Institute of Engineering & Technology (IET)

J.K. Lakshmipat University

CS1138: Machine Learning

Credit Risk Assessment Report

Faculty Guide:

Arpan Gupta

Submitted by:

Group 3(Section- A)

Adhiraj Singh (2022btech005)

Ashmit Sharma (2022Btech022)

Divyanshu Gautam (2022Btech034)

Kanishk Tiwari (2022Btech045)

INDEX

Introduction

In today's world, lending plays a crucial role in keeping our economies running smoothly. But it's not all smooth sailing. One big challenge we face is figuring out if borrowers can actually pay back their loans. As banks and other financial institutions try to lend money responsibly while still helping people access the funds they need, they're starting to question if the old-fashioned ways of assessing credit are good enough.

Our goal is to use the smarts of machine learning to build a tool that can predict if someone might not be able to repay a loan. We're going to dig into all the little details about borrowers—like their job, income, and past financial behavior—as well as details about the loans themselves. By doing this, we hope to uncover all the things that could point to someone having trouble repaying a loan.

By using data to get a better understanding of who might struggle with repayments, we want to give lenders a powerful new way to make decisions. Our aim is to make borrowing and lending safer and smarter for everyone involved.

Problem Statement:

In our credit risk assessment project, our main goal is to predict if a borrower is likely to default on a loan. To do this, we're analyzing a bunch of data about borrowers and loans. Our aim is to build a model that helps lenders make smarter decisions about who to lend money to.

Our approach includes a few important steps:

1. **Balancing the Dataset:** We know that there are more cases of people repaying loans than defaulting. So, we're working on making sure our dataset has a fair balance of both types of cases. This helps our model learn better and be fairer.
2. **Trying Different Machine Learning Models:** Instead of just using one method to build our model, we're trying out a bunch of different ones. We're testing methods like Support Vector Machines, k-Nearest Neighbors, Artificial Neural Networks, Random Forest, Decision Trees, and Gradient Boosting to see which works best for our data.
3. **Researching and Choosing Features:** Before we even start building our model, we're spending time researching which features (like a borrower's job or income) are most important for predicting loan defaults. We're only including the most useful features in our model to make sure it's accurate and easy to understand.
4. **Exploring the Data:** We're digging deep into our dataset to understand it better. By looking closely at the different parts of our data, we can spot patterns and trends that help us build a better model.
5. **Comparing with Other Research:** We're not working in a vacuum. We're looking at what other researchers have done in this area and comparing our results with theirs. This helps us make sure our model is doing a good job and maybe even improving on what's been done before.

With these steps, we hope to build a model that's not only better than what's out there but also helps lenders make fairer and more informed decisions about lending money.

Literature Review

RESEARCH PAPER-1 (K-nearest-neighbor classifier)

Name: Bank credit risk analysis with k-nearest neighbor classifier: Case of Tunisian banks
Issue-1, Volume-14, 2015

Analysis:

- Research paper shows the application and usage of KNN classifiers in assessing a person's credit risk.
- Techniques implemented on 924 credit files with multidimensional data of Tuisian bank from 2003 to 2006, with a focus on short-term commercial credits.
- The goal is to develop models capable of determining whether to approve or decline a credit application with minimum error rates, often by using Larger number of k.
- Those attributes in the Dataset are chosen by observing linear regression models between many of them.
- Selected Euclidean metric an adjusted version of the metric that contours for class membership. Also had a data-dependent metric.
- Financial criteria such as the short term debt to sales ratio, financial costs to revenue ratio, fixed asset turnover collateral, and company size are used to identify organizations as dangerous or healthy.
- For k=3, the best model, which links accrual and cash flow data, had a respectable classification rate of 88.63%.
- Additionally, a (ROC) curve is shown to evaluate the model's performance, and it meets the 95.6% Area Under Curve (AUC) threshold.

Conclusion:
Identify Defaulter on the basis of Cash flow features and Non-cash flow features, at last comparing it to Overall Info given:

Table 6. Criterion of the type I and II error

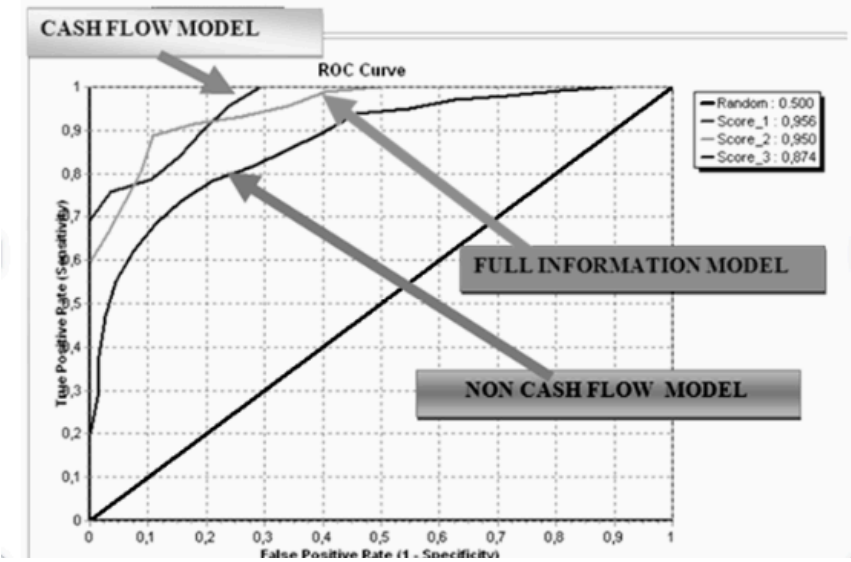
	ERROR	K=2	K=3	K=4	K=5
NON CASH FLOW MODEL	Type I	**21.83%**	**16.73%**	**27.51%**	**27.25%**
	Type II	**21.45%**	**20.52%**	**26.18%**	**27.72%**
CASH FLOW MODEL	Type I	**12.66%**	**12.01%**	**15.45%**	**19.74%**
	Type II	**13.75%**	**10.69%**	**15.50%**	**18.12%**
FULL INFORMATION MODEL	Type I	**14.8%**	**14.80%**	**21.24%**	**24.24%**
	Type II	**14.19%**	**11.35%**	**16.81%**	**16.37%**

Moreover, they have shown variances of classification of a person into good and bad through increasing values of k alongside their company's performance through the past.

Table 5. Results for full information models (Appendix3 panels 1, 2, 3 and 4)

	K=2		K=3		K=4		K=5	
	Healthy	**Risky**	**Healthy**	**Risky**	**Healthy**	**Risky**	**Healthy**	**Risky**
Healthy companies	**393**	**65**	**406**	**52**	**381**	**77**	**383**	**75**
Risky companies	**69**	**397**	**69**	**397**	**99**	**367**	**113**	**353**
% Total Good and Bad Classification								
Good classification	**85.5%**		**86.90%**		**80.95%**		**79.65%**	
Bad classification	**14.50%**		**13.10%**		**19.05%**		**20.35%**	

Figure 2. ROC curve of three models



According to studies, accrual and cash flow data provided the greatest information set for evaluating credit risk, with a high categorization rate attained. Additionally, the model's performance was assessed using the Area Under Curve criteria; the top model utilizing cash flow data had an AUC of 95.6%, according to reports.

RESEARCH PAPER 2 – (SVM)

About: Presented ICCGANT 2020 conference

Publisher: Journal of Physics: Conference Series Volume 1836 in 2021 by IOP Publishing Ltd.

Introduction**:** For credit risk analysis and classification of customers into good or bad credit classes, machine learning techniques such as Support Vector Machines SVM may be used.

Features:

1. Gender
2. Plafond (loan amount/limit)
3. Rate (interest rate)
4. Term of time (loan tenure/duration)
5. Job
6. Income
7. Face amount
8. Warranty (collateral type)
9. Loan history
10. Credit status (target/dependent variable)

Analysis**:**

1. Suggested categorizing consumers into excellent or bad credit classes using the Support Vector Machines (SVM) algorithm for credit risk analysis.
2. SVM determines the best hyperplane to divide data into classes with the biggest possible margin.
3. Used four different kernel functions (linear, polynomial, RBF, sigmoid) with SVM to handle non-linear data.
4. Dataset consisted of 610 data points from Bank XX, with variables like gender, income, loan history, and credit status.
5. Performed data preprocessing steps: cleaning, type checking, and normalization of the dataset.
6. Split dataset into 80% for training and 20% for testing.
7. Built SVM models using training data with each of the four kernel functions.
8. Confusion matrix, accuracy, sensitivity, specificity, precision, F1-score, FPR, FNR, and AUC were among the metrics used to evaluate the models.
9. SVM model with polynomial kernel performed best, with highest accuracy (0.9508) and AUC (0.9419) on testing data.
10. Polynomial kernel model had low FPR (0.0421) and FNR (0.0741), indicating low misclassification rates, and can assist Bank XX in credit decisions.

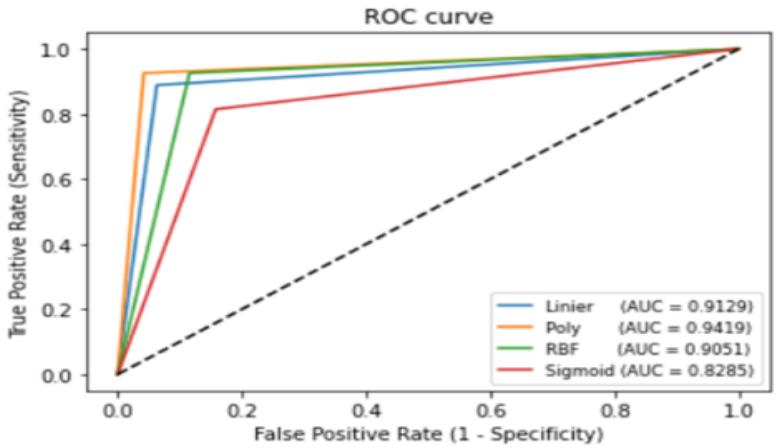


Figure 4. ROC Curves of SVM.

Conclusion:

- All SVM models were good at classifying data, but the polynomial kernel model was the best.
- By classifying customers as good or bad credit classes, the polynomial kernel SVM model can help Bank XX to accept or reject a loan application.
- The risk of losses due to bad debt can be reduced through this model.

RESEARCH PAPER-3

Techniques by Varsha Aithal, Roshan David Jathanna)

Name: International Journal of Innovative Technology and Exploring Engineering (IJITEE)

ISSN: 2278-3075 (Online), Volume-9 Issue-1, November 2019

Dataset Used:

ATTRIBUTE NUMBER	DESCRIPTION	CLASS
1)	Creditability	Categorical
2)	Account Balance	Categorical
3)	Credit length (in months)	Numeric
4)	Status of payment	Categorical
5)	Purpose	Categorical
6)	Credit Amount	Numeric
7)	Savings in cost	Categorical
8)	Current employment period	Categorical
9)	Installment	Numeric
10)	Sex and Marital Status	Categorical
11)	Guarantors	Categorical
12)	Current address duration	Numerical
13)	Most precious resources	Categorical
14)	Lifespan	Numeric
15)	Simultaneous loans	Categorical
16)	Type of house	Categorical
17)	Amount of loans from this bank	Numeric
18)	Employment	Categorical
19)	Number of dependents	Categorical
20)	Telephone	Categorical
21)	Foreign Workers	Categorical

Analysis:

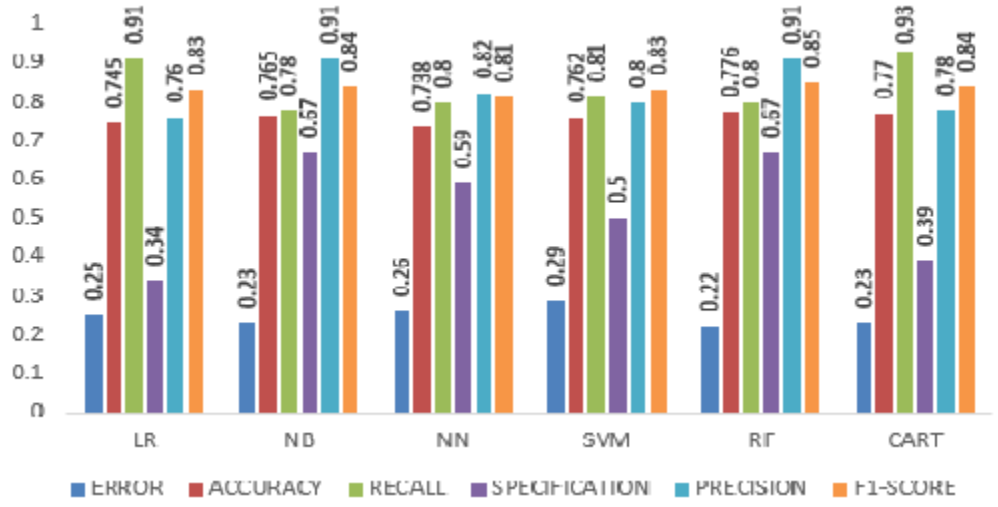
- Research paper presents a comparison of various machine learning techniques for evaluating credit risk.
- Techniques implemented on the **German credit** dataset from the **UCI repository**.
- Dataset comprises 1000 rows and 21 columns.
- The objective is to train models to accept or reject credit profiles based on attributes.
- Machine learning algorithms compared:
    - **Support Vector Network**
    - **Neural Network**
    - **Logistic Regression**
    - **Naive Bayes**
    - **Random Forest**
    - **Classification and Regression Trees (CART) algorithm**
- Results indicate that the Random Forest algorithm outperformed others in predicting credit risk with higher accuracy

Result:

Evaluation metrics of the model:

	ERR	ACC	REC	SP	PREC	F1-S
LR	0.25	0.75	0.91	0.34	0.76	0.83
NB	0.23	0.77	0.78	0.67	0.91	0.84
NN	0.26	0.74	0.80	0.59	0.82	0.81
SVN	0.29	0.76	0.81	0.50	0.80	0.83
RF	**0.22**	**0.78**	0.80	**0.67**	**0.91**	**0.85**
CART	0.23	0.77	**0.93**	0.39	0.78	0.84

Graph Comparing the Accuracy, Recall, Precision and F1 Score of all the models implemented.



Conclusion and Future Scope of paper:

Results indicate that the Random Forest methodology yields higher accuracy in credit risk evaluation compared to other techniques. The authors suggest that future work could involve evaluating deep learning techniques to potentially further improve accuracy in credit risk assessment.

RESEARCH PAPER-4 (EVALUATING CREDIT RISK USING ARTIFICIAL NEURAL NETWORKS)

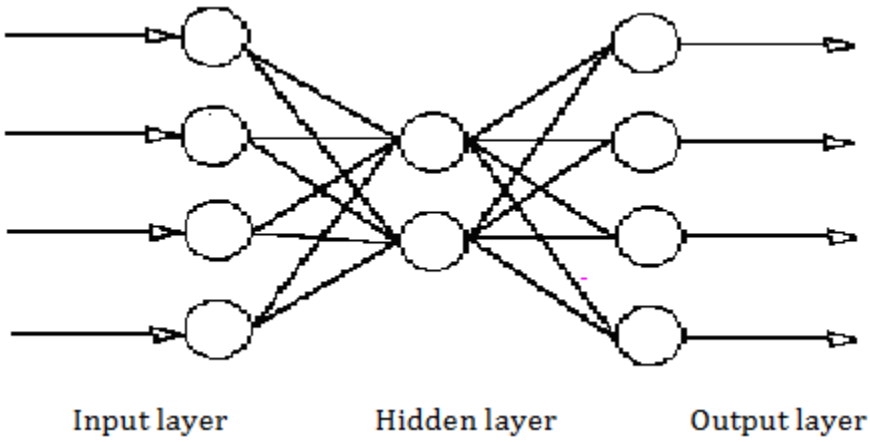
Name:QEETHARA K. AL-SHAYEA1, and GHALEB A. EL-REFAE

Dataset Used:

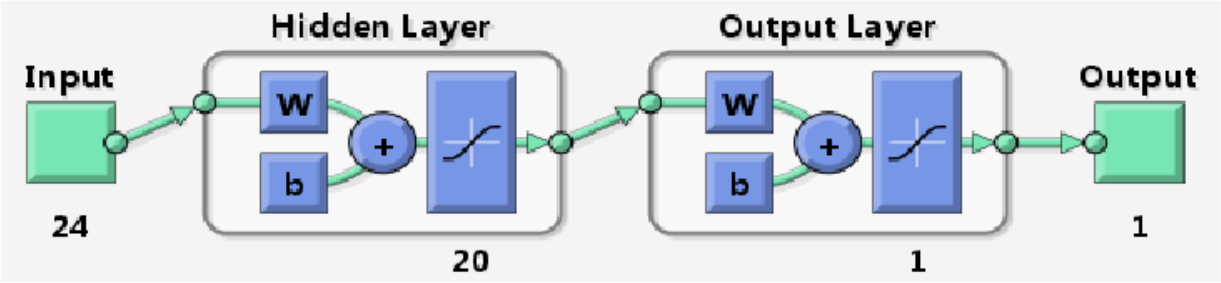
Dataset	No. of samples	Classes
German	1000	700 good ; 300 bad
Australian	690	307 good; 383 bad

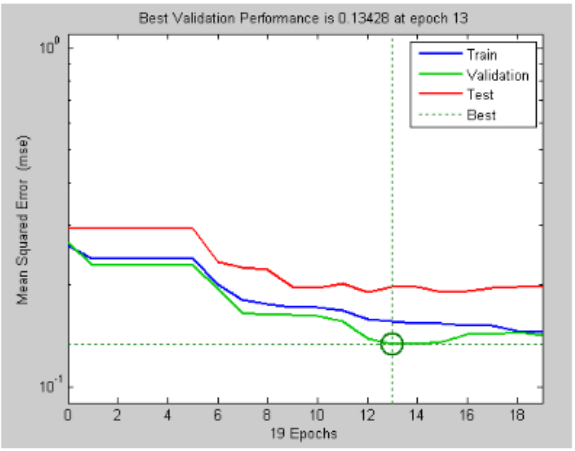
Analysis:

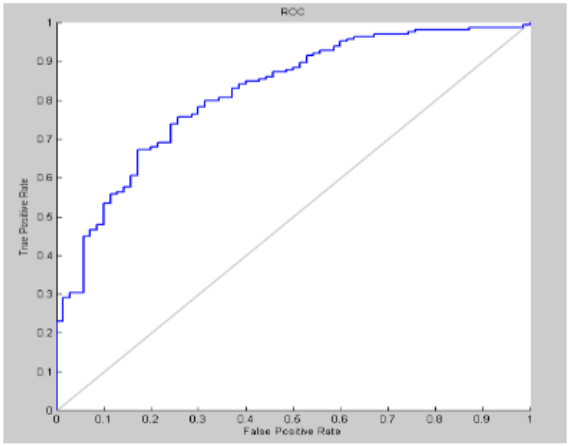
The study evaluates the performance of feed-forward back propagation neural networks trained on two different datasets - German and Australian. In both cases, a two-layer network with sigmoid hidden neurons is employed, utilizing scaled conjugate gradient backpropagation for training.



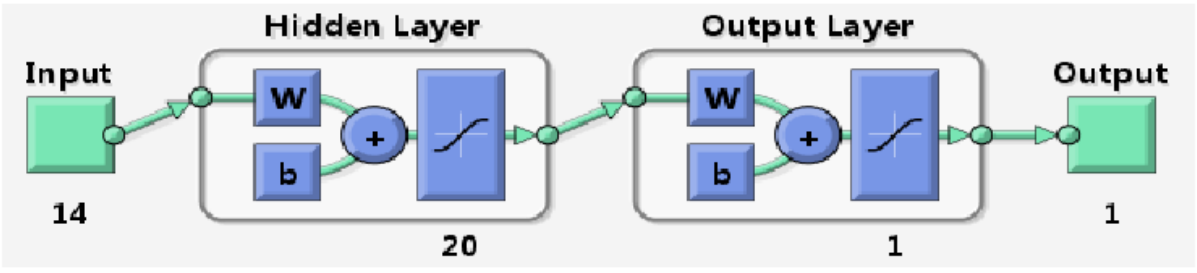
For the German dataset, the network achieved a training set classification accuracy of 80.4%, with a validation performance of 0.13428 at epoch 13. The ROC curve demonstrated a 76.6% correct classification rate in the simulation sample.

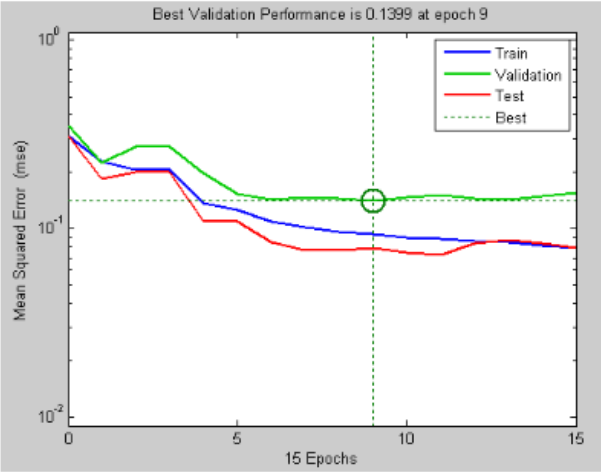


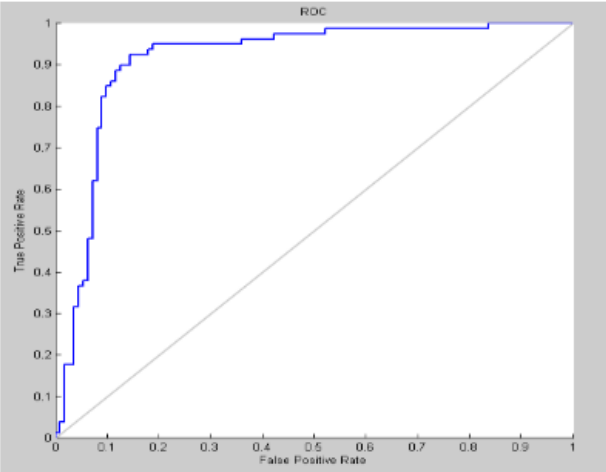




In contrast, the Australian dataset yielded even better results, with a training set classification accuracy of 89.7% and a validation performance of 0.024484 at epoch 15. The confusion matrices illustrated high numbers of correct responses and low numbers of incorrect responses, indicating a robust performance. The ROC curve for the simulation sample showed an approximately 86% correct classification rate.







Overall, both networks exhibited strong abilities to learn patterns and classify applicants based on selected parameters. The Australian dataset, in particular, demonstrated higher accuracy and validation performance compared to the German dataset, indicating potential variations in dataset complexity or suitability for the task.

Conclusion:

The study tested the feed-forward back propagation neural network with supervised learning for classifying credit risk as worthy or not worthy. The results demonstrated significant effectiveness in classification, with artificial neural networks showcasing notable performance. This was further evidenced by the Receiver Operating Characteristic (ROC) curve, which closely represented the classifier's performance.

Proposed Methodology:

1. **k-Nearest Neighbors (KNN):** This model predicts the credit risk of a borrower based on the similarity of its neighbors in the feature space. It calculates the creditworthiness of a borrower by comparing it with the credit histories of its k nearest neighbors.
2. **Support Vector Machines (SVM):** SVM is a powerful algorithm used for classification tasks. It works by finding the hyperplane that best separates the data points of different classes in the feature space. In our case, SVM aims to classify borrowers into two categories: likely to default or not likely to default.
3. **Artificial Neural Networks (ANN):** ANN is a computational model inspired by the structure and functioning of the human brain. It consists of interconnected nodes organized in layers, including input, hidden, and output layers. ANN learns from the data to make predictions and can capture complex relationships between input features and the target variable.
4. **Random Forest:** Random Forest is an ensemble learning method that combines multiple decision trees to improve predictive accuracy and reduce overfitting. Each tree in the forest is trained on a random subset of the data, and the final prediction is made by averaging the predictions of all trees.
5. **Principal Component Analysis (PCA):** PCA is not a predictive model but a dimensionality reduction technique. It transforms the original features into a lower-dimensional space while preserving the most important information. By reducing the dimensionality of the data, PCA can improve computational efficiency and reduce the risk of overfitting when used in conjunction with predictive models.
6. **Decision Tree:** Decision trees are simple yet powerful models for classification tasks. They partition the feature space into regions based on the values of input features and make predictions by following the decision path from the root to the leaf nodes of the tree.
7. **Gradient Boosting:** Gradient Boosting is another ensemble learning method that builds a strong predictive model by combining multiple weak learners, typically decision trees. Unlike Random Forest, which builds trees independently, Gradient Boosting builds trees sequentially, with each tree attempting to correct the errors made by the previous ones.

By employing these diverse models, we aimed to explore different approaches to credit risk assessment and identify the most effective ones for our specific dataset and task. Through thorough evaluation and comparison, we sought to select the models that

provide the highest predictive accuracy and reliability, ultimately empowering lenders to make more informed lending decisions.

Experiments:

Data cleaning

Original dataset

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_status	loan_percent_
0	22	59000	RENT	123.0	PERSONAL	D	35000	16.02	1	
1	21	9600	OWN	5.0	EDUCATION	B	1000	11.14	0	
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.87	1	
3	23	65500	RENT	4.0	MEDICAL	C	35000	15.23	1	
4	24	54400	RENT	8.0	MEDICAL	C	35000	14.27	1	
...	...	...	...	...	...	...	...	...	...	
32576	57	53000	MORTGAGE	1.0	PERSONAL	C	5800	13.16	0	
32577	54	120000	MORTGAGE	4.0	PERSONAL	A	17625	7.49	0	
32578	65	76000	RENT	3.0	HOMEIMPROVEMENT	B	35000	10.99	1	
32579	56	150000	MORTGAGE	5.0	PERSONAL	B	15000	11.48	0	
32580	66	42000	RENT	2.0	MEDICAL	B	6475	9.99	0	

32581 rows × 12 columns

Encoded dataset

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_status	loan_percent_income
0	22	59000	3	123.0	4	3	35000	16.02	1	0.59
1	21	9600	2	5.0	1	1	1000	11.14	0	0.10
2	25	9600	0	1.0	3	2	5500	12.87	1	0.57
3	23	65500	3	4.0	3	2	35000	15.23	1	0.53
4	24	54400	3	8.0	3	2	35000	14.27	1	0.55

Encoded Columns

```
person_home_ownership:
'RENT' in df --> '3'
'OWN' in df --> '2'
'MORTGAGE' in df --> '0'
'OTHER' in df --> '1'
loan_intent:
'PERSONAL' in df --> '4'
'EDUCATION' in df --> '1'
'MEDICAL' in df --> '3'
'VENTURE' in df --> '5'
'HOMEIMPROVEMENT' in df --> '2'
'DEBTCONSOLIDATION' in df --> '0'
loan_grade:
'D' in df --> '3'
'B' in df --> '1'
'C' in df --> '2'
'A' in df --> '0'
'E' in df --> '4'
'F' in df --> '5'
'G' in df --> '6'
cb_person_default_on_file:
'Y' in df --> '1'
'N' in df --> '0'
```

Null Values taken care of

```
df_encoded.isnull().sum()
```

✓ 0.0s

```
person_age                      0
person_income                   0
person_home_ownership           0
person_emp_length             895
loan_intent                     0
loan_grade                      0
loan_amnt                       0
loan_int_rate                3116
loan_status                     0
loan_percent_income             0
cb_person_default_on_file       0
cb_person_cred_hist_length      0
dtype: int64
```

```
#Filling NA/missing values
df_encoded.fillna(df_encoded["loan_int_rate"].mean(),inplace=True)
df_encoded.fillna(df_encoded["person_emp_length"].mean(),inplace=True)
```

✓ 0.0s

Data exploration

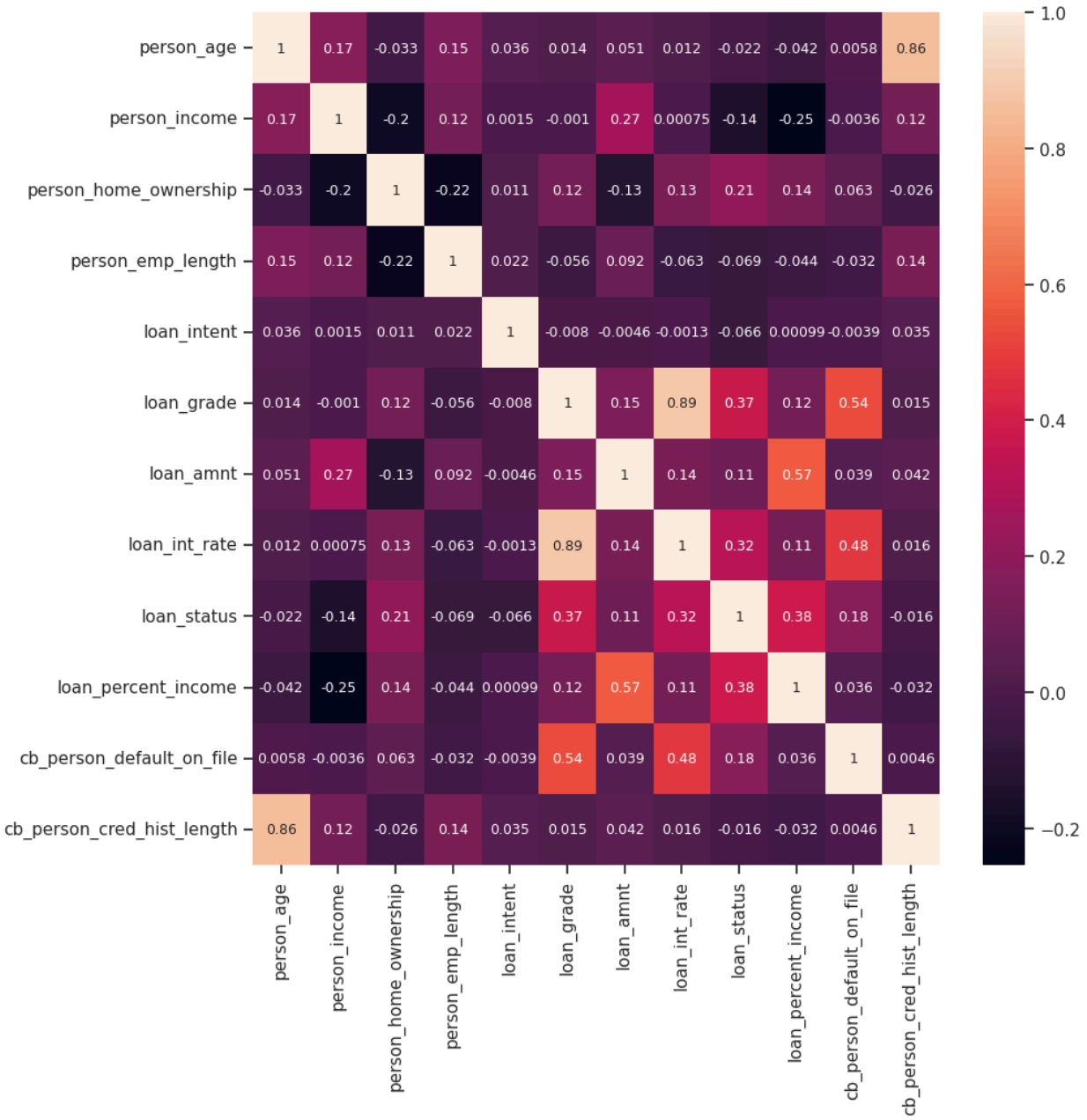
Information

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_status
count	32581.000000	3.258100e+04	32581.000000	31686.000000	32581.000000	32581.000000	32581.000000	29465.000000	32581.000000
mean	27.734600	6.607485e+04	1.676222	4.789686	2.533839	1.218195	9589.371106	11.011695	0.218164
std	6.348078	6.198312e+04	1.433116	4.142630	1.731182	1.166336	6322.086646	3.240459	0.413006
min	20.000000	4.000000e+03	0.000000	0.000000	0.000000	0.000000	500.000000	5.420000	0.000000
25%	23.000000	3.850000e+04	0.000000	2.000000	1.000000	0.000000	5000.000000	7.900000	0.000000
50%	26.000000	5.500000e+04	3.000000	4.000000	3.000000	1.000000	8000.000000	10.990000	0.000000
75%	30.000000	7.920000e+04	3.000000	7.000000	4.000000	2.000000	12200.000000	13.470000	0.000000
max	144.000000	6.000000e+06	3.000000	123.000000	5.000000	6.000000	35000.000000	23.220000	1.000000

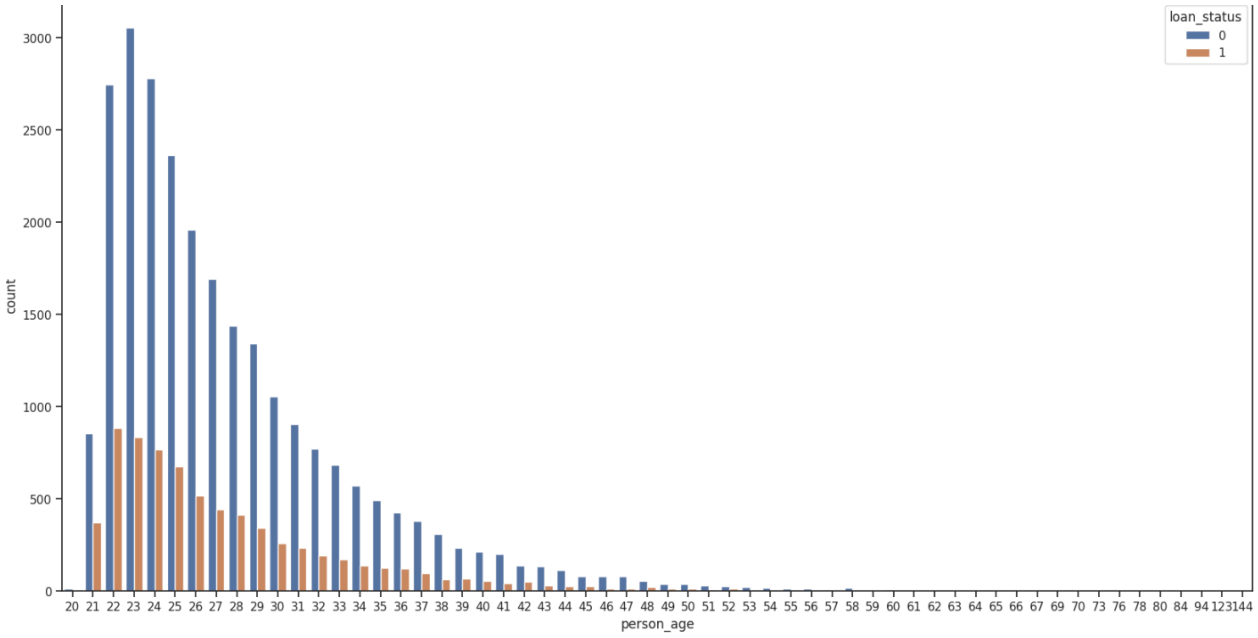
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   person_age                  32581 non-null  int64
 1   person_income               32581 non-null  int64
 2   person_home_ownership       32581 non-null  int64
 3   person_emp_length           31686 non-null  float64
 4   loan_intent                 32581 non-null  int64
 5   loan_grade                  32581 non-null  int64
 6   loan_amnt                   32581 non-null  int64
 7   loan_int_rate               29465 non-null  float64
 8   loan_status                 32581 non-null  int64
 9   loan_percent_income         32581 non-null  float64
 10  cb_person_default_on_file   32581 non-null  int64
 11  cb_person_cred_hist_length  32581 non-null  int64
dtypes: float64(3), int64(9)
memory usage: 3.0 MB
```

Correlation

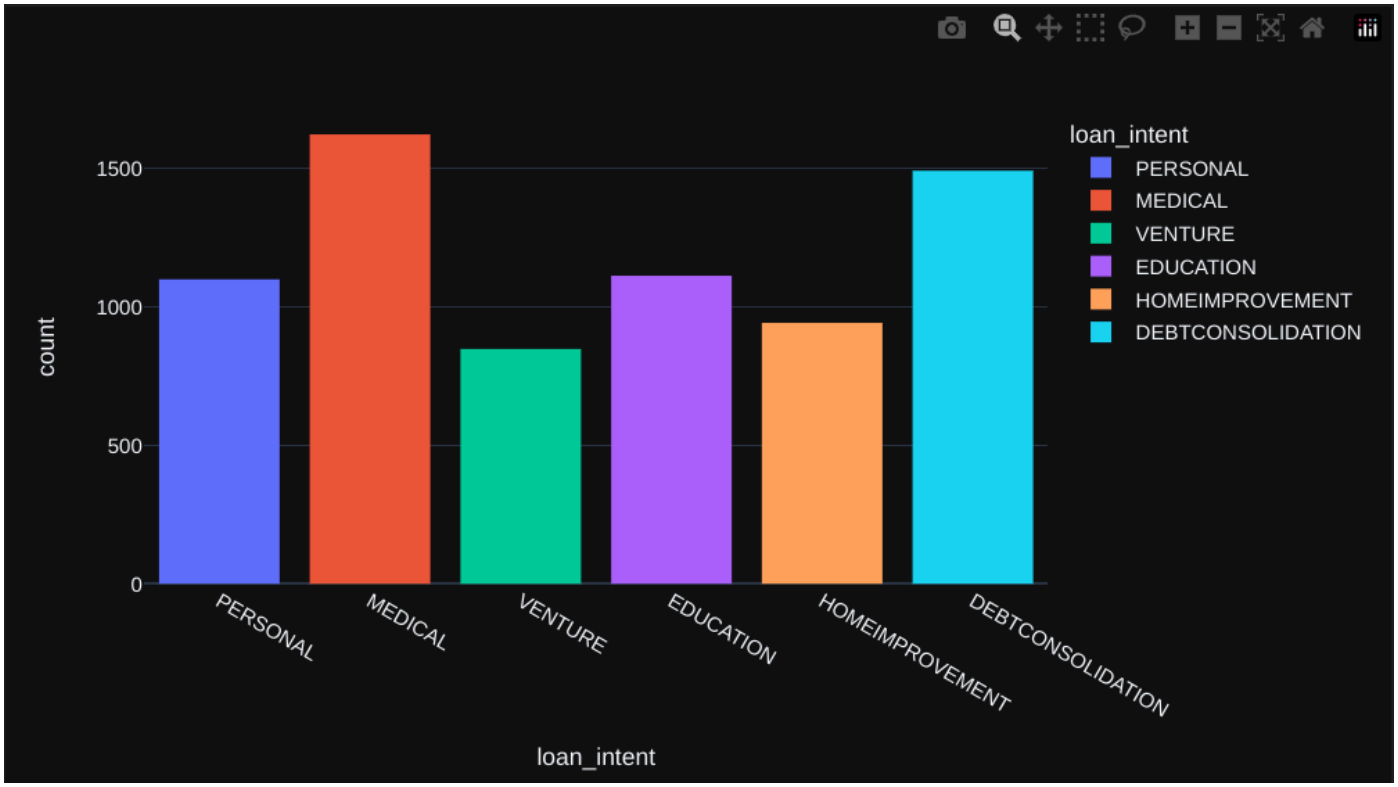
By this we can examine our informed decisions about which variables to include in predictive models, identify redundant variables, and gain insights into the underlying structure of the dataset. Identifying correlations between certain borrower attributes and default rates can help in building more accurate predictive models.



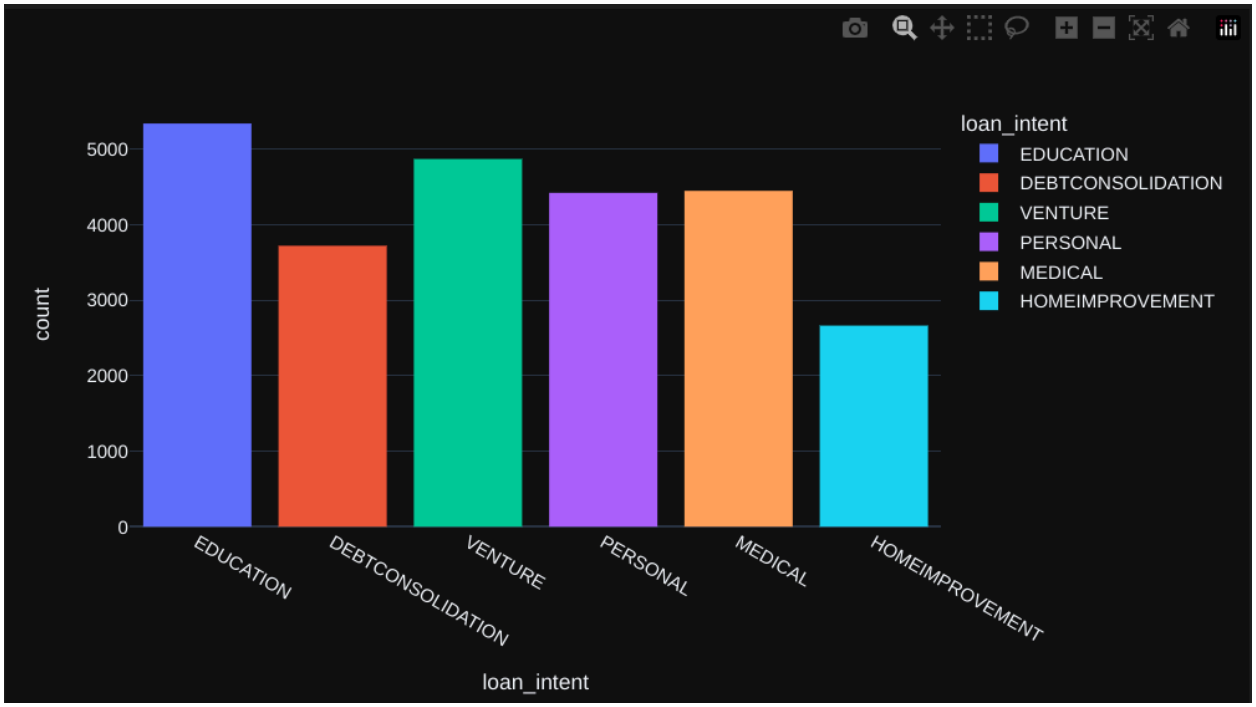
We notice a trend where younger individuals tend to default on their loans more frequently, with a higher proportion of defaults among this demographic.

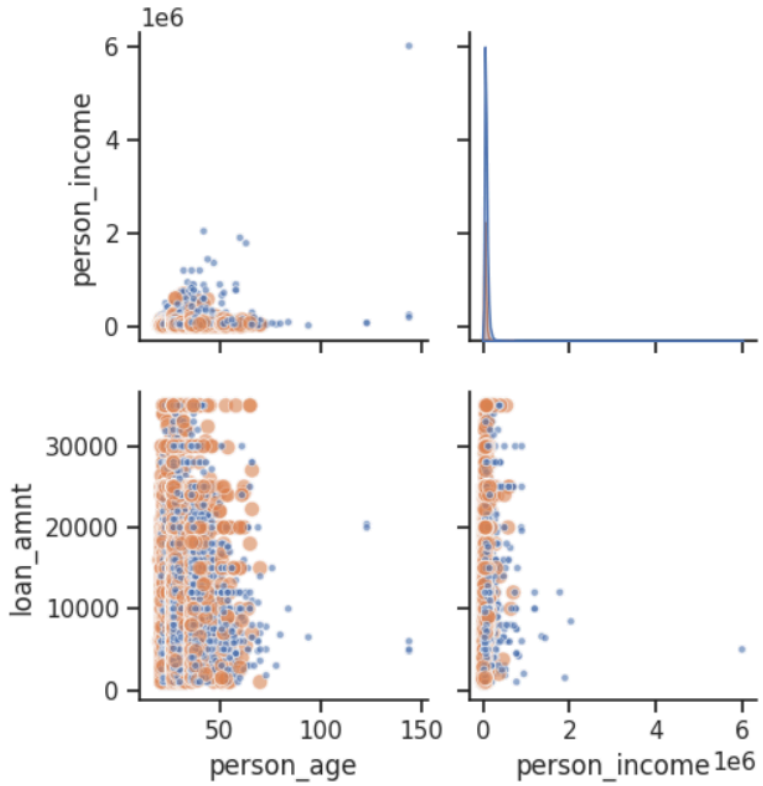


People most likely to default on the loan are the youngest, and the biggest expense on loans is for medical expenses. One of the reasons may be that many do not have health insurance and, in an emergency, end up borrowing money.



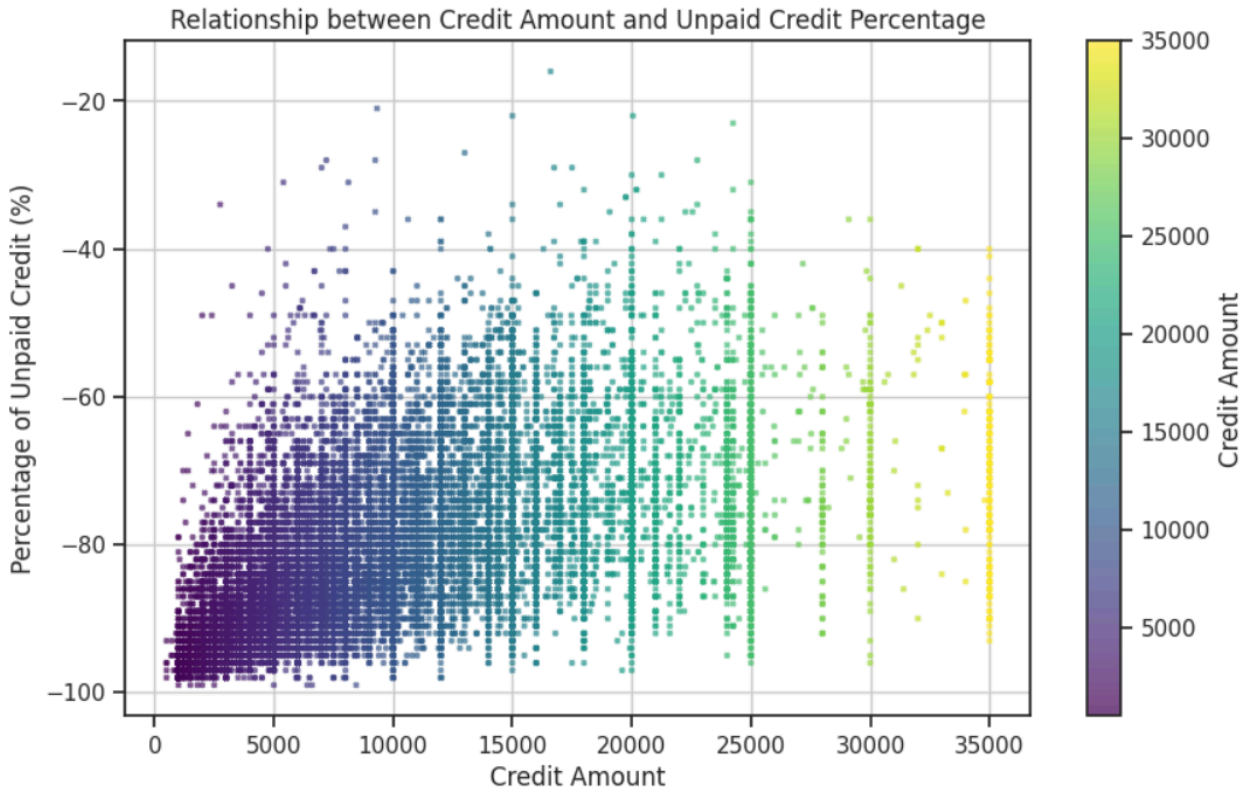
In this graph we can see that those who pay the loan used the amount to pay the student loan, hence the education factor ends up being an interesting one for the repayment.



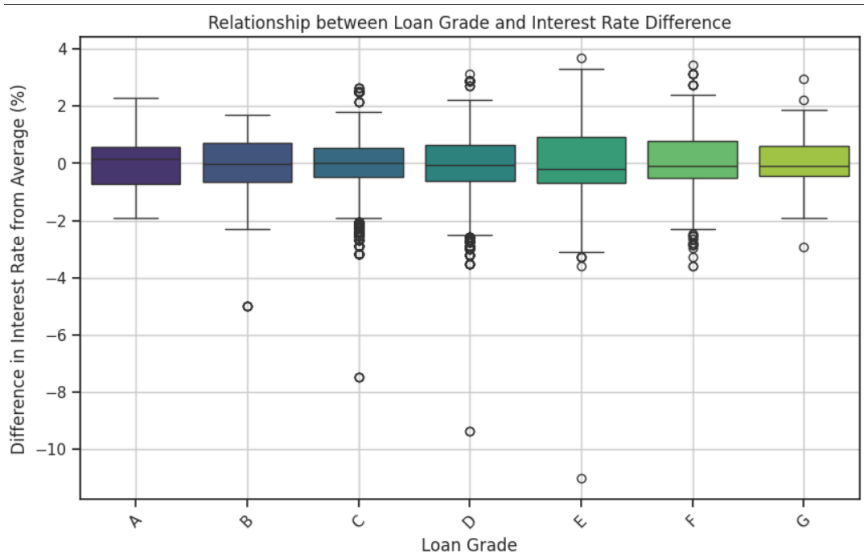


Person age to person income shows, adults as in 40s, always have more income then any age group around and Loan amt to Person age shows, the growing, teens and adults are the only two in need for the loans, as for the upcoming future.

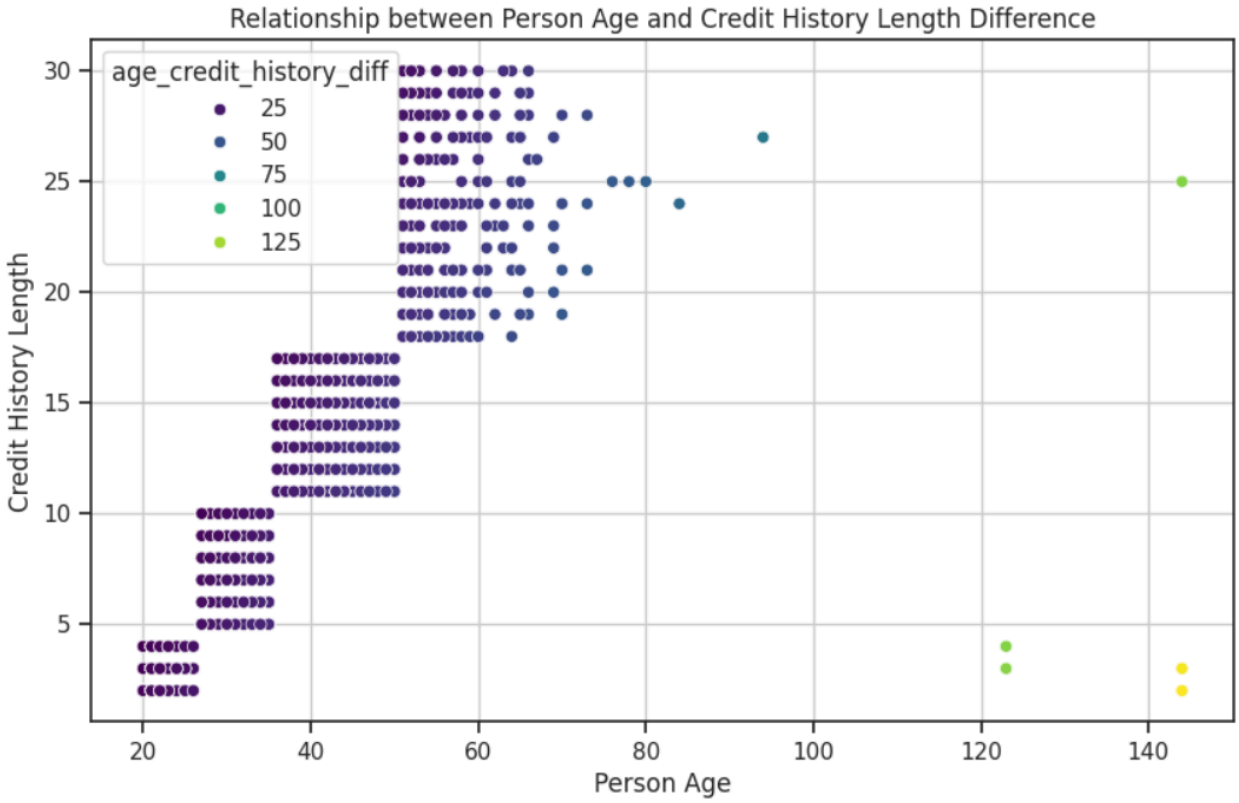
Positive Trend: The linear regression line indicates a positive trend, suggesting that as the credit amount increases, the percentage of unpaid credit also tends to increase. This implies that borrowers with higher loan amounts may have a higher percentage of unpaid credit, indicating potential financial strain or difficulty in managing existing debts relative to their income.



Grade B,C,D Loans: Tend to have interest rates closer to the average rate for these grade loans, with fewer outliers.
Grade E Loans: Loans with grade E might have a wider spread of interest rates compared to other grades, indicating higher variability, leading to higher risks.



Positive differences b/w age and credit history indicate individuals who have had credit accounts for a longer time relative to their age, suggesting a history of responsible being. On the other hand, negative differences may indicate individuals who have recently established credit accounts or have shorter credit histories relative to their age. Individuals with unusually long or short credit histories relative to their age group bring further risk to their credit management.



Models Trained

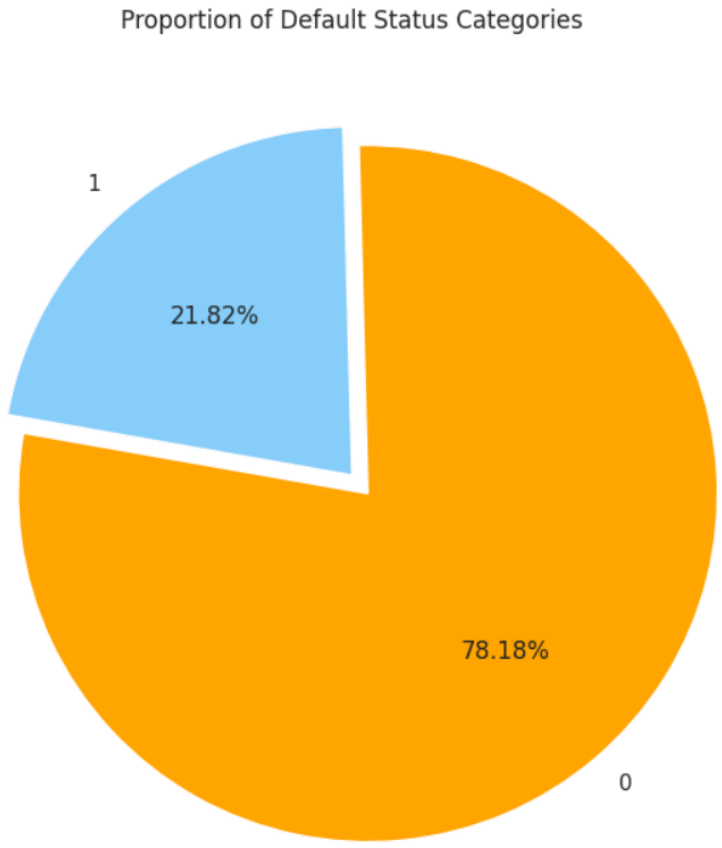
Loan_status is the target variable for all the classifiers to be applied on the dataset.

```
X = df_encoded.drop(columns=['loan_status'])
Y = df_encoded['loan_status']
```
✓ 0.0s

Training and test sets

```
Training set shapes:
X_train: (26064, 11)
Y_train: (26064,)

Test set shapes:
X_test: (6517, 11)
Y_test: (6517,)
```

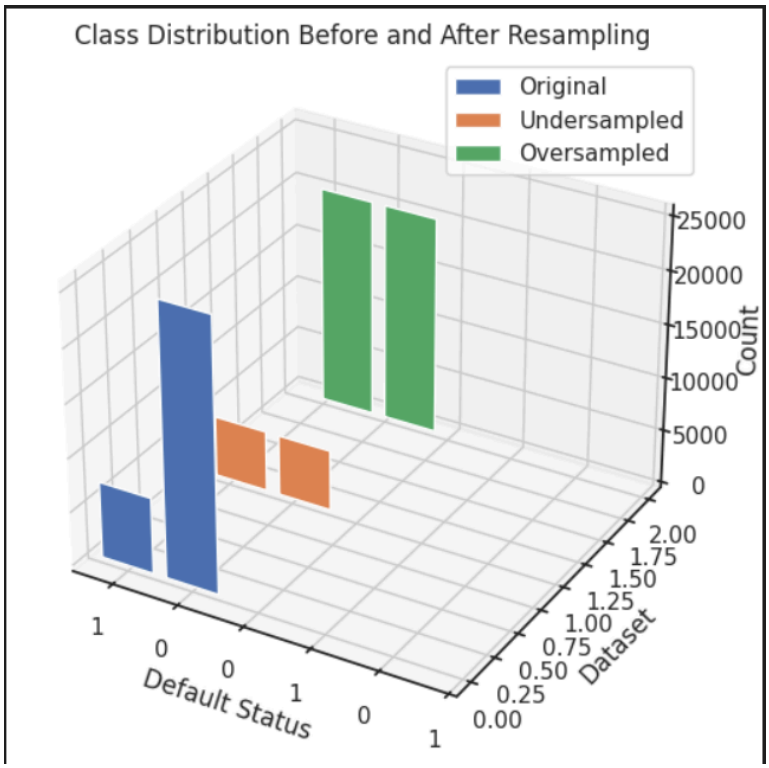
Imbalance in the dataset



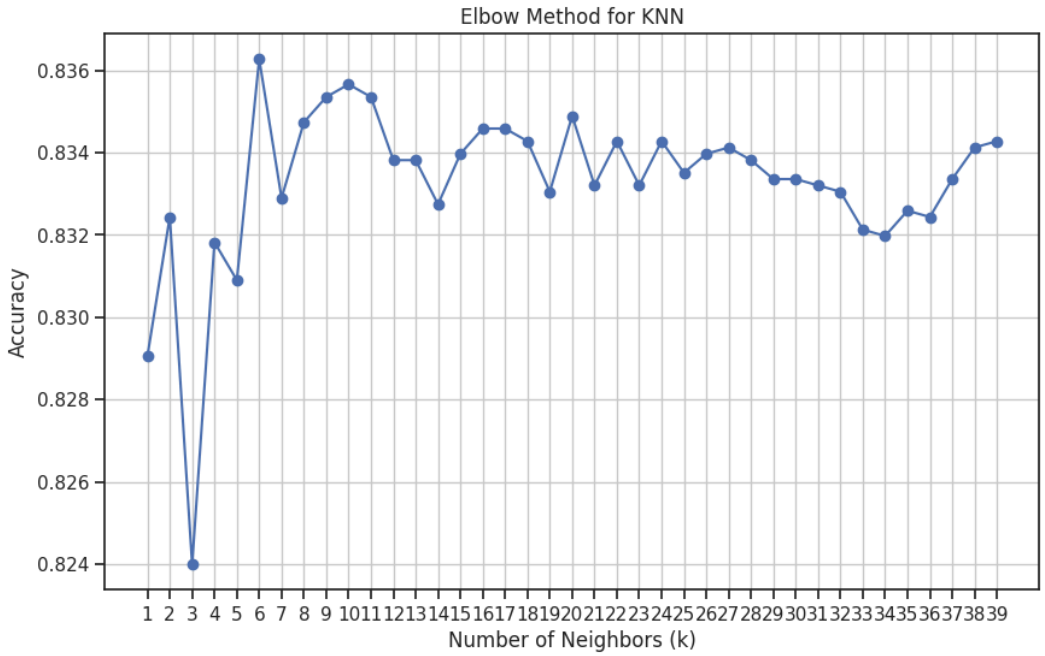
Balancing the dataset

```
Original dataset shape: Counter({0: 20401, 1: 5663})
Undersampled dataset shape Counter({0: 5663, 1: 5663})
```

```
Original dataset shape: Counter({0: 20401, 1: 5663})
Oversampled dataset shape: Counter({0: 20401, 1: 20401})
```



K-nearest-neighbor (KNN)



Selection of number of neighbor by observing at what point graph becomes relatively stable than before to the stability of graph.

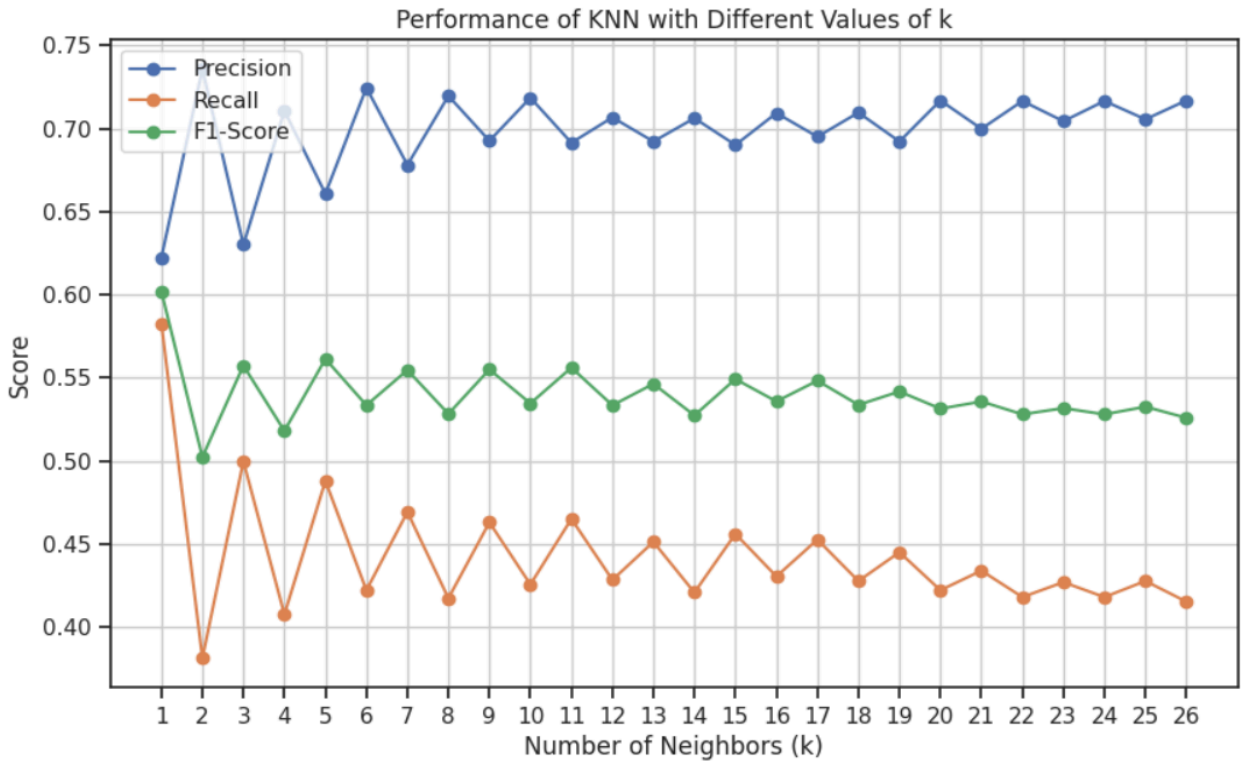
KNN using standardized data and dimensionality reduction (PCA)

```
Best k: 9
Accuracy: 0.8032837195028387
```

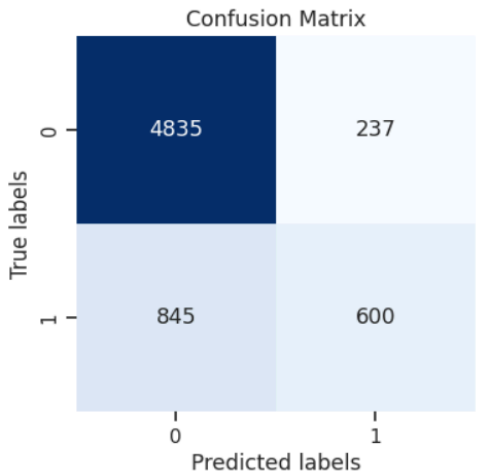
KNN without using standardized data and dimensionality reduction (PCA)

```
Best k: 26
Accuracy: 0.8339726868190885
```

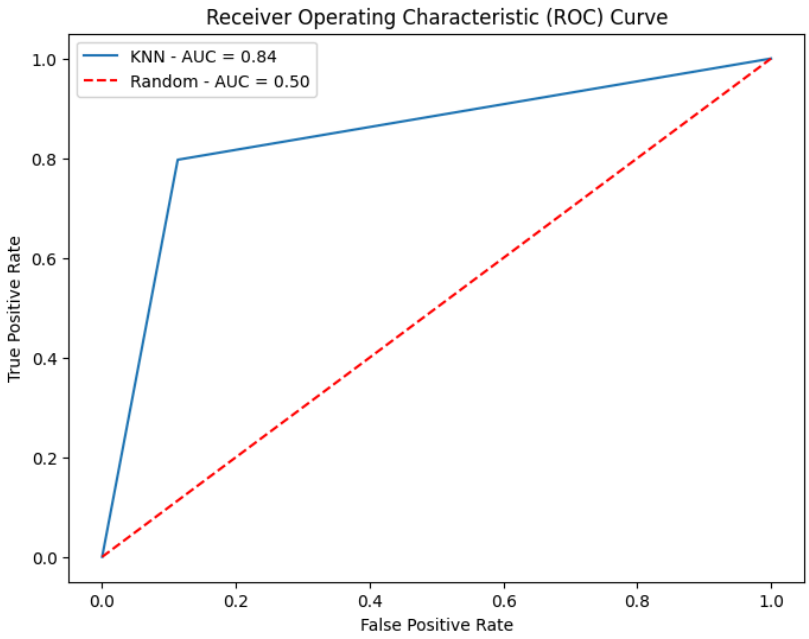
Precision, Recall, F1-score



Confusion Matrix displaying all the stats (i.e. TP, TN, FP, FN)



ROC Curve with 0.84 Area under curve

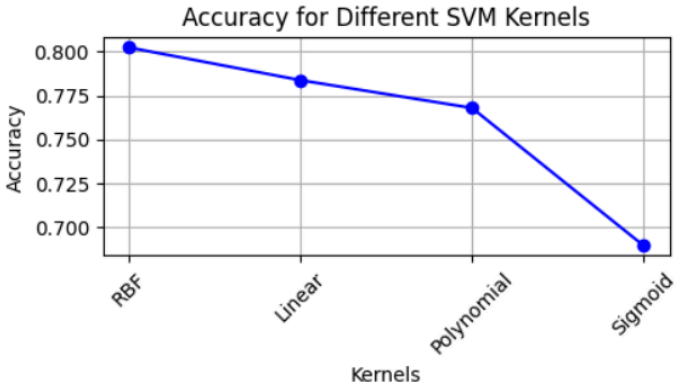


Support Vector Machine(SVM)

For Applying SVM algorithm, some columns have been dropped for better analysis: ['loan_status','person_age','person_home_ownership','loan_intent',]

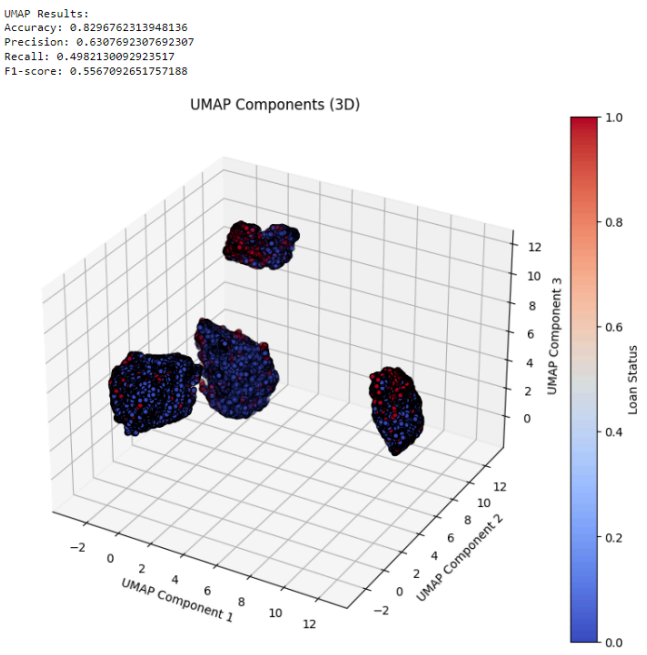
The prediction data set is trained on different SVM kernel functions for checking the best accuracy achieved.

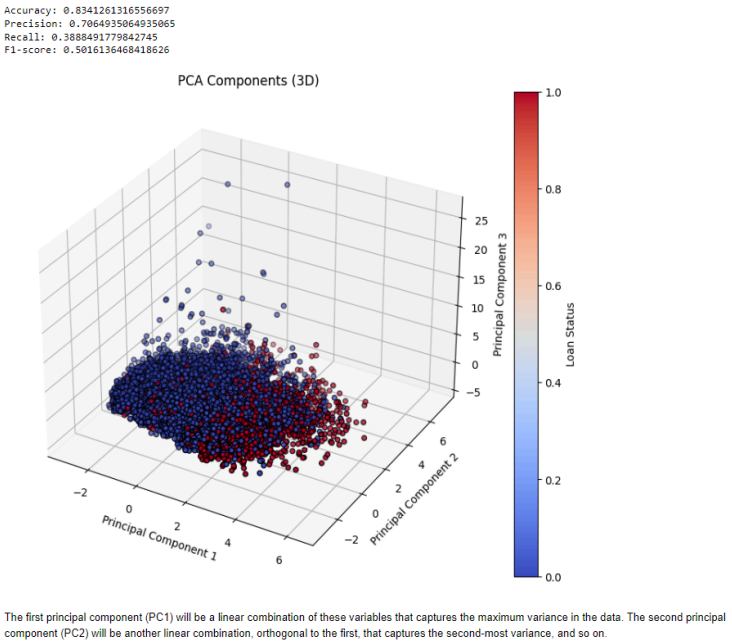
Kernel	Accuracy	Precision	Recall	F1-score
RBF	0.802391	0.830414	0.749282	0.787764
Linear	0.783755	0.789278	0.761494	0.775137
Polynomial	0.767932	0.843985	0.645115	0.73127
Sigmoid	0.689522	0.679352	0.692529	0.685877



On hyper tuning we got the value of c as 1.0 and gamma as 0.5
We got the updated accuracy as 85%

On analyzing by dimensionality reduction



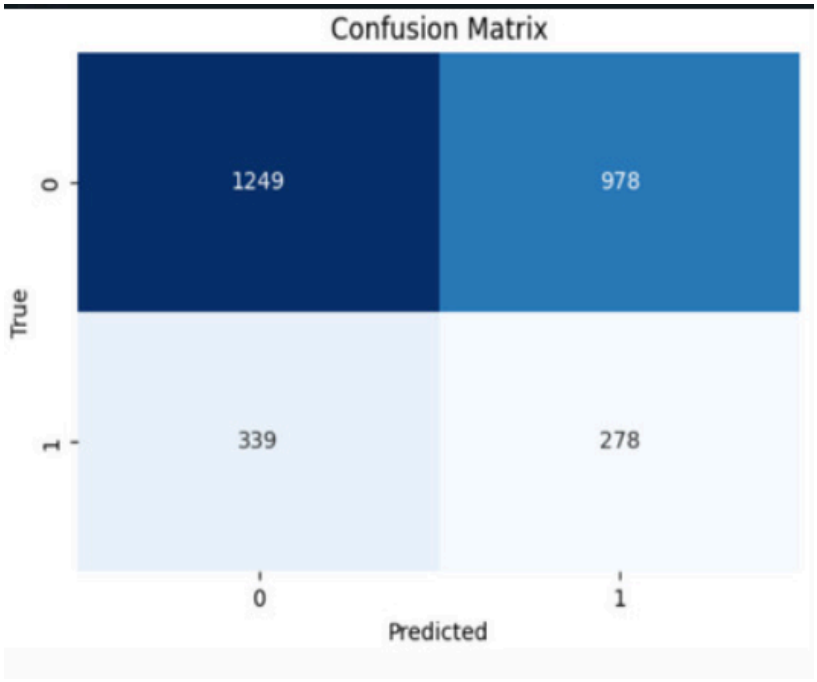


The first principal component (PC1) will be a linear combination of these variables that captures the maximum variance in the data. The second principal component (PC2) will be another linear combination, orthogonal to the first, that captures the second-most variance, and so on.

PCA dimenionality Reduction

Results obtained are close to optimal value but less than analyzing with all features

Confusion Matrix obtained :

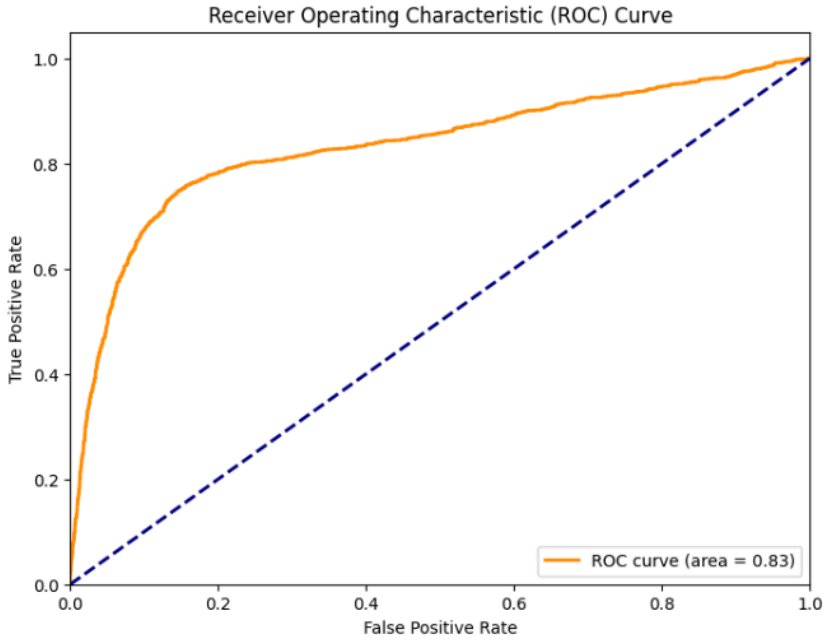


Confusion matrix on updated hyper parameters:

```
Accuracy: 0.8551480742673009
Precision: 0.7215189873417721
Recall: 0.5296640457469621
F1-score: 0.6108821104699094
```

c = 1 and gamma = 0.5

ROC Curve :

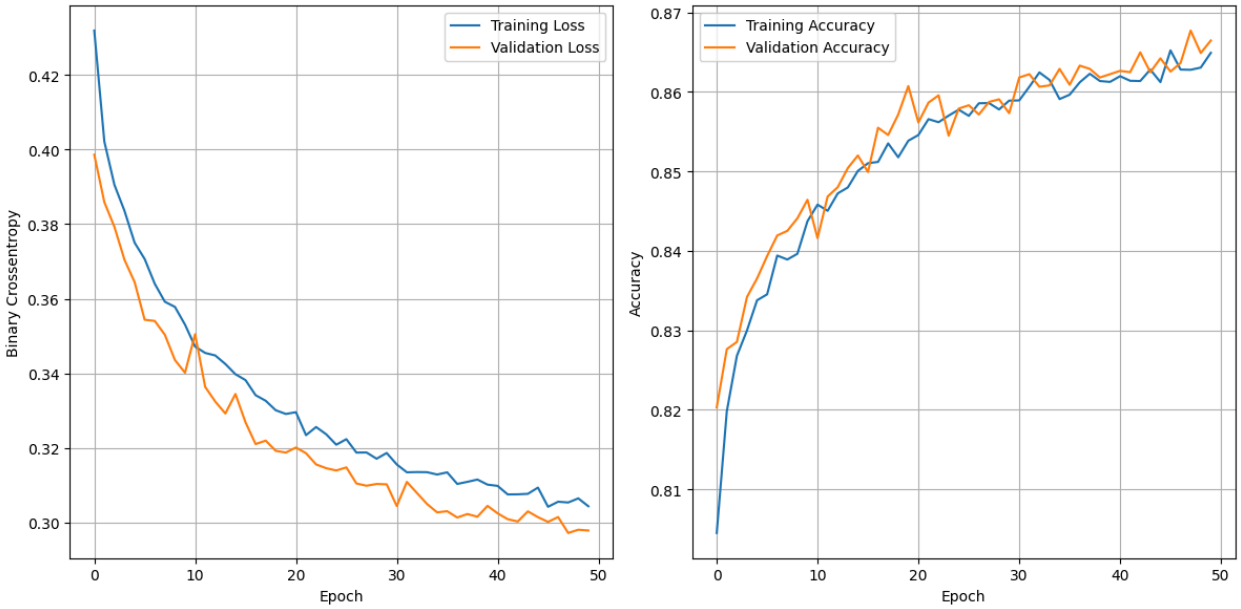


Area under ROC curve shown above determines optimal accuracy (85%) has been achieved by hyper tuning and validated by cross validation.

Artificial Neural Network and Random Forest:

For ANN and Random Forest we have dropped person_age and cb_person_cred_hist_length as they are not very much related with our output feature.

This graph on the left is showing that as the epoch is increasing we are getting less training and validation loss and the graph on the right showing the increasing accuracy on the training and validation set as the epochs are increasing.



Total Learnable parameter and the structure of the neural network

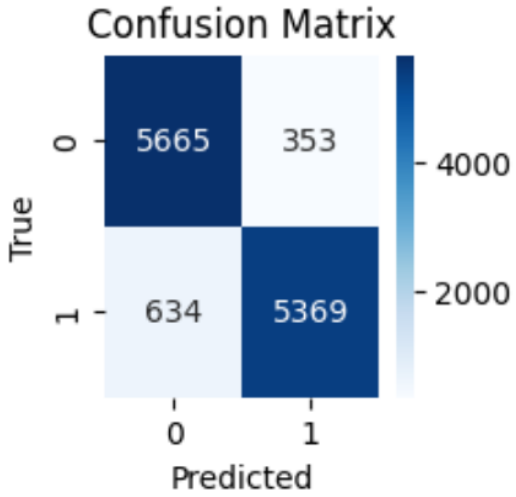
Layer (type)	Output Shape	Param #
dense_27 (Dense)	(None, 64)	640
dropout_18 (Dropout)	(None, 64)	0
dense_28 (Dense)	(None, 16)	1,040
dropout_19 (Dropout)	(None, 16)	0
dense_29 (Dense)	(None, 1)	17

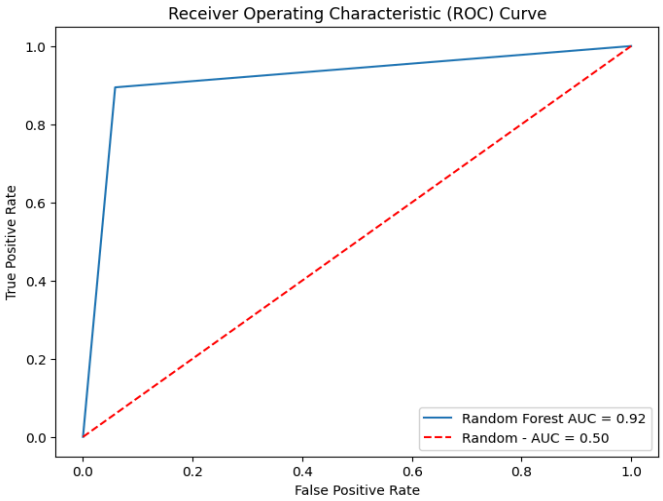
```
Total params: 5,093 (19.90 KB)

Trainable params: 1,697 (6.63 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 3,396 (13.27 KB)
```

Accuracy given by our Artificial Neural Network is 0.86

Confusion Matrix and ROC curve for Random Forest :

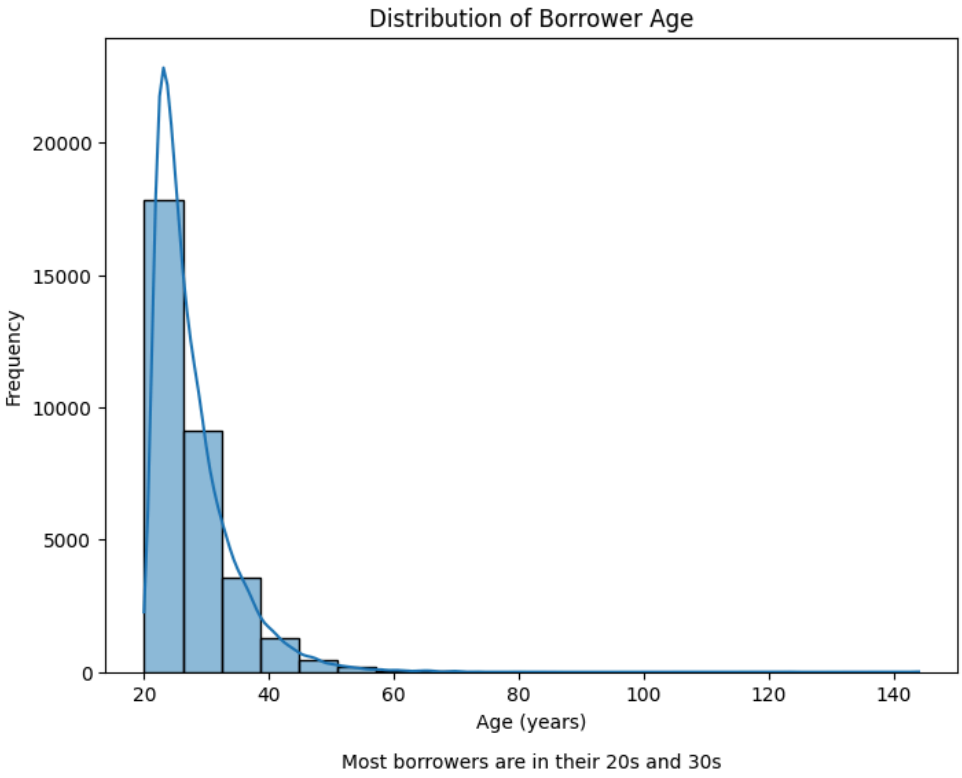
we have taken criteria as gini, max_depth = 50 and min_samples_split=10



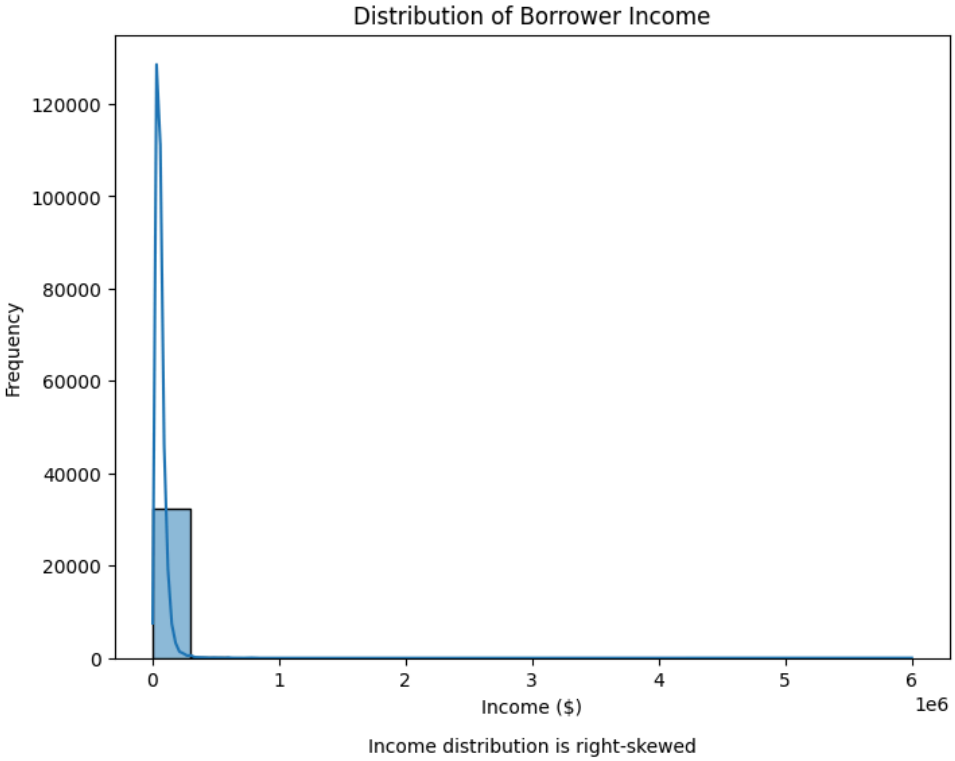


Evaluation Metrics for Random Forest:
Precision: 0.94
Recall: 0.89
F1-Score: 0.92
Accuracy: 0.917

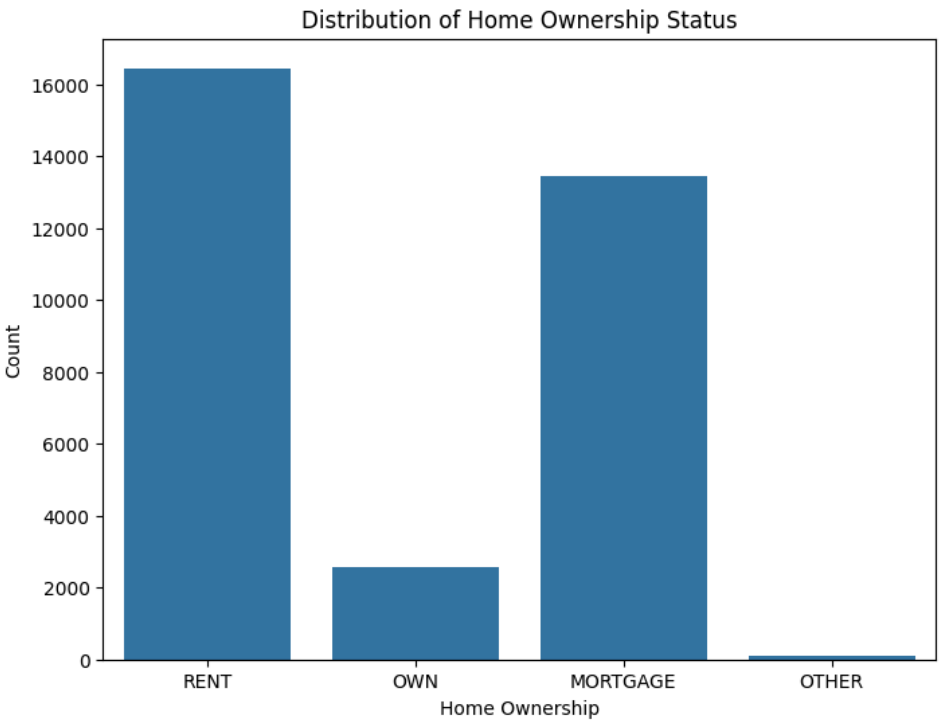
Decision Tree



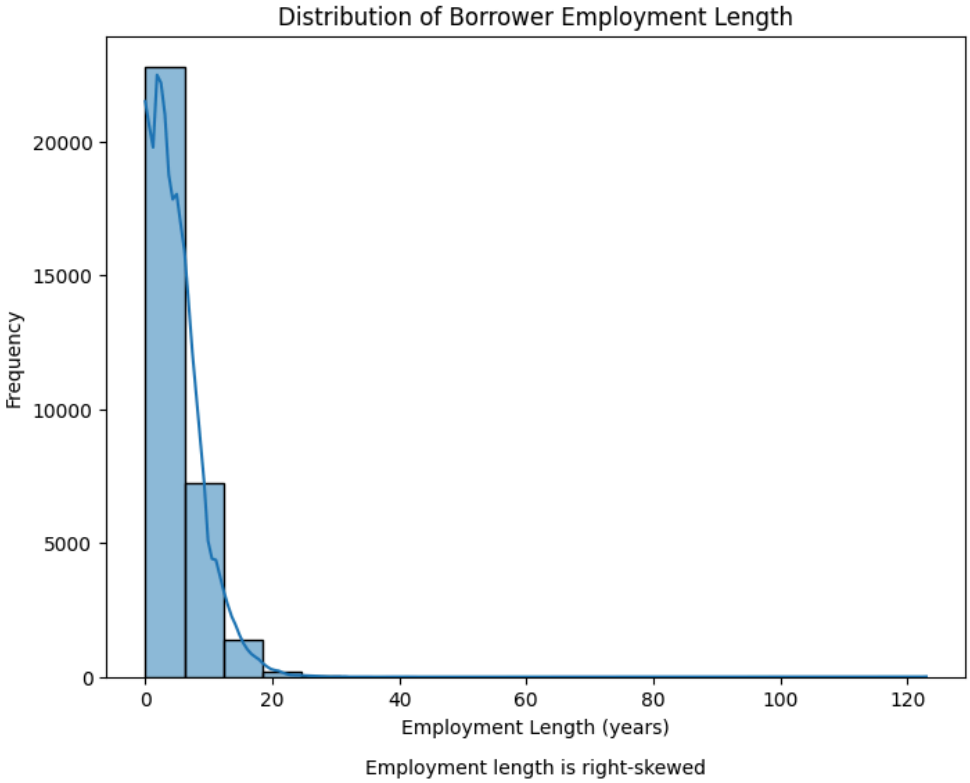
Most borrowers are in their 20s and 30s



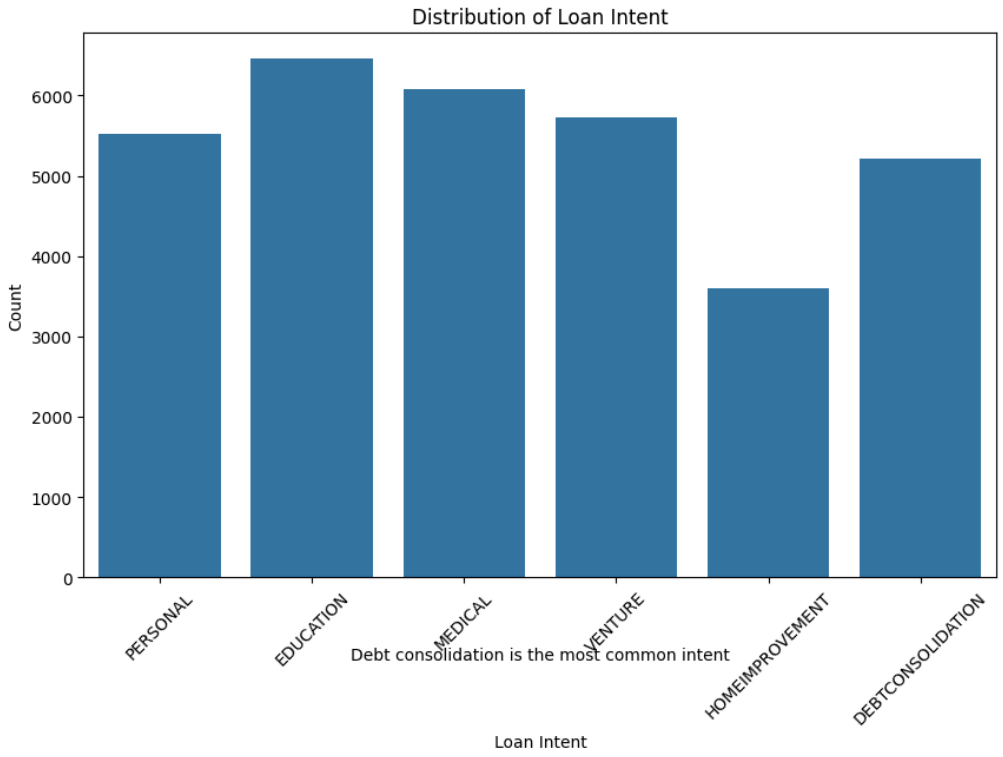
Income distribution is right-skewed

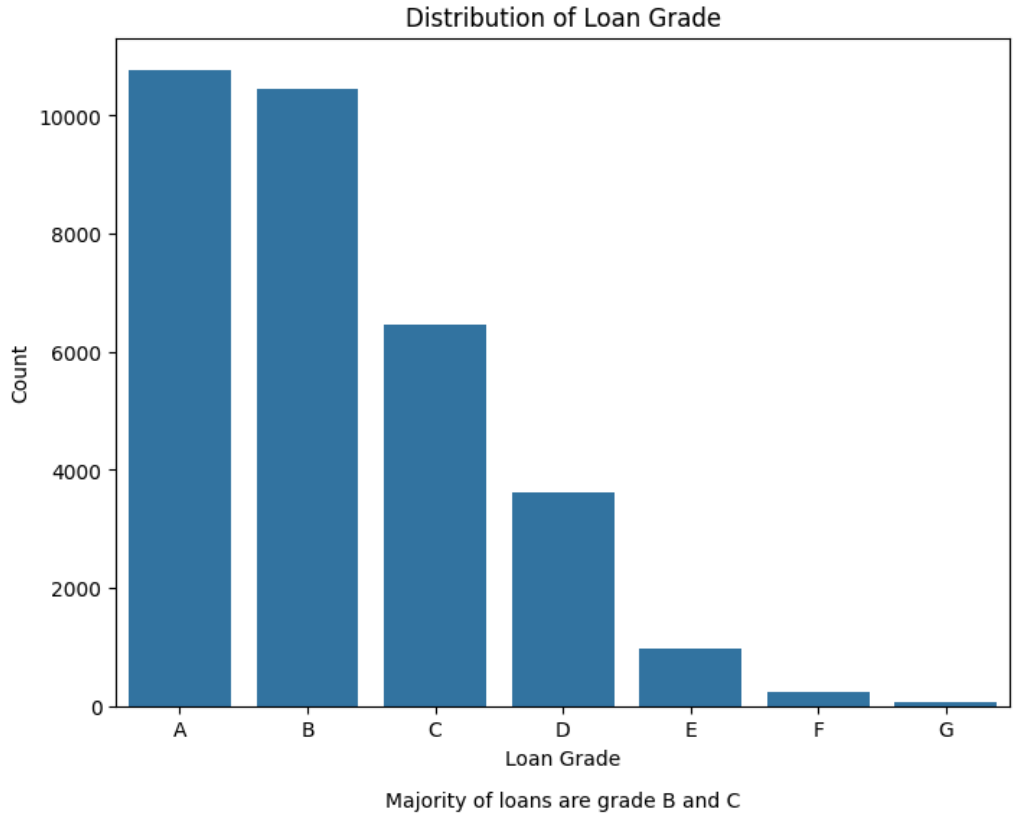


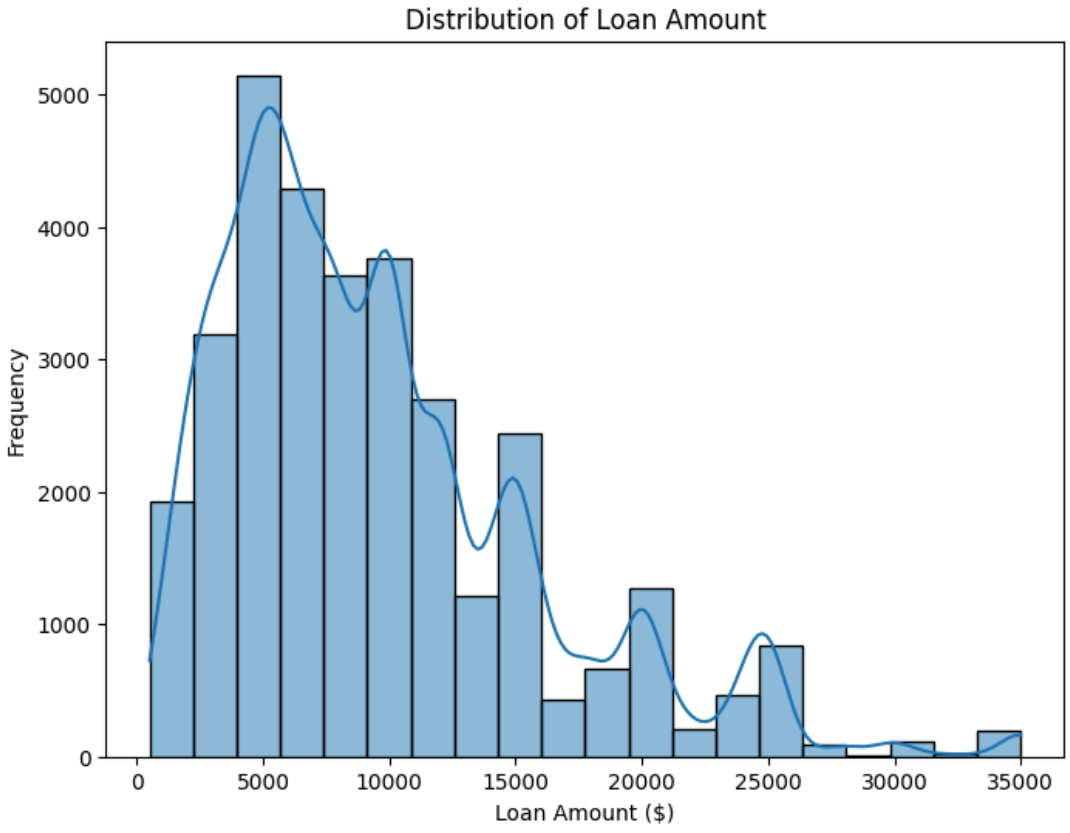
Majority of borrowers own a home



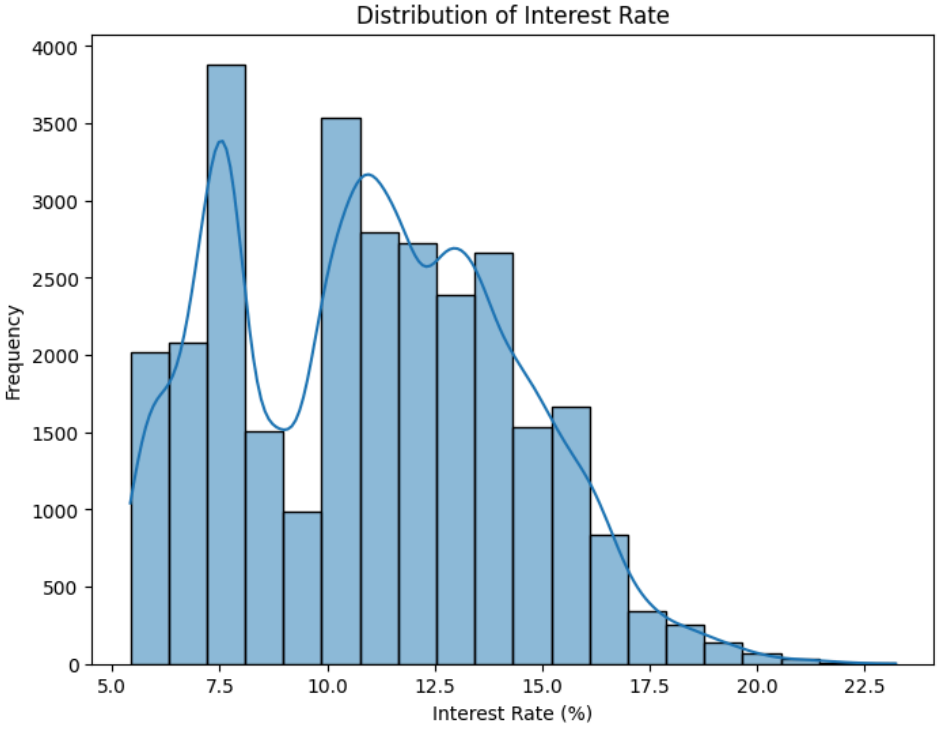
Employment length is right-skewed


Distribution of Loan Intent
Count
0
1000
2000
3000
4000
5000
6000
PERSONAL
EDUCATION
MEDICAL
VENTURE
HOMEIMPROVEMENT
DEBTCONSOLIDATION
Debt consolidation is the most common intent
Loan Intent

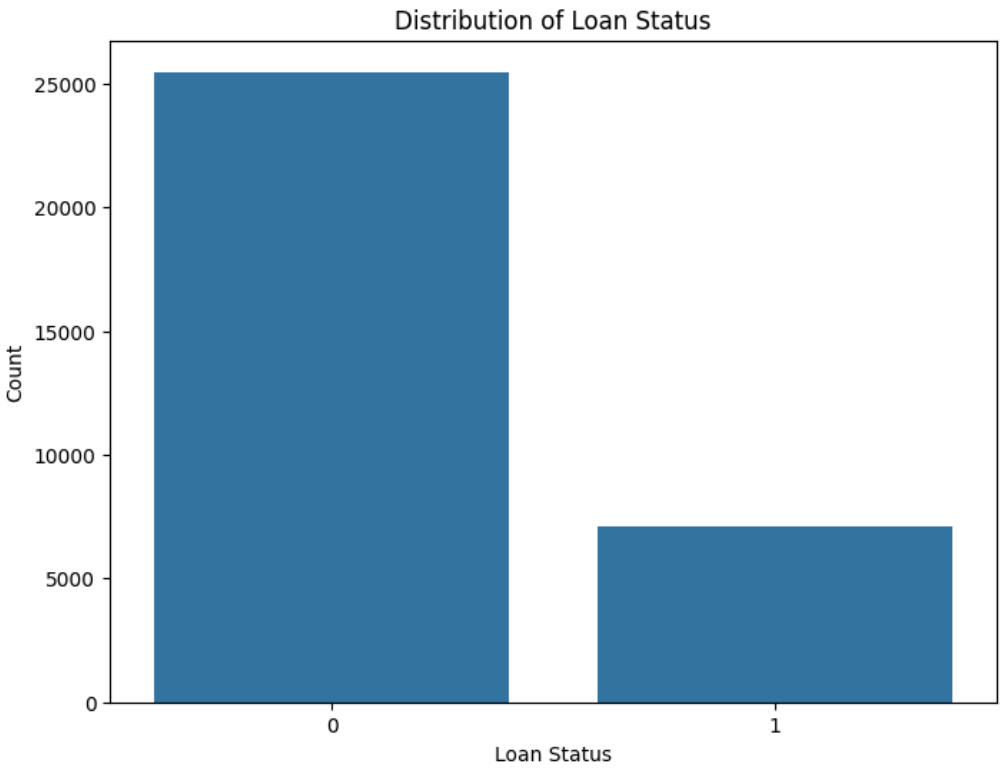

Distribution of Loan Grade
Count
0
2000
4000
6000
8000
10000
A
B
C
D
E
F
G
Loan Grade
Majority of loans are grade B and C



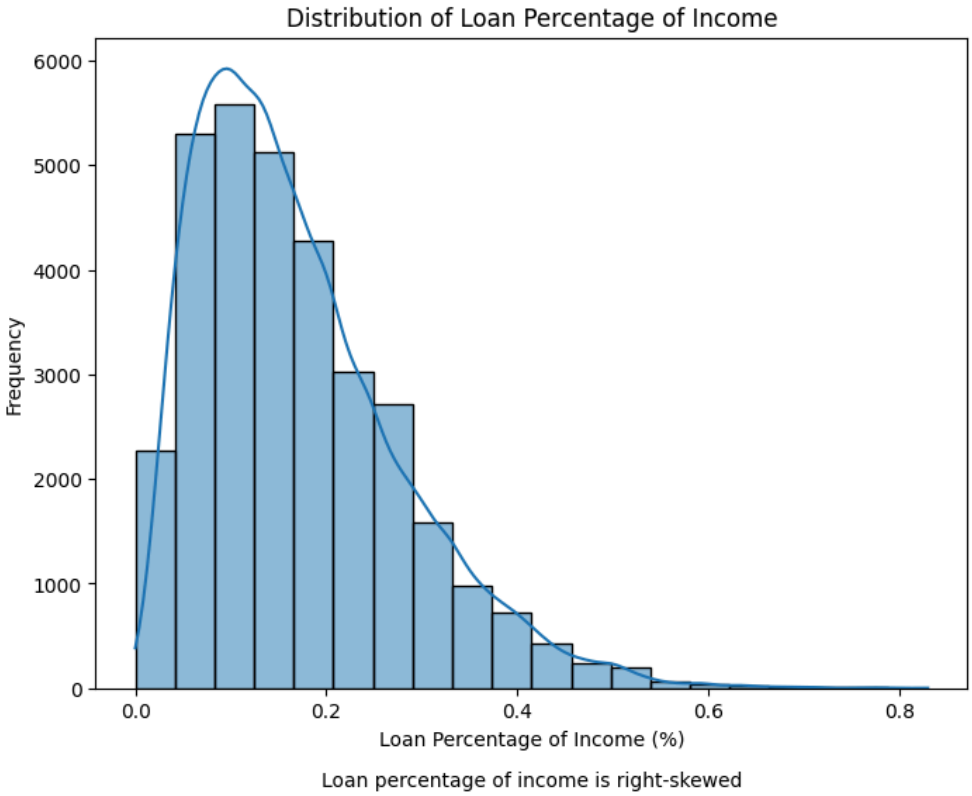
Loan amounts are right-skewed



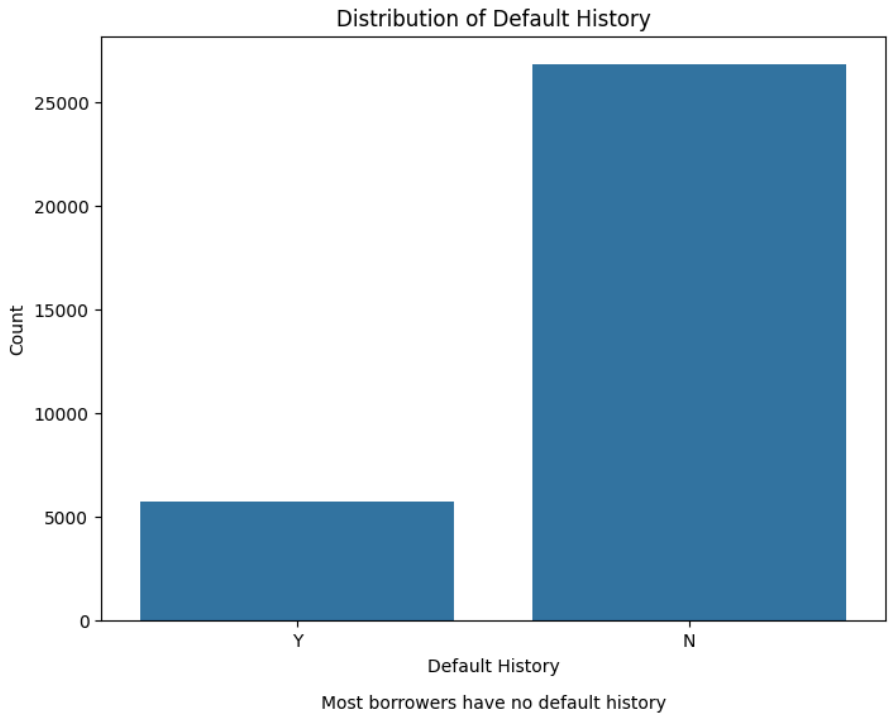
Interest rates are normally distributed



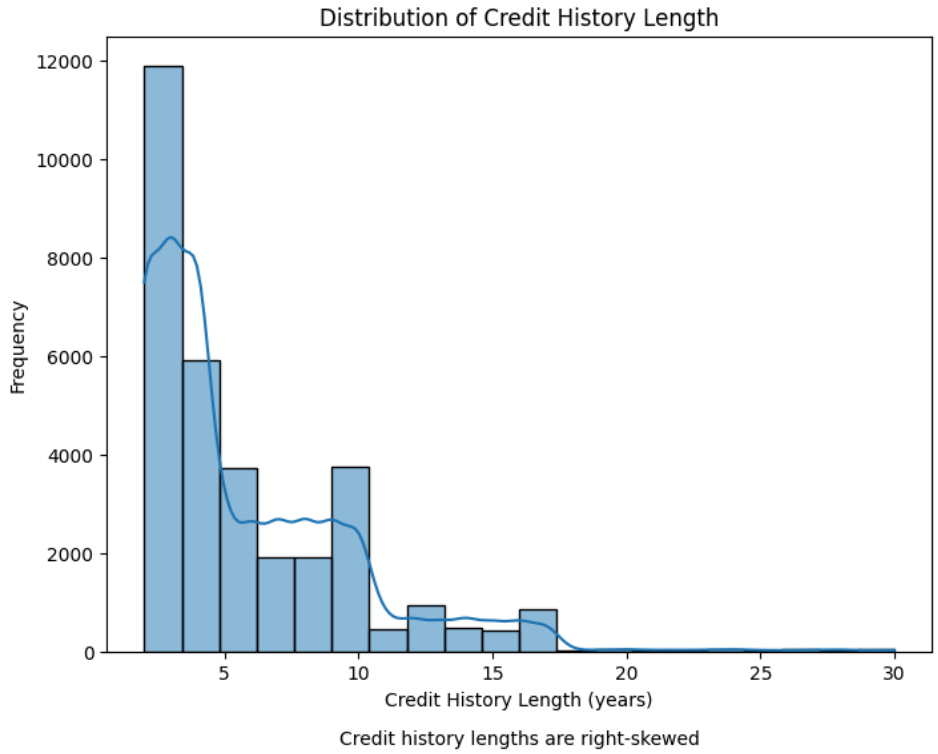
Most loans are fully paid



Loan percentage of income is right-skewed

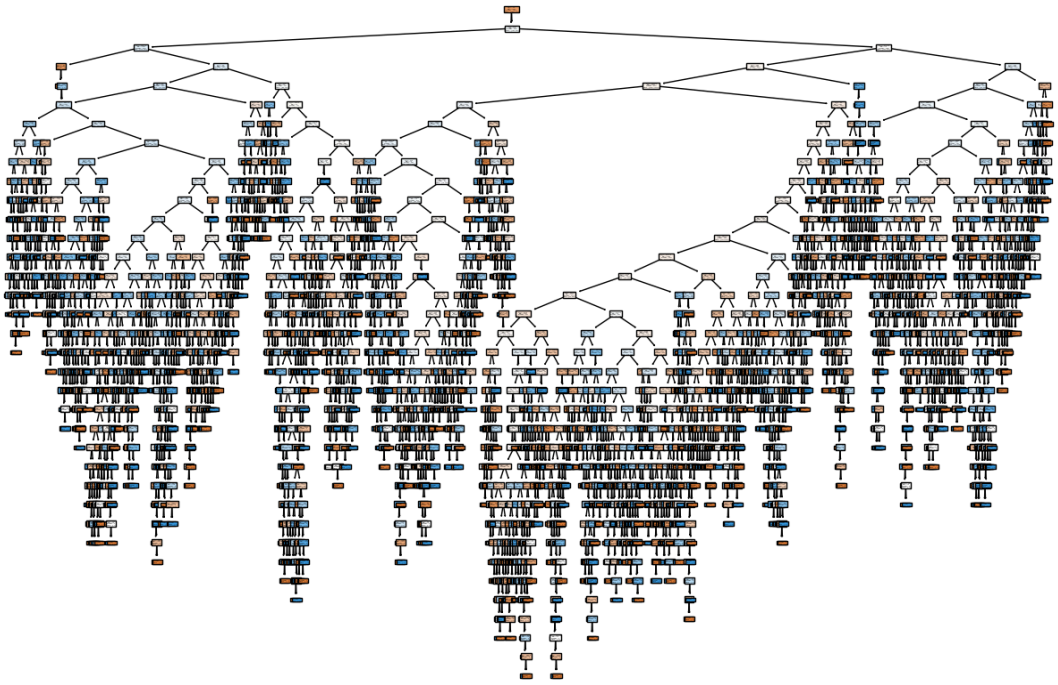


Most borrowers have no default history

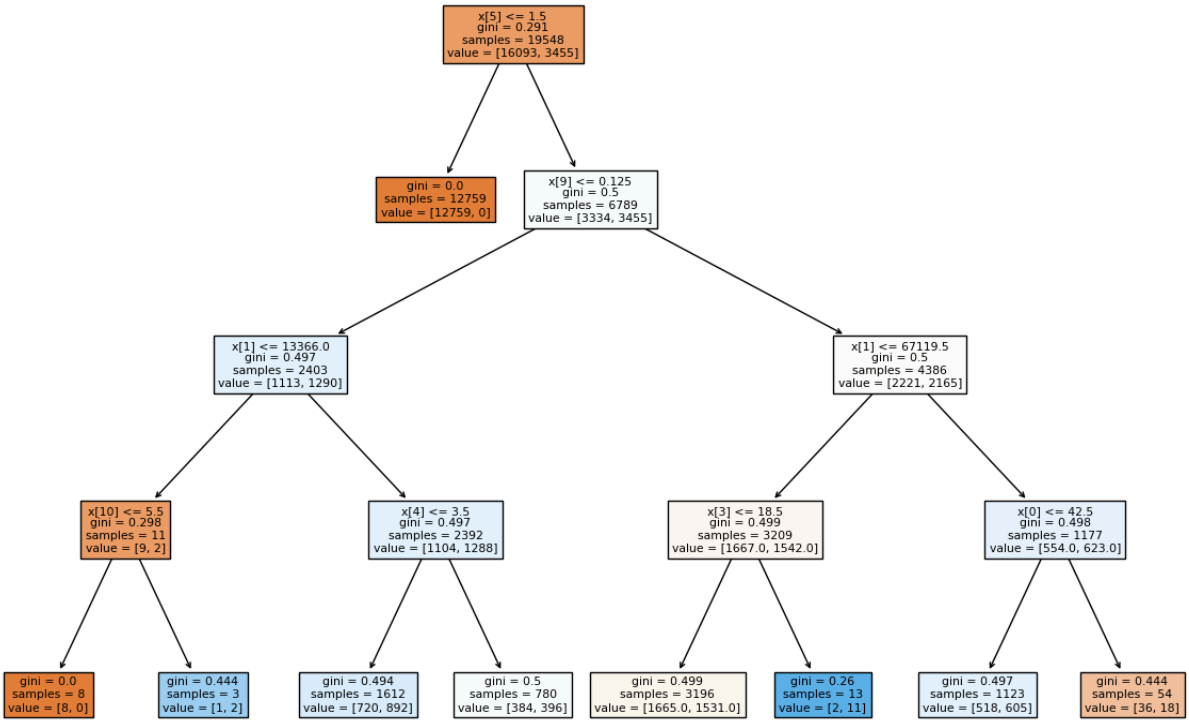


Credit history lengths are right-skewed

This is the tree without prunning.



This is the tree after Post prunning.



```
Accuracy:  0.8258401104802824

Confusion Matrix:
 [[4823  537]
 [ 598  559]]

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.90      0.89      5360
           1       0.51      0.48      0.50      1157

    accuracy                           0.83      6517
   macro avg       0.70      0.69      0.70      6517
weighted avg       0.82      0.83      0.82      6517
```

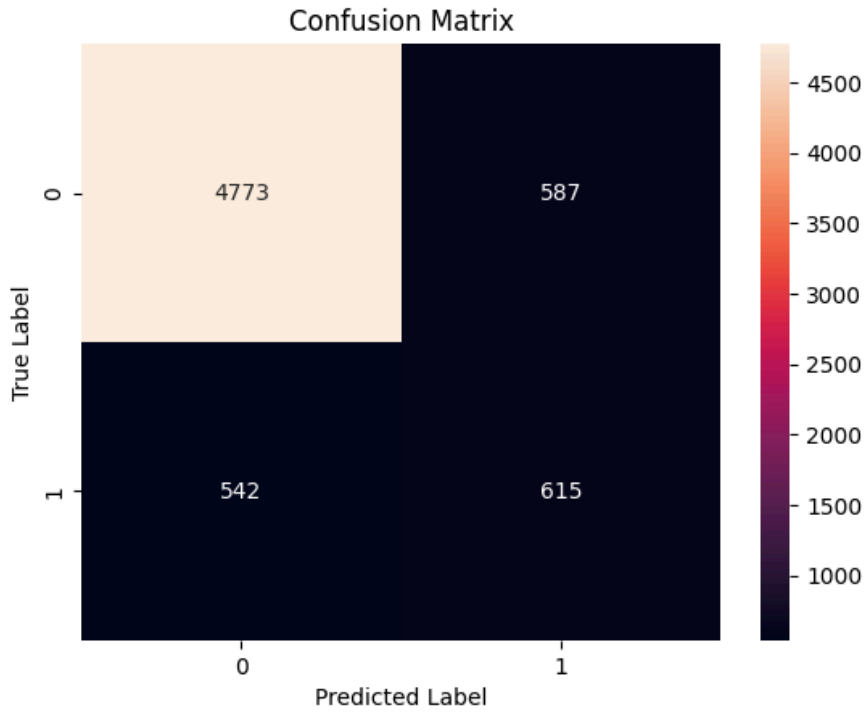
These are the best parameter after doing pre prunning.

```
cv.best_params_
```

```
{'criterion': 'entropy', 'max_features': 'log2', 'splitter': 'random'}
```

Accuracy after preprunning.

```
Accuracy:  0.8267607794997698
```

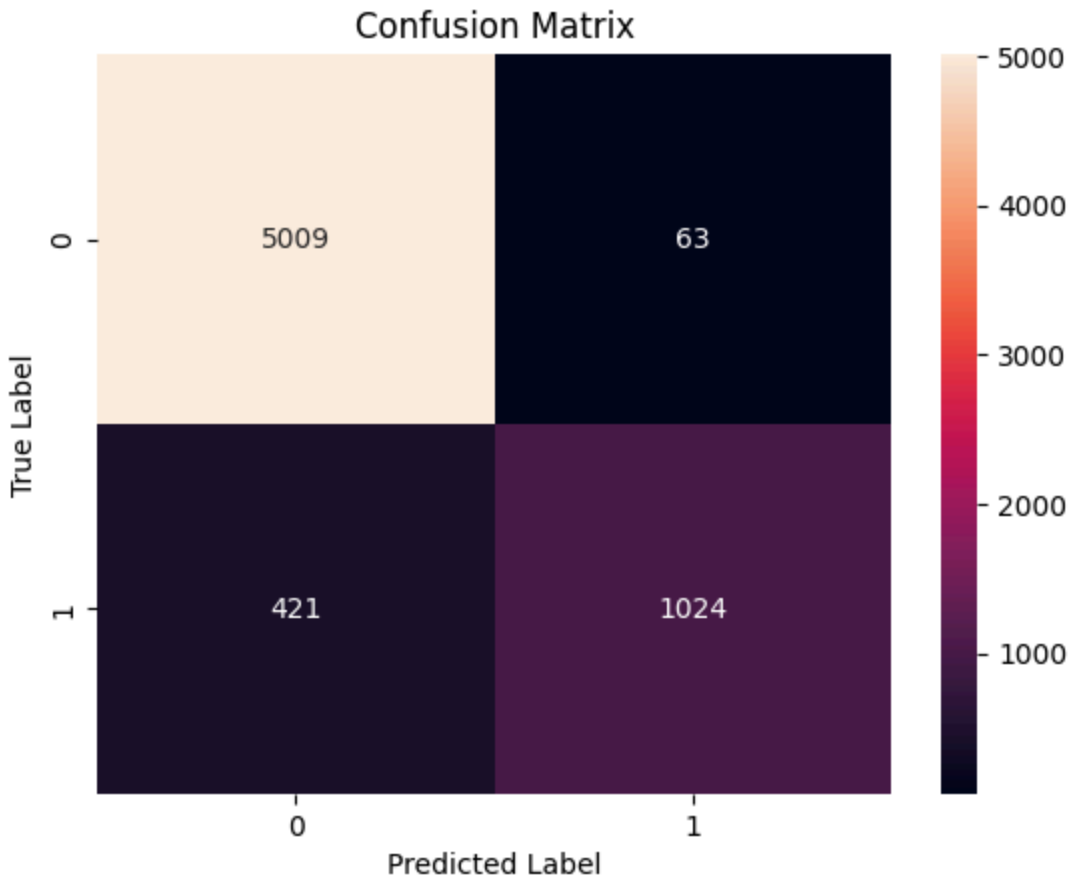


Classification Report:	precision	recall	f1-score	support
0	0.90	0.89	0.89	5360
1	0.51	0.53	0.52	1157
accuracy			0.83	6517
macro avg	0.70	0.71	0.71	6517
weighted avg	0.83	0.83	0.83	6517

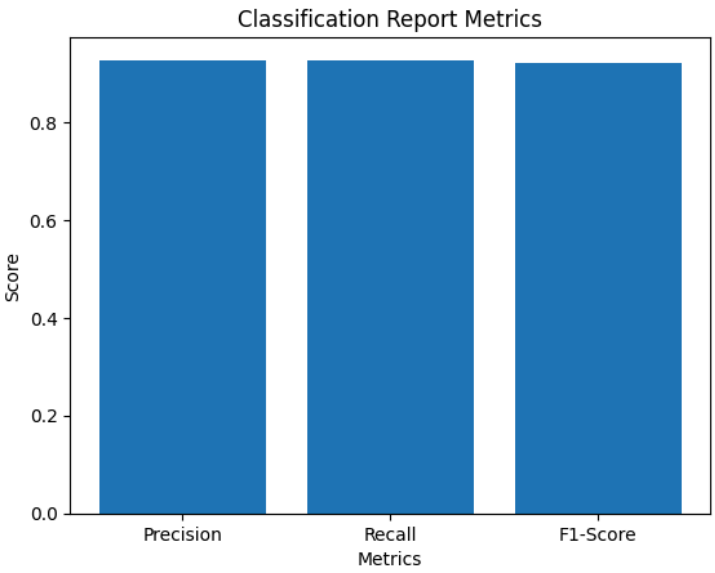
Gradient Boosting

I have used 5 cross validation in this.

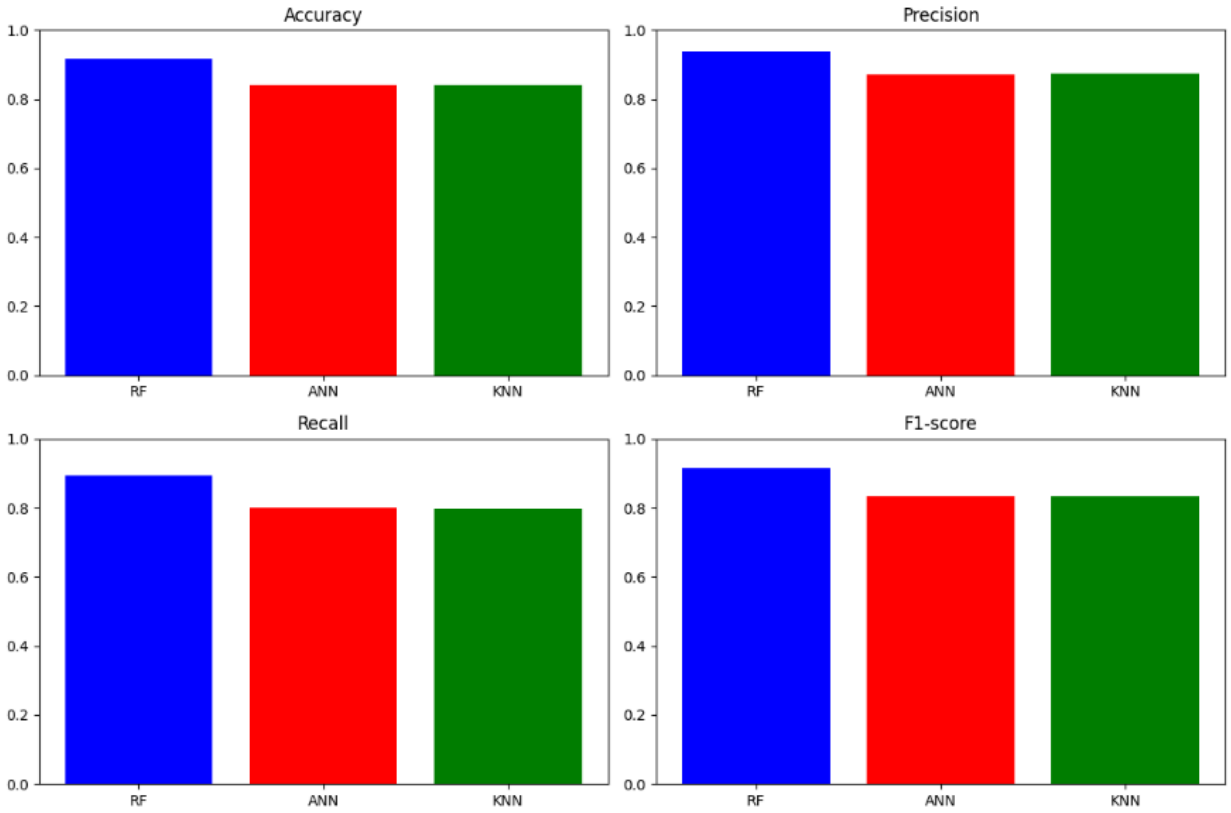
```
Accuracy: 0.9257326990946755
```

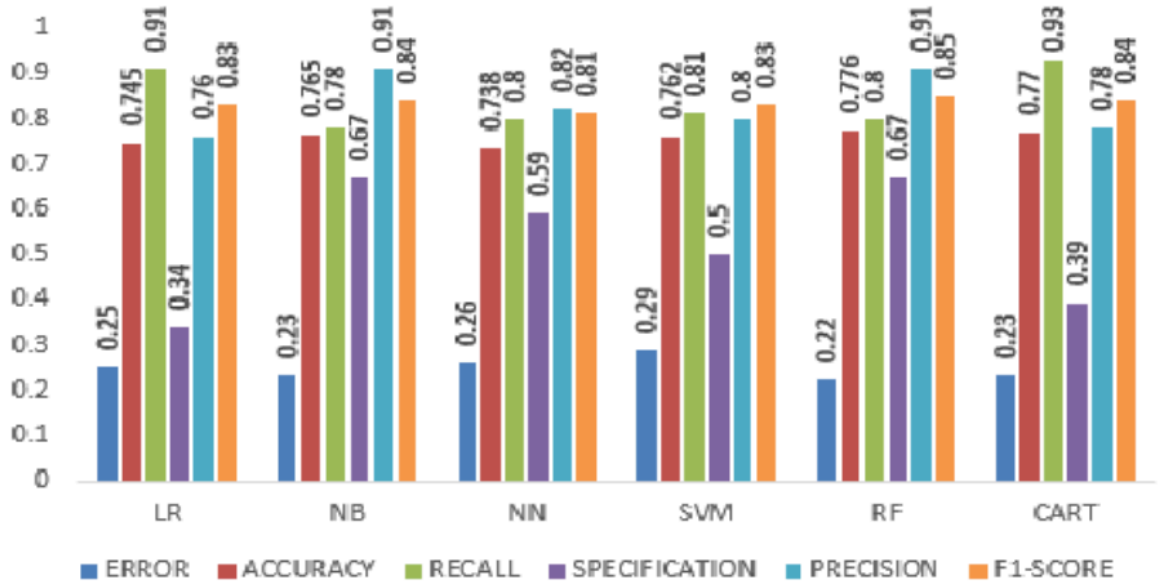



```
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.99      0.95      5072
           1       0.94      0.71      0.81      1445

    accuracy                           0.93      6517
   macro avg       0.93      0.85      0.88      6517
weighted avg       0.93      0.93      0.92      6517
```


Classification Report Metrics
Score
0.0
0.2
0.4
0.6
0.8
Precision
Recall
F1-Score
Metrics

Results and Discussion:





The graph above is obtained by our models whereas the graph below is taken from the Research paper and we can clearly see that after applying data preprocessing and analysis our models are performing well as compared to the model in the paper.

Conclusion and Future Scope:

In conclusion, our exploration of machine learning models for credit risk assessment has yielded promising results. Across ANN, KNN, SVM, DT, and RF, we observed significant improvements in accuracy, precision, recall, and F1-score, indicating enhanced predictive performance. These findings underscore the potential of machine learning in assisting lenders with more informed and accurate lending decisions.

Looking ahead, the future scope lies in further refining and optimizing our models, exploring additional feature engineering techniques, and evaluating the applicability of advanced deep learning methods. Additionally, extending our analysis to include real-time data and incorporating alternative data sources could enhance the robustness and agility of our credit risk assessment framework

References

1).Khashman, "Neural networks for credit risk evaluation: Investigation of different neural models and learning schemes," Expert Systems with Applications, vol. 37, no. 9, pp. 6233–6239, 2010.

2).X. Huang, X. Liu, and Y. Ren, "Enterprise credit risk evaluation based on neural network algorithm," Cognitive Systems Research, vol. 52, pp. 317–324, 2018.

3). A. Khashman, "Credit risk evaluation using neural networks: Emotional versus conventional models," Applied Soft Computing, vol. 11, no. 8, pp. 5477–5484, 2011.

4).A. Srivastava, M. Yadav, S. Basu, S. Salunkhe and M. Shabad, "Credit Card Fraud Detection at Merchant Side using Neural Networks", 2016 International Conference on Computing for Sustainable Global Development (INDIACom), pp. 667-670, 2016.

5)Henley, W. E. & Hand, D. J. (1996) "A k-Nearest-Neighbour classifier for assessing consumer credit risk", The Statistician, vol. 45(1): 77

6)Hellwig, M. (2000) "Financial intermediation with risk aversion", Review of Economic Studies, vol. 67(4): 719–742

7)Matoussi, H. (2010) "Credit risk evaluation of a Tunisian commercial: Logistic regression versus Neural Network Modeling", Accounting and Management Information Systems, vol. 9, no. 1

8). Al-Shayea, Q.K., El-Refae, G.A. and El-Itter, S.F. (2010): Neural Networks in Bank Insolvency Prediction.
International Journal of Computer Science and Network Security, Vol. 10, No. 5, pp. 240-245.

9.) Bahrammirzaee, A. (2010): A Comparative Survey of Artificial Intelligence Applications in Finance: Artificial Neural Networks, Expert System and Hybrid Intelligent Systems. Neural Computing & Applications, Vol. 19 No. 8, pp.1165-1195.

10). Bahrammirzaee, A., Ghatari, A., Ahmadi, P. and Madani, K. (2011): Hybrid Credit Ranking Intelligent System Using Expert Systems and Artificial Neural Networks. Applied Intelligence, Vol. 34 No.1, pp. 28-46.

11). Freeman, J.A. and Skapura, D.M. (1991): Neural Networks: Algorithms, Applications and Programming Techniques, Addison Wesley Longman.

12). Kashman, A. (2010): Neural Networks for Credit Risk Evaluation: Investigation of Different Neural Models and Learning Schemes. Expert Systems with Applications, Vol. 37 No.9, pp. 6233-6239.

Appendix

KNN (k-nearest-neighbor)

Elbow method

```
k_values = range(1, 40)
accuracy_scores = []
# Iterate over each value of k
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, Y_train)
    # Calculate accuracy scores on test set
    accuracy = knn.score(X_test, Y_test)
    accuracy_scores.append(accuracy)

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(k_values, accuracy_scores, marker='o', linestyle='-')
plt.title('Elbow Method for KNN')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.grid(True)
plt.show()
```

Applied:

- PCA for dimensionality reduction

```
# Apply PCA for dimensionality reduction
pca = PCA(n_components=10)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)
```

- Standardized data with the Scaler module

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(x_rus)
X_test_scaled = scaler.transform(X_test)
```

- Tunned Hyper parameters like number of neighbors, and etc

- Performed cross validation

Best results without PCA and Standardization:

```
precision_scores = []
recall_scores = []
f1_scores = []
# Evaluation metrics for different values of k
scores = {}
# Define the parameter grid
param_grid = {'n_neighbors': range(1, 27)}
KNN = KNeighborsClassifier()
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=KNN, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, Y_train)
# Get the best parameters
best_params = grid_search.best_params_
best_k = best_params['n_neighbors']
# Best estimator
best_estimator = grid_search.best_estimator_

for k in range(1, 27):
    # Classifying the query point using KNN by choosing appropriate number of k
    KNN = KNeighborsClassifier(n_neighbors=6, metric='minkowski', p=2)
    KNN.fit(X_train, Y_train)
    y_pred = KNN.predict(X_test)
    # Getting table of evaluation metrics
    precision = precision_score(Y_test, y_pred)
    recall = recall_score(Y_test, y_pred)
    f1 = f1_score(Y_test, y_pred)
    precision_scores.append(precision)
    recall_scores.append(recall)
    f1_scores.append(f1)
    scores[k] = {'Precision': precision, 'Recall': recall, 'F1-Score': f1}

# Best estimator to predict
y_pred_best = best_estimator.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred_best)
print("Best k:", best_k)
print("Accuracy:", accuracy)
✓ 31.9s
```

Confusion Matrix

```
# Calculate confusion matrix for the best estimator
conf_matrix = confusion_matrix(Y_test, y_pred_best)

# Plot confusion matrix
plt.figure(figsize=(4, 4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()
✓ 0.0s
```

ROC curve (AUC = 0.84)

```
# Plotting ROC curve
plt.figure(figsize=(8, 6))
fpr, tpr, thresholds = roc_curve(Y_test, y_pred)
plt.plot(fpr, tpr, label=f'KNN - AUC = {auc(fpr, tpr):.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='red', label='Random - AUC = 0.50')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

Support Vector Machine(SVM)

```
import warnings
import math
warnings.filterwarnings('ignore')

# import tensorflow as tf
import pandas as pd
import numpy as np
from sklearn.utils import shuffle

# Plots and stats
import matplotlib.pyplot as plt
import seaborn as sns

# Label Encoding & Scaling
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.neighbors import KNeighborsClassifier

# Model Building
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1_score, precision_score, recall_score
```

UnderSampling

```
LABELS = ["Normal", "Fraud"]
count_classes = pd.value_counts(df_encoded['loan_status'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.title("Transaction Class Distribution")
plt.xticks(range(2), LABELS)
plt.xlabel("Loan Status")
plt.ylabel("Frequency")
```

```
# undersampling
from imblearn.under_sampling import RandomUnderSampler
undersampler = RandomUnderSampler(random_state=42)
X_undersampled, y_undersampled = undersampler.fit_resample(X, y)
```

Splitting the data into test and train sets

```
X_train, X_test, y_train, y_test = train_test_split(X_undersampled, y_undersampled, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
count_classes = pd.value_counts(y_undersampled, sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.title("Transaction Class Distribution")
plt.xticks(range(2), LABELS)
plt.xlabel("Loan Status")
plt.ylabel("Frequency")
```

Analyzing data on SVM kernel Function:

```
# 1. RBF
svm1 = SVC(kernel='rbf', random_state=42)
svm1.fit(X_train_scaled, y_train)
y_pred_1 = svm1.predict(X_test_scaled)

accuracy_1 = accuracy_score(y_test, y_pred_1)
precision_1 = precision_score(y_test, y_pred_1)
recall_1 = recall_score(y_test, y_pred_1)
f1_1 = f1_score(y_test, y_pred_1)

print("Accuracy:", accuracy_1)
print("Precision:", precision_1)
print("Recall:", recall_1)
print("F1-score:", f1_1)
```

```
#2. Linear
svm = SVC(kernel='linear', random_state=42)
svm.fit(X_train_scaled, y_train)
y_pred = svm.predict(X_test_scaled)

accuracy_2 = accuracy_score(y_test, y_pred)
precision_2 = precision_score(y_test, y_pred)
recall_2 = recall_score(y_test, y_pred)
f1_2 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy_2)
print("Precision:", precision_2)
print("Recall:", recall_2)
print("F1-score:", f1_2)
```

```
# 3. Polynomial
svm = SVC(kernel='poly', random_state=42)
svm.fit(X_train_scaled, y_train)
y_pred = svm.predict(X_test_scaled)

accuracy_3 = accuracy_score(y_test, y_pred)
precision_3 = precision_score(y_test, y_pred)
recall_3 = recall_score(y_test, y_pred)
f1_3 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy_3)
print("Precision:", precision_3)
print("Recall:", recall_3)
print("F1-score:", f1_3)
```

```
#4. sigmoid
svm = SVC(kernel='sigmoid', random_state=42)
svm.fit(X_train_scaled, y_train)
y_pred = svm.predict(X_test_scaled)

accuracy_4 = accuracy_score(y_test, y_pred)
precision_4 = precision_score(y_test, y_pred)
recall_4 = recall_score(y_test, y_pred)
f1_4 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy_4)
print("Precision:", precision_4)
print("Recall:", recall_4)
print("F1-score:", f1_4)
```

Hyper tuning for parameters (c and gamma)

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import shuffle

# Shuffle and split the data
X_shuffled, y_shuffled = shuffle(X, y, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X_shuffled, y_shuffled, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
C_value = 1
gamma_value = 0.5
svm_classifier = SVC(kernel='rbf', C=C_value, gamma=gamma_value, random_state=42)
svm_classifier.fit(X_train_scaled, y_train)

y_pred = svm_classifier.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

Dimensionality reduction

Using PCA

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import matplotlib.cm as cm
pca = PCA(n_components=3)
X_train_pca_3d = pca.fit_transform(X_train_scaled)
svm_classifier = SVC(kernel='rbf', C=1, gamma=0.5, random_state=42)
svm_classifier.fit(X_train_pca_3d, y_train)
X_test_pca_3d = pca.transform(X_test_scaled)
y_pred_8 = svm_classifier.predict(X_test_pca_3d)
accuracy_8 = accuracy_score(y_test, y_pred_8)
precision_8 = precision_score(y_test, y_pred_8)
recall_8 = recall_score(y_test, y_pred_8)
f1_8 = f1_score(y_test, y_pred_8)
print("Accuracy:", accuracy_8)
print("Precision:", precision_8)
print("Recall:", recall_8)
print("F1-score:", f1_8)
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_train_pca_3d[:, 0], X_train_pca_3d[:, 1], X_train_pca_3d[:, 2], c=y_train, cmap=plt.cm.coolwarm, edgecolor
ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
ax.set_title('PCA Components (3D)')
cbar = plt.colorbar(cm.ScalarMappable(cmap=plt.cm.coolwarm), ax=ax)
cbar.set_label('Loan Status')
plt.show()
```

Confusion matrix and ROC Curve

```
from sklearn.metrics import confusion_matrix
import numpy as np

min_length = min(len(y_test), len(y_pred_1))
y_test_truncated = y_test[:min_length]
y_pred_1_truncated = y_pred_1[:min_length]
conf_matrix = confusion_matrix(y_test_truncated, y_pred_1_truncated)
# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import shuffle
X_shuffled, y_shuffled = shuffle(X, y, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X_shuffled, y_shuffled, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
C_value = 1
gamma_value = 0.5
svm_classifier = SVC(kernel='rbf', C=C_value, gamma=gamma_value, probability=True, random_state=42)
svm_classifier.fit(X_train_scaled, y_train)
y_probs = svm_classifier.predict_proba(X_test_scaled)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_probs)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

Decision tree

Code for feature graphs,

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load your dataset (assuming it's stored in a DataFrame called 'df')
```

```
# Example: df = pd.read_csv('your_dataset.csv')

# Select only numeric columns for correlation calculation
numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns
numeric_df = df[numeric_columns]

# Plot histograms for numerical variables
plt.figure(figsize=(12, 8))
for i, col in enumerate(numeric_columns, 1):
  plt.subplot(3, 3, i)
  sns.histplot(df[col], bins=20, kde=True)
  plt.title(col)
plt.tight_layout()
plt.show()
```

```
from sklearn.tree import DecisionTreeClassifier
treemodel = DecisionTreeClassifier() #gini is default criterion which is in between 0 and 0.5
treemodel.fit(X_train, Y_train)
second_treemodel = DecisionTreeClassifier(max_depth=4)
second_treemodel.fit(X_train, Y_train)
```

```
from sklearn import tree
plt.figure(figsize=(15,10))
tree.plot_tree(treemodel, filled=True)

plt.figure(figsize=(15,10))
tree.plot_tree(second_treemodel, filled=True)
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
print("Accuracy: ", accuracy_score(y_pred,Y_test ))
print("\nConfusion Matrix:\n", confusion_matrix(Y_test, y_pred))
print("\nClassification Report:\n", classification_report(Y_test, y_pred))
```

```
y_pred =treemodel.predict(X_test)
print(y_pred)
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
print("Accuracy: ", accuracy_score(y_pred,Y_test ))
print("\nConfusion Matrix:\n", confusion_matrix(Y_test, y_pred))
print("\nClassification Report:\n", classification_report(Y_test, y_pred))
```

Preprunning

```
#preprunning
parameter = {
  "criterion": ["gini", "entropy", "log_loss"],
  "splitter": ["best", "random"],
  "max_features": ["auto", "sqrt", "log2"],
}

from sklearn.model_selection import GridSearchCV
treemodel  = DecisionTreeClassifier()
cv = GridSearchCV(treemodel, param_grid=parameter, cv=5,scoring='accuracy')
cv.fit(X_train, Y_train)
cv.best_params_
y_pred=cv.predict(X_test)
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
print("Accuracy: ", accuracy_score(y_pred,Y_test ))


conf_matrix = confusion_matrix(Y_test, y_pred)

# Visualize Confusion Matrix
sns.heatmap(conf_matrix, annot=True, fmt='d')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

```
print("\nClassification Report:\n", classification_report(Y_test, y_pred))
```

Gradient Boosting

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Separate features (X) and target variable (y)
X = df.drop('loan_status', axis=1)  # Assuming 'loan_status' is the target variable
y = df['loan_status']

# Define categorical and numerical columns
categorical_columns = ['person_home_ownership', 'loan_intent', 'loan_grade',
'cb_person_default_on_file']
numeric_columns = [col for col in X.columns if col not in categorical_columns]

# Create preprocessing pipeline
preprocessor = ColumnTransformer(
  transformers=[
    ('num', SimpleImputer(strategy='median'), numeric_columns),  # impute missing values with median
for numeric columns
    ('cat', OneHotEncoder(), categorical_columns)  # one-hot encode categorical columns
  ])

# Initialize Gradient Boosting classifier
gb_model = GradientBoostingClassifier()
```

```
# Create pipeline with preprocessing and Gradient Boosting classifier
pipeline = Pipeline([
  ('preprocessor', preprocessor),
  ('classifier', gb_model)
])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit the model
pipeline.fit(X_train, y_train)

# Make predictions
y_pred = pipeline.predict(X_test)


import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
import numpy as np

# ... your existing code ...

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

conf_matrix = confusion_matrix(y_test, y_pred)

# Visualize Confusion Matrix
sns.heatmap(conf_matrix, annot=True, fmt='d')
```

```
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()


class_report = classification_report(y_test, y_pred)
print("\nClassification Report:\n", class_report)
import matplotlib.pyplot as plt
# Extract precision, recall, and F1-score from the classification report
report_data = classification_report(y_test, y_pred, output_dict=True)
precision = report_data['weighted avg']['precision']
recall = report_data['weighted avg']['recall']
f1_score = report_data['weighted avg']['f1-score']


# Create a bar plot
labels = ['Precision', 'Recall', 'F1-Score']
values = [precision, recall, f1_score]


plt.bar(labels, values)
plt.xlabel('Metrics')
plt.ylabel('Score')
plt.title('Classification Report Metrics')
plt.show()
```

ANN And Random Forest

Encoding and EDA

```
loan_status_values = df["loan_status"].unique()
plt.figure(figsize=(8, 6))
sns.countplot(x="loan_status", data=df, palette="Set1", order=loan_status_values)
plt.xlabel("Loan Status")
plt.ylabel("Count")
plt.title("Countplot of Loan Status")
plt.show()

df.fillna(df["loan_int_rate"].mean(),inplace=True)
df.fillna(df["person_emp_length"].mean(),inplace=True)
```

```
from sklearn.calibration import LabelEncoder

df_encoded = df.copy()

label_encoders = {}
columns_to_encode = ['person_home_ownership', 'loan_intent', 'loan_grade', 'cb_person_default_on_file']
for column in columns_to_encode:
    label_encoders[column] = LabelEncoder()
    df_encoded[column] = label_encoders[column].fit_transform(df_encoded[column])

df_encoded.head()
```

Finding the related features

```
corr = df_encoded.corr()['loan_status'].sort_values(ascending=False)
fig = px.bar(x = corr.keys(), y = corr.values, title = 'correlation with target value',
             labels = {'x':'Features', 'y': 'Correlation'})
fig.show()
```

X and Y for our model

```
X = df_encoded.drop(['loan_status'], axis=1)
y = df_encoded['loan_status']
X.shape

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=42)
X_train.shape
```

Oversampling

```
from imblearn.combine import SMOTETomek
smk = SMOTETomek(random_state=42)
X_res,y_res=smk.fit_resample(X_train,y_train)
X_res.shape,y_res.shape

from collections import Counter
print('Original dataset shape {}'.format(Counter(y)))
print('Resampled dataset shape {}'.format(Counter(y_res)))
```

New Oversampled dataset

```
count_classes = pd.value_counts(y_res, sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.title("Transaction Class Distribution")
plt.xticks(range(2), ["Default","NonDefault"])
plt.xlabel("Loan Status")
plt.ylabel("Frequency")
```

Standardizing the dataset

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
X_train=sc.fit_transform(X_train)
X_test=sc.transform(X_test)
```

ANN Model

```python
import numpy as np
import tensorflow as tf
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

def create_model(units1=64, units2=32, dropout_rate=0.3, learning_rate=0.001):
    model = Sequential()
    model.add(Dense(units=units1, activation='relu', input_shape=(9,)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(units=units2, activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(units=1, activation='sigmoid'))

    optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
    return model

param_dist = {
    'units1': [32, 64, 128],
    'units2': [16, 32, 64],
    'dropout_rate': [0.2, 0.3, 0.4],
    'learning_rate': [0.001, 0.01, 0.1]
}
```

```python
best_accuracy = 0
best_hyperparameters = None

for _ in range(10):
    hyperparameters = {param: np.random.choice(values) for param, values in param_dist.items()}
    model = create_model(**hyperparameters)
    model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2)  # Added batch_size and validation_split
    y_pred = (model.predict(X_test) > 0.5).astype('int32')
    accuracy = accuracy_score(y_test, y_pred)

    print(f'Hyperparameters: {hyperparameters}, Accuracy: {accuracy:.4f}')

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_hyperparameters = hyperparameters

print("Best Hyperparameters:", best_hyperparameters)
print("Best Accuracy:", best_accuracy)
```

```python
final_model = create_model(**best_hyperparameters)
history = final_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test), verbose=1)
```

Plotting the Error vs Epoch and Accuracy vs Epoch

```python
def plot_learning_curve(history):
    # Create subplots with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

    # Plot training and validation loss
    ax1.plot(history.history['loss'], label='Training Loss')
    ax1.plot(history.history['val_loss'], label='Validation Loss')
    ax1.set_xlabel('Epoch')
    ax1.set_ylabel('Binary Crossentropy')
    ax1.legend()
    ax1.grid(True)

    # Plot training and validation accuracy
    ax2.plot(history.history['accuracy'], label='Training Accuracy')
    ax2.plot(history.history['val_accuracy'], label='Validation Accuracy')
    ax2.set_xlabel('Epoch')
    ax2.set_ylabel('Accuracy')
    ax2.legend()
    ax2.grid(True)

    # Adjust layout to prevent clipping of labels
    plt.tight_layout()

    # Show the plot
    plt.show()

plot_learning_curve(history)
```

Random Forest Model without PCA

```python
from sklearn.ensemble import RandomForestClassifier
model=RandomForestClassifier(criterion="gini",max_depth=50,min_samples_split=10,random_state=42)
model.fit(X_train,y_train)
model.score(X_test,y_test)
y_predrf=model.predict(X_test)
y_predrf
```

Confusion Matrix

```
from sklearn.metrics import confusion_matrix

confusion_mat = confusion_matrix(y_test, y_predrf)

plt.figure(figsize=(2, 2))
sns.heatmap(confusion_mat, annot=True, fmt="d", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title(f'Confusion Matrix')
```

Roc Curve for Random Forest

```
from sklearn.metrics import roc_curve, auc

fpr, tpr, thresholds = roc_curve(y_test, y_predrf)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'Random Forest AUC = {auc(fpr, tpr):.2f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='red', label='Random - AUC = 0.50')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```

Random Forest Model with PCA

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
scaler = StandardScaler()
scaler.fit(X)
scaled_data = scaler.transform(X)
pca = PCA(n_components=2)
pca.fit(scaled_data)
x_pca = pca.transform(scaled_data)
```

```
from sklearn.ensemble import RandomForestClassifier
model=RandomForestClassifier()
model.fit(X_train,y_train)
model.score(X_test,y_test)
```