ML Project: Credit Score

Project to analyse and get the best model for Credit Score prediction of unknown data.

Areen Vaghasiya
IMT2022048
areen.vaghasiya@iiitb.ac.in

Aryan Vaghasiya
IMT2022046
aryan.vaghasiya@iiitb.ac.in

Shreyank Gopalkrishna Bhat
IMT2022516
shreyank.bhat@iiitb.ac.in

I. Introduction

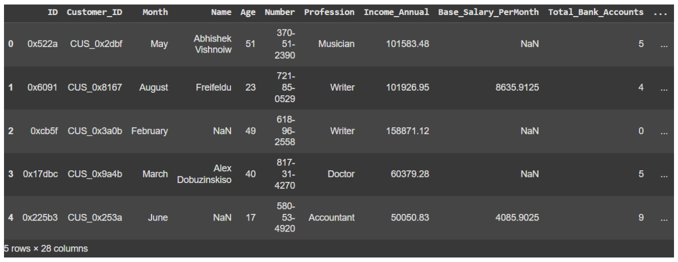
This project focuses on building machine learning models to predict the **Credit Score** of individuals based on various financial and behavioral features. By leveraging historical data, the goal is to create a robust and accurate model that can assist in evaluating the risk associated with extending credit. Credit scoring is a critical aspect of financial decision-making, providing institutions with a quantitative measure of an individual's creditworthiness.

The dataset used for this project includes two parts:

- *train.csv* - A labeled dataset used to train the machine learning models.
- *test.csv* - An unlabeled dataset(doesn't have the column 'Credit_Score') to evaluate the model's final performance and predict credit scores for unseen data.

The columns of the dataset(train.csv) are:

1) 'ID'
2) 'Customer_ID'
3) 'Month'
4) 'Name'
5) 'Age'
6) 'Number'
7) 'Profession'
8) 'Income_Annual'
9) 'Base_Salary_PerMonth'
10) 'Total_Bank_Accounts'
11) 'Total_Credit_Cards'
12) 'Rate_Of_Interest'
13) 'Total_Current_Loans'
14) 'Loan_Type'
15) 'Delay_from_due_date'
16) 'Total_Delayed_Payments'
17) 'Credit_Limit'
18) 'Total_Credit_Enquiries'
19) 'Credit_Mix'
20) 'Current_Debt_Outstanding'
21) 'Ratio_Credit_Utilization'
22) 'Credit_History_Age'
23) 'Payment_of_Min_Amount'
24) 'Per_Month_EMI'
25) 'Monthly_Investment'
26) 'Payment_Behaviour'



	ID	Customer_ID	Month	Name	Age	Number	Profession	Income_Annual	Base_Salary_PerMonth	Total_Bank_Accounts	...
0	0x522a	CUS_0x2dbf	May	Abhishek Vishnoiw	51	370-51-2390	Musician	101583.48	NaN	5	...
1	0x6091	CUS_0x8167	August	Freifeldu	23	721-85-0529	Writer	101926.95	8635.9125	4	...
2	0xcb5f	CUS_0x3a0b	February	NaN	49	618-96-2558	Writer	158871.12	NaN	0	...
3	0x17dbc	CUS_0x9a4b	March	Alex Dobuzinskiso	40	817-31-4270	Doctor	60379.28	NaN	5	...
4	0x225b3	CUS_0x253a	June	NaN	17	580-53-4920	Accountant	50050.83	4085.9025	9	...

5 rows × 28 columns

Fig. 1. Shows the result of $print(train\_data.head())$

27) 'Monthly_Balance'
28) 'Credit_Score'

A sample data of what the dataset looks like is shown in Fig 1.

II. Exploratory Data Analysis & Preprocessing

The dataset underwent various preprocessing steps to prepare it for modeling. Key steps included:

- Handling mistyped numerical values.(eg - '_4_')
- Handling missing values in columns with specific data transformations to avoid bias.
- Removing irrelevant columns
- Handling miscellaneous values
- Encoding categorical variables using one-hot encoding and label encoding where necessary.
- Normalizing or scaling numerical features to improve model convergence and performance.

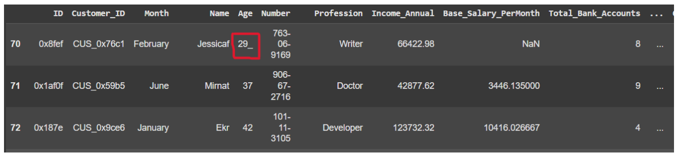
A. Handling Mistyped Values:

There are many columns in the dataset that are supposed to have numerical type data (integer,float) but have the datatype defined for them as 'object' after storing them in python's Panda data frame. Fig 2 is such an example.

certain columns expected to have numerical values may be of type 'object' in Panda's Dataframe due to errors such as **improper formatting**, **inclusion of non-numeric characters**, or **missing values** represented as strings.

The columns susceptible to this error are :

- 'Age'
- 'Income_Annual'
- 'Total_Current_Loans'
- 'Total_Delayed_Payments'



	ID	Customer_ID	Month	Name	Age	Number	Profession	Income_Annual	Base_Salary_PerMonth	Total_Bank_Accounts	...
70	0x8fef	CUS_0x76c1	February	Jessicaf	29_	763-06-9169	Writer	66422.98	NaN	8	...
71	0x1af0f	CUS_0x59b5	June	Mirnat	37	906-67-2716	Doctor	42877.62	3446.135000	9	...
72	0x187e	CUS_0x9ce6	January	Ekr	42	101-11-3105	Developer	123732.32	10416.026667	4	...

Fig. 2. Shows the result of $print(train\_data[70:80])$

- 'Credit_Limit'
- 'Credit_Mix'
- 'Current_Debt_Outstanding'
- 'Monthly_Investment'
- 'Monthly_Balance'

Correcting these columns involve 2 steps:

→ Stripping '_'s from the column

→ Manually converting the *object* dtypes to *float* or *integer* dtypes.

Correcting these errors is essential for the following reasons:

- **Facilitating Numerical Operations**
- **Ensuring Consistency in Data Representation:** Incorrect data types can lead to inconsistencies when processing or visualizing data.
- **Avoiding Bias or Skew in Analysis:** Numerical values incorrectly stored as strings may be excluded or misinterpreted by algorithms, reducing the model's ability to learn relationships in the data.

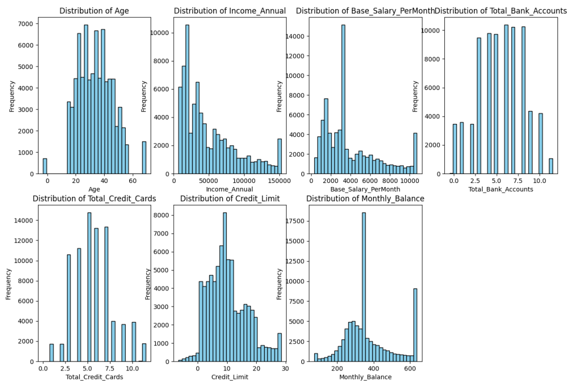


Fig. 3. Univariate Data Analysis

Fig 3 shows the missing data pattern.

B. Handling Outliers:

There are many columns that are wrongly predicted due to the presence of outliers. Columns like Age, BaseSalaryPerMonth, IncomeAnnual, TotalBankAccounts, CreditLimit, PerMonthEMI, etc have large outliers in them as shown in fig 5 .

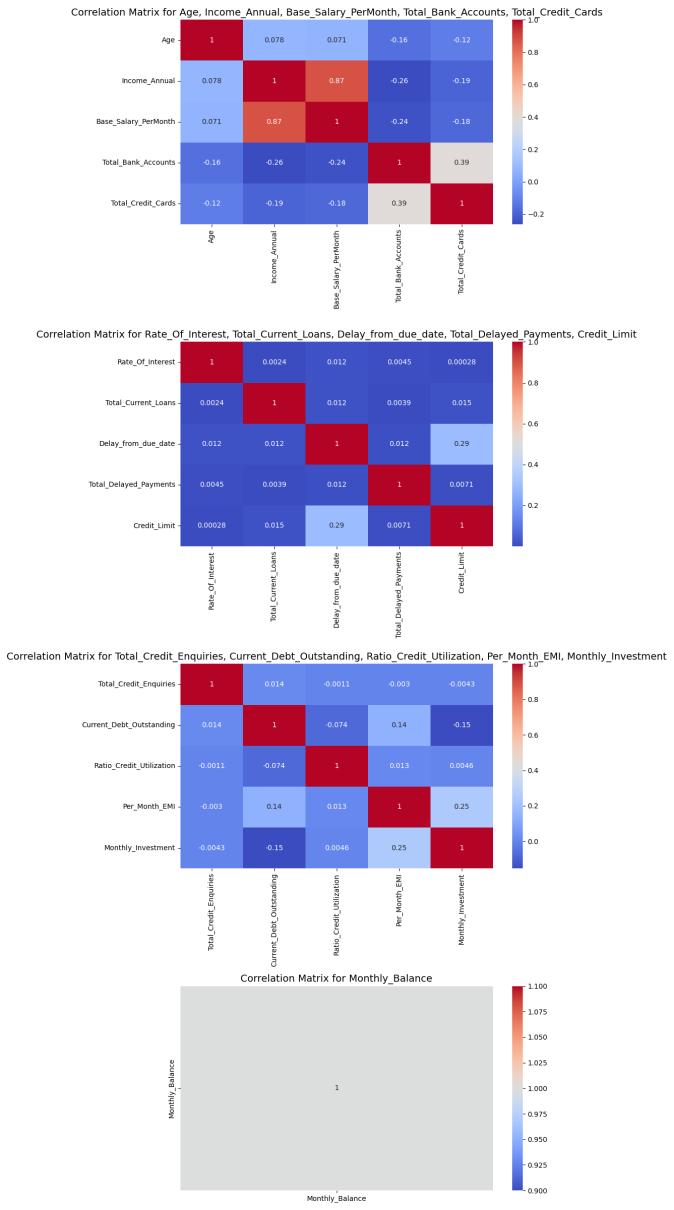


Fig. 4. Enter Caption

However, this can be easily solved as for the columns with outliers have numerical values so we can simply replace these values with the upper and lower whisker values. Defining the quartiles from 0.25 to 0.75 and adding or subtracting 1.5(inter quartile-range) will give an appropriate values to replace the outliers as in fig 6.

Having corrected the outlier values, we can also do analysis like distribution of CreditScore by age 8.
We can see which age values contribute more to the creditScores and which don't. We can see that for the age range between age 50 to 60, there are more people that maintain a good CreditScore as compared to those having Standard or Poor CreditScore.

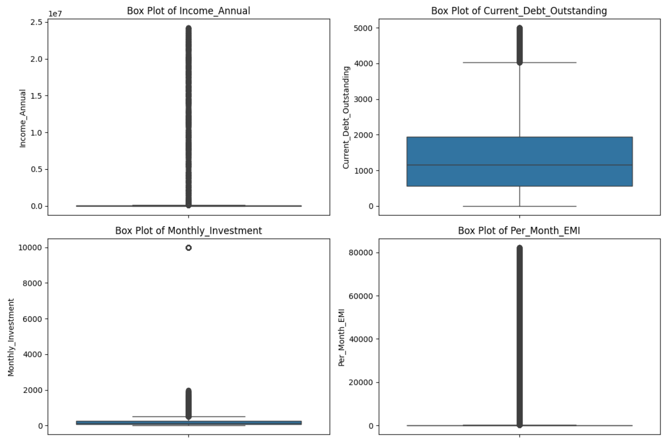


Fig. 5. Columns with Outliers present

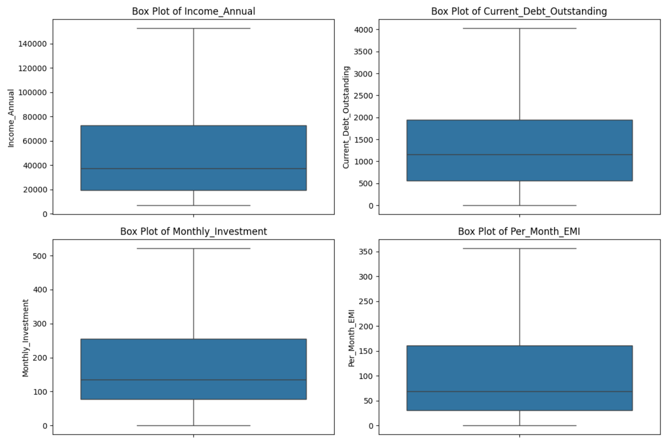


Fig. 6. Columns with outliers removed

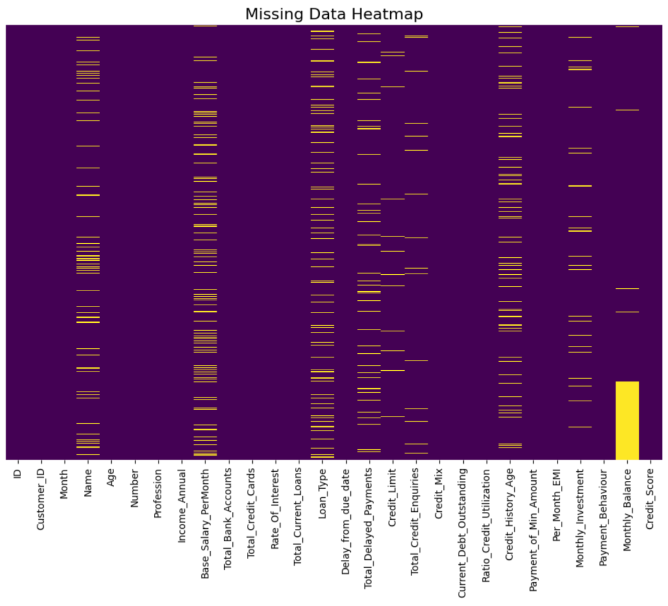


Fig. 7. Shows the Missing Data(null) in various columns

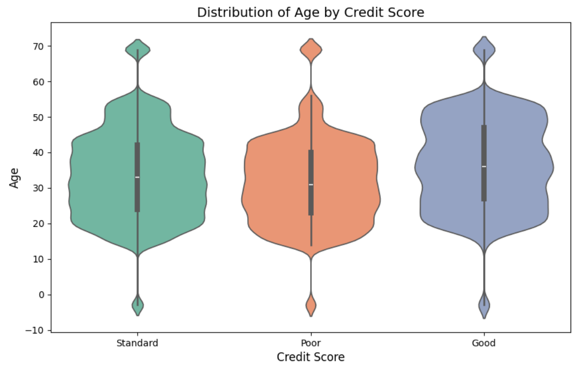


Fig. 8. Age Distribution by CreditScore

We can also do scatterPlots to identify trends and try to correlate between the different variables and see how they affect each other. We can have a Age vs. Income_Annual or Age vs. Monthly Balance to identify potential trends with age. A shown in 10, 11, 9

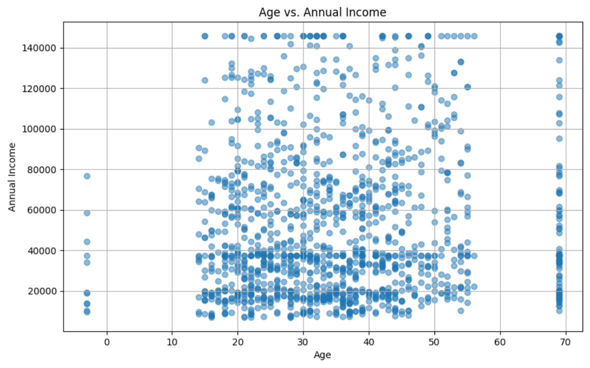


Fig. 9. Scatter Plot of Annual Income vs Age

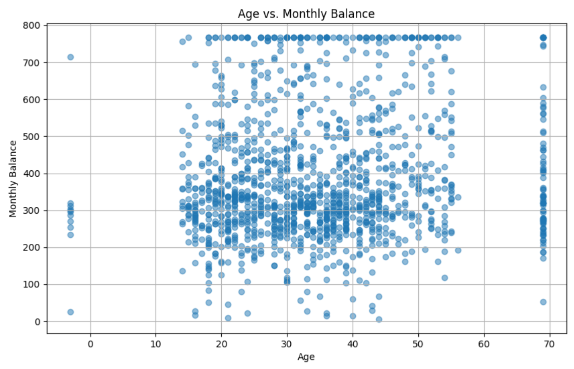


Fig. 10. Scatter Plot of Monthly_Balance vs Age

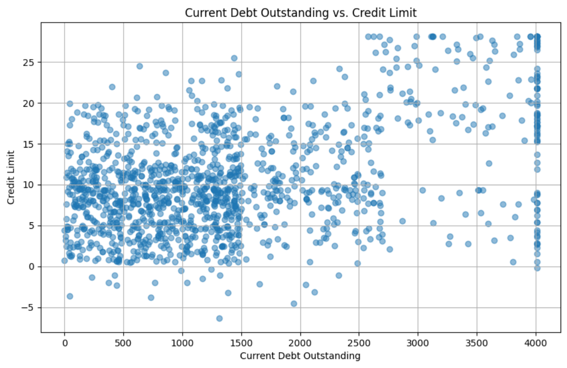


Fig. 11. Scatter Plot of Current_Debt_Outstanding vs CreditLimit

C. *Correcting Miscellaneous Variables*

There are a few columns in the dataset that have data in an inaccessible/unreadable format.
For ex- columns such as 'Profession' has value '_______', which doesn't make sense.

There is another column 'Credit_History_Age' which contains the information about how long is the credit history of the corresponding customer. This is stored in the dataset as strings, e.g '1 year and 5 months'. This has been changed to integer values representing number of months. The code updating this column is:

```
import numpy as np

def calc_total_months(history):
   if pd.isna(history): # Check for NaN values
       directly using pandas
      return np.nan
   histlist = history.split() # Split by whitespace
   years = int(histlist[0]) if
       histlist[0].isdigit() else 0
   months = int(histlist[3]) if len(histlist) > 3
       and histlist[3].isdigit() else 0
   return years * 12 + months

train['Credit_History_Age'] =
train['Credit_History_Age'].apply(calc_total_months)
test['Credit_History_Age'] =
    test['Credit_History_Age'].apply(calc_total_months)
```

Similarly, column 'Loan_Type' has string values such as *'Payday Loan, Student Loan, Payday Loan, and Debt Consolidation Loan '* as a single string. This has been handled by adding corresponding 'loan' columns. The following columns have been added:

- Loan_Type_Mortgage Loan
- Loan_Type_Debt Consolidation Loan
- Loan_Type_Auto Loan
- Loan_Type_Student Loan
- Loan_Type_Payday Loan
- Loan_Type_Personal Loan
- Loan_Type_Not Specified
- Loan_Type_Home Equity Loan
- Loan_Type_Credit-Builder Loan

Columns such as 'Payment_Behaviour' have garbage values '!@9#%8' have also been replaced with value 'Not Specified'.

D. *Removing Irrevelant columns*

Columns such as 'Name', 'Customer_ID', 'Number' have no significance in the Credit Score of the customer. Hence, these columns are not used for training models.

E. *Filling null / missing values*

Numerical columns identified with missing values are:

- Income_Annual
- Total_Delayed_Payments
- Monthly_Investment
- Monthly_Balance
- Total_Credit_Enquiries
- Credit_Limit
- Base_Salary_PerMonth

Imputation Approach:
The missing values were imputed with the median of the respective columns.

```
numeric_columns2 = ['Income_Annual',
    'Total_Delayed_Payments', 'Monthly_Investment',
               'Monthly_Balance',
                   'Total_Credit_Enquiries',
                   'Credit_Limit',
               'Base_Salary_PerMonth']
for col in numeric_columns2:
   train_data[col].fillna(train_data[col].median(),
       inplace=True)
   test_data[col].fillna(test_data[col].median(),
       inplace=True)
```

Why Median?
The median is a robust measure of central tendency and is less affected by outliers compared to the mean. This choice suggests an awareness of potential skewness in the data distribution.

F. *Encoding techniques*

Since multiple kinds of models have been experimented on the dataset, the categorical columns of have been both label encoded and one-hot encoded for different model*(one-hot encoded for models such as Bayesian classifier and label encoded for models such as decision trees)*.Categorical columns identified are:

- Profession
- Credit_Mix
- Payment_of_Min_Amount
- Payment _Behaviour

1) Label Encoding: Label encoding is done to convert categorical data into a numerical format. Certain algorithms can leverage the ordinal relationship between encoded values (if applicable) to make better predictions.

2) One-Hot Encoding: Converts categories into binary columns. This is done to avoid bias in independent non-hierarchical data values.

III. Training Models

A. *Decision Tree*

Description:
A Decision Tree is a supervised learning model used for classification and regression tasks. It splits the dataset into subsets based on the value of input features, creating a tree structure that represents the decision-making process. While simple and interpretable, decision trees are prone to overfitting, which can be mitigated with proper hyperparameter tuning.

- **Best Hyperparameters** (from *RandomizedSearchCV*):
  - *min_samples_split*: 5 (minimum number of samples required to split an internal node)
  - *min_samples_leaf*: 4 (minimum number of samples required to be at a leaf node)
  - *max_depth*: 10 (maximum depth of the tree)
- **Performance Metrics:**
  - *Test Accuracy*: 0.6985
  - *Test Precision (weighted)*: 0.704114
  - *Test Recall (weighted)*: 0.6985
  - *Test F1 Score (weighted)*: 0.699378
- **Insights and Observations**:
  1) *Model Performance*:
     The Decision Tree model achieved an accuracy of approximately 69.85%, with similar values for precision, recall, and F1-score. The metrics suggest moderately good performance, though the model may benefit from further refinement.
  2) *Evaluation of Hyperparameters*:
     - Max Depth (10): Restricting the tree depth helps reduce overfitting by limiting the complexity of the tree.
     - Min Samples Split (5): Ensures that nodes are not split unless there are at least 5 samples, preventing overly granular splits.
     - Min Samples Leaf (4): Forces each leaf node to contain at least 4 samples, promoting generalization and robustness.
  3) *Possible Reasons for Limited Accuracy*:
     - Feature Complexity: Features in the dataset may not provide sufficient predictive power, leading to limited performance.
     - Class Imbalance: If target classes are imbalanced, the tree might be biased towards the majority class.
     - Limited Tree Depth: While reducing overfitting, the restriction on tree depth may have prevented the model from capturing complex patterns.

While the Decision Tree provides interpretable results, its performance may improve with additional feature engineering, balancing the dataset, or exploring advanced ensemble methods such as Gradient Boosting or Random Forest. A deeper hyperparameter search and fine-tuning might also yield better results.

B. *Random Forest*

Description:
Random Forest is an ensemble learning method that constructs multiple decision trees and merges their results for more accurate predictions. It's robust to overfitting due to the averaging of trees and often performs well on structured data.

- **Best Hyperparameters** (from *RandomizedSearchCV*):
  - *n_estimators*: 150 (number of trees)
  - *min_samples_split*: 2 (minimum number of samples required to split an internal node)
  - *min_samples_leaf*: 2 (minimum number of samples required to be at a leaf node)
  - *max_depth*: None (trees are expanded fully unless limited by other parameters)
  - *bootstrap*: False (does not use bootstrapping, which may improve accuracy but increases overfitting risk slightly)
- **Performance Metrics:**
  - *Test Accuracy*: 0.79225
  - *Test Precision (weighted)*: 0.79201
  - *Test Recall (weighted)*: 0.79225
  - *Test F1 Score (weighted)*: 0.79211
- **Insights and Observations**:
  1) *Model Performance*:
     The Random Forest model has achieved an accuracy of approximately 79%, with similar values for precision, recall, and F1-score. This consistency across metrics indicates a balanced performance on both positive and negative samples within the classes.
  2) *Evaluation of Hyper parameters*:
     - Number of Estimators (150): Using a higher number of trees increases the model's stability and ability to generalize, while 150 trees are typically enough to balance performance and computational efficiency.
     - Max Depth (None): Allowing trees to grow to full depth can help capture complex patterns but may increase overfitting. With the selected settings, the effect seems well-balanced.
     - Bootstrap (False): The lack of bootstrapping may have contributed to a slight overfitting but generally increased the model's accuracy.
  3) *Possible Reasons for Limited Accuracy*:
     - Feature Complexity: Some features in the dataset (e.g., *Credit_Mix*, *Payment_Behaviour*) may not have straightforward relationships with the *Credit_Score*, causing a reduction in model accuracy.
     - Class Imbalance: If the *Credit_Score* categories are imbalanced, the model might favor the majority class, reducing overall performance.
     - Non-linear Relationships: The data may contain non-linear patterns that the Random Forest model captures well to an extent but could be

improved by trying other ensemble or boosting models.

Perhaps we can improve accuracy by doing a more intensive hyperparameter search to using finer parameters around the best parameters found. We can also create additional features or remove highly correlated features to reduce noise in the model. We could also try scaling or transforming some features differently from how we have done, particularly if they contain outliers or non-normal distributions.

C. Naive Bayes Classifier

Description:
Gaussian Naive Bayes is a probabilistic classification technique based on Bayes' Theorem, assuming independence among features. It is particularly effective for high-dimensional data and is known for its simplicity and speed. The Gaussian variant assumes that the likelihood of the features is normally distributed.

- **Assumptions and Key Features**:
  - Assumes that all features are independent given the class label, which might not hold true for all datasets.
  - Uses the Gaussian distribution to model the likelihood of continuous data, making it suitable for numerical datasets.
- **Performance Metrics**:
  - *Training Accuracy*: 0.5745 (approximately 57.45%)
  - *Test Accuracy*: 0.5796 (approximately 57.96%)
  - *Test Precision (weighted)*: 0.643 (indicates alignment of predictions with actual positive instances across classes)
  - *Test Recall (weighted)*: 0.580 (proportion of true positives correctly identified by the model)
  - *Test F1 Score (weighted)*: 0.583 (harmonic mean of precision and recall, providing a balanced metric)
- **Insights and Observations**:
  1) *Model Performance*:
     - The training and test accuracies are very close, indicating that the model generalizes well without overfitting to the training data.
     - Despite its simplicity, Naive Bayes achieves a weighted F1 score of approximately 58.3%, showing its potential for tasks with moderately complex data.
  2) *Advantages of the Naive Bayes Classifier*:
     - **Simplicity**: The model is computationally efficient and requires minimal parameter tuning, making it suitable for quick prototyping.
     - **Robustness to Small Datasets**: Performs well even with limited data due to its probabilistic nature.
     - **Handling Continuous Features**: Gaussian assumption allows effective handling of numerical data without the need for extensive preprocessing.
  3) *Challenges and Potential Issues*:
     - **Feature Independence Assumption**: The assumption of feature independence might oversimplify the data, reducing its ability to capture feature interactions.
     - **Sensitivity to Distribution Assumptions**: The Gaussian assumption might not hold true for all features, leading to suboptimal results in some cases.
- **Suggestions for Improvement**:
  1) *Feature Engineering*: Incorporating additional features or transforming existing ones (e.g., scaling or normalizing) might improve model performance.
  2) *Handling Dependencies*: Employing models that can handle feature dependencies, like Bayesian Networks or ensemble methods, could provide better results.
  3) *Alternative Distributions*: For features that do not follow a Gaussian distribution, consider using other variants like Multinomial Naive Bayes or Kernel Density Estimation.

D. XG Boosting

Description:
XGBoost (Extreme Gradient Boosting) is a powerful ensemble learning technique that combines multiple decision trees in a boosting manner. It is known for its scalability and efficiency in handling large datasets and complex feature interactions. The model iteratively improves by focusing on previously misclassified instances, making it robust for tasks like classification and regression.

- **Best Hyperparameters** (from *RandomizedSearchCV*):
  - *subsample*: 1.0 (uses the full dataset for each boosting round, which can capture maximum information from the data)
  - *reg_lambda*: 1 (ridge regularization to control overfitting by penalizing large weights)
  - *reg_alpha*: 0 (no lasso regularization applied)
  - *n_estimators*: 100 (number of boosting rounds)
  - *min_child_weight*: 1 (minimum sum of instance weight in a child, a regularization parameter to avoid overfitting)
  - *max_depth*: 15 (maximum depth of each tree, allowing for deeper trees to capture complex relationships)
  - *learning_rate*: 0.3 (controls the contribution of each tree to the final model, with a moderately high rate here for faster convergence)
  - *gamma*: 0 (no minimum loss reduction for splits, allowing for unrestricted splitting)
  - *colsample_bytree*: 0.6 (60% of features are used to build each tree, adding diversity to each tree in the ensemble)
- **Performance Metrics**:
  - *Training Accuracy*: 1.0 (100%)
  - *Test Accuracy*: 0.796 (approximately 79.56%)

  - *Test Precision (weighted)*: 0.795 (alignment of predictions with actual positive instances across classes)
  - *Test Recall (weighted)*: 0.796 (indicates the proportion of true positives correctly classified by the model)
  - *Test F1 Score (weighted)*: 0.795 (harmonic mean of precision and recall, providing a balanced metric)
- **Insights and Observations**
  1) *Model Performance*:
     - The model achieved a high training accuracy, suggesting it fit the training data well. However, test accuracy of 79.56% indicates a potential risk of overfitting since the model might be memorizing rather than generalizing.
     - XGBoost performed well across metrics, showing balanced precision, recall, and F1 scores. This suggests that the model is consistently accurate across the 'Credit_Score' classes.
  2) *Advantages of Hyperparameters*:
     - 'max_depth=15': Allows the model to capture complex patterns in the data.
     - 'learning_rate=0.3' and 'n_estimators=100': A moderately high learning rate with a sufficient number of estimators allowed quick convergence without needing a larger model.
     - 'colsample_bytree=0.6': Using only 60% of features per tree likely mitigates overfitting and enhances the model's generalizability.
  3) *Challenges and Potential Issues*:
     - Risk of Overfitting: The high training accuracy indicates that the model might have overfit to the training set. Reducing *max_depth* or adding regularization could be tested to improve generalization.
     - Complexity of the Model: XGBoost's ability to capture nuanced patterns means it might also be sensitive to noise in the dataset.

We can try experimenting with strong regularization (higher *reg_alpha* and *reg_lambda*) could help reduce overfitting, making the model less sensitive to noise. Perhaps we could also try lowering *max_depth* (e.g., 10 or 12) in order to reduce the model's sensitivity to noise while still capturing the primary patterns in the data. We improve on the feature engineering and selection part also.

E. ADA Boosting

Description: AdaBoost (Adaptive Boosting) is a robust ensemble learning method that combines multiple weak learners (typically shallow decision trees) to create a strong predictive model. By iteratively focusing on misclassified instances, AdaBoost assigns higher weights to difficult examples, improving overall model accuracy and robustness. It is particularly effective for classification tasks and is simple to implement.

- **Best Hyperparameters** (from *RandomizedSearchCV*):
  - **n_estimators**: 200 (Higher number of estimators allows the model to iteratively improve accuracy over more rounds).
  - *learning_rate*: 0.5 (Moderates the contribution of each weak learner, balancing between underfitting and overfitting).
- **Performance Metrics**:
  - *Test Accuracy*: 65.61
  - *Test Precision (weighted)*: 65.50% (Indicates how closely predictions align with actual classes).
  - *Test Recall (weighted)*: 65.61% Represents the proportion of true positives captured).
  - *Test F1 Score (weighted)*: 65.12% (Balances precision and recall, showing consistent performance across classes).
- **Insights and Observations**:
  1) *Model Performance:*
     - AdaBoost achieved moderate test accuracy (65.61%) compared to XGBoost (79.56%). The lower accuracy reflects AdaBoost's tendency to struggle with very complex datasets compared to gradient boosting methods.
     - Balanced precision, recall , and F1 scores indicate that the model performs consistently across the 'Credit Score' classes but may fail to capture the most complex patterns.
  2) *Advantages of Hyperparameters*:
     - *n_estimators* = 200 (A larger number of boosting rounds ensures the model can iteratively focus on challenging examples).
     - *learning_rate* = 0.5 (Provides a middle ground to ensure weak learners contribute meaningfully without dominating the ensemble).
  3) *Challenges and Potential Issues*:
     - *Risk of Underfitting:*
       The model, while consistent, has lower accuracy compared to XGBoost. AdaBoost's reliance on shallow weak learners may limit its capacity to capture highly non-linear relationships.
     - *Sensitivity to Noise:*
       Assigning higher weights to misclassified samples may amplify noise in the data, reducing overall performance.

F. K Nearest Neighbours

K-Nearest Neighbors (KNN) is a simple, non-parametric, and instance-based machine learning algorithm commonly used for classification and regression tasks. It works by identifying the `k` nearest data points (neighbors) to a query point and making predictions based on the majority class (for classification) or averaging the values (for regression) of these neighbors.

Selecting the Value of K: Scatter plots were created to identify an optimal value of k for achieving the best accuracy. In these plots, blue represents the 'Standard' class of the credit score, while red and green represent the 'Poor' and 'Good' classes, respectively.

The figure 13 shows the scatter plot for the variables Credit History Age (Y-axis) and Current Debt Outstanding (X-axis).

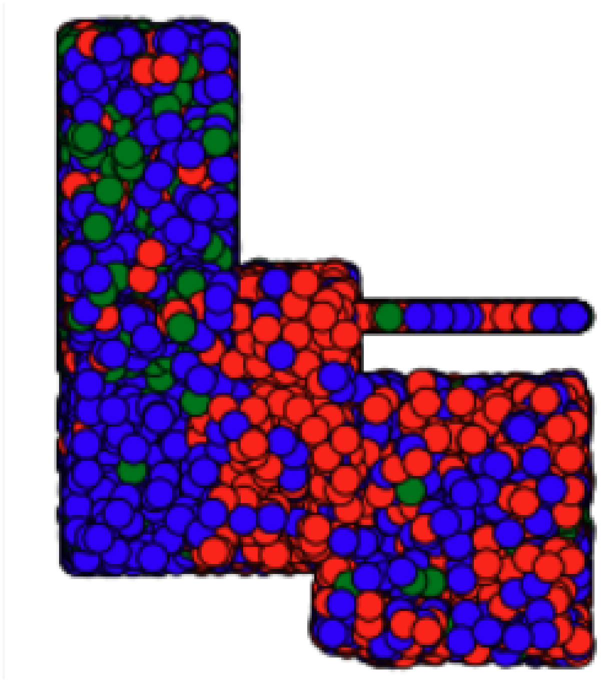


Fig. 12. Scatter Plot of Credit History Age vs Current Debt Outstanding

A similar approach was followed for multiple pairs of variables plotted against the categorical 'Credit Score' variable. These scatter plots helped visualize the degree of separation between clusters corresponding to the credit score classes. Better separation among the clusters is crucial for improving the accuracy of the K-Nearest Neighbor (KNN) algorithm. However, some variables exhibited poor separation of the red, green, and blue clusters.

For example, Figure 13 illustrates a scatter plot where the classes are not well-separated. This plot was generated with Credit History Age on the Y-axis and Rate of Interest on the X-axis. Poorly separated clusters like these posed challenges in achieving high accuracy with KNN.

Training the model: The model has been trained for k values between 2 and 21. The training score and testing score has been printed out. The test data taken here is based on the train_test_split from the Scikit learn's preprocessing library. The accuracy is based on the formula

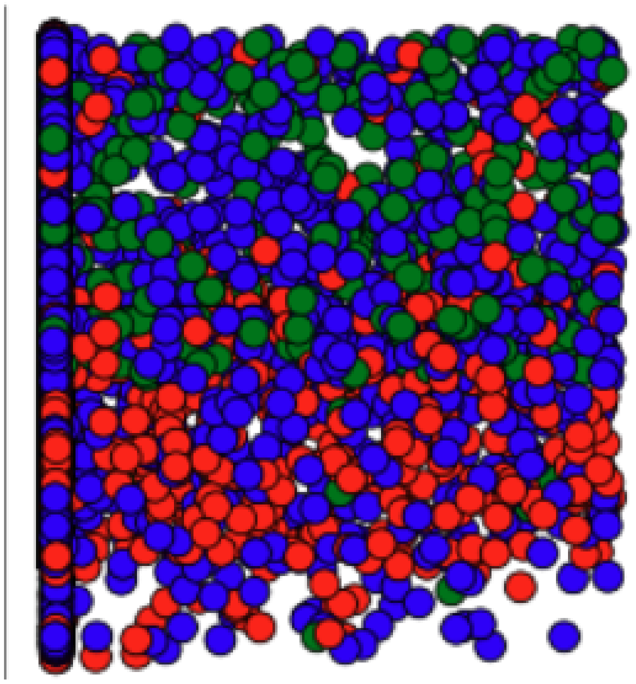


Fig. 13. Scatter Plot of Credit History Age vs Rate of Interest

Accuracy=Total Number of Predictions/Number of Correct Predictions

The training and testing accuracies were as follows: {2: [0.83865625, 0.641], 3: [0.82884375, 0.657625], 4: [0.792515625, 0.647], 5: [0.776109375, 0.64325], 6: [0.755765625, 0.6405], 7: [0.7408125, 0.636], 8: [0.726953125, 0.6321875], 9: [0.715453125, 0.6271875], 10: [0.70509375, 0.625125], 11: [0.69603125, 0.6239375], 12: [0.688921875, 0.6219375], 13: [0.68378125, 0.62025], 14: [0.677734375, 0.62025], 15: [0.675203125, 0.6229375], 16: [0.66965625, 0.6214375], 17: [0.666984375, 0.620625], 18: [0.663640625, 0.619125], 19: [0.66046875, 0.618], 20: [0.65771875, 0.618625]} The keys represent the values of k. The 1st element in the value array is the train accuracy, and the 2nd element in the value array is the test accuracy. The test accuracy was poor showing a maximum of 65.7625% where the k value is 3. So this model was not used for the final prediction as the other models performed a lot more better.

IV. PART 2

A. SVM

1) Brief Description of the Model: The model uses **Support Vector Machine (SVM)** with a radial basis function (RBF) kernel for classifying the credit score into three categories: 0, 1, and 2. SVM is a powerful supervised machine learning algorithm used for both classification and regression tasks. The RBF kernel maps data to a higher-dimensional space to handle non-linear relationships.

Model Details

- **Kernel**: RBF (Radial Basis Function)
- **Random State**: 42 (ensures reproducibility)
- **Evaluation Metrics**:
  - Accuracy
  - Precision, Recall, F1-score (from the classification report)

Performance Metrics

- **Training Accuracy**: 68.52%
- **Test Accuracy**: 66.31%
- **Classification Report**:
  - Class **0** (Precision: 59%, Recall: 47%, F1-score: 52%)
  - Class **1** (Precision: 71%, Recall: 56%, F1-score: 62%)
  - Class **2** (Precision: 66%, Recall: 79%, F1-score: 72%)

2) *Observations:* **Accuracy**

- The test accuracy of 66.31% means that the model has captured some patterns in the data but is still not highly reliable.
- There is slight overfitting as the training accuracy (68.52%) is higher than the test accuracy.

Imbalance in Precision and Recall

- For class **2**, precision (66%) and recall (79%) are relatively high, showing better identification of this class.
- For class **0**, both precision (59%) and recall (47%) are lower, indicating challenges in predicting this minority class correctly.

3) *Why is the Accuracy Low?:*

- **Class Imbalance**: The dataset has an imbalance in the target variable (**Credit_Score**). Class **2** has more instances (8469) compared to class **0** (2880) and class **1** (4651).
- **Feature Representation**: The relationship between features and the target variable may be complex, and the default SVM parameters might not fully capture these nuances, so we might need to find the optimal hyperparameters.
- **Model Parameters**: The default hyperparameters (e.g., **C**, **gamma**) might not be optimal for this dataset.

4) *Recommendations to Improve Accuracy:*

1) **Handle Class Imbalance**:
   - Use *oversampling* (e.g., SMOTE) to balance the minority classes or *undersampling* the majority class.
   - Assign **class_weights='balanced'** in SVM to penalize misclassification of minority classes.
2) **Hyperparameter Tuning**:
   - Perform *grid search* or *random search* for parameters like `C`, `gamma`, and kernel choice (e.g., `linear`, `poly`). However we did not do this since it is computationally very expensive and time consuming.
   - Use cross-validation for robust evaluation.
3) **Feature Engineering**:
   - Analyze feature importance and remove irrelevant or redundant features since it may be that we could still improve on the feature engineering part.
   - It may improve on creating derived features, such as ratios (e.g., **Income_Annual/Base_Salary_PerMonth**).
4) **Dimensionality Reduction**:
   - Using *PCA* or feature selection it may be possible to reduce noise and improve performance.
5) **Regularization of Features**:
   - Normalize the data (e.g., using *MinMaxScaler*) to enhance kernel function performance.
6) **Evaluation Beyond Accuracy**:
   - Focusing on metrics like F1-score, precision, and recall, especially for minority classes could also give better results.
   - Use a *confusion matrix* to analyze class-wise misclassifications.

Other Insights

- **Credit Utilization Ratio**: Features like **Ratio_Credit_Utilization** might strongly correlate with the target (**Credit_Score**). Analyze feature correlations to identify high-impact predictors.
- **Temporal Effects**: Investigate the role of **Month** and its interactions with other variables (e.g., seasonal trends).
- **Income and Debt**: Relationships between **Income_Annual**, **Current_Debt_Outstanding**, and **Credit_Limit** likely influence **Credit_Score**. Check for non-linear dependencies.

B. *Logistic regression*

1) Brief Description of the Model: The model uses various forms of **Logistic Regression** to predict the credit score categories. Logistic regression is a linear model for binary or multi-class classification problems. It predicts the probability of each class and applies different solvers and regularization techniques to improve performance.

Model Variants

- **Binary Logistic Regression**: Used for binary classification by splitting the target into two classes.
- **Multinomial Logistic Regression**: Handles multi-class classification problems directly.
- **Ordinal Logistic Regression**: Assumes an inherent order in the target classes.
- **L1-Regularized Logistic Regression**: Applies L1 regularization to induce sparsity in the model coefficients.

- **L2-Regularized Logistic Regression**: Applies L2 regularization to penalize large coefficients and reduce overfitting.
- **Elastic Net Logistic Regression**: Combines L1 and L2 regularization for a balanced approach.

2) Performance Metrics: **Accuracy for Each Model**

- **Binary Logistic Regression**:
  - Training Accuracy: 62.54%
  - Testing Accuracy: 62.39%
- **Multinomial Logistic Regression (solver: saga)**:
  - Training Accuracy: 62.55%
  - Testing Accuracy: 62.39%
- **Ordinal Logistic Regression**:
  - Training Accuracy: 53.45%
  - Testing Accuracy: 52.76%
- **L1-Regularized Logistic Regression**:
  - Training Accuracy: 62.56%
  - Testing Accuracy: 62.41%
- **L2-Regularized Logistic Regression**:
  - Training Accuracy: 62.54%
  - Testing Accuracy: 62.39%
- **Elastic Net Logistic Regression**:
  - Training Accuracy: 62.55%
  - Testing Accuracy: 62.41%

3) Observations: **Accuracy**

- All logistic regression models (except ordinal regression) achieve a similar test accuracy of approximately 62.39%.
- Ordinal regression performs significantly worse with a test accuracy of 52.76%, likely due to the assumption of ordinal relationships in the target that may not hold.
- L1 and Elastic Net regularizations slightly improve accuracy by reducing overfitting and selecting more relevant features.

Why is the Accuracy Low?

- **Class Imbalance**: The dataset has imbalanced classes, causing the model to underperform for minority classes.
- **Linear Relationships**: Logistic regression assumes linear relationships between features and the log-odds of the target class, which may not capture complex patterns.
- **Feature Representation**: Some features may lack sufficient explanatory power or require transformation.

4) Recommendations to Improve Accuracy:

1) **Handle Class Imbalance**:
   - Use oversampling (e.g., SMOTE) or undersampling techniques.
   - Assign class weights to penalize misclassification of minority classes.
2) **Feature Engineering**:
   - Create interaction terms and polynomial features to capture non-linear relationships.
   - Encode categorical variables effectively (e.g., one-hot encoding, target encoding).
3) **Regularization Tuning**:
   - Optimize the `C` parameter (inverse of regularization strength) for L1, L2, and Elastic Net models.
   - Perform a grid search to find the best combination of `L1_ratio` for Elastic Net.
4) **Dimensionality Reduction**:
   - Use PCA or feature selection to reduce noise in the dataset.
5) **Evaluation Beyond Accuracy**:
   - Focus on metrics like F1-score, precision, and recall for imbalanced datasets.
   - Use confusion matrices to analyze misclassification patterns.

5) Other Insights:

- **Regularization**: L1 and Elastic Net provide robustness by mitigating overfitting and improving feature selection.
- **Target Analysis**: The ordinal assumption may not hold for `Credit_Score`, as the differences between classes might not follow a strict order.
- **Feature Importance**: Investigate the impact of key features like `Ratio_Credit_Utilization` and `Credit_History_Age` on model predictions.
- **Comparison to SVM**: Logistic regression is computationally simpler but might not capture non-linear patterns as effectively as SVM with RBF kernel.

C. Neural Networks

1) *Brief description of the model* This model uses the Dense neural network architecture with a mixture of layers to produce the output or prediction. It makes use of different activation functions at every layer and a softmax function at the output layer as it is a 3 way classification problem. The output at jth neuron at the lth layer is given by

$$z_j^{(l)} = \sum_k w_{jk}^{(l)} a_k^{(l-1)} + b_j^{(l)}$$

$$a_j^{(l)} = activation(z_j^{(l)})$$

2) Model variations:
   - varitaion 1: Using sigmoid activation functions for intermediate layers and softmax for output layers. Dropout layer was added to prevent overfitting. Model was trained on 10 epochs and 0.001 learning rate. Optimiser used was Adam. Loss function uses is categorical crossentropy loss.
     The above model gave a test accuracy of 66.72% and F1 score of 66.08.
   - variation 2: Using the Relu activation function for hidden layers and softmax activation function for the output layer. Dropout layer was ignored in this variation since lesser neurons were present. Model was trained on 15 epochs and 0.001 learning rate.

Layer (type)	**Output Shape**	**Param #**
dense_3 (Dense)	(None, 48)	2,352
dropout_2 (Dropout)	(None, 48)	0
dense_4 (Dense)	(None, 24)	1,176
dropout_3 (Dropout)	(None, 24)	0
dense_5 (Dense)	(None, 3)	75

TABLE I
MODEL SUMMARY TABLE

Optimiser used was Adam. Loss function uses is categorical crossentropy loss.

Layer (type)	**Output Shape**	**Param #**
dense_9 (Dense)	(None, 48)	2,352
dense_10 (Dense)	(None, 24)	1,176
dense_11 (Dense)	(None, 3)	75

TABLE II
MODEL SUMMARY TABLE

Test accuracy=68.24% F1 Score=68.33%

- Variation 3: More layers were added in this variation. It used Relu and softmax activation function similar to variation 2. Optimiser used was Adam and Loss function was Categorical Cross entropy.

Layer (type)	**Output Shape**	**Param #**
dense_68 (Dense)	(None, 64)	3,136
dense_69 (Dense)	(None, 32)	2,080
dense_70 (Dense)	(None, 16)	528
dense_71 (Dense)	(None, 8)	136
dense_72 (Dense)	(None, 3)	27

TABLE III
MODEL SUMMARY TABLE

Test accuracy= 68.306% F1 score=68.585%