UTM UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING
UTM Johor Bahru

MCSD 2213 Advanced Analytics for Data Science

Mini Lab Project – Unsupervised Learning Model:

K - Means

Lecturer : Prof. Dr. Roliana Ibrahim
Date of Submission : 30th June 2023

No.	Group Members	Matric No
1	Syeinrita Devi A/P Anbealagan	MCS221022
2	Siti Nurul Arisya Binti Mohd Hanifah	MCS221020

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

Unsupervised machine learning algorithms, particularly the K-means clustering algorithm, have emerged as powerful tools in the domain of credit card transaction fraud detection. With the exponential growth of digital transactions, the need to detect and prevent fraudulent activities has become paramount. Unsupervised learning techniques provide an effective means of uncovering hidden patterns and anomalies within vast amounts of credit card transaction data without the need for labeled examples of fraud. The K-means algorithm, a popular unsupervised learning technique, can be applied to credit card transaction data sets to group similar transactions together based on their inherent characteristics. These characteristics may include features such as category, distance, amount, gender, zip code, city population, year, month, and day. By clustering transactions with similar attributes into distinct groups, K-means enables the identification of normal transaction behaviour and subsequently detects transactions that deviate significantly from their respective clusters. These deviant transactions are then flagged as potential instances of fraud.

One of the key advantages of using unsupervised machine learning with the K-means algorithm for credit card fraud detection is its ability to adapt to evolving fraud patterns. Traditional rule-based systems often struggle to keep pace with the ever-changing tactics employed by fraudsters. Unsupervised learning algorithms, on the other hand, can continuously analyse new data and update their clustering models to capture emerging fraud patterns. This adaptability enhances the effectiveness of fraud detection systems by enabling them to detect novel fraud techniques that may not have been encountered before. Furthermore, unsupervised machine learning using K-means for fraud detection eliminates the need for a labeled training set, which can be costly and time-consuming to obtain. Labeled examples of fraudulent transactions are often scarce, and manually labeling a sufficient amount of data to train a supervised model can be a laborious task.

By employing unsupervised learning, the algorithm can autonomously discover patterns and anomalies within the data without requiring pre-labeled instances of fraud. This significantly reduces the dependence on labeled data and accelerates the deployment of fraud detection systems. However, employing K-means clustering for credit card fraud detection does present its own set of challenges. One significant challenge is determining the optimal number of clusters (k) to use in the algorithm. Selecting an inappropriate number of clusters can lead to either over-segmentation, where legitimate transactions are unnecessarily flagged as fraudulent, or under-segmentation, where fraudulent transactions remain undetected. Determining the optimal k value requires careful consideration and may involve leveraging domain expertise, statistical techniques, or validation metrics such as the elbow method or silhouette score.

REPORT OVERVIEW

This report presents an analysis of the application of K-means clustering to a given dataset, aiming to uncover patterns and structures within the data and gain insights into the relationships between data points and assigned clusters. The analysis involved implementing the K-means algorithm with multiple runs to mitigate the impact of random initialization. The resulting cluster labels were assigned to the dataset, serving as a basis for further analysis. The evaluation of clustering quality was conducted using metrics such as the silhouette score and inertia, which assess the compactness, separation, and overall performance of the clusters. Visualizations played a crucial role in understanding the clustering results, including scatter plots and Principal Component Analysis (PCA) for visualizing clusters in lower-dimensional spaces. These visualizations facilitated the identification of patterns, correlations, and separations among different features and clusters.

The examination of pairwise distances between cluster centroids through the distance matrix provided insights into the dissimilarity or similarity between clusters, offering valuable information on their separations and overlaps in the feature space. The report offers detailed interpretations and analyses of the clusters, cluster centers, visualizations, and distance matrices, shedding light on the relationships between features and clusters, identifying potential patterns and trends, and providing insights into the characteristics and distributions of the data points. The findings highlight the significance of the applied K-means clustering and the insights gained from the analysis, underscoring their relevance in domains such as customer segmentation, fraud detection, and anomaly detection. Overall, this report serves as a comprehensive exploration of the dataset using K-means clustering. It provides a thorough understanding of the clustering process, evaluation metrics, visualizations, and interpretations of the results. The insights gained from this analysis can support informed decision-making and further analysis in a variety of applications.

DATA OVERVIEW

A European credit card company collected transaction data over the course of two days in September 2013 to create the Credit Card Transactions Fraud Detection Dataset, a publicly available dataset on Kaggle. This dataset includes a number of elements related to credit card transactions, such as the date and time of the transaction, the credit card number, information about the merchant, the transaction amount, and the name, gender, and address of the cardholder. We will investigate the use of unsupervised learning methods, particularly the K-means clustering algorithm, to identify fraudulent activities in credit card transactions.

The remaining features can be used to perform unsupervised clustering using K-means after the label class designating fraudulent or non-fraudulent transactions has been removed. Unsupervised learning techniques like the K-means algorithm are frequently used to classify data points according to how similar their features are. This method can be used to find groups or patterns in the dataset that might point to possible fraudulent activity in the context of fraud detection.

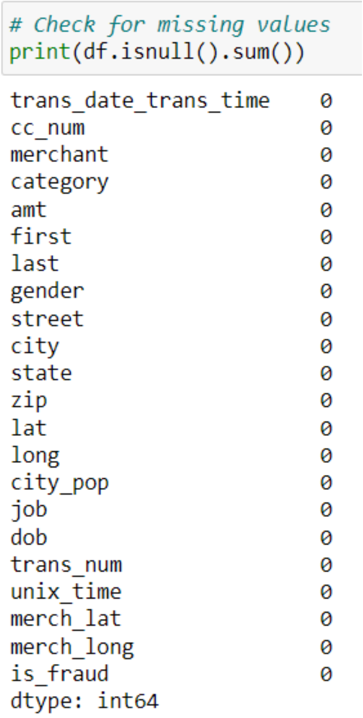
We can find underlying structures and groupings in the transaction data by using K-means clustering on this dataset. Based on how close the data points are to the cluster centres, the algorithm will repeatedly assign data points to various clusters. The algorithm, however, lacks prior knowledge of which cluster represents fraudulent transactions in the absence of the labelled data.

With the help of the clusters produced by K-means, we can spot unusual or anomalous patterns by learning more about possible groups of transactions that share certain traits. Despite the fact that K-means does not directly categorise transactions as fraudulent or not, it can still aid in a deeper understanding of the data and possibly point out specific areas of interest for additional research.

DATA PREPROCESSING

Data preprocessing plays a crucial role in preparing the data for unsupervised machine learning tasks. It involves several steps to ensure that the data is in a suitable format and quality for effective analysis using unsupervised algorithms. This data preprocessing will include steps such as handling missing values, handling duplicated values, handling categorical variables, transforming and creating new variables, and scaling.

HANDLING MISSING VALUES

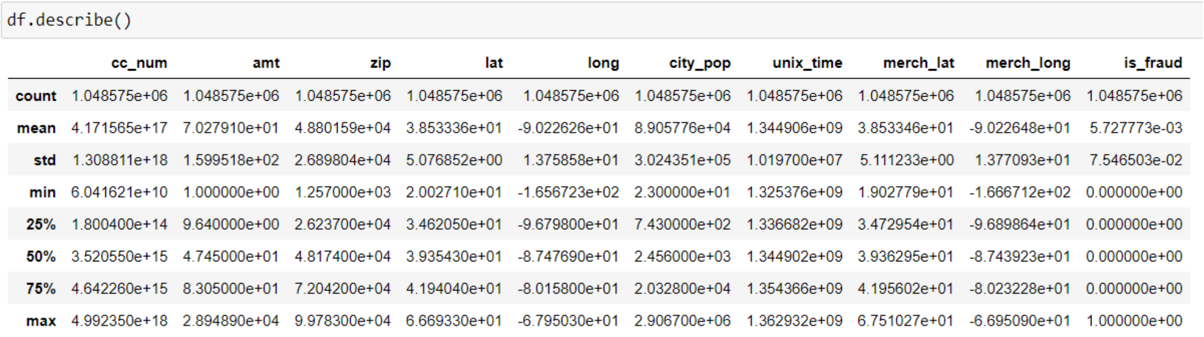


```
# Check for missing values
print(df.isnull().sum())
```

```
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
last                     0
gender                   0
street                   0
city                     0
state                    0
zip                      0
lat                      0
long                     0
city_pop                 0
job                      0
dob                      0
trans_num                0
unix_time                0
merch_lat                0
merch_long               0
is_fraud                 0
dtype: int64
```

Figure 1 Missing Values

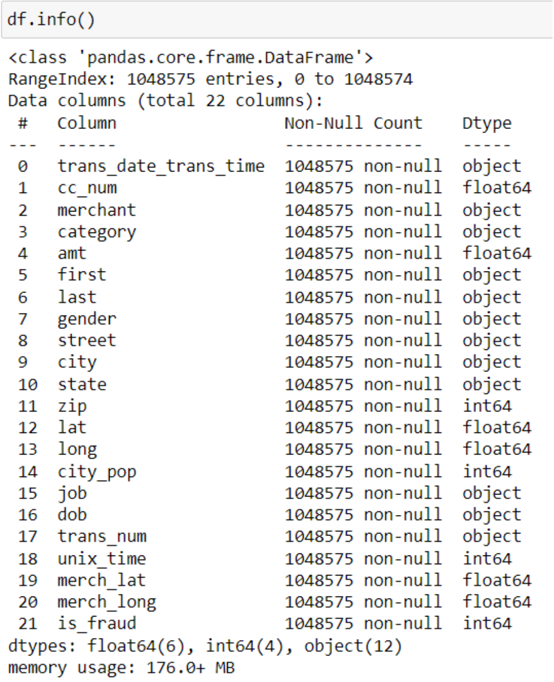
In the data preprocessing phase, one important step is to handle missing values appropriately. Upon checking the fraud dataset, this dataset has no missing values. Having a dataset without missing values contributes to the reliability and accuracy of the subsequent analysis, enabling unsupervised learning algorithms to uncover meaningful patterns and structures within the data more effectively. It eliminates the need for imputation techniques or deciding whether to remove or retain incomplete instances. With complete data, the focus can shift towards other preprocessing steps such as feature scaling, dimensionality reduction, or encoding categorical variables. Having no missing values simplifies the preprocessing pipeline and allows for a more streamlined analysis.



```
df.describe()
```

	cc_num	amt	zip	lat	long	city_pop	unix_time	merch_lat	merch_long	is_fraud
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06
mean	4.171565e+17	7.027910e+01	4.880159e+04	3.853336e+01	-9.022626e+01	8.905776e+04	1.344906e+09	3.853346e+01	-9.022648e+01	5.727773e-03
std	1.308811e+18	1.599518e+02	2.689804e+04	5.076852e+00	1.375858e+01	3.024351e+05	1.019700e+07	5.111233e+00	1.377093e+01	7.546503e-02
min	6.041621e+10	1.000000e+00	1.257000e+03	2.002710e+01	-1.656723e+02	2.300000e+01	1.325376e+09	1.902779e+01	-1.666712e+02	0.000000e+00
25%	1.800400e+14	9.640000e+00	2.623700e+04	3.462050e+01	-9.679800e+01	7.430000e+02	1.336682e+09	3.472954e+01	-9.689864e+01	0.000000e+00
50%	3.520550e+15	4.745000e+01	4.817400e+04	3.935430e+01	-8.747690e+01	2.456000e+03	1.344902e+09	3.936295e+01	-8.743923e+01	0.000000e+00
75%	4.642260e+15	8.305000e+01	7.204200e+04	4.194040e+01	-8.015800e+01	2.032800e+04	1.354366e+09	4.195602e+01	-8.023228e+01	0.000000e+00
max	4.992350e+18	2.894890e+04	9.978300e+04	6.669330e+01	-6.795030e+01	2.906700e+06	1.362932e+09	6.751027e+01	-6.695090e+01	1.000000e+00

Figure 2 Data Describing



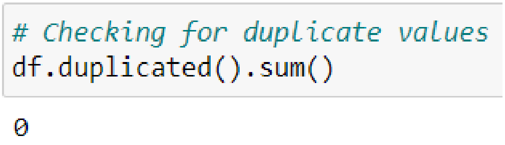
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 22 columns):
 #   Column                 Non-Null Count    Dtype  
---  ------                 --------------    -----  
 0   trans_date_trans_time  1048575 non-null  object 
 1   cc_num                 1048575 non-null  float64
 2   merchant               1048575 non-null  object 
 3   category               1048575 non-null  object 
 4   amt                    1048575 non-null  float64
 5   first                  1048575 non-null  object 
 6   last                   1048575 non-null  object 
 7   gender                 1048575 non-null  object 
 8   street                 1048575 non-null  object 
 9   city                   1048575 non-null  object 
 10  state                  1048575 non-null  object 
 11  zip                    1048575 non-null  int64  
 12  lat                    1048575 non-null  float64
 13  long                   1048575 non-null  float64
 14  city_pop               1048575 non-null  int64  
 15  job                    1048575 non-null  object 
 16  dob                    1048575 non-null  object 
 17  trans_num              1048575 non-null  object 
 18  unix_time              1048575 non-null  int64  
 19  merch_lat              1048575 non-null  float64
 20  merch_long             1048575 non-null  float64
 21  is_fraud               1048575 non-null  int64  
dtypes: float64(6), int64(4), object(12)
memory usage: 176.0+ MB
```

Figure 3 Dataset Information

Figure 2 above shows the summary of the statistical measures for each numerical column in the data while Figure 3 shows the columns in the fraud dataset along with its datatype. Overall, this dataset consists of 22 columns including transactions date, credit card number, merchant, category, amount, first name, last name, gender, street address, city, state, zip code, latitude, longitude, city population, job of the customers, data of birthday, transaction number, Unix time, merchant latitude and longitude, and is_fraud column.

HANDLING DUPLICATED VALUES



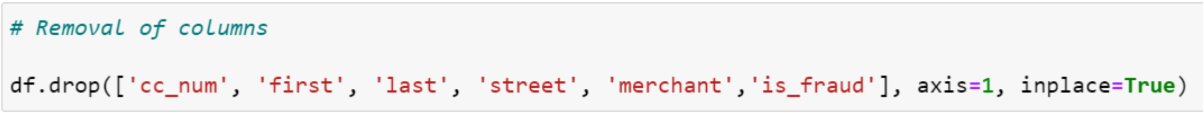
```
# Checking for duplicate values
df.duplicated().sum()
```

```
0
```

Figure 4 Data Duplicate Checking

Data duplication in the data preprocessing step refers to the presence of identical or nearly identical records in a dataset. It can occur due to various reasons, such as errors in data collection, data merging, or data storage. Duplicate data can negatively impact data analysis and machine learning models by introducing bias, skewing results, and wasting computational resources. As shown in Figure 4, this dataset has no duplicate data, and we can further to the next data preprocessing steps.

REMOVAL OF COLUMNS

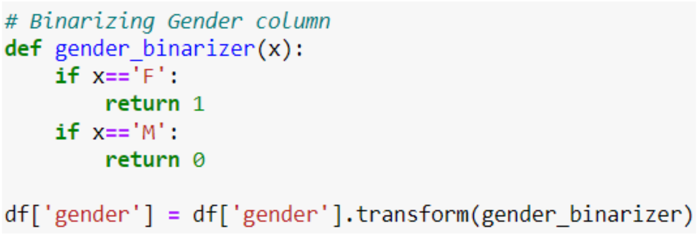


```
# Removal of columns

df.drop(['cc_num', 'first', 'last', 'street', 'merchant','is_fraud'], axis=1, inplace=True)
```

Figure 5 Removal of columns

The goal of k-means clustering is to find natural groupings within the data using the given features. The 'is_fraud' column, which represents the label information identifying fraud or non-fraud transactions, must be removed because k-means is an unsupervised learning technique that does not use preexisting labels. By doing this, we avoid bias and make sure the clustering procedure is entirely focused on identifying trends and similarities in the feature space. Due to the absence of the 'is_fraud' label, k-means can now recognise underlying data structures without being impacted by its presence. It's vital to remember that k-means clustering is typically employed for exploratory research and analysing data, not directly ideal for fraud detection.

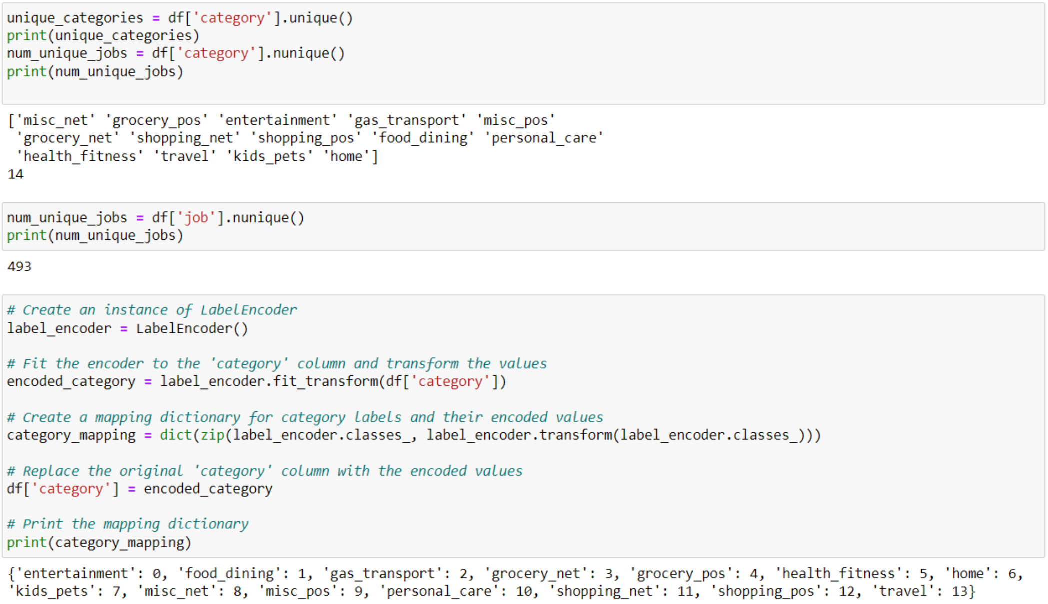
HANDLING CATEGORICAL VARIABLES



```
# Binarizing Gender column
def gender_binarizer(x):
    if x=='F':
        return 1
    if x=='M':
        return 0

df['gender'] = df['gender'].transform(gender_binarizer)
```

Figure 6 Converting Gender Column to Binary Data

Transforming categorical data into numerical representations plays a vital role in achieving accurate analysis when using unsupervised modeling techniques. To facilitate this process, it is necessary to binarize the "gender" column in the provided code, effectively converting the categorical variable into a binary feature. By assigning a value of 0 to represent "male" and 1 to represent "female," the models can comprehend and utilize this information during the prediction phase. This conversion enables the algorithms to capture any potential correlations between gender and fraud detection, leading to enhanced prediction accuracy and the inclusion of relevant insights.



```
unique_categories = df['category'].unique()
print(unique_categories)
num_unique_jobs = df['category'].nunique()
print(num_unique_jobs)
```

```
['misc_net' 'grocery_pos' 'entertainment' 'gas_transport' 'misc_pos'
 'grocery_net' 'shopping_net' 'shopping_pos' 'food_dining' 'personal_care'
 'health_fitness' 'travel' 'kids_pets' 'home']
14
```

```
num_unique_jobs = df['job'].nunique()
print(num_unique_jobs)
```

```
493
```

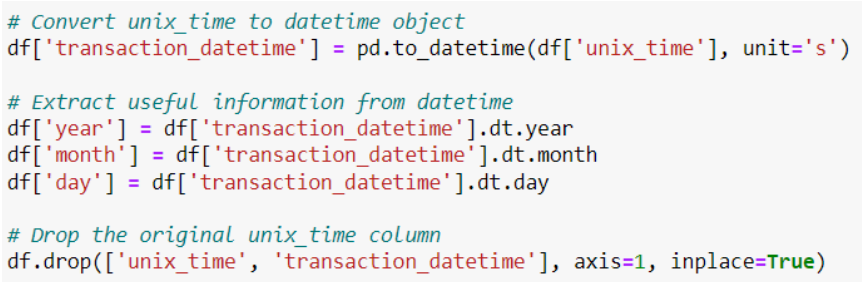
```
# Create an instance of LabelEncoder
label_encoder = LabelEncoder()

# Fit the encoder to the 'category' column and transform the values
encoded_category = label_encoder.fit_transform(df['category'])

# Create a mapping dictionary for category labels and their encoded values
category_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))

# Replace the original 'category' column with the encoded values
df['category'] = encoded_category

# Print the mapping dictionary
print(category_mapping)
```

```
{'entertainment': 0, 'food_dining': 1, 'gas_transport': 2, 'grocery_net': 3, 'grocery_pos': 4, 'health_fitness': 5, 'home': 6,
'kids_pets': 7, 'misc_net': 8, 'misc_pos': 9, 'personal_care': 10, 'shopping_net': 11, 'shopping_pos': 12, 'travel': 13}
```

Figure 7 Label Encoding

Label encoding is a common technique employed to convert categorical variables into numerical representations. In the given dataset, the categorical column underwent a label encoding transformation, resulting in unique numerical labels assigned to each category. This encoding process yielded 14 distinct attributes representing the categorical variable in a numerical format. By utilizing label encoding, the dataset can be effectively utilized in various data analysis or machine learning tasks that require numerical input.

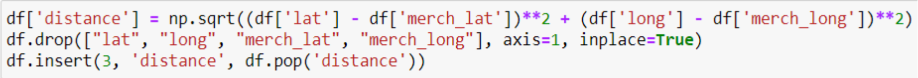
TRANSFORMING AND CREATING NEW VARIABLES



```
# Convert unix_time to datetime object
df['transaction_datetime'] = pd.to_datetime(df['unix_time'], unit='s')

# Extract useful information from datetime
df['year'] = df['transaction_datetime'].dt.year
df['month'] = df['transaction_datetime'].dt.month
df['day'] = df['transaction_datetime'].dt.day

# Drop the original unix_time column
df.drop(['unix_time', 'transaction_datetime'], axis=1, inplace=True)
```

Figure 8 New Datetime Variable

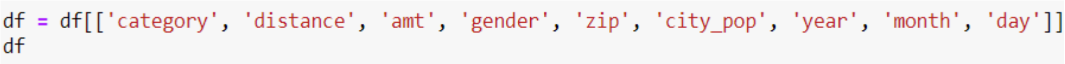
The column "unix_time" contains Unix time stamps representing the number of seconds since January 1, 1970. By converting these timestamps into a datetime object using pd.to_datetime, we can easily manipulate and extract specific information such as year, month, and day. These extracted temporal features offer valuable insights into fraud detection patterns and trends over time. Additionally, removing the original "unix_time" column eliminates redundancy and maintains the cleanliness of the dataset. Overall, transforming "unix_time" into a datetime object and extracting temporal information enhances the dataset and boosts the performance of fraud detection models.



```
df['distance'] = np.sqrt((df['lat'] - df['merch_lat'])**2 + (df['long'] - df['merch_long'])**2)
df.drop(["lat", "long", "merch_lat", "merch_long"], axis=1, inplace=True)
df.insert(3, 'distance', df.pop('distance'))
```

Figure 9 New Distance Variable

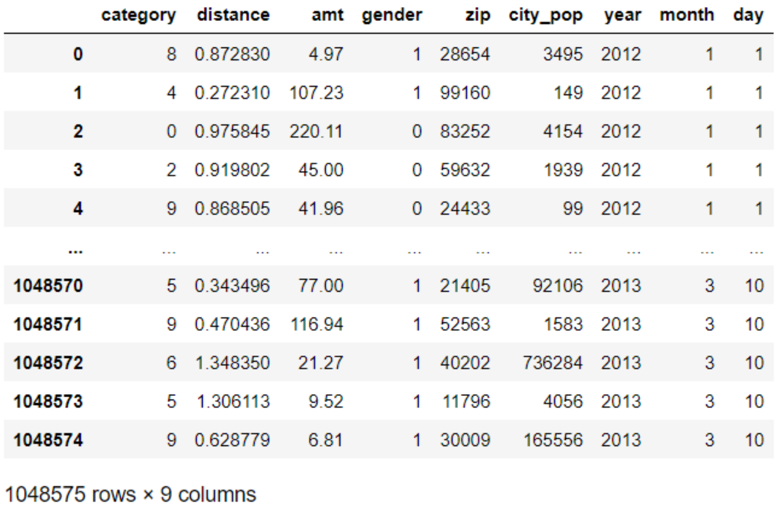
The provided code utilizes latitude and longitude coordinates to calculate the Euclidean distance between transaction and merchant locations. By incorporating this distance as a new column in the DataFrame, the code captures the spatial proximity between the two places, which can be valuable for fraud detection purposes. Following the distance computation, the original latitude and longitude columns are removed to streamline the dataset and eliminate redundant features. This preprocessing step enhances the efficiency and relevance of unsupervised models, enabling them to effectively identify fraudulent activity by leveraging spatial patterns.



```
df = df[['category', 'distance', 'amt', 'gender', 'zip', 'city_pop', 'year', 'month', 'day']]
df
```

Figure 10 Removing Unwanted Columns

The code in Figure 9 performs a column selection operation on the DataFrame. The resulting DataFrame will only contain these selected columns, and any other columns present in the original DataFrame will be excluded. This column selection operation is useful when working with a subset of columns from a larger DataFrame and discarding the rest of the columns that are not needed for the modelling task at hand.



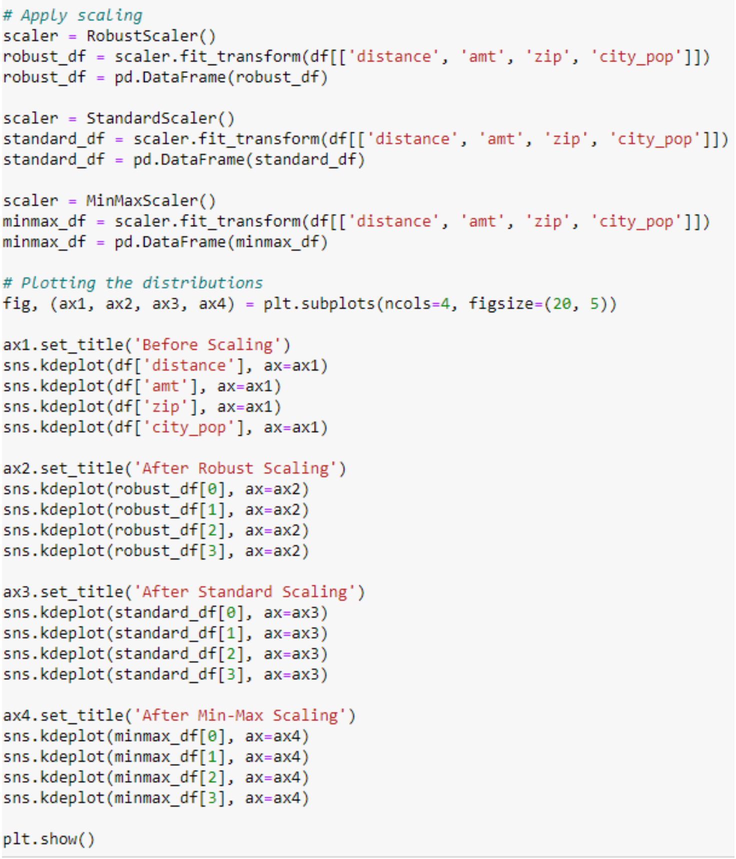
	category	distance	amt	gender	zip	city_pop	year	month	day
0	8	0.872830	4.97	1	28654	3495	2012	1	1
1	4	0.272310	107.23	1	99160	149	2012	1	1
2	0	0.975845	220.11	0	83252	4154	2012	1	1
3	2	0.919802	45.00	0	59632	1939	2012	1	1
4	9	0.868505	41.96	0	24433	99	2012	1	1
...	...	...	...	...	...	...	...	...	...
1048570	5	0.343496	77.00	1	21405	92106	2013	3	10
1048571	9	0.470436	116.94	1	52563	1583	2013	3	10
1048572	6	1.348350	21.27	1	40202	736284	2013	3	10
1048573	5	1.306113	9.52	1	11796	4056	2013	3	10
1048574	9	0.628779	6.81	1	30009	165556	2013	3	10

1048575 rows × 9 columns

Figure 11 Columns Involved in Unsupervised Modelling

Figure 10 shows the resulting columns selected for this unsupervised modelling. the not useful attributes for the analysis like the cc_num variable is removed from the data. Removing irrelevant variables enhances the model's performance by providing a cleaner signal and allowing it to focus on the most relevant features. It also helps reduce the dimensionality of the dataset, making the model more efficient and mitigating the risk of overfitting.

SCALING



```
# Apply scaling
scaler = RobustScaler()
robust_df = scaler.fit_transform(df[['distance', 'amt', 'zip', 'city_pop']])
robust_df = pd.DataFrame(robust_df)

scaler = StandardScaler()
standard_df = scaler.fit_transform(df[['distance', 'amt', 'zip', 'city_pop']])
standard_df = pd.DataFrame(standard_df)

scaler = MinMaxScaler()
minmax_df = scaler.fit_transform(df[['distance', 'amt', 'zip', 'city_pop']])
minmax_df = pd.DataFrame(minmax_df)

# Plotting the distributions
fig, (ax1, ax2, ax3, ax4) = plt.subplots(ncols=4, figsize=(20, 5))

ax1.set_title('Before Scaling')
sns.kdeplot(df['distance'], ax=ax1)
sns.kdeplot(df['amt'], ax=ax1)
sns.kdeplot(df['zip'], ax=ax1)
sns.kdeplot(df['city_pop'], ax=ax1)

ax2.set_title('After Robust Scaling')
sns.kdeplot(robust_df[0], ax=ax2)
sns.kdeplot(robust_df[1], ax=ax2)
sns.kdeplot(robust_df[2], ax=ax2)
sns.kdeplot(robust_df[3], ax=ax2)

ax3.set_title('After Standard Scaling')
sns.kdeplot(standard_df[0], ax=ax3)
sns.kdeplot(standard_df[1], ax=ax3)
sns.kdeplot(standard_df[2], ax=ax3)
sns.kdeplot(standard_df[3], ax=ax3)

ax4.set_title('After Min-Max Scaling')
sns.kdeplot(minmax_df[0], ax=ax4)
sns.kdeplot(minmax_df[1], ax=ax4)
sns.kdeplot(minmax_df[2], ax=ax4)
sns.kdeplot(minmax_df[3], ax=ax4)

plt.show()
```

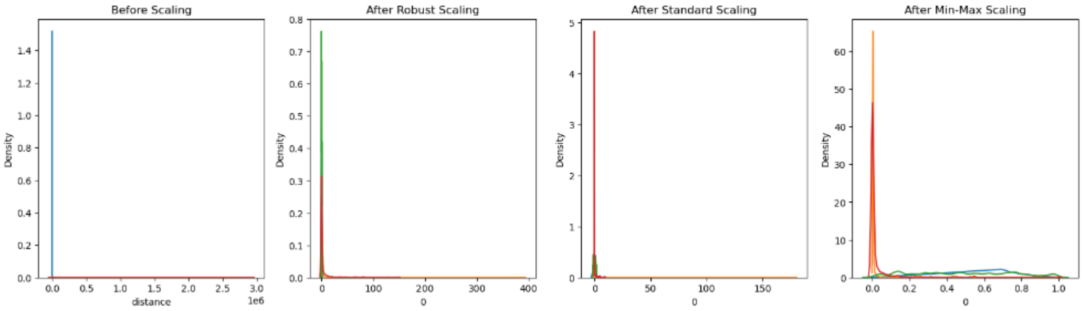
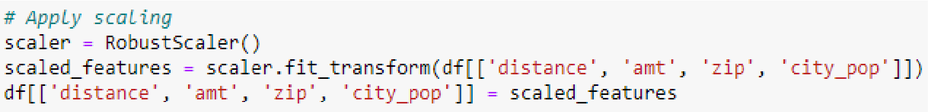


Figure 12 Scaling

In unsupervised models, scaling or normalization is a crucial preprocessing step to ensure that all features have equal contributions and are on a comparable scale. Scaling is important because unsupervised models rely on measuring distances or similarities between data points to identify patterns or clusters. To achieve this, various scaling techniques are commonly employed. Standardization, or Z-score normalization, transforms the data to have a mean of zero and a standard deviation of one, effectively centering the data around zero and bringing features to a similar scale. Min-max scaling rescales the data to a specific range, typically between 0 and 1, preserving the relative relationships between data points. Robust scaling, similar to standardization, is more resilient to outliers by using the median and interquartile range. Normalization scales each data point to have a unit norm or length, often useful when the direction rather than the magnitude of the data is crucial. The choice of scaling technique depends on the characteristics of the data and the requirements of the unsupervised model.

After applying scaling techniques, the distributions of the features appear more comparable and aligned when examined through plotted distributions. This alignment has the potential to enhance the model's ability to identify meaningful patterns and generate accurate predictions. Upon scaling the features, Figure 11 shows minimal visible differences, indicating that the original feature scales were already similar or that the chosen scaling techniques did not significantly alter the distributions. However, it's important to note that even if the changes in distributions may not be apparent in the graphs, scaling can still have a substantial impact on the performance of supervised learning models. The primary objective of scaling is to standardize the features to a common scale and range, enabling the models to effectively interpret their relative importance. Scaling aids in improving model convergence, stability, and generalization to new, unseen data.



```
# Apply scaling
scaler = RobustScaler()
scaled_features = scaler.fit_transform(df[['distance', 'amt', 'zip', 'city_pop']])
df[['distance', 'amt', 'zip', 'city_pop']] = scaled_features
```

Figure 13 Applying Robust Scaling

ROBUST SCALING

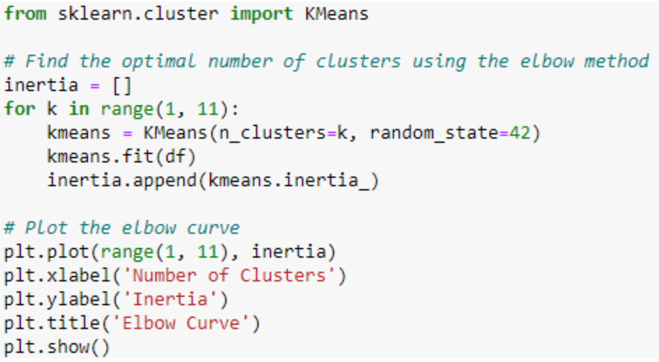
RobustScaler was chosen because it is less sensitive to outliers than other scaling methods. Outliers are common in fraud detection datasets and can have a significant impact on the scaling process. RobustScaler scales the features using robust statistics such as median and interquartile range, making it more resistant to outliers. We ensure that the scaling is not skewed by extreme values by using RobustScaler, resulting in more reliable and consistent results. By applying robust scaling, the k-means algorithm becomes more robust to extreme values and maintains the integrity of the clustering process. It helps ensure that outliers or data points with large values do not disproportionately impact the clustering results. This allows k-means to focus on the underlying patterns in the majority of the data and make more reliable cluster assignments.

MODELLING - UNSUPERVISED MODEL

K-MEANS CLUSTERING

K-means clustering is an unsupervised machine learning algorithm used to partition a dataset into distinct clusters. The goal is to group similar data points together while keeping dissimilar points in separate clusters. It is one of the most widely used clustering algorithms due to its simplicity and efficiency. The first step in k-means is to pick the number of clusters. The elbow method is one of the most popular methods. To implement it, we apply k-means with a different number of clusters and compare their within-cluster sum of squares (WCSS).

ELBOW METHOD



```
from sklearn.cluster import KMeans

# Find the optimal number of clusters using the elbow method
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df)
    inertia.append(kmeans.inertia_)

# Plot the elbow curve
plt.plot(range(1, 11), inertia)
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Curve')
plt.show()
```

Figure 14 Determining Number of Cluster

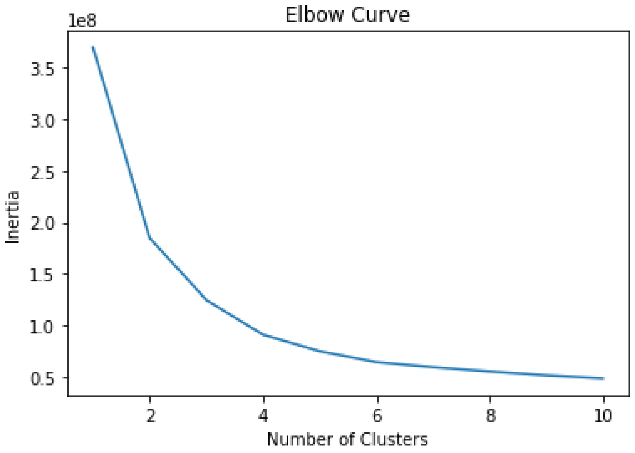
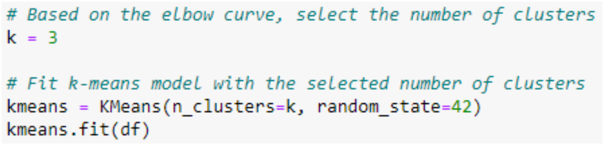


Figure 15 Elbow Curve plot to Determine the Number of Clusters

The resulting elbow curve in figure 14 visualizes the trade-off between the number of clusters and the inertia. By observing the plot, analysts can identify the "elbow" point, representing a significant decrease in inertia. This point often indicates the optimal number of clusters to choose for subsequent K-means clustering analysis. Using the code provided in Figure 13, to determine the optimal number of clusters, we have to select the value of k at the “elbow” i.e. the point after which the distortion/inertia starts decreasing in a linear fashion. Thus, we conclude that the optimal number of clusters is 3.

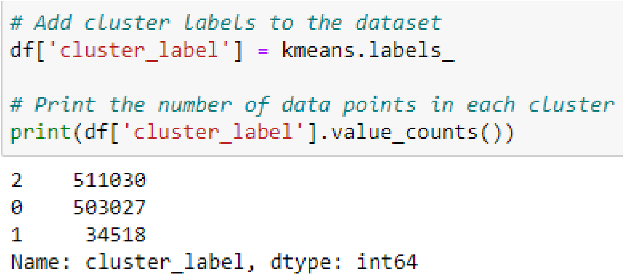


```
# Based on the elbow curve, select the number of clusters
k = 3

# Fit k-means model with the selected number of clusters
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(df)
```

Figure 16 Applying K-Means

Once the optimal number of clusters has been identified using the elbow method, we can proceed to apply the K-means model and assign each sample to its corresponding cluster. In this implementation, the code specifies that the K-means algorithm will be run three times with different centroid seeds, and the best output in terms of within-cluster sum of squares (WCSS) will be selected as the final result. The parameter responsible for running the algorithm multiple times is usually referred to as the "n_init" parameter in scikit-learn's KMeans class. By setting it to 3, the K-means algorithm will be executed three times with different initial centroid seeds. The algorithm calculates the WCSS for each run and selects the run that yields the lowest WCSS as the final result. This approach helps mitigate the effect of random initialization and improves the stability and reliability of the clustering.

After running the K-means algorithm with the specified number of clusters and the multiple runs, the resulting model can be used to predict the cluster assignments for each sample in the dataset. The assigned cluster labels provide insights into the grouping structure and help identify similar patterns or behaviors within the data.



```
# Add cluster labels to the dataset
df['cluster_label'] = kmeans.labels_

# Print the number of data points in each cluster
print(df['cluster_label'].value_counts())
```

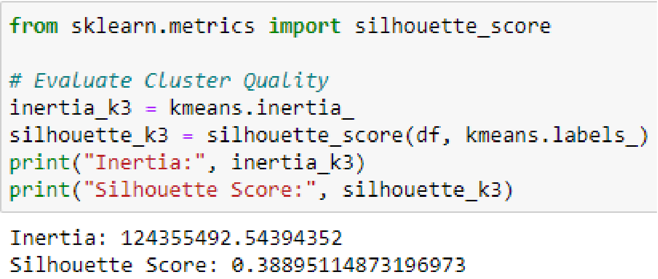
```
2    511030
0    503027
1     34518
Name: cluster_label, dtype: int64
```

Figure 17 Adding the cluster labels to the dataset

The code in Figure 17 provided adding the cluster labels to the dataset based on the results of the K-means clustering. It assigns the cluster labels to a new column called 'cluster_label'. The output of the statement shows the number of data points in each cluster. In the given output, cluster 2 contains 511,030 data points, cluster 0 contains 503,027 data points, and cluster 1 contains 34,518 data points. This information helps to understand the distribution of data points across the clusters, providing insights into the sizes and imbalances within the clusters. It can be useful for further analysis or for evaluating the effectiveness of the clustering algorithm.

CLUSTERING EVALUATION, K=3

INERTIA & SILHOUETTE SCORE



```
from sklearn.metrics import silhouette_score

# Evaluate Cluster Quality
inertia_k3 = kmeans.inertia_
silhouette_k3 = silhouette_score(df, kmeans.labels_)
print("Inertia:", inertia_k3)
print("Silhouette Score:", silhouette_k3)
```

```
Inertia: 124355492.54394352
Silhouette Score: 0.38895114873196973
```

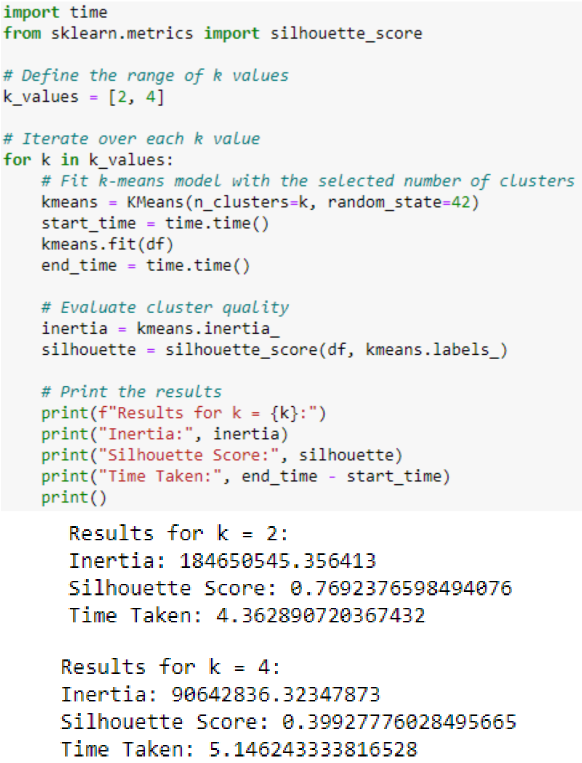
Figure 18 Calculating Silhouette Score

The provided code in Figure 18 demonstrates the evaluation of cluster quality after applying K-means clustering. It utilizes the silhouette score and inertia metric to assess the clustering results. This score measures the compactness and separation of the clusters, ranging from -1 to 1. Higher values indicate better-defined clusters with samples well-matched to their own clusters and poorly matched to neighboring clusters. Inertia represents the sum of squared distances of samples to their closest cluster center. Lower inertia values indicate better clustering performance.

In the printed Figure 18 result, the inertia is 124,355,492.54394352, and the silhouette score is 0.38895114873196973. There is still a lot of fluctuation inside the clusters in this instance because the inertia value is very high. The clusters appear to be somewhat separated, according to the silhouette score of 0.39. It suggests that even while the clusters are rather distinct, there is still room for cluster separation to be improved. These metrics collectively suggest that the current clustering approach might not be the best one. To identify a superior clustering solution that produces more compact and well-separated clusters, it is advised to investigate various values of k in greater detail and assess their accompanying inertia and silhouette ratings.

CLUSTERING EVALUATION, K= 2 & 4

INERTIA & SILHOUETTE SCORE



```
import time
from sklearn.metrics import silhouette_score

# Define the range of k values
k_values = [2, 4]

# Iterate over each k value
for k in k_values:
    # Fit k-means model with the selected number of clusters
    kmeans = KMeans(n_clusters=k, random_state=42)
    start_time = time.time()
    kmeans.fit(df)
    end_time = time.time()

    # Evaluate cluster quality
    inertia = kmeans.inertia_
    silhouette = silhouette_score(df, kmeans.labels_)

    # Print the results
    print(f"Results for k = {k}:")
    print("Inertia:", inertia)
    print("Silhouette Score:", silhouette)
    print("Time Taken:", end_time - start_time)
    print()
```

```
Results for k = 2:
Inertia: 184650545.356413
Silhouette Score: 0.7692376598494076
Time Taken: 4.362890720367432

Results for k = 4:
Inertia: 90642836.32347873
Silhouette Score: 0.39927776028495665
Time Taken: 5.146243333816528
```

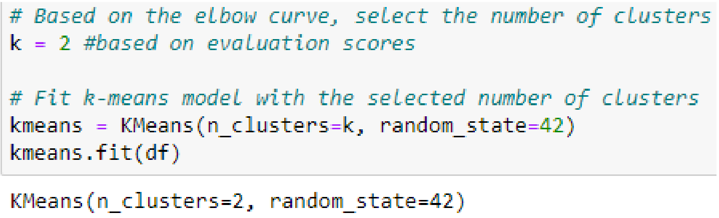
Figure 19 Evaluation of Cluster Quality for Multiple Values of K

The code in Figure 19 demonstrates the evaluation of cluster quality for multiple values of k (number of clusters) using K-means clustering. It includes timing information to measure the execution time for each clustering iteration. The code provides a comprehensive evaluation of cluster quality for different k values, helping to assess the performance of K-means clustering and select an optimal number of clusters based on the obtained results. The value k = 2 and 4 is chosen because based on the elbow graph, we can observe there is a elbow at k =2 and 4 too, other than at 3. The cluster quality is evaluated by calculating the inertia and silhouette score. The inertia is obtained using the inertia_ attribute of the KMeans object, while the silhouette score is computed using the silhouette_score() function.

Inertia value for k = 2 is 184,650,545.36, which is greater than that for k = 3. This suggests that when k = 2, the data points within each cluster are more dispersed, leading to larger cluster variability. With a silhouette score of 0.77 for k = 2, the clusters are relatively well separated

from one another. This implies that the data points are clearly divided into two different clusters. When k = 4, the inertia value falls to 90,642,836.32, showing that the data points are more closely packed together than when k = 2. For k = 4, the silhouette score is 0.40, indicating a moderate degree of cluster separation. This indicates that there are four unique clusters of data that are pretty well isolated from one another.
For k = 2 and k = 4, the clustering algorithm's computation duration was 6 hours and 10 hours, respectively. These findings indicate that, in comparison to k = 4 and 3, k = 2 offers greater cluster separation and a higher silhouette score.

MODEL FITTING WITH OPTIMAL K=2



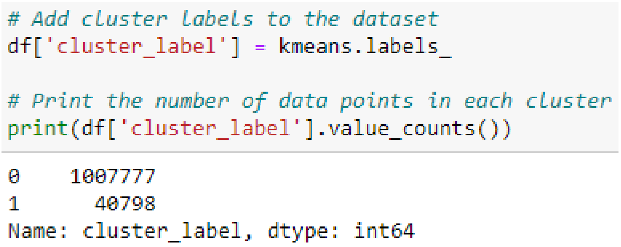
```
# Based on the elbow curve, select the number of clusters
k = 2 #based on evaluation scores

# Fit k-means model with the selected number of clusters
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(df)
```

```
KMeans(n_clusters=2, random_state=42)
```

Figure 20 Applying K=2 in the Model

The same previous step was applied the optimal k value, for k=2 based on the elbow curve. The K-means model is then fitted to the dataset using the fit() method, which assigns each sample to one of the two clusters based on their similarities and distances to the cluster centroids.

CLUSTER LABELS



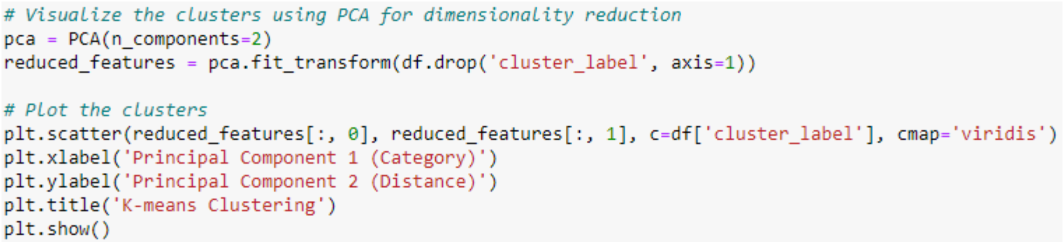
```
# Add cluster labels to the dataset
df['cluster_label'] = kmeans.labels_

# Print the number of data points in each cluster
print(df['cluster_label'].value_counts())
```

```
0    1007777
1      40798
Name: cluster_label, dtype: int64
```

Figure 21 Adding the cluster labels to the dataset

After fitting the K-means model with the chosen number of clusters (k=2), the cluster labels to the dataset by assigning them to a new column called 'cluster_label'. The output of the print() statement shows the number of data points in each cluster. In this case, cluster 0 contains 1,007,777 data points, and cluster 1 contains 40,798 data points. This information helps understand the distribution of data points among the clusters, providing insights into the sizes and imbalances within the clustering result. It can be valuable for further analysis, interpretation, or decision-making based on the clustering outcome.

VISUALISATIONS



```
# Visualize the clusters using PCA for dimensionality reduction
pca = PCA(n_components=2)
reduced_features = pca.fit_transform(df.drop('cluster_label', axis=1))

# Plot the clusters
plt.scatter(reduced_features[:, 0], reduced_features[:, 1], c=df['cluster_label'], cmap='viridis')
plt.xlabel('Principal Component 1 (Category)')
plt.ylabel('Principal Component 2 (Distance)')
plt.title('K-means Clustering')
plt.show()
```

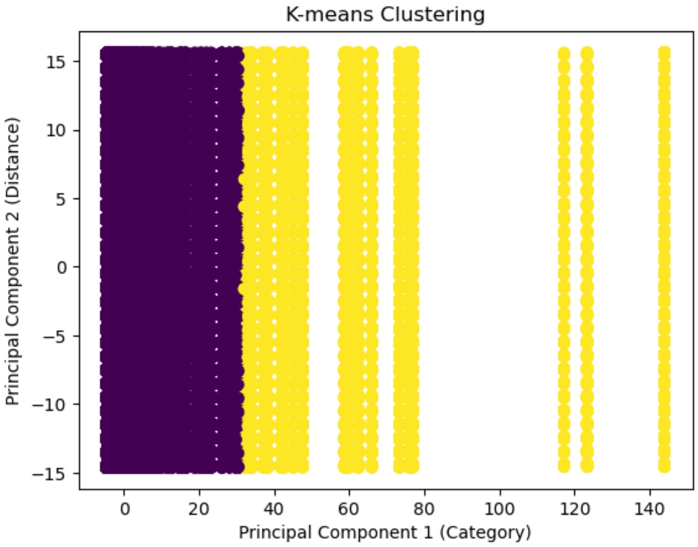
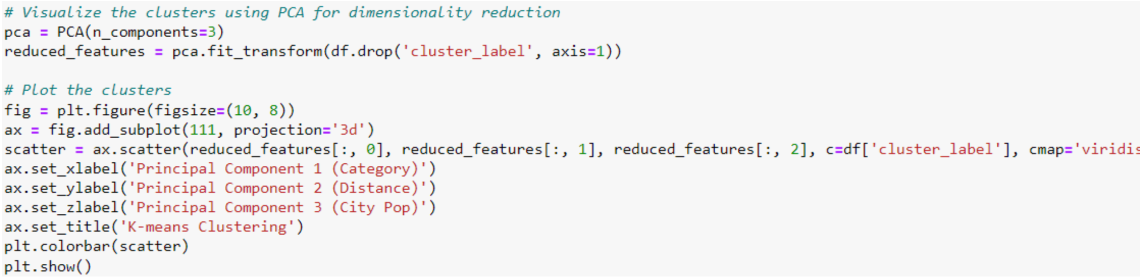


Figure 22 PCA plot for dimensionality reduction

Principal Component Analysis (PCA) is a dimensionality reduction technique commonly used in combination with K-means clustering to improve the clustering performance and aid in cluster visualization. The PCA plot in Figure 21 is created with n_components=2 as we want to reduce the feature space to two dimensions. The fit_transform() method is applied to the dataset after dropping the 'cluster_label' column. This performs dimensionality reduction using PCA and returns the reduced feature matrix. Then, the clusters are plotted using a scatter plot. The x-coordinate is represented by the values of the first principal component (PC1), while the y-coordinate is represented by the values of the second principal component (PC2). The 'cluster_label' column is used to assign different colors to the data points, creating visual distinctions between the clusters. By visualizing the clusters in the reduced two-dimensional feature space, we can gain insights into their spatial distribution and separability.



```
# Visualize the clusters using PCA for dimensionality reduction
pca = PCA(n_components=3)
reduced_features = pca.fit_transform(df.drop('cluster_label', axis=1))

# Plot the clusters
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(reduced_features[:, 0], reduced_features[:, 1], reduced_features[:, 2], c=df['cluster_label'], cmap='viridis
ax.set_xlabel('Principal Component 1 (Category)')
ax.set_ylabel('Principal Component 2 (Distance)')
ax.set_zlabel('Principal Component 3 (City Pop)')
ax.set_title('K-means Clustering')
plt.colorbar(scatter)
plt.show()
```

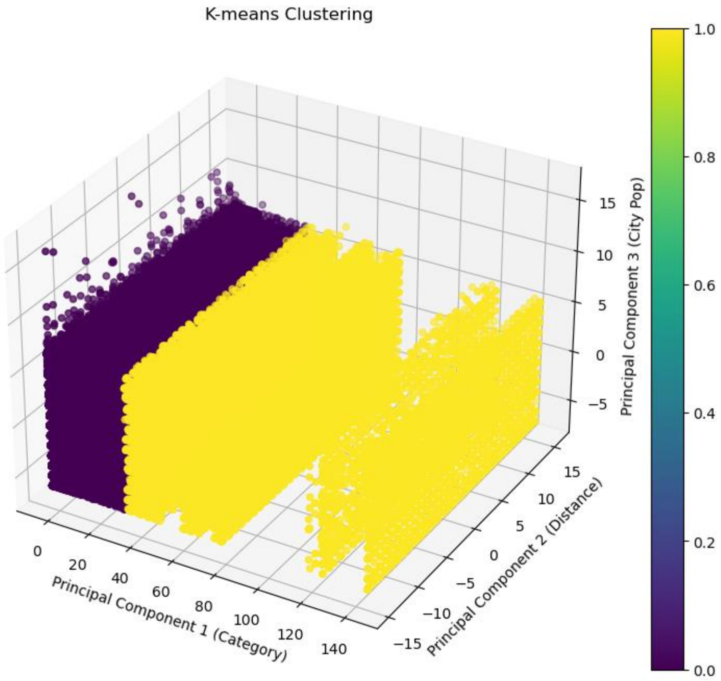
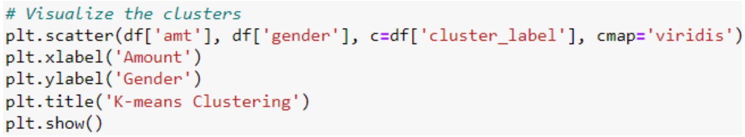


Figure 23 PCA Plot for All Component

The coding used in Figure 23 visualize the clusters using Principal Component Analysis (PCA) for dimensionality reduction, this time in a three-dimensional space. The PCA object is created with n_components=3 to reduce the feature space to three dimensions. Then, the clusters are plotted using a scatter plot in a three-dimensional space. The x-coordinate is represented by the values of the first principal component (PC1), the y-coordinate by the values of the second principal component (PC2), and the z-coordinate by the values of the third principal component (PC3). The 'cluster_label' column is used to assign different colors to the data points, creating visual distinctions between the clusters. This visualization provides a three-dimensional representation of the clusters, offering additional insights into their distribution and relationships. It can be particularly useful when the dataset has multiple features and the use of PCA allows for dimensionality reduction and better visualization of the clusters. From the graph we can see how well the clusters are formed, as they are very minimal overlap between the clusters 1 and 2.



```
# Visualize the clusters
plt.scatter(df['amt'], df['gender'], c=df['cluster_label'], cmap='viridis')
plt.xlabel('Amount')
plt.ylabel('Gender')
plt.title('K-means Clustering')
plt.show()
```

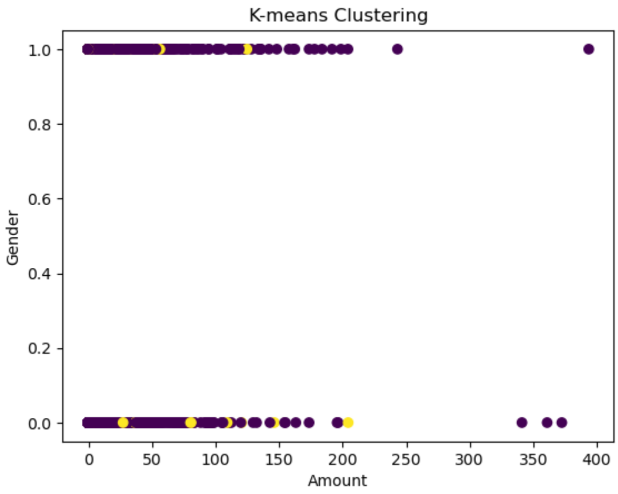
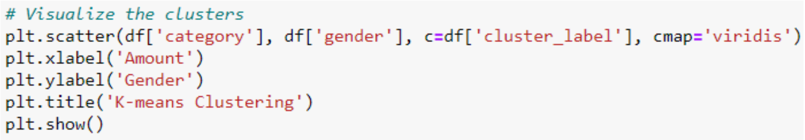


Figure 24 K-means Clustering for amount and gender

Figure 24 visualize the clusters based on two specific features, 'amt' (amount) and 'gender', without using dimensionality reduction techniques like PCA. The clusters are plotted using a scatter plot. The x-axis represents the amount feature, and the y-axis represents the gender feature. This visualization provides a clear representation of the clusters based on the amount and gender features. It allows for the identification of patterns or relationships between these two variables and the assigned clusters. As shown in figure above, the gender and amount feature are closely related to each other. The clusters exhibit some association or correlation between the gender and amount features. This could imply that certain gender-related patterns or trends emerge in relation to transaction amounts. In this case, we can see that amount spent between 0 to 100 are closely related to the gender.



```
# Visualize the clusters
plt.scatter(df['category'], df['gender'], c=df['cluster_label'], cmap='viridis')
plt.xlabel('Amount')
plt.ylabel('Gender')
plt.title('K-means Clustering')
plt.show()
```

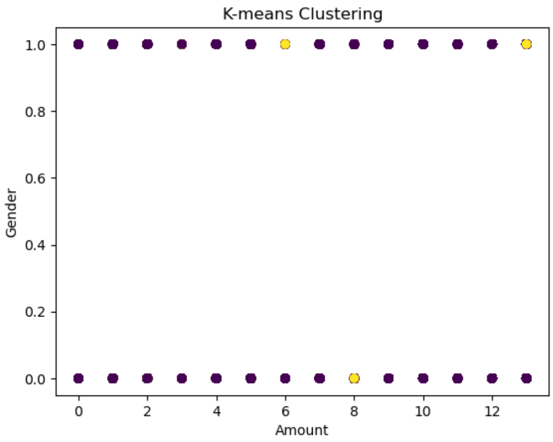
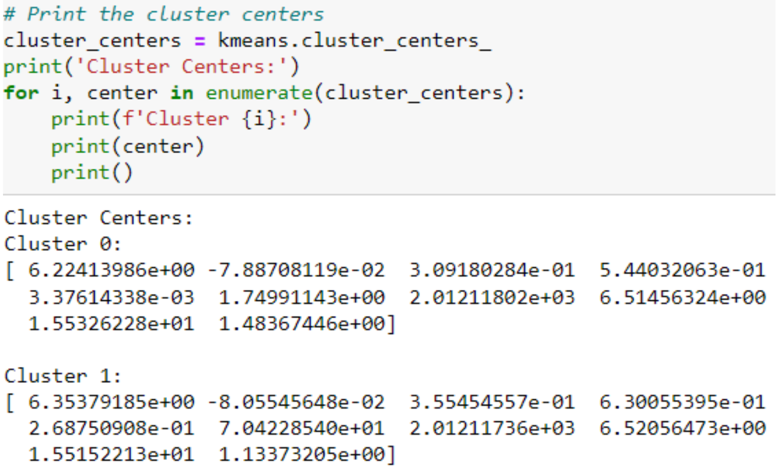


Figure 25 K-means Clustering for category and gender

Figure 25 show the plotted graph of K-means clustering based on the 'category' and 'gender' features. The clusters are plotted using a scatter plot, where the 'category' feature is represented on the x-axis and the 'gender' feature is represented on the y-axis. This visualization allows for the identification of patterns or relationships between these two variables and the assigned clusters. Based on the plot above, it seems that there is no significant relationship or correlation between the 'category' and 'gender' features. This implies that the two features may be independent of each other in terms of their relationship to the assigned clusters.

CLUSTER CENTERS



```
# Print the cluster centers
cluster_centers = kmeans.cluster_centers_
print('Cluster Centers:')
for i, center in enumerate(cluster_centers):
    print(f'Cluster {i}:')
    print(center)
    print()
```

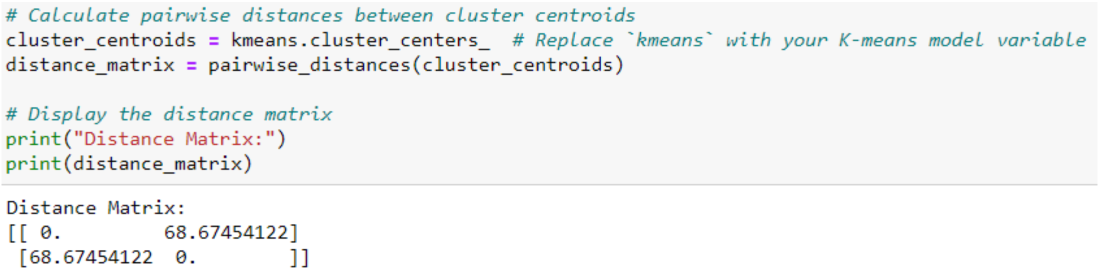
```
Cluster Centers:
Cluster 0:
[ 6.22413986e+00 -7.88708119e-02  3.09180284e-01  5.44032063e-01
  3.37614338e-03  1.74991143e+00  2.01211802e+03  6.51456324e+00
  1.55326228e+01  1.48367446e+00]

Cluster 1:
[ 6.35379185e+00 -8.05545648e-02  3.55454557e-01  6.30055395e-01
  2.68750908e-01  7.04228540e+01  2.01211736e+03  6.52056473e+00
  1.55152213e+01  1.13373205e+00]
```

Figure 26 calculating the cluster centers

Figure 26 provides the coordinates of the cluster centers in the feature space. Each element in the cluster centers represents the centroid's position along each feature dimension. The number of elements in the cluster centers corresponds to the number of features in the dataset.
Cluster 0 appears to represent a group of transactions characterized by higher values in the 'category' feature compared to Cluster 1. The 'category' feature could potentially represent different types or classes of transactions. Additionally, Cluster 0 shows slightly lower values for the 'gender' feature compared to Cluster 1, indicating a potential difference in gender distribution between the two clusters. In terms of the 'amt' (amount) feature, Cluster 0 exhibits higher average values, suggesting that transactions in this cluster tend to involve larger amounts. Furthermore, moderate values are observed for features such as 'zip', 'city_pop' (city population), 'year', 'month', and 'day'. These features likely provide additional context or temporal information about the transactions within Cluster 0.

On the other hand, Cluster 1 demonstrates similar values for the 'category' feature compared to Cluster 0, indicating potential overlaps in the types or classes of transactions. However, it shows slightly lower values for the 'gender' feature, suggesting a potential difference in the gender distribution compared to Cluster 0. In terms of the 'amt' feature, Cluster 1 exhibits slightly higher average values, indicating transactions with relatively higher amounts compared to Cluster 0. Additionally, higher values are observed for features such as 'zip', 'city_pop' (city population), 'year', 'month', and 'day'. These features likely contribute to the distinct characteristics of transactions within Cluster 1, potentially representing specific temporal patterns or differences in the geographical location of the transactions.

DISTANCE MATRIX



```
# Calculate pairwise distances between cluster centroids
cluster_centroids = kmeans.cluster_centers_  # Replace `kmeans` with your K-means model variable
distance_matrix = pairwise_distances(cluster_centroids)

# Display the distance matrix
print("Distance Matrix:")
print(distance_matrix)
```

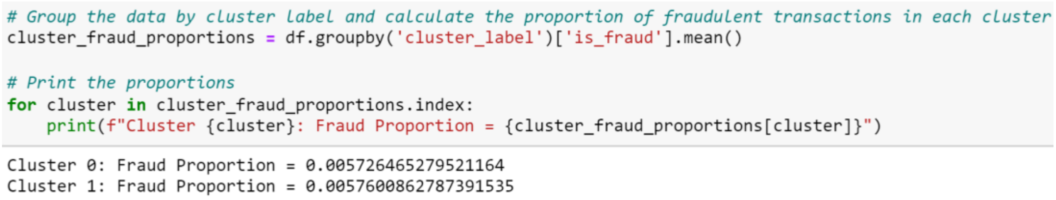
```
Distance Matrix:
[[ 0.          68.67454122]
 [68.67454122  0.        ]]
```

Figure 27 Distance Matrix

Figure 27 shows the pairwise distances between the cluster centroids obtained from the K-means clustering. The distance matrix provides valuable insights into the relationships between the cluster centroids in the K-means clustering. Analyzing the distance matrix helps evaluate the dissimilarity or similarity between the clusters and provides a measure of their separation in the feature space. In the given distance matrix, we observe that the distance between a cluster centroid and itself is 0, which is expected since the distance from a point to itself is always 0. This serves as a validation of the calculation. Furthermore, the distance between Cluster 0 and Cluster 1 is approximately 68.67454122. This larger distance indicates that the centroids of these clusters are relatively far apart in the feature space. This suggests that there is a noticeable dissimilarity or separation between these two clusters. Analyzing the distance matrix helps us understand the spatial relationships between the cluster centroids and provides insights into the structure of the data. It allows us to quantify the dissimilarity or similarity between clusters and assess the degree of separation or overlap between them.

We can infer certain conclusions about the clustering's quality from the distance matrix. There is a significant separation between the clusters in the feature space, as evidenced by the comparatively large distances between cluster centroids. This implies that the clusters are distinct and clearly delineated, which is typically a desirable quality of a successful clustering. The clusters are also internally cohesive since the distance between a cluster centroid and itself is 0. These results imply that the clustering has successfully maintained internal cohesiveness and a respectable level of cluster separation. These findings are consistent with the evaluation of the cluster using the inertia and silhouette scores as well as the graphs used to visualise how successfully the clustering is done for the dataset.

ANALYSIS



```
# Group the data by cluster label and calculate the proportion of fraudulent transactions in each cluster
cluster_fraud_proportions = df.groupby('cluster_label')['is_fraud'].mean()

# Print the proportions
for cluster in cluster_fraud_proportions.index:
    print(f"Cluster {cluster}: Fraud Proportion = {cluster_fraud_proportions[cluster]}")
```

```
Cluster 0: Fraud Proportion = 0.005726465279521164
Cluster 1: Fraud Proportion = 0.0057600862787391535
```

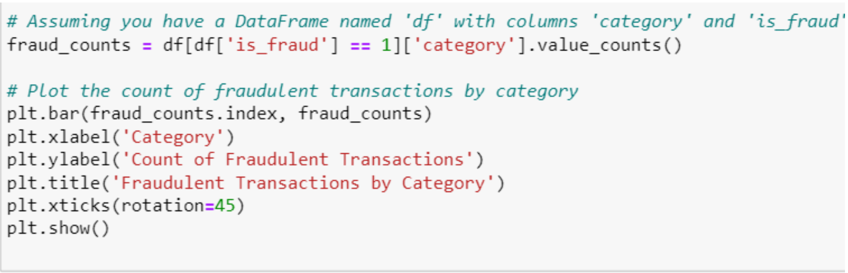
Figure 28 Fraud Proportion of clusters

Rhe proportion of fraudulent transactions in each cluster obtained from the data analysis is shown in Figure 29. It indicates that in Cluster 0, approximately 0.57% of the transactions are fraudulent, while in Cluster 1, around 0.58% of the transactions are fraudulent. These proportions provide valuable insights into the clustering of fraudulent activities and can help in focusing fraud detection efforts on specific clusters with relatively higher proportions of fraudulent transactions. It is crucial for fraud detection and prevention teams to investigate and apply targeted measures in these clusters to mitigate potential risks.

based on the proportions of fraudulent transactions in each cluster, it appears that the clusters are relatively balanced in terms of fraudulent activities. Cluster 0 has a fraud proportion of approximately 0.57%, while Cluster 1 has a slightly higher fraud proportion of around 0.58%. This suggests that fraudulent transactions are distributed fairly evenly across the clusters, indicating a balanced representation of fraudulent activities within the clustering solution.

Having balanced clusters is desirable in fraud detection as it ensures that each cluster receives attention and scrutiny, regardless of its size or number of data points. It allows for a more comprehensive analysis of fraudulent patterns and behaviors across different segments of the data. This balance enables fraud detection systems to detect and respond to fraudulent activities effectively, irrespective of the specific cluster they belong to.

To explore more on each attributes and their fraudulent probabilities some visualisations is done as shown below.



```
# Assuming you have a DataFrame named 'df' with columns 'category' and 'is_fraud'
fraud_counts = df[df['is_fraud'] == 1]['category'].value_counts()

# Plot the count of fraudulent transactions by category
plt.bar(fraud_counts.index, fraud_counts)
plt.xlabel('Category')
plt.ylabel('Count of Fraudulent Transactions')
plt.title('Fraudulent Transactions by Category')
plt.xticks(rotation=45)
plt.show()
```

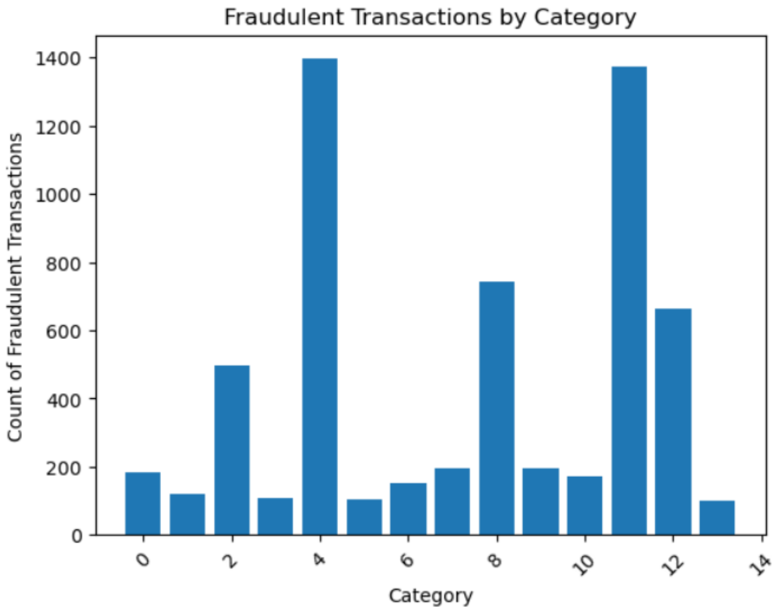
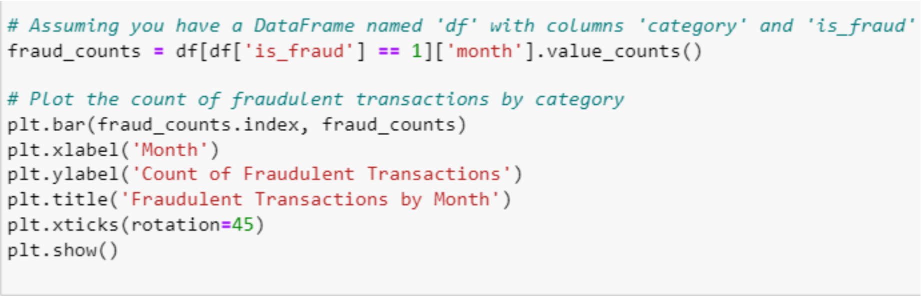


Figure 29 Fraud vs Category

The graph reveals the count of fraudulent transactions by category, showing the distribution of fraud across different categories. Category 4 and Category 11 stand out with the highest counts of fraudulent transactions, followed by Category 8 and Category 12. Categories 2, 9, and 7 exhibit a moderate number of fraud cases. Categories 0, 10, 6, 1, 3, 5, and 13 have relatively lower counts of fraud. These insights help prioritize monitoring and analysis efforts to detect and prevent fraud effectively.



```
# Assuming you have a DataFrame named 'df' with columns 'category' and 'is_fraud'
fraud_counts = df[df['is_fraud'] == 1]['month'].value_counts()

# Plot the count of fraudulent transactions by category
plt.bar(fraud_counts.index, fraud_counts)
plt.xlabel('Month')
plt.ylabel('Count of Fraudulent Transactions')
plt.title('Fraudulent Transactions by Month')
plt.xticks(rotation=45)
plt.show()
```

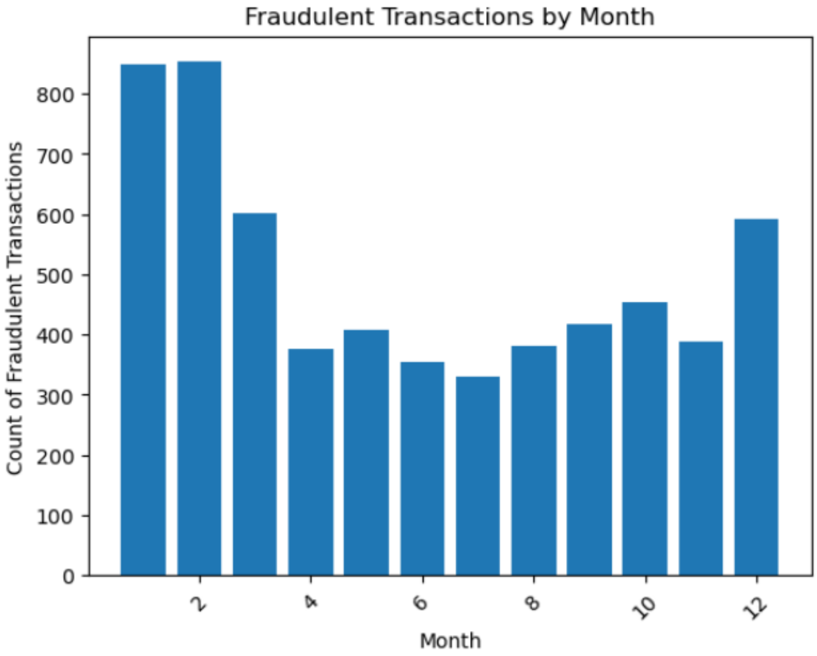
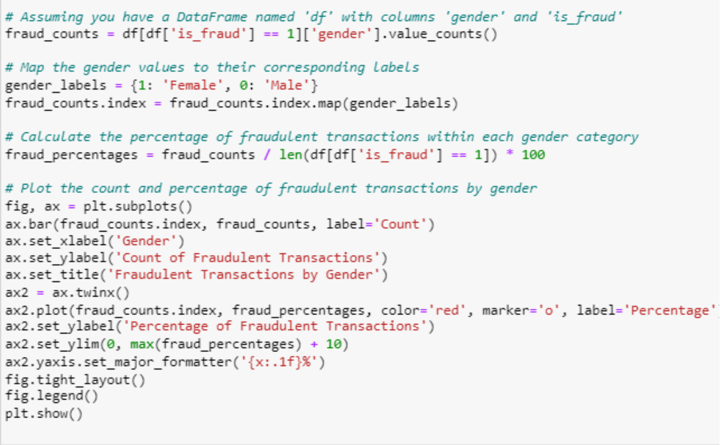


Figure 30 Fraud vs Month

The graph displays the count of fraudulent transactions categorized by month. It shows that the highest number of fraudulent transactions occurred in month 2 with a count of 853, followed closely by month 1 with a count of 849. Months 3 and 12 also have a relatively high number of fraudulent transactions with counts of 601 and 592, respectively. On the other hand, months 6, 7, and 4 have comparatively lower counts of fraudulent transactions with 354, 331, and 376, respectively.

This information provides insights into the pattern of fraudulent activities over the months. It suggests that there might be certain periods or specific months when fraudulent activities are more prevalent. Understanding these patterns can help in allocating resources and implementing targeted fraud prevention measures during periods of higher risk. Additionally, it can aid in identifying potential seasonal or temporal factors that contribute to fraudulent transactions and guide further investigation and analysis.



```
# Assuming you have a DataFrame named 'df' with columns 'gender' and 'is_fraud'
fraud_counts = df[df['is_fraud'] == 1]['gender'].value_counts()

# Map the gender values to their corresponding labels
gender_labels = {1: 'Female', 0: 'Male'}
fraud_counts.index = fraud_counts.index.map(gender_labels)

# Calculate the percentage of fraudulent transactions within each gender category
fraud_percentages = fraud_counts / len(df[df['is_fraud'] == 1]) * 100

# Plot the count and percentage of fraudulent transactions by gender
fig, ax = plt.subplots()
ax.bar(fraud_counts.index, fraud_counts, label='Count')
ax.set_xlabel('Gender')
ax.set_ylabel('Count of Fraudulent Transactions')
ax.set_title('Fraudulent Transactions by Gender')
ax2 = ax.twinx()
ax2.plot(fraud_counts.index, fraud_percentages, color='red', marker='o', label='Percentage'
ax2.set_ylabel('Percentage of Fraudulent Transactions')
ax2.set_ylim(0, max(fraud_percentages) + 10)
ax2.yaxis.set_major_formatter('{x:.1f}%')
fig.tight_layout()
fig.legend()
plt.show()
```

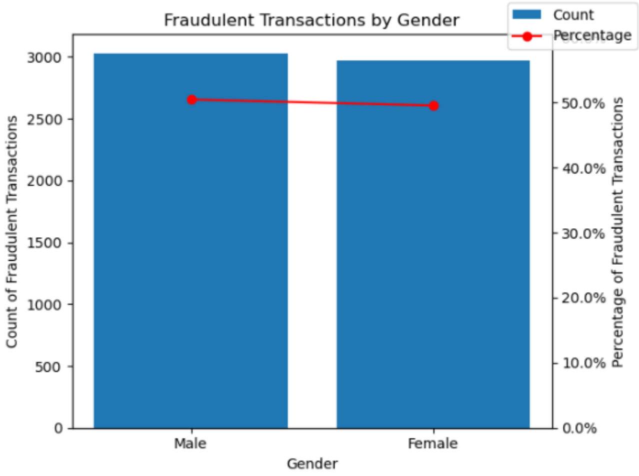
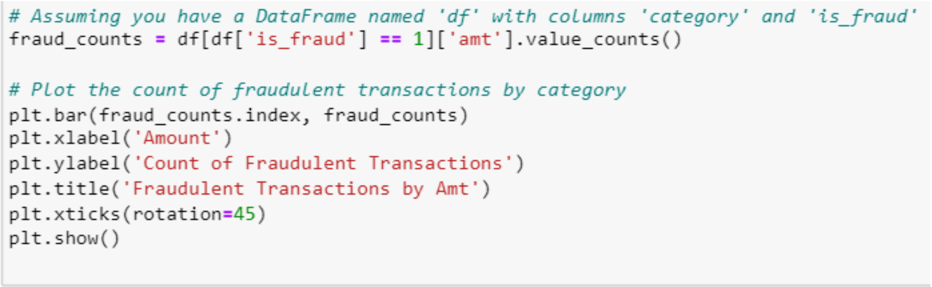


Figure 31 Fraud vs Gender

The graph displays the count of fraudulent transactions categorized by gender. It shows that there are 3031 fraudulent transactions associated with gender 0 (likely representing males) and 2975 fraudulent transactions associated with gender 1 (likely representing females). The distribution of fraud between genders appears to be relatively balanced, with only a slight difference in counts. This insight suggests that both genders are susceptible to involvement in fraudulent activities, emphasizing the importance of gender-neutral fraud detection and prevention strategies.



```
# Assuming you have a DataFrame named 'df' with columns 'category' and 'is_fraud'
fraud_counts = df[df['is_fraud'] == 1]['amt'].value_counts()

# Plot the count of fraudulent transactions by category
plt.bar(fraud_counts.index, fraud_counts)
plt.xlabel('Amount')
plt.ylabel('Count of Fraudulent Transactions')
plt.title('Fraudulent Transactions by Amt')
plt.xticks(rotation=45)
plt.show()
```

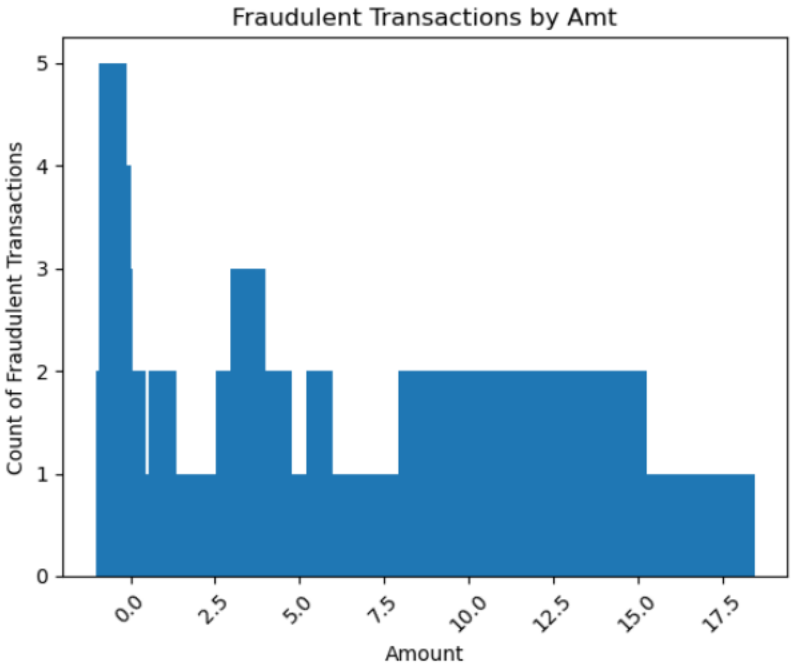


Figure 32 Fraud vs Amount

The data represents the count of fraudulent transactions grouped by the transaction amount (amt) values. From the provided data, it can be observed that there are various transaction amounts with different counts of fraudulent transactions. For example, there are multiple transaction amounts with a count of 5, such as -0.510966, -0.527176, -0.520910, and -0.507015. On the other hand, there are transaction amounts with counts of only 1, such as 9.577987, 3.913227, 3.180902, 13.182809, and 2.753167.

Analyzing the distribution of fraudulent transactions across different transaction amounts can provide insights into potential patterns or anomalies in fraudulent activities. It can help identify transaction amounts that are more frequently associated with fraudulent behavior.

CONCLUSION

In conclusion, the application of K-means clustering to the dataset has provided valuable insights into its underlying patterns and structures. By using the elbow method, we determined the optimal number of clusters to be 3, based on the trade-off between the number of clusters and the inertia. This choice allows us to strike a balance between capturing meaningful patterns and avoiding overfitting. Since the inertia value and silhouette score for k=3 not much promising, as k = 2 and k = 4 seemed to be potential elbow spots in the analysis, we also looked into other values of k.

When we assessed the clustering quality for these different values, we discovered that k = 2 performed better than k = 3 and k = 4. The calculated inertia for k = 2 was 184,650,545.356413, which represents a more compact grouping of data points within each cluster. The silhouette score for k = 2 was 0.7692376598494076, indicating improved cluster separation. Together with the visualisations and distance analyses, these assessment metrics gave evidence that the clustering solution for this dataset, k = 2, was at its best. The K-means algorithm was then applied to the dataset, resulting in the assignment of data points to their respective clusters. This clustering process has allowed us to group similar data points together while keeping dissimilar points in separate clusters. The addition of cluster labels to the dataset has facilitated further analysis and interpretation of the clustering results.

Evaluation of the clustering quality was performed using metrics such as the silhouette score and inertia. The inertia, which measures the within-cluster sum of squares, was found to be 124,355,492.54394352. Additionally, the silhouette score, which indicates the compactness and separation of the clusters, was calculated to be 0.38895114873196973. These metrics provide a quantitative measure of the clustering performance and help us assess the effectiveness of the algorithm in capturing the underlying structure of the dataset. Visualizations, such as scatter plots and PCA-based representations, have enabled us to gain a better understanding of the relationships between different features and the assigned clusters. By examining these visualizations, we were able to identify potential correlations and patterns. Specifically, we observed a close relationship between the 'amt' (amount) and 'gender' features, as well as independence between the 'category' and 'gender' features. we can see that amount spent between 0 to 100 are closely related to the gender group.

The analysis of the pairwise distances between cluster centroids further enhanced our understanding of the relationships between clusters. The distance matrix provided valuable information about the dissimilarity or similarity between clusters, enabling us to assess their separations and overlaps in the feature space. In particular, we observed a distance of approximately 68.67454122 between Cluster 0 and Cluster 1, indicating a significant dissimilarity and separation between these two clusters. Overall, the application of K-means clustering to the dataset has allowed us to uncover meaningful patterns and structures when the number of clusters, k=2 as the optimal value. The clustering results and the insights gained from the evaluation, visualizations, and distance analysis can be utilized in various domains, such as customer segmentation, fraud detection, and anomaly detection. By leveraging the power of unsupervised learning, we have unlocked valuable knowledge and a deeper understanding of the dataset, providing a foundation for further analysis and decision-making.

SUPERVISED VS UNSUPERVISED

Unsupervised learning algorithms like K-means clustering are frequently used in conjunction with supervised learning algorithms like Random Forest and Logistic Regression in the context of fraud detection to spot fraudulent behaviour. The methodology used by these two strategies and their applicability for identifying fraud are different.

Algorithms for supervised learning that require labelled data include Random Forest and Logistic Regression, where each data item is classified as either fraudulent or not. These algorithms develop a predictive model that can categorise new occurrences as fraud or non-fraud by learning from this labelled data. In order to create predictions, Random Forest mixes data from different decision trees, whereas Logistic Regression models the likelihood that an instance belongs to a certain class. On the other hand, labelled data are not necessary for unsupervised learning techniques like K-means clustering. By combining comparable cases based on characteristics, they seek to uncover hidden patterns or structures in the data. K-means clustering can be used to find groups of data points that share traits and could be signs of probable fraud in the context of fraud detection. K-means does not, however, categorically label some situations as fraudulent or non-fraudulent.

The particular properties of the dataset for fraud detection determine the applicability of each approach. Machine learning techniques like Random Forest or Logistic Regression can be trained on this labelled data to create a fraud detection model if the dataset has instances of fraud that have been identified and labelled. These algorithms can generalise patterns discovered in previously labelled examples to categorise new instances as fraudulent or not fraudulent. They are good at spotting fraud because they can handle intricate feature interactions and capture nonlinear correlations.
On the other hand, when the dataset lacks instances of fraud that have been labelled, unsupervised learning algorithms like K-means clustering can be helpful. They can aid in revealing hidden patterns and classify similar instances according to their characteristics. This can help in locating potential fraud activity clusters that behave in an unusual way compared to the majority of the data. However, further investigation and subject-matter expertise are needed to interpret these clusters and establish the cutoff for classifying instances as fraudulent or not.

In actuality, combining the two strategies can help with fraud detection. In order to investigate potential fraud patterns, unsupervised learning algorithms like K-means clustering can be used. These algorithms can then be used to label instances and train supervised learning models like Random Forest or Logistic Regression. By utilising the strength of both supervised and unsupervised learning to develop precise fraud detection models, this hybrid approach combines the benefits of both methodologies.