Machine Learning implementation:

**Categorical and numeric variables:**

Data types are divided into two broad groups which are categorical and numeric or continuous variables.

Categorical variables usually have a finite number of categories or distinct groups. Categorical variables do not typically have a logical order. Examples of categorical data are gender, marital type, payment method etc. In machine learning, categorical variables are usually converted to numeric variables as most machine leaning models require that all the data be represented in numeric values. This process is known as categorical encoding. E.g., gender category which is mainly male, and female can be represented as 0 and 1 respectively.

Continuous or numeric variables have an infinite number of values between any two values. A continuous variable can be numeric such as numbers or other data types such as dates. Examples of continuous variables are the weight of an object or the time a payment is made.

**Categorical encoding: Label Encoding and One hot encoding:**

Since machines only understand numbers and not texts, all categorical variables need to be converted to numbers for the machine to process them using mathematical equations. There are two major ways to do this in python which are label encoding and one hot encoding.

**Label Encoding:** Label encoding is the most popular encoding technique and often the most misused technique. In this technique, each label is assigned a unique integer based on alphabetical ordering. This technique is available in the scikit learn library available in python and the easiest to implement. The disadvantage of encoding this way however is that machine learning models can regard the numbers as hierarchical hence this technique is not suitable for features with more than two categories. E.g., True or False categories.

using label encoding on a dataset of countries such as India, Japan and USA will result in a new column with 0, 1, 2. 2 is greater than 1 and 1 is greater than 0. This is not the performance we will want from our dataset, hence for columns that have more than two categories, some of the categories will be assumed to carry a higher significance based on the values they have been encoded to. Since the encoding is mostly done alphabetically, this will affect the performance of the model.

**One hot encoding:** One hot encoding is used to overcome the challenge that is associated with label encoding. It creates additional features based on the number of unique values in the categorical column or feature. It creates dummy variables to represent the different categories in the column. Using one hot encoding on our country dataset will result in the table below.

|  |  |  |
| --- | --- | --- |
| India | Japan | USA |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

Encoding this way resolves the problem of hierarchy that is often associated with label encoding.

Although, one of the challenges with one encoding is should features or columns have many distinct/unique values, this can make it difficult to train a model appropriately. A work around this is to collapse some of the categories.

**Absolute Error:**

Absolute error is the absolute value of the difference between an observed value and a true value. For example, if an item weighs 30kg, but an observer records 29kg, the absolute error in this case will be the 1kg difference between 30 and 29, represented as |30kg - 29kg|. Since we are dealing with absolute values, we only take the positive difference between the two values.

**Mean Absolute Error (MEA):**

Mean absolute error refers to the magnitude of difference between the prediction of an observation and the true value of that observation. In simple terms, mean absolute error takes the average absolute error from a group of predictions and observations as a measure for the magnitude of error of the entire group.

The equation for mean absolute error is given below:

Graphical user interface

Description automatically generated with low confidence

The equation above shows that mean absolute error is calculated by getting the mean of the absolute difference between the true value and the observed or estimated or predicted value.

In regression and categorical problems, mean absolute error helps makes it easy understand and quantify the degree/measurement of errors of a machine learning model.

The sklearn library in python allow users to directly find the mean absolute errors of predictions by importing the function like this:

from sklearn.metrics import mean\_absolute\_error

**Accuracy Score:**

In machine learning, classification accuracy also known as accuracy score is the number of correct predictions made as a ratio of all the predictions made. If 3 correct predictions were made from a total of 10 predictions, the accuracy score will be 0.3 which is equivalent to 30%. The values are usually produced in decimals and are converted to percentage by multiplying by 100.

The sklearn library has the accuracy score function and it can be imported with the simple code line below;

from sklearn.metrics import accuracy\_score

**Train Test Split:**

The train test split is a technique used to divide datasets into two subsets. It involves taking the dataset into the function and the specifying the ratio of split e.g. 80% by 20% or 70% by 30%, the first subset of the datasets (80% or 70%) are then used to train the data, while the second subset (20% or 30%) are used to test the data.

The function is also available in sklearn and can be imported by simply using the code line below:

from sklearn.model\_selection import train\_test\_split

The default behavior is to split 80% of the data as the training dataset and 20% as the test dataset.

**Model Hyperparameters:**

Part of the process that is involved in training a model is choosing the optimal hyperparameters that the machine learning algorithm or model will use to learn the optimal parameters that correctly map the input features i.e., independent variables to the independent variable. Model hyperparameters are not usually known before hand and are usually determined by a form of trial and error to select the value(s) that produce the based outcome based on the performance of the model.

These values are used to train the models and the values are usually determined before you begin to train the model. The values are not determined by the data and there are no specific order or rules to follow when selecting them.

An example of an hyperparameters is the k value in KNN model (also known as the n\_neighbors).

**Data Preparation and exploration:**

To train the machine learning model, I needed to have a complete understanding of the dataset as well as prepare the datasets i.e., clean the data, format the data to make it consistent, create new features out of existing ones and check the data quality. Some of the things that I needed to consider for this project to ensure my data was good enough to train a model are:

**Exploring the datasets:** After reading in my datasets using pandas, I decided to explore the datasets to check the data layout and get a better understanding of the dataset. I read the data description document which can be found at (input link) and tried to understand all the features that are in the data sets. From here I could see the data types of all the columns as well as the number of non-null values that are in each of the columns.

Text

Description automatically generated

I went further to explore the data to check the number of unique values that are in the dataset.

Text

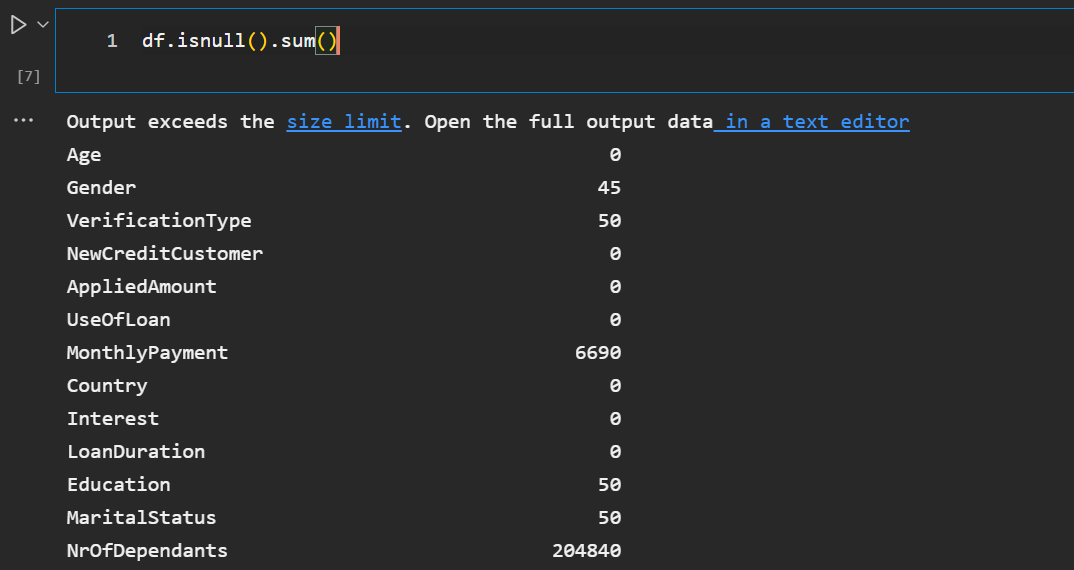
Description automatically generated

Comparing this to the data description, I concluded that some of the columns are not required to build a good model, columns such as loanid, loannumber and other date columns were assumed to not be needed in building a model, hence, I only selected the columns that I was sure will be helpful in building the model.

Graphical user interface, text

Description automatically generated

I went further to check the number of null values that are in each column of the dataset. I needed to do this to determine the best way to handle the null values.



By examining the null values, I decided to drop some of the columns that have a very large number of missing values. Columns such as NrOfDependants and WorkExperience had over 80% of the column data missing.

Text

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

I then dropped the columns that had over 30% of their values missing:

Text

Description automatically generated

By carefully examining the data description, I realized that:

For loans that are late, they cannot be classified as defaulted because although the repayment is just late and not defaulted. A loan becomes defaulted when the borrower fails to make payment 60 days after the grace period (according to [link](https://ccr.equifax.com.au/resources/explaining-the-difference-between-a-late-payment-and-a-default-to-your-customers#:~:text=Repayment%20history%20information%20is%20recorded,was%20in%20a%20specific%20month.&text=A%20payment%20later%20than%2060%20days%20is%20a%20default.)) Since this date can only be defined by the loan company, this might be difficult to achieve. However, the loan company has provided us with a Defaultdate column which is the date the loan went into default state.

This means for rows or borrowers that have a default date, the borrower is assumed to have defaulted the payment of the loan. This means that the default date is what will be used to determine whether a borrower has defaulted a loan or not. This also means that the defaultdate column cannot be dropped yet.

So, I created the default column based on the rows that have a default date.

Text

Description automatically generated

Rows without a default date were filled with empty string, and then I checked all the rows to see the values in each row, if the value is an empty string, the individual has not defaulted, hence he is represented with 0, while for those that have a valid default date, it means they have defaulted, and they are represented with 1.

Graphical user interface, application

Description automatically generated

I went further to make the following assumptions:

Since the status column for a new applicant cannot be known, it will be dropped. Also, since the default date cannot be known and it has been used to determine if the applicant defaulted or not, it will also be dropped.

Text

Description automatically generated

Once again, I checked the number of missing values in the remaining dataset and

Text

Description automatically generated

For the remaining columns, I decided to use the following approach for the null values:

* Looking at the data, I will be dropping columns that the monthly payment is unknown, because that will be difficult to infer as this is heavily dependent on a lot of factors (e.g., amount the person is borrowing, what he wants it for, his salary etc.)
* For the null values in gender, I will assign all the null values to unknown which is 2 from the data description.
* For verification type, all null values will become not set which is 0 from the data description.
* For education, I will replace all the educations with the modal (highest occurrence value) value in that column.
* I will also replace the missing marital status values with the modal value of that column
* I will replace the missing employment status values with the modal values.
* I will replace the missing EmploymentDurationCurrentEmployer values with the modal values in that column.
* I will replace missing occupationArea with other which is 0.
* I will replace missing Homeownership type with “other” which is 10 from the data description.
* I will replace missing debtToIncome values with the mean value of that column.
* For the NoOfPreviousLoansBeforeLoan, AmountOfPreviousLoansBeforeLoan and PreviousEarlyRepaymentsCountBeforeLoan I will replace their values with the modal values in their columns.

This way, I will be losing about 6690 rows of the data, only from the monthly payment column. Some of the other columns may not need to be dropped as some of the data may be in the same row that is being dropped once the rows with empty datasets are dropped from the monthly payment column.

Text

Description automatically generated

Once I dropped the missing values in the monthly payment column alone, I was left with only four columns with missing values which were; EmploymentDurationCurrentEmployer, NoOfPreviousLoansBeforeLoan, AmountOfPreviousLoansBeforeLoan and PreviousEarlyRepaymentsCountBeforeLoan.

Text

Description automatically generated

**Categorical Encoding:**

I needed to convert all the categorical variables to features. Looking at the datasets, data like gender, verificationtype, newcreditcustomer etc, are categorical variables as they have meanings beyond being just numbers.

I used one hot encoding instead of label encoding to convert all the categorical variables into numerical variable. Although label encoding is more direct and easier to implement, I recognized that it could throw off the model as it assigns values to numbers based on their alphabetical order, the model can therefore assume that 4 is greater than 1 and make a prediction based on that.

One hot encoding, however, creates additional features based on the number of unique values in the categorical feature. It has the disadvantage of adding more columns to the dataset. Every unique value in the category will be added as a feature.

The categorical variables that were identified by checking the data description are:

* Gender
* VerificationType
* NewCreditCustomer
* UseOfLoan
* Country
* LoanDuration
* Education
* MaritalStatus
* EmploymentStatus
* EmploymentDurationCurrentEmployer
* OccupationArea
* HomeOwnershipType

The code I used to implement this can be found in the github repository. I dropped the original columns once the new features had been created.

The default column was left alone since it only involves two variables 1 and 0. Like what's expected from label encoding. Using label encoding this way is fine.

**Defining the dependent and independent variables:**

After specifying my default column, I regarded that as the dependent variable and considered all the other columns as the independent variables, so I selected the default column as y and selected all other column that isn’t the default column as x. Text

Description automatically generated

**Train – Test- Split:**

I needed to then split my data into training and test data, the training data will be used to train the model while the test data will be used to test the performance of the model on a smilingly new dataset.

I set the test size to 25% of the entire dataset and ensured that the test data was selected at random to ensure that my model is isn’t just performing on a specific group. Text

Description automatically generated

The random controls the shuffling process of the train and test data. Since I have set a specific value, the shuffling process will always be the same no matter how many times I run the code.

**Random Forest Implementation:**

Once the data had been properly prepared and cleaned, there are no string values on any of the column, I went on to use the training part of the dataset which had the variable name X\_train, to train the random forest model. The random forest function was imported from the sklearn library and then the model was trained with the X\_train and y\_train datasets.

Some of the parameters I used for the random forest model in are the n\_estimator (the number of trees I would like to build before taking the maximum voting or averages of prediction) and the max\_depth (the depth I wish every tree in my random forest to grow). The random state value of the random forest is used to ensure that the randomization of the dataset is constant, hence, if I run the model multiple times, I will get the same accuracy and the same mean absolute error.

On fitting the model, I used it to predict the test data and compared the accuracy of the model with that of the actual values from the test data. From my result, 76% of the prediction was accurate (i.e. the model predicted accurately that 76% of the loan applicants will either default or not), and the mean absolute error stood at 0.23.

Text

Description automatically generated

To further optimize the model, I decided to play around with the hyperparameters I used in creating the model to see how the model performs with other hyperparameters. I wrote a function that changes the values of the n\_estimator as well as the values of the max\_depth. I used random n\_estimator values and random max\_depth values and printed out the value with the best accuracy and max\_depth. This process took a very long time to complete and went on for hours, however, I was able to get the best hyperparameter values from my randomly selected values.

Text

Description automatically generated

Text

Description automatically generated

From my result, I realized that the best n\_estimator to use for my model was 400 while the best depth was 100.

Combining the two hyperparameters into my model, I got a model with a 79% accuracy and a 0.209 mean absolute error.

Text

Description automatically generated

**Decision Tree Implementation:**

For the decision tree, I used the default hyperparameters available in the model by not specifying my hyperparameters. From the performance of the model, I got an accuracy score of 71.7% and a mean absolute error of 28% or 0.28.

Text

Description automatically generated

I then created random max\_depth values and tested them on the model to see the performance of the model by based on different hyperparameter values.

A screenshot of a computer

Description automatically generated with medium confidence

I printed out the value with the least mean absolute error and discovered the best max\_depth value is 8.

I introduced that to the model and check the performance of the model, the model predicted 77% of the loan applicants will either default or not default accurately and had a mean absolute error of 22.78%.

Text

Description automatically generated

**KNN Implementation:**

For the KNN model, I started the model by setting the n\_neighbours hyperparameter to 200. The model showed an accuracy of 71%.

Text

Description automatically generated

However, to ensure that I am using the best hyperparameter, I decided to play around with the n\_neighbour, I set the hyperparameter values to choose values between 1 and 500 with an interval of 10, meaning 1, 11, 21 etc.

Text

Description automatically generated

Based on the n\_neighbour values, I discovered that the n\_neighbour with the least mean absolute error is 41.

Therefore, I changed the n\_neighbour in the KNN model to 41 and on testing the model on the test data, I discovered that the model had a 72.6% accuracy and a mean absolute error of 0.27.

Text

Description automatically generated

**XG Boost Implementation:**

For the XG Boost model, I trained the model using the default hyperparameters available in XGBoost and this resulted in a model with a 78.5% accuracy and 0.214 mean absolute error.

Text

Description automatically generated

Although I believe I could have improved the performance of this model by playing around with the hyperparameters, I soon realized that this put significant strain on my pc, and I had to quit the process after over 550 minutes of continued processing.

**Conclusion:**

On comparing the performance of the model, the random forest showed the highest accuracy on comparing the model prediction with the actual default data. It predicted accurately the possibility of defaulting for 79% of the loan borrowers and showed a mean absolute error of only 0.2099. This means that this model will be accurate 80% of the time that it is used on a new loan applicant. This will be very useful to loan companies as it will help them to decide whether to give a loan to an individual or not.

**Recommendation:**

Although the random forest seemed to perform better, this was because I was able to play around with hyperparameters and get the values with the best performance. I believe that the XG Boost would have performed a lot better if I had the opportunity to play around with its hyperparameters as well.