lab3A

April 7, 2024

1 LAB 03 - Python version

1.0.1 Disclaimer

The purpose of creating this material is to enhance the knowledge of students who are interested in learning how to solve problems presented in laboratory classes using Python. This decision stems from the observation that some students have opted to utilize Python for tackling exam projects in recent years.

To solve these exercises using Python, you need to install Python (version 3.9.6 or later) and some libraries using pip or conda.

Here's a list of the libraries needed for this case:

- `os`: Provides operating system dependent functionality, commonly used for file operations such as reading and writing files, interacting with the filesystem, etc.
- `pandas`: A data manipulation and analysis library that offers data structures and functions to efficiently work with structured data.
- `numpy`: A numerical computing library that provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- `matplotlib.pyplot`: A plotting library for creating visualizations like charts, graphs, histograms, etc.
- `sklearn`: Machine learning algorithms and tools.
- `sklearn_extra`: Additional machine learning algorithms and extensions.
- `nltk`: The Natural Language Toolkit, a library for natural language processing tasks such as tokenization, stemming, part-of-speech tagging, and more.
- `xlrd`: A Python library used for reading data and formatting information from Excel files (.xls and .xlsx formats). It provides functionality to extract data from Excel worksheets, including cells, rows, columns, and formatting details.

You can download Python from here and follow the installation instructions for your operating system.

For installing libraries using pip or conda, you can use the following commands:

- Using pip:

  ```
  pip install pandas numpy matplotlib nltk scikit-learn xlrd scikit-learn-extra
  ```

- Using conda:

  ```
  conda install pandas numpy matplotlib nltk scikit-learn xlrd scikit-learn-extra
  ```

Make sure to run these commands in your terminal or command prompt after installing Python. You can also execute them in a cell of a Jupyter Notebook file (`.ipynb`) by starting the command with '!'.

2 Exercise 1

Import some libraries

[2]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA

from sklearn_extra.cluster import KMedoids
from sklearn.cluster import AgglomerativeClustering
from sklearn.cluster import DBSCAN

from sklearn.metrics import silhouette_score
```

2.1 Read file excel

To read the Excel file using a function integrated into the pandas library, you can use the `pd.read_excel()` function. Rewrite the instruction with the argument as the path of the file to be read

[3]:
```
dataset = pd.read_excel("/Users/luca/Library/Mobile Documents/
↪com~apple~CloudDocs/Business Intelligence per Big Data/Laboratories/LAB03/
↪Lab3Materiale/UsersSmall.xls")
```

In a Jupyter Notebook cell, you can print a subset of the representation by simply calling the name of the variable containing the DataFrame.

[4]:
```
dataset
```

[4]:

	Age	Workclass	Education	Marital Status	Occupation	\
0	-15	State-gov	Bachelors	Never-married	Adm-clerical	
1	150	Private	PhD	Never-married	Exec-managerial	
2	39	State-gov	Bachelors	Never-married	Adm-clerical	
3	50	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	
4	38	Private	HS-grad	Divorced	Handlers-cleaners	
..	…	…	…	…	…	
297	65	Private	HS-grad	Married-civ-spouse	Transport-moving	
298	37	Self-emp-inc	Bachelors	Divorced	Sales	
299	39	?	Masters	Married-civ-spouse	?	

```
300   24           Private    HS-grad       Never-married       Craft-repair
301   38           Private    HS-grad            Divorced              Sales

      Relationship                Race     Sex Native Country  Response
0    Not-in-family               White    Male  United-States  Negative
1                ?               White  Female        Jamaica  Negative
2    Not-in-family               White    Male  United-States  Negative
3          Husband               White    Male  United-States  Negative
4    Not-in-family               White    Male  United-States  Negative
..             …                   …     …            …       …
297        Husband               White    Male  United-States  Negative
298  Not-in-family               White  Female  United-States  Negative
299           Wife  Asian-Pac-Islander  Female              ?  Negative
300      Own-child               White    Male  United-States  Negative
301  Not-in-family               White    Male  United-States  Negative

[302 rows x 10 columns]
```

2.2 How to handle Missing values?

Find if there are missing values.

Usually in a real dataset the missing values are stored with a nan value. In this case we have ? as missing values representation.

So first of all we can replace each '?' symbol in a nan value. The we will apply some important and classical functions.

[5]:
```
dataset.replace(to_replace = '?', value = np.nan, inplace = True)
```

[6]:
```
dataset.isnull().sum() # count the number of missing values for each column
```

[6]:
```
Age               0
Workclass        16
Education         0
Marital Status    0
Occupation       16
Relationship      1
Race              0
Sex               0
Native Country    8
Response          0
dtype: int64
```

As you have seen in class there are different methodologies for filling the nan values. Here we will use the average for the numerical data and the most frequent string for non-numerical columns

[7]:
```
# Replace NaN values with the average value for numerical columns
for col in dataset.select_dtypes(include=np.number).columns:
    dataset[col].fillna(dataset[col].mean(), inplace=True) # Get the average␣
 ↪value for the column and replace NaN values with it

# Replace NaN values with the most frequent value for non-numerical columns
for col in dataset.select_dtypes(exclude=np.number).columns:
    mode_val = dataset[col].mode()[0]  # Get the most frequent string value
    dataset[col].fillna(mode_val, inplace=True) # Get the most frequent value␣
 ↪for the column and replace NaN values with it
```

[8]:
```
dataset
```

[8]:

	Age	Workclass	Education	Marital Status	Occupation	Relationship	Race	Sex	Native Country	Response
0	-15	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	United-States	Negative
1	150	Private	PhD	Never-married	Exec-managerial	Husband	White	Female	Jamaica	Negative
2	39	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	United-States	Negative
3	50	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States	Negative
4	38	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	United-States	Negative
..	…	…	…	…	…	…	…	…	…	…
297	65	Private	HS-grad	Married-civ-spouse	Transport-moving	Husband	White	Male	United-States	Negative
298	37	Self-emp-inc	Bachelors	Divorced	Sales	Not-in-family	White	Female	United-States	Negative
299	39	Private	Masters	Married-civ-spouse	Prof-specialty	Wife	Asian-Pac-Islander	Female	United-States	Negative
300	24	Private	HS-grad	Never-married	Craft-repair	Own-child	White	Male	United-States	Negative
301	38	Private	HS-grad	Divorced	Sales	Not-in-family	White	Male	United-States	Negative

[302 rows x 10 columns]

We can now check the number of missing values in the dataset.

We can see that there are no missing values in the dataset.

[9]:
```
dataset.isnull().sum()
```

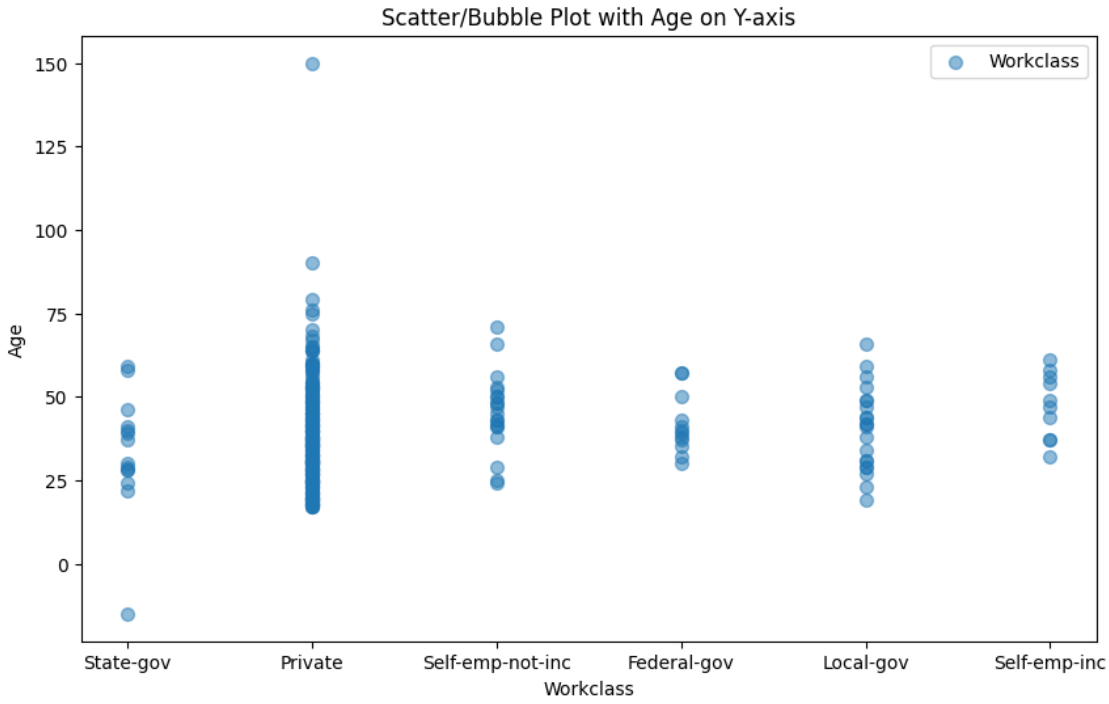
[9]:
```
Age                0
Workclass          0
Education          0
Marital Status     0
Occupation         0
Relationship       0
Race               0
Sex                0
Native Country     0
Response           0
dtype: int64
```

2.3 Outlier detection

You can plot a scatter/bubble plot to identify some outliers

[10]:
```
# Fix the 'Age' attribute on the y-axis
age_values = dataset['Age']

# Plot scatter/bubble plot with an attribute on the x-axis. Ypu caan choose␣
 ↪what ever attribute you want
attribute = "Workclass"
plt.figure(figsize=(10, 6))
plt.scatter(dataset[attribute], age_values, s=50, alpha=0.5, label=attribute)
plt.xlabel(attribute, fontsize=10)  # Adjusted x-axis label font size
plt.ylabel('Age')
plt.title('Scatter/Bubble Plot with Age on Y-axis')
plt.legend()
plt.show()
```



As evident, the 'Age' attribute in our dataset contains errors, such as improbable values like 150 for age or an age less than 0. To ensure the integrity of our data, we need to perform cleaning by filtering out such rows from the dataset.

[11]:
```
condition = (dataset['Age'] >= 0) & (dataset['Age'] < 101) # Get the condition␣
↪for the age values (between 0 and 105 years old)
dataset = dataset[condition].reset_index(drop=True) # Apply the condition to␣
↪the dataset and store the result in the dataset variable
```

2.4 Select attributes

Remove 'Response attribute' (that is in the last column) from the variable dataset

The `.iloc` function in Pandas is used for integer-location based indexing. It allows you to select rows and columns from a DataFrame by their integer position, rather than by label. This function provides a way to select data by position, similar to indexing in NumPy arrays.

2.4.1 Syntax

`DataFrame.iloc[row_indexer, column_indexer]`

- `row_indexer`: Specifies the rows to select. It can be:
  - An integer, e.g., `2`.
  - A list or array of integers, e.g., `[1, 3, 5]`.
  - A slice object with integers, e.g., `1:4`.
  - A boolean array.

- `column_indexer`: Specifies the columns to select. It follows the same rules as `row_indexer`.

2.4.2 Example Usage

```
import pandas as pd

# Creating a sample DataFrame
data = {'A': [1, 2, 3, 4],
        'B': [5, 6, 7, 8],
        'C': [9, 10, 11, 12]}
df = pd.DataFrame(data)

# Selecting specific rows and columns using iloc
selected_data = df.iloc[1:3, 0:2]
print(selected_data)
```

2.4.3 Output

```
   A  B
1  2  6
2  3  7
```

2.4.4 Notes

- `.iloc` is exclusive of the end index when using slices, similar to Python indexing conventions.
- If you want to select specific rows and columns by label instead of position, you should use the `.loc` function.

[12]:
```
dataset = dataset.iloc[:, :-1] # Remove the last column from the dataset
```

[13]:
```
dataset
```

[13]:

	Age	Workclass	Education	Marital Status	Occupation	\
0	39	State-gov	Bachelors	Never-married	Adm-clerical	
1	50	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	
2	38	Private	HS-grad	Divorced	Handlers-cleaners	
3	53	Private	11th	Married-civ-spouse	Handlers-cleaners	
4	28	Private	Bachelors	Married-civ-spouse	Prof-specialty	
..	…	…	…	…	…	
295	65	Private	HS-grad	Married-civ-spouse	Transport-moving	
296	37	Self-emp-inc	Bachelors	Divorced	Sales	
297	39	Private	Masters	Married-civ-spouse	Prof-specialty	
298	24	Private	HS-grad	Never-married	Craft-repair	
299	38	Private	HS-grad	Divorced	Sales	

	Relationship	Race	Sex	Native Country
0	Not-in-family	White	Male	United-States
1	Husband	White	Male	United-States
2	Not-in-family	White	Male	United-States

```
3          Husband                Black    Male  United-States
4             Wife                Black  Female           Cuba
..             …                    …     …              …
295        Husband                White    Male  United-States
296  Not-in-family                White  Female  United-States
297           Wife  Asian-Pac-Islander  Female  United-States
298      Own-child                White    Male  United-States
299  Not-in-family                White    Male  United-States

[300 rows x 9 columns]
```

2.5 Normalization

Normalize age attribute

[14]:
```
# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Normalize the 'age' attribute
dataset['Age'] = scaler.fit_transform(dataset['Age'].values.reshape(-1, 1)) #␣
 ↪Standardize the 'Age' column
```

Age in range [0-1]

[15]:
```
dataset
```

[15]:
```
          Age         Workclass  Education      Marital Status  \
0    0.301370         State-gov  Bachelors       Never-married
1    0.452055  Self-emp-not-inc  Bachelors  Married-civ-spouse
2    0.287671           Private    HS-grad            Divorced
3    0.493151           Private       11th  Married-civ-spouse
4    0.150685           Private  Bachelors  Married-civ-spouse
..        …                 …        …                  …
295  0.657534           Private    HS-grad  Married-civ-spouse
296  0.273973      Self-emp-inc  Bachelors            Divorced
297  0.301370           Private    Masters  Married-civ-spouse
298  0.095890           Private    HS-grad       Never-married
299  0.287671           Private    HS-grad            Divorced

            Occupation   Relationship                Race     Sex  \
0         Adm-clerical  Not-in-family               White    Male
1      Exec-managerial        Husband               White    Male
2    Handlers-cleaners  Not-in-family               White    Male
3    Handlers-cleaners        Husband               Black    Male
4       Prof-specialty           Wife               Black  Female
..                 …            …                  …     …
295   Transport-moving        Husband               White    Male
```

```
296              Sales  Not-in-family                White  Female
297     Prof-specialty           Wife  Asian-Pac-Islander  Female
298       Craft-repair      Own-child                White    Male
299              Sales  Not-in-family                White    Male

    Native Country
0    United-States
1    United-States
2    United-States
3    United-States
4             Cuba
..             …
295  United-States
296  United-States
297  United-States
298  United-States
299  United-States

[300 rows x 9 columns]
```

2.6 K-Medoids

The k-Medoids algorithm is a clustering algorithm that partitions a dataset into k clusters, where each cluster is represented by one of its data points, known as a medoid. Unlike k-Means, which uses centroids (the mean of the data points in each cluster), k-Medoids uses medoids, making it more robust to outliers.

LabelEncoder, instead, converts categorical labels into numeric representations, essential for machine learning models. Used for transforming target variables or encoding categorical features.

[16]:
```python
# copy the dataset
dataset_copy = dataset.copy()

label_encoder = LabelEncoder()
for column in dataset_copy.select_dtypes(include=['object']).columns:
    dataset_copy[column] = label_encoder.fit_transform(dataset_copy[column])

# Apply K-Medoids Algorithm

# Instantiate the KMedoids object
kmedoids = KMedoids(n_clusters=2, random_state=0)

# Fit the k-medoids model to the dataset_copy
cluster_labels = kmedoids.fit_predict(dataset_copy)
# Add the cluster labels to the dataset_copy
dataset_copy['cluster'] = cluster_labels

cluster_labels
```

[16]:
```
array([1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0])
```

Silhouette score is a metric used to evaluate the quality of clustering in unsupervised learning. It quantifies how similar an object is to its own cluster compared to other clusters. The silhouette score ranges from -1 to 1.

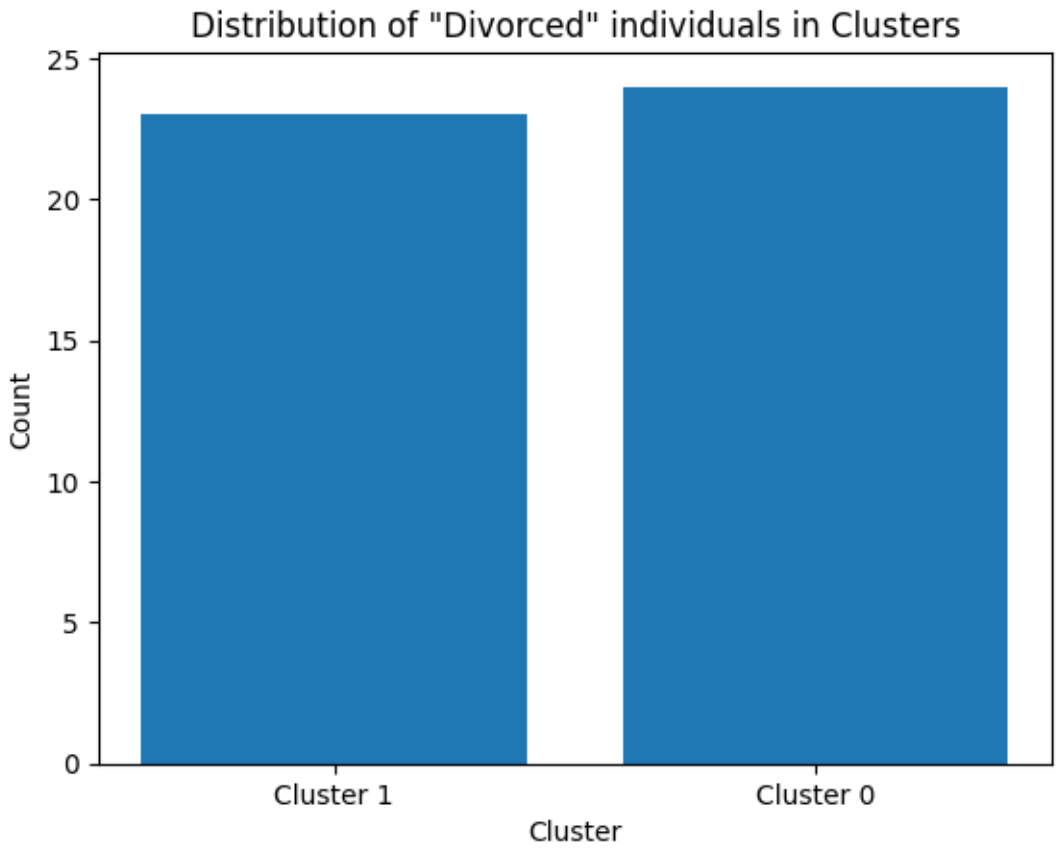
[17]:
```
silhouette_score(dataset_copy, cluster_labels)
```

[17]:
```
0.25922508530852917
```

Visualize the distribution of people with Marital-Status “Divorced” within the clusters

[18]:
```
# Please pay attention that is not an error using dataset and dataset_copy in␣
 ↪the this row because we are using the same dataset
# but with different transformations, infact in dataset variable (as you can␣
 ↪see using a print statement) we have the original dataset woth the string␣
 ↪values,
# instead in the dataset copy we have the dataset with the label encoded values␣
 ↪(intneger values)
divorced_counts = dataset_copy[dataset['Marital Status'] ==␣
 ↪'Divorced']['cluster'].value_counts()

# Plotting
plt.bar(divorced_counts.index, divorced_counts.values, tick_label=['Cluster 0',␣
 ↪'Cluster 1'])
plt.title('Distribution of "Divorced" individuals in Clusters')
plt.xlabel('Cluster')
plt.ylabel('Count')
plt.show()
```



Applying `get_dummies` to our dataset:

Now, let's apply `get_dummies` to our dataset to convert categorical variables into dummy variables. This allows us to use these variables in our k-Medoids clustering algorithm.

`get_dummies` is a function in pandas library used for converting categorical variables into dummy/indicator variables. When applied to a DataFrame column containing categorical data, it creates new binary columns for each category present in the original column. Each binary column indicates whether a particular category is present or not for each row in the dataset.

For example, consider a column "Color" with categories "Red", "Green", and "Blue". After applying `get_dummies`, the DataFrame will have three new columns: "Color_Red", "Color_Green", and "Color_Blue". For each row, only one of these columns will have a value of 1, indicating the presence of that color, while the others will have a value of 0.

[19]:
```
# copy the dataset
dataset_copy = dataset.copy()
dataset_copy = pd.get_dummies(dataset_copy)

# Apply K-Medoids Algorithm

# Instantiate the KMedoids object
```

```
kmedoids = KMedoids(n_clusters=2, random_state=0)

# Fit the k-medoids model to the dataset_copy
cluster_labels = kmedoids.fit_predict(dataset_copy)
cluster_labels
```

[19]:
```
array([0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0,
       1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1])
```

Silhouette score is a metric used to evaluate the quality of clustering in unsupervised learning. It quantifies how similar an object is to its own cluster compared to other clusters. The silhouette score ranges from -1 to 1.

[20]:
```
silhouette_score(dataset_copy, cluster_labels)
```

[20]:
```
0.1126906129210901
```

2.7 Agglomerative Clustering

Agglomerative clustering is a strategy of hierarchical clustering. Hierarchical clustering (also known as Connectivity based clustering) is a method of cluster analysis which seeks to build a hierarchy of clusters. Hierarchical clustering, is based on the core idea of objects being more related to nearby objects than to objects farther away.

[21]:
```
dataset_copy = dataset.copy()

label_encoder = LabelEncoder()
for column in dataset_copy.select_dtypes(include=['object']).columns:
    dataset_copy[column] = label_encoder.fit_transform(dataset_copy[column])

agglomerative = AgglomerativeClustering(n_clusters=2)
# Fit the agglomerative clustering model to the data
cluster_labels = agglomerative.fit(dataset_copy)
cluster_labels.labels_
```

[21]:
```
array([0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Silhouette score is a metric used to evaluate the quality of clustering in unsupervised learning. It quantifies how similar an object is to its own cluster compared to other clusters. The silhouette score ranges from -1 to 1.

[22]:
```
silhouette_score(dataset_copy, cluster_labels.labels_)
```

[22]:
```
0.42738510305834493
```

Applying `get_dummies` to our dataset:

Now, let's apply `get_dummies` to our dataset to convert categorical variables into dummy variables. This allows us to use these variables in our clustering algorithm.

`get_dummies` is a function in pandas library used for converting categorical variables into dummy/indicator variables. When applied to a DataFrame column containing categorical data, it creates new binary columns for each category present in the original column. Each binary column indicates whether a particular category is present or not for each row in the dataset.

For example, consider a column "Color" with categories "Red", "Green", and "Blue". After applying `get_dummies`, the DataFrame will have three new columns: "Color_Red", "Color_Green", and "Color_Blue". For each row, only one of these columns will have a value of 1, indicating the presence of that color, while the others will have a value of 0.

[23]:
```
dataset_copy = dataset.copy()
dataset_copy = pd.get_dummies(dataset_copy)

agglomerative = AgglomerativeClustering(n_clusters=2)
# Fit the agglomerative clustering model to the data
cluster_labels = agglomerative.fit(dataset_copy)
cluster_labels.labels_
```

[23]:
```
array([0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1,
```

```
0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1,
0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0,
0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0,
1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0])
```

Silhouette score is a metric used to evaluate the quality of clustering in unsupervised learning. It quantifies how similar an object is to its own cluster compared to other clusters. The silhouette score ranges from -1 to 1.

[24]:
```
silhouette_score(dataset_copy, cluster_labels.labels_)
```

[24]:
```
0.15493481321187733
```