PENNSYLVANIA STATE UNIVERSITY

DATA MINING

STAT 557 - FALL 2022

PROJECT - 3

REPORT

Done by:

Denesh Kumar Manoharan [dzm5964@psu.edu]
Aadya Sanwal [afs6594@psu.edu]

Abstract:

The popularity of blockchain-based currencies, such as Bitcoin and Ethereum, has grown among enthusiasts since 2009. Relying on the anonymity provided by the blockchain, hustlers have adapted offline scams to this new ecosystem. So in this project we developed a clustering algorithm to detect the fraudulent transaction in cryptocurrency by clustering the transactions as fraudulent and valid transaction. This clustering is performed on the Ethereum fraud detection dataset from kaggle. The dataset can be downloaded here.

Exploring data:

The data consists in a series of crypto-currency transactions, each one containing 48 features describing the operation (i.e., amount sent or received, time from last transaction) and the account (i.e., average and maximum times between transactions, total token sent or received ever) performing the operation. The most important label in this data, is about the transactions that were confirmed to be fraudulent, with a tag of 0 for valid transactions, and 1 for transactions that were fraud. With 2179 fraudulent transactions (22.1%) out of a total of 9841 operations, the data set is unbalanced. The imbalance dataset is visualized in fig1.

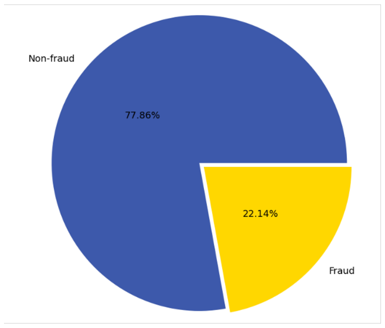


Fig 1: Pie chart visualization of class distribution

To understand the distribution of values of each features, the box plot is plotted for feature as shown in fig2, from the following plot we can observe that data is extremely skewed with maximum prediction value as zero.

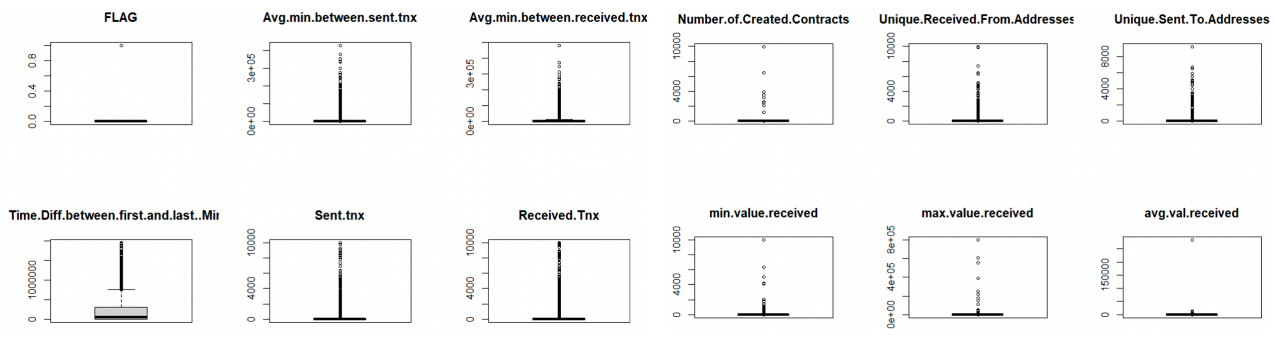


Fig 2: Box plot of features in dataset

Data Pre-processing:

- Remove variables that aren't useful: At this step, we removed the Address (a long string associated with a memory address), which was unique to the transaction but lacked any discernible pattern on its own

-Two variables, the most sent and received token type, have several missing values. These variables were strings with even more than a thousand possible values, this would imply a difficult encoding to translate this strings into numerical data. The two variables were removed

- The remaining variables are the type float. Some of them showed to have zero variance (found with the describe() command). Here, 7 variables were dropped.

- Finally, some missing values were present in the remaining columns. This were filled with the mean value of each column.

The figure 3 shows the visualization of the rows that have missing values of column

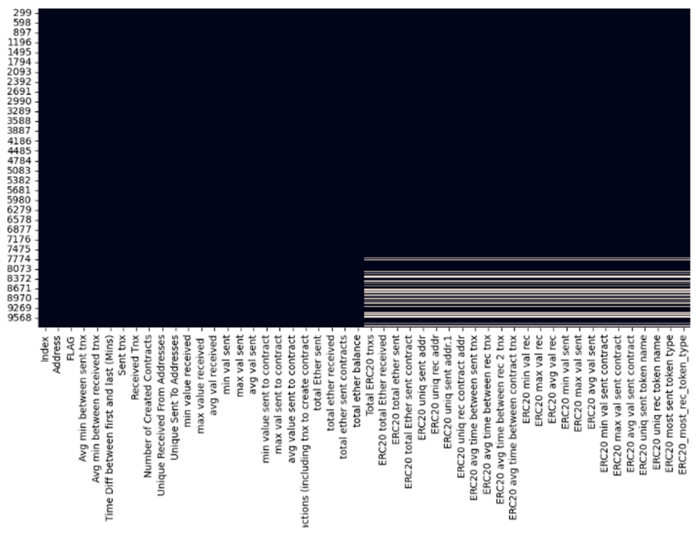


Fig 3: Map of missing values in dataset

We performed K-means clustering in this dataset, we observed that the rand index value of the clustering was very low no matter how many clusters we assume, even for the original cluster K as 2 the rand index value was very low, even zero sometimes, plotting 2D data, we observed this issue is mostly caused by outliers in the dataset, so we removed the row with any feature beyond 3.0 standard deviation from general mean. By this procedure, 120 true values of 2179 were removed from dataset. After cleaning the dataset we have 7485 observations with 34 features. Since 34 feature is higher value and would increase the necessity or more computational power, so we implemented PCA on this with explained variance 95 precent and implemented the clustering algorithm.

K-Means:

The K-mean clustering algorithm was run to separate the data into two clusters. The random initialization was considered for running K-Means multiple times to observe the results. The results are very stable and not high dependency on the random state. The observed rand index score was unchanged in most of iterations as shown in fig 4a.

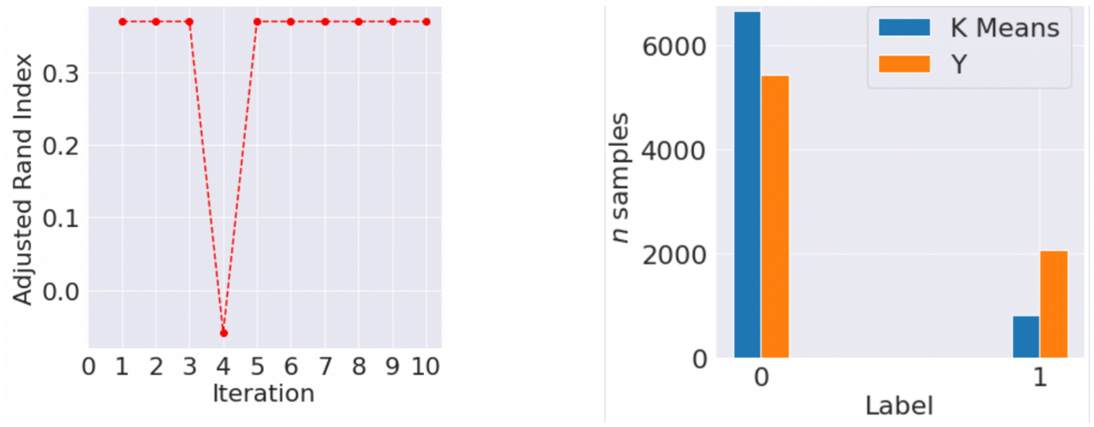


Fig 4: a) the adjusted rand index vs. the number of clusters. b) The number of samples belonging to each label in reality vs. the results obtained from the K-Mean method with K=2.

The final clustering is shown in Fig 2b were the results are compared against the actual labels. The obtained randi index score for two clustering is 0.3695. Evaluating the performance of K-Means for different number of clusters, the algorithm was run on each iteration few times, and the best result is provided by the KMeans().fit in sklearn with the keyword n_init. On each iteration, the adjusted rand index was calculated, and the results are presented in Fig 5. There can be observed that the performance of the algorithm steeply decreases when more than 2 clusters are used, obeying the nature of the data. Although some oscillations are observed after 14 clusters, the performance is much lower at that point than using the simpler case of 2 clusters. The data separation using two clusters can be observed in the Fig 5b, using the first two principal components of the data.

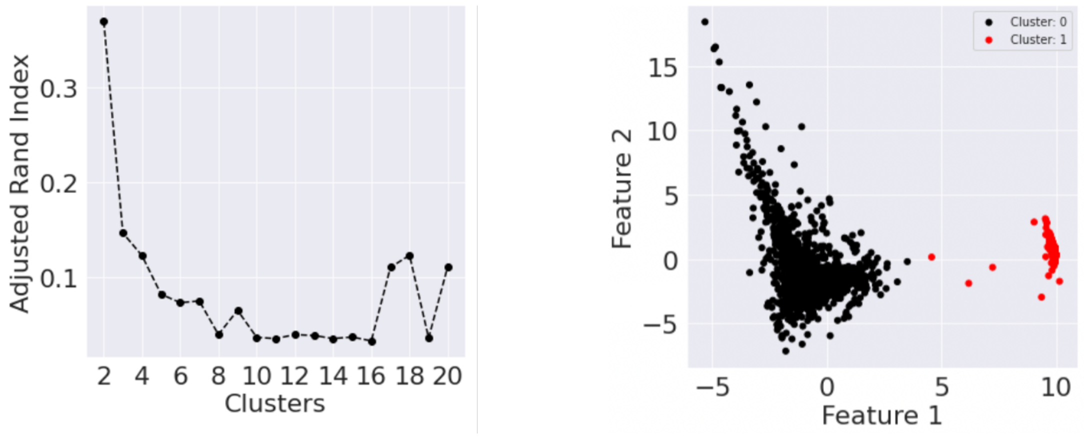


Fig 5: Performance of K-Means for different clusters and a 2D representation of a run with K=2

Gaussian Mixture:

A Gaussian Mixture is a function consisting of several Gaussians, each identified by k ∈ 1,..., K, where K is the number of clusters of the data. Therefore, a Gaussian mixture model (GMM) is a probabilistic model assuming that the data points are from a mixture of K Gaussian distributions. The GMM uses the expectation- maximization (EM) algorithm to estimate the Gaussian models' unknown parameters. In this section, we used GMM for clustering our dataset. First, using "GridSearchCV," we tried to optimize the parameters of the Gaussian mixture model by cross-validated grid- search over a parameter grid. The parameters we optimized are *covariance_type* and *tol*.*covariance_type* describes the type of covariance parameters to use that must be one of the following: spherical, diagonal, tied, or full covariance. "tol" is the convergence threshold, and the EM iterations will stop when the lower bound average gain is below this threshold value. The optimized "*covariance_type*" is "full" and optimized "tol" value is equal to 0.001. Then we trained the model using the training data while trying several different K values from 2 to 20. Then, we evaluated the results using the adjusted rand index.

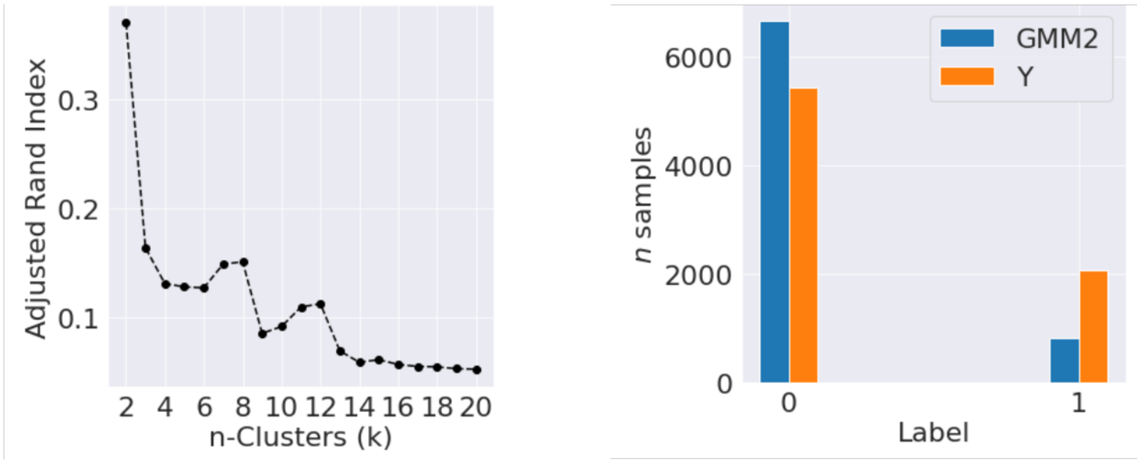


Fig6: a) the adjusted rand index vs. the number of clusters. b) The number of samples belonging to each label in reality vs. the results obtained from the GMM method with K=2.

PCA with 2 components:

On the dataset, we used PCA to compress the feature space to only two dimensions. PCA is a method for minimizing information loss while also lowering the dimensionality of huge datasets, improving interpretability. It accomplishes this by producing new, uncorrelated variables that maximize variance one after the other. Consequently, a great deal of information was lost when the feature space was reduced to 2 dimensions, and the explained variance we obtained for the dataset was 56%.

K-Means:

K-Means algorithm is applied to the reduced 2-D dataset. We ran the algorithm multiple times to make sure that the random initialization does not impact the algorithm performance. Here the number of clusters was set to 2. The 7a) is the graph between Adjusted Rand Index and Multiple runs, it is clear from the graph that random initialization did not affect the performance of the algorithm. The Figure 7b) is the comparison of K-Means Algorithm results with the true label.

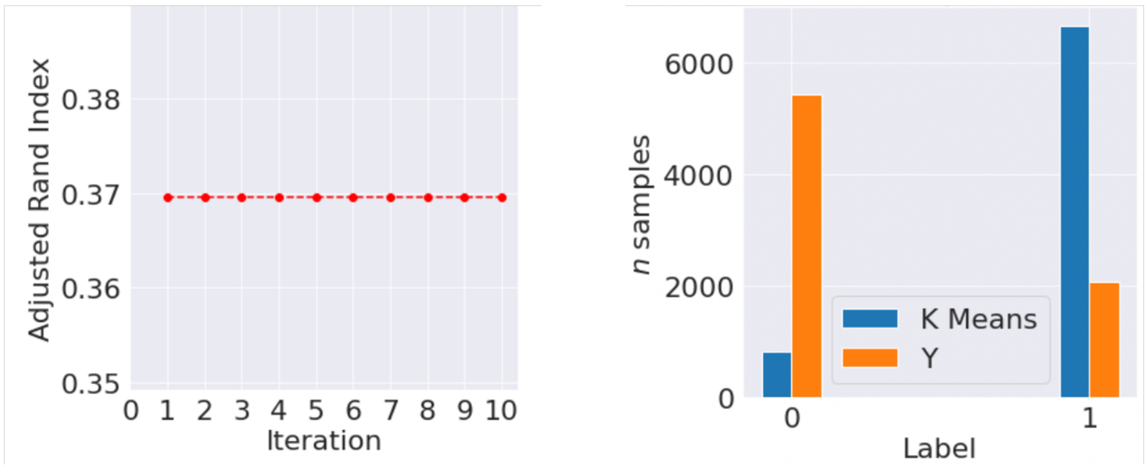


Fig7: Adjusted rand index for K-Means and comparison with the actual labels for the reduced 2-D Dataset

In fig 8a we have plotted the objective function of the K-mean clustering vs the clusters, this plot is a stoping criteria for unsupervised learning, so when we visualize from a huge change is slope from one cluster to other cluster, this will be our stoping criteria to identify the optimistic cluster in dataset. In our case we see a huge change in slop from cluster 2 to cluster 3, so the number of cluster is 2 for the dataset.

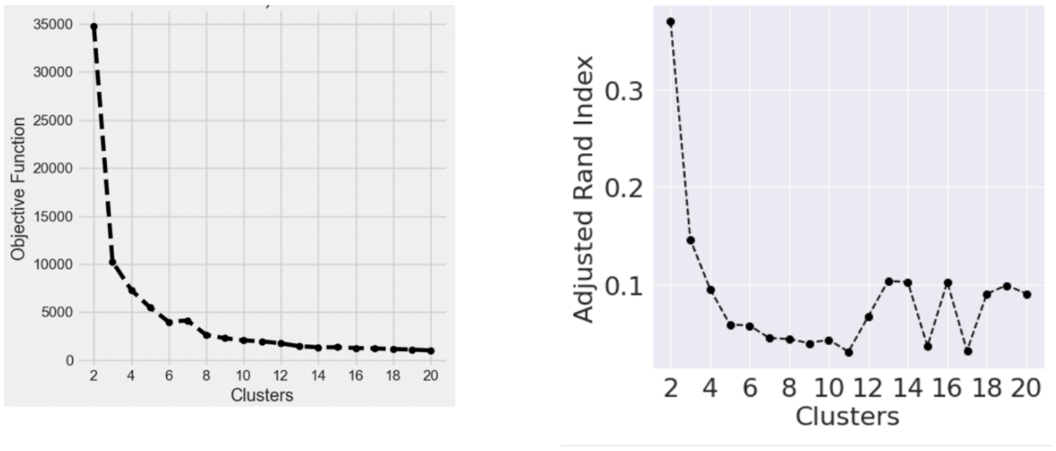


Fig8: Objective function vs clusters and adjust rand index vs clusters

Gaussian Mixture:

Gaussian Mixture algorithm is applied to the reduced 2-D dataset. We ran the algorithm multiple times to make sure that the random initialization does not impact the algorithm performance. The Fig 9 given below depicts the performance of the algorithm. Here the number of clusters was set to 2. The Fig 9a is the graph between Adjusted Rand Index and Multiple runs, it is clear from the graph that random initialization did not affect the performance of the algorithm. The Fig 9b is the comparison of Gaussian Mixture Algorithm results with the true label. So from the graphs we can come to a conclusion that Gaussian Mixture algorithm performs slightly better compared to the K-Means Algorithm.

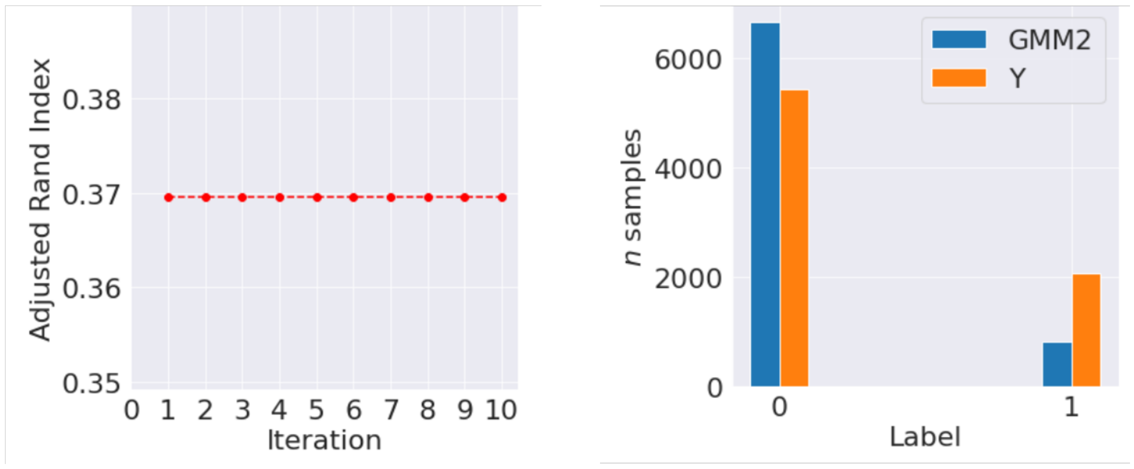


Fig9: Adjusted rand index for GMM and comparison with the actual labels for the reduced 2-D Dataset

Conclusions:

Although the random state can affect the results of K-Means, in our data set, the adjusted rand index tend to be 0.3695 in all the iteration, where 2 is the number of clusters that maximizes this error criterion

Using GMM and trying several different K we obtained the max adjusted rand score equal to 0.37 for K=2 with PCA as 2 components.

When the feature space was reduced to 2 dimensions, the 34 dimensional data is now projected to a 2 dimension due to which the class of the dataset will overlaps, Its the reason for GMM performing a little better than K-Mean clustering

When the K-Mean and GMM was performed on the 11 component PCA, we obtained the same results (Rand Index) for both clustering, I think this is because the data points are perfectly distinct and the classes are not overlapped to each other. This also proves that K-Mean cluster is a subset of Gaussian clustering.

Individual contributions:

Aadya Sanwal: Implemented K-Mean clustering and Gaussian mixture model and worked on outliers results as suggested in class presentation

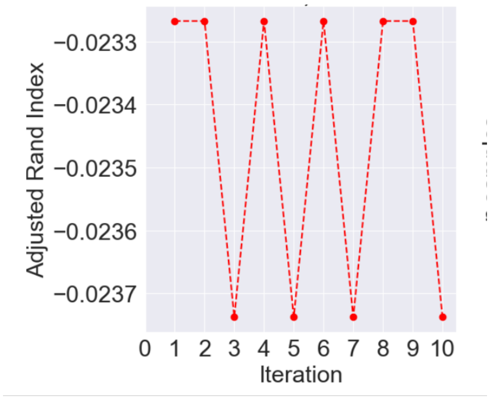
Denesh Kumar: Worked on Data exploration, preprocessing and performed PCA with 2 components, Applied K-Mean and GMM to 2 component dataset.

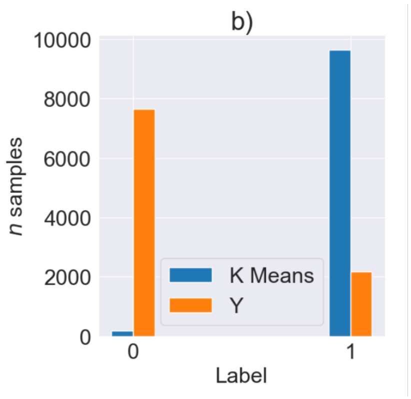
NOTE:
As suggested by professor during class presentation

We tried the K-Mean clustering without removing the outliers, and the resulted rand index value vs iteration

1)

Here we observe that the values are highly fluctuating and its all negative. The important reason why we have to remove outliers is because in random index initialization of any clustering algorithm, if the outlier is considered as the initial index then the entire cluster of data will be messes and that leads to a bad clustering results as show in below figures.

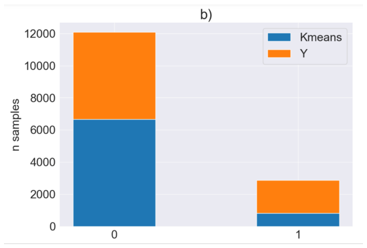




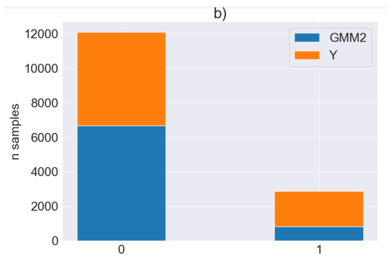
The results we obtained by excluding outlier obtained better results.

2)

Stacked bar plot [K-Means]



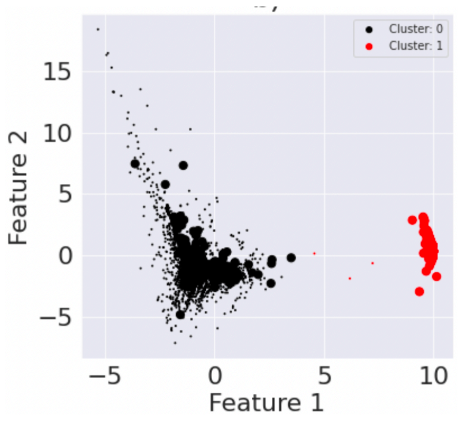
Stacked bar plot [Gaussian Mixture]



3)

Visualizing clustering of K-Means with differentiating the misclassified variables

The datapoints small in size in both class is the misclassified datapoints by K-means clustering.



CODE:

```
from matplotlib import colors
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split, validation_curve, learning_curve
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import f1_score, roc_curve, precision_score, auc, recall_score,
classification_report, confusion_matrix
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn.mixture import GaussianMixture

import seaborn as sns
import time
plt.rcParams.update({'font.size': 22})
sns.set_style("darkgrid")

def readCleanDf():
    ### This function reads our dataframe and clean it
    # according to the findings during exploratory analysis ###
    df = pd.read_csv('transaction_dataset.csv')
    df = df.drop(columns=['Unnamed: 0','Index','Address'])

    print(f'Original dataframe shape (including Y): {df.shape}')

    # Rename:
    A = df.loc[:,df.columns!='FLAG'].columns.copy()
    B = [f'F{i}' for i in range(len(A))]
    dictR = dict(zip(A,B))
    dictL = dict(zip(B,A))
    df.rename(columns=dictR, inplace=True)

    # Dropping text columns with nan and high number of unique values
    nanDF = df.loc[:,df.isna().any()].copy()
    cols = nanDF.select_dtypes(include=['object'])
    df.drop(columns=cols, inplace=True)

    # Filling the rest of the nans with mean value
    cols = nanDF.select_dtypes(include=['float64'])
    for col in cols:
        df[col].fillna(df[col].mean(), inplace=True)
    #Dropping zero variance columns
    df.drop(df.var()[df.var()==0].index, axis=1, inplace=True)

    # Renaming FLAG by Y
    df.rename(columns={"FLAG":"Y"}, inplace=True)

    # New drops removing outliers
    trueValues = np.sum(df.loc[:,'Y']==1)
    print(f'Number of true values after cleaning: {trueValues}')
    df.drop(['F0','F6', 'F13', 'F27'],axis=1, inplace=True)
    return df

def removeOutliers(df):
    trueDropped=[]
    i=0
    for col in df.columns:
        if col != 'Y':
            mean = df[col].mean()
            std = df[col].std()
            mask = np.abs(df.loc[:,col]-mean)/std>3.0

            trueDropped.append(np.sum(df.loc[mask,'Y']==1))
            print(f'Number of true values dropped from {col}: {trueDropped[i]}'); i+=1

            df.drop(df.loc[mask,col].index, inplace=True)
    print(f'Total true values dropped: {np.sum(trueDropped)}')
    return df

def removeOutliers2(df):
    for col in df.columns:
        mask = np.abs(df.loc[:,col])>3.0
        df.drop(df.loc[mask].index, inplace=True)
    return df

data = readCleanDf()
data = removeOutliers(data)
data.describe().transpose()

X = data.loc[:,data.columns!="Y"].values
Y = data.loc[:, "Y"].astype('int').values
print(X)

scaler = StandardScaler()
scaler.fit(X)
X = scaler.transform(X)
print(f'Shape of data: {X.shape}')
pd.DataFrame(X).describe().transpose()
```

```
#%% PCA
var = 0.95
pca = PCA(var)
pca.fit(X) # Fitting the PCA
print(f'Explained variance: {np.sum(pca.explained_variance_ratio_)*100}% with {pca.n_components_}
features')
X = pca.transform(X) # Applying the PCA
pd.DataFrame(X).describe().transpose()

kMScores = []
iTry =  range(1,11)
for i in iTry:
    kM = KMeans(n_clusters=2, init='random', verbose=0, n_init=5)
    kM.fit(X)
    score = adjusted_rand_score(data['Y'], kM.labels_)
    kMScores.append(score)

kMScores

#%%
fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(12,6))
ax[0].plot(iTry, kMScores, marker='o',linestyle='dashed', color='red')
ax[0].set_xlabel('Iteration')
ax[0].set_ylabel('Adjusted Rand Index')
ax[0].set_title('a)')
ax[0].set_xticks(np.arange(11))
ax[0].set_ylim()
YBars = [np.sum(Y==0), np.sum(Y==1)]
kBars = [np.sum(kM.labels_==0), np.sum(kM.labels_==1)]
offset = 0.1
width = 0.1
x = np.arange(len(['0', '1']))  # the label locations
ax[1].bar(x - offset*0.5, kBars, width, label='K Means')
ax[1].bar(x + offset*0.5, YBars, width, label='Y')
ax[1].set_xlabel('Label')
ax[1].set_ylabel('$n$ samples')
ax[1].set_title('b)')
ax[1].set_xticks(np.arange(2))
ax[1].legend(loc='best')
fig.tight_layout()

kMScores = []
kMScores2 = []
iK =  np.array(range(2,21))
for k in iK:
    kMt = KMeans(n_clusters=k, init='random', verbose=0, n_init=10)
    kMt.fit(X)
    score = adjusted_rand_score(kMt.labels_, data['Y'])
    kMScores.append(score)
    kMScores2.append(kMt.inertia_)

#%%
fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(12,6))
ax[0].plot(iK, kMScores, marker='o',linestyle='dashed', color='k')
ax[0].set_xlabel('Clusters')
ax[0].set_ylabel('Adjusted Rand Index')
ax[0].set_title('a)')
ax[0].set_xticks(iK[::2])
ax[0].set_ylim()

colores = np.array(['k', 'r', 'g','y'])
markers = np.array(['o', '+', 'g','y'])
k = 2
kM = KMeans(n_clusters=k, init='random', verbose=0, n_init=10)
kM.fit(X)

for i in range(k):
    mask = kM.labels_==i
    ax[1].scatter(X[mask,0],X[mask,1],s=25, c=colores[kM.labels_[mask]],label=f'Cluster: {i}')
ax[1].set_xlabel('Feature 1')
ax[1].set_ylabel('Feature 2')
ax[1].set_title('b)')
ax[1].legend(fontsize=10)
fig.tight_layout()

plt.plot(iK,kMScores2)

# %%
from sklearn.model_selection import GridSearchCV
parameters = {'covariance_type':['full', 'tied', 'diag', 'spherical'],'tol':[1e-3,1e-4,1e-5,1e-6]}
GMM2_Tn = GaussianMixture(n_components=2, max_iter=5000, random_state=42)
clf = GridSearchCV(GMM2_Tn, parameters, n_jobs=-1)
clf.fit(X)
clf.best_params_

# %%
# Gaussian Mixture Model
# number of clusters equal to k=2
GMM2 = GaussianMixture(n_components=2, max_iter=5000, random_state=42)
GMM2.fit(X)
GMM2_pred=GMM2.predict(X)
# Evaluate the results of GMM2 using adjusted rand index
GMM2_score=adjusted_rand_score(data['Y'], GMM2_pred)

GMM2_score

GMM_Scores=[]
```

```
j=  np.array(range(2,21))
for k in j:
    GMM = GaussianMixture(n_components=k, max_iter=5000, random_state=42)
    GMM.fit(X)
    GMM_pred=GMM.predict(X)

    # Evaluate the results of GMM using adjusted rand index
    GMM_score=adjusted_rand_score(data['Y'], GMM_pred)

    GMM_Scores.append(GMM_score)

# visualization
fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(12,6))
ax[0].plot(j, GMM_Scores, marker='o',linestyle='dashed', color='k')
ax[0].set_xlabel('n-Clusters (k)')
ax[0].set_ylabel('Adjusted Rand Index')
ax[0].set_title('a)')
ax[0].set_xticks(iK[::2])
ax[0].set_ylim()

YBars = [np.sum(Y==0), np.sum(Y==1)]
GMM2Bars = [np.sum(GMM2_pred==0), np.sum(GMM2_pred==1)]
offset = 0.1
width = 0.1
x = np.arange(len(['0', '1']))  # the label locations

ax[1].bar(x - offset*0.5, GMM2Bars, width, label='GMM2')
ax[1].bar(x + offset*0.5, YBars, width, label='Y')
ax[1].set_xlabel('Label')
ax[1].set_ylabel('$n$ samples')
ax[1].set_title('b)')
ax[1].set_xticks(np.arange(2))
ax[1].legend(loc='best')

fig.tight_layout()


GMM2_score

plt.figure(figsize=(12,6))
plt.scatter(kMScores, GMM_Scores, marker='+',linestyle='-',linewidth=3, color='green')
plt.xlabel('Kmeans Score')
plt.ylabel('Gaussian Mixure Score')
plt.title('Comparision of Algorithms')
plt.ylim()

# pca with components = 2

M = data.loc[:,data.columns!="Y"].values
```

```
N = data.loc[:, "Y"].astype('int').values

scaler = StandardScaler()
scaler.fit(M)
M = scaler.transform(M)
print(f'Shape of data: {M.shape}')
pd.DataFrame(M).describe().transpose()




#%% PCA
pca = PCA(n_components=2)
pca.fit(M) # Fitting the PCA
print(f'Explained variance: {np.sum(pca.explained_variance_ratio_)*100}% with {pca.n_components_}
features')
M = pca.transform(M) # Applying the PCA
pd.DataFrame(M).describe().transpose()

kMScores = []
iTry =  range(1,11)
for i in iTry:
    kM = KMeans(n_clusters=2, init='random', verbose=0, n_init=5)
    kM.fit(M)
    score = adjusted_rand_score(data['Y'], kM.labels_)
    kMScores.append(score)

kMScores

#%%
fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(12,6))
ax[0].plot(iTry, kMScores, marker='o',linestyle='dashed', color='red')
ax[0].set_xlabel('Iteration')
ax[0].set_ylabel('Adjusted Rand Index')
ax[0].set_title('a)')
ax[0].set_xticks(np.arange(11))
ax[0].set_ylim()
YBars = [np.sum(N==0), np.sum(N==1)]
kBars = [np.sum(kM.labels_==0), np.sum(kM.labels_==1)]
offset = 0.1
width = 0.1
x = np.arange(len(['0', '1']))  # the label locations
ax[1].bar(x - offset*0.5, kBars, width, label='K Means')
ax[1].bar(x + offset*0.5, YBars, width, label='Y')
ax[1].set_xlabel('Label')
ax[1].set_ylabel('$n$ samples')
ax[1].set_title('b)')
ax[1].set_xticks(np.arange(2))
```

```
ax[1].legend(loc='best')
fig.tight_layout()

kMScores = []
kMScores2 = []
iK =  np.array(range(2,21))
for k in iK:
    kMt = KMeans(n_clusters=k, init='random', verbose=0, n_init=10)
    kMt.fit(M)
    score = adjusted_rand_score(kMt.labels_, data['Y'])
    kMScores.append(score)
    kMScores2.append(kMt.inertia_)

#%%
fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(12,6))
ax[0].plot(iK, kMScores, marker='o',linestyle='dashed', color='k')
ax[0].set_xlabel('Clusters')
ax[0].set_ylabel('Adjusted Rand Index')
ax[0].set_title('a)')
ax[0].set_xticks(iK[::2])
ax[0].set_ylim()

colores = np.array(['k', 'r', 'g','y'])
markers = np.array(['o', '+', 'g','y'])
k = 2
kM = KMeans(n_clusters=k, init='random', verbose=0, n_init=10)
kM.fit(M)

for i in range(k):
    mask = kM.labels_==i
    ax[1].scatter(M[mask,0],M[mask,1],s=25, c=colores[kM.labels_[mask]],label=f'Cluster: {i}')
ax[1].set_xlabel('Feature 1')
ax[1].set_ylabel('Feature 2')
ax[1].set_title('b)')
ax[1].legend(fontsize=10)
fig.tight_layout()

kM.labels_

# %%
from sklearn.model_selection import GridSearchCV
parameters = {'covariance_type':['full', 'tied', 'diag', 'spherical'],'tol':[1e-3,1e-4,1e-5,1e-6]}
GMM2_Tn = GaussianMixture(n_components=2, max_iter=5000, random_state=42)
clf = GridSearchCV(GMM2_Tn, parameters, n_jobs=-1)
clf.fit(M)
clf.best_params_

# %%
# Gaussian Mixture Model
# number of clusters equal to k=2
GMM2 = GaussianMixture(n_components=2, max_iter=5000, random_state=42)
GMM2.fit(M)
GMM2_pred=GMM2.predict(M)
# Evaluate the results of GMM2 using adjusted rand index
GMM2_score=adjusted_rand_score(data['Y'], GMM2_pred)

GMM2_score

GMM_Scores=[]

j=  np.array(range(2,21))
for k in j:
    GMM = GaussianMixture(n_components=k, max_iter=5000, random_state=42)
    GMM.fit(M)
    GMM_pred=GMM.predict(M)

    # Evaluate the results of GMM using adjusted rand index
    GMM_score=adjusted_rand_score(data['Y'], GMM_pred)

    GMM_Scores.append(GMM_score)

# visualization
fig, ax = plt.subplots(nrows=1,ncols=2, figsize=(12,6))
ax[0].plot(j, GMM_Scores, marker='o',linestyle='dashed', color='k')
ax[0].set_xlabel('n-Clusters (k)')
ax[0].set_ylabel('Adjusted Rand Index')
ax[0].set_title('a)')
ax[0].set_xticks(iK[::2])
ax[0].set_ylim()

YBars = [np.sum(Y==0), np.sum(Y==1)]
GMM2Bars = [np.sum(GMM2_pred==0), np.sum(GMM2_pred==1)]
offset = 0.1
width = 0.1
x = np.arange(len(['0', '1']))  # the label locations

ax[1].bar(x - offset*0.5, GMM2Bars, width, label='GMM2')
ax[1].bar(x + offset*0.5, YBars, width, label='Y')
ax[1].set_xlabel('Label')
ax[1].set_ylabel('$n$ samples')
ax[1].set_title('b)')
ax[1].set_xticks(np.arange(2))
ax[1].legend(loc='best')

fig.tight_layout()
GMM2_score
```