Team 5 Final Project - Manikandan Ramalingam, Jay Patel, Brian Johnson

Credit Default Analysis

In machine learning, the algorithms are just the tools, the raw material is the data - it's the ore that makes the gold. Thus, to build useful models, one needs to get intimate with data — it's strengths, flaws, nuances, patterns, cycles, etc. Graphical data analysis is much more than mere visualization.

Feel the Credit Default Risk Data set

There are multiple ways to analyze the data like human judgement based on the experience in the domain, utilizing various statistical and graphical analysis tools or picking features based on popular existing well defined models like Random forest classification, Principal component analysis etc. But, for all, the preliminary step would be to get the feel of the data set. The shape would provide the number of rows and columns (or features). This would enable us to use appropriate techniques for data cleansing. The below code does check the shape and type of parameters by printing the top 5 rows.

Get the Credit Default Dataset

By convention, seaborn is imported with the shorthand 'sns'. Seaborn includes a few example datasets. Let's import seaborn and load a dataset to start plotting. spellling

In [1]:

```
# Import seaborn
import seaborn as sns
import matplotlib.pyplot as plt #to allow subplot creation
import pandas as pd

# Fetch the train data into the data frame
df = pd.read_csv('/Users/manikanr/Downloads/assignment/train_data.csv')
# Apply the seaborn theme
sns.set_theme() #overwrite default Matplotlib styling parameters

shape = df.shape
print("Shape of the dataframe (row, col):",shape,"\r\n")

# Show the dataframe
df.head()
df.shape
```

```
Shape of the dataframe (row, col): (153755, 122)
```

Out[1]:

```
(153755, 122)
```

Analyze a Target variable without Feature Engineering

First, use all 121 features to analyze the target variable. Use GBC Tree classifier to predict the values and test for accuracy. Since we are not using Feature Engineering, we can select only numeric columns. So, first select all numeric columns from the data frame. Otherwise, we cannot apply the model classification with the combination of strings and numeric values. Also, drop the entire row when null values are present. Although this a feature enginnering step, without this basic data cleansing, we cannot predict the results.

In [2]:

```
import numpy as np

# Select only numeric columns
numeric_df = df.select_dtypes(include=['number'])

# Function to impute NaN with mean and floor the result
def impute_and_floor(df):
    # Select numeric columns
    numeric_cols = df.select_dtypes(include=[np.number])

    # Impute NaN with mean and floor the values
    for col in numeric_cols.columns:
        mean_value = numeric_cols[col].mean()
        df[col].fillna(mean_value, inplace=True)
        df[col] = np.floor(df[col])

    return df

# Apply the function to the DataFrame
df_cleaned = impute_and_floor(numeric_df)

df_cleaned.head()
df_cleaned.shape
```

Out[2]:

```
(153755, 106)
```

Check for Precision, Recall, F1-score and Accuracy without Feature Engineering Using GradientBoostingClassifier

This data will provide the baseline.

In [3]:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report

# Just drop the target column from 106 numeric feature columns
x_train, x_test, y_train, y_test = train_test_split(df_cleaned.drop(['TARGET'], axis='co
lumns'), df_cleaned.TARGET, test_size=0.2)

# Initialize and train the model
xgb_clf = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, r
andom_state=42)
xgb_clf.fit(x_train, y_train)

# Make predictions
y_pred = xgb_clf.predict(x_test)

# Evaluate the model
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

         0.0       0.92      1.00      0.96     28319
         1.0       0.00      0.00      0.00      2432

    accuracy                           0.92     30751
   macro avg       0.46      0.50      0.48     30751
weighted avg       0.85      0.92      0.88     30751
```

Feature Engineering Techniques

Applying Human Judgement First

There are 122 columns (or features) in this credit risk default file. Since we are predicting whether to provide loan or not based on the credit profile, this is a classification task in Machine Learning Paradigm. The loan re-payment depends on various factors like income, number of children in family, family members, type of occupation, assets, previous credits, previous credit account defaults, desperate to get loan (credit enquiries in past few months), instances of 30/60 day past due or earlier credits etc. So, before applying any statistical, graphical or Machine learning models, some important features are selected based on experience (application for earlier credits would also be considered experience) in the given domain.

The top features based on human judgement and reasons is below.

1. **AMT_INCOME_TOTAL - Income of Client**
2. **AMT_CREDIT - Loan Amount**
3. **AMT_ANNUITY - Loan Annuity**
4. **AMT_GOODS_PRICE - For Consumer loans**
5. **NAME_INCOME_TYPE - Income through family business, working salaried professional or Other)**
6. **NAME_EDUCATION_TYPE - This is important because well educated individuals tend to get more salaries over time**

   and experience.

7. **NAME_HOUSING_TYPE - Rent or Own plays a role.**
8. **NAME_FAMILY_STATUS - Married, separated and paying alimony matters.**
9. **CNT_CHILDREN - Number of children if a person has to do child support.**
10. **FLAG_OWN_CAR - Do you own a car**
11. **FLAG_OWN_REALTY - Own any Realty**
12. **DAYS_BIRTH - How many since the client is born. The more in the range (>21 < 37), the better.**
13. **DAYS_EMPLOYED - Employment days. The more years results in higher salary.**
14. **FLAG_CONT_MOBILE - Mobile phone reachable to call in case of default**
15. **CNT_FAM_MEMBERS - Number of family members**
16. **REG_REGION_NOT_LIVE_REGION - If permanent address matches with contact address**
17. **LIVE_REGION_NOT_WORK_REGION - If work address not closer to contact address**
18. **ORGANIZATION_TYPE - Type of organization where client works. This is important to judge future growth on**

    client's salary.

19. **DEF_60_CNT_SOCIAL_CIRCLE - How many observation of client's social surroundings defaulted on 60 DPD (days past due)**
20. **AMT_REQ_CREDIT_BUREAU_MON - Number of enquiries to Credit Bureau about the client one month before application**

20 Features Extraction

Extract the features in a data frame. Also, do some data cleansing with null values populated with a mean value of columns. This will make the analysis of the feature set easier.

In [4]:

```
# Extract these 21 variables
df_extract_21 = df[['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
'NAME_INCOME_TYPE',
                    'NAME_EDUCATION_TYPE', 'NAME_HOUSING_TYPE', 'NAME_FAMILY_STATUS', 'CN
T_CHILDREN', 'FLAG_OWN_CAR',
                    'FLAG_OWN_REALTY', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'FLAG_CONT_MOBILE',
'CNT_FAM_MEMBERS',
                    'REG_REGION_NOT_LIVE_REGION', 'LIVE_REGION_NOT_WORK_REGION', 'ORGANIZ
ATION_TYPE',
                    'DEF_60_CNT_SOCIAL_CIRCLE', 'AMT_REQ_CREDIT_BUREAU_MON']]
df_extract_21.head()
```

Out[4]:

	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE
0	157500.0	900000.0	26446.5	900000.0	Working	Secondary / secondary special
1	90000.0	733176.0	21438.0	612000.0	Working	Higher education
2	189000.0	1795500.0	62541.0	1795500.0	Pensioner	Secondary / secondary special
3	175500.0	494550.0	45490.5	450000.0	Pensioner	Higher education
4	270000.0	1724688.0	54283.5	1575000.0	Working	Higher education

Feature Engineering Technique1 - Mean Imputation and Normalization

For all the numeric values in 20 features set, populate the mean. Also, select the minimum value when selecting the mean as some features might not be floating point values.

In [5]:

```
import numpy as np

# Extract these 21 variables
df_extract_21 = df[['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
'NAME_INCOME_TYPE',
                    'NAME_EDUCATION_TYPE', 'NAME_HOUSING_TYPE', 'NAME_FAMILY_STATUS', 'CN
T_CHILDREN', 'FLAG_OWN_CAR',
                    'FLAG_OWN_REALTY', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'FLAG_CONT_MOBILE',
'CNT_FAM_MEMBERS',
                    'REG_REGION_NOT_LIVE_REGION', 'LIVE_REGION_NOT_WORK_REGION', 'ORGANIZ
ATION_TYPE',
                    'DEF_60_CNT_SOCIAL_CIRCLE', 'AMT_REQ_CREDIT_BUREAU_MON']]

# Replace NaN values in specific columns with mean
columns_to_fill = ['AMT_ANNUITY', 'AMT_GOODS_PRICE', 'CNT_FAM_MEMBERS', 'DEF_60_CNT_SOCI
AL_CIRCLE', 'AMT_REQ_CREDIT_BUREAU_MON']

# Calculate the mean of specific columns and round down to the nearest integer
mean_values = df_extract_21[columns_to_fill].mean().apply(np.floor)

# Fill NaN values in df_extract_21 with the calculated mean values
df_extract_21[columns_to_fill] = df_extract_21[columns_to_fill].fillna(mean_values)
df_extract_21.head()
```

```
<ipython-input-5-fe77968494d2>:17: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_21[columns_to_fill] = df_extract_21[columns_to_fill].fillna(mean_values)
```

Out[5]:

	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE
0	157500.0	900000.0	26446.5	900000.0	Working	Secondary / secondary special
1	90000.0	733176.0	21438.0	612000.0	Working	Higher education
2	189000.0	1795500.0	62541.0	1795500.0	Pensioner	Secondary / secondary special
3	175500.0	494550.0	45490.5	450000.0	Pensioner	Higher education
4	270000.0	1724688.0	54283.5	1575000.0	Working	Higher education

Feature Engineering Technique 2 - Encoding Categorical Variables

Convert the categorical variables to numeric values using encoding tenchnique. Also, convert few columns selected with negative values to positive values.

In [103]:

```
from sklearn.preprocessing import LabelEncoder

columns_to_encode = ['NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_HOUSING_TYPE', 'NAM
E_FAMILY_STATUS',
                     'ORGANIZATION_TYPE', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'FLAG_CONT_
MOBILE']

# Encode categorical columns
label_encoder = LabelEncoder()
for col in columns_to_encode:
    df_extract_21[col] = label_encoder.fit_transform(df_extract_21[col])
df_extract_21.head()
```

```
<ipython-input-103-bc6de7c0550c>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_21[col] = label_encoder.fit_transform(df_extract_21[col])
<ipython-input-103-bc6de7c0550c>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_21[col] = label_encoder.fit_transform(df_extract_21[col])
<ipython-input-103-bc6de7c0550c>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_21[col] = label_encoder.fit_transform(df_extract_21[col])
<ipython-input-103-bc6de7c0550c>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_21[col] = label_encoder.fit_transform(df_extract_21[col])
<ipython-input-103-bc6de7c0550c>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_21[col] = label_encoder.fit_transform(df_extract_21[col])
<ipython-input-103-bc6de7c0550c>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_21[col] = label_encoder.fit_transform(df_extract_21[col])
<ipython-input-103-bc6de7c0550c>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_21[col] = label_encoder.fit_transform(df_extract_21[col])
<ipython-input-103-bc6de7c0550c>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_21[col] = label_encoder.fit_transform(df_extract_21[col])
```

Out[103]:

	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE
0	157500.0	900000.0	26446.5	900000.0	7	4
1	90000.0	733176.0	21438.0	612000.0	7	1
2	189000.0	1795500.0	62541.0	1795500.0	3	4
3	175500.0	494550.0	45490.5	450000.0	3	1
4	270000.0	1724688.0	54283.5	1575000.0	7	1

Feature Engineering Technique 3 - Data Transformation

Note that Days birth and days employed are negative values. It is transformed to positive values for getting good prediction.

In [7]:

```
# Convert negative to positive values
columns_to_convert_positive = ['DAYS_BIRTH', 'DAYS_EMPLOYED']
for col in columns_to_convert_positive:
    df_extract_21[col] = df_extract_21[col].abs()

df_extract_21.head()
```

```
<ipython-input-7-8bdef085ec8d>:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_21[col] = df_extract_21[col].abs()
```

Out[7]:

	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE
0	157500.0	900000.0	26446.5	900000.0	7	4
1	90000.0	733176.0	21438.0	612000.0	7	1
2	189000.0	1795500.0	62541.0	1795500.0	3	4
3	175500.0	494550.0	45490.5	450000.0	3	1
4	270000.0	1724688.0	54283.5	1575000.0	7	1

Feature Engineering Technique 4 - Dimensionality Reduction to Extract 10 Most Important Features from 21

Extract the 10 most important features in a data frame out of 21. There are many techniques that can be used for this.

1. **Use Random Forest classifier and select top 10.**
2. **Use Prinicipal Component Analysis.**
3. **SelectKBest an univariate method to select K=10 best features.**

SelectKBest Classification with Chi2

In [8]:

```
from sklearn.datasets import make_classification
from sklearn.feature_selection import SelectKBest, chi2

# Generate a sample regression dataset
#X, y = chi2(n_samples=df_extract_21.shape[0], n_features=df_extract_21.shape[1], random_
state=42)
# Perform feature selection using chi-squared test
selector = SelectKBest(score_func=chi2, k=10)  # Select top 10 features
y= df[['TARGET']]
X_new = selector.fit_transform(df_extract_21, y)
# Print the selected features
selected_features = df_extract_21.columns[selector.get_support()]
print("\nSelected features from SelectKBest with chi2 classification:\n")
print(selected_features)
```

```
Selected features from SelectKBest with chi2 classification:

Index(['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
       'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'DAYS_BIRTH',
       'DAYS_EMPLOYED', 'ORGANIZATION_TYPE', 'DEF_60_CNT_SOCIAL_CIRCLE'],
      dtype='object')
```

Random Forest Classifier to identify top features

In [9]:

```
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier

# Train Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
y= df[['TARGET']]
rf_model.fit(df_extract_21, y)

# Get feature importances
importances = rf_model.feature_importances_

# Get indices of top 10 features
top10_indices = np.argsort(importances)[::-1][:10]
print("\nSelected Features from Random Forest classification:\n")
for i, idx in enumerate(top10_indices):
    print(df_extract_21.columns[idx])
```

```
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/base.py:1152: DataConversio
nWarning: A column-vector y was passed when a 1d array was expected. Please change the sh
ape of y to (n_samples,), for example using ravel().
  return fit_method(estimator, *args, **kwargs)
```

```
Selected Features from Random Forest classification:

DAYS_BIRTH
AMT_ANNUITY
DAYS_EMPLOYED
AMT_CREDIT
AMT_INCOME_TOTAL
AMT_GOODS_PRICE
ORGANIZATION_TYPE
NAME_FAMILY_STATUS
CNT_FAM_MEMBERS
CNT_CHILDREN
```

PCA Analysis (unsupervised) to identify top 10 features

In [10]:

```
from sklearn.decomposition import PCA
```

```
from sklearn.decomposition import PCA

# Initialize PCA with desired number of components (e.g., 10 for selecting top 10 compone
nts)
n_components = 10
pca = PCA(n_components=n_components)

# Fit PCA on the data and transform it
X_pca = pca.fit_transform(df_extract_21)

# Optionally, you can also access the principal components (eigenvectors)
principal_components = pca.components_

# Get the indices of the top 10 principal components with the largest explained variance
top10_indices = np.argsort(pca.explained_variance_ratio_)[::-1][:10]

# Get the names of the top 10 features corresponding to the top principal components
top10_features = []
for idx in top10_indices:
    component = principal_components[idx]
    relevant_features = df_extract_21.columns[np.abs(component) > 0.1]
    top10_features.extend(relevant_features)

# Remove duplicates (if any)
top10_features = list(set(top10_features))

# Print the names of the top 10 features
print("Top 10 features:")
print(top10_features)
```

```
Top 10 features:
['NAME_EDUCATION_TYPE', 'AMT_INCOME_TOTAL', 'CNT_CHILDREN', 'ORGANIZATION_TYPE', 'AMT_GOO
DS_PRICE', 'DAYS_EMPLOYED', 'NAME_INCOME_TYPE', 'DAYS_BIRTH', 'NAME_FAMILY_STATUS', 'NAME
_HOUSING_TYPE', 'AMT_ANNUITY', 'CNT_FAM_MEMBERS', 'AMT_CREDIT']
```

Conclusion on Top 10 features from above analysis

Top 10 features based on mode from above 3 supervised/unsupervised analysis results:-

'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE' 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED' 'ORGANIZATION_TYPE', 'CNT_FAM_MEMBERS'

In [61]:

```
df_extract_10 = df_extract_21[['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOOD
S_PRICE', 'NAME_INCOME_TYPE',
                  'NAME_EDUCATION_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'ORGANIZATION_T
YPE',
                  'CNT_FAM_MEMBERS']]
df_extract_10.head()
#df_extract_10.shape
```

Out[61]:

	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE
0	157500.0	900000.0	26446.5	900000.0	7	4
1	90000.0	733176.0	21438.0	612000.0	7	1
2	189000.0	1795500.0	62541.0	1795500.0	3	4
3	175500.0	494550.0	45490.5	450000.0	3	1
4	270000.0	1724688.0	54283.5	1575000.0	7	1

Prediction After Applying Feature Engineering Technique

Used RandomForestClassifier on the top 10 features after Feature Engineering techniques

Used RandomForestClassifier on the top 10 features after Feature Engineering techniques.

In [12]:

```
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Train Random Forest model
# Split the data
x_train, x_test, y_train, y_test = train_test_split(df_extract_10, df['TARGET'], test_si
ze=0.2, random_state=42)

# Initialize the classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_classifier.fit(x_train, y_train)

# Make predictions
y_pred = rf_classifier.predict(x_test)

# Evaluate the model
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     28294
           1       0.21      0.00      0.00      2457

    accuracy                           0.92     30751
   macro avg       0.57      0.50      0.48     30751
weighted avg       0.86      0.92      0.88     30751
```

Conclusion

From the analysis above with human judgement, different models and graphical analysis it seems the top 10 features out of 122 features in the train_set.csv seems to be 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED' 'ORGANIZATION_TYPE', 'CNT_FAM_MEMBERS'. This will enable the credit decision when applied with different machine learning models and hyper parameter tuning. On classification report analysis, the report is similar to the one we have at the top with 106 features. But, it took long time to train those compared to reduced dimensions of 10 values. So, it shouldn't be interpreted that we can use entire 106 features. It boils down to just using top 10 features perform with accuracy of 92%. This shows the importance of Feature Engineering.

Disclosure This is based on only 122 features and not the data in other csv files. There might be appropriate data in other files which might be more relevant. This exercise is focused on train_test.csv file.

In [62]:

```
# The Cleansed data frame with 10 features after applying Feature Engineering is shown be
low.
# Further model fits will be done using this
df_extract_10 = df_extract_10.join(df[['TARGET']])
df_extract_10.head()
```

Out[62]:

	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE
0	157500.0	900000.0	26446.5	900000.0	7	4
1	90000.0	733176.0	21438.0	612000.0	7	1
2	189000.0	1795500.0	62541.0	1795500.0	3	4
3	175500.0	494550.0	45490.5	450000.0	3	1

	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE
4	270000.0	1724688.0	54283.5	1575000.0	7	1

Decision Tree Classifier

In [118]:

```
#DecisionTree w/ no creditscore range limit w/ metric scores

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, f1_score, roc_auc_score, precision_sco
re, recall_score, accuracy_score

# Example data loading (adjust the path and file name as needed)
df_extract_21 = pd.read_csv('/Users/manikanr/Downloads/assignment/train_data.csv')

# Extract relevant features
df_extract_10 = df_extract_21[['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOOD
S_PRICE', 'NAME_INCOME_TYPE',
                               'NAME_EDUCATION_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'OR
GANIZATION_TYPE',
                               'CNT_FAM_MEMBERS']]

# Ensure the target column is present
if 'TARGET' not in df_extract_21.columns:
    raise KeyError("The target column 'TARGET' is not found in the dataset.")

# Handle missing values
numerical_columns = ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
'DAYS_BIRTH', 'DAYS_EMPLOYED', 'CNT_FAM_MEMBERS']
categorical_columns = ['NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'ORGANIZATION_TYPE']

df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_10[
numerical_columns].mean())
for column in categorical_columns:
    df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0]
)

# Encode categorical features
label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    df_extract_10[column] = le.fit_transform(df_extract_10[column])
    label_encoders[column] = le

# Define features and target
features = df_extract_10
target = df_extract_21['TARGET']

# Split the data
x_train, x_test, y_train, y_test = train_test_split(features, target, test_size=0.2, ran
dom_state=42)

# Train Decision Tree model with adjusted parameters
decision_tree_classifier = DecisionTreeClassifier(random_state=42, min_samples_leaf=50,
max_depth=10)
decision_tree_classifier.fit(x_train, y_train)

# Function to calculate credit worthiness score with a finer granularity
def get_credit_worthiness_score(prob):
    return int(prob * 100)  # Scaling probability to a score between 0 and 100

# Make predictions on new data and get the probabilities
new_data = pd.DataFrame({
    'AMT_INCOME_TOTAL': [200000],
    'AMT_CREDIT': [500000],
```

```
    'AMT_ANNUITY': [25000],
    'AMT_GOODS_PRICE': [450000],
    'NAME_INCOME_TYPE': ['Working'],
    'NAME_EDUCATION_TYPE': ['Higher education'],
    'DAYS_BIRTH': [-10000],
    'DAYS_EMPLOYED': [-2000],
    'ORGANIZATION_TYPE': ['Business Entity Type 3'],
    'CNT_FAM_MEMBERS': [2]
})

# Ensure new data columns match the training data
new_data = new_data[features.columns]

# Encode new data using the same label encoders
for column in categorical_columns:
    new_data[column] = label_encoders[column].transform(new_data[column])

# Get the probability of the positive class
prob = decision_tree_classifier.predict_proba(new_data)[:, 1][0]

# Generate the credit worthiness score
score = get_credit_worthiness_score(prob)
print(f'Credit Worthiness Score: {score}')

# Predict on the test set
y_pred = decision_tree_classifier.predict(x_test)
y_pred_proba = decision_tree_classifier.predict_proba(x_test)[:, 1]

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred_proba)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
print(f'AUC Score: {auc}')

# Detailed classification report
report = classification_report(y_test, y_pred)
print(report)

# Predict probabilities for the entire dataset
probabilities = decision_tree_classifier.predict_proba(features)[:, 1]

# Generate credit worthiness scores
scores = [get_credit_worthiness_score(prob) for prob in probabilities]

# Add the scores to the DataFrame
df_extract_10['Credit Worthiness Score'] = scores

# Summarize the scores
summary = df_extract_10['Credit Worthiness Score'].describe()
print(summary)

# Optionally, print the distribution of scores
score_distribution = df_extract_10['Credit Worthiness Score'].value_counts().sort_index(
)
print(score_distribution)
```

```
<ipython-input-118-ad48fd5f8cc4>:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_1
0[numerical_columns].mean())
<ipython-input-118-ad48fd5f8cc4>:27: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0])
<ipython-input-118-ad48fd5f8cc4>:33: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-118-ad48fd5f8cc4>:33: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-118-ad48fd5f8cc4>:33: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
```

```
Credit Worthiness Score: 14
Accuracy: 0.9201001593444116
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
AUC Score: 0.6066785697096011
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     28294
           1       0.00      0.00      0.00      2457

    accuracy                           0.92     30751
   macro avg       0.46      0.50      0.48     30751
weighted avg       0.85      0.92      0.88     30751

count    153755.000000
mean          7.591512
std           5.949095
min           0.000000
25%           4.000000
50%           6.000000
75%          11.000000
max          45.000000
Name: Credit Worthiness Score, dtype: float64
Credit Worthiness Score
0     13268
1      6110
2      4886
3     13645
4      8785
5     25467
6      9921
7     14135
8      5615
9      5103
10     6159
11     6221
12     6737
13     3137
14     3737
15     3657
16     3188
17     3835
18     2655
19     2530
```

```
19      2530
20       850
21       149
22       202
23       390
24      1414
25       467
27        81
28       196
30       585
33        79
34       115
35       137
36       115
40        74
45       110
Name: count, dtype: int64
```

```
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
<ipython-input-118-ad48fd5f8cc4>:107: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10['Credit Worthiness Score'] = scores
```

In [120]:

```
#updated code to try and make score in range of 0-20
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, f1_score, roc_auc_score, precision_sco
re, recall_score, accuracy_score

# Example data loading (adjust the path and file name as needed)
df_extract_21 = pd.read_csv('/Users/manikanr/Downloads/assignment/train_data.csv')

# Extract relevant features
df_extract_10 = df_extract_21[['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOOD
S_PRICE', 'NAME_INCOME_TYPE',
                               'NAME_EDUCATION_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'OR
GANIZATION_TYPE',
                               'CNT_FAM_MEMBERS']]

# Ensure the target column is present
if 'TARGET' not in df_extract_21.columns:
    raise KeyError("The target column 'TARGET' is not found in the dataset.")

# Handle missing values
numerical_columns = ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
'DAYS_BIRTH', 'DAYS_EMPLOYED', 'CNT_FAM_MEMBERS']
```

```
categorical_columns = ['NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'ORGANIZATION_TYPE']

df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_10[
numerical_columns].mean())
for column in categorical_columns:
    df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0]
)

# Encode categorical features
label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    df_extract_10[column] = le.fit_transform(df_extract_10[column])
    label_encoders[column] = le

# Define features and target
features = df_extract_10
target = df_extract_21['TARGET']

# Split the data
x_train, x_test, y_train, y_test = train_test_split(features, target, test_size=0.2, ran
dom_state=42)

# Train Decision Tree model with adjusted parameters
decision_tree_classifier = DecisionTreeClassifier(random_state=42, min_samples_leaf=50,
max_depth=10)
decision_tree_classifier.fit(x_train, y_train)

# Function to calculate credit worthiness score with a range of 0 to 20
def get_credit_worthiness_score(prob):
    return int(prob * 20)  # Scaling probability to a score between 0 and 20

# Make predictions on new data and get the probabilities
new_data = pd.DataFrame({
    'AMT_INCOME_TOTAL': [200000],
    'AMT_CREDIT': [500000],
    'AMT_ANNUITY': [25000],
    'AMT_GOODS_PRICE': [450000],
    'NAME_INCOME_TYPE': ['Working'],
    'NAME_EDUCATION_TYPE': ['Higher education'],
    'DAYS_BIRTH': [-10000],
    'DAYS_EMPLOYED': [-2000],
    'ORGANIZATION_TYPE': ['Business Entity Type 3'],
    'CNT_FAM_MEMBERS': [2]
})

# Ensure new data columns match the training data
new_data = new_data[features.columns]

# Encode new data using the same label encoders
for column in categorical_columns:
    new_data[column] = label_encoders[column].transform(new_data[column])

# Get the probability of the positive class
prob = decision_tree_classifier.predict_proba(new_data)[:, 1][0]

# Generate the credit worthiness score
score = get_credit_worthiness_score(prob)
print(f'Credit Worthiness Score: {score}')

# Predict on the test set
y_pred = decision_tree_classifier.predict(x_test)
y_pred_proba = decision_tree_classifier.predict_proba(x_test)[:, 1]

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred_proba)

print(f'Accuracy: {accuracy}')
```

```python
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
print(f'AUC Score: {auc}')

# Detailed classification report
report = classification_report(y_test, y_pred)
print(report)

# Predict probabilities for the entire dataset
probabilities = decision_tree_classifier.predict_proba(features)[:, 1]

# Generate credit worthiness scores
scores = [get_credit_worthiness_score(prob) for prob in probabilities]

# Add the scores to the DataFrame
df_extract_10['Credit Worthiness Score'] = scores

# Summarize the scores
summary = df_extract_10['Credit Worthiness Score'].describe()
print(summary)

# Optionally, print the distribution of scores
score_distribution = df_extract_10['Credit Worthiness Score'].value_counts().sort_index(
)
print(score_distribution)
```

```
<ipython-input-120-3e5245198902>:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_1
0[numerical_columns].mean())
<ipython-input-120-3e5245198902>:26: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0])
<ipython-input-120-3e5245198902>:32: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-120-3e5245198902>:32: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-120-3e5245198902>:32: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
```

```
Credit Worthiness Score: 2
Accuracy: 0.9201001593444116
Precision: 0.0
Recall: 0.0
F1 Score: 0.0
AUC Score: 0.6066785697096011
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     28294
```

```
           0       0.92      1.00      0.96     28294
           1       0.00      0.00      0.00      2457

    accuracy                           0.92     30751
   macro avg       0.46      0.50      0.48     30751
weighted avg       0.85      0.92      0.88     30751

count    153755.000000
mean          1.193964
std           1.164563
min           0.000000
25%           0.000000
50%           1.000000
75%           2.000000
max           9.000000
Name: Credit Worthiness Score, dtype: float64
Credit Worthiness Score
0    46694
1    60241
2    25991
3    15865
4     3005
5      744
6      779
7      252
8       74
9      110
Name: count, dtype: int64
```

```
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
<ipython-input-120-3e5245198902>:106: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10['Credit Worthiness Score'] = scores
```

Linear Regression Model

In [121]:

```python
#Linear Regression Model
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Example data loading (adjust the path and file name as needed)
df_extract_21 = pd.read_csv('/Users/manikanr/Downloads/assignment/train_data.csv')

# Extract relevant features
```

```
df_extract_10 = df_extract_21[['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOOD
S_PRICE', 'NAME_INCOME_TYPE',
                               'NAME_EDUCATION_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'OR
GANIZATION_TYPE',
                               'CNT_FAM_MEMBERS']]

# Ensure the target column is present
if 'TARGET' not in df_extract_21.columns:
    raise KeyError("The target column 'TARGET' is not found in the dataset.")

# Handle missing values
numerical_columns = ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
'DAYS_BIRTH', 'DAYS_EMPLOYED', 'CNT_FAM_MEMBERS']
categorical_columns = ['NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'ORGANIZATION_TYPE']

df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_10[
numerical_columns].mean())
for column in categorical_columns:
    df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0]
)

# Encode categorical features
label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    df_extract_10[column] = le.fit_transform(df_extract_10[column])
    label_encoders[column] = le

# Define features and target
features = df_extract_10
target = df_extract_21['TARGET']

# Standardize numerical features
scaler = StandardScaler()
features[numerical_columns] = scaler.fit_transform(features[numerical_columns])

# Split the data
x_train, x_test, y_train, y_test = train_test_split(features, target, test_size=0.2, ran
dom_state=42)

# Train Linear Regression model
linear_regression_model = LinearRegression()
linear_regression_model.fit(x_train, y_train)

# Function to calculate credit worthiness score with a range of 0 to 20
def get_credit_worthiness_score(value):
    return int(min(max(value * 20, 0), 20))  # Scale to 0-20 and clip values to be withi
n the range

# Make predictions on new data
new_data = pd.DataFrame({
    'AMT_INCOME_TOTAL': [200000],
    'AMT_CREDIT': [500000],
    'AMT_ANNUITY': [25000],
    'AMT_GOODS_PRICE': [450000],
    'NAME_INCOME_TYPE': ['Working'],
    'NAME_EDUCATION_TYPE': ['Higher education'],
    'DAYS_BIRTH': [-10000],
    'DAYS_EMPLOYED': [-2000],
    'ORGANIZATION_TYPE': ['Business Entity Type 3'],
    'CNT_FAM_MEMBERS': [2]
})

# Ensure new data columns match the training data
new_data = new_data[features.columns]

# Encode and scale new data using the same label encoders and scaler
for column in categorical_columns:
    new_data[column] = label_encoders[column].transform(new_data[column])
new_data[numerical_columns] = scaler.transform(new_data[numerical_columns])

# Get the predicted value for the new data
```

```python
predicted_value = linear_regression_model.predict(new_data)[0]

# Generate the credit worthiness score
score = get_credit_worthiness_score(predicted_value)
print(f'Credit Worthiness Score: {score}')

# Predict values for the entire dataset
predicted_values = linear_regression_model.predict(features)

# Generate credit worthiness scores
scores = [get_credit_worthiness_score(value) for value in predicted_values]

# Add the scores to the DataFrame
df_extract_10['Credit Worthiness Score'] = scores

# Summarize the scores
summary = df_extract_10['Credit Worthiness Score'].describe()
print(summary)

# Optionally, print the distribution of scores
score_distribution = df_extract_10['Credit Worthiness Score'].value_counts().sort_index(
)
print(score_distribution)
```

```
<ipython-input-121-3c7401192e7d>:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_1
0[numerical_columns].mean())
<ipython-input-121-3c7401192e7d>:26: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0])
<ipython-input-121-3c7401192e7d>:32: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-121-3c7401192e7d>:32: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-121-3c7401192e7d>:32: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-121-3c7401192e7d>:41: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  features[numerical_columns] = scaler.fit_transform(features[numerical_columns])
```

```
Credit Worthiness Score: 1
count    153755.000000
mean          1.138304
std           0.703117
min           0.000000
```

```
min           0.000000
25%           1.000000
50%           1.000000
75%           2.000000
max          16.000000
Name: Credit Worthiness Score, dtype: float64
Credit Worthiness Score
0     26867
1     80638
2     44384
3      1863
4         1
5         1
16        1
Name: count, dtype: int64
```

```
<ipython-input-121-3c7401192e7d>:90: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10['Credit Worthiness Score'] = scores
```

In [122]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score, roc_
auc_score, classification_report

# Example data loading (adjust the path and file name as needed)
df_extract_21 = pd.read_csv('/Users/manikanr/Downloads/assignment/train_data.csv')

# Extract relevant features
df_extract_10 = df_extract_21[['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOOD
S_PRICE', 'NAME_INCOME_TYPE',
                               'NAME_EDUCATION_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'OR
GANIZATION_TYPE',
                               'CNT_FAM_MEMBERS']]

# Ensure the target column is present
if 'TARGET' not in df_extract_21.columns:
    raise KeyError("The target column 'TARGET' is not found in the dataset.")

# Handle missing values
numerical_columns = ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
'DAYS_BIRTH', 'DAYS_EMPLOYED', 'CNT_FAM_MEMBERS']
categorical_columns = ['NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'ORGANIZATION_TYPE']

df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_10[
numerical_columns].mean())
for column in categorical_columns:
    df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0]
)

# Encode categorical features
label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    df_extract_10[column] = le.fit_transform(df_extract_10[column])
    label_encoders[column] = le

# Define features and target
features = df_extract_10
target = df_extract_21['TARGET']

# Standardize numerical features
scaler = StandardScaler()
features[numerical_columns] = scaler.fit_transform(features[numerical_columns])
```

```
# Split the data
x_train, x_test, y_train, y_test = train_test_split(features, target, test_size=0.2, ran
dom_state=42)

# Train Linear Regression model
linear_regression_model = LinearRegression()
linear_regression_model.fit(x_train, y_train)

# Function to calculate credit worthiness score with a range of 0 to 20
def get_credit_worthiness_score(value, min_value, max_value):
    # Scale the value to a range of 0 to 20
    scaled_value = 20 * (value - min_value) / (max_value - min_value)
    return int(min(max(scaled_value, 0), 20))

# Make predictions on the test set
y_test_pred = linear_regression_model.predict(x_test)

# Convert regression output to binary classification using a threshold (e.g., 0.5)
threshold = 0.5
y_test_pred_class = (y_test_pred >= threshold).astype(int)

# Calculate additional metrics
f1 = f1_score(y_test, y_test_pred_class)
roc_auc = roc_auc_score(y_test, y_test_pred)
precision = precision_score(y_test, y_test_pred_class)
recall = recall_score(y_test, y_test_pred_class)
accuracy = accuracy_score(y_test, y_test_pred_class)

print(f'F1 Score: {f1}')
print(f'ROC AUC Score: {roc_auc}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'Accuracy: {accuracy}')
print(classification_report(y_test, y_test_pred_class))

# Make predictions on new data
new_data = pd.DataFrame({
    'AMT_INCOME_TOTAL': [200000],
    'AMT_CREDIT': [500000],
    'AMT_ANNUITY': [25000],
    'AMT_GOODS_PRICE': [450000],
    'NAME_INCOME_TYPE': ['Working'],
    'NAME_EDUCATION_TYPE': ['Higher education'],
    'DAYS_BIRTH': [-10000],
    'DAYS_EMPLOYED': [-2000],
    'ORGANIZATION_TYPE': ['Business Entity Type 3'],
    'CNT_FAM_MEMBERS': [2]
})

# Ensure new data columns match the training data
new_data = new_data[features.columns]

# Encode and scale new data using the same label encoders and scaler
for column in categorical_columns:
    new_data[column] = label_encoders[column].transform(new_data[column])
new_data[numerical_columns] = scaler.transform(new_data[numerical_columns])

# Get the predicted value for the new data
predicted_value = linear_regression_model.predict(new_data)[0]

# Get the minimum and maximum predicted values from the training set to use for scaling
train_predictions = linear_regression_model.predict(x_train)
min_train_pred = train_predictions.min()
max_train_pred = train_predictions.max()

# Generate the credit worthiness score
score = get_credit_worthiness_score(predicted_value, min_train_pred, max_train_pred)
print(f'Credit Worthiness Score: {score}')

# Predict values for the entire dataset
predicted_values = linear_regression_model.predict(features)
```

```python
# Generate credit worthiness scores
scores = [get_credit_worthiness_score(value, min_train_pred, max_train_pred) for value i
n predicted_values]

# Add the scores to the DataFrame
df_extract_10['Credit Worthiness Score'] = scores

# Summarize the scores
summary = df_extract_10['Credit Worthiness Score'].describe()
print(summary)

# Optionally, print the distribution of scores
score_distribution = df_extract_10['Credit Worthiness Score'].value_counts().sort_index(
)
print(score_distribution)
```

```
<ipython-input-122-9ca30e7f81b8>:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_1
0[numerical_columns].mean())
<ipython-input-122-9ca30e7f81b8>:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0])
<ipython-input-122-9ca30e7f81b8>:31: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-122-9ca30e7f81b8>:31: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-122-9ca30e7f81b8>:31: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-122-9ca30e7f81b8>:40: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  features[numerical_columns] = scaler.fit_transform(features[numerical_columns])
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this behavi
```

```
or.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
F1 Score: 0.0
ROC AUC Score: 0.6231901794918688
Precision: 0.0
Recall: 0.0
Accuracy: 0.9201001593444116
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     28294
           1       0.00      0.00      0.00      2457

    accuracy                           0.92     30751
   macro avg       0.46      0.50      0.48     30751
weighted avg       0.85      0.92      0.88     30751

Credit Worthiness Score: 4
count    153755.000000
mean          3.975617
std           0.754009
min           0.000000
25%           4.000000
50%           4.000000
75%           4.000000
max          20.000000
Name: Credit Worthiness Score, dtype: float64
Credit Worthiness Score
0          9
1        106
2       3207
3      34244
4      79854
5      35423
6        910
8          1
20         1
Name: count, dtype: int64
```

```
<ipython-input-122-9ca30e7f81b8>:117: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10['Credit Worthiness Score'] = scores
```

Logistic Regression Model fit

In [123]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score, roc_
auc_score, classification_report

# Example data loading (adjust the path and file name as needed)
df_extract_21 = pd.read_csv('/Users/manikanr/Downloads/assignment/train_data.csv')

# Extract relevant features
df_extract_10 = df_extract_21[['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOOD
S_PRICE', 'NAME_INCOME_TYPE',
                               'NAME_EDUCATION_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'OR
GANIZATION_TYPE',
```

```
                                'CNT_FAM_MEMBERS']]

# Ensure the target column is present
if 'TARGET' not in df_extract_21.columns:
    raise KeyError("The target column 'TARGET' is not found in the dataset.")

# Handle missing values
numerical_columns = ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
'DAYS_BIRTH', 'DAYS_EMPLOYED', 'CNT_FAM_MEMBERS']
categorical_columns = ['NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'ORGANIZATION_TYPE']

df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_10[
numerical_columns].mean())
for column in categorical_columns:
    df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0]
)

# Encode categorical features
label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    df_extract_10[column] = le.fit_transform(df_extract_10[column])
    label_encoders[column] = le

# Define features and target
features = df_extract_10
target = df_extract_21['TARGET']

# Standardize numerical features
scaler = StandardScaler()
features[numerical_columns] = scaler.fit_transform(features[numerical_columns])

# Split the data
x_train, x_test, y_train, y_test = train_test_split(features, target, test_size=0.2, ran
dom_state=42)

# Train Logistic Regression model
logistic_regression_model = LogisticRegression(random_state=42)
logistic_regression_model.fit(x_train, y_train)

# Make predictions on the test set
y_test_pred_prob = logistic_regression_model.predict_proba(x_test)[:, 1]
y_test_pred = logistic_regression_model.predict(x_test)

# Calculate additional metrics
f1 = f1_score(y_test, y_test_pred)
roc_auc = roc_auc_score(y_test, y_test_pred_prob)
precision = precision_score(y_test, y_test_pred)
recall = recall_score(y_test, y_test_pred)
accuracy = accuracy_score(y_test, y_test_pred)

print(f'F1 Score: {f1}')
print(f'ROC AUC Score: {roc_auc}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'Accuracy: {accuracy}')
print(classification_report(y_test, y_test_pred))

# Function to calculate credit worthiness score with a range of 0 to 20
def get_credit_worthiness_score(prob):
    return int(prob * 20)  # Scaling probability to a score between 0 and 20

# Make predictions on new data
new_data = pd.DataFrame({
    'AMT_INCOME_TOTAL': [200000],
    'AMT_CREDIT': [500000],
    'AMT_ANNUITY': [25000],
    'AMT_GOODS_PRICE': [450000],
    'NAME_INCOME_TYPE': ['Working'],
    'NAME_EDUCATION_TYPE': ['Higher education'],
    'DAYS_BIRTH': [-10000],
    'DAYS_EMPLOYED': [-2000],
```

```
    'ORGANIZATION_TYPE': ['Business Entity Type 3'],
    'CNT_FAM_MEMBERS': [2]
})

# Ensure new data columns match the training data
new_data = new_data[features.columns]

# Encode and scale new data using the same label encoders and scaler
for column in categorical_columns:
    new_data[column] = label_encoders[column].transform(new_data[column])
new_data[numerical_columns] = scaler.transform(new_data[numerical_columns])

# Get the probability of the positive class for the new data
prob = logistic_regression_model.predict_proba(new_data)[:, 1][0]

# Generate the credit worthiness score
score = get_credit_worthiness_score(prob)
print(f'Credit Worthiness Score: {score}')

# Predict probabilities for the entire dataset
probabilities = logistic_regression_model.predict_proba(features)[:, 1]

# Generate credit worthiness scores
scores = [get_credit_worthiness_score(prob) for prob in probabilities]

# Add the scores to the DataFrame
df_extract_10['Credit Worthiness Score'] = scores

# Summarize the scores
summary = df_extract_10['Credit Worthiness Score'].describe()
print(summary)

# Optionally, print the distribution of scores
score_distribution = df_extract_10['Credit Worthiness Score'].value_counts().sort_index(
)
print(score_distribution)
```

```
<ipython-input-123-f82a35283643>:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_1
0[numerical_columns].mean())
<ipython-input-123-f82a35283643>:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0])
<ipython-input-123-f82a35283643>:31: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-123-f82a35283643>:31: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-123-f82a35283643>:31: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
```

```
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-123-f82a35283643>:40: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  features[numerical_columns] = scaler.fit_transform(features[numerical_columns])
```

```
F1 Score: 0.0
ROC AUC Score: 0.6243525631028283
Precision: 0.0
Recall: 0.0
Accuracy: 0.9201001593444116
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     28294
           1       0.00      0.00      0.00      2457

    accuracy                           0.92     30751
   macro avg       0.46      0.50      0.48     30751
weighted avg       0.85      0.92      0.88     30751

Credit Worthiness Score: 1
count    153755.000000
mean          1.128776
std           0.758295
min           0.000000
25%           1.000000
50%           1.000000
75%           2.000000
max          17.000000
Name: Credit Worthiness Score, dtype: float64
Credit Worthiness Score
0     28171
1     84527
2     34518
3      6191
4       338
5         8
10        1
17        1
Name: count, dtype: int64
```

```
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logistic.py:4
60: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/manikanr/anaconda3/lib/python3.8/site-packages/sklearn/metrics/_classification.py:
1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0
in labels with no predicted samples. Use `zero_division` parameter to control this behavi
or.
  _warn_prf(average, modifier, msg_start, len(result))
<ipython-input-123-f82a35283643>:107: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10['Credit Worthiness Score'] = scores
```

In [124]:

```
#Updated Code with Class Weights and Adjusted Threshold

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score, roc_
auc_score, classification_report

df_extract_21 = pd.read_csv('/Users/manikanr/Downloads/assignment/train_data.csv')

# Extract relevant features
df_extract_10 = df_extract_21[['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOOD
S_PRICE', 'NAME_INCOME_TYPE',
                               'NAME_EDUCATION_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'OR
GANIZATION_TYPE',
                               'CNT_FAM_MEMBERS']]

# Ensure the target column is present
if 'TARGET' not in df_extract_21.columns:
    raise KeyError("The target column 'TARGET' is not found in the dataset.")

# Handle missing values
numerical_columns = ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
'DAYS_BIRTH', 'DAYS_EMPLOYED', 'CNT_FAM_MEMBERS']
categorical_columns = ['NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'ORGANIZATION_TYPE']

df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_10[
numerical_columns].mean())
for column in categorical_columns:
    df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0]
)

# Encode categorical features
label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    df_extract_10[column] = le.fit_transform(df_extract_10[column])
    label_encoders[column] = le

# Define features and target
features = df_extract_10
target = df_extract_21['TARGET']

# Standardize numerical features
scaler = StandardScaler()
features[numerical_columns] = scaler.fit_transform(features[numerical_columns])

# Split the data
x_train, x_test, y_train, y_test = train_test_split(features, target, test_size=0.2, ran
dom_state=42)

# Train Logistic Regression model with class weights
logistic_regression_model = LogisticRegression(random_state=42, class_weight='balanced')
logistic_regression_model.fit(x_train, y_train)

# Make predictions on the test set with adjusted threshold
y_test_pred_prob = logistic_regression_model.predict_proba(x_test)[:, 1]
threshold = 0.3  # Adjusted threshold
y_test_pred = (y_test_pred_prob >= threshold).astype(int)

# Calculate additional metrics
```

```
f1 = f1_score(y_test, y_test_pred)
roc_auc = roc_auc_score(y_test, y_test_pred_prob)
precision = precision_score(y_test, y_test_pred)
recall = recall_score(y_test, y_test_pred)
accuracy = accuracy_score(y_test, y_test_pred)

print(f'F1 Score: {f1}')
print(f'ROC AUC Score: {roc_auc}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'Accuracy: {accuracy}')
print(classification_report(y_test, y_test_pred))

# Function to calculate credit worthiness score with a range of 0 to 20
def get_credit_worthiness_score(prob):
    return int(prob * 20)  # Scaling probability to a score between 0 and 20

# Make predictions on new data
new_data = pd.DataFrame({
    'AMT_INCOME_TOTAL': [200000],
    'AMT_CREDIT': [500000],
    'AMT_ANNUITY': [25000],
    'AMT_GOODS_PRICE': [450000],
    'NAME_INCOME_TYPE': ['Working'],
    'NAME_EDUCATION_TYPE': ['Higher education'],
    'DAYS_BIRTH': [-10000],
    'DAYS_EMPLOYED': [-2000],
    'ORGANIZATION_TYPE': ['Business Entity Type 3'],
    'CNT_FAM_MEMBERS': [2]
})

# Ensure new data columns match the training data
new_data = new_data[features.columns]

# Encode and scale new data using the same label encoders and scaler
for column in categorical_columns:
    new_data[column] = label_encoders[column].transform(new_data[column])
new_data[numerical_columns] = scaler.transform(new_data[numerical_columns])

# Get the probability of the positive class for the new data
prob = logistic_regression_model.predict_proba(new_data)[:, 1][0]

# Generate the credit worthiness score
score = get_credit_worthiness_score(prob)
print(f'Credit Worthiness Score: {score}')

# Predict probabilities for the entire dataset
probabilities = logistic_regression_model.predict_proba(features)[:, 1]

# Generate credit worthiness scores
scores = [get_credit_worthiness_score(prob) for prob in probabilities]

# Add the scores to the DataFrame
df_extract_10['Credit Worthiness Score'] = scores

# Summarize the scores
summary = df_extract_10['Credit Worthiness Score'].describe()
print(summary)

# Optionally, print the distribution of scores
score_distribution = df_extract_10['Credit Worthiness Score'].value_counts().sort_index(
)
print(score_distribution)
```

```
<ipython-input-124-4b249e2c7746>:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_1
0[numerical_columns].mean())
```

```
<ipython-input-124-4b249e2c7746>:26: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0])
<ipython-input-124-4b249e2c7746>:32: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-124-4b249e2c7746>:32: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-124-4b249e2c7746>:32: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-124-4b249e2c7746>:41: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  features[numerical_columns] = scaler.fit_transform(features[numerical_columns])
```

```
F1 Score: 0.15301038284740748
ROC AUC Score: 0.6240602374411662
Precision: 0.08306575457210666
Recall: 0.9686609686609686
Accuracy: 0.14314981626613768
              precision    recall  f1-score   support

           0       0.96      0.07      0.13     28294
           1       0.08      0.97      0.15      2457

    accuracy                           0.14     30751
   macro avg       0.52      0.52      0.14     30751
weighted avg       0.89      0.14      0.13     30751

Credit Worthiness Score: 10
count    153755.000000
mean          9.057488
std           2.272846
min           0.000000
25%           8.000000
50%           9.000000
75%          11.000000
max          18.000000
Name: Credit Worthiness Score, dtype: float64
Credit Worthiness Score
0         5
1        18
2       163
3       764
4      2744
5      6423
6      9762
7     18495
8     24851
9     23796
10    23130
11    20049
```

```
11    20049
12    14236
13     7420
14     1729
15      163
16        5
18        2
Name: count, dtype: int64
```

```
<ipython-input-124-4b249e2c7746>:109: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10['Credit Worthiness Score'] = scores
```

Random Forest Classifier Model Fit

In [125]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.preprocessing import LabelEncoder
```

In [126]:

```
df_extract_21 = pd.read_csv('/Users/manikanr/Downloads/assignment/train_data.csv')

# Extract relevant features
df_extract_10 = df_extract_21[['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOOD
S_PRICE', 'NAME_INCOME_TYPE',
                               'NAME_EDUCATION_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'OR
GANIZATION_TYPE',
                               'CNT_FAM_MEMBERS']]

# Ensure the target column is present
if 'TARGET' not in df_extract_21.columns:
    raise KeyError("The target column 'TARGET' is not found in the dataset.")

# Handle missing values
numerical_columns = ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
'DAYS_BIRTH', 'DAYS_EMPLOYED', 'CNT_FAM_MEMBERS']
categorical_columns = ['NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'ORGANIZATION_TYPE']

df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_10[
numerical_columns].mean())
for column in categorical_columns:
    df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0]
)

# Encode categorical features
label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    df_extract_10[column] = le.fit_transform(df_extract_10[column])
    label_encoders[column] = le

# Define features and target
features = df_extract_10
target = df_extract_21['TARGET']
```

```
<ipython-input-126-64fb78cd4935>:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_1
```

```
0[numerical_columns].mean())
<ipython-input-126-64fb78cd4935>:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0])
<ipython-input-126-64fb78cd4935>:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-126-64fb78cd4935>:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-126-64fb78cd4935>:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
```

In [128]:

```python
x_train, x_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(x_train, y_train)

y_pred = rf_classifier.predict(x_test)
report = classification_report(y_test, y_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     28294
           1       0.15      0.00      0.00      2457

    accuracy                           0.92     30751
   macro avg       0.54      0.50      0.48     30751
weighted avg       0.86      0.92      0.88     30751
```

Ran the Random Forest again and checked for like results, with this in mind I moved forward with spreading the data out to see where the distribution of applicants fell out on a score from one to ten. The first iteration noted that 73 percent of them were at a score of four or lower, indicating to me that the output scoring data was unbalanced. Considering this, I increased the scoring system to 20 to get more granularity. This resulted in a cut line very low, probably in the 3 to 5 range. Those results are below. This will be compared to the XGBoost results to see how much of an idea we can get as to where our comfortability with lending decision will be.

In [129]:

```python
def get_credit_worthiness_score(prob):
    return int(prob * 16 + 4)

# Make predictions on new data and get the probabilities
new_data = pd.DataFrame({
    'AMT_INCOME_TOTAL': [200000],
    'AMT_CREDIT': [500000],
    'AMT_ANNUITY': [25000],
    'AMT_GOODS_PRICE': [450000],
```

```
    'NAME_INCOME_TYPE': ['Working'],
    'NAME_EDUCATION_TYPE': ['Higher education'],
    'DAYS_BIRTH': [-10000],
    'DAYS_EMPLOYED': [-2000],
    'ORGANIZATION_TYPE': ['Business Entity Type 3'],
    'CNT_FAM_MEMBERS': [2]
})

# Encode new data using the same label encoders
for column in categorical_columns:
    new_data[column] = label_encoders[column].transform(new_data[column])

# Get the probability of the positive class
prob = rf_classifier.predict_proba(new_data)[:, 1][0]

# Generate the credit worthiness score
score = get_credit_worthiness_score(prob)
print(f'Credit Worthiness Score: {score}')
```

```
Credit Worthiness Score: 7
```

In [130]:

```
probabilities = rf_classifier.predict_proba(features)[:, 1]

# Generate credit worthiness scores
scores = [get_credit_worthiness_score(prob) for prob in probabilities]

# Add the scores to the DataFrame
df_extract_10['Credit Worthiness Score'] = scores

# Summarize the scores
summary = df_extract_10['Credit Worthiness Score'].describe()
print(summary)

# Optionally, print the distribution of scores
score_distribution = df_extract_10['Credit Worthiness Score'].value_counts().sort_index(
)
print(score_distribution)
```

```
count    153755.000000
mean          4.964736
std           2.578755
min           4.000000
25%           4.000000
50%           4.000000
75%           5.000000
max          18.000000
Name: Credit Worthiness Score, dtype: float64
Credit Worthiness Score
4     112601
5      22300
6       5625
7       2011
8        857
9        265
10        86
11        47
12       215
13      1502
14      3903
15      3304
16       970
17        66
18         3
Name: count, dtype: int64
```

```
<ipython-input-130-41a2719daed5>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
```

```
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10['Credit Worthiness Score'] = scores
```

The data and features were the same. This resulted in a tighter spread of scores between 4 and 16, but were much more evenly distributed, indicating output scoring data was more balanced. I believe with the XGBoost data brought to a deciding body, we could more easily identify a cut score, likely between 5 and 6.

XGBoost Classifier Model Fit

In [133]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBClassifier
from sklearn.metrics import classification_report

# Example data loading (adjust the path and file name as needed)
df_extract_21 = pd.read_csv('/Users/manikanr/Downloads/assignment/train_data.csv')

# Extract relevant features
df_extract_10 = df_extract_21[['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOOD
S_PRICE', 'NAME_INCOME_TYPE',
                               'NAME_EDUCATION_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'OR
GANIZATION_TYPE',
                               'CNT_FAM_MEMBERS']]

# Ensure the target column is present
if 'TARGET' not in df_extract_21.columns:
    raise KeyError("The target column 'TARGET' is not found in the dataset.")

# Handle missing values
numerical_columns = ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
'DAYS_BIRTH', 'DAYS_EMPLOYED', 'CNT_FAM_MEMBERS']
categorical_columns = ['NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'ORGANIZATION_TYPE']

df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_10[
numerical_columns].mean())
for column in categorical_columns:
    df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0]
)

# Encode categorical features
label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    df_extract_10[column] = le.fit_transform(df_extract_10[column])
    label_encoders[column] = le

# Define features and target
features = df_extract_10
target = df_extract_21['TARGET']

# Split the data
x_train, x_test, y_train, y_test = train_test_split(features, target, test_size=0.2, ran
dom_state=42)

# Train XGBoost model
xgb_classifier = XGBClassifier(n_estimators=100, random_state=42)
xgb_classifier.fit(x_train, y_train)

# Function to calculate credit worthiness score
def get_credit_worthiness_score(prob):
    return int(prob * 16 + 4)

# Make predictions on new data and get the probabilities
new_data = pd.DataFrame({
    'AMT_INCOME_TOTAL': [200000],
    'AMT_CREDIT': [500000],
```

```
    'AMT_ANNUITY': [25000],
    'AMT_GOODS_PRICE': [450000],
    'NAME_INCOME_TYPE': ['Working'],
    'NAME_EDUCATION_TYPE': ['Higher education'],
    'DAYS_BIRTH': [-10000],
    'DAYS_EMPLOYED': [-2000],
    'ORGANIZATION_TYPE': ['Business Entity Type 3'],
    'CNT_FAM_MEMBERS': [2]
})

# Ensure new data columns match the training data
new_data = new_data[features.columns]

# Encode new data using the same label encoders
for column in categorical_columns:
    new_data[column] = label_encoders[column].transform(new_data[column])

# Get the probability of the positive class
prob = xgb_classifier.predict_proba(new_data)[:, 1][0]

# Generate the credit worthiness score
score = get_credit_worthiness_score(prob)
print(f'Credit Worthiness Score: {score}')

# Predict probabilities for the entire dataset
probabilities = xgb_classifier.predict_proba(features)[:, 1]

# Generate credit worthiness scores
scores = [get_credit_worthiness_score(prob) for prob in probabilities]

# Add the scores to the DataFrame
df_extract_10['Credit Worthiness Score'] = scores

# Summarize the scores
summary = df_extract_10['Credit Worthiness Score'].describe()
print(summary)

# Optionally, print the distribution of scores
score_distribution = df_extract_10['Credit Worthiness Score'].value_counts().sort_index(
)
print(score_distribution)
```

```
<ipython-input-133-9482c24a22a4>:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_1
0[numerical_columns].mean())
<ipython-input-133-9482c24a22a4>:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0])
<ipython-input-133-9482c24a22a4>:31: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-133-9482c24a22a4>:31: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-133-9482c24a22a4>:31: SettingWithCopyWarning:
```

```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
```

```
Credit Worthiness Score: 5
count    153755.000000
mean          4.788443
std           1.046105
min           4.000000
25%           4.000000
50%           5.000000
75%           5.000000
max          16.000000
Name: Credit Worthiness Score, dtype: float64
Credit Worthiness Score
4     75715
5     51546
6     16445
7      6060
8      2455
9       946
10      321
11      130
12       63
13       39
14       18
15       14
16        3
Name: count, dtype: int64
```

```
<ipython-input-133-9482c24a22a4>:84: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10['Credit Worthiness Score'] = scores
```

In [134]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, accuracy_score, roc_auc_score, precision_score


df_extract_21 = pd.read_csv('/Users/manikanr/Downloads/assignment/train_data.csv')

# Extract relevant features
df_extract_10 = df_extract_21[['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_INCOME_TYPE',
                               'NAME_EDUCATION_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'ORGANIZATION_TYPE',
                               'CNT_FAM_MEMBERS']]

# Ensure the target column is present
if 'TARGET' not in df_extract_21.columns:
    raise KeyError("The target column 'TARGET' is not found in the dataset.")

# Handle missing values
numerical_columns = ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'CNT_FAM_MEMBERS']
categorical_columns = ['NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'ORGANIZATION_TYPE']

df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_10[numerical_columns].mean())
for column in categorical_columns:
```

```
    df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0]
)

# Encode categorical features
label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    df_extract_10[column] = le.fit_transform(df_extract_10[column])
    label_encoders[column] = le

# Define features and target
features = df_extract_10
target = df_extract_21['TARGET']

# Split the data
x_train, x_test, y_train, y_test = train_test_split(features, target, test_size=0.2, ran
dom_state=42)

# Train XGBoost model
xgb_classifier = XGBClassifier(n_estimators=100, random_state=42)
xgb_classifier.fit(x_train, y_train)

# Make predictions
y_pred_xgb = xgb_classifier.predict(x_test)

# Calculate precision score
xgb_precision = precision_score(y_test, y_pred_xgb)

print(f"XGBoost Precision Score: {xgb_precision}")

# Additional evaluation metrics
xgb_accuracy = accuracy_score(y_test, y_pred_xgb)
xgb_auc_roc = roc_auc_score(y_test, y_pred_xgb)
xgb_report = classification_report(y_test, y_pred_xgb)

print("\nXGBoost Evaluation:")
print(f"Accuracy: {xgb_accuracy}")
print(f"AUC-ROC: {xgb_auc_roc}")
print("Classification Report:\n", xgb_report)
```

```
<ipython-input-134-3e8b13797b43>:23: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[numerical_columns] = df_extract_10[numerical_columns].fillna(df_extract_1
0[numerical_columns].mean())
<ipython-input-134-3e8b13797b43>:25: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = df_extract_10[column].fillna(df_extract_10[column].mode()[0])
<ipython-input-134-3e8b13797b43>:31: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-134-3e8b13797b43>:31: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
<ipython-input-134-3e8b13797b43>:31: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  df_extract_10[column] = le.fit_transform(df_extract_10[column])
```

```
XGBoost Precision Score: 0.5

XGBoost Evaluation:
Accuracy: 0.9201001593444116
AUC-ROC: 0.5007433144494006
Classification Report:
              precision    recall  f1-score   support

           0       0.92      1.00      0.96     28294
           1       0.50      0.00      0.00      2457

    accuracy                           0.92     30751
   macro avg       0.71      0.50      0.48     30751
weighted avg       0.89      0.92      0.88     30751
```

Random Forest Accuracy: 92% XGBoost Accuracy: 92% Both had an f1-score of .96, indicating both are good models as we are applying them.

Correct the data imbalance

The test target values with 1 is just ~12k entries while with 0 is ~140,000. Correct this data imbalance with over sampling the minority class with SMOTE method

In [72]:

```
# SMOTE method
from imblearn.over_sampling import SMOTE

# Step 2: Load your dataset
X = df_extract_10.drop('TARGET', axis=1).values
y = df_extract_10['TARGET'].values

# Step 3: Apply SMOTE to generate synthetic samples for the minority class
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
count_minority_class = np.sum(y_resampled == 0)
print(f'Count of y_resampled with values of 0: {count_minority_class}')
```

```
Count of y_resampled with values of 0: 141343
```

Model fit with Convolutional Neural Networks

Convolutional Nural Networks are good for classification tasks. Follow the below steps to do model fit. Develop an aggregate score with a scale (1-20) and set a threshold for a YES/NO decision point.

1. **Do the scaling and split the train-test using scikit-learn.**
2. **Build the model using CNN.**
3. **Train the model.**
4. **Model Evaluation.**
5. **Getting Decision Threshold.**

Scaling and Split the train-test data

Do the scaling and split the train-test using scikit-learn with python code below.

In [87]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

```
from tensorflow.keras.utils import to_categorical

# Scale features
scaler = StandardScaler()
features_scaled = scaler.fit_transform(X_resampled)

# Reshape features for CNN
features_scaled_reshaped = features_scaled.reshape(features_scaled.shape[0], features_sc
aled.shape[1], 1)

# Split the data
x_train, x_test, y_train, y_test = train_test_split(features_scaled_reshaped, y_resample
d, test_size=0.2, random_state=42)
```

Building Model using CNN

Build the model and do the fit with Tensorflow-keras libraries for CNN. Use Relu as activation layer, use Adam optimizer and use mean squared error.

In [89]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv1D, Flatten, Dropout, MaxPooling1D

# Reshape the data to be compatible with Conv1D
#X_train_reshaped = X_train_scaled.reshape((X_train_scaled.shape[0], X_train_scaled.shape
[1], 1))
#X_test_reshaped = X_test_scaled.reshape((X_test_scaled.shape[0], X_test_scaled.shape[1],
1))

# Define the CNN model
model = Sequential()
model.add(Conv1D(filters=64, kernel_size=2, activation='relu', input_shape=(features_sca
led_reshaped.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Model Training

Now Train the model with fit method using python code below.

In [91]:

```
# Train the model
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_split=0.2)
```

```
Epoch 1/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4742 - accuracy: 0.758
4 - val_loss: 0.4619 - val_accuracy: 0.7648
Epoch 2/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4712 - accuracy: 0.760
1 - val_loss: 0.4591 - val_accuracy: 0.7671
Epoch 3/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4697 - accuracy: 0.760
8 - val_loss: 0.4621 - val_accuracy: 0.7632
Epoch 4/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4678 - accuracy: 0.762
6 - val_loss: 0.4666 - val_accuracy: 0.7578
Epoch 5/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4663 - accuracy: 0.762
5 - val_loss: 0.4619 - val_accuracy: 0.7630
Epoch 6/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4644 - accuracy: 0.76
```

```
46 - val_loss: 0.4519 - val_accuracy: 0.7710
Epoch 7/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4637 - accuracy: 0.76
43 - val_loss: 0.4531 - val_accuracy: 0.7695
Epoch 8/50
5654/5654 [==============================] - 11s 2ms/step - loss: 0.4621 - accuracy: 0.76
52 - val_loss: 0.4608 - val_accuracy: 0.7632
Epoch 9/50
5654/5654 [==============================] - 8s 2ms/step - loss: 0.4593 - accuracy: 0.766
0 - val_loss: 0.4532 - val_accuracy: 0.7671
Epoch 10/50
5654/5654 [==============================] - 7s 1ms/step - loss: 0.4587 - accuracy: 0.767
0 - val_loss: 0.4576 - val_accuracy: 0.7637
Epoch 11/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4570 - accuracy: 0.767
4 - val_loss: 0.4431 - val_accuracy: 0.7748
Epoch 12/50
5654/5654 [==============================] - 11s 2ms/step - loss: 0.4568 - accuracy: 0.76
74 - val_loss: 0.4437 - val_accuracy: 0.7735
Epoch 13/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4545 - accuracy: 0.76
84 - val_loss: 0.4423 - val_accuracy: 0.7746
Epoch 14/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4536 - accuracy: 0.76
90 - val_loss: 0.4429 - val_accuracy: 0.7755
Epoch 15/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4534 - accuracy: 0.769
7 - val_loss: 0.4426 - val_accuracy: 0.7733
Epoch 16/50
5654/5654 [==============================] - 11s 2ms/step - loss: 0.4517 - accuracy: 0.77
04 - val_loss: 0.4431 - val_accuracy: 0.7744
Epoch 17/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4512 - accuracy: 0.77
06 - val_loss: 0.4510 - val_accuracy: 0.7668
Epoch 18/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4491 - accuracy: 0.77
20 - val_loss: 0.4378 - val_accuracy: 0.7776
Epoch 19/50
5654/5654 [==============================] - 11s 2ms/step - loss: 0.4484 - accuracy: 0.77
27 - val_loss: 0.4375 - val_accuracy: 0.7781
Epoch 20/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4482 - accuracy: 0.77
27 - val_loss: 0.4420 - val_accuracy: 0.7711
Epoch 21/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4483 - accuracy: 0.77
33 - val_loss: 0.4410 - val_accuracy: 0.7764
Epoch 22/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4472 - accuracy: 0.773
2 - val_loss: 0.4378 - val_accuracy: 0.7788
Epoch 23/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4459 - accuracy: 0.77
46 - val_loss: 0.4369 - val_accuracy: 0.7794
Epoch 24/50
5654/5654 [==============================] - 11s 2ms/step - loss: 0.4459 - accuracy: 0.77
44 - val_loss: 0.4367 - val_accuracy: 0.7790
Epoch 25/50
5654/5654 [==============================] - 11s 2ms/step - loss: 0.4445 - accuracy: 0.77
52 - val_loss: 0.4294 - val_accuracy: 0.7842
Epoch 26/50
5654/5654 [==============================] - 11s 2ms/step - loss: 0.4438 - accuracy: 0.77
54 - val_loss: 0.4348 - val_accuracy: 0.7783
Epoch 27/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4440 - accuracy: 0.77
66 - val_loss: 0.4332 - val_accuracy: 0.7805
Epoch 28/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4418 - accuracy: 0.77
68 - val_loss: 0.4426 - val_accuracy: 0.7756
Epoch 29/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4406 - accuracy: 0.777
1 - val_loss: 0.4288 - val_accuracy: 0.7834
Epoch 30/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4406 - accuracy: 0.777
```

```
5 - val_loss: 0.4308 - val_accuracy: 0.7833
Epoch 31/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4406 - accuracy: 0.77
74 - val_loss: 0.4374 - val_accuracy: 0.7781
Epoch 32/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4398 - accuracy: 0.777
7 - val_loss: 0.4340 - val_accuracy: 0.7787
Epoch 33/50
5654/5654 [==============================] - 8s 1ms/step - loss: 0.4392 - accuracy: 0.778
2 - val_loss: 0.4354 - val_accuracy: 0.7778
Epoch 34/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4390 - accuracy: 0.778
5 - val_loss: 0.4285 - val_accuracy: 0.7805
Epoch 35/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4393 - accuracy: 0.77
83 - val_loss: 0.4264 - val_accuracy: 0.7865
Epoch 36/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4383 - accuracy: 0.779
4 - val_loss: 0.4269 - val_accuracy: 0.7862
Epoch 37/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4386 - accuracy: 0.778
7 - val_loss: 0.4261 - val_accuracy: 0.7869
Epoch 38/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4368 - accuracy: 0.780
7 - val_loss: 0.4226 - val_accuracy: 0.7889
Epoch 39/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4377 - accuracy: 0.77
93 - val_loss: 0.4241 - val_accuracy: 0.7877
Epoch 40/50
5654/5654 [==============================] - 11s 2ms/step - loss: 0.4364 - accuracy: 0.78
02 - val_loss: 0.4274 - val_accuracy: 0.7831
Epoch 41/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4365 - accuracy: 0.780
6 - val_loss: 0.4291 - val_accuracy: 0.7843
Epoch 42/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4369 - accuracy: 0.78
05 - val_loss: 0.4220 - val_accuracy: 0.7889
Epoch 43/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4350 - accuracy: 0.78
15 - val_loss: 0.4240 - val_accuracy: 0.7884
Epoch 44/50
5654/5654 [==============================] - 9s 2ms/step - loss: 0.4347 - accuracy: 0.781
6 - val_loss: 0.4216 - val_accuracy: 0.7878
Epoch 45/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4356 - accuracy: 0.78
08 - val_loss: 0.4270 - val_accuracy: 0.7882
Epoch 46/50
5654/5654 [==============================] - 11s 2ms/step - loss: 0.4343 - accuracy: 0.78
16 - val_loss: 0.4190 - val_accuracy: 0.7907
Epoch 47/50
5654/5654 [==============================] - 11s 2ms/step - loss: 0.4342 - accuracy: 0.78
27 - val_loss: 0.4258 - val_accuracy: 0.7866
Epoch 48/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4336 - accuracy: 0.78
29 - val_loss: 0.4283 - val_accuracy: 0.7850
Epoch 49/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4336 - accuracy: 0.78
22 - val_loss: 0.4194 - val_accuracy: 0.7897
Epoch 50/50
5654/5654 [==============================] - 10s 2ms/step - loss: 0.4323 - accuracy: 0.78
26 - val_loss: 0.4189 - val_accuracy: 0.7924
```

We get ~80% accuracy for our model for 50 epochs. We can see the model is getting better with an increase in number of epochs. With more epochs, it would take more time to train the model. Care should be taken so that model doesn't overfit as well.

Model Evaluation

Evaluate the model and predict the values for test set.

In [112]:

```
# Predict on the test set
y_pred_proba = model.predict(x_test).flatten()
y_pred = (y_pred_proba > 0.5).astype(int)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
auc = roc_auc_score(y_test, y_pred_proba)

print(f'Accuracy: {accuracy}')
print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'F1 Score: {f1}')
print(f'AUC Score: {auc}')

# Detailed classification report
report = classification_report(y_test, y_pred)
print(report)
```

```
1767/1767 [==============================] - 1s 676us/step
Accuracy: 0.792387420849694
Precision: 0.8740814210360218
Recall: 0.6842904034165108
F1 Score: 0.7676287761808608
AUC Score: 0.8756238699735143
              precision    recall  f1-score   support

           0       0.74      0.90      0.81     28205
           1       0.87      0.68      0.77     28333

    accuracy                           0.79     56538
   macro avg       0.81      0.79      0.79     56538
weighted avg       0.81      0.79      0.79     56538
```

Decision threshold

To set a threshold for a YES/NO decision, we'll classify the predictions based on the threshold value (e.g., 6.0).

In [116]:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, conf
usion_matrix, roc_auc_score

# Detailed classification report
report = classification_report(y_test, y_pred)
print(report)

# Predict probabilities for the entire dataset
probabilities = model.predict(features_scaled_reshaped).flatten()

# Generate credit worthiness scores
scores = [get_credit_worthiness_score(prob) for prob in probabilities]

X_resampled_df = pd.DataFrame(X_resampled, columns=['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'A
MT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_INCOME_TYPE',
                    'NAME_EDUCATION_TYPE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'ORGANIZATION_T
YPE',
                    'CNT_FAM_MEMBERS'])
# Add the scores to the DataFrame
X_resampled_df['Credit Worthiness Score'] = scores

# Summarize the scores
summary = X_resampled_df['Credit Worthiness Score'].describe()
print(summary)
```

```
# The distribution of scores
score_distribution = X_resampled_df['Credit Worthiness Score'].value_counts().sort_index
()
print(score_distribution)
```

```
              precision    recall  f1-score   support

           0       0.74      0.90      0.81     28205
           1       0.87      0.68      0.77     28333

    accuracy                           0.79     56538
   macro avg       0.81      0.79      0.79     56538
weighted avg       0.81      0.79      0.79     56538

8834/8834 [==============================] - 6s 668us/step
count    282686.000000
mean          9.353785
std           6.726406
min           0.000000
25%           4.000000
50%           7.000000
75%          16.000000
max          20.000000
Name: Credit Worthiness Score, dtype: float64
Credit Worthiness Score
0     11328
1     19343
2     17691
3     16655
4     17190
5     18246
6     20503
7     27053
8     14678
9      9360
10     7197
11     6682
12     6206
13     5941
14     5857
15     5325
16     4239
17     2931
18     2153
19    32164
20    31944
Name: count, dtype: int64
```

From above, we can see the credit worthiness scores are classified for different entries with mean value of 9. The accuracy is ~80% in this model with epoch size of 50 and it is still increasing. We might get better values of more than 90% with additional time for model fit and epochs close to 250.

In [ ]: