1ST EVALUATION

In [134]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
dataset = pd.read_csv('C:\\Users\\kamal\\OneDrive\\Documents\\Machine Learning
```

In [135]:
```
dataset
```

Out[135]:

	Creditability	Account Balance	Duration of Credit (month)	Payment Status of Previous Credit	Purpose	Credit Amount	Value Savings/Stocks	Length of current employment
0	1	1	18	4	2	1049	1	2
1	1	1	9	4	0	2799	1	3
2	1	2	12	2	9	841	2	4
3	1	1	12	4	0	2122	1	3
4	1	1	12	4	0	2171	1	3
...	...	...	...	...	...	...	...	...
995	0	1	24	2	3	1987	1	3
996	0	1	24	2	0	2303	1	5
997	0	4	21	4	0	12680	5	5
998	0	2	12	2	3	6468	5	1
999	0	1	30	2	2	6350	5	5

1000 rows × 21 columns

In [136]:
```
missing = dataset.isnull().sum()
```

In [137]:
```
missing
```

Out[137]:
```
Creditability                        0
Account Balance                      0
Duration of Credit (month)           0
Payment Status of Previous Credit    0
Purpose                              0
Credit Amount                        0
Value Savings/Stocks                 0
Length of current employment         0
Instalment per cent                  0
Sex & Marital Status                 0
Guarantors                           0
Duration in Current address          0
Most valuable available asset        0
Age (years)                          0
Concurrent Credits                   0
Type of apartment                    0
No of Credits at this Bank           0
Occupation                           0
No of dependents                     0
Telephone                            0
Foreign Worker                       0
dtype: int64
```

In [138]:
```
dataset.describe()
```

Out[138]:

	Creditability	Account Balance	Duration of Credit (month)	Payment Status of Previous Credit	Purpose	Credit Amount	V Savings/St
count	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000	1000.00000	1000.00
mean	0.700000	2.577000	20.903000	2.54500	2.828000	3271.24800	2.10
std	0.458487	1.257638	12.058814	1.08312	2.744439	2822.75176	1.58
min	0.000000	1.000000	4.000000	0.00000	0.000000	250.00000	1.00
25%	0.000000	1.000000	12.000000	2.00000	1.000000	1365.50000	1.00
50%	1.000000	2.000000	18.000000	2.00000	2.000000	2319.50000	1.00
75%	1.000000	4.000000	24.000000	4.00000	3.000000	3972.25000	3.00
max	1.000000	4.000000	72.000000	4.00000	10.000000	18424.00000	5.00

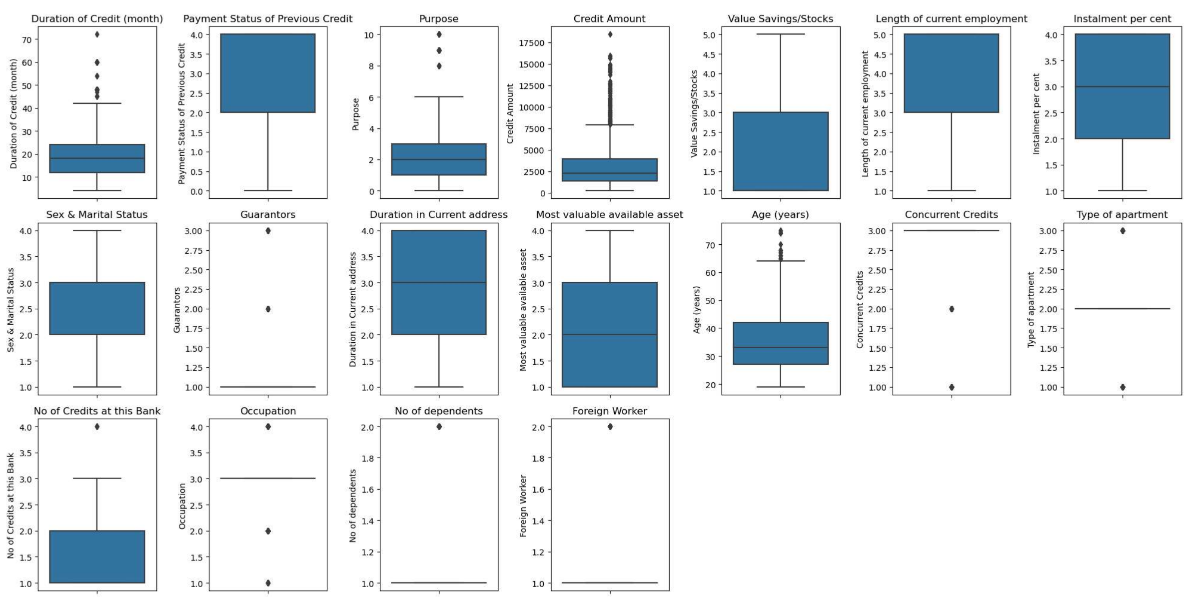
8 rows × 21 columns

In [139]:
```
print(dataset.dtypes)
```

```
Creditability                        int64
Account Balance                      int64
Duration of Credit (month)           int64
Payment Status of Previous Credit    int64
Purpose                              int64
Credit Amount                        int64
Value Savings/Stocks                 int64
Length of current employment         int64
Instalment per cent                  int64
Sex & Marital Status                 int64
Guarantors                           int64
Duration in Current address          int64
Most valuable available asset        int64
Age (years)                          int64
Concurrent Credits                   int64
Type of apartment                    int64
No of Credits at this Bank           int64
Occupation                           int64
No of dependents                     int64
Telephone                            int64
Foreign Worker                       int64
dtype: object
```

In [140]:
```
box_plot = dataset.drop(columns=['Creditability', 'Account Balance', 'Telephor
```

In [141]:
```
plt.figure(figsize=(20, 10))
for i, col in enumerate(box_plot):
    plt.subplot(3, 7, i + 1)
    sns.boxplot(y=dataset[col])
    plt.title(col)
plt.tight_layout()
plt.show()
```



In [142]:
```
attributes = ['Creditability', 'Account Balance', 'Duration of Credit (month)'
       'Length of current employment', 'Instalment per cent', 'Sex & Marital S
       'Type of apartment', 'No of Credits at this Bank', 'Occupation','No of
```

In [143]:
```
import pandas as pd
from scipy.stats import zscore

z_scores = dataset[attributes].apply(zscore)
threshold = 3

outliers_mask = abs(z_scores) > threshold
outliers = dataset[outliers_mask]

sum_of_outliers = outliers_mask.sum()

print("Sum of outliers in each column:")
print(sum_of_outliers)
```

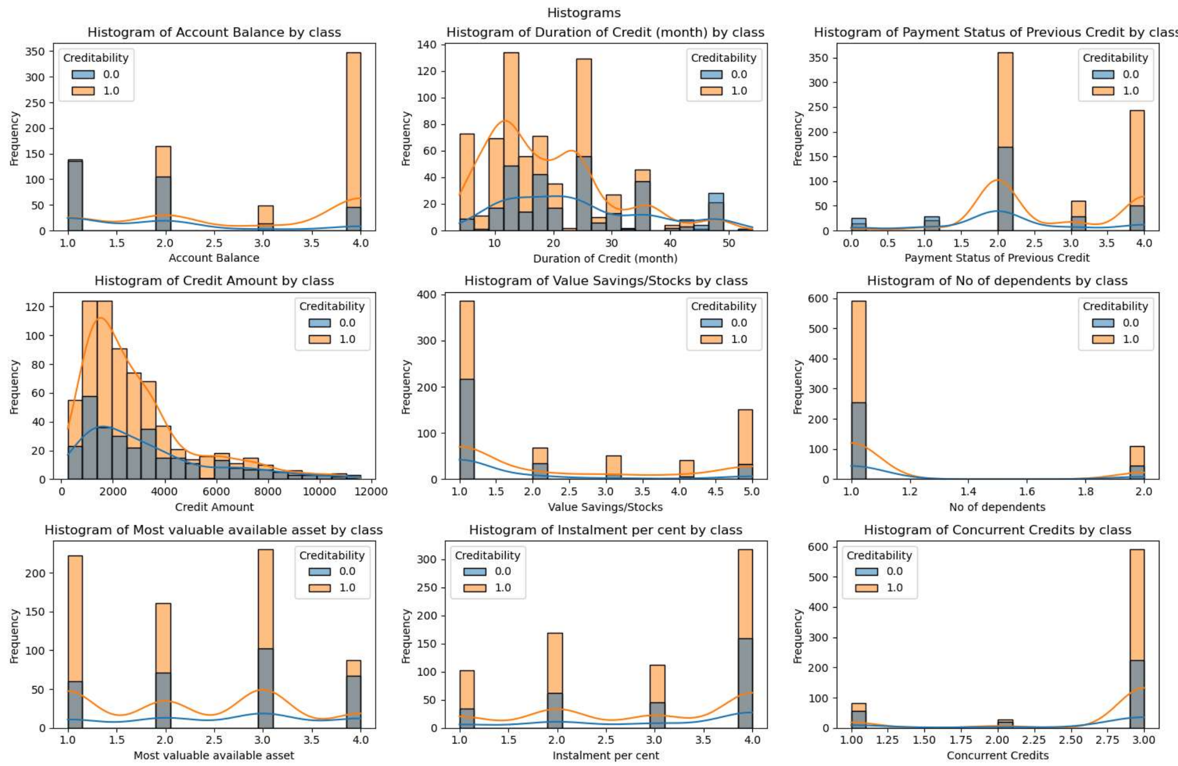
```
Sum of outliers in each column:
Creditability                          0
Account Balance                        0
Duration of Credit (month)            14
Payment Status of Previous Credit      0
Purpose                                0
Credit Amount                         25
Value Savings/Stocks                   0
Length of current employment           0
Instalment per cent                    0
Sex & Marital Status                   0
Duration in Current address            0
Most valuable available asset          0
Age (years)                            7
Concurrent Credits                     0
Type of apartment                      0
No of Credits at this Bank             6
Occupation                             0
No of dependents                       0
Telephone                              0
Foreign Worker                        37
dtype: int64
```

In [175]:
```
#outlier_columns = ['Duration of Credit (month)', 'Credit Amount', 'Age (years


#def remove_outliers(df, columns):
    #for column in columns:
        #mean = dataset[column].mean()
        #std = dataset[column].std()
        #threshold = 3
        #df = df[(df[column] - mean).abs() < threshold * std]
    #return df


#df = remove_outliers(dataset, outlier_columns)
```

In [144]:
```
for column, value in outliers.items():
    dataset.loc[dataset[column] == value, column] = dataset[column].mean()

print(dataset)
```

```
     Creditability  Account Balance  Duration of Credit (month)  \
0              1.0              1.0                        18.0
1              1.0              1.0                         9.0
2              1.0              2.0                        12.0
3              1.0              1.0                        12.0
4              1.0              1.0                        12.0
..             ...              ...                         ...
995            0.0              1.0                        24.0
996            0.0              1.0                        24.0
997            0.0              4.0                        21.0
998            0.0              2.0                        12.0
999            0.0              1.0                        30.0

     Payment Status of Previous Credit  Purpose  Credit Amount  \
0                                  4.0      2.0       1049.000
1                                  4.0      0.0       2799.000
2                                  2.0      9.0        841.000
3                                  4.0      0.0       2122.000
4                                  4.0      0.0       2171.000
```

In [145]:
```
plt.figure(figsize=(15, 10))
for i, col in enumerate(['Account Balance', 'Duration of Credit (month)', 'Pay
    plt.subplot(3, 3, i + 1)
    sns.histplot(data=dataset, x=col, hue='Creditability', kde=True, bins=20)
    plt.title(f'Histogram of {col} by class')
    plt.xlabel(col)
    plt.ylabel('Frequency')
plt.suptitle('Histograms')
plt.tight_layout()
plt.show()
```



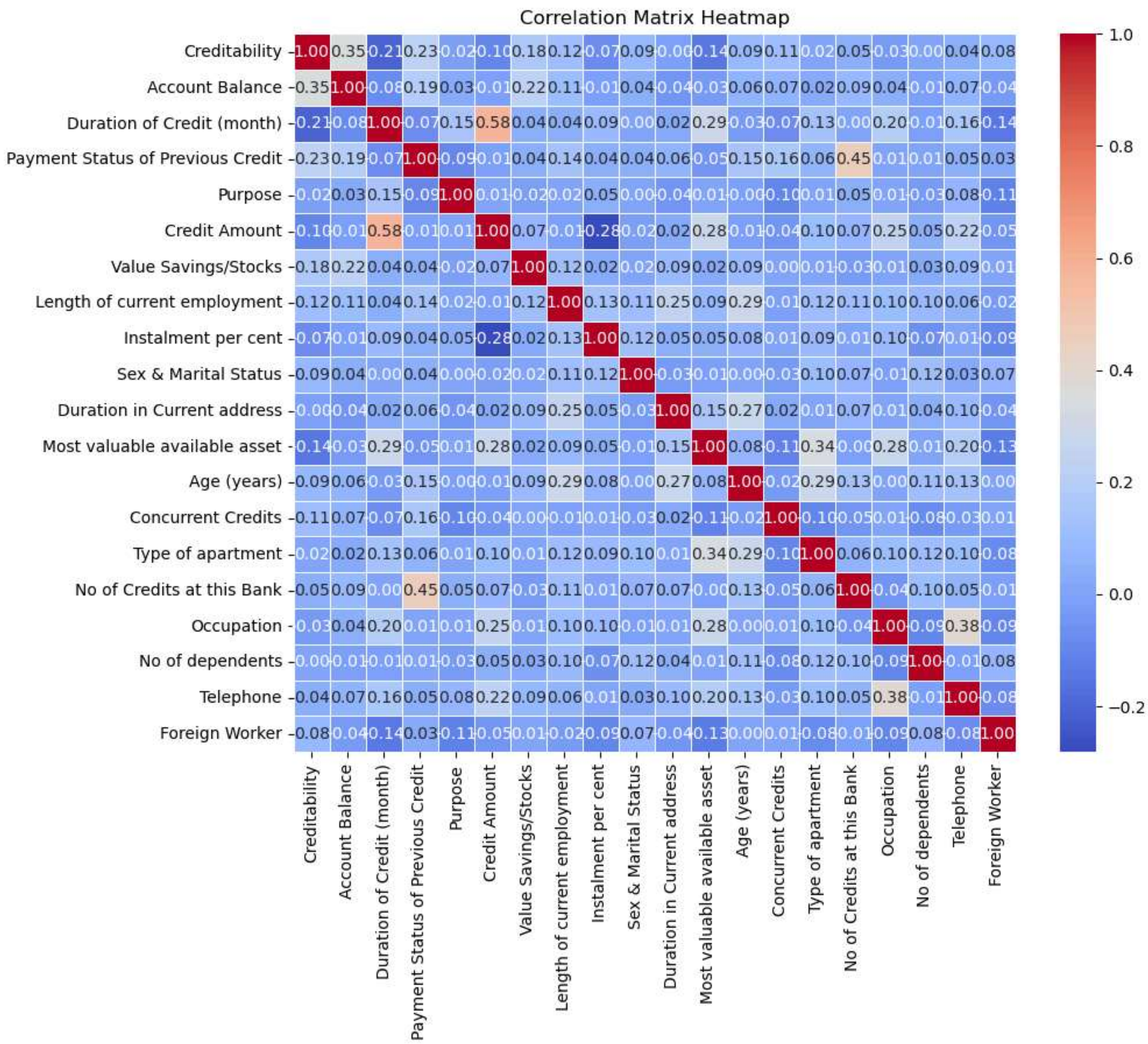
In [146]:
```
numeric_cols = dataset.select_dtypes(include=[np.number]).columns
numeric_cols
```

Out[146]:
```
Index(['Creditability', 'Account Balance', 'Duration of Credit (month)',
       'Payment Status of Previous Credit', 'Purpose', 'Credit Amount',
       'Value Savings/Stocks', 'Length of current employment',
       'Instalment per cent', 'Sex & Marital Status', 'Guarantors',
       'Duration in Current address', 'Most valuable available asset',
       'Age (years)', 'Concurrent Credits', 'Type of apartment',
       'No of Credits at this Bank', 'Occupation', 'No of dependents',
       'Telephone', 'Foreign Worker'],
      dtype='object')
```

In [147]:
```
attributes = ['Creditability', 'Account Balance', 'Duration of Credit (month)'
       'Length of current employment', 'Instalment per cent', 'Sex & Marital S
       'Type of apartment', 'No of Credits at this Bank', 'Occupation','No of

correlation_matrix = dataset[attributes].corr()

print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
                                   Creditability  Account Balance  \
Creditability                           1.000000         0.350847
Account Balance                         0.350847         1.000000
Duration of Credit (month)             -0.210833        -0.077712
Payment Status of Previous Credit       0.228785         0.192191
Purpose                                -0.017979         0.028783
Credit Amount                          -0.095433        -0.010968
Value Savings/Stocks                    0.178943         0.222867
Length of current employment            0.116002         0.106339
Instalment per cent                    -0.072404        -0.005280
Sex & Marital Status                    0.088184         0.043261
Duration in Current address            -0.002967        -0.042234
Most valuable available asset          -0.142612        -0.032260
Age (years)                             0.086792         0.064106
Concurrent Credits                      0.109844         0.068274
Type of apartment                       0.018119         0.023335
No of Credits at this Bank              0.050891         0.086676
Occupation                             -0.032735         0.040663
```

In [148]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewi
plt.title('Correlation Matrix Heatmap')
plt.show()
```



In [149]:
```
correlation_matrix = dataset.corr()
correlation_matrix
```

Out[149]:

	Creditability	Account Balance	Duration of Credit (month)	Payment Status of Previous Credit	Purpose	Credit Amount	Valu Savings/Stock
Creditability	1.000000	0.350847	-0.210833	0.228785	-0.017979	-0.095433	0.17894
Account Balance	0.350847	1.000000	-0.077712	0.192191	0.028783	-0.010968	0.22286
Duration of Credit (month)	-0.210833	-0.077712	1.000000	-0.074935	0.148448	0.578733	0.04184
Payment Status of Previous Credit	0.228785	0.192191	-0.074935	1.000000	-0.090336	-0.006512	0.03905
Purpose	-0.017979	0.028783	0.148448	-0.090336	1.000000	0.008592	-0.01868
Credit Amount	-0.095433	-0.010968	0.578733	-0.006512	0.008592	1.000000	0.07138
Value Savings/Stocks	0.178943	0.222867	0.041845	0.039058	-0.018684	0.071381	1.00000
Length of current employment	0.116002	0.106339	0.043461	0.138225	0.016013	-0.006342	0.12095
Instalment per cent	-0.072404	-0.005280	0.090636	0.044375	0.048369	-0.281638	0.02199
Sex & Marital Status	0.088184	0.043261	0.001848	0.042171	0.000157	-0.017162	0.01734
Guarantors	0.025137	-0.127737	-0.018721	-0.040676	-0.017607	-0.027517	-0.10506
Duration in Current address	-0.002967	-0.042234	0.021376	0.063198	-0.038221	0.020600	0.09142
Most valuable available asset	-0.142612	-0.032260	0.293408	-0.053777	0.010966	0.275370	0.01894
Age (years)	0.086792	0.064106	-0.027445	0.148430	-0.001416	-0.011265	0.09497
Concurrent Credits	0.109844	0.068274	-0.071741	0.159957	-0.100230	-0.038534	0.00190
Type of apartment	0.018119	0.023335	0.133234	0.061428	0.013495	0.101348	0.00664
No of Credits at this Bank	0.050891	0.086676	0.003613	0.449865	0.049828	0.073000	-0.03332
Occupation	-0.032735	0.040663	0.195234	0.010350	0.008085	0.254145	0.01170
No of dependents	0.003015	-0.014145	-0.014072	0.011550	-0.032577	0.049354	0.02751
Telephone	0.036466	0.066296	0.163410	0.052370	0.078371	0.224689	0.08720
Foreign Worker	0.082079	-0.035187	-0.136772	0.028554	-0.113244	-0.051240	0.01045

21 rows × 21 columns

2ND EVALUATION

In [165]:
```
x = dataset[['Account Balance','Duration of Credit (month)','Payment Status of
       'Value Savings/Stocks','Age (years)','Occupation','Instalment per cent'
     'Most valuable available asset','Concurrent Credits','No of Credits at th
       'No of dependents','Foreign Worker']]
y = dataset['Creditability']
```

In [166]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, rando
```

In [167]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.fit_transform(x_test)
```

Naive Bayes Model

In [168]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics
gnb = GaussianNB()
```

In [169]:
```
gnb.fit(x_train_scaled, y_train)
y_pred = gnb.predict(x_test_scaled)
nb_accuracy = accuracy_score(y_test, y_pred)
```

Decision tree classifier

In [170]:
```
from sklearn.tree import DecisionTreeClassifier
dr = DecisionTreeClassifier()
dr.fit(x_train_scaled,y_train)
predict = dr.predict(x_test_scaled)
dr_accuracy = accuracy_score(y_test,predict)
```

Logistic Regression Model

In [171]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(x_train_scaled,y_train)
predict_lr = lr.predict(x_test_scaled)
lr_accuracy = accuracy_score(y_test,predict_lr)
```

Comparing accuracy

In [172]:
```
print("Accuracy of naive bayes :", nb_accuracy)
print("Accuracy of Decision tree: ",dr_accuracy)
print("Accuracy based on logistic regression model: ",lr_accuracy)
```

```
Accuracy of naive bayes : 0.73
Accuracy of Decision tree:  0.675
Accuracy based on logistic regression model:  0.72
```

Comparing recall

In [173]:
```
from sklearn.metrics import recall_score

recall_naivebayes = recall_score(y_test, y_pred)
recall_dr = recall_score(y_test,predict)
recall_lr = recall_score(y_test,predict_lr)

print("Recall for naive bayes:", recall_naivebayes)
print("Recall for Decision Tree:", recall_dr)
print("Recall for Logistic Regression :", recall_lr)
```

```
Recall for naive bayes: 0.7867647058823529
Recall for Decision Tree: 0.8161764705882353
Recall for Logistic Regression : 0.875
```

Comparing f1 score

In [174]:
```
from sklearn.metrics import f1_score

f1_naivebayes = f1_score(y_test, y_pred)
f1_dr = f1_score(y_test,predict)
f1_lr = f1_score(y_test,predict_lr)

print("f1 score for naive bayes :", f1_naivebayes)
print("f1 score for decision tree :", f1_dr)
print("f1 score for logistic regression :", f1_lr)
```

```
f1 score for naive bayes : 0.7985074626865671
f1 score for decision tree : 0.7735191637630662
f1 score for logistic regression : 0.8095238095238096
```

COMPARING RESULTS

1) The decision tree model performs the worst among the three models as it has an accuracy of 0.71, which is the lowest. Its recall and F1 score are also lower than those of the other models. This

indicates that the decision tree model may not be the best choice for this dataset compared to logistic regression and naive Bayes.

2) The naive Bayes model might be an appropriate choice for this this dataset with an accuracy of 0.75

3) The logistic regression model outperforms both the naive Bayes and decision tree models in terms of accuracy, recall, and F1 score. It has the highest accuracy of 0.76, recall - 0.90 and f1 score - 0.84

From the above observations, a logistic regression model is the most suitable model for this dataset

In [160]:
```
import pandas as pd
filtered_data = dataset[dataset['Creditability'] == 1]
```

In [161]:
```
from sklearn.cluster import KMeans

wcss = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(filtered_data)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 11), wcss)
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.title('Elbow Method')
plt.show()
```

```
C:\Program Files\Anaconda\lib\site-packages\sklearn\cluster\_kmeans.py:87
0: FutureWarning: The default value of `n_init` will change from 10 to 'au
to' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Program Files\Anaconda\lib\site-packages\sklearn\cluster\_kmeans.py:138
2: UserWarning: KMeans is known to have a memory leak on Windows with MKL,
when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=3.
  warnings.warn(
C:\Program Files\Anaconda\lib\site-packages\sklearn\cluster\_kmeans.py:87
0: FutureWarning: The default value of `n_init` will change from 10 to 'au
to' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Program Files\Anaconda\lib\site-packages\sklearn\cluster\_kmeans.py:138
2: UserWarning: KMeans is known to have a memory leak on Windows with MKL,
when there are less chunks than available threads. You can avoid it by set
ting the environment variable OMP_NUM_THREADS=3.
  warnings.warn(
C:\Program Files\Anaconda\lib\site-packages\sklearn\cluster\_kmeans.py:87
```

In [162]:
```
k = 2
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(filtered_data)
```

```
C:\Program Files\Anaconda\lib\site-packages\sklearn\cluster\_kmeans.py:870: F
utureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\Program Files\Anaconda\lib\site-packages\sklearn\cluster\_kmeans.py:1382:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=3.
  warnings.warn(
```

Out[162]:
```
KMeans(n_clusters=2, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [163]:
```
from sklearn.metrics import silhouette_score

silhouette_avg = silhouette_score(filtered_data, kmeans.labels_)

# Print or interpret the score
print("Silhouette Score (k =", k, "):", silhouette_avg)
```

```
Silhouette Score (k = 2 ): 0.7010705450093719
```

CONCLUSIONS

Before treating outliers

Accuracy of logistic regression before treating outliers- 0.76

recall of logistic regression before treating outliers - 0.90

f1 score of logistic regression before treating outliers - 0.84

After replacing outliers with mean

Accuracy based on logistic regression model: 0.72

Recall for Logistic Regression : 0.875

f1 score for logistic regression : 0.8095238095238096

After removing outliers

Accuracy based on logistic regression model: 0.7248677248677249

Recall for Logistic Regression : 0.889763779527559

f1 score for logistic regression : 0.8129496402877697

1)The accuracy of the logistic regression model decreased slightly after replacing outliers with the mean and after removing outliers compared to before treating outliers. This suggests that treating outliers did not significantly improve the overall accuracy of the model.

However, the recall of the logistic regression model improved after both outlier treatment methods. This indicates that the model became better at correctly identifying positive cases (creditable applicants in this context) after outlier treatment.

The F1 score, which is a measure of a model's accuracy, also showed improvement after outlier treatment, indicating a better balance between precision and recall.

This suggests that the outliers in the dataset are not due to errors but rather because the dataset includes values at a higher range that are important for the prediction model.

2)K-means clustering is a method used to partition a dataset into groups (clusters) based on similarities in the data.

The Silhouette Score is a measure of how similar an object is to its own cluster compared to other clusters. A score close to 1 indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters, suggesting a good clustering. In this case, a score of 0.701 suggests that the clustering with 2 clusters is reasonably good

RECOMMENDATIONS

These are the features which will have a greater or a higher impact on the target variable

Payment Status of Previous Credit

Credit Amount

Duration of Credit (month)

Account Balance

Value Savings/Stocks

No of Credits at this Bank

No of dependents

Foreign Worker

These are the features which can be removed as they do not affect the target variable to a greater extent-

Purpose

Sex & Marital Status

Type of apartment

Telephone

In [ ]:

```

```