UNIVERSITY OF ECONOMICS AND LAW

COURSE: DATA ANALYTICS WITH R/PYTHON

Machine learning for Bussiness: Credit Scoring

EPT TEAM

List of members:

Nguyen Dang Khoa	K194060850	(Team leader)
Nguyen Phan Dieu Gam	K194060846	
Phan Thi Le Hien	K194060847	
Truong Thanh Long	K194060853	
Le Mai Ngoc Thy	K194060875	

Supervisor:

Nguyen Van Ho M.A

Ho Chi Minh City, 2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: PROJECT OVERVIEW

1.1. Reasons

In the era of society and the country is developing like today, banking is one of the key industries and plays a very important role that the country is developing comprehensively and strongly. Credit is one of the core factors and flows of the economy. The development of credit is the driving force to promote economic development. But this credit loan is only good when it is at a moderate level. The credit flow crisis has become one of the most pressing problems, the bank has over-issued cash and credit regardless of the customer's ability to repay and also the history of excessive assumptions there. This situation adversely affects the relationship between consumers and financial companies. The growing demand for consumer credit has led to competition in the credit industry.

Along with the development of technology, which has promoted the increasing level of people's consumption, the people's borrowing demand has been boosted, especially for personal consumption loans. Today we are completely dependent on online applications, online shopping and other non-essential needs. From using online shopping apps or any other need that people today spend a lot, from there they also borrow a lot. A lot of people in numerology have not been able to control their spending and income, leading to the inability to pay for the loans they have borrowed. This is one of the banking industry's dilemmas in today's personal consumer lending,

Credit managers must get through this situation as efficiently and quickly as they can. So, by developing and applying models, they can manage and analyze credit data. This will save time, minimize errors and maximize the credit scoring process. Credit scoring is defined as a technique that helps lenders decide whether to grant credit to a customer based on various parameters of the customer such as age, income and marital status, past their relationship with the bank, if the client is involved in any bankruptcy cases or not. Among all existing methods, data mining methods have gained more popularity than the others because of their incomparable capability of determining practical knowledge from the database and changing them into beneficial information. This project will help you better understand Credit Score, predictive machine learning models applied to credit scoring and suggest ‘best model’ for sample datasets.

1.2. Objectives

The objective of the report is to build a basic algorithm-based model for credit classification, compare the difference between applying the algorithm to both new and old customer files, and then recommend the most reliable credit classification model based on the proposed algorithms.

One of the important goals of this report is to suggest a suitable model and improve decision making based on customer credit score thereby saving time and minimizing the process. This will help businesses save a lot of costs and resources.

To reduce research and data processing time, we used tools to support and build algorithms based on the python programming language. So the output will be more stable and systematic.

1.3. Objects and Scopes

Objects:

- Proposing a credit scoring model for financial institutions, lenders, in the banking field.
- Learn and apply the Python programming language and Machine learning Algorithms.

Scopes:

- Using the data set of Credit Score Classification in Kaggle.com to execute data processing. This is one of the websites that provide knowledge about data science and it has the most reliable and large data files used for machine learning research today.

1.4. Research Method

Theoretical research:

- Learn the definition, theory of algorithms, credit scores, how to compare the difference between new and old customer files when applying algorithms, and related research.

- Learn credit classification methods, recommend the right model to improve decision making based on customer's credit score.

Experimental research:

- Collect data, perform pre-processing and data exploration, suggest analytic models to compare the difference between new and old customers, thereby proposing the most reliable credit classification model based on the proposed algorithms.

1.5. Tools and programming language

Programming language: Python.

Tool: PyCharm/Visual Studio Code.

1.6. Structure of project

Chapter 1: Project overview

Chapter 2: Theoretical basis and literature review

Chapter 3: Proposal model for customer

Chapter 4: Implementation on actual data sets

Chapter 5: Experimental result

Chapter 6: Conclusion and Future work

CHAPTER 2: THEORETICAL BASIS AND LITERATURE REVIEW

2.1. Theoretical Basic

2.1.1. Credit scoring

Credit Scoring is a statistical analysis system implemented by financial institutions and lenders with the purpose of determining the creditworthiness of an individual or organization, especially in Banking.

2.1.2. Logistic regression

The logistic regression model is a predictive analysis with two possible outcomes based on a given data set consisting of the independent variables. Since the result is a probability, the dependent variable is bounded between 0 and 1.

In the logistic regression model, a logit transformation is applied to the odds—that is, the probability of success divided by the probability of failure. This is also commonly known as the log odds, or the natural logarithm of odds, and this logistic function is represented by the following formulas:

$$logistic(\eta) = \frac{1}{1 + exp(-\eta)} \quad (1)$$

$$ln\left(\frac{P(y=1)}{1-P(y=1)}\right) = log\left(\frac{P(y=1)}{P(y=0)}\right) = \beta 0 + \beta 1x1 + \ldots + \beta pxp \quad (2)$$

There are three types of logistic regression models:

- Binary logistic regression: has only two possible outcomes (0 or 1). This is the most commonly used approach, and more generally, it is one of the most common classifiers for binary classification.
- Multinomial logistic regression: has three or more possible outcomes and these values have no specified order.
- Ordinal logistic regression: has three or more possible outcomes, but in this case, these values do have a defined order.

2.1.3. Decision tree

Decision Tree is the most powerful and popular tool for classification and prediction. They can be used for both regression and classification. Decision tree analysis is a predictive model that can be applied across many different fields. Decision trees can

be built using an algorithmic approach that can split the data set in different ways based on different conditions. The two main entities of the tree are the decision nodes, where the data is split and leave, where we get the results.

2.1.4. K-Nearest Neighbors

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another. K-Nearest Neighbors captures the idea of similarity (sometimes referred to as distance, proximity, or proximity) to certain operations such as calculating the distance between points on a graph. There are other ways to calculate distance and one may be more suitable depending on the problem we are solving. However, the straight-line distance (also called the Euclidean distance) is a popular and familiar choice.

2.1.5. Random Forest

Random Forests is a supervised learning algorithm. It can be used for both classification and regression. Random Forest is a collection of Decision Trees, each of which is selected according to a random-based algorithm and then takes predictions from each of them and finally selects the best solution by means of voting. This is a better aggregation method than a single tree because it reduces overfitting by averaging the results.

2.2. Literature Review

In the study “Credit Scoring Using Machine Learning”, Kenneth Kennedy and his colleagues (2013) identified and investigated a number of specific challenges that occur during credit scorecard development. Four main contributions are made in their thesis. Firstly, they test the performance of several supervised classification techniques on an unbalanced set of credit-scoring datasets. They then demonstrate that oversampling the minority class results in no overall improvement on the best performing classifiers. Secondly, they investigated a particularly severe form of class imbalance, which in credit scoring is known as the low default portfolio problem. They then suggest two approaches worthy of consideration when dealing with low default portfolios: the use of a semi-supervised classification algorithm with a logistic regression algorithm. Thirdly, they

quantify classifier performance differences that arise from different implementations of the real-world behavioural scoring dataset. Finally, they demonstrate how artificial data can be used to overcome difficulties associated with real-world data collection and use.

CHAPTER 3: PROPOSAL MACHINE MODEL FOR CREDIT DATASET

3.1. Define the analysis goal

Bank customers are divided into two main objects. One is that they have done transactions at the bank, and the other is that they have never used any service. With each object, we will have different data sets, and dependent data will be different, thereby giving different prediction results for the data set of customers who have ever borrowed or used the bank's services, containing information on demographics, loan history, and credit accounts,... As for the new customer data set, it will be missing, losing the history of transactions and loans at the bank,... but overall we can still use machine learning to predict whether this bank machine learning will find out which model is the “best model” from which the results are applied to predict the next set of customers in the future to optimize costs in business strategies and compete in the market in the long run.

3.2. Collect data

The banking sector is the engine that powers economies and countries. And it also generates a lot of data. Every transaction leaves a trace and generates huge amounts of data. A “footprint” or “digital shadow” results from all the activities, actions and traces that a person leaves behind when browsing the Internet. In general, collecting customer data, there are many ways and in this day and age, data is the main factor to determine success in all fields including banking.

3.3. Data exploratory analytics

This is an important stage as it can help or disrupt the entire in-depth data analysis later on. It is the process of analyzing and visualizing data to better understand the data and gather detailed information such as the data size, the meaning of each data field, the data type of each field and the exact distribution of the data. The rate of each field, the correlation between the data fields,...

3.4. Data preprocessing

After data collection, the next step will be data preprocessing. As we know our dataset is created from some raw data resources. Therefore, it is very normal for some data to be incomplete or inconsistent.

- Handling error data: Data in the collection process often fails for a variety of reasons, possibly due to human factors or automated program factors that lack prerequisites. For example: '__', '!@9#%8',…
- Handling redundant data: The data will have unnecessary or invalid fields for analysis that still frequently appear in data sets for purposes other than analysis such as: 'SSN' and 'Name'. So analysis we can drop those columns to increase work efficiency.
- Handling missing data: One of the main steps in data preprocessing is processing data that is missing. Missing data results in no data to observe, and when the data is fed into the machine learning model, it can lead to incorrect predictions and classifications.
- Handling duplicate values: Duplicate entries can crash the split between training, validation and test sets in case of identical entries. This can lead to skewed performance estimates.
- Data transformation: Machine learning models will not be able to work with many different types of data, so if the data has values that are not suitable for machine learning we need to convert them to the appropriate values. It is common to digitize almost any data so that models will be able to perform at their best.
- Handling data imbalance: When the data is too severely imbalanced, it will cause great problems in terms of results and predictions. So in any raw data set, we need to check and handle the problem of data imbalance.

3.5. Building analytical models

The model-building process will go through four stages: model training, cross validation, model testing, and parameter refinement to improve the model.

(i) Training and testing

- The initial data set will be divided with a larger weight for the training set and the rest is used to evaluate the predictive ability of the model. Then, the test set acts as a sample against the population which is the actual data of the problem being solved. Therefore, the results of the model shown on the test set are a reliable measure of the effectiveness of the model.

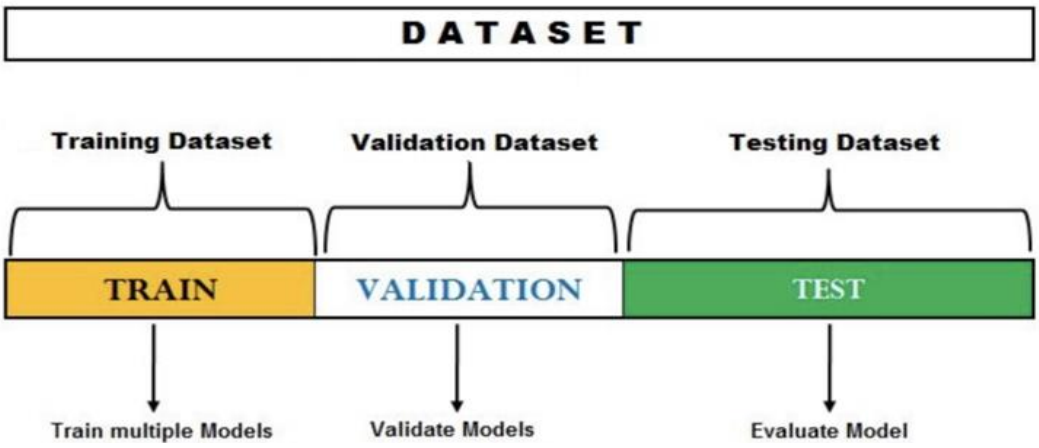


Figure 3. 1. Split data to build models

(ii) Cross Evaluation

- Cross Validation helps to evaluate and compare the predictive ability of machine learning algorithms based on datasets divided into multiple test sets while preventing overmatching.
- Cross evaluation algorithms used:
  - Holdout: this is the simplest cross validation method, the input data set is divided into a training set and a random test set, using the training set to train and the test set to test the model.
  - K-fold: the method is developed based on Holdout, the data set is divided into k subsets, is run for k times, each execution will have k 1 sets used for training and the rest used to test the model.
  - Leave one out: same method as Kfold but will maximize k equal to the number of data.
- The K-fold cross validation method is commonly used, especially when the input data to train the model is not much. Because if we divide the model into three parts as usual, the training set, the evaluation set and the test set, since the division is completely random, it is very likely that some data points are useful for the training process. retraining is in the evaluation or test set, and the model has no chance to learn that data point.

(iii) Model Refinement

- The finetuning process is the modification of the algorithm's set of parameters to improve the results in the prediction. The model is built in turn based on

this set of parameters and then compares the results to choose the parameter set that gives the highest result. Current programming languages have libraries to support this automatically such as GridSearchCV and RandomizedSearchCV in the sklearn library for Python language.

3.6. Proposed model

Based on data set with dependent variables and prediction results. Assume that the assessment of credit scores depends on the dependent variables. There are suitable regression analysis models to use such as:

- **Decision Tree:** Decision trees can be built using an algorithmic approach that can split the data set in different ways based on different conditions. The two main entities of the tree are the decision nodes, where the data is split and left, where we get the results.
- **K-Nearest Neighbors:** The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.
- **Random Forest:** Random Forests is a supervised learning algorithm. Because it is a better aggregation method than a single tree because it reduces overfitting by averaging the results.

CHAPTER 4: IMPLEMENTATION ON ACTUAL DATA SETS

4.1. Machine learning model for Old Customer

4.1.1. Data introduction

Feature	Description
ID	Represents a unique identification of an entry
Customer_ID	Represents a unique identification of a person
Month	Represents the month of the year
Name	Represents the name of a person
Age	Represents the age of the person
SSN	Represents the social security number of a person
Occupation	Represents the occupation of the person
Annual_Income	Represents the annual income of the person
Monthly_Inhand_Salary	Represents the monthly base salary of a person
Num_Bank_Accounts	Represents the number of bank accounts a person holds
Num_Credit_Card	Represents the number of other credit cards held by a person
Interest_Rate	Represents the interest rate on credit card
Num_of_Loan	Represents the number of loans taken from the bank
Num_of_Delayed_Payment	Represents the average number of payments delayed by a person
Changed_Credit_Limit	Represents the percentage change in credit card limit
Num_Credit_Inquiries	Represents the number of credit card inquiries
Credit_Mix	Represents the classification of the mix of credits

Outstanding_Debt	Represents the remaining debt to be paid (in USD)
Credit_Utilization_Ratio	Represents the utilization ratio of credit card
Credit_History_Age	Represents the age of credit history of the person
Payment_of_Min_Amount	Represents whether only the minimum amount was paid by the person
Total_EMI_per_month	Represents the monthly EMI payments (in USD)
Amount_invested_monthly	Represents the monthly amount invested by the customer (in USD)
Payment_Behaviour	Represents the payment behavior of the customer (in USD)
Monthly_Balance	Represents the monthly balance amount of the customer (in USD)

Table 4. 1. Data description

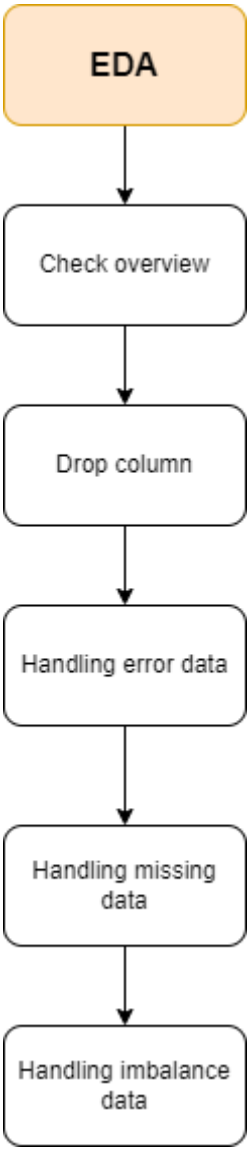
Datasets have 27 columns and 30000 rows. This is the bank's old customer data containing all the information from demographics, base salary, loans, etc. at a bank. In the last column is an assessment of the person's credit score. We can see at part 4.1.2.

4.1.2. Data exploratory analytics

```
train_data = pd.read_csv('data/train_OldCustomer.csv', index_col='ID')
train_data.info()
train_data.describe()
```

```
Index: 100000 entries, 0x1602 to 0x25fed
Data columns (total 27 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   Customer_ID               100000 non-null  object
 1   Month                     100000 non-null  object
 2   Name                      90015 non-null   object
 3   Age                       100000 non-null  object
 4   SSN                       100000 non-null  object
 5   Occupation                100000 non-null  object
 6   Annual_Income             100000 non-null  object
 7   Monthly_Inhand_Salary     84998 non-null   float64
 8   Num_Bank_Accounts         100000 non-null  int64
 9   Num_Credit_Card           100000 non-null  int64
 10  Interest_Rate             100000 non-null  int64
 11  Num_of_Loan               100000 non-null  object
 12  Type_of_Loan              88592 non-null   object
 13  Delay_from_due_date       100000 non-null  int64
 14  Num_of_Delayed_Payment    92998 non-null   object
 15  Changed_Credit_Limit      100000 non-null  object
 16  Num_Credit_Inquiries      98035 non-null   float64
 17  Credit_Mix                100000 non-null  object
 18  Outstanding_Debt          100000 non-null  object
 19  Credit_Utilization_Ratio  100000 non-null  float64
 20  Credit_History_Age        90970 non-null   object
 21  Payment_of_Min_Amount     100000 non-null  object
 22  Total_EMI_per_month       100000 non-null  float64
 23  Amount_invested_monthly   95521 non-null   object
 24  Payment_Behaviour         100000 non-null  object
 25  Monthly_Balance           98800 non-null   object
 26  Credit_Score              100000 non-null  object
dtypes: float64(4), int64(4), object(19)
memory usage: 21.4+ MB
```

This is actual data so there will be a lot of errors. Here is our step by step process, shown in the diagram below:



Step 1: Drop the two columns 'SSN', 'Name', 'Type_of_Loan' because it does not affect the prediction models.

```
train_data.drop(columns=['SSN', 'Name'], axis=1, inplace=True)
```

Step 2: Find columns with sorted values to process

```
categorical_cols = [c for c in train_data.columns if
train_data[c].dtype == 'object']
for col in categorical_cols:
    print(f"Unique Values of {col}")
    print(train_data[col].unique())
    print("==================================================")
```

```
Unique Values of Customer_ID
['CUS_0xd40' 'CUS_0x21b1' 'CUS_0x2dbc' ... 'CUS_0xaf61' 'CUS_0x8600'
 'CUS_0x942c']
==================================================
Unique Values of Month
['January' 'February' 'March' 'April' 'May' 'June' 'July' 'August']
==================================================
Unique Values of Age
['23' '-500' '28_' ... '4808_' '2263' '1342']
==================================================
Unique Values of Occupation
['Scientist' '_______' 'Teacher' 'Engineer' 'Entrepreneur' 'Developer'
 'Lawyer' 'Media_Manager' 'Doctor' 'Journalist' 'Manager' 'Accountant'
 'Musician' 'Mechanic' 'Writer' 'Architect']
==================================================
Unique Values of Annual_Income
['19114.12' '34847.84' '34847.84_' ... '20002.88' '39628.99' '39628.99_']
==================================================
Unique Values of Num_of_Loan
['4' '1' '3' '967' '-100' '0' '0_' '2' '3_' '2_' '7' '5' '5_' '6' '8' '8_'
 '9' '9_' '4_' '7_' '1_' '1464' '6_' '622' '352' '472' '1017' '945' '146'
```

```
Unique Values of Type_of_Loan
['Auto Loan, Credit-Builder Loan, Personal Loan, and Home Equity Loan'
 'Credit-Builder Loan' 'Auto Loan, Auto Loan, and Not Specified' ...
 'Home Equity Loan, Auto Loan, Auto Loan, and Auto Loan'
 'Payday Loan, Student Loan, Mortgage Loan, and Not Specified'
 'Personal Loan, Auto Loan, Mortgage Loan, Student Loan, and Student Loan']
==================================================
```

```
Unique Values of Num_of_Delayed_Payment
['7' nan '4' '8_' '6' '1' '-1' '3_' '0' '8' '5' '3' '9' '12' '15' '17'
 '10' '2' '2_' '11' '14' '20' '22' '13' '13_' '14_' '16' '12_' '18' '19'
 '23' '24' '21' '3318' '3083' '22_' '1338' '4_' '26' '11_' '3104' '21_'
 '25' '10_' '183_' '9_' '1106' '834' '19_' '24_' '17_' '23_' '2672' '20_'
 '2008' '-3' '538' '6_' '1_' '16_' '27' '-2' '3478' '2420' '15_' '707'
 '708' '26_' '18_' '3815' '28' '5_' '1867' '2250' '1463' '25_' '7_' '4126'
 '2882' '1941' '2655' '2628' '132' '3069' '306' '0_' '3539' '3684' '1823'
 '4128' '1946' '827' '2297' '2566' '904' '182' '929' '3568' '2503' '1552'
 '2812' '1697' '3764' '851' '3905' '923' '88' '1668' '3253' '808' '2689'
 '3858' '642' '3457' '1402' '1732' '3154' '847' '3037' '2204' '3103'
```

```
==================================================
Unique Values of Changed_Credit_Limit
['11.27' '_' '6.27' ... '17.509999999999998' '25.16' '21.17']
==================================================
Unique Values of Credit_Mix
['_' 'Good' 'Standard' 'Bad']
==================================================
Unique Values of Outstanding_Debt
['809.98' '605.03' '1303.01' ... '3571.7_' '3571.7' '502.38']
==================================================
Unique Values of Credit_History_Age
['22 Years and 1 Months' nan '22 Years and 3 Months'
 '22 Years and 4 Months' '22 Years and 5 Months' '22 Years and 6 Months'
 '22 Years and 7 Months' '26 Years and 7 Months' '26 Years and 8 Months'
```

```
Unique Values of Payment_of_Min_Amount
['No' 'NM' 'Yes']
==================================================
Unique Values of Amount_invested_monthly
['80.41529543900253' '118.28022162236736' '81.699521264648' ...
 '24.02847744864441' '251.67258219721603' '167.1638651610451']
==================================================
Unique Values of Payment_Behaviour
['High_spent_Small_value_payments' 'Low_spent_Large_value_payments'
 'Low_spent_Medium_value_payments' 'Low_spent_Small_value_payments'
 'High_spent_Medium_value_payments' '!@9#%8'
 'High_spent_Large_value_payments']
==================================================
Unique Values of Monthly_Balance
['312.49408867943663' '284.62916249607184' '331.2098628537912' ...
 516.8090832742814 319.1649785257098 393.6736955618808]
==================================================
Unique Values of Credit_Score
['Good' 'Standard' 'Poor']
==================================================
```

In the columns found above, we handle cases with faulty data such as prefixed with '_', empty string, '!@9#%8', '#F%$D@*&8' all instead. Replace with NaN and try to cast the data type to real number (float64) if possible. Then the values of the data frame categorical_cols remaining are the actual categorical fields

```
for col in categorical_cols:
    train_data[col] = train_data[col].str.strip('_')

    try:
        train_data[col] = train_data[col].astype('float64')
    except:
        train_data[col] = train_data[col]


for col in categorical_cols:
    train_data[col] = train_data[col].replace({'':np.nan})
```

```
    try:
        train_data[col] = train_data[col].astype('float64')
    except:
        train_data[col] = train_data[col]

for col in categorical_cols:
    train_data[col]    =    train_data[col].replace({'!@9#%8':np.nan,
'#F%$D@*&8':np.nan})

#after handle
train_data[categorical_cols].info()
```

```
Data columns (total 4 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Occupation              89616 non-null  object
 1   Credit_Mix              89616 non-null  object
 2   Payment_of_Min_Amount   89616 non-null  object
 3   Payment_Behaviour       89616 non-null  object
dtypes: object(4)
memory usage: 5.4+ MB
```

Step 3: Handle error data of Credit_History_Age

```
credit_age = []
for i in train_data['Credit_History_Age']:
    credit_age.append(str(i).split(' ')[0])

train_data['Credit_History_Age'] = credit_age

train_data['Credit_History_Age']                                    =
train_data['Credit_History_Age'].replace({'nan':np.nan})
train_data['Credit_History_Age']                                    =
train_data['Credit_History_Age'].astype('float64')
```

Step 4: Handle missing data

- Check missing data

```
missing = train_data.isnull().sum()
missing = missing[missing>0]
missing
```

```
Age                         2781
Occupation                  7062
Annual_Income               2783
Monthly_Inhand_Salary      15002
Num_Bank_Accounts           1315
Num_Credit_Card             2271
Interest_Rate               2034
Num_of_Loan                 4348
Delay_from_due_date         4002
Num_of_Delayed_Payment      7002
Changed_Credit_Limit        2091
Num_Credit_Inquiries        1965
Credit_Mix                 20195
Outstanding_Debt            5272
Credit_Utilization_Ratio       4
Credit_History_Age          9030
Total_EMI_per_month         6795
Amount_invested_monthly     4479
Payment_Behaviour           7600
Monthly_Balance             2868
dtype: int64
```

- Handle missing data using ‘fillna’

```
missing_cols = [col for col in missing.index]
missing_cols.append('Customer_ID')
missing_cols
missing_cols_set = set(missing_cols)
missing_cols_set

# numerical cols with missing values
numerical_cols_set = {col for col in train_data.columns if
(train_data[col].dtype=='int64') |
(train_data[col].dtype=='float64')}
numerical_cols_missing = [col for col in
numerical_cols_set.intersection(missing_cols_set)]
numerical_cols_missing.append('Customer_ID')

numerical_cols_missing.pop()

based_customer_ID = [col for col in numerical_cols if
train_data[col].head(8).nunique() == 1]
non_based_customer_ID = [col for col in numerical_cols if
train_data[col].head(8).nunique() != 1]

train_data[based_customer_ID] =
train_data.groupby(by=['Customer_ID'])[based_customer_ID].transform(
'median')
train_data[non_based_customer_ID] =
train_data[non_based_customer_ID].fillna(method='bfill')
```

```
missing = train_data.isnull().sum()
missing = missing[missing>0]

missing_cols_set = {col for col in missing.index}

categorical_cols_set = {col for col in train_data.columns if
train_data[col].dtype == 'object'}

categorical_cols_missing = [col for col in
categorical_cols_set.intersection(missing_cols_set)]

train_data[categorical_cols_missing] =
train_data[categorical_cols_missing].fillna(method='bfill')
```

Step 5: Encode month manually

```
clean_train_data['Month'] =
clean_train_data['Month'].map({'January':1, 'February':2, 'March':3,
'April':4, 'May':5, 'June':6, 'July':7, 'August':8, 'September':9,
'October':10, 'November':11, 'December':12})
```

Step 6: Handle imbalance data

- Check imbalance data:

```
y.value_counts()
```

```
2    47976
1    25279
3    16361
Name: Credit_Score, dtype: int64
```

Handle imbalanced data using imblearn.over_sampling of SMOTE library, it will automatically use the sample() function in python to increase the number of predictors by the variable with the highest number. According to the over-sampling method.

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy = 'minority')
Final_X, y = smote.fit_resample(Final_X,y)
```

4.1.3. Proposed Model

Select variables to predict using models

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(Final_X,y,test_size=0
.20,random_state=500)
```

4.1.3.1. Logistic regression

```
from sklearn.linear_model import LogisticRegression

lr=LogisticRegression()
lr.fit(x_train,y_train)

y_pred_train_lr=lr.predict(x_train)
y_pred_test_lr=lr.predict(x_test)
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score
print(classification_report(y_train,y_pred_train_lr))
print('*'*20)
print(classification_report(y_test,y_pred_test_lr))
```

```
              precision    recall  f1-score   support

           1       0.65      0.51      0.57     20209
           2       0.64      0.64      0.64     38236
           3       0.72      0.81      0.76     38539

    accuracy                           0.68     96984
   macro avg       0.67      0.65      0.66     96984
weighted avg       0.67      0.68      0.67     96984

********************
              precision    recall  f1-score   support

           1       0.65      0.51      0.57      5070
           2       0.65      0.64      0.64      9740
           3       0.71      0.80      0.75      9437

    accuracy                           0.68     24247
   macro avg       0.67      0.65      0.66     24247
weighted avg       0.67      0.68      0.67     24247
```

```
print(accuracy_score(y_train, y_pred_train_lr))
print("\n")
print(accuracy_score(y_test, y_pred_test_lr))
```

[61] ✓ 0.3s

```
0.6777715911902994
```

4.1.3.2. Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

dt_gini=DecisionTreeClassifier(criterion='gini')
```

```
dt_entropy=DecisionTreeClassifier(criterion='entropy')
dt_gini.fit(x_train,y_train)
dt_entropy.fit(x_train,y_train)

y_pred_g_train=dt_gini.predict(x_train)
y_pred_g_test=dt_gini.predict(x_test)
y_pred_e_train=dt_entropy.predict(x_train)
y_pred_e_test=dt_entropy.predict(x_test)

print(classification_report(y_train,y_pred_g_train))
print("\n")
print(classification_report(y_test,y_pred_g_test))
```

```
              precision    recall  f1-score   support

           1       1.00      1.00      1.00     20209
           2       1.00      1.00      1.00     38236
           3       1.00      1.00      1.00     38539

    accuracy                           1.00     96984
   macro avg       1.00      1.00      1.00     96984
weighted avg       1.00      1.00      1.00     96984



              precision    recall  f1-score   support

           1       0.69      0.68      0.69      5070
           2       0.75      0.74      0.74      9740
           3       0.85      0.86      0.86      9437

    accuracy                           0.78     24247
   macro avg       0.76      0.76      0.76     24247
weighted avg       0.78      0.78      0.78     24247
```

```
print(accuracy_score(y_train,y_pred_g_train))
print("\n")
print(accuracy_score(y_test,y_pred_g_test))
print("\n")
print(accuracy_score(y_train,y_pred_e_train))
print("\n")
print(accuracy_score(y_test,y_pred_e_test))
```

[66] ✓ 0.4s

```
1.0


0.7761372540932899


1.0


0.7731265723594671
```

4.1.3.3. Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=500,criterion='gini')
rf_model.fit(x_train, y_train)

y_pred_rf_train = rf_model.predict(x_train)
y_pred_rf_test = rf_model.predict(x_test)

print(classification_report(y_train, y_pred_rf_train))
print(classification_report(y_test, y_pred_rf_test))
```

```
                precision    recall  f1-score   support

           1       1.00      1.00      1.00     20209
           2       1.00      1.00      1.00     38236
           3       1.00      1.00      1.00     38539

    accuracy                           1.00     96984
   macro avg       1.00      1.00      1.00     96984
weighted avg       1.00      1.00      1.00     96984

                precision    recall  f1-score   support

           1       0.81      0.83      0.82      5070
           2       0.88      0.80      0.83      9740
           3       0.88      0.96      0.92      9437

    accuracy                           0.86     24247
   macro avg       0.86      0.86      0.86     24247
weighted avg       0.86      0.86      0.86     24247
```

```
print(accuracy_score(y_train, y_pred_rf_train))
print("\n")
print(accuracy_score(y_test, y_pred_rf_test))
```

[72] ✓ 0.5s

```
1.0


0.8648080174866993
```

4.1.3.4. K-Nearest Neighbors

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x_knn)
scaled_features=scaler.transform(x_knn)

df_feat=pd.DataFrame(scaled_features,columns=x_knn.columns)
df_feat

y_pred_knn_train=knn.predict(x_train_knn)
y_pred_knn_test=knn.predict(x_test_knn)

print(classification_report(y_train,y_pred_knn_train))
print("\n")
```

```
print(classification_report(y_test,y_pred_knn_test))
```

```
...                precision    recall  f1-score   support

           1       0.78      0.73      0.76     19002
           2       0.84      0.75      0.79     36009
           3       0.82      0.94      0.87     35912

    accuracy                           0.82     90923
   macro avg       0.81      0.81      0.81     90923
weighted avg       0.82      0.82      0.82     90923



                precision    recall  f1-score   support

           1       0.74      0.68      0.71      6277
           2       0.80      0.70      0.74     11967
           3       0.79      0.92      0.85     12064

    accuracy                           0.78     30308
   macro avg       0.78      0.77      0.77     30308
weighted avg       0.78      0.78      0.78     30308
```

```
print(accuracy_score(y_train,y_pred_knn_train))
print("\n")
print(accuracy_score(y_test,y_pred_knn_test))
```

```
[ ]
...  0.8196495936121774


     0.7831265672429721
```

4.2. New Customer

4.2.1. Data introduction

Feature	Description
ID	Represents a unique identification of an entry

Customer_ID	Represents a unique identification of a person
Month	Represents the month of the year
Name	Represents the name of a person
Age	Represents the age of the person
SSN	Represents the social security number of a person
Occupation	Represents the occupation of the person
Annual_Income	Represents the annual income of the person
Monthly_Inhand_Salary	Represents the monthly base salary of a person
Num_Bank_Accounts	Represents the number of bank accounts a person holds
Num_Credit_Card	Represents the number of other credit cards held by a person
Interest_Rate	Represents the interest rate on credit card
Num_of_Loan	Represents the number of loans taken from the bank
Amount_invested_monthly	Represents the monthly amount invested by the customer (in USD)
Monthly_Balance	Represents the monthly balance amount of the customer (in USD)

Table 4. 2. Data description

Assuming the above data set, removing the fields of previous transactions such as: Num_of_Loan, Num_Credit_Inquiries, Credit_History_Age .etc the data set will become completely new customer data, we only have the following information. Just their basic beliefs. Datasets have 15 columns and 30000 rows. This is the bank's new customer data containing all the information from demographics, base salary,…

4.2.2. Data exploratory analytics

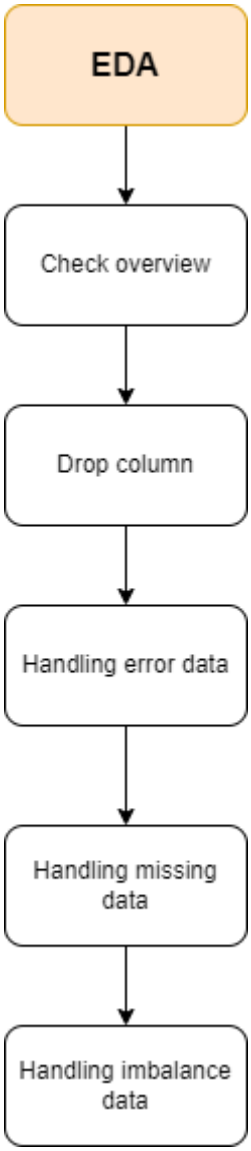
```
train_data = pd.read_csv('data/train_NewCustomer.csv', index_col='ID')
train_data.info()
```

```
train_data.describe()
```

```
Data columns (total 15 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   Customer_ID               100000 non-null  object
 1   Month                     100000 non-null  object
 2   Name                      90015 non-null   object
 3   Age                       100000 non-null  object
 4   SSN                       100000 non-null  object
 5   Occupation                100000 non-null  object
 6   Annual_Income             100000 non-null  object
 7   Monthly_Inhand_Salary     84998 non-null   float64
 8   Num_Bank_Accounts         100000 non-null  int64
 9   Num_Credit_Card           100000 non-null  int64
 10  Num_of_Loan               100000 non-null  object
 11  Payment_of_Min_Amount     100000 non-null  object
 12  Total_EMI_per_month       100000 non-null  float64
 13  Amount_invested_monthly   95521 non-null   object
 14  Credit_Score              100000 non-null  object
dtypes: float64(2), int64(2), object(11)
memory usage: 12.2+ MB
```

Fig: Dataset structure

This is actual data so there will be a lot of errors. Here is our step by step process, shown in the diagram below:



Step 1: Drop the two columns 'SSN', 'Name', 'Type_of_Loan' because it does not affect the prediction models.

```
train_data.drop(columns=['SSN', 'Name'], axis=1, inplace=True)
```

Step 2: Find columns with sorted values to process

```
categorical_cols = [c for c in train_data.columns if
train_data[c].dtype == 'object']
for col in categorical_cols:
    print(f"Unique Values of {col}")
    print(train_data[col].unique())
    print("==================================================")
```

```
Unique Values of Customer_ID
['CUS_0xd40' 'CUS_0x21b1' 'CUS_0x2dbc' ... 'CUS_0xaf61' 'CUS_0x8600'
 'CUS_0x942c']
==================================================
Unique Values of Month
['January' 'February' 'March' 'April' 'May' 'June' 'July' 'August']
==================================================
Unique Values of Age
['23' '-500' '28_' ... '4808_' '2263' '1342']
==================================================
Unique Values of Occupation
['Scientist' '_______' 'Teacher' 'Engineer' 'Entrepreneur' 'Developer'
 'Lawyer' 'Media_Manager' 'Doctor' 'Journalist' 'Manager' 'Accountant'
 'Musician' 'Mechanic' 'Writer' 'Architect']
==================================================
Unique Values of Annual_Income
['19114.12' '34847.84' '34847.84_' ... '20002.88' '39628.99' '39628.99_']
==================================================
Unique Values of Num_of_Loan
['4' '1' '3' '967' '-100' '0' '0_' '2' '3_' '2_' '7' '5' '5_' '6' '8' '8_'
 '9' '9_' '4_' '7_' '1_' '1464' '6_' '622' '352' '472' '1017' '945' '146'
```

```
Unique Values of Payment_of_Min_Amount
['No' 'NM' 'Yes']
==================================================
Unique Values of Amount_invested_monthly
['80.41529543900253' '118.28022162236736' '81.699521264648' ...
 '24.02847744864441' '251.67258219721603' '167.1638651610451']
==================================================
Unique Values of Payment_Behaviour
['High_spent_Small_value_payments' 'Low_spent_Large_value_payments'
 'Low_spent_Medium_value_payments' 'Low_spent_Small_value_payments'
 'High_spent_Medium_value_payments' '!@9#%8'
 'High_spent_Large_value_payments']
==================================================
Unique Values of Monthly_Balance
['312.49408867943663' '284.62916249607184' '331.2098628537912' ...
 516.8090832742814 319.1649785257098 393.6736955618808]
==================================================
Unique Values of Credit_Score
['Good' 'Standard' 'Poor']
==================================================
```

In the columns found above, we handle cases with faulty data such as prefixed with '_', empty string, '!@9#%8', '#F%$D@*&8' all instead. Replace with NaN and try to cast the data type to real number (float64) if possible. Then the values of the data frame categorical_cols remaining are the actual categorical fields

```
for col in categorical_cols:
    train_data[col] = train_data[col].str.strip('_')

    try:
        train_data[col] = train_data[col].astype('float64')
```

```
    except:
        train_data[col] = train_data[col]


for col in categorical_cols:
    train_data[col] = train_data[col].replace({'':np.nan})

    try:
        train_data[col] = train_data[col].astype('float64')
    except:
        train_data[col] = train_data[col]

for col in categorical_cols:
    train_data[col]    =    train_data[col].replace({'!@9#%8':np.nan,
'#F%$D@*&8':np.nan})
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 100000 entries, 0x1602 to 0x25fed
Data columns (total 9 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   Customer_ID               100000 non-null  object
 1   Month                     100000 non-null  object
 2   Age                       100000 non-null  float64
 3   Occupation                92938 non-null   object
 4   Annual_Income             100000 non-null  float64
 5   Num_of_Loan               100000 non-null  float64
 6   Payment_of_Min_Amount     100000 non-null  object
 7   Amount_invested_monthly   95521 non-null   float64
 8   Credit_Score              100000 non-null  object
dtypes: float64(4), object(5)
```

Step 3: handle error data of Credit_History_Age

```
credit_age = []
for i in train_data['Credit_History_Age']:
    credit_age.append(str(i).split(' ')[0])

train_data['Credit_History_Age'] = credit_age

train_data['Credit_History_Age']                                    =
train_data['Credit_History_Age'].replace({'nan':np.nan})
train_data['Credit_History_Age']                                    =
train_data['Credit_History_Age'].astype('float64')
```

Step 4: Handle missing data

- Check missing data

```
missing = train_data.isnull().sum()
missing = missing[missing>0]
missing
```

```
Age                       2781
Occupation                7062
Annual_Income             2783
Monthly_Inhand_Salary    15002
Num_Bank_Accounts         1315
Num_Credit_Card           2271
Num_of_Loan               4348
Total_EMI_per_month       6795
Amount_invested_monthly   4479
dtype: int64
```

- Handle missing data using ‘fillna’

```
missing_cols = [col for col in missing.index]
missing_cols.append('Customer_ID')
missing_cols
missing_cols_set = set(missing_cols)
missing_cols_set

# numerical cols with missing values
numerical_cols_set = {col for col in train_data.columns if
(train_data[col].dtype=='int64') |
(train_data[col].dtype=='float64')}
numerical_cols_missing = [col for col in
numerical_cols_set.intersection(missing_cols_set)]
numerical_cols_missing.append('Customer_ID')

numerical_cols_missing.pop()

based_customer_ID = [col for col in numerical_cols if
train_data[col].head(8).nunique() == 1]
non_based_customer_ID = [col for col in numerical_cols if
train_data[col].head(8).nunique() != 1]

train_data[based_customer_ID] =
train_data.groupby(by=['Customer_ID'])[based_customer_ID].transform(
'median')
train_data[non_based_customer_ID] =
train_data[non_based_customer_ID].fillna(method='bfill')

missing = train_data.isnull().sum()
missing = missing[missing>0]

missing_cols_set = {col for col in missing.index}

categorical_cols_set = {col for col in train_data.columns if
train_data[col].dtype == 'object'}
```

```
categorical_cols_missing = [col for col in
categorical_cols_set.intersection(missing_cols_set)]

train_data[categorical_cols_missing] =
train_data[categorical_cols_missing].fillna(method='bfill')
```

Step 5: Encode month manually

```
clean_train_data['Month'] =
clean_train_data['Month'].map({'January':1, 'February':2, 'March':3,
'April':4, 'May':5, 'June':6, 'July':7, 'August':8, 'September':9,
'October':10, 'November':11, 'December':12})
```

Step 6: Handle imbalance data

Check imbalance data:

```
y.value_counts()
```

```
2    47976
1    25279
3    16361
Name: Credit_Score, dtype: int64
```

Handle imbalanced data using imblearn.over_sampling of SMOTE library, it will automatically use the sample() function in python to increase the number of predictors by the variable with the highest number. According to the over-sampling method

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(sampling_strategy = 'minority')
Final_X, y = smote.fit_resample(Final_X,y)
```

4.2.3. Proposed Model

Select variables to predict using models

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(Final_X,y,test_size=0
.20,random_state=500)
```

4.2.3.1. Logistic regression

```
from sklearn.linear_model import LogisticRegression

lr=LogisticRegression()
lr.fit(x_train,y_train)

y_pred_train_lr=lr.predict(x_train)
```

```
y_pred_test_lr=lr.predict(x_test)
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score
print(classification_report(y_train,y_pred_train_lr))
print('*'*20)
print(classification_report(y_test,y_pred_test_lr))
```

```
**********
              precision    recall  f1-score   support

           1       0.53      0.36      0.43      5626
           2       0.55      0.52      0.53     10197
           3       0.66      0.81      0.73     10336

    accuracy                           0.60     26159
   macro avg       0.58      0.56      0.56     26159
weighted avg       0.59      0.60      0.59     26159
```

```
print(accuracy_score(y_train, y_pred_train_lr))
print("\n")
print(accuracy_score(y_test, y_pred_test_lr))
```

```
0.5980522397331626


0.6006345808326007
```

4.2.3.2. Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

dt_gini=DecisionTreeClassifier(criterion='gini')
dt_entropy=DecisionTreeClassifier(criterion='entropy')
dt_gini.fit(x_train,y_train)
dt_entropy.fit(x_train,y_train)

y_pred_g_train=dt_gini.predict(x_train)
y_pred_g_test=dt_gini.predict(x_test)
y_pred_e_train=dt_entropy.predict(x_train)
y_pred_e_test=dt_entropy.predict(x_test)

print(classification_report(y_train,y_pred_g_train))
print("\n")
print(classification_report(y_test,y_pred_g_test))
```

	precision	recall	f1-score	support
1	0.71	0.71	0.71	5626
2	0.76	0.74	0.75	10197
3	0.88	0.89	0.88	10336
accuracy			0.80	26159
macro avg	0.78	0.78	0.78	26159
weighted avg	0.79	0.80	0.79	26159

4.2.3.3. Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf_model = RandomForestClassifier(n_estimators=500,criterion='gini')
rf_model.fit(x_train, y_train)

y_pred_rf_train = rf_model.predict(x_train)
y_pred_rf_test = rf_model.predict(x_test)

print(classification_report(y_train, y_pred_rf_train))
print(classification_report(y_test, y_pred_rf_test))
```

	precision	recall	f1-score	support
1	0.80	0.80	0.80	5626
2	0.83	0.82	0.82	10197
3	0.92	0.94	0.93	10336
accuracy			0.86	26159
macro avg	0.85	0.85	0.85	26159
weighted avg	0.86	0.86	0.86	26159

```
[69]
print(accuracy_score(y_train, y_pred_rf_train))
print("\n")
print(accuracy_score(y_test, y_pred_rf_test))
```

```
1.0


0.8608509499598609
```

4.2.3.4. K-Nearest Neighbors

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x_knn)
scaled_features=scaler.transform(x_knn)

df_feat=pd.DataFrame(scaled_features,columns=x_knn.columns)
df_feat

y_pred_knn_train=knn.predict(x_train_knn)
y_pred_knn_test=knn.predict(x_test_knn)

print(classification_report(y_train,y_pred_knn_train))
print("\n")
print(classification_report(y_test,y_pred_knn_test))
```

```
              precision    recall  f1-score   support

           1       0.61      0.59      0.60      6864
           2       0.69      0.61      0.65     12823
           3       0.78      0.87      0.82     13011

    accuracy                           0.71     32698
   macro avg       0.69      0.69      0.69     32698
weighted avg       0.71      0.71      0.71     32698
```

```
print(accuracy_score(y_train,y_pred_knn_train))
print("\n")
print(accuracy_score(y_test,y_pred_knn_test))
```

[81]

```
0.770567007156401


0.7115725732460702
```

CHAPTER 5: EXPERIMENTAL RESULT

Applying machine learning models to forecast, we see that the accuracy scores are quite high, almost all of them are greater than 50%, proving that the application of machine learning models to this bank can give predictable results. can be trusted. The specific accuracy score results are as follows:

Machine learning model	Old Customer	New Customer
Logistic Regression	67.3%	60%
Decision Tree	77,89%	79.8%
Random Forest	86.4%	86%
K-Nearest Neighbors	78%	71%

Table 5. 1. Accuracy Score

For old customers, it is very clear that with the Random Forest model predicting the highest rate, it can be concluded that the bank with the above data set can reliably apply the Random Forest model in the jobs. prediction in the future. As for new customers, although the fake data is taken from old customer data, omitting the fields that only old customers have, it still gives fairly high predictive data, close to the rate achieved. Prediction from original data, so if new customers do not have information about transaction history, we can completely use Random Forest model to predict. However, basic customer data is still needed to make predictive models increasingly accurate.

CHAPTER 6: CONCLUSION AND FUTURE WORK

Nowadays, information has become an important part of people's lives, especially for businesses. The application of data analytics is one of the very wise steps to develop and expand the market. Data analytic is the process of diving into data sets to analyze and find hidden values within them, thereby extracting and turning them into an understandable structure to use for a variety of purposes. In predicting credit score reliability in particular and in the banking sector in general, machine learning models can be applied very effectively. The evidence analyzed above has shown that the models are almost predictive with a good rate more than 50%. Thereby we can completely use machine learning models for future business strategies.

In this report, we analyzed that more accurately to predict the credit score reliability. Using the Decision Tree, Logistic Regression, Random Forest, and K-Nearest Neighbors Algorithms by applying the following steps: DEA, Handling Error data, Handling Missing Data, Handling Imbalance Data. The output indicates that the Random Forest model is the most reliable with indexes: 86,4% (old customer), 86,4% (new customer) . In conclusion, we can conclude that we should use Random Forest as the classifier to create the model and move on to further analysis.

REFERENCES

[1] Kenneth Kennedy, Brian Mac Namee, Sarah Jane Delany, P´adraig Cunningham. (2013). Credit Scoring Using Machine Learning, Available at: https://arrow.tudublin.ie/cgi/viewcontent.cgi?article=1138&context=sciendoc (Accessed: 1 Dec 2022).

[2] Sounak Sarkar. Credit Score Classification, , Available at: https://www.kaggle.com/code/sounaksarkar/credit-score-classification/data (Accessed: 1 Dec 2022).