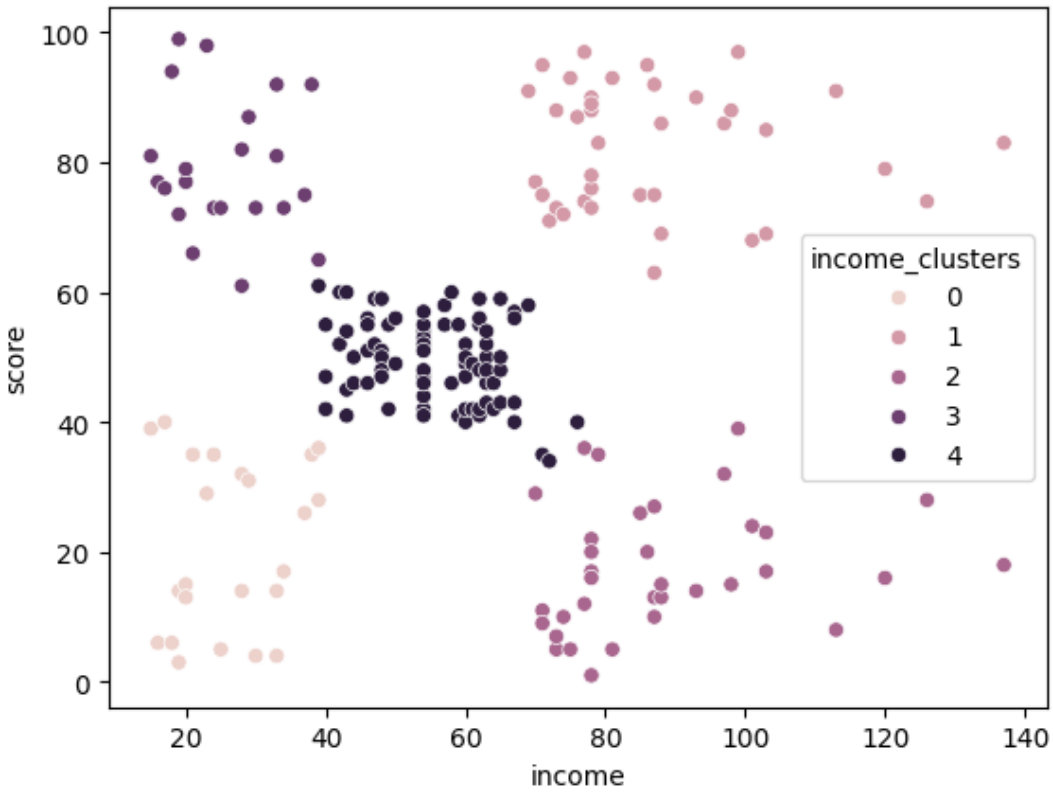
Output

The output is a 2D scatter plot where:

- X-axis represents `income`
- Y-axis represents `score`
- Different colors represent different clusters (`income_clusters`)

The plot visually shows distinct customer groups, indicating how well the clustering algorithm has separated the data into meaningful segments.

[19]:
```
<Axes: xlabel='income', ylabel='score'>
```



4.12 K-Means Clustering with 2 Clusters

Explanation

In this step, the K-Means clustering algorithm is applied to the dataset using two features: `age` and `score`. The number of clusters is set to 2, meaning the algorithm will divide the data into two distinct groups based on similarity.

The model is first initialized with `n_clusters = 2` and then trained using the `fit()` method on the selected columns. After training, the cluster centers are extracted using `cluster_centers_`, which represent the average position (centroid) of each cluster in the feature space.

[20]:
```
#trying with 2 clusters
kmeans = cluster.KMeans(n_clusters = 2)
```

[21]:
```
kmeans = kmeans.fit(df[['age','score']])
```

[22]:
```
kmeans.cluster_centers_
```

Output

The output of this code is the coordinates of the two cluster centers:

[22]:
```
array([[46.16521739, 32.88695652],
       [28.95294118, 73.62352941]])
```

These cluster centers are used to understand how the dataset is grouped into two segments based on similarity in age and score.

4.13 Assigning Cluster Labels to Dataset

Explanation

In this step, the output labels generated by the K-Means clustering algorithm are assigned to the original dataset. The attribute `kmeans.labels_` contains the cluster ID for each data point after model fitting.

These cluster labels represent which group each record belongs to based on similarity in feature space. The labels are stored in a new column called `age_clusters`, allowing easy interpretation and analysis of clustered groups within the dataset.

[23]:
```
df['age_clusters'] = kmeans.labels_
```

[24]:
```
df
```

Output

After executing this code, a new column `age_clusters` is added to the dataset `df`. Each row now contains a cluster number (e.g., 0, 1, 2, etc.), indicating the group assigned by the K-Means algorithm.

[24]:

	CustomerID	gender	age	income	score	income_clusters	age_clusters
0	1	Male	19	15	39	0	0
1	2	Male	21	15	81	3	1
2	3	Female	20	16	6	0	0
3	4	Female	23	16	77	3	1
4	5	Female	31	17	40	0	0
..	…	…	…	…	…	…	
195	196	Female	35	120	79	1	1
196	197	Female	45	126	28	2	0
197	198	Male	32	126	74	1	1
198	199	Male	32	137	18	2	0
199	200	Male	30	137	83	1	1

[200 rows x 7 columns]

The `age_clusters` column helps in understanding customer segmentation based on similarity in features.

4.14 Cluster Distribution Analysis using value-counts()

Explanation

In this step, the distribution of clustered data is analyzed using the `value_counts()` function on the column `age_clusters`. This column represents cluster labels assigned to each data point after applying a clustering algorithm (such as K-Means).

The function counts how many data points belong to each cluster, helping to understand the balance or imbalance of the formed clusters. Additionally, displaying the full dataframe (`df`) allows verification of cluster assignments alongside original features.

[25]:
```
df['age_clusters'].value_counts()
```

Output

The output shows the frequency of each cluster label in the dataset.

[25]:
```
age_clusters
0    115
1     85
Name: count, dtype: int64
```

[26]:
```
df
```

[26]:

	CustomerID	gender	age	income	score	income_clusters	age_clusters
0	1	Male	19	15	39	0	0
1	2	Male	21	15	81	3	1
2	3	Female	20	16	6	0	0
3	4	Female	23	16	77	3	1
4	5	Female	31	17	40	0	0
..	…	…	…	…	…	…	…
195	196	Female	35	120	79	1	1
196	197	Female	45	126	28	2	0
197	198	Male	32	126	74	1	1
198	199	Male	32	137	18	2	0
199	200	Male	30	137	83	1	1

[200 rows x 7 columns]

4.15 Scatter Plot Visualization using Seaborn

Explanation

This code uses the `seaborn.scatterplot()` function to visualize the relationship between `age` and `score` in the dataset. The parameter `hue='age_clusters'` is used to differentiate data points based on cluster labels generated from a clustering algorithm (such as K-Means).

Each cluster is represented by a different color, allowing us to visually identify how data points are grouped based on age similarity and score distribution. This helps in understanding the clustering pattern and separation between different groups.

[27]:
```
seaborn.scatterplot(x= 'age', y = 'score', hue= 'age_clusters', data= df)
```

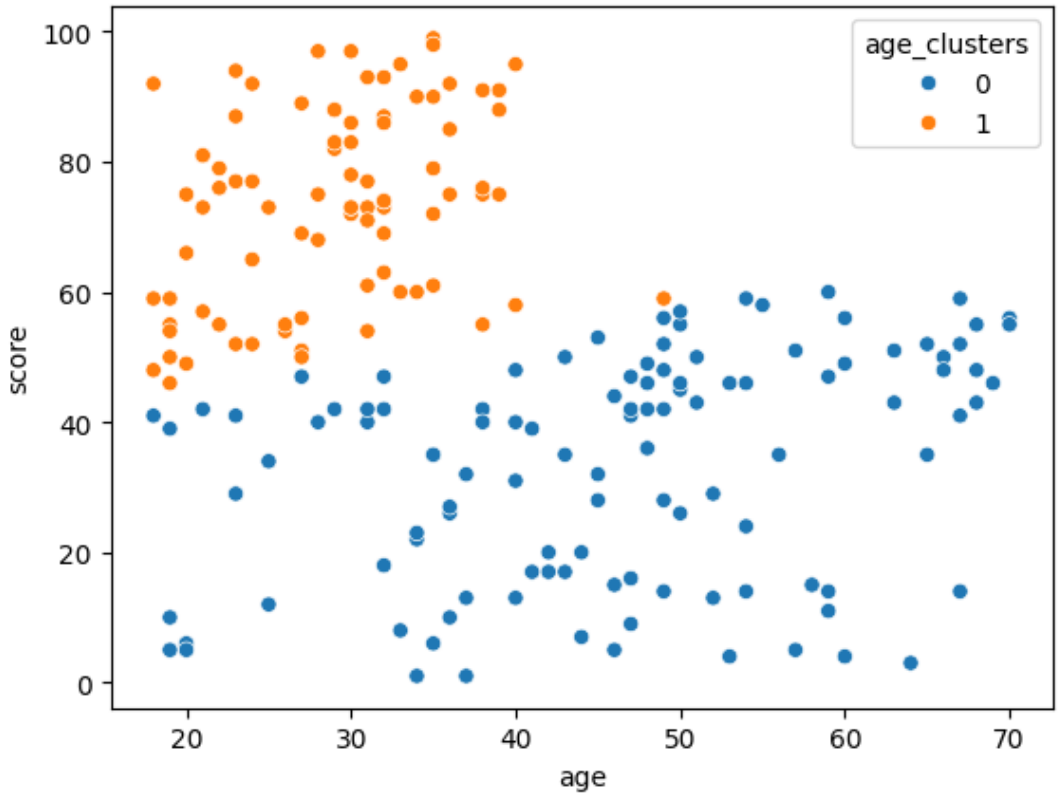
Output

The output is a scatter plot where:

- X-axis represents `age`
- Y-axis represents `score`
- Different colors represent different `age_clusters`

The plot visually shows clustered groups of customers, where similar age groups with similar scores are grouped together, indicating the effectiveness of the clustering algorithm.

[27]:
```
<Axes: xlabel='age', ylabel='score'>
```



4.16 K-Means Clustering (Elbow Method - WCSS Calculation)

Explanation

In this step, the K-Means clustering algorithm is applied to determine the optimal number of clusters for the dataset. The model uses two features: `income` and `score`.

The process calculates the Within-Cluster Sum of Squares (WCSS) for different values of k (number of clusters). WCSS measures how compact the clusters are; lower values indicate better clustering.

A loop is used to test cluster values from 1 to 11. For each value of k, the KMeans algorithm is trained, and the inertia (WCSS value) is stored in a list.

This helps in identifying the "elbow point", where increasing clusters no longer significantly reduces WCSS.

[28]:
```
#from sklearn.cluster import kMeans
from sklearn.cluster import KMeans
```

[29]:
```
# to calculate 12 clusters
k_range = range(1,12)
wcss = []
```

[30]:
```
# implementing for loop for 12 clusters to find out wcss value for each␣
 ↪cluster
# adding wcss values of clusters in wcss array

for k in k_range:
    km= KMeans(n_clusters =k)
    km.fit(df[['income','score']])
    wcss.append (km.inertia_)
```

Output

The output of this step is a list of WCSS values corresponding to different cluster numbers:

```
k = 1  -> WCSS = high value
k = 2  -> WCSS decreases
k = 3  -> WCSS decreases further
...
k = 11 -> WCSS becomes relatively stable
```

These values are later plotted in an Elbow Curve graph to determine the optimal number of clusters. The point where the curve bends (elbow point) indicates the best value of k.

4.17 Showing the values of WCSS (Within-Cluster Sum of Squares)

Explanation

WCSS (Within-Cluster Sum of Squares) is a metric used in K-Means clustering to measure the compactness of clusters. It calculates the sum of squared distances between each data point and the centroid of its assigned cluster. A lower WCSS value indicates that data points are closer to their respective cluster centers, meaning better clustering performance.

WCSS is commonly used in the Elbow Method to determine the optimal number of clusters (K). As the number of clusters increases, WCSS decreases, and the "elbow point" helps identify the most

suitable value of K.

[31]:
```
wcss
```

Output

The output of WCSS is a list or array of values corresponding to different numbers of clusters (K values).

[31]:
```
[269981.28,
 186362.95600651752,
 106348.37306211116,
 73880.64496247197,
 44454.47647967974,
 38797.9027638142,
 31600.209115340018,
 25056.895153616184,
 23311.584210674235,
 21797.817560054325,
 18092.765869660056]
```

4.18 Elbow Method Visualization for Optimal Clusters

Explanation

This code is used to visualize the Elbow Method in K-Means clustering. The Elbow Method helps to determine the optimal number of clusters (k) by plotting the relationship between the number of clusters and the Within-Cluster Sum of Squares (WCSS).

Here:

- `k_range` represents different values of k (number of clusters).
- `wcss` stores the sum of squared distances between data points and their respective cluster centroids.
- The graph shows how WCSS decreases as the number of clusters increases.

The point where the curve starts to bend (forming an "elbow") indicates the optimal number of clusters.

[32]:
```
plt.xlabel ('Number of Clusters(k)')
plt.ylabel ('Sum of squared error')
plt.plot (k_range , wcss)
```

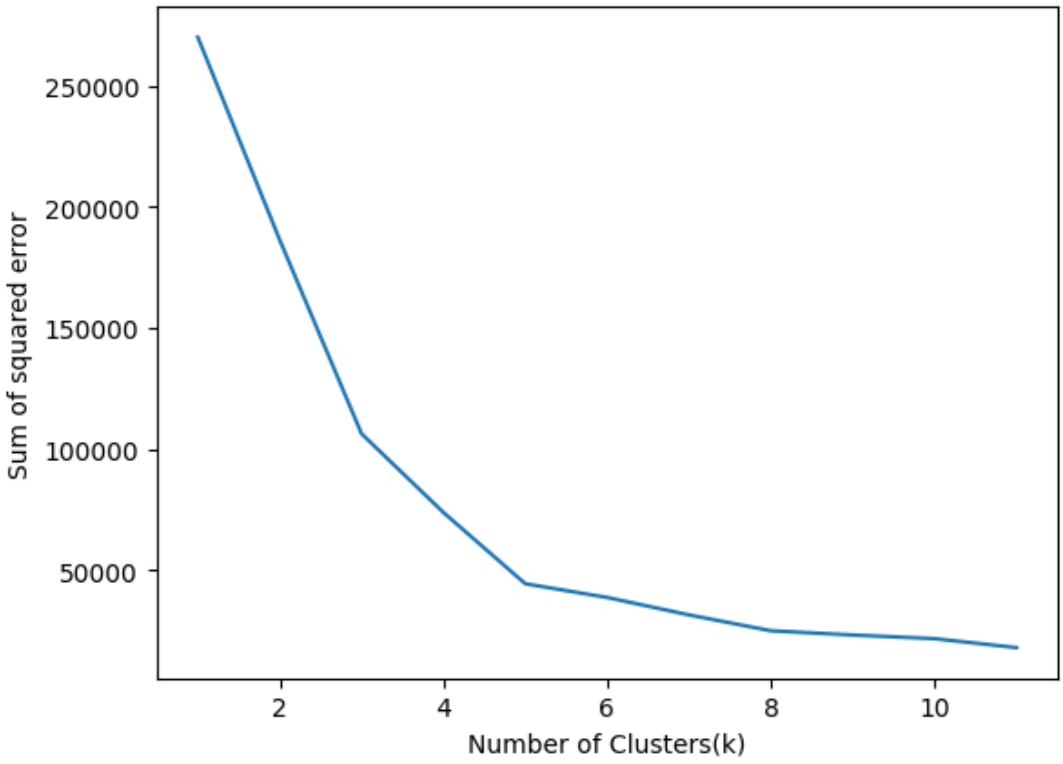
Output

A line plot is generated with:

- X-axis: Number of Clusters (k)
- Y-axis: Sum of Squared Error (WCSS)

The graph typically shows a decreasing curve, and the "elbow point" is used to select the best value of k for clustering.

[32]:
```
[<matplotlib.lines.Line2D at 0x7f616580d910>]
```


250000
200000
150000
100000
50000
Sum of squared error
2
4
6
8
10
Number of Clusters(k)