



Fall 2023

CIS 660 - DATA MINING

PROJECT REPORT FOR CRIME FORCASTING USING
CLUSTERING TECHNIQUES

Prof. Sunnie S. Chung

NAME: Dhrumil Shah

CSU ID: 2860168

NAME: Kajal Shrivastava

CSU ID: 2861721

1. Platform/System Set Up Procedures/Instructions

- For this project, we have used the **Python programming language(version 3.11.4)** for coding.

Downloading link:

<https://www.python.org/downloads/release/python-3114/>

```
(base) C:\Users\shahd>python --version
Python 3.11.4
```

- Installed **Anaconda** launched **Jupyter Notebook(version 6.5.4)** platform for executing all operations, which provides a versatile and interactive setting for data preprocessing and data analysis.

Downloading link:

<https://www.anaconda.com/open-source>

```

Anaconda Prompt
(base) C:\Users\shahd>cd .ipynb_checkpoints
(base) C:\Users\shahd\.ipynb_checkpoints>anaconda --version
anaconda Command line client (version 1.12.0)
(base) C:\Users\shahd\.ipynb_checkpoints>jupyter --version
Selected Jupyter core packages...
IPython          : 8.12.0
ipykernel        : 6.19.2
ipywidgets       : 8.0.4
jupyter_client   : 7.4.9
jupyter_core     : 5.3.0
jupyter_server   : 1.23.4
jupyterlab       : 3.6.3
nbclient         : 0.5.13
nbconvert        : 6.5.4
nbformat         : 5.7.0
notebook         : 6.5.4
qtconsole        : 5.4.2
traitlets        : 5.7.1
(base) C:\Users\shahd\.ipynb_checkpoints>pip show pandas
Name: pandas
Version: 1.5.3
Summary: Powerful data structures for data analysis, time series, and statistics
Home-page: https://pandas.pydata.org
Author: The Pandas Development Team
Author-email: pandas-dev@python.org
License: BSD-3-Clause
Location: C:\Users\shahd\anaconda3\Lib\site-packages
Requires: numpy, numpy, python-dateutil, pytz
Required-by: bokeh, datashader, holoviews, hvplot, panel, seaborn, statsmodels, xarray
```

- Modules that are used: **sklearn.cluster**, and **sklearn.mixture**.
- Libraries such as **Panda**, **Numpy**, **Scikit-learn**, **Matplotlib**, and **Sci-py**.
You can check the version for every single libraries below:

```
Anaconda Prompt

(base) C:\Users\shahd\.ipynb_checkpoints>pip show numpy
Name: numpy
Version: 1.24.3
Summary: Fundamental package for array computing in Python
Home-page: https://www.numpy.org
Author: Travis E. Oliphant et al.
Author-email:
License: BSD-3-Clause
Location: C:\Users\shahd\anaconda3\Lib\site-packages
Requires:
Required-by: astropy, bokeh, Bottleneck, contourpy, daal4py, datashader, datashape, fastcluster, gensim, h5py, holoviews, hvplot, hyperopt, imagecodecs, ima
geio, imbalanced-learn, matplotlib, mkl-fft, mkl-random, ml-dtypes, numba, numexpr, opt-einsum, pandas, patsy, pyarrow, pyerfa, PyWavelets, scikit-image, sc
ikit-learn, scipy, seaborn, statsmodels, tables, tensorboard, tensorflow-intel, tifffile, transformers, xarray

(base) C:\Users\shahd\.ipynb_checkpoints>pip show matplotlib
Name: matplotlib
Version: 3.7.1
Summary: Python plotting package
Home-page: https://matplotlib.org
Author: John D. Hunter, Michael Droettboom
Author-email: matplotlib-users@python.org
License: PSF
Location: C:\Users\shahd\anaconda3\Lib\site-packages
Requires: contourpy, cycler, fonttools, kiwisolver, numpy, packaging, pillow, pyparsing, python-dateutil
Required-by: seaborn

(base) C:\Users\shahd\.ipynb_checkpoints>pip show scipy
Name: scipy
Version: 1.10.1
Summary: SciPy: Scientific Library for Python
Home-page: https://www.scipy.org
Author:
Author-email:
License: BSD
Location: C:\Users\shahd\anaconda3\Lib\site-packages
Requires: numpy
Required-by: datashader, gensim, hyperopt, imbalanced-learn, scikit-image, scikit-learn, statsmodels

(base) C:\Users\shahd\.ipynb_checkpoints>
```

```
(base) C:\Users\shahd\.ipynb_checkpoints>pip show scikit-learn
Name: scikit-learn
Version: 1.3.2
Summary: A set of python modules for machine learning and data mining
Home-page: http://scikit-learn.org
Author:
Author-email:
License: new BSD
Location: C:\Users\shahd\anaconda3\Lib\site-packages
Requires: joblib, numpy, scipy, threadpoolctl
Required-by: daal4py, imbalanced-learn, scikit-learn-intelex
```

- We have downloaded the data from the **NIJ(National Institute of Justice)**. Below is the link from where we have downloaded the dataset.

<https://nij.ojp.gov/funding/real-time-crime-forecasting-challenge-posting#data>

NIJ2015_Jan01_Dec31

[2015, full year \(zip, 13.8 MB\)](#) — posted September 1, 2016

2. Evaluation and Visualization of the Results

```
[36]: import pandas as pd
      from sklearn.model_selection import train_test_split
      from sklearn.neighbors import KNeighborsClassifier
      from sklearn.metrics import accuracy_score

      X = df[toconsider[1:]]
      y = df[toconsider[0]]
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
      knn = KNeighborsClassifier(n_neighbors=7)
      knn.fit(X_train, y_train)
      y_pred = knn.predict(X_test)
      accuracy = accuracy_score(y_test, y_pred)
      print(f'Accuracy: {accuracy * 100:.2f}%')
```

Accuracy: 81.38%

Accuracy of KNN algorithm for this model is 81.38% for k=7, which was obtained from the Elbow method.

Note: Accuracy= Total Number of Predictions/Number of Correct Predictions

```
[35]: k_means_7 = KMeans(n_clusters=7)
      model = k_means_7.fit(N)
      y_hat_7 = k_means_7.predict(N)
      labels_7 = k_means_7.labels_
      print(metrics.silhouette_score(N, labels_7, metric = 'euclidean'))
      #print(metrics.calinski_harabasz_score(X, labels_7))

C:\Users\shahd\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning:
change from 10 to 'auto' in 1.4. Set the value of 'n_init' explicitly to suppress the wa
super()._check_params_vs_input(X, default_n_init=10)

0.4834387897465314
```

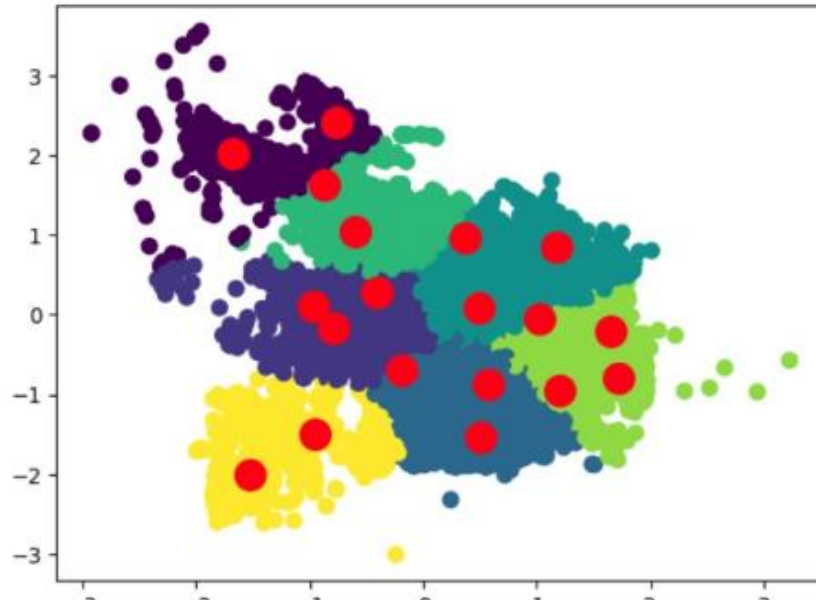
Silhouette score for K_means is 0.4834 for cluster = 7.

Note: Silhouette Coefficient or silhouette score is a metric used to calculate the goodness of a clustering technique. Its value ranges from -1 to 1. 1: Means clusters are well apart from each other and clearly distinguished.

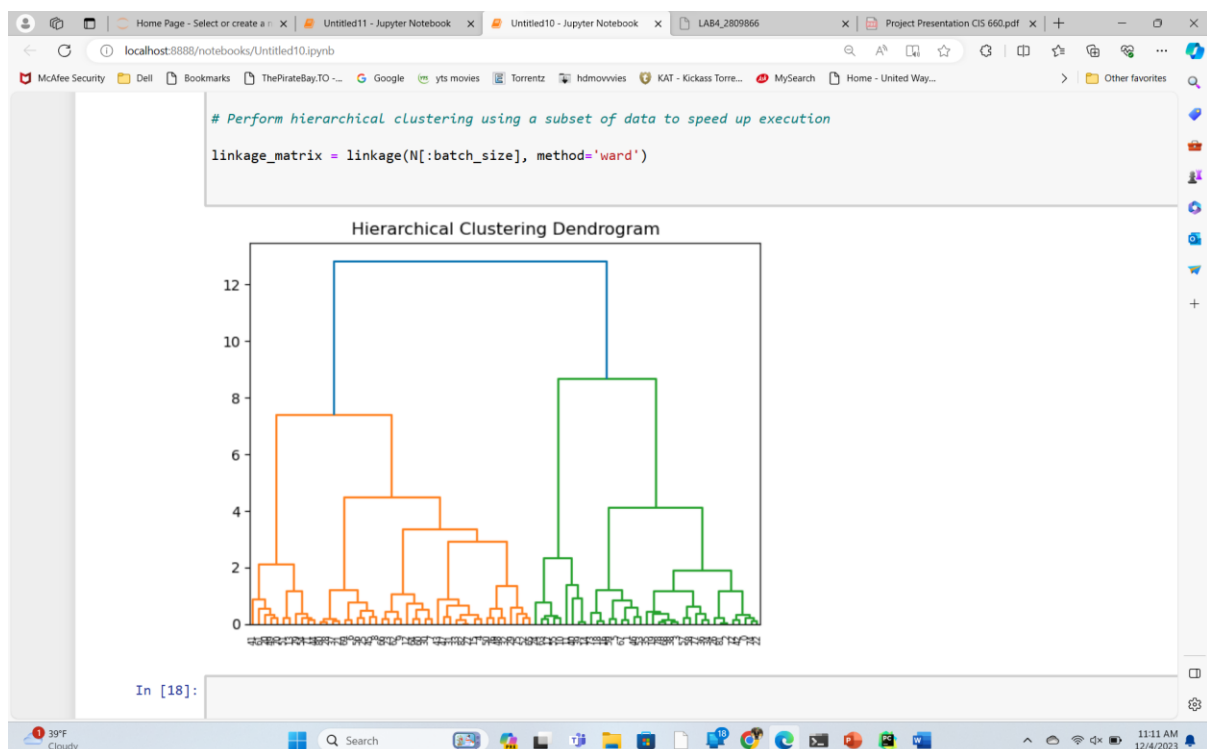
```
In [37]: plt.scatter(N[:,0],N[:,1],c=labels_7,s=50,cmap='viridis')

centers = k_means.cluster_centers_
plt.scatter(centers[:,0],centers[:,1],c='red',s=200,alpha=1)

Out[37]: <matplotlib.collections.PathCollection at 0x1cb1f009110>
```

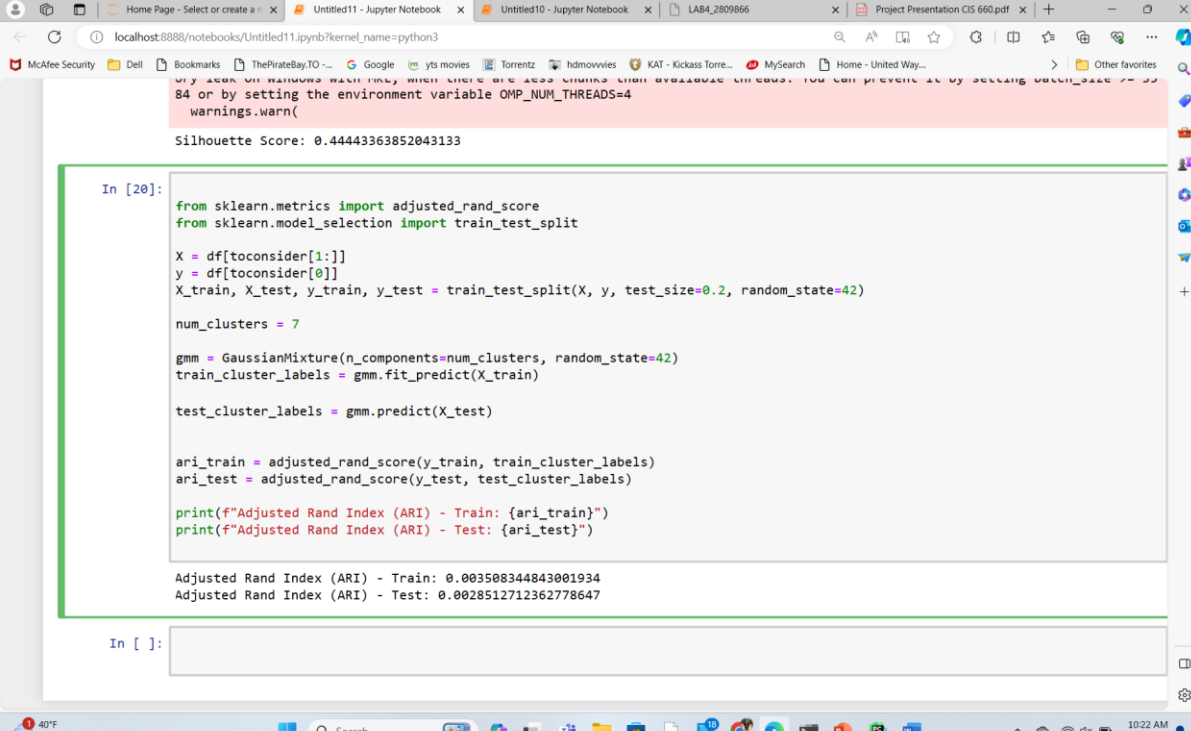


Scatter plot for KNN algorithm is shown above.



Dendrogram for Hierarchical clustering.

Note: Dendrogram illustrates hierarchical relationships between data points, used in clustering analysis to show how groups are formed. A scatter plot, on the other hand, displays individual data points on a two-dimensional graph, emphasizing the relationship between two variables. While a dendrogram shows hierarchical clustering structure, a scatter plot reveals patterns, correlations, and distributions in the data.



The screenshot shows a Jupyter Notebook running on a local host. The notebook contains a Python script that performs Gaussian Mixture Model (GMM) clustering and calculates the Adjusted Rand Index (ARI) for both training and testing data. The script imports necessary libraries, splits the data into training and testing sets, fits a GMM to the training data, and then predicts cluster labels for both sets. Finally, it prints the ARI values for the training and testing data.

```
In [20]:  
from sklearn.metrics import adjusted_rand_score  
from sklearn.model_selection import train_test_split  
  
X = df[toconsider[1:]]  
y = df[toconsider[0]]  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  
  
num_clusters = 7  
  
gmm = GaussianMixture(n_components=num_clusters, random_state=42)  
train_cluster_labels = gmm.fit_predict(X_train)  
  
test_cluster_labels = gmm.predict(X_test)  
  
ari_train = adjusted_rand_score(y_train, train_cluster_labels)  
ari_test = adjusted_rand_score(y_test, test_cluster_labels)  
  
print(f"Adjusted Rand Index (ARI) - Train: {ari_train}")  
print(f"Adjusted Rand Index (ARI) - Test: {ari_test}")  
  
Adjusted Rand Index (ARI) - Train: 0.003508344843001934  
Adjusted Rand Index (ARI) - Test: 0.0028512712362778647
```

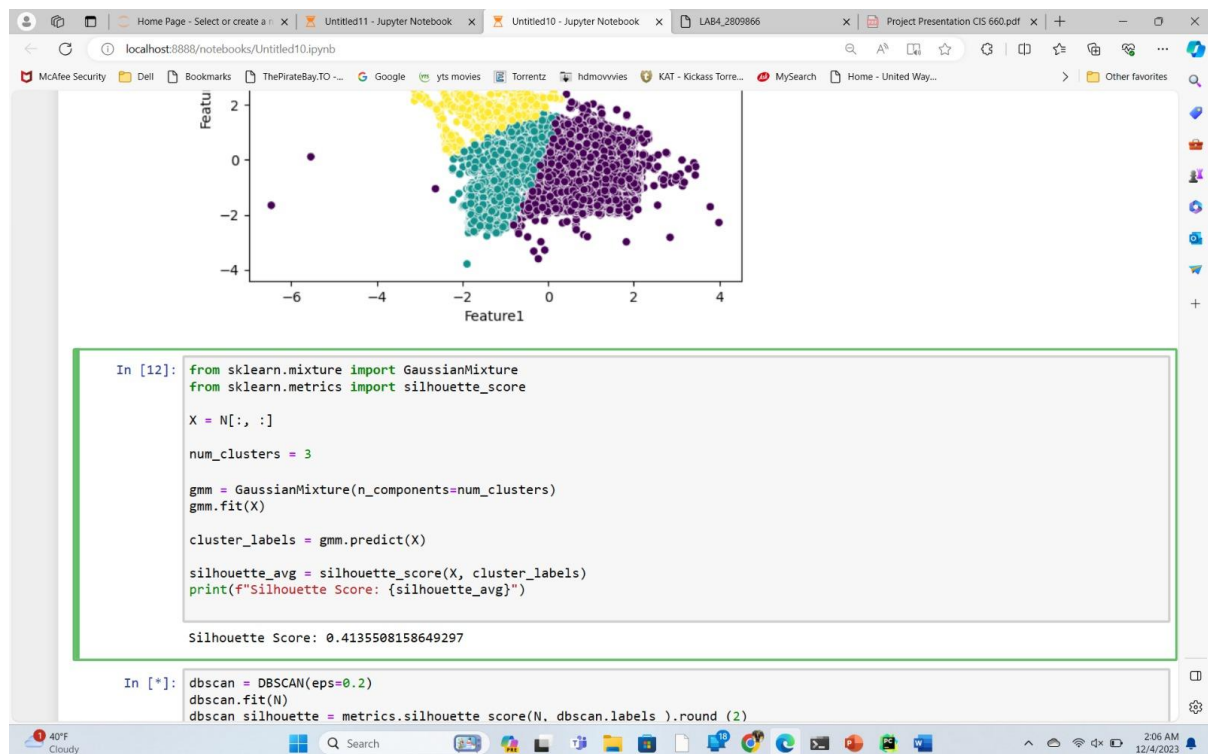
The output of the script is displayed below the code cell, showing the ARI values for the training and testing data.

```
In [ ]:
```

ARI test: 0.002851

ARI train: 0.003580

Note: Adjusted Rand Index (ARI), it is a measure of the similarity between two clustering results, considering both the agreement and disagreement between clusters. ARI values range from -1 to 1, where 1 indicates perfect agreement, 0 indicates a random clustering, and negative values suggest disagreement.



Silhouette score Gaussian algorithm for is 0.4135508

While presentation we got to learn, the major element of this project, which is to get the type and frequency of crime with respect to the specific area. Since, we took according to the coordinates we were not able to specify the result according to the area.

The KNN algorithm, when applied with $k=7$ on the National Institute of Justice dataset, proves its accuracy by correctly classifying instances 81.38% of the time. The silhouette score of 0.4834 for KNN indicates that it's good at creating clusters that are well-defined and separate. On the other hand, the Gaussian algorithm, while also effective, has a slightly lower silhouette score of 0.4135508.

The scatter plot for KNN gives us a visual representation of how it groups the data, helping us understand its patterns. Meanwhile, in hierarchical clustering, the dendrogram shows us the hierarchical relationships within the data, providing additional insights.

However, when we look at the ARI values for the Gaussian algorithm (0.002851 for the test set and 0.003580 for the training set), we notice that there's only a small agreement between the predicted clusters and the true clusters. This suggests that the Gaussian algorithm might need some adjustments to better capture the underlying structure of the data.

In summary, KNN performs well with high accuracy and clear clusters, while the Gaussian algorithm shows reasonable performance but could benefit from further refinement. The visual aids, like scatter plots and dendrograms, make it easier to grasp how these algorithms behave with the given dataset.

3. Execution Steps, all the source code scripts, all intermediate outputs, and final outputs.

Step 1: Importing the essential libraries

Step2: Importing and Reading the obtained csv file of the dataset that is being considered.

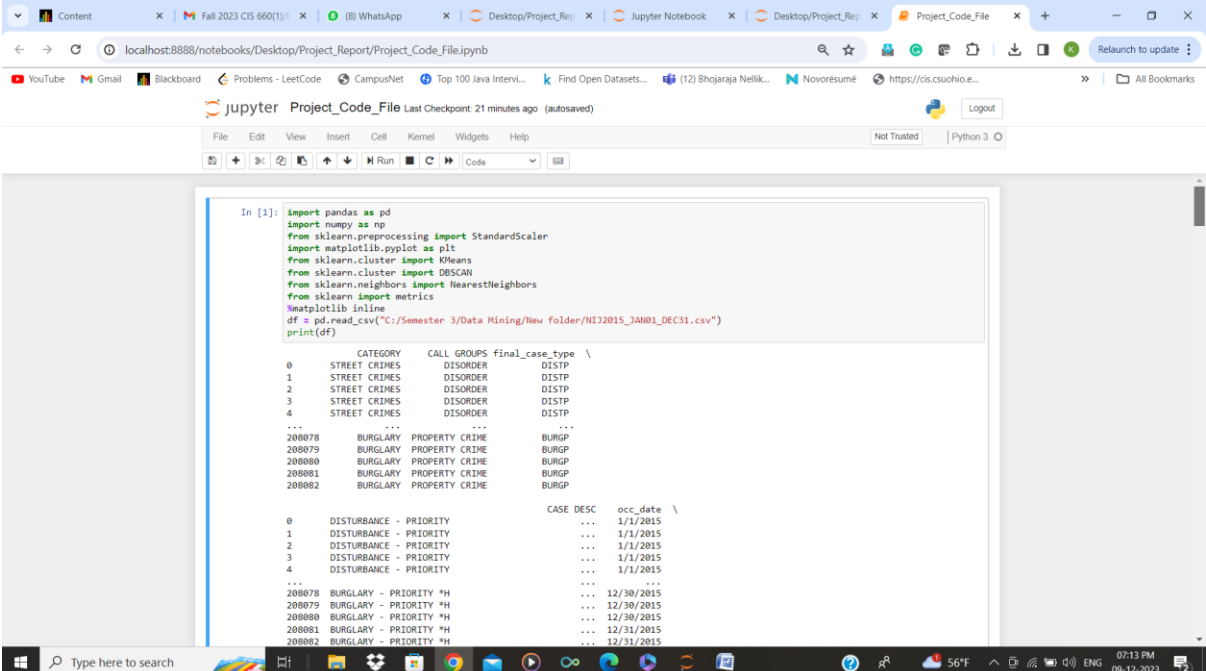
Step 3: Preprocessing the Data with the use of certain methods such as:

DATA PREPROCESSING METHODS USED:

1. Handling NULL values:
 - Managing missing or null values in a dataset.
 - Replacing null values with estimated or calculated values (mean, median, mode).
2. FEATURE SELECTION:
 - In the feature selection, we will be selecting the relevant features from the set of all the features, that could impact the label.
3. BINARIZATION and STANDARDIZATION:
 - Binarization: Converting numerical values into binary values (0 or 1) based on a threshold.
 - Standardization: It Ensures all features have a similar scale, preventing dominance by features with larger numeric ranges.
4. NORMALIZATION:
 - Normalization will convert the data in the range of 0 to 1 which will give the same scale to every features. This will enhance the model performance.
5. Handling OUTLIERS:
 - It refers to the process of identifying and managing data points that deviate significantly from the majority of the observations in a dataset.
 - Common methods include z-scores, the IQR (Interquartile Range) method, or visualizations such as box plots and scatter plots.

Code Script:

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
from sklearn import metrics
%matplotlib inline
df = pd.read_csv("C:/Semester 3/Data Mining/New
folder/NIJ2015_JAN01_DEC31.csv")
print(df)
```



The screenshot shows a Jupyter Notebook window with the following code in the first cell:

```
In [1]: import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
from sklearn import metrics
%matplotlib inline
df = pd.read_csv("C:/Semester 3/Data Mining/New folder/NIJ2015_JAN01_DEC31.csv")
print(df)
```

The output of the code is a preview of the CSV file, showing two tables of data. The first table has columns: CATEGORY, CALL GROUPS, final_case_type, and \. The second table has columns: CASE DESC, occ_date, and \.

	CATEGORY	CALL GROUPS	final_case_type	\
0	STREET CRIMES	DISORDER	DISTP	
1	STREET CRIMES	DISORDER	DISTP	
2	STREET CRIMES	DISORDER	DISTP	
3	STREET CRIMES	DISORDER	DISTP	
4	STREET CRIMES	DISORDER	DISTP	
...
208078	BURGLARY	PROPERTY CRIME	BURGP	
208079	BURGLARY	PROPERTY CRIME	BURGP	
208080	BURGLARY	PROPERTY CRIME	BURGP	
208081	BURGLARY	PROPERTY CRIME	BURGP	
208082	BURGLARY	PROPERTY CRIME	BURGP	

	CASE DESC	occ_date	\
0	DISTURBANCE - PRIORITY	1/1/2015	
1	DISTURBANCE - PRIORITY	1/1/2015	
2	DISTURBANCE - PRIORITY	1/1/2015	
3	DISTURBANCE - PRIORITY	1/1/2015	
4	DISTURBANCE - PRIORITY	1/1/2015	
...
208078	BURGLARY - PRIORITY *H	12/30/2015	
208079	BURGLARY - PRIORITY *H	12/30/2015	
208080	BURGLARY - PRIORITY *H	12/30/2015	
208081	BURGLARY - PRIORITY *H	12/31/2015	
208082	BURGLARY - PRIORITY *H	12/31/2015	

Code Script:

```
df.isnull().sum()
```

```
0      DISTURBANCE - PRIORITY    ...  1/1/2015
1      DISTURBANCE - PRIORITY    ...  1/1/2015
2      DISTURBANCE - PRIORITY    ...  1/1/2015
3      DISTURBANCE - PRIORITY    ...  1/1/2015
4      DISTURBANCE - PRIORITY    ...  1/1/2015
...
200078  BURGLARY - PRIORITY *H    ...  12/30/2015
200079  BURGLARY - PRIORITY *H    ...  12/30/2015
200080  BURGLARY - PRIORITY *H    ...  12/30/2015
200081  BURGLARY - PRIORITY *H    ...  12/31/2015
200082  BURGLARY - PRIORITY *H    ...  12/31/2015

...
0      x_coordinate  y_coordinate  census_tract
0      7624403    663901    30502.0
1      7626061    707425    4102.0
2      7626432    655739    6501.0
3      7638375    689076    4500.0
4      7648361    706901    3901.0
...
200078  7674417    685601    8100.0
200079  7688761    683719    9301.0
200080  7689764    675992    9101.0
200081  7671619    666586    602.0
200082  7680989    679340    9201.0

[200003 rows x 8 columns]
```

```
In [2]: df.isnull().sum()
Out[2]: CATEGORY      0
CALL_GROUPS      0
final_case_type    0
CASE_DESC      0
occ_date      0
x_coordinate      0
y_coordinate      0
census_tract      9830
dtype: int64
```

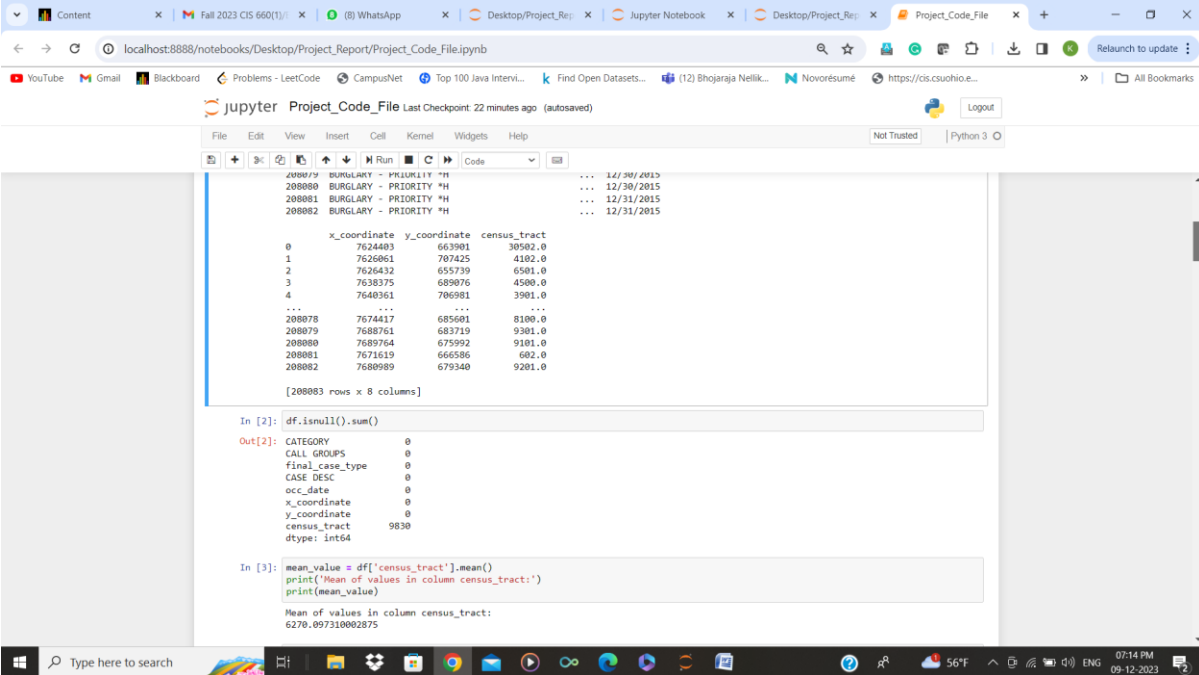
Here,

isnull() function is used to identify missing values in a DataFrame. This method is used to create a boolean mask of the same shape as the DataFrame, where each element is True if the corresponding element in the DataFrame is None or NaN, and False otherwise.

sum(): This method, when applied to a boolean mask, treats True as 1 and False as 0. When you use sum() on a boolean mask created by isnull(), it effectively counts the number of True values, which corresponds to the number of missing values in each column.

Code Script:

```
mean_value = df['census_tract'].mean()
print('Mean of values in column census_tract:')
print(mean_value)
```



The screenshot shows a Jupyter Notebook window titled 'Project_Code_File'. The notebook contains a data script and its output. The script defines a DataFrame with columns: 'census_tract', 'x_coordinate', 'y_coordinate', 'occ_date', 'final_case_type', 'CASE_DESC', 'CALL_GROUPS', and 'CATEGORY'. The output shows the first few rows of the DataFrame, followed by the result of the `df.isnull().sum()` operation, which shows zero missing values for all columns. Finally, the script calculates the mean of the 'census_tract' column, resulting in 6270.097310002875.

```
200079 BURGLARY - PRIORITY *H ... 12/30/2015
200080 BURGLARY - PRIORITY *H ... 12/30/2015
200081 BURGLARY - PRIORITY *H ... 12/31/2015
200082 BURGLARY - PRIORITY *H ... 12/31/2015

   x_coordinate  y_coordinate  census_tract
0      7624403      663901      30502.0
1      7626061      707425      4102.0
2      7626432      655739      6501.0
3      7638375      689076      4500.0
4      7640361      706081      3901.0
...         ...         ...         ...
200078  7674417      685601      8100.0
200079  7688761      683719      9301.0
200080  7689764      675992      9101.0
200081  7671619      666586      602.0
200082  7680909      679340      9201.0

[200083 rows x 8 columns]
```

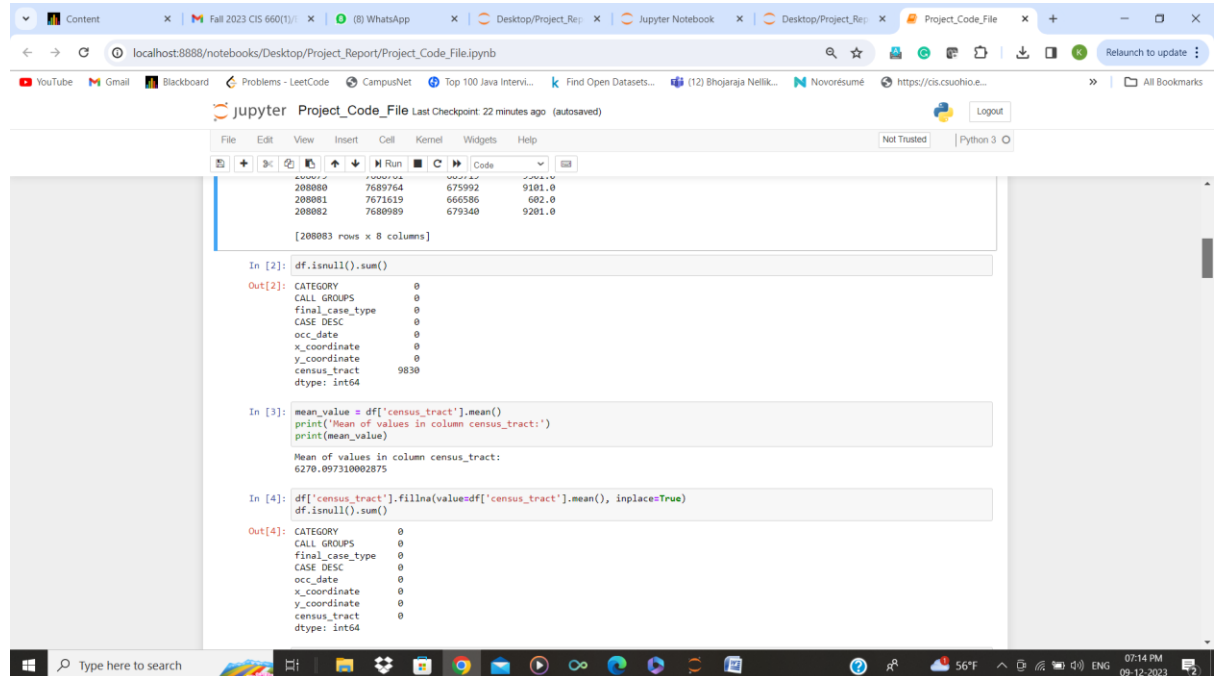
```
In [2]: df.isnull().sum()
Out[2]: CATEGORY      0
CALL_GROUPS      0
final_case_type    0
CASE_DESC         0
occ_date          0
x_coordinate       0
y_coordinate       0
census_tract      9830
dtype: int64

In [3]: mean_value = df['census_tract'].mean()
print('Mean of values in column census_tract:')
print(mean_value)
Mean of values in column census_tract:
6270.097310002875
```

.mean() method is used to calculate the mean (average) of numeric columns in a DataFrame. In this case, when applied to a DataFrame, it computes the mean value for each column, treating NaN (Not a Number) values as missing and excluding them from the calculation.

Code Script:

```
df['census_tract'].fillna(value=df['census_tract'].mean(), inplace=True)
df.isnull().sum()
```



fillna(value = df['census_tract'].mean()) fills NaN values in the DataFrame with the corresponding mean value of each column.

Code Script:

```
toconsider = ['CATEGORY', 'x_coordinate', 'y_coordinate']
info=df[toconsider]
print(info)
```

```

In [2]: mean_value = df['census_tract'].mean()
print('Mean of values in column census_tract:')
print(mean_value)

Mean of values in column census_tract:
6270.097310002875

In [4]: df['census_tract'].fillna(value=df['census_tract'].mean(), inplace=True)
df.isnull().sum()

Out[4]: CATEGORY      0
CALL_GROUPS      0
final_case_type    0
CASE_DESC         0
occ_date          0
x_coordinate       0
y_coordinate       0
census_tract      0
dtype: int64

In [5]: toconsider = ['CATEGORY', 'x_coordinate', 'y_coordinate']
info = df[toconsider]
print(info)

CATEGORY  x_coordinate  y_coordinate
0  STREET CRIMES      7624403      663901
1  STREET CRIMES      7626061      707425
2  STREET CRIMES      7626432      655739
3  STREET CRIMES      7638375      689076
4  STREET CRIMES      7640361      706981
...      ...          ...          ...
208078  BURGLARY      7674417      685601
208079  BURGLARY      7688761      683719
208080  BURGLARY      7689764      675992
208081  BURGLARY      7671619      666586
208082  BURGLARY      7680989      679340

[208083 rows x 3 columns]

```

Selecting features to consider for a machine learning model that directly impacts the model's performance.

How is feature selection done?

Using the Filter Method(Correlation-based method):

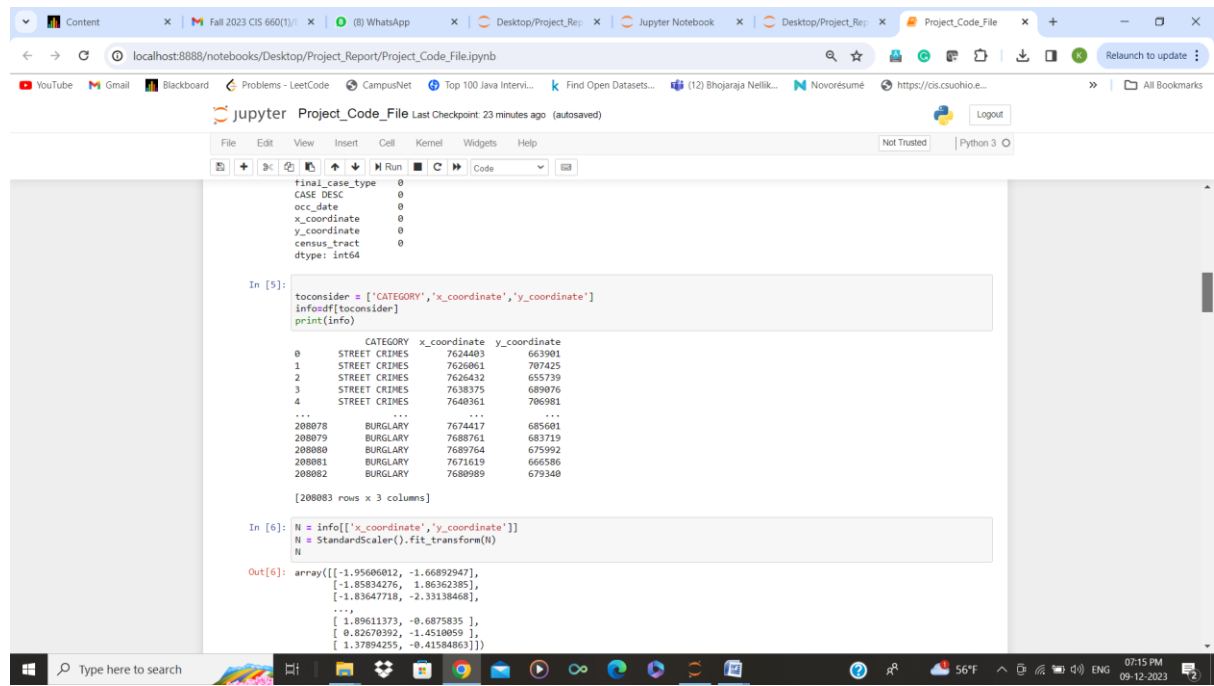
Selecting the relevant features from the set of all the features, that could impact the target variable and hence, the result of analysis.

We have taken 3 features for the prediction which includes:

CATEGORY	Nominal	Discrete
X_coordinate	Ratio	Continuous
Y_coordinate	Ratio	Continuous

Code Script:

```
N = info[['x_coordinate', 'y_coordinate']]
N = StandardScaler().fit_transform(N)
N
```



The screenshot shows a Jupyter Notebook window titled 'Project_Code_File'. The interface includes a top toolbar with icons for file operations, a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), and a status bar at the bottom. The notebook contains two code cells. The first cell, labeled 'In [5]:', defines 'toconsider' as a list of column names, uses 'info.drop(toconsider, axis=1)' to remove them, and prints the resulting 'info' object. The output shows a DataFrame with columns 'CATEGORY', 'x_coordinate', and 'y_coordinate', and 208883 rows. The second cell, labeled 'In [6]:', uses 'N = info[['x_coordinate', 'y_coordinate']]' to select the coordinates and 'N = StandardScaler().fit_transform(N)' to scale them. The output, labeled 'Out[6]:', shows a NumPy array of the scaled coordinates.

```
final_case_type    0
CASE_DESC          0
occ_date           0
x_coordinate        0
y_coordinate        0
census_tract       0
dtype: int64

In [5]:
toconsider = ['CATEGORY', 'x_coordinate', 'y_coordinate']
info.drop(toconsider, axis=1)
print(info)

   CATEGORY  x_coordinate  y_coordinate
0  STREET CRIMES      7624403      663901
1  STREET CRIMES      7626961      707425
2  STREET CRIMES      7626432      655739
3  STREET CRIMES      7638375      689076
4  STREET CRIMES      7640361      706981
...
208878  BURGLARY      7674417      685601
208879  BURGLARY      7688761      683719
208880  BURGLARY      7689764      675992
208881  BURGLARY      7671619      666586
208882  BURGLARY      7680989      679340

[208883 rows x 3 columns]

In [6]:
N = info[['x_coordinate', 'y_coordinate']]
N = StandardScaler().fit_transform(N)
N

Out[6]: array([[ -1.95060612,  -1.66892947],
               [-1.85834276,   1.86362385],
               [-1.83647718,  -2.33138468],
               ...,
               [ 1.89611373,  -0.6875835 ],
               [ 0.82670392,  -1.4510059 ],
               [ 1.37894255,  -0.41584863]])
```

StandardScaler is fitted to the data using the **fit_transform()** method, which both computes the mean and standard deviation of each feature and scales the features accordingly. After scaling, the resulting 'N' is a NumPy array with the standardized values.

Code Script:

```
sum_of_squared_distances = []
K = range(1,20)
for k in K:
    k_means = KMeans(n_clusters=k)
    model = k_means.fit(N)
    sum_of_squared_distances.append(k_means.inertia_)
```

The screenshot shows a Jupyter Notebook with the following code and output:

```
In [5]: toconsider = ['CATEGORY', 'x_coordinate', 'y_coordinate']
        info = info[toconsider]
        print(info)
```

	CATEGORY	x_coordinate	y_coordinate
0	STREET CRIMES	7624403	663901
1	STREET CRIMES	7626061	707425
2	STREET CRIMES	7626432	655739
3	STREET CRIMES	7638375	689076
4	STREET CRIMES	7640361	706981
...
208078	BURGLARY	7674417	685601
208079	BURGLARY	7688761	683719
208080	BURGLARY	7689764	675992
208081	BURGLARY	7671619	666586
208082	BURGLARY	7680989	679340

```
[208083 rows x 3 columns]
```

```
In [6]: N = info[['x_coordinate', 'y_coordinate']]
        N = StandardScaler().fit_transform(N)
        N
```

```
Out[6]: array([[ -1.95606812, -1.66892947],
                [-1.85834276,  1.86362385],
                [-1.83647718, -2.33138468],
                ...,
                [ 1.89611373, -0.6875835 ],
                [ 0.82670392, -1.4510059 ],
                [ 1.37894255, -0.41584863]])
```

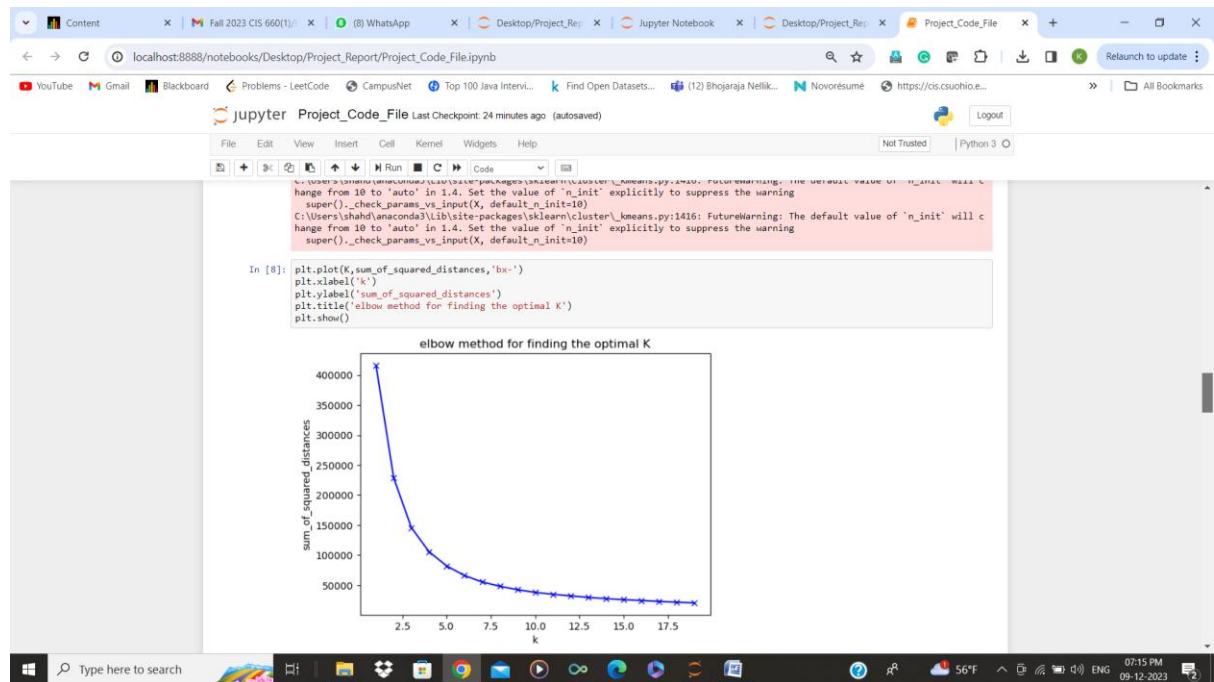
```
In [7]: sum_of_squared_distances = []
        K = range(1,20)
        for k in K:
            k_means = KMeans(n_clusters=k)
            model = k_means.fit(N)
            sum_of_squared_distances.append(k_means.inertia_)
```

Here, we used **elbow method** to determine the optimal number of clusters (k) in a K-means clustering algorithm. It involves running the K-means algorithm on the dataset for a range of values of k and plotting the inertia for each k. Looking at "elbow" point on the plot, where the rate of decrease in WCSS starts to slow down, indicating that adding more clusters does not significantly reduce the variance within each cluster.

kmeans.inertia_ represents the sum of squared distances of samples to their closest cluster center, which is a measure of WCSS.

Code Script:

```
plt.plot(K, sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('sum_of_squared_distances')
plt.title('elbow method for finding the optimal K')
plt.show()
```



This plot generated by the above code shows a curve, and the elbow of the curve represents the point where adding more clusters doesn't significantly reduce WCSS. The optimal value for k is often chosen at the elbow point.

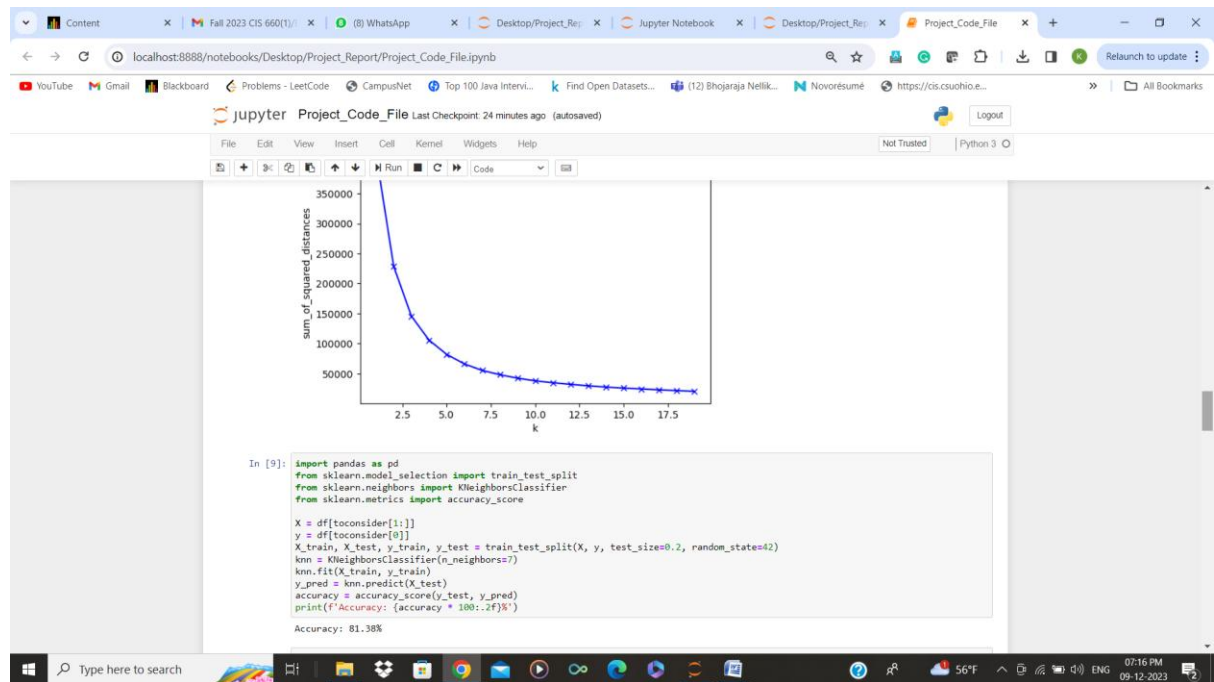
In this case, Elbow point or the optimal value of k is 7.

Which means we will be considering $k = 7$ or 7 clusters for our further analysis.

Code Script:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

X = df[toconsider[1:]]
y = df[toconsider[0]]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

Splitting the Data using **train_test_split** into training and testing sets.

The **test_size=0.2** parameter indicates that 20% of the data will be used for testing.

Hence, **train_size=0.8** paramter which indicates that 80% of the data will be used for training.

knn = KNeighborsClassifier(n_neighbors=7) This line initializes the KNN classifier with $k=7$, indicating that the algorithm will consider the labels of the seven nearest neighbours when making predictions.

fit(X_train, y_train): this is used to train the model on the provided training data (X_train for features and y_train for labels).

predict(X_test): method is used to make predictions on the test data (X_test). The resulting y_pred contains the predicted labels for the test data.

accuracy_score(y_test, y_pred): Compares the predicted labels (y_pred) with the actual labels (y_test) and calculates the accuracy.

The accuracy is a measure of how many predictions were correct out of the total number of predictions.

Accuracy = 81.38%

Code Script:

```
k_means_7 = KMeans(n_clusters=7)
model = k_means_7.fit(N)
y_hat_7 = k_means_7.predict(N)
labels_7 = k_means_7.labels_
print(metrics.silhouette_score(N,labels_7,metric = 'euclidean'))
```

```
[35]: k_means_7 = KMeans(n_clusters=7)
      model = k_means_7.fit(N)
      y_hat_7 = k_means_7.predict(N)
      labels_7 = k_means_7.labels_
      print(metrics.silhouette_score(N,labels_7,metric = 'euclidean'))
      #print(metrics.calinski_harabasz_score(x,labels_7))

C:\Users\shahd\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1416: FutureWarning:
change from 10 to 'auto' in 1.4. Set the value of 'n_init' explicitly to suppress the wa
super()._check_params_vs_input(X, default_n_init=10)

0.4834387897465314
```

The **silhouette_score** is a metric used to calculate the goodness of a clustering technique. It takes the data (N), the cluster labels (labels_7), and the distance metric (Euclidean) as parameters.

Note: The silhouette score is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). It ranges from -1 to 1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighbouring clusters.

In this case, we obtained **Silhouette_score : 0.483**

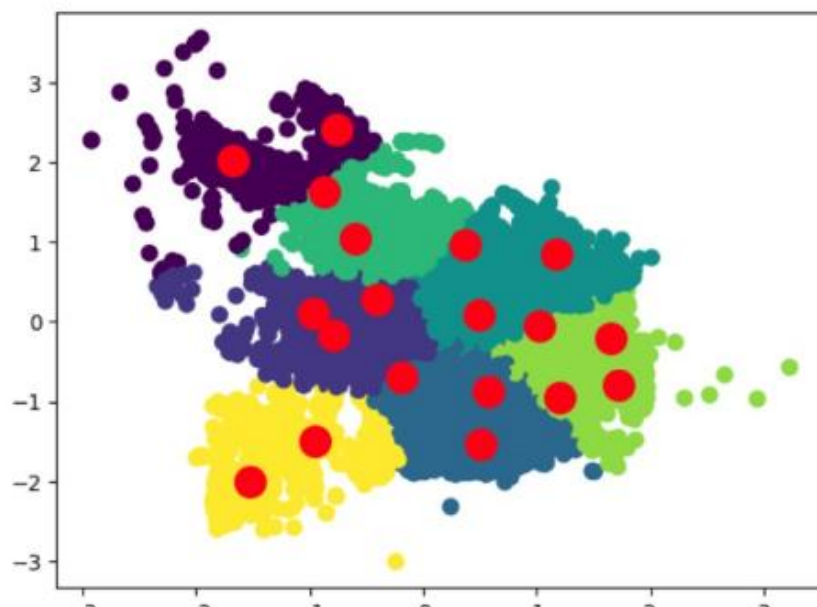
Code Script:

```
plt.scatter(N[:,0],N[:,1],c=labels_7,s=50,cmap='viridis')

centers = k_means.cluster_centers_
plt.scatter(centers[:,0],centers[:,1],c='red',s=200,alpha=1)
```

```
In [37]: plt.scatter(N[:,0],N[:,1],c=labels_7,s=50,cmap='viridis')
        centers = k_means.cluster_centers_
        plt.scatter(centers[:,0],centers[:,1],c='red',s=200,alpha=1)

Out[37]: <matplotlib.collections.PathCollection at 0x1cb1f009110>
```



Code Script:

```
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering

batch_size = 1000 # Set an appropriate batch size
num_samples = N.shape[0]

for i in range(0, num_samples, batch_size):
    batch_data = N[i:i+batch_size, :]

    model = AgglomerativeClustering(n_clusters=None, distance_threshold=0)
```

```
model.fit(batch_data)
```

“””

The dataset N is processed in batches using the AgglomerativeClustering algorithm. n_clusters=None indicates that the algorithm should not enforce a specific number of clusters, and distance_threshold=0 implies that clusters are formed regardless of distance.

“””

Plot dendrogram

```
def plot_dendrogram(model, **kwargs):
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1 # Leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)

    dendrogram(linkage_matrix, **kwargs)
```

“””

This function generates a dendrogram from the AgglomerativeClustering model. It calculates the linkage matrix from the model's children and distances.

“””

```
plt.title('Hierarchical Clustering Dendrogram')
plot_dendrogram(model, labels=model.labels_)
plt.show()
```

“””

This code generates and displays a dendrogram for the hierarchical clustering of the batch data.

“””

Perform hierarchical clustering using a subset of data to speed up execution

```
linkage_matrix = linkage(N[:batch_size], method='ward')
```

```
# Plot the dendrogram
dendrogram(linkage_matrix, labels=target.tolist() if 'target' in
df.columns else None, leaf_rotation=90)
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()

"""
```

This code generates and displays a dendrogram for hierarchical clustering using Ward's method on the subset of data.

```
"""
```

Content x Fall 2023 CIS 660(1) x (8) WhatsApp x Desktop/Project_Re... x Jupyter Notebook x Desktop/Project_Re... x Project_Code_File x + - □ X

localhost:8888/notebooks/Desktop/Project_Report/Project_Code_File.ipynb

Project_Code_File Last Checkpoint: 26 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
In [10]: import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering

batch_size = 1000 # Set an appropriate batch size
num_samples = N.shape[0]

for i in range(0, num_samples, batch_size):
    batch_data = N[i:i+batch_size, :]

    model = AgglomerativeClustering(n_clusters=None, distance_threshold=0)
    model.fit(batch_data)

    # Plot dendrogram
    def plot_dendrogram(model, **kwargs):
        counts = np.zeros(model.children_.shape[0])
        n_samples = len(model.labels_)
        for i, merge in enumerate(model.children_):
            current_count = 0
            for child_idx in merge:
                if child_idx < n_samples:
                    current_count += 1 # Leaf node
                else:
                    current_count += counts[child_idx - n_samples]
            counts[i] = current_count

        linkage_matrix = np.column_stack([model.children_, model.distances_,
                                         counts]).astype(float)

        dendrogram(linkage_matrix, **kwargs)

    plt.title('Hierarchical Clustering Dendrogram')
    plot_dendrogram(model, labels=model.labels_)
    plt.show()
```

Type here to search

Content x Fall 2023 CIS 660(1) x (8) WhatsApp x Desktop/Project_Re... x Jupyter Notebook x Desktop/Project_Re... x Project_Code_File x + - □ X

localhost:8888/notebooks/Desktop/Project_Report/Project_Code_File.ipynb

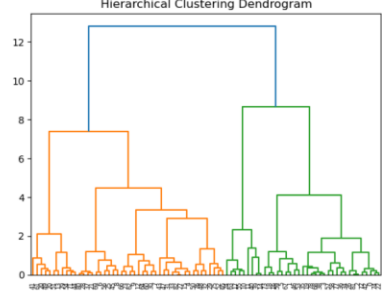
Project_Code_File Last Checkpoint: 26 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
# Perform hierarchical clustering using a subset of data to speed up execution
linkage_matrix = linkage(N[:batch_size], method='ward')

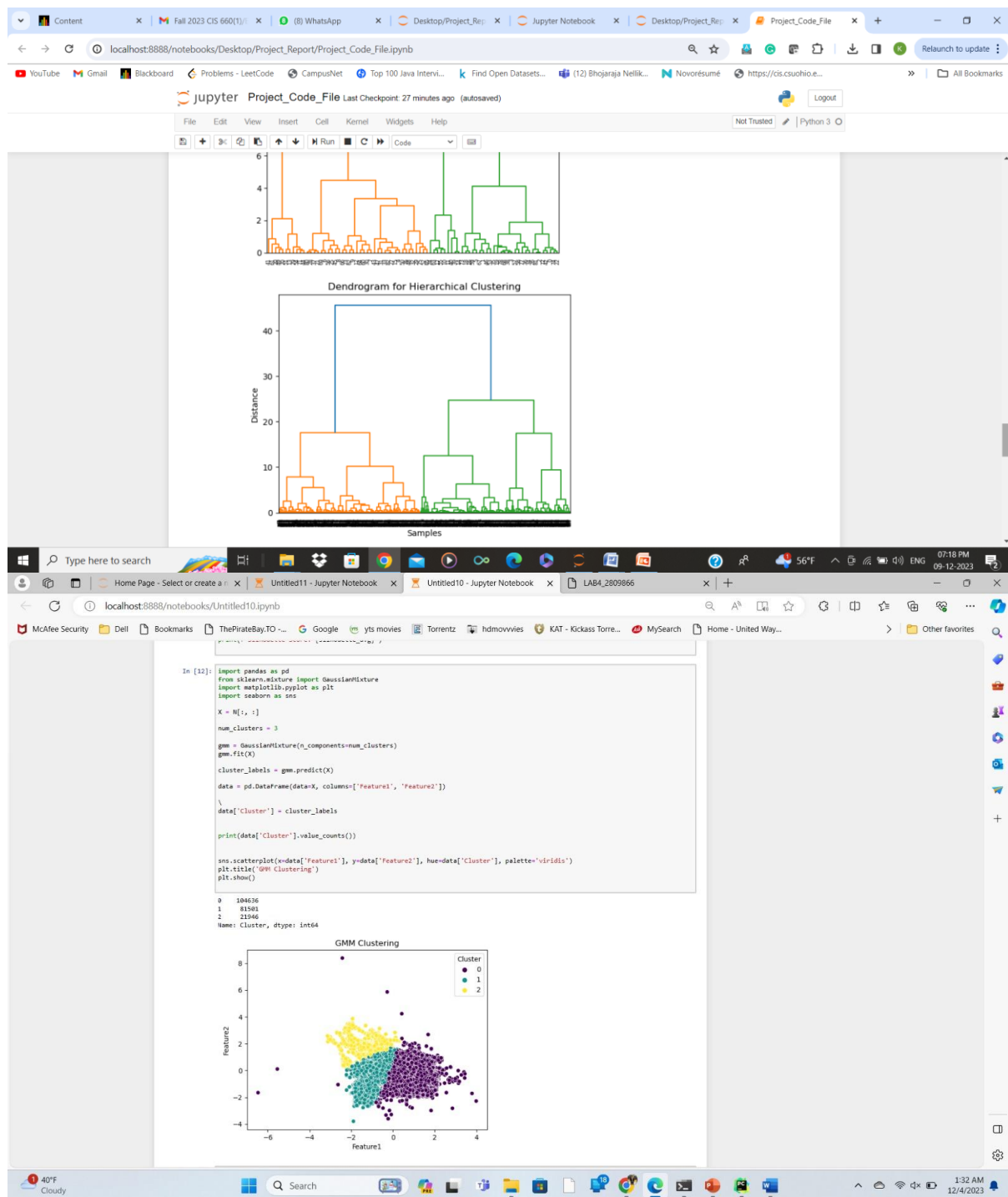
# Plot the dendrogram
dendrogram(linkage_matrix, labels=target.tolist() if 'target' in df.columns else None, leaf_rotations=90)
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Samples')
plt.ylabel('Distance')
plt.show()
```

Hierarchical Clustering Dendrogram



Dendrogram for Hierarchical Clustering

Type here to search



Code Script:

```
from sklearn.cluster import MiniBatchKMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
```

```
# Assuming you have your data in the variable 'N'
pca = PCA(n_components=2) # Choose the number of components you want
```

```

N_reduced = pca.fit_transform(N)

# Adjust the batch_size based on your available memory
batch_size = 1000

# Create MiniBatchKMeans model
model = MiniBatchKMeans(n_clusters=3, batch_size=batch_size)
labels = model.fit_predict(N_reduced)

# Calculate Silhouette Score
silhouette_avg = silhouette_score(N_reduced, labels)
print(f"Silhouette Score: {silhouette_avg}")

```

“””

This code applies PCA to reduce the dimensionality to 2, performs MiniBatchKMeans clustering with 3 clusters, and calculates the silhouette score.

“””

```

import pandas as pd
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming your dataset has features that you want to use for clustering
X = N[:, :] # Direct array indexing instead of iloc

# Choose the number of clusters (you can adjust this based on your data)
num_clusters = 3

# Fit a Gaussian Mixture Model to the data
gmm = GaussianMixture(n_components=num_clusters)
gmm.fit(X)

# Predict the cluster labels
cluster_labels = gmm.predict(X)

# Create a DataFrame for visualization (optional, but useful for seaborn plotting)
data = pd.DataFrame(data=X, columns=['Feature1', 'Feature2']) # Replace with actual column names

# Add the cluster labels to the DataFrame
data['Cluster'] = cluster_labels

# Print the cluster distribution
print(data['Cluster'].value_counts())

# Visualize the clusters (assuming you have 2D data, you may need to

```



```
modify for more dimensions)
sns.scatterplot(x=data['Feature1'], y=data['Feature2'],
hue=data['Cluster'], palette='viridis')
plt.title('GMM Clustering')
plt.show()
```

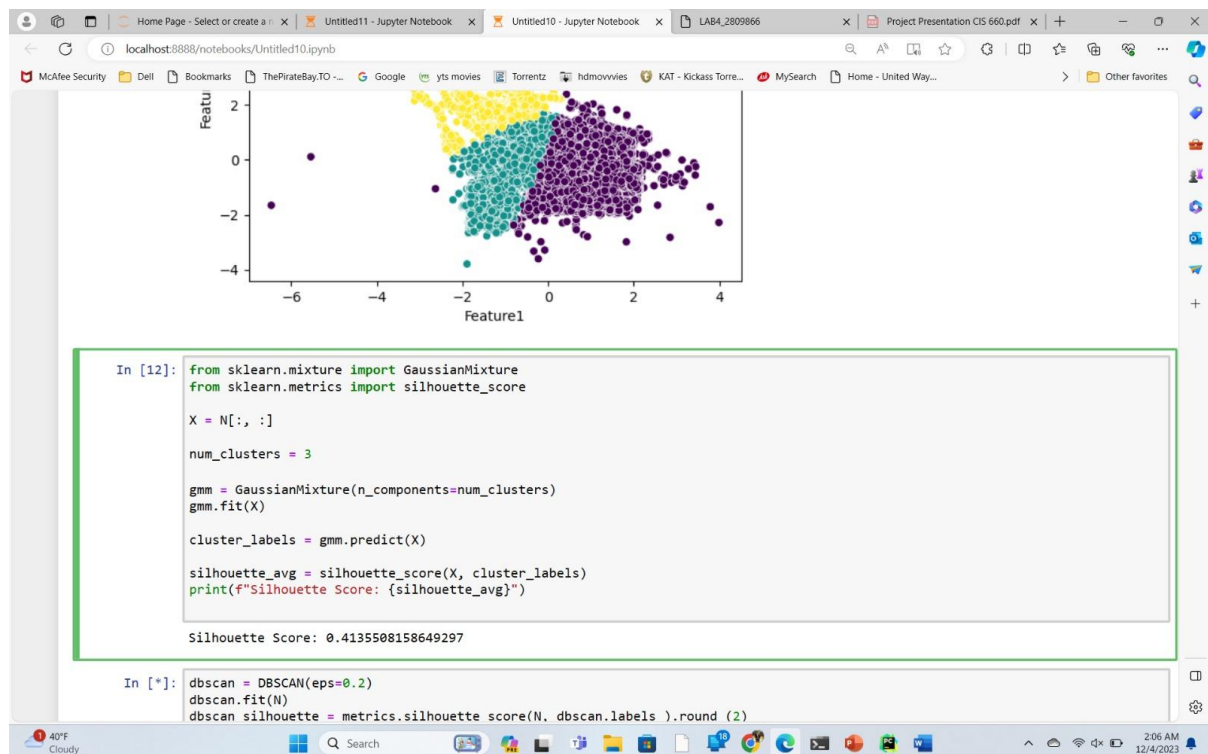
“””

This code applies GMM clustering to the data, prints the cluster distribution, and visualizes the clusters using a scatter plot.

“””

Code Script:

```
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
X = N[:, :]
num_clusters = 3
Feature1
gmm = GaussianMixture (n_components=num_clusters)
gmm.fit (x)
cluster_labels gmm.predict(X)
silhouette_avg silhouette_score (X, cluster_labels)
print(f"Silhouette Score: {silhouette_avg}")
```



In this code, we are calculating the silhouette_score for the GMM clustering using silhouette_score (X, cluster_labels).

We get **silhouette_score = 0.413** which is lesser in comparison to the silhouette score of KNN algorithm for K=7.

Code Script:

```
X = df[toconsider [1:]]
y = df[toconsider [0]]

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

num_clusters = 7

gmm = GaussianMixture (n_components=num_clusters, random_state=42)
train_cluster_labels = gmm.fit_predict(X_train)
test_cluster_labels = gmm.predict(X_test)
ari_train adjusted_rand_score(y_train, train_cluster_labels)
ari_test adjusted_rand_score(y_test, test_cluster_labels)
print (f"Adjusted Rand Index (ARI) - Train: {ari_train}")
print (f"Adjusted Rand Index (ARI) - Test: {ari_test}")
```

The screenshot shows a Jupyter Notebook running in a web browser. At the top, a warning message is displayed: "OMP_NUM_THREADS: 84 or by setting the environment variable OMP_NUM_THREADS=4". Below this, the "Silhouette Score" is calculated as 0.44443363852043133. The main code cell, labeled "In [20]:", contains the following Python code:

```
from sklearn.metrics import adjusted_rand_score
from sklearn.model_selection import train_test_split

X = df[toconsider[1:]]
y = df[toconsider[0]]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

num_clusters = 7

gmm = GaussianMixture(n_components=num_clusters, random_state=42)
train_cluster_labels = gmm.fit_predict(X_train)

test_cluster_labels = gmm.predict(X_test)

ari_train = adjusted_rand_score(y_train, train_cluster_labels)
ari_test = adjusted_rand_score(y_test, test_cluster_labels)

print(f"Adjusted Rand Index (ARI) - Train: {ari_train}")
print(f"Adjusted Rand Index (ARI) - Test: {ari_test}")
```

The output of the code cell shows the Adjusted Rand Index (ARI) for the training set as 0.003508344843901934 and for the testing data as 0.0028512712362778647.

Adjusted Rand Index (ARI) is a metric used to evaluate the accuracy of clustering algorithms. It measures the similarity between the true labels of the data points and the predicted labels obtained from a clustering algorithm, while correcting for chance. The ARI score ranges from -1 to 1, where:

ARI = 1: Perfect clustering (the predicted labels match the true labels).

ARI = 0: Random clustering (the predicted labels are no better than random chance).

ARI = -1: Perfect disagreement between predicted and true labels.

In this case, we are getting the **Adjusted Rand Index for training set as 0.003508** and **ARI for testing data as 0.002851**.

4. Challenges/ Error / Warnings

1) MEMORYERROR:

Due to the large dataset we faced Memory issue while performing Hierarchical Clustering algorithm. Also, algorithms were taking an enormous amount of time for processing the result.

2) FEATURESELECTION:

During the Data preprocessing phase, we faced difficulty in selection for all the essential features. In this we did a mistake of not considering census_tract as well. It would have been much better if we have considered census_tract as it would have provided us with areas. Which would eventually helped us in finding the hotspots of the crime.

3) FUTURE WARNING:

```
C:\Users\shahd\anaconda3\Lib\site-  
packages\sklearn\cluster\_kmeans.py:1416: FutureWarning: The  
default value of `n_init` will change from 10 to 'auto' in 1.4.  
Set the value of `n_init` explicitly to suppress the warning  
super()._check_params_vs_input(X, default_n_init=10)
```

5. References

<https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html>

[K-means Clustering — scikit-learn 1.0.1 documentation.](#)

<https://www.geeksforgeeks.org/k-nearest-neighbours/>