BUAN 6341.003 Applied Machine Learning

Group 7

Project Report

Credit Score Classification

Group Members

Abiodun Bello

Ying-Tzu Chen

Mrudula Jethe Bhanushali

Shruthi Ramesh

Arwin Kumar Ravi

Rohith Thiagarajan

MOTIVATION:

Credit scores are essential in evaluating an individual's financial health and determining access to services like loans and credit cards. However, traditional models like FICO often rely on limited features, leading to inefficiencies, biases, and inaccuracies in assessing creditworthiness. This highlights the need for more advanced, data-driven approaches.

Machine learning offers a transformative solution for credit score classification, enabling financial institutions to make fairer, more accurate, and personalized decisions. By leveraging data and sophisticated algorithms, these systems can improve credit accessibility, reduce bias, and promote financial inclusion while enhancing efficiency and scalability.

In this project implementation we offer a promising solution to this challenge which will improve the following factors:

1) Enhanced Risk Assessment
2) Personalized Credit Scoring
3) Reducing System Bias
4) Improved Financial Inclusion
5) Cost Efficiency and Automation
6) Scalability

DATA:

A) OVERVIEW:

- This particular dataset was acquired from Kaggle. The link to the same is: https://www.kaggle.com/datasets/ayushsharma0812/dataset-for-credit-score-classification/data?select=credit_score.csv.
- The dataset has 100,000 records providing labelled customer credit score data relating to various financial prospects.
- Number of Categorical Features =12
  Number of Numerical Features = 16

Column Name	Description	Data Type
ID	Unique row identifier	object
Customer_ID	Unique identification of a customer	object
Month	Month of the year for which the data has been gathered	object

Name	Name of the customer	object
Age	Age of the customer	Int64
SSN	Social security number of the customer	object
Occupation	Occupation of the customer	object
Annual_Income	Annual Income of the customer	Float64
Monthly_Inhand_Salary	Monthly in hand salary of the customer	Float64
Num_Bank_Accounts	Number of back accounts held by the customer	Int64
Num_Credit_Card	Number of credit cards held by the customer	Int64
Interest_Rate	Interest rate on credit card	Float64
Num_of_Loan	Number of loans borne by the customer	Int64
Type_of_Loan	Type of loans held by the customer in the order that they were taken	object
Delay_from_due_date	Average number of days delayed from the payment due date	Int64
Num_of_Delayed_Payment	Number of payments delayed	Int64
Changed_Credit_Limit	Percentage change in credit card limit	Float64
Num_Credit_Inquiries	Number of credit card inquiries made by lenders on customer	Int64
Credit_Mix	Type of the mix of credits that the customer has	object
Outstanding_Debt	Remaining debt to be paid (in USD)	Float64
Credit_Utilization_Ratio	Utilization ratio of credit card	Float64
Credit_History_Age	The age of credit history of the customer in months	object
Payment_of_Min_Amount	Whether minimum amount was paid by the customer or not	object
Total_EMI_per_month	Monthly EMI payments (in USD)	Float64
Amount_invested_monthly	Monthly amount invested by the customer (in USD)	Float64
Payment_Behaviour	Payment behaviour of the customer	object

Monthly_Balance	Monthly balance amount of the customer (in USD)	Float64
Credit_Score(Target Variable)	Credit score assigned to the customer for given month	object

B) DATA CLEANING:

I. Checking for Duplicate records in the data. Fortunately, the chosen dataset has no duplicate records.

II. Cleaning numerical columns containing entries with some characters as anomalies. In most numerical columns there were values having an underscore as an anomaly as shown below. The underscore was removed, and the values were converted to a numerical type.

Num_of_Loan	Type_of_Loan	Delay_from_due_date	Num_of_Delayed_Payment
5_	Not Specified, Student Loan, Student Loan, Credit-Builder Loan, and Au	15	13_
2	Payday Loan, and Home Equity Loan	8	14
2	Payday Loan, and Home Equity Loan	12	14
2	Payday Loan, and Home Equity Loan	8	14
2	Payday Loan, and Home Equity Loan	8	14
2	Payday Loan, and Home Equity Loan	8	14_
2_	Payday Loan, and Home Equity Loan	8	14
-100	Payday Loan, and Home Equity Loan	8	16

III. Dropping columns that will not be required for the design of the models. Here, we drop the columns 'SSN' and 'Name'.

IV. Handling null values and anomalies in numerical columns. For instance, most numerical columns had outliers and negative values as well as shown below. We replaced these values and Null with the value occurring before or after that specific record using the forward (ffill()) and backward (bfill()) fill functions while the data was grouped by 'Customer_Id'. A range of values was enforced for the numerical variables to avoid extremely high or negative values using the Inter Quartile Range formula.

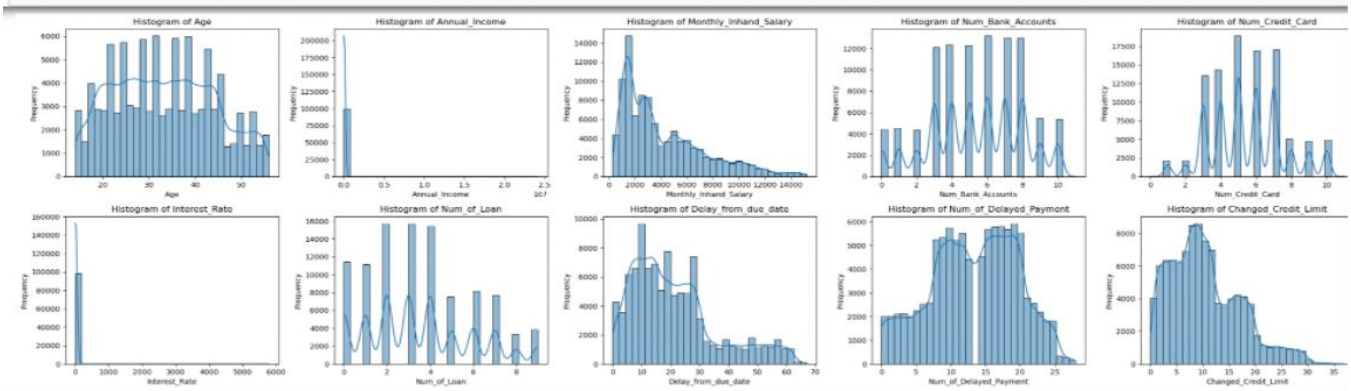
	Age	Annual_Income	Monthly_Inhand_Salary	Num_Bank_Accounts	Num_Credit_Card	Interest_Rate	Num_of_Loan	Delay_from_due_date
count	100000.000000	1.000000e+05	84998.000000	100000.000000	100000.00000	100000.000000	100000.000000	100000.000000
mean	110.649700	1.764157e+05	4194.170850	17.091280	22.47443	72.466040	3.009960	21.068780
std	686.244717	1.429618e+06	3183.686167	117.404834	129.05741	466.422621	62.647879	14.860104
min	-500.000000	7.005930e+03	303.645417	-1.000000	0.00000	1.000000	-100.000000	-5.000000
25%	24.000000	1.945750e+04	1625.568229	3.000000	4.00000	8.000000	1.000000	10.000000
50%	33.000000	3.757861e+04	3093.745000	6.000000	5.00000	13.000000	3.000000	18.000000
75%	42.000000	7.279092e+04	5957.448333	7.000000	7.00000	20.000000	5.000000	28.000000
max	8698.000000	2.419806e+07	15204.633333	1798.000000	1499.00000	5797.000000	1496.000000	67.000000

V. Typecasted the 'Credit_History_Age' column to a float by converting the original string with number of years and months to total number of years (float) using the split() function as shown below.

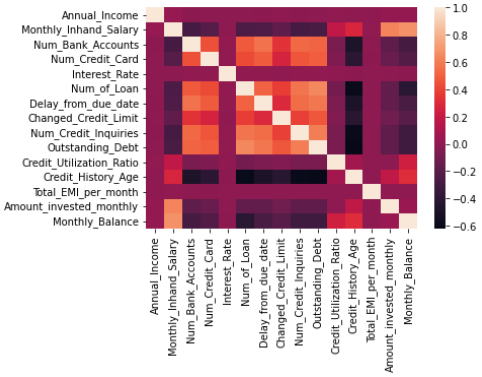
VI. Handling anomalies in categorical variables. Some variables had some random values which were replaced by a similar value when grouped by the 'Customer_Id'.

C) EDA:

Initially, the histograms for all the numerical columns are plotted to check for the range of values and skewness of the data.



Next, we construct a correlation matrix and plot the same using a heatmap to visualize the relationships between the numerical variables.



D) Data Preprocessing (1): One-Hot Encoding for SVM & Logistic Regression Models

a. There are 8 categorical features (excluding CustomerID and Credit_Score) need the encoding, and the Type_of_loan feature is a challenge because it has more than 6,000 unique values. Directly

using one-hot encoding would have more than 6,000 columns, which is not practical.

```
Categorical Features are:
0 Customer_ID
1 Month
2 Occupation
3 Type_of_Loan
4 Credit_Mix
5 Payment_of_Min_Amount
6 Payment_Behaviour
7 Credit_Score
8 Type_of_Loan_Split
9 Type_of_Loan_Cleaned
```

	Type	Unique Values
Customer_ID	object	12500
Month	object	8
Age	int64	43
Occupation	object	15
Annual_Income	float64	13487
Monthly_Inhand_Salary	float64	13235
Num_Bank_Accounts	float64	12
Num_Credit_Card	float64	12
Interest_Rate	int64	1750
Num_of_Loan	float64	10
Type_of_Loan	object	6260
Delay_from_due_date	float64	68
Num_of_Delayed_Payment	float64	29

b. Assumption: Through observation, each Type_of_Loan value contains multiple types that were combined and separated by commas. It may indicate that there are a set of "basic" values that are combined to create these 6,000 unique values. The distinct "basic" values are likely to be fewer. If the assumption is correct, we could encode the basic units and reduce the dimensionality significantly.

c. Testing the assumption: For each row, split the string and sum up the new strings to calculate the types. In this result, there were 17 loan types. Then, we noticed that there were duplicates due to the description in strings. Sometimes the loan type had an 'and' before the type's name, but they were the same type (as shown in the left screenshot below). For example, 'and Home Equity Loan' is the same as 'Home Equity Loan'. After removing the 'and' and recalculating the types, the result has only 9 distinct loan types (as shown in the right screenshot below).

```
In [46]: print(df['Type_of_Loan'][0:15])

0             Credit-Builder Loan, and Home Equity Loan
1             Credit-Builder Loan, and Home Equity Loan
2             Credit-Builder Loan, and Home Equity Loan
3             Credit-Builder Loan, and Home Equity Loan
4             Credit-Builder Loan, and Home Equity Loan
5             Credit-Builder Loan, and Home Equity Loan
6             Credit-Builder Loan, and Home Equity Loan
7             Credit-Builder Loan, and Home Equity Loan
8     Not Specified, Home Equity Loan, Credit-Builde...
9     Not Specified, Home Equity Loan, Credit-Builde...
10    Not Specified, Home Equity Loan, Credit-Builde...
11    Not Specified, Home Equity Loan, Credit-Builde...
12    Not Specified, Home Equity Loan, Credit-Builde...
13    Not Specified, Home Equity Loan, Credit-Builde...
14    Not Specified, Home Equity Loan, Credit-Builde...
Name: Type_of_Loan, dtype: object
```

```
16  Auto Loan
17  Credit-Builder Loan
18  Debt Consolidation Loan
19  Home Equity Loan
20  Mortgage Loan
21  Not Specified
22  Payday Loan
23  Personal Loan
24  Student Loan
```

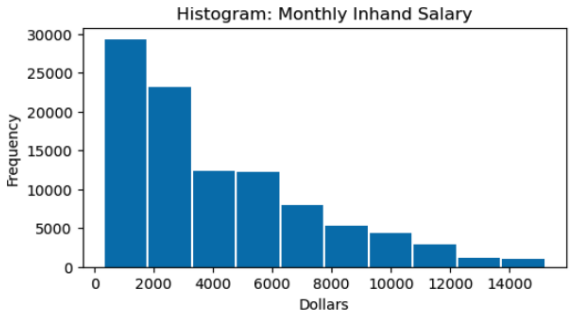
d. **One-Hot Encoding for All Categorical Features:** The new dataset has 61 columns.

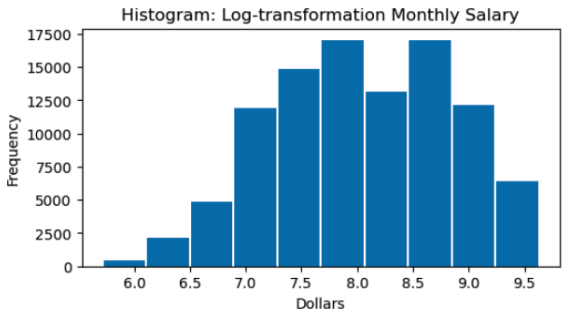
```
   Auto Loan  Credit-Builder Loan  Debt Consolidation Loan  Home Equity Loan  \
0          0                    1                        0                 1
1          0                    1                        0                 1
2          0                    1                        0                 1
3          0                    1                        0                 1
4          0                    1                        0                 1

   Mortgage Loan  Not Specified  Payday Loan  Personal Loan  Student Loan
0              0              0            0              0             0
1              0              0            0              0             0
2              0              0            0              0             0
3              0              0            0              0             0
4              0              0            0              0             0
```

```
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 61 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   Age                       100000 non-null  int64
 1   Monthly_Inhand_Salary     100000 non-null  float64
 2   Num_Bank_Accounts         100000 non-null  float64
```

E) **Data Preprocessing (2): Log Transformation for Skewed data**

- The images below show the log transformation for the salary feature.
- We only transformed one skewed data for demonstration purpose. This part was not included in our model training because by the time we noticed this part, there was insufficient time to renew all the features and retrain all the models.





MODELS:

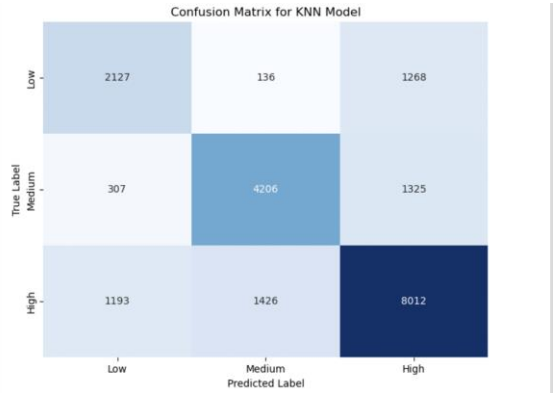
A) K- Nearest Neighbour:

K-Nearest Neighbors (K-NN) is a supervised learning algorithm widely used for classification tasks. The model classifies data points based on the majority class of their k-nearest neighbours in the feature space. In this project, k=5 was chosen after experimentation, and the Euclidean distance metric was used to calculate the distances between data points. This approach is particularly effective for standardized financial datasets, where absolute distances between features are meaningful.

KNN Classification Report:

	precision	recall	f1-score	support
0	0.59	0.63	0.61	3566
1	0.75	0.73	0.74	5799
2	0.76	0.75	0.76	10635
accuracy			0.72	20000
macro avg	0.70	0.70	0.70	20000
weighted avg	0.73	0.72	0.72	20000

KNN ROC AUC Score: 0.8464557989362519



The dataset was preprocessed by normalizing numerical features using StandardScaler and encoding categorical variables with LabelEncoder to ensure compatibility with the K-NN algorithm. An 80-20 train-

test split was applied, with 80% of the data used for training and 20% for testing. These steps ensured that all features contributed equally during distance calculations and minimized potential biases in the model.

Performance:

The K-NN model achieved an accuracy of 72% with a strong ROC-AUC score of 0.85, as shown in the classification report below:

The confusion matrix highlights the distribution of predictions:

- Class 0 (Low Credit Score): Precision: 0.59, Recall: 0.63
- Class 1 (Medium Credit Score): Precision: 0.75, Recall: 0.73
- Class 2 (High Credit Score): Precision: 0.76, Recall: 0.75

Overall, K-NN proved to be a suitable choice for the credit score classification task due to its simplicity and ability to handle mixed numerical and categorical data. Although the performance was decent, there is room for improvement through techniques like weighted K-NN, alternative distance metrics (e.g., Manhattan), or feature selection to optimize the model further.

B) Decision Tree:

A decision tree algorithm in Python is typically implemented to model a decision-making process by splitting data into branches based on specific conditions. It represents decisions and their possible outcomes as a tree-like structure, where Nodes represent features or attributes, Edges correspond to decision rules or thresholds, Leaf nodes provide the final prediction (class label or value).

In this project, the target variable (Credit_Score) Y was encoded such that 0 represents "Poor" credit rating, 1 represents "Standard" credit rating, and 2 represents "Good" credit rating. This encoding ensured the classes were well-represented in the model. The dataset was processed to handle categorical variables and missing values, followed by scaling of numerical features to ensure uniformity in the range of values. A 70-30 (test_size = 0.3) train-test split was applied, and the Decision Tree model achieved an overall test accuracy of 71%.

Decision Tree Classifier Report:

	precision	recall	f1-score	support
0	0.65	0.64	0.65	5300
1	0.70	0.69	0.69	8812
2	0.74	0.75	0.74	15888
accuracy			0.71	30000
macro avg	0.70	0.69	0.69	30000
weighted avg	0.71	0.71	0.71	30000

The Decision Tree algorithm proved to be a more effective model for predicting credit scores, outperforming linear classifiers like SVM because it naturally handles nonlinear relationships in data by splitting features at thresholds. It can also handle categorical and numerical features without requiring one-hot encoding or scaling.

C) Random Forest (Ensemble Model):

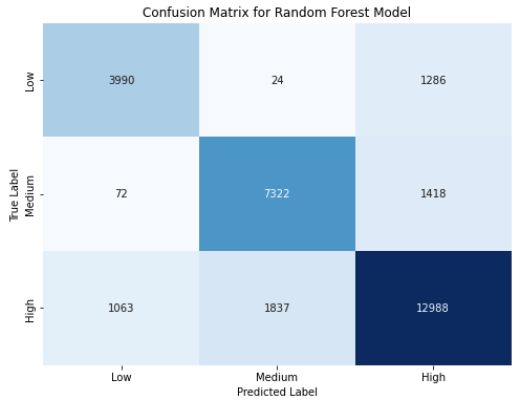
Random Forest is an ensemble learning method that builds multiple decision trees during training and combines their outputs for improved predictive accuracy and control over overfitting. In this project, the target variable YYY was encoded such that 0 represents "Poor" credit rating, 1 represents "Standard" credit rating, and 2 represents "Good" credit rating. This encoding ensured the classes were well-represented in the model.

The dataset was processed to handle categorical variables and missing values, followed by scaling of numerical features to ensure uniformity in the range of values. Using cross-validation techniques, the model achieved consistent performance across folds, ensuring its robustness and generalizability. An 80-20 train-test split was applied, and the Random Forest model achieved an overall test accuracy of 81%, with a macro-average ROC-AUC score of 0.92, indicating excellent predictive capabilities.

```
# Define the custom order for the target variable
custom_order = ['Poor', 'Standard', 'Good']

# Convert y to a NumPy array if it's not already
y = np.array(y, dtype=str)  # Ensure y is a NumPy array of strings

# Handle missing values (replace 'nan' or np.nan with a default label or remove them)
y = np.where((y == 'nan') | (y == 'NaN') | (pd.isnull(y)), 'missing', y)

# Manually map the labels to the desired integers, including missing if necessary
if 'missing' in y:
    custom_order = custom_order + ['missing']

label_encoder = LabelEncoder()
label_encoder.classes_ = np.array(custom_order)  # Set the classes in the desired order

# Encode the target variable
y_encoded = label_encoder.transform(y)
print(y_encoded[:50])
```

```
[1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 0 0]
```

```
Randomforest Classification Report:
              precision    recall  f1-score   support

           0       0.79      0.84      0.81      5799
           1       0.83      0.82      0.82     10635
           2       0.78      0.76      0.77      3566

    accuracy                           0.81     20000
   macro avg       0.80      0.80      0.80     20000
weighted avg       0.81      0.81      0.81     20000

Randomforest ROC AUC Score: 0.9249824104182799
```



The confusion matrix highlights the model's ability to classify instances correctly across all three credit score classes. Specifically, Class 0 (Poor) achieved a precision of 0.79 and a recall of 0.84, while Class 1 (Standard) demonstrated the highest precision and recall at 0.83 and 0.82, respectively. Class 2 (Good) achieved slightly lower precision and recall values but still performed well overall. These metrics, along with the high ROC-AUC score, emphasize the effectiveness of Random Forest for this multi-class classification problem.

The Random Forest algorithm proved to be a highly effective model for predicting credit scores, outperforming linear classifiers like SVM in terms of accuracy and generalization. Its strength lies in handling non-linear relationships and high-dimensional data efficiently.

D) <u>Support Vector Machine</u>:

A Support Vector Machine (SVM) is a supervised machine learning algorithm commonly used for classification tasks. It works by finding a hyperplane that best separates the data into different classes. The key idea behind SVM is to maximize the margin, or the distance, between the closest data points of each class (known as support vectors) and the separating hyperplane.

In the case of a linear SVM, the model assumes that the data can be separated by a linear hyperplane. This hyperplane is a decision boundary that divides the feature space into two regions, one for each class. The goal is to identify the hyperplane that maximizes the margin between the two classes.

We have chosen a Linear SVM model because our data is large and has about 61 possible variables. This model was trained using Grid Search and 5-fold Cross Validation using the 'C' parameter values as [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000].

The model that showed the best performance had a 'C' parameter value of '1' and a model accuracy of around 64.7% as shown below.

```
Model Performance:
Best Parameter Value: {'C': 1}
Best Model Accuracy: 0.6478533333333334
Testinf Accuracy: 0.64796
Training Accuracy: 0.6483066666666667
```

	precision	recall	f1-score	support
Good	0.51	0.75	0.61	4438
Poor	0.66	0.50	0.57	7290
Standard	0.71	0.69	0.70	13272
accuracy			0.65	25000
macro avg	0.63	0.65	0.63	25000
weighted avg	0.66	0.65	0.65	25000

The poor performance of a linear SVM can be due to a variety of factors such as non-linearly separable data, high dimensionality, class imbalance, noisy data, or improper parameter tuning. By addressing these factors through feature engineering, kernel selection, parameter optimization, and other strategies, you can improve the model's performance. If the linear SVM still doesn't perform well after addressing these issues, switching to a more flexible model like a non-linear SVM or other classifiers might be necessary.

E) **Logistic Regression:**

Although we acknowledged that scikit-learn's logistic regression might be less suitable for our 3 levels credit score, we decided to train the models as it is a well-known method for classification problems. For hyperparameter tuning, we set the regularization parameter 'C': [0.01, 0.1, 1, 10, 100], the solvers: default and 'saga', and the maximum iteration 'max_iter': [100, 200, 300, 500, 1000, 2000]. We set max_iter to large numbers according to the system's recommended message in the early versions of models. After completing the training, we identified an input data issue and changed to the version 2 dataset. With the adjustment, the max-iter-related message no longer appeared when iteration reached 500. The best model for the version 2 dataset is 'C' = 100, max_iter = 500. The accuracy score is 0.65.

F) **Ensemble Logistic Regression and SVM:**

The best accuracy of our logistic regression model and SVM are not ideal. To experiment if we could gain a better result, we used ensemble approach combining them. Unfortunately, the accuracy is only 0.60.

MODEL COMPARISON:

Our best model is the random forest. One possible explanation may be that decision trees and random forests can capture nonlinear relationships between features. Moreover, our dataset is imbalanced, the distribution of score types are: Standard 53%, Poor 28%, and Good 17%. The random forest has built-in approaches to handle the imbalance. In contrast, the SVM and logistic regression performed less well because these two are more suitable for linear relationships and balanced data.

Model	**Hyperparameters**	**Best Acc. Score**
Random Forest	Cross-validation = 8	0.81
KNN	Cross-validation = 8	0.72
Decision Tree	Cross-validation = 8	0.72
Ensemble, Voting: KNN, D-Tree, SVM	Cross-validation = 8	0.70
Linear SVM	'C': [0.01, 0.1, 1, 10, 100, 1000], 'max_iter': [100, 200]	0.64
Logisitic Regression	C: [0.01, 0.1, 1, 10, 100, 300, 500] Slover: lbfgs, saga Max iter = 100, 200, 500, 1000, 2000	0.65
Ensemble, voting: SVM + Log-Reg 300	Max iter = 500, Slover: lbfgs	0.60

CONCLUSION:

Throughout the project, we successfully built and evaluated models that can accurately predict an individual's or entity's credit score based on historical financial data. We employed various techniques, including feature engineering, model selection, and evaluation metrics, to address challenges such as class imbalance, overfitting, and interpretability.

Our findings indicate that machine learning models, when properly trained and validated, outperform traditional scoring methods in terms of predictive accuracy, and have the potential to mitigate biases inherent in older credit assessment models. Moreover, the automation of this process could lead to significant cost savings for financial institutions, while offering consumers a more dynamic and personalized approach to credit evaluation.

However, we encounter several challenges on the way which are as follows

- Missing data in financial datasets is one of the biggest challenges. This is particularly problematic in credit scoring, where certain fields (e.g., income) may be crucial for accurate predictions.
- Class imbalance is another challenge because a lot of time this leads to overfitting of the model.
- Credit scoring is also a significant challenge as it involves not only static features like income and employment but also temporal or sequential data such as changes in credit history over time, spending patterns, and fluctuations in financial status.
- After pre-processing the data, model implementations had to be done with data having a dimensionality of 61. It is challenging to implement classification models on data of such high dimensionality because a lot of models show very poor accuracy for higher dimensional data. Some models can't even be implemented due to the same reason (e.g., Kernel SVM model).

The interpretability of machine learning models continues to be a critical concern, particularly when decisions about creditworthiness can have significant life-altering consequences. Additionally, ensuring data privacy and addressing ethical concerns around algorithmic decision-making are important considerations for the widespread adoption of machine learning in credit scoring.

APPENDIX:

1. In the presentation, we understood that our x and y axes' labels were confusing, and we fixed them in this final version. The previous version of the credit scores were 0: Good, 1: Poor, 2: Standard. The updated encodings are 'Poor':0, 'Standard':1, 'Good':2.
2. Fixed the log transformation for skewed data image for clarity in ‘Data Preprocessing (2)’ part.
3. When combining everyone's codes and models, we noticed that we have different versions of the data processing methods. Since retraining the models would be time-consuming, we eventually retained multiple files.