**Machine Learning**

**Course Code: MEAD-656**

**Lab Practical File**

**Submitted by**

**Shantanu Shukla**

**01211805424**

**in partial fulfillment for the award of the degree**

**of**

**Master of Technology (AI & DS)**

**batch of 2024 - 26 (2nd Semester)**

****

**Center for Development of Advanced Computing, Noida**

**Affiliated to Guru Gobind Singh Indraprastha University**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| 1. **No** | **Title** | **Page No.** | **Signature** |
| 1 | **Exercise-1:** Decision Tree I |  |  |
| 2 | **Exercise-2:** Decision Tree II |  |  |
| 3 | **Exercise-3:** Data Preprocessing |  |  |
| 4 | **Exercise-4:** Decision Tree III |  |  |
| 5 | **Exercise-5:** Artificial Neural Networks |  |  |
| 6 | **Exercise-6:** Naïve Bayesian Classifier |  |  |
| 7 | **Exercise-7:** Logistic Regression |  |  |
| 8 | **Exercise-8:** Nearest-Neighbour Classifier |  |  |
| 9 | **Exercise-9:** Clustering |  |  |
| 10 | **Exercise-10:** K-means Clustering |  |  |
| 11 | **Exercise-11:** Credit card Approval detection using Machine Learning through financial data |  |  |

**Exercise-1**

**Objective:** Decision Tree I.

# Step1. Open the data/iris.arff Dataset

Click the “***Open file…***” button to open a data set and double click on the “***data***” directory.

Weka provides a number of small common machine learning datasets that you can use to practice on.

Select the “***iris.arff***” file to load the Iris dataset.

A screenshot of a computer

AI-generated content may be incorrect.

The Iris flower dataset is a famous dataset from statistics and is heavily borrowed by researchers in machine learning. It contains 150 instances (rows) and 4 attributes (columns) and a class attribute for the species of iris flower (one of setosa, versicolor, virginica).

# Step2. Select and Run an Algorithm

Now that you have loaded a dataset, it’s time to choose a machine learning algorithm to model the problem and make predictions.

Click the “***Classify***” tab. This is the area for running algorithms against a loaded dataset in Weka. You will note that the “***ZeroR***” algorithm is selected by default.

The ZeroR algorithm selects the majority class in the dataset (all three species of iris are equally present in the data, so it picks the first one: setosa) and uses that to make all predictions. This is the baseline for the dataset and the measure by which all algorithms can be compared. The result is **33%**, as expected (3 classes, each equally represented, assigning one of the three to each prediction results in 33% classification accuracy).

You will also note that the test options uses Cross Validation by default with 10 folds. This means that the dataset is split into 10 parts, the first 9 are used to train the algorithm, and the 10th is used to assess the algorithm. This process is repeated allowing each of the 10 parts of the split dataset a chance to be the held out test set.

# Step3: Choose other Algorithm

Click the “***Choose***” button in the “Classifier” section and click on “***trees***” and click on the “***J48***” algorithm.

This is an implementation of the C4.8 algorithm in Java (“J” for Java, 48 for C4.8, hence the J48 name) and is a minor extension to the famous C4.5 algorithm.

Click the “***Start***” button to run the algorithm.

A screenshot of a computer

AI-generated content may be incorrect.

# Step4: Review Results

After running the J48 algorithm, you can note the results in the “Classifier output” section.

The algorithm was run with 10 fold cross validation, this means it was given an opportunity to make a prediction for each instance of the dataset (with different training folds) and the presented result is a summary of those predictions.

Firstly, note the Classification Accuracy. You can see that the model achieved a result of **144/150**

correct or **96%**, which seems a lot better than the baseline of **33%**.

Secondly, look at the Confusion Matrix. You can see a table of actual classes compared predicted classes and you can see that was 1 error where a Iris-setosa was classified as a Iris-versicolor, 2 cases where Iris-virginica was classified as a Iris-versicolor and 3 cases where a Iris-versicolor was classified as a Iris-setosa (a total of 6 errors). This table can help to explain the accuracy achieved by the algorithm.

You can see the output given by J48 algorithm in a tree fashion by right click on the Results List and by choosing the option Visualize Tree.

A screenshot of a computer

AI-generated content may be incorrect.

**Results:**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise-2**

**Objective:** Decision Tree II.

In this assignment we use a data set of animals and their attributes. Using a decision tree classifier the computer learns to classify animals into different categories (mammals, fish, reptiles etc). Use the **zoo.xls** data set

1. Study the animals in the Excel document (zoo.xls). Without using a data mining tool, draw a decision tree of three to five levels deep that classifies animals into a mammal, bird, reptile, fish, amphibian, insect or invertebrate.
2. Open the zoo.arff data set in WEKA and Find out:
   1. How many animals are there in the data set?
   2. How many attributes are known of each animal?
3. Let us build some classifiers. Go to the classifier tab. We will use 66% of the animals to build the models, and the remaining 34% to evaluate the quality of the model, so select **percentage split – 66%.** First we will build a ‘naïve’ model that just predicts the most occurring class in the data set for each animal. This corresponds to a decision tree of depth 0. Click start to build a model.
   1. What % of animals is correctly classified?
   2. Into what category are all these animals classified and why?
4. Now build a **decision tree of depth 1** (a.k.a. a decision stump - select choose – trees – **decision stump**).
   1. Draw the discovered decision tree.
   2. What % of animals is correctly classified?
   3. Give an example of an animal that would not be classified correctly by this model.
5. Now build a decision tree of **any depth** (a.k.a. a **J48 tree**).Go to the classifier tab and select the decision tree classifier j48. Click on the line behind the choose button. This shows you the parameters you can set and a button called 'More'.
   1. Which algorithm is implemented by j48?
   2. Draw the discovered decision tree.
   3. What percentage of instances is correctly classified by j48?
   4. Which families are mistaken for each other?
6. Again go to the parameter settings by clicking on the box after the 'Choose' button. Now change binarySplit to true and build a new decision tree.
   1. Draw the discovered decision tree.
   2. What is the difference?
   3. What % of animals is correctly classified?
   4. Give an example of an animal that would not be classified correctly by this model.

**Results:**

**Dataset Overview:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise-3**

**Objective:** Data Preprocessing.

# Step1. Open the data/bank‐data.csv Dataset

Click the “***Open file…***” button to open a data set and double click on the “***data***” directory. Select the “bank‐data.csv” file to load the bank dataset.

|  |  |
| --- | --- |
| id | a unique identification number |
| age | age of customer in years (numeric) |
| sex | MALE / FEMALE |
| region | inner\_city/rural/suburban/town |
| income | income of customer (numeric) |
| married | is the customer married (YES/NO) |
| children | number of children (numeric) |
| car | does the customer own a car (YES/NO) |
| save\_acct | does the customer have a saving account (YES/NO) |
| current\_acct | does the customer have a current account (YES/NO) |
| mortgage | does the customer have a mortgage (YES/NO) |
| pep | did the customer buy a PEP (Personal Equity Plan) after the last mailing (YES/NO) |

# Step2. Selecting or Filtering Attributes

In our sample data file, each record is uniquely identified by a customer id (the "id" attribute). We need to remove this attribute before the data mining step.

* We can do this by simply select the attribute and click on “Remove button”
* Using the Attribute filters in WEKA. In the "Filter" panel, click on the "Choose" button. This will show a popup window with a list available filters. Scroll down the list and select the "weka.filters.unsupervised.attribute.Remove" and click apply.
* Save the file with a name "bank‐data‐R1.arff"

# Step3: Discretization

Some techniques, such as association rule mining, can only be performed on categorical data. This requires performing discretization on numeric or continuous attributes. There are 3 such attributes in this data set: "age", "income", and "children". In the case of the "children" attribute the range of possible values are only 0, 1, 2, and 3. In this case, we have opted for keeping all of these values in the data. This means we can simply discretize by removing the keyword "numeric" as the type for the "children" attribute in the ARFF file, and replacing it with the set of discrete values. We do this directly in our text editor. In this case, we have saved the resulting relation in a separate file "bank‐ data2.arff".

Load the bank‐data2.arff dataset into weka.

If we select the "children" attribute in this new data set, we see that it is now a categorical attribute with four possible discrete values.

once again we choose the Filter dialog box, but this time, we will select Discretize from the list. Next, to change the defaults for this filter, click on the box immediately to the right of the "Choose" button. This will open the Discretize Filter dialog box. We enter the index for the attributes to be discretized. In this case we enter 1 corresponding to attribute "age". We also enter 3 as the number of bins (note that it is possible to discretize more than one attribute at the same time by using a list of attribute indices).

Save the file as bank‐data3.arff and check the dataset in text editor.

Next, we apply the same process to discretize the "income" attribute into 3 bins. Again, Weka automatically performs the binning and replaces the values in the "income" column with the appropriate automatically generated labels. We save the new file into "bank‐data3.arff", replacing the older version. Editing by text editor made bank‐data‐final.arff.

# Step4: Missing Values

1. Open file “bank‐data.arff”
2. Check if there is any missing values in any attribute.
3. Edit data to make some missing values.
4. Delete some data in “region”(Nominal) and “children”(Numeric) attributes. Click on “OK” button when finish.
5. Make note of Label that has Max Count in “region” and Mean of “children” attributes.
6. Choose “ReplaceMissingValues” filter (weka.filters.unsupervised.attribute.ReplaceMissingValues). Then, click on Apply button.
7. Look into the data. How did those missing values get replaced ?
8. Edit “bank‐data.arff” with text editor. Make some data missing by replacing them with ‘?’. (Try with nominal and numeric attributes). Save to “bank‐data‐missing.arff”.
9. Load “bank‐data‐missing.arff” into WEKA, observe the data and attribute information.
10. Replace missing values by the same procedure you had done before.

**Results:**

**Step-1: Open and observe the “bank” dataset.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Step-2: Remove attribute “id**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Step-3:**

1. **Discretize “children” attribute manually using text-editor.**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **Discretize “age” attribute in weka.**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **Discretize “income” attribute in weka.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Step-4:**

1. **Create missing values in “region” and “children” attributes.**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **Replace above created missing values.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

* The missing values were automatically replaced randomly with one of the existing values in other records in the dataset.

1. **Make missing data using text-editor.**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **Replace above missing data.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise-4**

**Objective:** Decision Tree III.

1. You are required to work on the "diabetes.arff" data set. You have to apply the J48 classification algorithm with -split percentage as 80. This option will create a training data set with 80% of the points, and use the remaining points for testing.

The focus of this exercise is to visualise the effect of the parameter called the 'Confidence Factor'. Once the tree has been built (nodes with fewer than minNumObj data points are not split), a further pass of pruning is performed by applying the confidence factor. The confidence factor may be viewed as how confident the algorithm is about the training data. (confidence factor represents a threshold of allowed inherent error in data while pruning the decision tree.) A low value leads to more pruning; a high value keeps the model close to the original tree built from the training data (the parameter is used in estimating error probabilities at leaves).

Run experiments with the following values of the confidence factor (default 0.25, maximum

0.5), with the train/test split as specified.

**Confidence Factor:** 0.002 , 0.008 , 0.02 , 0.08 , 0.1 , 0.2 , 0.3 , 0.4 , 0.5.

For these values, record both the training and test accuracy.

1. Comparing decision trees and neural networks

Apply both J48 and the MultilayerPerceptron to the following data sets: "bank-data.arff". Note down the accuracy (5-fold cross-validation) while keeping the number of epochs for the Multilayer Perceptron as 3. Keep all the other parameters at their default settings.

1. What accuracy do these methods achieve on "bank-data.arff" ?
2. Can you think of reasons to explain their relative accuracy on these data sets?
3. What are the properties of the data sets?
4. Describe the decision tree model that is learned in each case.

**Results:**

1. **Comparing decision trees and neural networks**

A screenshot of a computer

AI-generated content may be incorrect.

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise-5**

**Objective:** Artificial Neural Networks.

1. Run the classifier and observe the results shown in the “Classifier output” window. (Note that

by default, WEKA use one hidden layer with the number of hidden neurons = (# of input

attributes + # of classes) / 2.)

1.1 How many units in the input layer, how many in the hidden layer, and how many in

the output layer?

1.2 What is the MAE (mean absolute error) made by the learned NN?

1.3 What is the RMSE (root mean squared error) made by the learned NN?

1.4 Visualize the errors made by the learned NN. In the plot, how can you see the detailed

information of a test instance?

1.5 Draw (on paper) the topology (together with the weights) of the learned NN.

2. To modify the value of a parameter, click on the “MultiplayerPerceptron” label. Set the value

of the “momentum” parameter equal to 0.7. Run the classifier and observe the results shown

in the “Classifier output” window.

2.1 What is the MAE (mean absolute error) made by the learned NN?

2.2 What is the RMSE (root mean squared error) made by the learned NN?

2.3 Visualize the errors made by the learned NN. In the plot, how can you see the detailed

information of a test instance?

3. Now, we shall modify the network structure. Click on the “MultiplayerPerceptron” label. Set

the value of the “hiddenLayers” to “3, 2”. Run the classifier and observe the results shown in

the “Classifier output” window.

3.1 How many layers does the current NN have?

3.2 How many units in each (input/hidden/output) layer?

3.3 What is the MAE (mean absolute error) made by the learned NN?

3.4 What is the RMSE (root mean squared error) made by the learned NN?

3.5 Visualize the errors made by the learned NN. In the plot, how can you see the detailed

information of a test instance?

3.6 Draw (on paper) the topology (together with the weights) of the learned NN.

**Results:**

**1.4.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**2.3.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**3.5.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise-6**

**Objective:** Naïve Bayesian Classifier.

The use of the Naive Bayesian classifier in Weka is demonstrated in this assignment. The “weather-nominal” data set used in this experiment is available in ARFF format. This assignment assumes that the data has been properly preprocessed.

The Bayes’ Theorem is used to build a set of classification algorithms known as Naive Bayes classifiers. It is a family of algorithms that share a common concept, namely that each pair of features being classified is independent of the others.

**Results:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise-7**

**Objective:** Logistic Regression.

Logistic regression is popular and powerful. It uses logit transform to predict probabilities directly.

1. Launch the WEKA tool, and activate the Explorer environment.
2. Open file diabetes.arff (pima\_diabetes)(you can find this sample dataset in the folder \data).
3. Go to the Classify tab. Click on choose button. Open function folder and select Logistic.
4. Click on percentage split and change it to 80% and click start.
5. Discuss the results.

**Results:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise-9**

**Objective:** Clustering.

**Results:**

* **Step1: Open the data/ bmw-browsers.arff Dataset**

**A screenshot of a computer

AI-generated content may be incorrect.**

* **Step2: Creating the Clusters with WEKA**

**A screenshot of a computer

AI-generated content may be incorrect.**

* **Step3: Interpreting the Clusters**

A screenshot of a computer

AI-generated content may be incorrect.

* **Step4: Cluster visual inspection**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise-10**

**Objective:** K-means Clustering.

This assignment is based on analyzing clustering techniques. The objective of this assignment is to become familiar with **clustering algorithm** and evaluate or compare their performance using *Weka*.

1. Select the “**segment-test.arff**” dataset in Weka and run the Simple K-means algorithm on it using 2, 4, 8, and 16 clusters with 2 different distance functions: ***EuclideanDistance*** and ***ManhattanDistance*** respectively with a 44% percentage split (in total you would need to run it 8 times – 4 clusters with each distance function). You can keep the default values of the remaining parameters like maximum number of iterations, random seed, etc.
   1. How many clusters do you get?
   2. Visualize the clusters (graph) and take a screenshot of your results.
2. In the Explorer application, open the **cpu.arff** data file and do the following:
   1. Cluster the data (using the simple k-Means algorithm, with **k=3**) and report on the nature and composition of the extracted clusters.
3. Load the ‘weather.arff’ data set and click on the ‘Cluster’ tab. Choose the

‘SimpleKMeans’ classifier with the default options. Under the ‘Cluster mode’ choose ‘**Classes to clusters evaluation’** to evaluate the clustering performance using the class labels provided in the data file. Click the ‘Start’ button.

* 1. Explain the purpose of the menu under the option ‘Classes to clusters evaluation’ in **one sentence**.
  2. What is the number of clusters?
  3. What is the number of iterations?
  4. What class label is assigned to Cluster 0?
  5. What is the percentage of correctly clustered instances ?

**Results:**

**1.2.**

A screen shot of a computer

AI-generated content may be incorrect.

**2.1.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Exercise-11**

**Objective:** Credit card Approval detection using Machine Learning through financial data.

# Credit Card Approval Detection Using Machine Learning Through Financial Data

## Context

Credit score cards are a common risk control method in the financial industry. It uses personal information and data submitted by credit card applicants to predict the probability of future defaults and credit card borrowings. The bank is able to decide whether to issue a credit card to the applicant. Credit scores can objectively quantify the magnitude of risk.

Here we have various machine learning algorithms on the given financial data to draw comparisons between their performances.

## Task

Build a machine learning model to predict if an applicant is 'good' or 'bad' client, different from other tasks, the definition of 'good' or 'bad' is not given. You should use some techique, such as vintage analysis to construct you label. Also, unbalance data problem is a big problem in this task.

## Downloading Dataset

import kagglehub  
  
# Download latest version  
path = kagglehub.dataset\_download("rikdifos/credit-card-approval-prediction")  
  
print('Dataset download complete.')

Dataset download complete.

## Imports

import warnings  
import missingno  
import numpy as np  
import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt

# Set display options to show all columns  
pd.set\_option('display.max\_columns', None) # Show all columns  
pd.set\_option('display.width', None) # Adjust width dynamically  
pd.set\_option('display.expand\_frame\_repr', False) # Prevent column wrapping  
warnings.filterwarnings('ignore') # To avoid all warnings

## Preprocessing

Firstly, we will be focusing on the application dataset.

application = pd.read\_csv(path + "/application\_record.csv")  
credit\_record = pd.read\_csv(path + "/credit\_record.csv")

print(f'application shape: {application.shape}')  
print(f'credit\_record shape: {credit\_record.shape}')

application shape: (438557, 18)  
credit\_record shape: (1048575, 3)

application

ID CODE\_GENDER FLAG\_OWN\_CAR FLAG\_OWN\_REALTY CNT\_CHILDREN AMT\_INCOME\_TOTAL NAME\_INCOME\_TYPE NAME\_EDUCATION\_TYPE NAME\_FAMILY\_STATUS NAME\_HOUSING\_TYPE DAYS\_BIRTH DAYS\_EMPLOYED FLAG\_MOBIL FLAG\_WORK\_PHONE FLAG\_PHONE FLAG\_EMAIL OCCUPATION\_TYPE CNT\_FAM\_MEMBERS  
0 5008804 M Y Y 0 427500.0 Working Higher education Civil marriage Rented apartment -12005 -4542 1 1 0 0 NaN 2.0  
1 5008805 M Y Y 0 427500.0 Working Higher education Civil marriage Rented apartment -12005 -4542 1 1 0 0 NaN 2.0  
2 5008806 M Y Y 0 112500.0 Working Secondary / secondary special Married House / apartment -21474 -1134 1 0 0 0 Security staff 2.0  
3 5008808 F N Y 0 270000.0 Commercial associate Secondary / secondary special Single / not married House / apartment -19110 -3051 1 0 1 1 Sales staff 1.0  
4 5008809 F N Y 0 270000.0 Commercial associate Secondary / secondary special Single / not married House / apartment -19110 -3051 1 0 1 1 Sales staff 1.0  
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...  
438552 6840104 M N Y 0 135000.0 Pensioner Secondary / secondary special Separated House / apartment -22717 365243 1 0 0 0 NaN 1.0  
438553 6840222 F N N 0 103500.0 Working Secondary / secondary special Single / not married House / apartment -15939 -3007 1 0 0 0 Laborers 1.0  
438554 6841878 F N N 0 54000.0 Commercial associate Higher education Single / not married With parents -8169 -372 1 1 0 0 Sales staff 1.0  
438555 6842765 F N Y 0 72000.0 Pensioner Secondary / secondary special Married House / apartment -21673 365243 1 0 0 0 NaN 2.0  
438556 6842885 F N Y 0 121500.0 Working Secondary / secondary special Married House / apartment -18858 -1201 1 0 1 0 Sales staff 2.0  
  
[438557 rows x 18 columns]

It appears that most of the null values in 'OCCUPATION\_TYPE' feature has something in common, they all appear to be pensioners in the 'NAME\_INCOME\_TYPE' feature, which is logical if you think about it, that a retired person shouldn't be working.

application.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 438557 entries, 0 to 438556  
Data columns (total 18 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 ID 438557 non-null int64   
 1 CODE\_GENDER 438557 non-null object   
 2 FLAG\_OWN\_CAR 438557 non-null object   
 3 FLAG\_OWN\_REALTY 438557 non-null object   
 4 CNT\_CHILDREN 438557 non-null int64   
 5 AMT\_INCOME\_TOTAL 438557 non-null float64  
 6 NAME\_INCOME\_TYPE 438557 non-null object   
 7 NAME\_EDUCATION\_TYPE 438557 non-null object   
 8 NAME\_FAMILY\_STATUS 438557 non-null object   
 9 NAME\_HOUSING\_TYPE 438557 non-null object   
 10 DAYS\_BIRTH 438557 non-null int64   
 11 DAYS\_EMPLOYED 438557 non-null int64   
 12 FLAG\_MOBIL 438557 non-null int64   
 13 FLAG\_WORK\_PHONE 438557 non-null int64   
 14 FLAG\_PHONE 438557 non-null int64   
 15 FLAG\_EMAIL 438557 non-null int64   
 16 OCCUPATION\_TYPE 304354 non-null object   
 17 CNT\_FAM\_MEMBERS 438557 non-null float64  
dtypes: float64(2), int64(8), object(8)  
memory usage: 60.2+ MB

* ID: Unique Id of the row
* CODE\_GENDER: Gender of the applicant. M is male and F is female.
* FLAG\_OWN\_CAR: Is an applicant with a car. Y is Yes and N is NO.
* FLAG\_OWN\_REALTY: Is an applicant with realty. Y is Yes and N is No.
* CNT\_CHILDREN: Count of children.
* AMT\_INCOME\_TOTAL: the amount of the income.
* NAME\_INCOME\_TYPE: The type of income (5 types in total).
* NAME\_EDUCATION\_TYPE: The type of education (5 types in total).
* NAME\_FAMILY\_STATUS: The type of family status (6 types in total).
* DAYS\_BIRTH: The number of the days from birth (Negative values).
* DAYS\_EMPLOYED: The number of the days from employed (Negative values). This column has error values.
* FLAG\_MOBIL: Is an applicant with a mobile. 1 is True and 0 is False.
* FLAG\_WORK\_PHONE: Is an applicant with a work phone. 1 is True and 0 is False.
* FLAG\_PHONE: Is an applicant with a phone. 1 is True and 0 is False.
* FLAG\_EMAIL: Is an applicant with a email. 1 is True and 0 is False.
* OCCUPATION\_TYPE: The type of occupation (19 types in total). This column has missing values.
* CNT\_FAM\_MEMBERS: The count of family members.

### 1. Duplicates

application.duplicated(subset='ID').value\_counts()

False 438510  
True 47  
Name: count, dtype: int64

It appears that the application dataset has some duplicates, so we will be dropping them.

application.drop\_duplicates(subset='ID', inplace=True)

### 2. Unique Values

features = application.select\_dtypes(include='object').columns.tolist()  
  
for feature in features:  
 print(f'{feature}: {application[feature].nunique()}')  
 print(f'{application[feature].unique()}')

CODE\_GENDER: 2  
['M' 'F']  
FLAG\_OWN\_CAR: 2  
['Y' 'N']  
FLAG\_OWN\_REALTY: 2  
['Y' 'N']  
NAME\_INCOME\_TYPE: 5  
['Working' 'Commercial associate' 'Pensioner' 'State servant' 'Student']  
NAME\_EDUCATION\_TYPE: 5  
['Higher education' 'Secondary / secondary special' 'Incomplete higher'  
 'Lower secondary' 'Academic degree']  
NAME\_FAMILY\_STATUS: 5  
['Civil marriage' 'Married' 'Single / not married' 'Separated' 'Widow']  
NAME\_HOUSING\_TYPE: 6  
['Rented apartment' 'House / apartment' 'Municipal apartment'  
 'With parents' 'Co-op apartment' 'Office apartment']  
OCCUPATION\_TYPE: 18  
[nan 'Security staff' 'Sales staff' 'Accountants' 'Laborers' 'Managers'  
 'Drivers' 'Core staff' 'High skill tech staff' 'Cleaning staff'  
 'Private service staff' 'Cooking staff' 'Low-skill Laborers'  
 'Medicine staff' 'Secretaries' 'Waiters/barmen staff' 'HR staff'  
 'Realty agents' 'IT staff']

application.hist(figsize=(20,20))

array([[<Axes: title={'center': 'ID'}>,  
 <Axes: title={'center': 'CNT\_CHILDREN'}>,  
 <Axes: title={'center': 'AMT\_INCOME\_TOTAL'}>],  
 [<Axes: title={'center': 'DAYS\_BIRTH'}>,  
 <Axes: title={'center': 'DAYS\_EMPLOYED'}>,  
 <Axes: title={'center': 'FLAG\_MOBIL'}>],  
 [<Axes: title={'center': 'FLAG\_WORK\_PHONE'}>,  
 <Axes: title={'center': 'FLAG\_PHONE'}>,  
 <Axes: title={'center': 'FLAG\_EMAIL'}>],  
 [<Axes: title={'center': 'CNT\_FAM\_MEMBERS'}>, <Axes: >, <Axes: >]],  
 dtype=object)

A group of blue and white graphs

AI-generated content may be incorrect.

### 3. Null values

Like I stated before most of the null values are pensioners, so we will be further exploring that concept.

application.isnull().sum()

ID 0  
CODE\_GENDER 0  
FLAG\_OWN\_CAR 0  
FLAG\_OWN\_REALTY 0  
CNT\_CHILDREN 0  
AMT\_INCOME\_TOTAL 0  
NAME\_INCOME\_TYPE 0  
NAME\_EDUCATION\_TYPE 0  
NAME\_FAMILY\_STATUS 0  
NAME\_HOUSING\_TYPE 0  
DAYS\_BIRTH 0  
DAYS\_EMPLOYED 0  
FLAG\_MOBIL 0  
FLAG\_WORK\_PHONE 0  
FLAG\_PHONE 0  
FLAG\_EMAIL 0  
OCCUPATION\_TYPE 134193  
CNT\_FAM\_MEMBERS 0  
dtype: int64

missingno.bar(application)

<Axes: >

A grey and white vertical lines

AI-generated content may be incorrect.

print((application['OCCUPATION\_TYPE'].isnull().sum() / application.shape[0]) \* 100 ,'%')

30.602038722035985 %

It appears that nearly a third of 'OCCUPATION\_TYPE' feature are nulls.

pensioners = application['OCCUPATION\_TYPE'][application['NAME\_INCOME\_TYPE'] == 'Pensioner']  
others = application['OCCUPATION\_TYPE'][application['NAME\_INCOME\_TYPE'] != 'Pensioner']  
  
print(f"Percentage of nulls that are pensioners: {(pensioners.isnull().sum() / application['OCCUPATION\_TYPE'].isnull().sum()) \* 100} %")  
print(f"Percentage of nulls that are NOT pensioners: {(others.isnull().sum() / application['OCCUPATION\_TYPE'].isnull().sum()) \* 100} %")

Percentage of nulls that are pensioners: 56.15196023637596 %  
Percentage of nulls that are NOT pensioners: 43.84803976362404 %

As we see that half of the null values are pensioners and the other 4 types are less than 50% .

According to the context of the data, there are 2 features that decides the 'OCCUPATION\_TYPE' for the clients and they are 'NAME\_INCOME\_TYPE', 'NAME\_EDUCATION\_TYPE'.

So we will be checking the mode for every type of job and their academic degree.

df = pd.DataFrame(application[['NAME\_INCOME\_TYPE', 'NAME\_EDUCATION\_TYPE', 'OCCUPATION\_TYPE']])  
df.set\_index(['NAME\_INCOME\_TYPE', 'NAME\_EDUCATION\_TYPE'], inplace=True)  
  
df.dropna(inplace=True)

df

OCCUPATION\_TYPE  
NAME\_INCOME\_TYPE NAME\_EDUCATION\_TYPE   
Working Secondary / secondary special Security staff  
Commercial associate Secondary / secondary special Sales staff  
 Secondary / secondary special Sales staff  
 Secondary / secondary special Sales staff  
 Secondary / secondary special Sales staff  
... ...  
Working Higher education Laborers  
 Secondary / secondary special Laborers  
 Secondary / secondary special Laborers  
Commercial associate Higher education Sales staff  
Working Secondary / secondary special Sales staff  
  
[304317 rows x 1 columns]

for job in application['NAME\_INCOME\_TYPE'].unique():  
 for edu in application['NAME\_EDUCATION\_TYPE'].unique():  
 if job == 'Student' and edu not in ['Higher education', 'Secondary / secondary special']:  
 continue  
 if job == 'Pensioner' and edu not in ['Higher education', 'Secondary / secondary special', 'Incomplete higher']:  
 continue  
 print(f"{job}, {edu}: {df.loc[job, edu]['OCCUPATION\_TYPE'].mode()[0]}")

Working, Higher education: Core staff  
Working, Secondary / secondary special: Laborers  
Working, Incomplete higher: Laborers

Working, Lower secondary: Laborers  
Working, Academic degree: Core staff  
Commercial associate, Higher education: Managers  
Commercial associate, Secondary / secondary special: Laborers  
Commercial associate, Incomplete higher: Managers  
Commercial associate, Lower secondary: Laborers  
Commercial associate, Academic degree: Sales staff  
Pensioner, Higher education: Core staff  
Pensioner, Secondary / secondary special: Laborers  
Pensioner, Incomplete higher: High skill tech staff  
State servant, Higher education: Core staff  
State servant, Secondary / secondary special: Core staff  
State servant, Incomplete higher: Core staff  
State servant, Lower secondary: Medicine staff  
State servant, Academic degree: Managers  
Student, Higher education: Core staff  
Student, Secondary / secondary special: Laborers

So there are the modes for their respective position and academic degree.

We will imputate the data accordingly.

Here is the percentage of nulls again before imputating any data.

print(f"{(application['OCCUPATION\_TYPE'].isnull().sum() / application.shape[0]) \* 100} %")

30.602038722035985 %

Now onto Data Imputation.

#### -> Data Imputation

# Working  
mask = (application['NAME\_INCOME\_TYPE'] == 'Working') & (application['NAME\_EDUCATION\_TYPE'].isin(['Higher education', 'Academic degree']))  
application.loc[mask, 'OCCUPATION\_TYPE'] = application.loc[mask, 'OCCUPATION\_TYPE'].fillna('Core staff')  
  
mask = (application['NAME\_INCOME\_TYPE'] == 'Working') & (application['NAME\_EDUCATION\_TYPE'].isin(['Secondary / secondary special', 'Incomplete higher', 'Lower secondary']))  
application.loc[mask, 'OCCUPATION\_TYPE'] = application.loc[mask, 'OCCUPATION\_TYPE'].fillna('Laborers')

# Commercial associate  
mask = (application['NAME\_INCOME\_TYPE'] == 'Commercial associate') & (application['NAME\_EDUCATION\_TYPE'].isin(['Higher education', 'Incomplete higher']))  
application.loc[mask, 'OCCUPATION\_TYPE'] = application.loc[mask, 'OCCUPATION\_TYPE'].fillna('Managers')  
  
mask = (application['NAME\_INCOME\_TYPE'] == 'Commercial associate') & (application['NAME\_EDUCATION\_TYPE'].isin(['Secondary / secondary special', 'Lower secondary']))  
application.loc[mask, 'OCCUPATION\_TYPE'] = application.loc[mask, 'OCCUPATION\_TYPE'].fillna('Laborers')  
  
mask = (application['NAME\_INCOME\_TYPE'] == 'Commercial associate') & (application['NAME\_EDUCATION\_TYPE'].isin(['Academic degree']))  
application.loc[mask, 'OCCUPATION\_TYPE'] = application.loc[mask, 'OCCUPATION\_TYPE'].fillna('Sales staff')

# State servant  
mask = (application['NAME\_INCOME\_TYPE'] == 'State servant') & (application['NAME\_EDUCATION\_TYPE'].isin(['Higher education', 'Secondary / secondary special', 'Incomplete higher']))  
application.loc[mask, 'OCCUPATION\_TYPE'] = application.loc[mask, 'OCCUPATION\_TYPE'].fillna('Core staff')  
  
mask = (application['NAME\_INCOME\_TYPE'] == 'State servant') & (application['NAME\_EDUCATION\_TYPE'].isin(['Lower secondary']))  
application.loc[mask, 'OCCUPATION\_TYPE'] = application.loc[mask, 'OCCUPATION\_TYPE'].fillna('Medicine staff')  
  
mask = (application['NAME\_INCOME\_TYPE'] == 'State servant') & (application['NAME\_EDUCATION\_TYPE'].isin(['Academic degree']))  
application.loc[mask, 'OCCUPATION\_TYPE'] = application.loc[mask, 'OCCUPATION\_TYPE'].fillna('Managers')

# Pensioner  
mask = (application['NAME\_INCOME\_TYPE'] == 'Pensioner') & (application['NAME\_EDUCATION\_TYPE'].isin(['Higher education']))  
application.loc[mask, 'OCCUPATION\_TYPE'] = application.loc[mask, 'OCCUPATION\_TYPE'].fillna('Core staff')  
  
mask = (application['NAME\_INCOME\_TYPE'] == 'Pensioner') & (application['NAME\_EDUCATION\_TYPE'].isin(['Secondary / secondary special']))  
application.loc[mask, 'OCCUPATION\_TYPE'] = application.loc[mask, 'OCCUPATION\_TYPE'].fillna('Laborers')  
  
mask = (application['NAME\_INCOME\_TYPE'] == 'Pensioner') & (application['NAME\_EDUCATION\_TYPE'].isin(['Incomplete higher']))  
application.loc[mask, 'OCCUPATION\_TYPE'] = application.loc[mask, 'OCCUPATION\_TYPE'].fillna('High skill tech staff')

# Student  
mask = (application['NAME\_INCOME\_TYPE'] == 'Student') & (application['NAME\_EDUCATION\_TYPE'].isin(['Higher education']))  
application.loc[mask, 'OCCUPATION\_TYPE'] = application.loc[mask, 'OCCUPATION\_TYPE'].fillna('Core staff')  
  
mask = (application['NAME\_INCOME\_TYPE'] == 'Student') & (application['NAME\_EDUCATION\_TYPE'].isin(['Secondary / secondary special']))  
application.loc[mask, 'OCCUPATION\_TYPE'] = application.loc[mask, 'OCCUPATION\_TYPE'].fillna('Laborers')

Now to check the percentage of nulls.

print(f"{(application['OCCUPATION\_TYPE'].isnull().sum() / application.shape[0]) \* 100} %")

0.368520672276573 %

There is still a small percentage of nulls, which we will fill using the mode of the entire feature.

application['OCCUPATION\_TYPE'].fillna(application['OCCUPATION\_TYPE'].mode()[0], inplace=True)  
  
print(f"{(application['OCCUPATION\_TYPE'].isnull().sum() / application.shape[0]) \* 100} %")

0.0 %

There are no more nulls.

### 4. Outliers

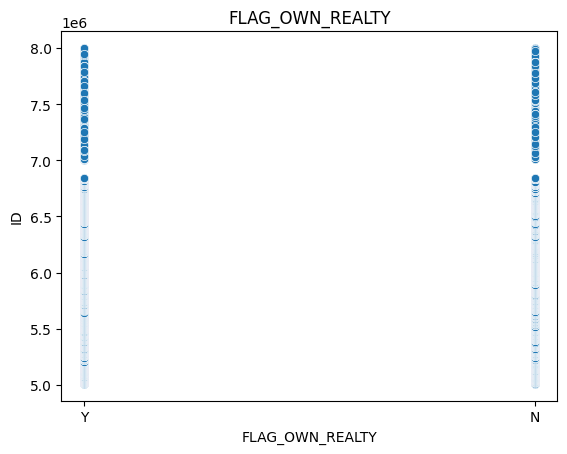
features = application.columns.tolist()  
  
for feature in features:  
 if feature == 'ID':  
 continue  
 sns.scatterplot(data=application, x=feature, y='ID')  
 plt.title(feature)  
 plt.show()

A graph with numbers and lines

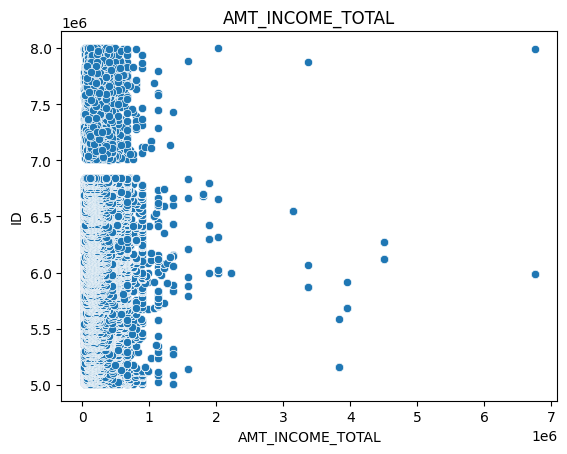
AI-generated content may be incorrect.

A graph of a flag car

AI-generated content may be incorrect.



A graph of blue dots

AI-generated content may be incorrect.

A graph of numbers and numbers

AI-generated content may be incorrect.

A group of blue and white lines

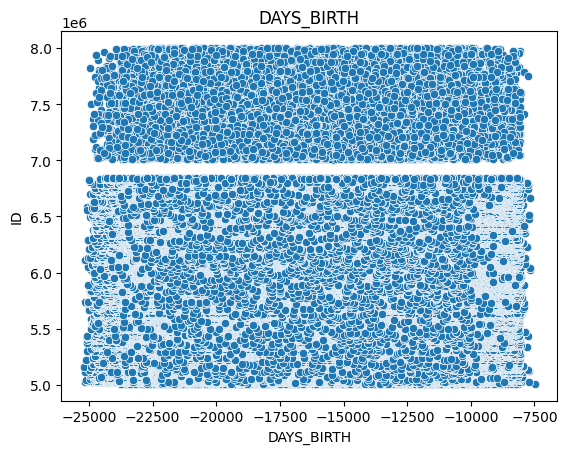
AI-generated content may be incorrect.

A graph of a family status

AI-generated content may be incorrect.

A blue and white chart

AI-generated content may be incorrect.



A graph with numbers and a number of days

AI-generated content may be incorrect.

A graph of a flag

AI-generated content may be incorrect.

A graph of a phone

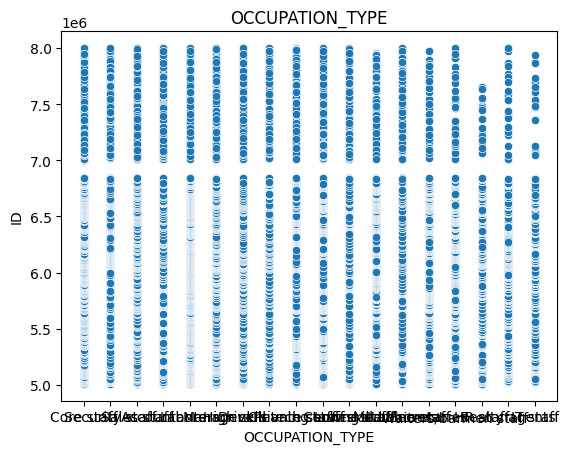
AI-generated content may be incorrect.

A graph of a number of objects

AI-generated content may be incorrect.

A graph with numbers and lines

AI-generated content may be incorrect.



A graph of blue dots

AI-generated content may be incorrect.

It appears there are outliers in the following columns:

* CNT\_CHILDREN
* AMT\_INCOME\_TOTAL
* CNT\_FAM\_MEMBERS

we will remove those outliers.

# CNT\_CHILDREN  
q\_hi = application['CNT\_CHILDREN'].quantile(0.999)  
q\_low = application['CNT\_CHILDREN'].quantile(0.001)  
application = application[(application['CNT\_CHILDREN'] > q\_low) & (application['CNT\_CHILDREN'] < q\_hi)]

# AMT\_INCOME\_TOTAL  
q\_hi = application['AMT\_INCOME\_TOTAL'].quantile(0.999)  
q\_low = application['AMT\_INCOME\_TOTAL'].quantile(0.001)  
application = application[(application['AMT\_INCOME\_TOTAL'] > q\_low) & (application['AMT\_INCOME\_TOTAL'] < q\_hi)]

# CNT\_FAM\_MEMBERS  
q\_hi = application['CNT\_FAM\_MEMBERS'].quantile(0.999)  
q\_low = application['CNT\_FAM\_MEMBERS'].quantile(0.001)  
application = application[(application['CNT\_FAM\_MEMBERS'] > q\_low) & (application['CNT\_FAM\_MEMBERS'] < q\_hi)]

### 5. Encoding

Encoding Categorical features.

from sklearn.preprocessing import LabelEncoder  
  
le = LabelEncoder()  
  
object\_features = application.select\_dtypes(include='object').columns.tolist()  
object\_features

['CODE\_GENDER',  
 'FLAG\_OWN\_CAR',  
 'FLAG\_OWN\_REALTY',  
 'NAME\_INCOME\_TYPE',  
 'NAME\_EDUCATION\_TYPE',  
 'NAME\_FAMILY\_STATUS',  
 'NAME\_HOUSING\_TYPE',  
 'OCCUPATION\_TYPE']

for feature in object\_features:  
 application[feature] = le.fit\_transform(application[feature])

application

ID CODE\_GENDER FLAG\_OWN\_CAR FLAG\_OWN\_REALTY CNT\_CHILDREN AMT\_INCOME\_TOTAL NAME\_INCOME\_TYPE NAME\_EDUCATION\_TYPE NAME\_FAMILY\_STATUS NAME\_HOUSING\_TYPE DAYS\_BIRTH DAYS\_EMPLOYED FLAG\_MOBIL FLAG\_WORK\_PHONE FLAG\_PHONE FLAG\_EMAIL OCCUPATION\_TYPE CNT\_FAM\_MEMBERS  
29 5008838 1 0 1 1 405000.0 0 1 1 1 -11842 -2016 1 0 0 0 10 3.0  
30 5008839 1 0 1 1 405000.0 0 1 1 1 -11842 -2016 1 0 0 0 10 3.0  
31 5008840 1 0 1 1 405000.0 0 1 1 1 -11842 -2016 1 0 0 0 10 3.0  
32 5008841 1 0 1 1 405000.0 0 1 1 1 -11842 -2016 1 0 0 0 10 3.0  
33 5008842 1 0 1 1 405000.0 0 1 1 1 -11842 -2016 1 0 0 0 10 3.0  
... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...  
438536 6837264 0 0 0 2 90000.0 2 1 3 1 -16062 -1275 1 0 0 0 3 4.0  
438539 6837454 1 1 1 1 162000.0 2 4 1 1 -10890 -2675 1 0 0 0 3 3.0  
438542 6837905 1 1 1 1 355050.0 4 4 1 1 -15904 -2614 1 0 0 0 8 3.0  
438543 6837906 1 1 1 1 355050.0 4 4 1 1 -15904 -2614 1 0 0 0 8 3.0  
438548 6839936 1 1 1 1 135000.0 4 4 1 1 -12569 -2095 1 0 0 0 8 3.0  
  
[114188 rows x 18 columns]

### -> Credit Dataset

* 0: 1-29 days past due
* 1: 30-59 days past due
* 2: 60-89 days overdue
* 3: 90-119 days overdue
* 4: 120-149 days overdue
* 5: Overdue or bad debts, write-offs for more than 150 days
* C: paid off that month
* X: No loan for the month

credit\_record

ID MONTHS\_BALANCE STATUS  
0 5001711 0 X  
1 5001711 -1 0  
2 5001711 -2 0  
3 5001711 -3 0  
4 5001712 0 C  
... ... ... ...  
1048570 5150487 -25 C  
1048571 5150487 -26 C  
1048572 5150487 -27 C  
1048573 5150487 -28 C  
1048574 5150487 -29 C  
  
[1048575 rows x 3 columns]

It appears that the credit record dataset are set in a format which need to be grouped by the 'ID'.

The approach is as of following:

* Each 'ID' keeps the latest month (max(MONTHS\_BALANCE)).
* Each 'ID' keeps the worst (highest) status after transformation.
* If any month had 'X' or 'C' (converted to 1), the final status is 1 (Good Client).
* If the user had only overdue payments, the final status is 0 (Bad Client).

Basically, we are hunting the bad clients, if 'STATUS' >= 2 represents serious overdue payments, this transformation marks those customers with 1 (high risk). If 'STATUS' < 2 means acceptable risk or good clients, they are marked with 0.

credit\_record['STATUS'].replace({'C': 0, 'X' : 0}, inplace=True)  
credit\_record['STATUS'] = credit\_record['STATUS'].astype('int')  
credit\_record['STATUS'] = credit\_record['STATUS'].apply(lambda x:1 if x >= 2 else 0)

credit\_record = credit\_record.groupby('ID').agg(max).reset\_index()  
  
credit\_record.drop('MONTHS\_BALANCE', axis=1, inplace=True)

credit\_record.head()

ID STATUS  
0 5001711 0  
1 5001712 0  
2 5001713 0  
3 5001714 0  
4 5001715 0

Now it's fixed.

It appears that a new problem arises and that is the imbalance of target value samples.

credit\_record['STATUS'].value\_counts(normalize=True)

STATUS  
0 0.985495  
1 0.014505  
Name: proportion, dtype: float64

We will address this problem in the Model Building section.

### 6. Merging

df = pd.merge(application, credit\_record, on='ID', how='inner')

df.head()

ID CODE\_GENDER FLAG\_OWN\_CAR FLAG\_OWN\_REALTY CNT\_CHILDREN AMT\_INCOME\_TOTAL NAME\_INCOME\_TYPE NAME\_EDUCATION\_TYPE NAME\_FAMILY\_STATUS NAME\_HOUSING\_TYPE DAYS\_BIRTH DAYS\_EMPLOYED FLAG\_MOBIL FLAG\_WORK\_PHONE FLAG\_PHONE FLAG\_EMAIL OCCUPATION\_TYPE CNT\_FAM\_MEMBERS STATUS  
0 5008838 1 0 1 1 405000.0 0 1 1 1 -11842 -2016 1 0 0 0 10 3.0 0  
1 5008839 1 0 1 1 405000.0 0 1 1 1 -11842 -2016 1 0 0 0 10 3.0 0  
2 5008840 1 0 1 1 405000.0 0 1 1 1 -11842 -2016 1 0 0 0 10 3.0 0  
3 5008841 1 0 1 1 405000.0 0 1 1 1 -11842 -2016 1 0 0 0 10 3.0 0  
4 5008842 1 0 1 1 405000.0 0 1 1 1 -11842 -2016 1 0 0 0 10 3.0 0

df.shape

(9516, 19)

There are 9516 rows ready for deployment.

## Model Building

Splitting the data into X & y and further into train & test.

X = df.iloc[:,1:-1]  
y = df.iloc[:,-1]

from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.3)

Scaling the training data.

from sklearn.preprocessing import MinMaxScaler  
  
scaler = MinMaxScaler()  
  
X\_train = scaler.fit\_transform(X\_train)  
X\_test = scaler.transform(X\_test)

Addressing the oversampling problem by evening the target samples using SMOTE.

from imblearn.over\_sampling import SMOTE  
oversample = SMOTE()  
X\_train, y\_train = oversample.fit\_resample(X\_train, y\_train)

#### Model Building:

##### 1. Decision Tree

from sklearn.tree import DecisionTreeClassifier  
  
clf\_entropy = DecisionTreeClassifier(  
 criterion="entropy", random\_state=100,  
 max\_depth=3, min\_samples\_leaf=5)  
model\_entropy = clf\_entropy.fit(X\_train, y\_train)  
prediction = model\_entropy.predict(X\_test)

from sklearn.metrics import classification\_report, accuracy\_score  
print(f"Accuracy: {accuracy\_score(y\_test, prediction) \* 100:.2f}%")  
print(classification\_report(y\_test, prediction))

Accuracy: 49.81%  
 precision recall f1-score support  
  
 0 0.98 0.50 0.66 2809  
 1 0.02 0.50 0.03 46  
  
 accuracy 0.50 2855  
 macro avg 0.50 0.50 0.35 2855  
weighted avg 0.97 0.50 0.65 2855

##### 2. Random Forest

from sklearn.ensemble import RandomForestClassifier  
clf = RandomForestClassifier(  
 n\_estimators=100, random\_state=100,  
 max\_depth=3, min\_samples\_leaf=5)  
model = clf.fit(X\_train, y\_train)  
prediction = model.predict(X\_test)

print(f"Accuracy: {accuracy\_score(y\_test, prediction) \* 100:.2f}%")  
print(classification\_report(y\_test, prediction))

Accuracy: 80.11%  
 precision recall f1-score support  
  
 0 0.99 0.81 0.89 2809  
 1 0.03 0.37 0.06 46  
  
 accuracy 0.80 2855  
 macro avg 0.51 0.59 0.47 2855  
weighted avg 0.97 0.80 0.88 2855

##### 3. KNN

from sklearn.neighbors import KNeighborsClassifier  
knn = KNeighborsClassifier(n\_neighbors=5)  
knn.fit(X\_train, y\_train)  
prediction = knn.predict(X\_test)

print(f"Accuracy: {accuracy\_score(y\_test, prediction) \* 100:.2f}%")  
print(classification\_report(y\_test, prediction))

Accuracy: 96.60%  
 precision recall f1-score support  
  
 0 0.99 0.97 0.98 2809  
 1 0.23 0.46 0.30 46  
  
 accuracy 0.97 2855  
 macro avg 0.61 0.72 0.64 2855  
weighted avg 0.98 0.97 0.97 2855

##### 4. XgBoost

from xgboost import XGBClassifier  
xgb = XGBClassifier()  
model = xgb.fit(X\_train, y\_train)  
prediction = xgb.predict(X\_test)

print(f"Accuracy: {accuracy\_score(y\_test, prediction) \* 100:.2f}%")  
print(classification\_report(y\_test, prediction))

Accuracy: 98.21%  
 precision recall f1-score support  
  
 0 0.99 0.99 0.99 2809  
 1 0.42 0.28 0.34 46  
  
 accuracy 0.98 2855  
 macro avg 0.70 0.64 0.66 2855  
weighted avg 0.98 0.98 0.98 2855

##### 5. SVM

from sklearn.svm import SVC  
svm = SVC(kernel='linear', C=1, random\_state=100)  
svm.fit(X\_train, y\_train)  
prediction = svm.predict(X\_test)

print(f"Accuracy: {accuracy\_score(y\_test, prediction) \* 100:.2f}%")  
print(classification\_report(y\_test, prediction))

Accuracy: 62.31%  
 precision recall f1-score support  
  
 0 0.99 0.62 0.77 2809  
 1 0.02 0.54 0.04 46  
  
 accuracy 0.62 2855  
 macro avg 0.51 0.58 0.40 2855  
weighted avg 0.97 0.62 0.75 2855

##### 6. Neural Network

from sklearn.neural\_network import MLPClassifier  
mlp = MLPClassifier(hidden\_layer\_sizes=(100, 50), max\_iter=1000, random\_state=100)  
mlp.fit(X\_train, y\_train)  
prediction = mlp.predict(X\_test)

print(f"Accuracy: {accuracy\_score(y\_test, prediction) \* 100:.2f}%")  
print(classification\_report(y\_test, prediction))

Accuracy: 96.71%  
 precision recall f1-score support  
  
 0 0.99 0.98 0.98 2809  
 1 0.22 0.41 0.29 46  
  
 accuracy 0.97 2855  
 macro avg 0.61 0.69 0.64 2855  
weighted avg 0.98 0.97 0.97 2855