MACHINE LEARNING
CONSUMER LOAN PROCESSING

By:
Ramkishore Rao
DSA 5900 / Credit Hrs: 4 hrs
Summer 2022
Faculty Advisors: Dr. Trafalis/Dr. Radhakrishnan
Faculty Coordinator: Dr. Beattie

July 15, 2022

TABLE OF CONTENTS

List of Tables

List of Exhibits

List of Appendices

1.0 Introduction

This project serves as my final practicum for my master's degree in Data Science and Analytics being completed at the University of Oklahoma. As part of this project, various machine learning algorithms were applied to a bank loan dataset (bandora dataset) to aid in the processing of loan applications from consumers at a bank. For this study, a git hub repository developed by Dr. Jeff Heaton for his Deep Learning (DL) (Heaton, 2022) class at Washington University at St. Louis and his accompanying book (Heaton, 2022) were leveraged. In addition, class notes from Dr. Nicholson and from Dr. Diochnos were also utilized during the study.

The primary programming language used was Python, with its pre-existing modules. Tableau has been used during the initial exploration phase of the data.

2.0 Objectives

The main objective of the project is to use the existing bank loan dataset to develop back-end statistics models in order to provide a decision on the loan applications. Training, validation, and testing were performed using the existing dataset.

3.0 Exploratory Data Analysis

A bank loan dataset (bandora dataset) that contained 112 features was utilized in this study. Of the 112 features, one of the features was default_date, i.e., this feature had the date on which default occurred. This feature was the target class, and if default had occurred, it was assigned a value of 1 and if default had not occurred, it was assigned a value of 0.

Percentage of data points that belonged to target classes 0 and 1 by total were 66% and 34%, respectively (see Table 1).

Table 1: Data Breakdown by Target Class

Overall Class Counts
Defaulted: 1
Not Defaulted: 0

Target Class	Count of Target Class	% of Total Count of Target Class)
0	156,588	66.0%
1	80,635	34.0%
Grand Total	237,223	100.0%

Count of Target Class and % of Total Count of Target Class) broken down by Target Class.

3.1 Analysis Summary

A few tables and exhibits are provided in the following pages. They present a breakout of aggregated values of several features by target class value (i.e., 0 if debtor has not defaulted and 1 if debtor has defaulted).

3.2 Analysis Findings

Box and whisker plots for features brokwn down by target class shown on Exhibit 1 indicate the following:

1. Higher spread in data and higher maximum observed for Target Class 1 for the following features:
   - Probability of Default
   - Debt Types
   - Interest Servicing
   - Principal Balance
   - Principal Overdue by Schedule
2. No Significant Differences Between Classes observed for the following features:
   - Applied Loan Amount
   - Income types

Lower debtor default rates are attributed to the following based on estimates of aggregated data values breakouts by target class:

1) Higher Income (Exhibit 2)
2) Lower Interest Servicing (Exhibit 3)
3) Higher Previous Credit (Exhibit 4)
4) Better Credit Rating (Exhibit 5)
5) Lower median probability of default and expected loss (Exhibits 6 and 7)
6) Lower Principal Overdue by Schedule (Exhibit 8)
7) Lower counts for late payment status on loans (Exhibit 9)
8) Higher Education (Exhibit 12)
9) Higher actual number of previous procured loans (Exhibit 13)
10) More Prompt Payment (Exhibit 14)

Exhibit 1: Box and Whisker Plots, Select Continuous Variables

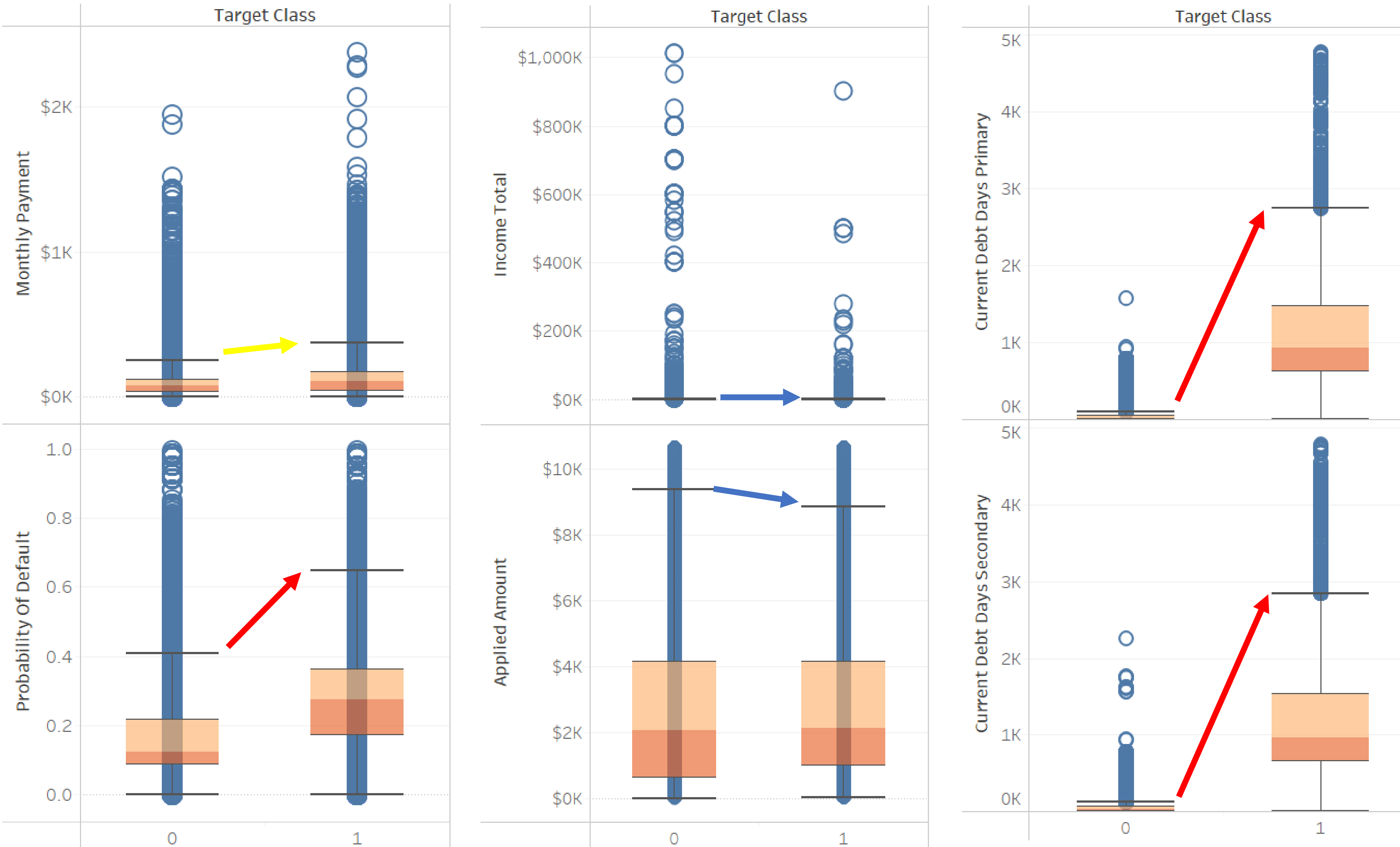


Exhibit 1 Continued: Box and Whisker Plots, Select Continuous Variables

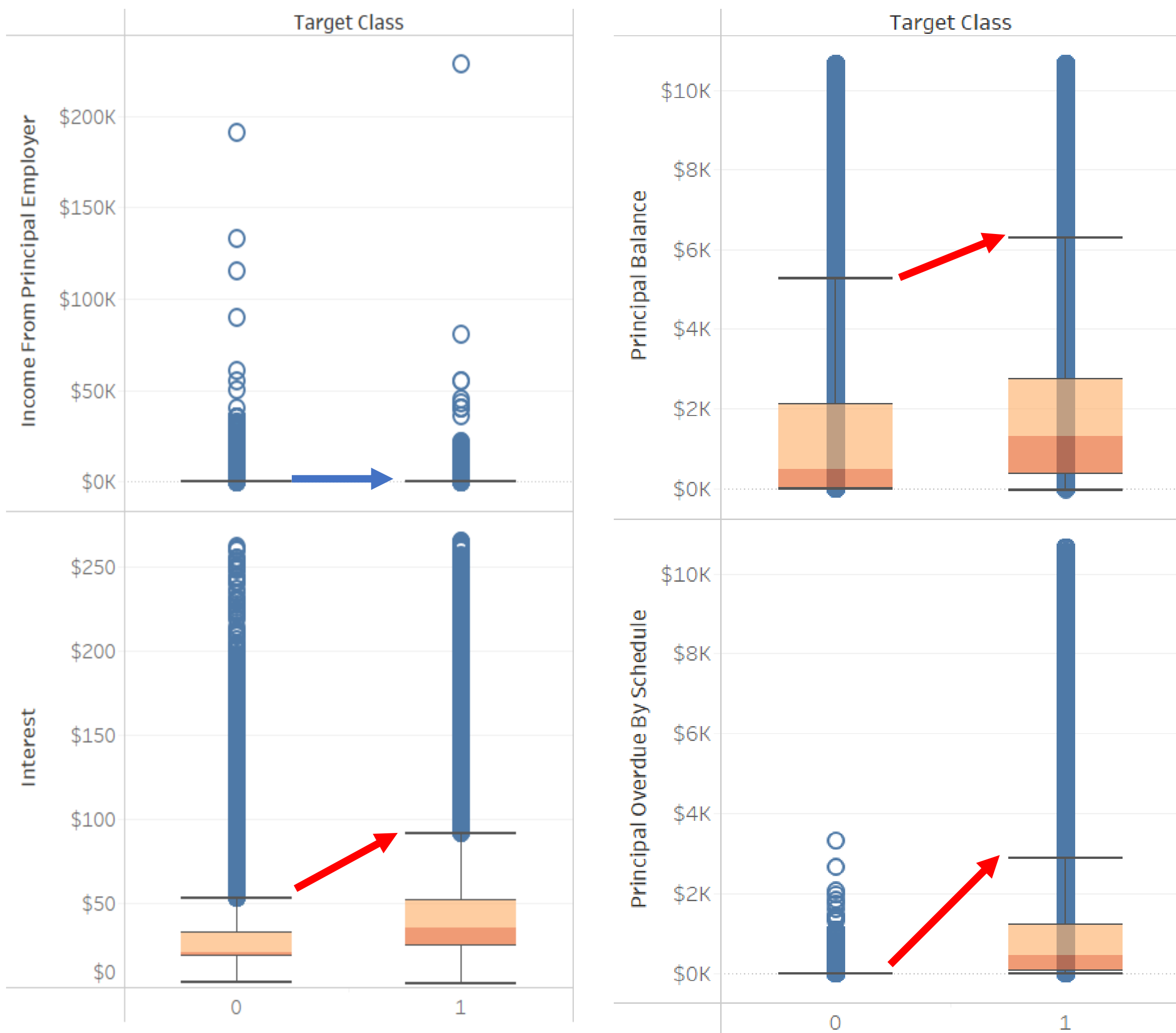


Exhibit 2: Income Breakouts by Target Class

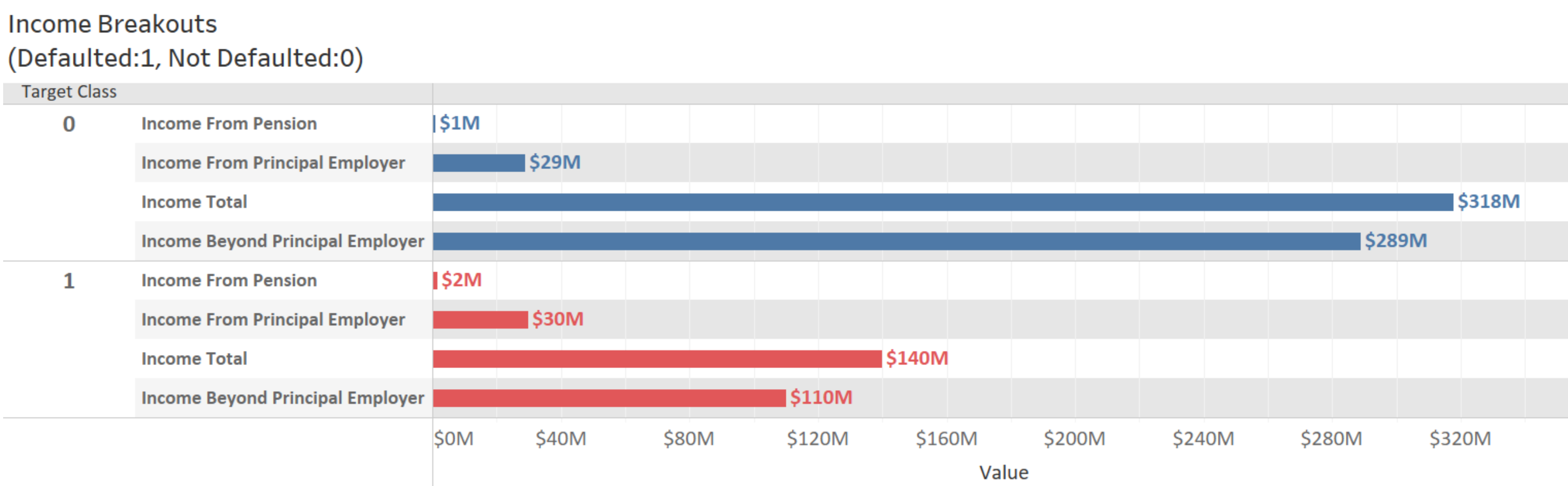


Exhibit 3: Interest Servicing Breakouts by Target Class

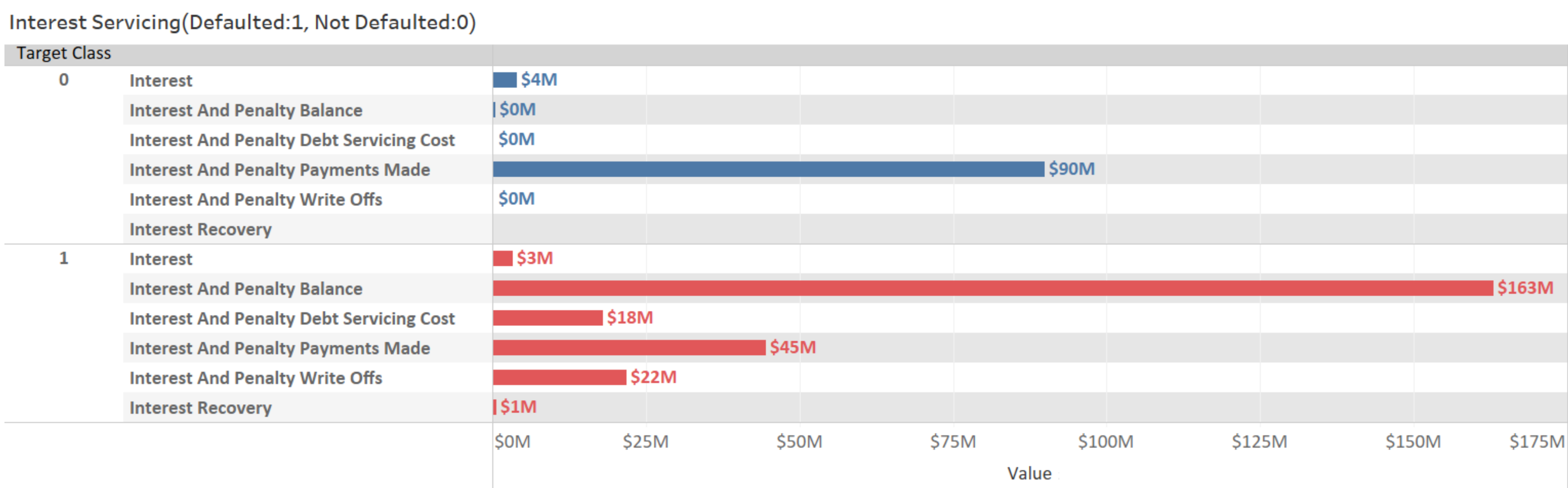


Exhibit 4: Liability Breakouts by Target Class

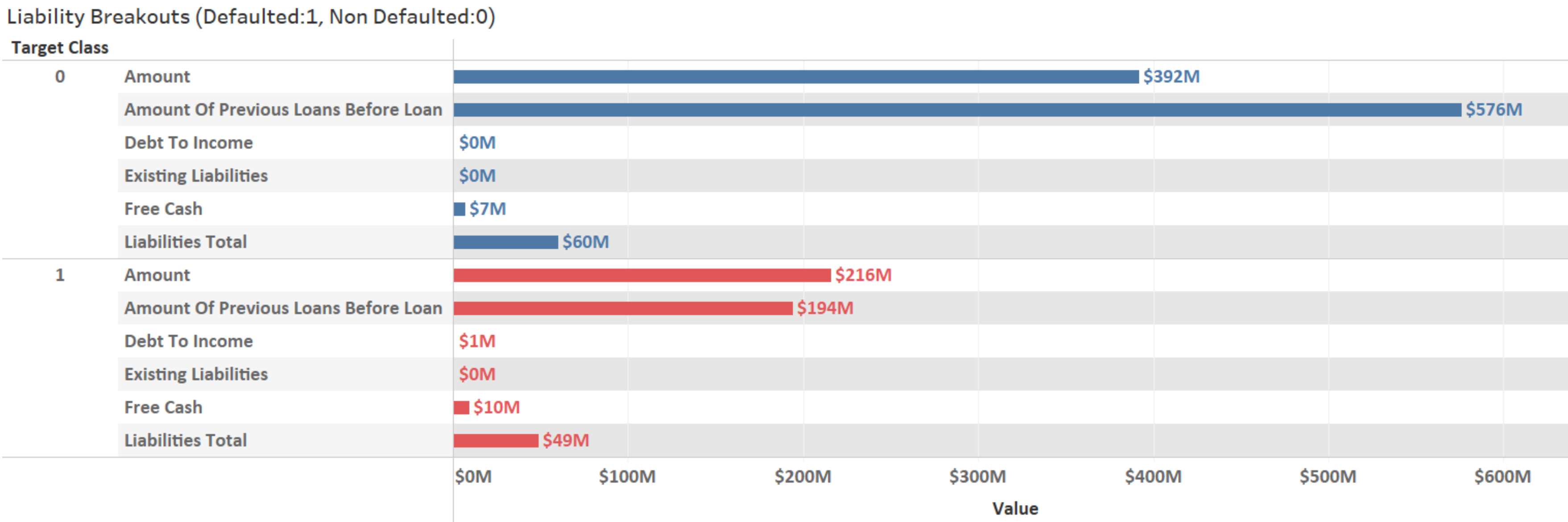


Exhibit 5: Credit Rating by Median Probability of Default

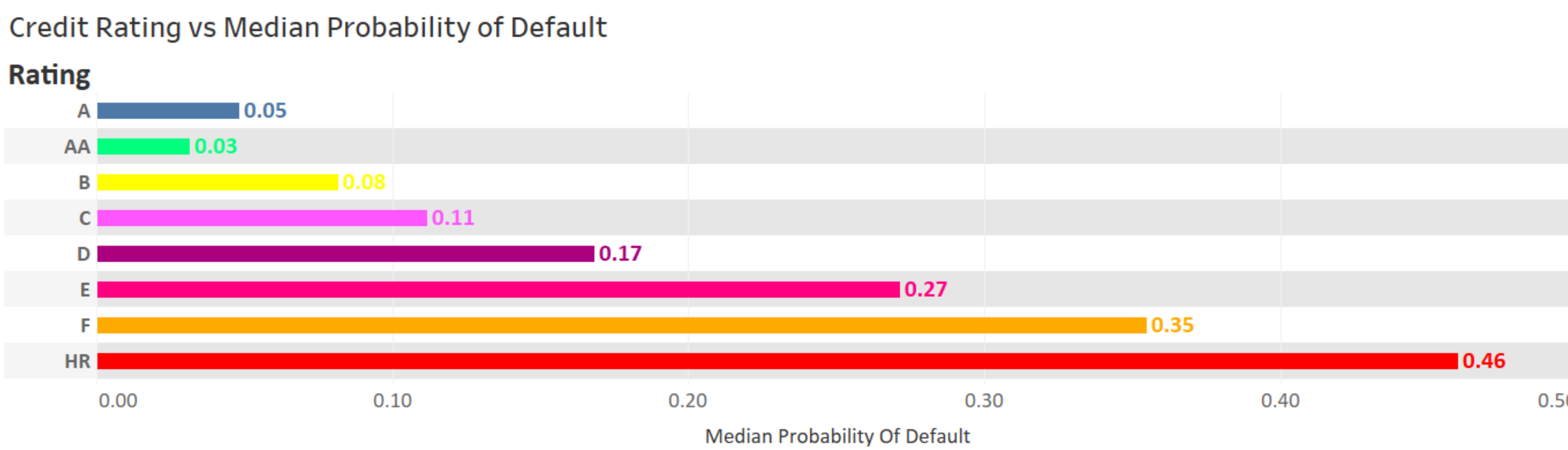


Exhibit 6: Credit Parameters by Target Class - I

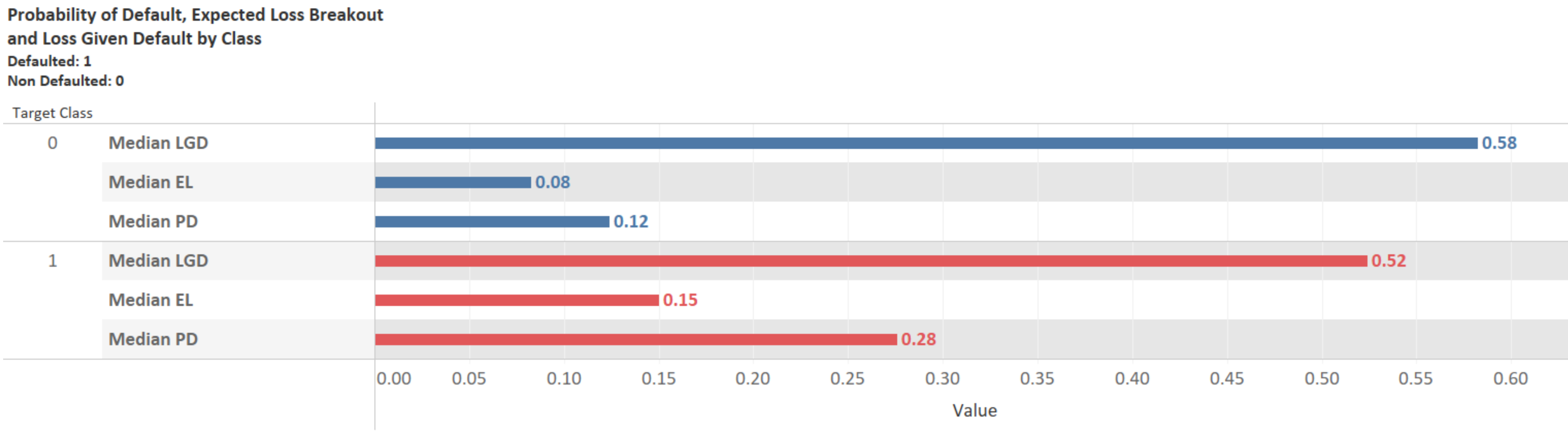
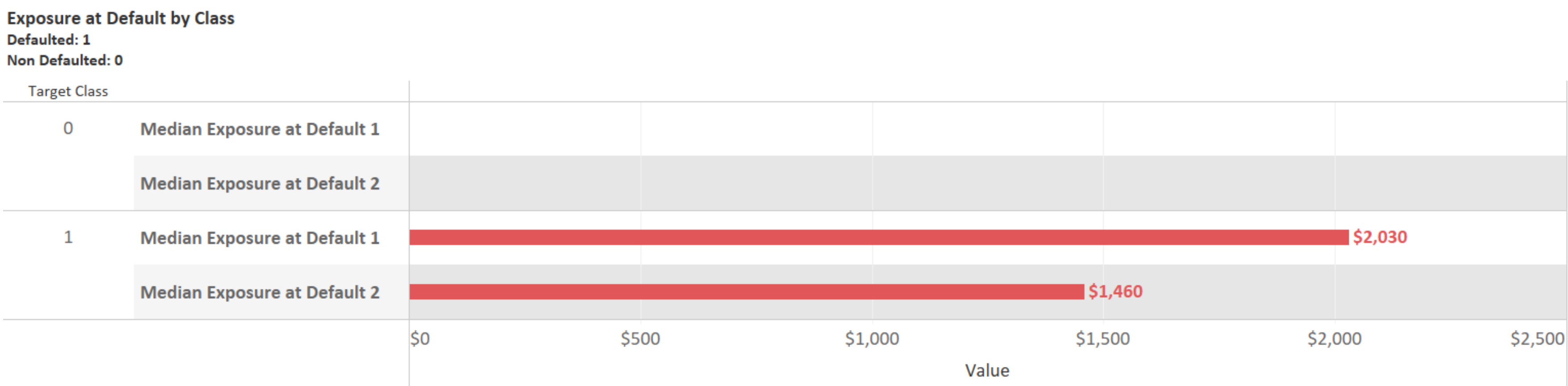


Exhibit 7: Credit Parameters by Target Class - II



Note:

EAD1: Exposure at default, outstanding principal at default, EAD 2: Exposure at default, loan amount less all payments prior to default

Exhibit 8: Principal Loan Data by Target Class

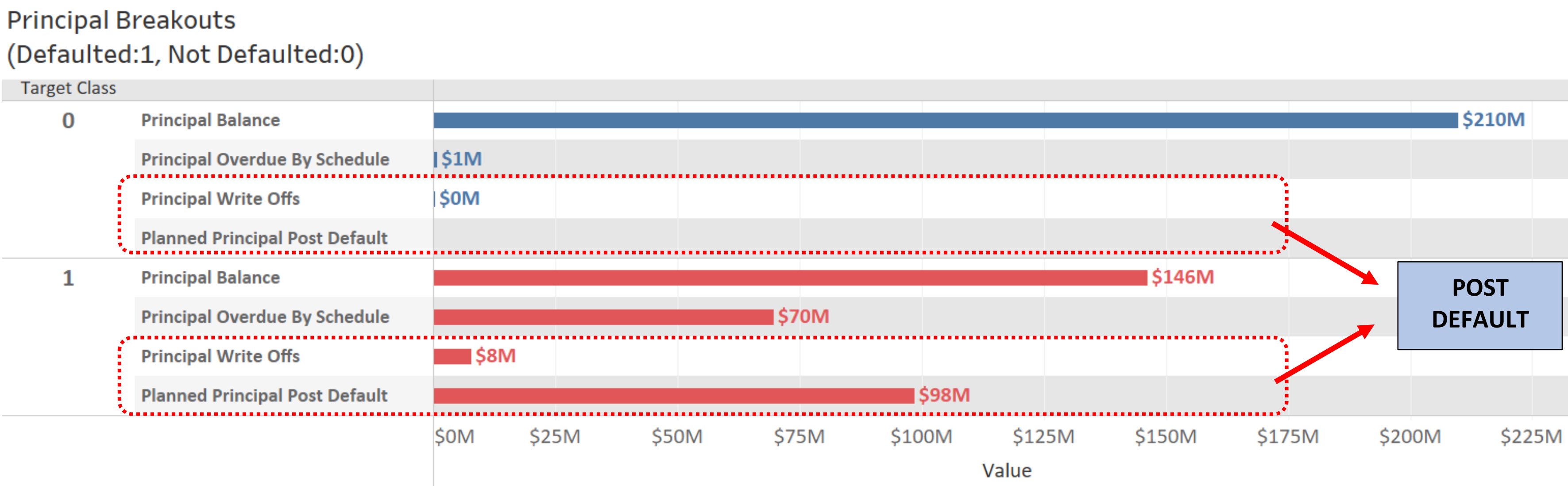


Exhibit 9: Status of Loan by Target Class

Status of Loan Category Counts by Class

Defaulted: 1
Not Defaulted: 0

Target Class	Status			
	Current	Late	Repaid	Grand Total
0	86,143 55.01%	9,839 6.28%	60,606 38.70%	156,588 100.00%
1	188 0.23%	64,406 79.87%	16,041 19.89%	80,635 100.00%
Grand Total	86,331 36.39%	74,245 31.30%	76,647 32.31%	237,223 100.00%

Notes for Table:
Target Class 0: Current+ Repaid = 93.71% of Total for Class
Target Class 1: Late, 79.87% of Total for Class

Exhibit 10: Employment Status Counts Breakdown by Target Class

Employment Status
Defaulted: 1
Not Defaulted: 0

Target Class	Employment Status							
	-1	0	2	3	4	5	6	Grand Total
0	140,054	5	456	13,782	428	1,147	595	156,467
1	60,581	27	728	16,278	875	860	1,205	80,554
Grand Total	200,635	32	1,184	30,060	1,303	2,007	1,800	237,021

Note:
1: Unemployed, 2: Partially employed, 3: Fully employed, 4: Self-employed, 5: Entrepreneur 6: Retiree

Exhibit 11: Work Experience/Home Ownership Type Counts Breakdown by Target Class

Work Experience/Home Ownership Category Breakouts
Defaulted: 1
Not Defaulted: 0

Target Class	Home Ownership Type	Work Experience					
		2-5 Yrs	5-10 Yrs	10-15 Yrs	15-25 Yrs	<2 Yrs	>25 Yrs
0	0			2		1	1
	1	394	941	917	1,369	205	1,567
	2	629	742	421	287	242	103
	3	436	608	407	348	204	248
	4	226	333	256	209	62	193
	5	15	23	36	31	7	58
	6	108	173	62	97	57	54
	7	162	285	244	326	100	232
	8	105	306	418	545	65	337
	9	18	36	63	96	3	76
	Total	2,093	3,447	2,826	3,308	946	2,869
1	0	8	8	3	5	2	8
	1	483	891	1,077	1,578	194	1,589
	2	872	1,106	728	598	341	209
	3	615	752	685	594	249	410
	4	322	533	458	438	103	484
	5	36	64	76	91	19	106
	6	144	183	150	119	53	86
	7	147	221	201	263	63	216
	8	73	207	355	532	29	418
	9	5	32	51	84	7	52
	Total	2,705	3,997	3,784	4,302	1,060	3,578

Notes:
0: Homeless, 1: Owner 2:Living with parents, 3:Tenant, pre-furnished property, 4: Tenant, unfurnished property, 5: Council house, 6: Joint tenant, 7: Joint ownership, 8: Mortgage, 9:Owner with encumbrance, 10:Other

Exhibit 12: Education/Country Type Counts Breakdown by Target Class

Education/Country Breakout Categories
Defaulted: 1
Not Defaulted: 0

Education	Country	Target Class	
		0	1
-1	EE	201	2
	ES		2
	FI	2,048	185
	Total	2,249	189
0	EE		8
	Total		8
1	EE	12,718	4,819
	ES	460	1,650
	FI	5,869	2,878
	Total	19,047	9,347
2	EE	2,079	2,490
	ES	131	654
	FI	288	798
	SK		4
	Total	2,498	3,946
3	EE	18,943	7,073
	ES	677	2,087
	FI	23,756	10,516
	SK	1	35
	Total	43,377	19,711
4	EE	44,575	17,282
	ES	2,592	7,265
	FI	5,687	3,713
	SK	13	175
	Total	52,867	28,435
5	EE	20,076	5,569

Notes:
1:Primary education, 2:Basic education, 3:Vocational education, 4:Secondary education, 5:Higher education

Exhibit 13: Amount of Previous Credit Breakdown by Target Class

Amount of Previous Credit Breakout
Defaulted: 1
Not Defaulted: 0

No Of Previous Loans Before Loan	Target Class	
	0	1
0	0	0
1	32,686	16,216
2	38,536	17,124
3	35,139	13,671
4	30,320	10,444
5	25,585	8,385
6	21,192	6,600
7	17,682	5,404
8	15,000	4,184
9	12,474	3,402
10	10,060	2,440
Grand Total	238,674	87,870

Exhibit 14: Days to Payments Percentage of Total Breakdown by Target Class

Days to Payments Percentage of Total by Target Class
Defaulted: 1
Non Defaulted: 0

Active Late Category	Target Class		
	0	1	Grand Total
0-7	95.84%	4.16%	100.00%
8-15	97.51%	2.49%	100.00%
16-30	86.07%	13.93%	100.00%
31-60	82.02%	17.98%	100.00%
61-90	60.72%	39.28%	100.00%
91-120	33.15%	66.85%	100.00%
121-150	4.34%	95.66%	100.00%
151-180	2.94%	97.06%	100.00%
180+	0.85%	99.15%	100.00%

4.0 Feature Evaluation/Extraction

The following further data exploration activities are described in this section. It includes a discussion on the following:

1) Missing value analysis;
2) Multi collinearity effects;
3) Correlation between predictor variable and target variable;
4) PCA analysis to identify how many principal components are able to explain the variance amongst the various continuous variables; and
5) Exploratory clustering analysis.

4.1 Missing Value Analysis

Of the 111 predictor variables, several of the categorical variables that do not have numerical value (e.g., Loan Id, Loan Number, etc.) were initially removed from the dataset.

Following this initial data cleansing effort, further analysis was conducted to evaluate features that had more than 10 pct missing data. The features that have more than 10 pct missing data are presented in Table 2. Given the large amount of predictor variables available in the dataset, these features were removed from the dataset. As can be seen later in the modeling effort, removal of these variables does not have significant effect on the prediction performance of the models.

Also note some of these variables such as Planned Principal Post Default, Planned Interest Post Default, those related to Recovery, those related to WriteOffs, and EAD1 and EAD2 should be removed as they were recorded following default and should not be used to predict the target class, and would have been removed from the dataset regardless of the number of missing values.

Table 2: Features with More than 10 Pct Missing Values

Features	Percentage of Total Missing
ContractEndDate	56.58%
DateOfBirth	100.00%
NrOfDependants	84.99%
WorkExperience	84.60%
PlannedPrincipalTillDate	77.04%
CurrentDebtDaysPrimary	63.27%
DebtOccuredOn	63.27%
CurrentDebtDaysSecondary	59.70%
DebtOccuredOnForSecondary	59.70%
PlannedPrincipalPostDefault	66.01%
PlannedInterestPostDefault	66.01%
EAD1	66.01%
EAD2	66.01%
PrincipalRecovery	66.01%
InterestRecovery	66.01%
RecoveryStage	41.56%
StageActiveSince	38.00%
EL_V1	94.55%
Rating_V1	94.55%
Rating_V2	89.40%
ActiveLateCategory	63.51%
WorseLateCategory	34.52%
CreditScoreEsMicroL	13.49%
CreditScoreEsEquifaxRisk	94.85%
CreditScoreFiAsiakasTietoRiskGrade	68.98%
CreditScoreEeMini	45.17%
PrincipalWriteOffs	63.55%
InterestAndPenaltyWriteOffs	63.55%
InterestAndPenaltyBalance	26.65%
PreviousRepaymentsBeforeLoan	37.12%
PreviousEarlyRepaymentsBefoleLoan	74.85%
GracePeriodStart	75.01%
GracePeriodEnd	75.01%
NextPaymentDate	59.58%
NextPaymentNr	39.82%
NrOfScheduledPayments	39.82%
ReScheduledOn	62.77%
PrincipalDebtServicingCost	63.55%
InterestAndPenaltyDebtServicingCost	63.55%
ActiveLateLastPaymentCategory	59.70%

Following the removal of the features noted above, the "surviving" features were further evaluated for "missingness". The percentage of datapoints missing for these features were less than 10% of the total data points. The actual numbers of the missing data points for the features that had missing values are presented on Exhibit 15.

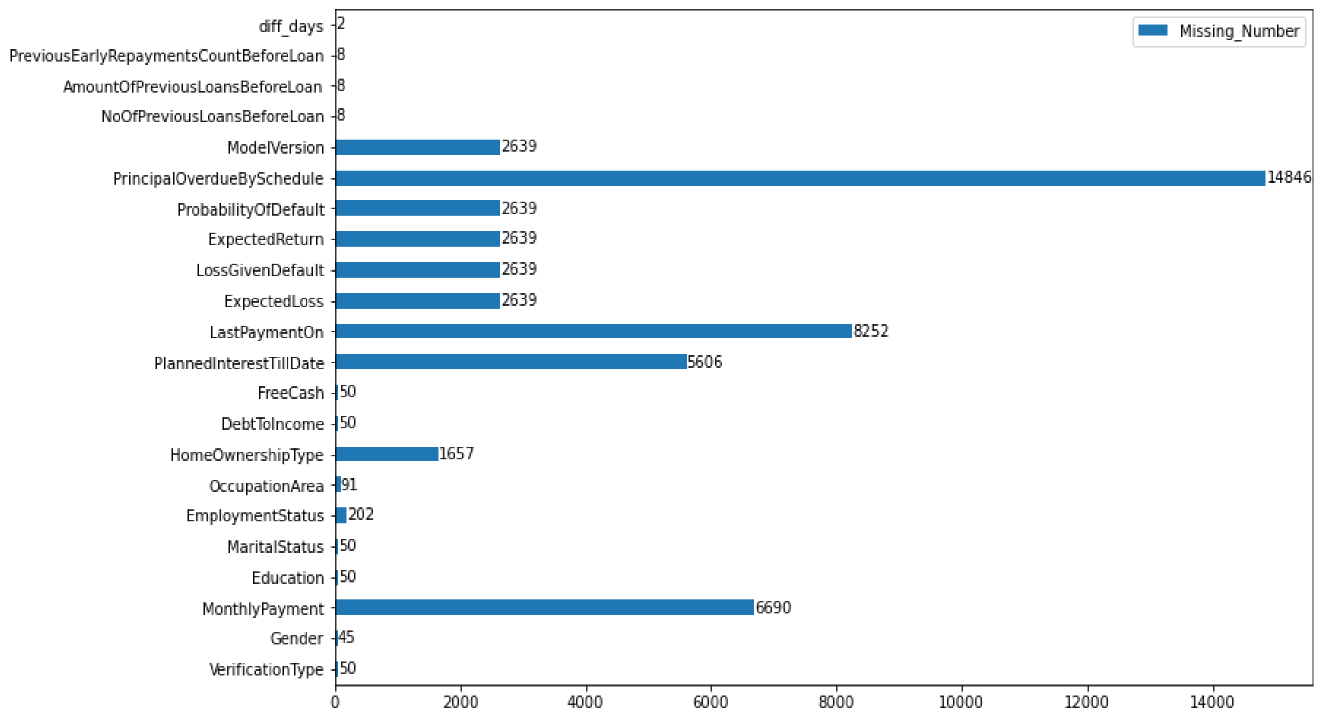
Following the removal of the rows in the dataset with these missing values, the total number of data points remaining in the dataset was 211,240, which is 10.90% less than the original number of 237,223 in the dataset.

The breakdown by target class of the final dataset used in the modeling is presented in Table 3 below:

Table 3:
Target Class Breakdown, Final Dataset

Target Class	Count of Target Class	% of Total Count of Target Class
0	137,895	65.28%
1	73,345	34.72%
Total	211,240	100.00%

Exhibit 15: Missing Values Count for Surviving Features



The distribution of the dataset and the breakdown by target class are similar to the original dataset with the missing values in it (see Table 1). A total of 58 predictor features survived in the final dataset used for further analysis and modeling. Final data cleansing consisted of "minmax" scaling of the continuous variables and one hot dummy encoding (Heaton, J, 2022a) of the categorical variables, where necessary. Note that several of the categorical variables were already assigned "ordinal" scores and did not require dummy encoding. Following this data cleansing and the one hot dummy encoding, 71 predictor variables were generated for the modeling effort.

4.2 Correlation Analysis

Table 4: Correlation Coefficients Between Variables

Variable_1	Variable_2	Correlation Coeff
MaritalStatus	DebtToIncome	0.767
DebtToIncome	MaritalStatus	0.767
NoOfPreviousLoansBeforeLoan	AmountOfPreviousLoansBeforeLoan	0.77
AmountOfPreviousLoansBeforeLoan	NoOfPreviousLoansBeforeLoan	0.77
UseOfLoan	MaritalStatus	0.774
MaritalStatus	UseOfLoan	0.774
MaritalStatus	OccupationArea	0.774
OccupationArea	MaritalStatus	0.774
Interest	ProbabilityOfDefault	0.785
ProbabilityOfDefault	Interest	0.785
EmploymentStatus	DebtToIncome	0.787
DebtToIncome	EmploymentStatus	0.787
AppliedAmount	MonthlyPayment	0.79
MonthlyPayment	AppliedAmount	0.79
UseOfLoan	EmploymentStatus	0.791
EmploymentStatus	UseOfLoan	0.791
EmploymentStatus	OccupationArea	0.791
OccupationArea	EmploymentStatus	0.791
Interest	ExpectedLoss	0.799
ExpectedLoss	Interest	0.799
ExpectedLoss	ProbabilityOfDefault	0.858
ProbabilityOfDefault	ExpectedLoss	0.858
MaritalStatus	EmploymentStatus	0.928
EmploymentStatus	MaritalStatus	0.928
AppliedAmount	Amount	0.947
Amount	AppliedAmount	0.947

Analysis was conducted to assess for multi-collinearity of the surviving predictor variables. This analysis was conducted on unscaled continuous variable data and available categorical data. The predictor variables that have correlation coefficient greater than 0.75 between each other are presented on Table 4. Only 2 pairs (or 4 variables) of the 71 surviving predictor variables have correlation coefficient exceeding 0.9.

These two pairs are marital status and employment status and amount and applied amount. Applied amount is the actual amount requested by the consumer and the amount is the amount of loan that was authorized by the financial institution.

Because the correlation coefficients outside of these 4 variables are not higher than 0.9 (see Table 4), multi-collinearity effects between predictor variables are not considered significant and none of the surviving variables were removed from further analysis.

Also evaluated was the correlation coefficient between the predictor variable and the target variable, and, as expected, a few of the predictor variables, Expected Loss, Probability of Default, Principal_Overdue_by_Schedule, and Status_Late have correlation coefficients exceeding 0.4 (see Table 5). These variables are estimates made during the application process and during loan servicing and not generated following default and hence were not removed from the predictor variable set.

Table 5: Correlation Coefficients Between Variables and Target Variable

Variable_Name	Defaulted
Rating_C	-0.182
Status_Repaid	-0.175
Rating_B	-0.136
AmountOfPreviousLoansBeforeLoan	-0.120
PrincipalPaymentsMade	-0.118
NoOfPreviousLoansBeforeLoan	-0.117
ModelVersion	-0.108
LossGivenDefault	-0.098
Rating_D	-0.080
Rating_AA	-0.070
EmploymentDurationCurrentEmployer_UpTo5Years	-0.067
EmploymentDurationCurrentEmployer_Other	-0.049
diff_days	-0.035
Country_FI	-0.032
MonthlyPaymentDay	-0.029
LoanDuration	-0.016
InterestAndPenaltyPaymentsMade	-0.011
LiabilitiesTotal	0.005
EmploymentDurationCurrentEmployer_UpTo1Year	0.005
PreviousEarlyRepaymentsCountBeforeLoan	0.013
EmploymentDurationCurrentEmployer_Retiree	0.013
IncomeFromLeavePay	0.019
Education	0.020
IncomeOther	0.032
HomeOwnershipType	0.033
EmploymentDurationCurrentEmployer_TrialPeriod	0.035
Amount	0.041
Country_SK	0.045
IncomeFromChildSupport	0.046
IncomeFromSocialWelfare	0.046
ExistingLiabilities	0.049
Restructured_True	0.068
AppliedAmount	0.075
EmploymentDurationCurrentEmployer_UpTo4Years	0.076
IncomeFromFamilyAllowance	0.082
FreeCash	0.084
IncomeFromPension	0.085

Table 5 Continued: Correlation Coefficients Between Variables and Target Variable

Variable_Name	Defaulted
EmploymentDurationCurrentEmployer_UpTo3Years	0.091
NewCreditCustomer_True	0.102
EmploymentDurationCurrentEmployer_UpTo2Years	0.108
PrincipalBalance	0.111
RefinanceLiabilities	0.119
Rating_E	0.120
IncomeFromPrincipalEmployer	0.144
MonthlyPayment	0.160
PlannedInterestTillDate	0.187
OccupationArea	0.237
DebtToIncome	0.245
Rating_HR	0.249
UseOfLoan	0.254
Rating_F	0.256
ExpectedReturn	0.273
ActiveScheduleFirstPaymentReached_True	0.277
MaritalStatus	0.282
EmploymentStatus	0.286
Country_ES	0.298
Interest	0.354
ExpectedLoss	0.409
ProbabilityOfDefault	0.432
PrincipalOverdueBySchedule	0.487
Status_Late	0.758
Defaulted	1.000

4.3 Principal Component Analysis

A Principal Component Analysis (PCA) analysis was conducted to perform exploratory analysis and to evaluate whether the variance in the predictor variables and separation in the target class variables can be explained by reducing dimensions of the predictor variables. The scaling was performed with standard scaler.

An analysis was conducted using only 5,000 dataset points. This analysis indicates that 50% of the variance can be explained with 5 principal components (see Exhibit 16).

Separability in the target class is not clearly discernable when 3 principal components are evaluated (see Exhibit 17).

Exhibit 16: Explained Variance vs Principal Component No.

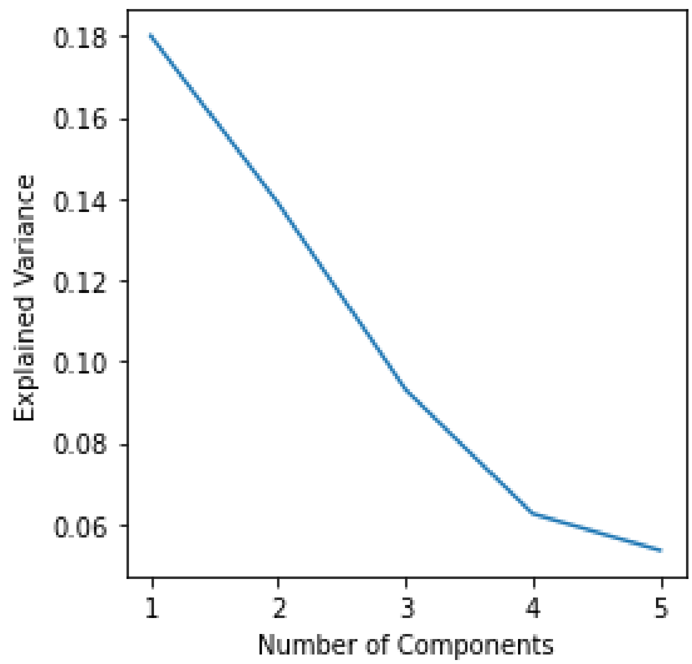
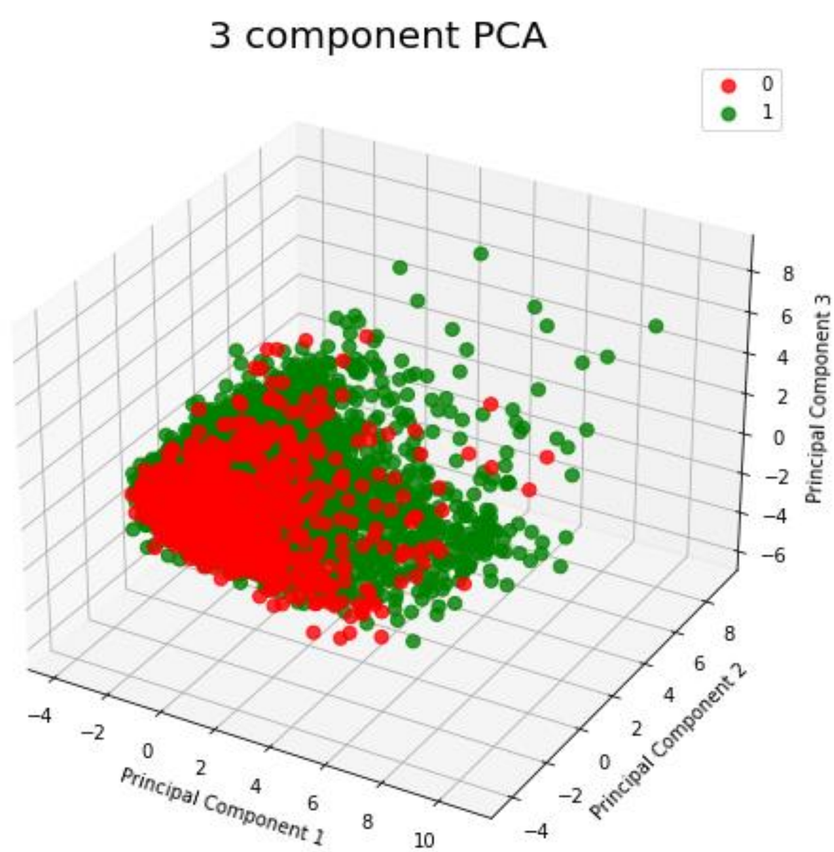


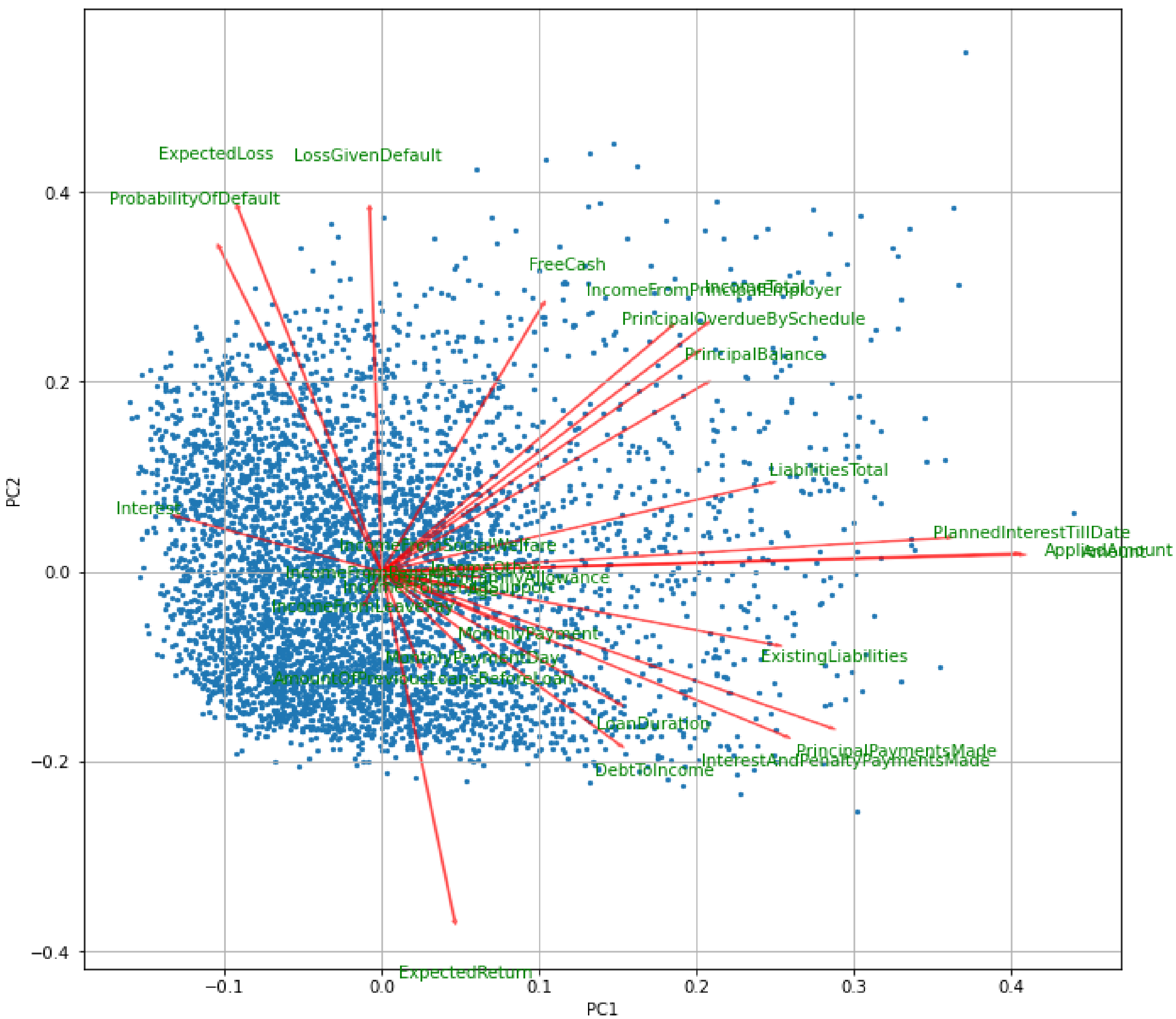
Exhibit 17: Target Class Separation from Three Principal Components



A PCA Bi Plot results from this analysis is presented on Exhibit 18. Based on the "vector" representation of some of the features, it does appear that the first two components may be a reasonable assimilator of a limited set of the continuous predictor variables.

Given the limited separability in target classes noted in Exhibit 17 and a large number of categorical variables (greater than 50 pct of surviving predictor variables), PCA components were not included in the modeling effort and the 71 surviving predictor variables were carried forward for the modeling effort.

Exhibit 18: PCA Bi Plot



4.4 Exploratory Clustering Analysis

4.4.1 Overview

Exploratory unsupervised learning in the form of clustering analysis was performed using the K-means clustering algorithm using the sklearn package.

4.4.2 Results

Elbow analysis that shows a plot of cumulative within cluster sum of squares (WCSS) vs No. of Clusters is depicted on Exhibit 19 for fully scaled data (all values between 0 and 1). Cumulative WCSS (sum of squared distance (SSE) between the data points and their respective assigned clusters centroid) is an indicator of the spread within each cluster. A clear elbow cannot be discerned from the exhibit, and the value of the cumulative WCSS of 500,000 (at the 12th cluster point), which is indicative of K means not adequately separating data into the first 12 clusters.

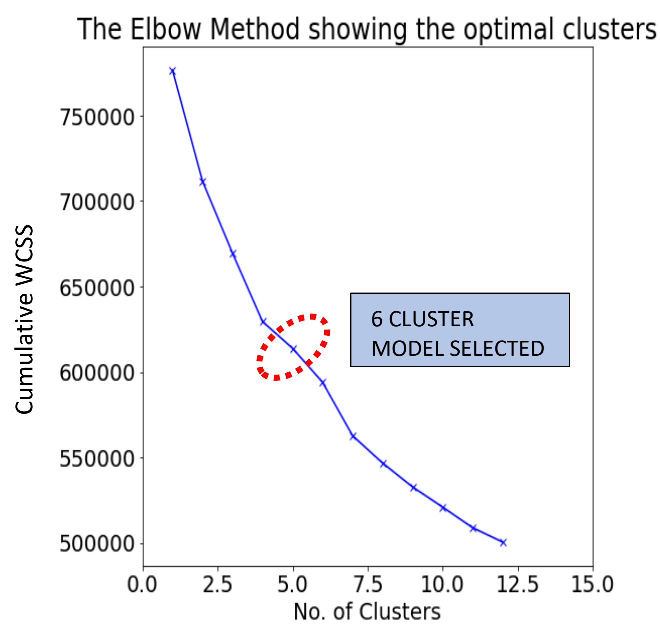


Exhibit 19: Elbow Analysis, K-means Clustering

For purpose of this analysis, 6 cluster K-means was further investigated. Values of maximum spread between cluster centroids (inter cluster spread) ranged from 0.096 to 0.424 (scale of 0 to 1) for the 10 top ranked continuous variables (Table 6). This spread appears low and maybe an indicator of poor performance of k-means on the data. Lastly, a scatter plot between two continuous variables further demonstrates of lack of separation into clusters by this method (see Exhibit 20).

Because of this apparent lack of promise, unsupervised learning was not further investigated.

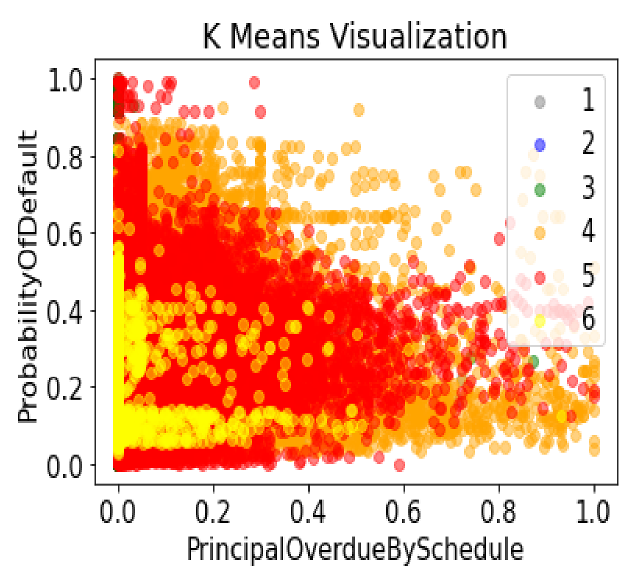


Exhibit 20: K-means Visualization

Table 6:

Maximum Separation of Farthest Centroids, Input Features, Clustering Analysis

Rank	Spread	Feature Names
1	0.424	DebtToIncome
2	0.398	LossGivenDefault
3	0.223	PrincipalBalance
4	0.172	ProbabilityOfDefault
5	0.163	PrincipalPaymentsMade
6	0.152	ExpectedLoss
7	0.135	PlannedInterestTillDate
8	0.109	PrincipalOverdueBySchedule
9	0.098	AppliedAmount
10	0.096	Interest

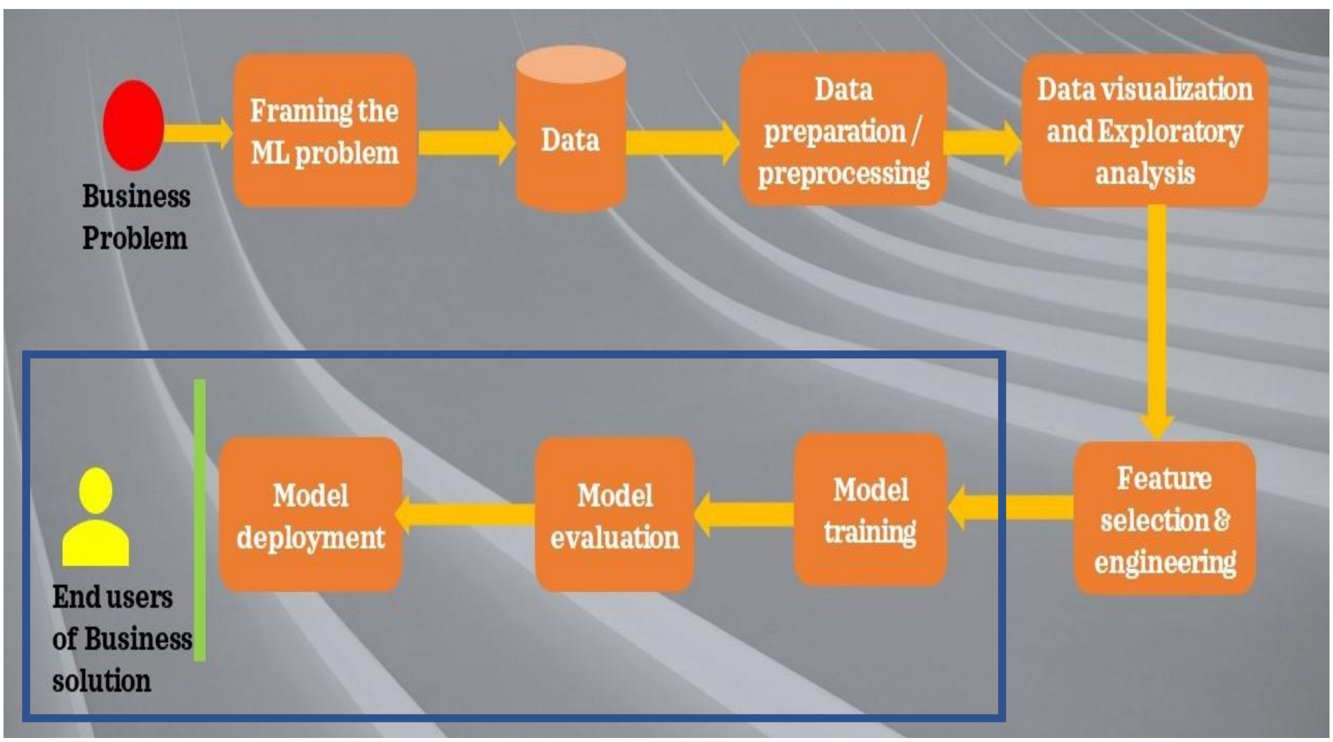
5.0 Machine Learning Modeling

Supervised learning via classification modeling was conducted using the final dataset (from Table 3) that contains 71 predictor variables and 1 target variable (see blue rectangle in schematic below for the work components in this phase). Python packages sklearn and tensorflow/keras were utilized for the development of the machine learning models. PyTorch with a PySyft wrapper was utilized for the remote (federated) machine learning phase of the project.

The final dataset was split into train (80%) and test (20%) components using sklearn's in built functions. The sklearn models were trained with 5-fold cross validation on the train portion of the dataset and its performance was evaluated on the test portion of the dataset. Training was conducted using two optimization approaches for comparative analysis: grid search optimization and Bayesian optimization.

For Tensorflow/keras, the model was first trained and tested on then full dataset with default parameters without cross validation. For the cross validation and testing portion of the modeling, because of time complexity, the model was trained with 3-fold cross validation on 10% of the dataset. This fraction was split into 80% train and test components.

The focus of PyTorch and PySft modeling effort was to identify the process to be used to train, build, and test the model on a remote dataset and to evaluate its effectiveness in achieving results that are comparable to the other models. Accordingly, to reduce the time required to run the models, 5% of the final dataset was used in the modeling effort. Similar to the workflow for the other models, this fraction of the final dataset was split into train (80%) and test (20%) components.



5.1 Logistic Regression

5.1.1 Model Overview and Results

Logistic regression models a relationship between predictor variables and a categorical response variable (James G, 2017). The log odds per logistic regression for a binary classification problem is given as follows:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta 0 + \beta 1\ X \quad \text{(James G, 2017)}$$

Where: p(X) is the probability that takes a value between 0 and 1, and is used as a predictor for one of the two classes for a binary classification problem based on its value. If the value is between 0 and 0.5, it is assigned to class 0; otherwise it is assigned to class 1.

sklearn's logistic regression module was used to model the logistic regression on the final dataset (sklearn-a). The modeling was conducted as follows:

class sklearn.linear_model.Logistic Regression(*penalty, C, solver, max_iter=200, l1_ratio*).

The noted hyperparameters were tuned per Grid Search CV with 5-fold cross validation per Exhibit 21. Results are provided on Exhibits 22-25.

Exhibit 21: LR Model Hyperparameters

Hyper-parameter	Range	Best Value
Penalty	L1, L2, Elasticnet	L1
C	1,5,10	5
Solver	Lbfgs, liblinear, and saga	liblinear
L1_ratio	0.2,0.6	Not Applicable for L1 Penalty

Exhibit 23: ROC Curve: Logistic Regression/Best Model Following Tuning

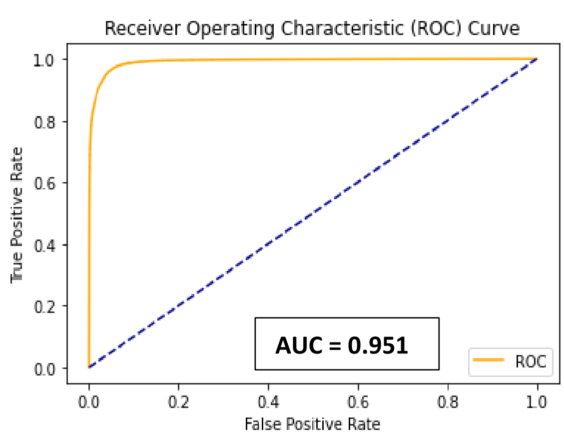


Exhibit 22: LR Model Grid Search CV Results

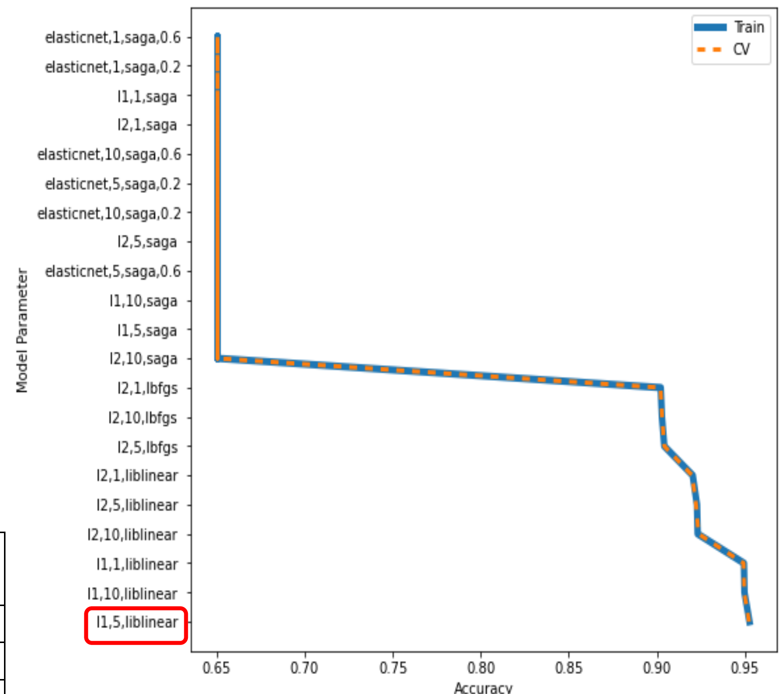


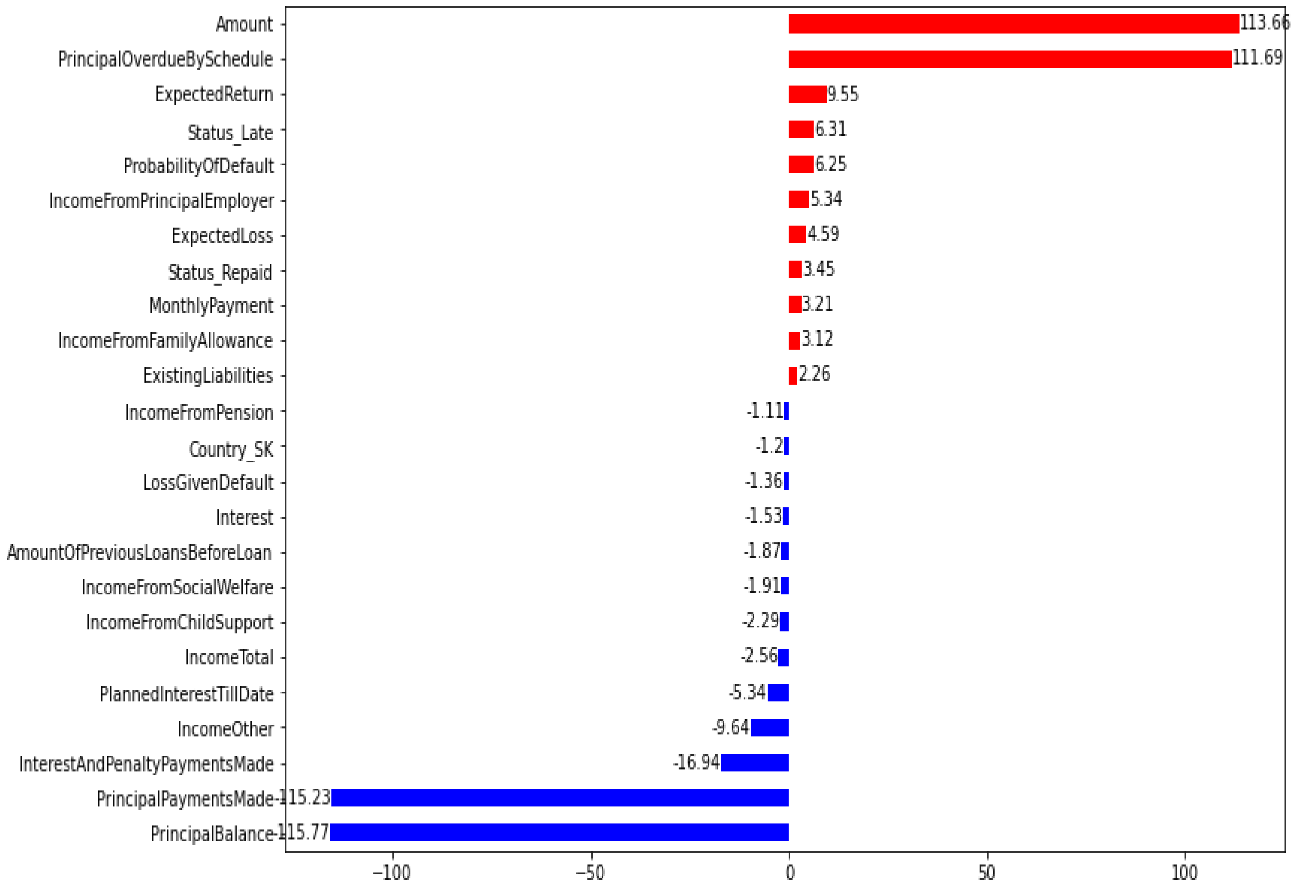
Exhibit 24: Performance Evaluation: Logistic Regression

Confusion Matrix, Test Dataset Following Tuning:

	Predicted No	**Predicted Yes**
Actual No	26,280	907
Actual Yes	928	13,687

Parameter	Value Following Tuning
RMSE	0.209
Precision	0.938
Accuracy	0.956
Recall	0.936
F1_Score	0.937

Exhibit 25: Important Features Coefficients: Logistic Regression/Best Model Following Tuning



5.1.2 Best Model Parameters

Based on the results of the tuning, the highest mean CV score of 0.952 (Exhibit 22) was obtained with the best values of hyperparameters noted on Exhibit 21. The best model was evaluated on the test dataset using these best model parameters. The results from this evaluation indicate that precision, recall, accuracy, F_1 score were all higher than 0.9 (Exhibit 24). The area under the curve of the receiver operating characteristic curve was 0.951 (Exhibit 23), which indicates that the model is effective in separating the target class between 0 and 1.

Top 5 positive coefficients (i.e., β1 values) were obtained *for loan amount*, *PrincipalOverduebySchedule*, *ExpectedReturn*, *StatusLate*, and *ProbabilityOfDefault*. Top 5 negative coefficients were obtained for *PrincipalBalance*, *PrincipalPaymentMade*, I*nterestAndPenaltyPaymentsMade*, *IncomeOther*, and *PlannedInterestTillDate* (see Exhibit 25). Positive coefficients drive the target class to 1 and negative coefficients drive the target Class to 0. Exhibit 25 can be used for interpretation of the best "logistic regression" model and to identify the features that drove the classification prediction in this model. Note that this exhibit only shows those features that have a regression coefficient of greater than or equal to an absolute value of 1.0.

5.2 Multinomial Bayes

5.2.1 Model Overview and Results

Multinomial Bayes models help predict that particular observation belongs to a certain class (Y=k) based on the prior probability of the occurrence of a class (πk) and the density function of X (fk(x)) that comes from an observation comes from that kth class:

$$\Pr(Y = k|X = x) = \frac{\pi k f_k(x)}{\sum_1^l \pi l\, f_l(x)}$$ (Hastie, T., 2017)

The denominator is ignored in the calculation.

sklearn's multinominal bayes module was used to model the logistic regression on the final dataset (sklearn-b). The modeling was as follows:

class sklearn.naive_bayes.MultinomialNB(

alpha, *fit_prior=True*)

The noted hyperparameters were tuned per Grid Search CV with 5-fold cross validation per Exhibit 26. Results are provided on Exhibits 27-30.

Exhibit 26: MNB Model Hyperparameters

Hyper- -parameter	Range	Best Value
Alpha	1E-4, 1E-2, 1E-1, 1	1

Exhibit 27: Performance Evaluation: Multinomial Bayes

Confusion Matrix, Test Dataset Following Tuning:

	Predicted No Default	Predicted Yes Default
Actual No Default	24,283	2,904
Actual Yes Default	3,762	10,853

Parameter	Value Following Tuning
RMSE	0.399
Precision	0.789
Accuracy	0.841
Recall	0.743
F1_Score	0.765

Exhibit 28: MNB Grid Search CV Results

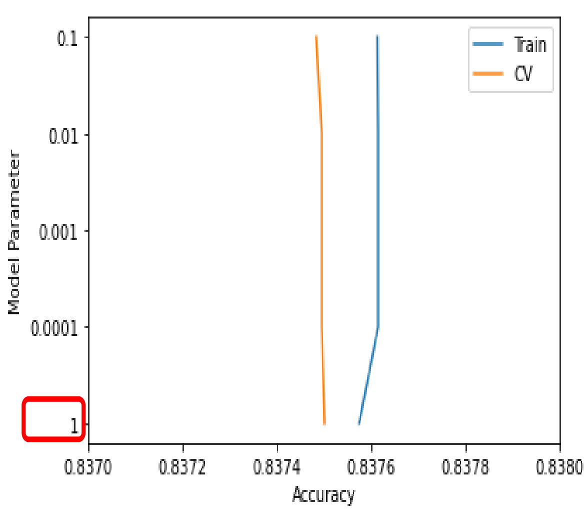


Exhibit 29: ROC Curve: Multinomial Bayes/Best Model Following Tuning

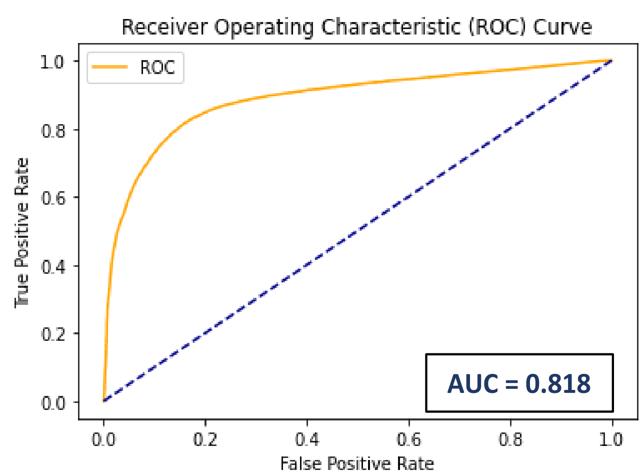
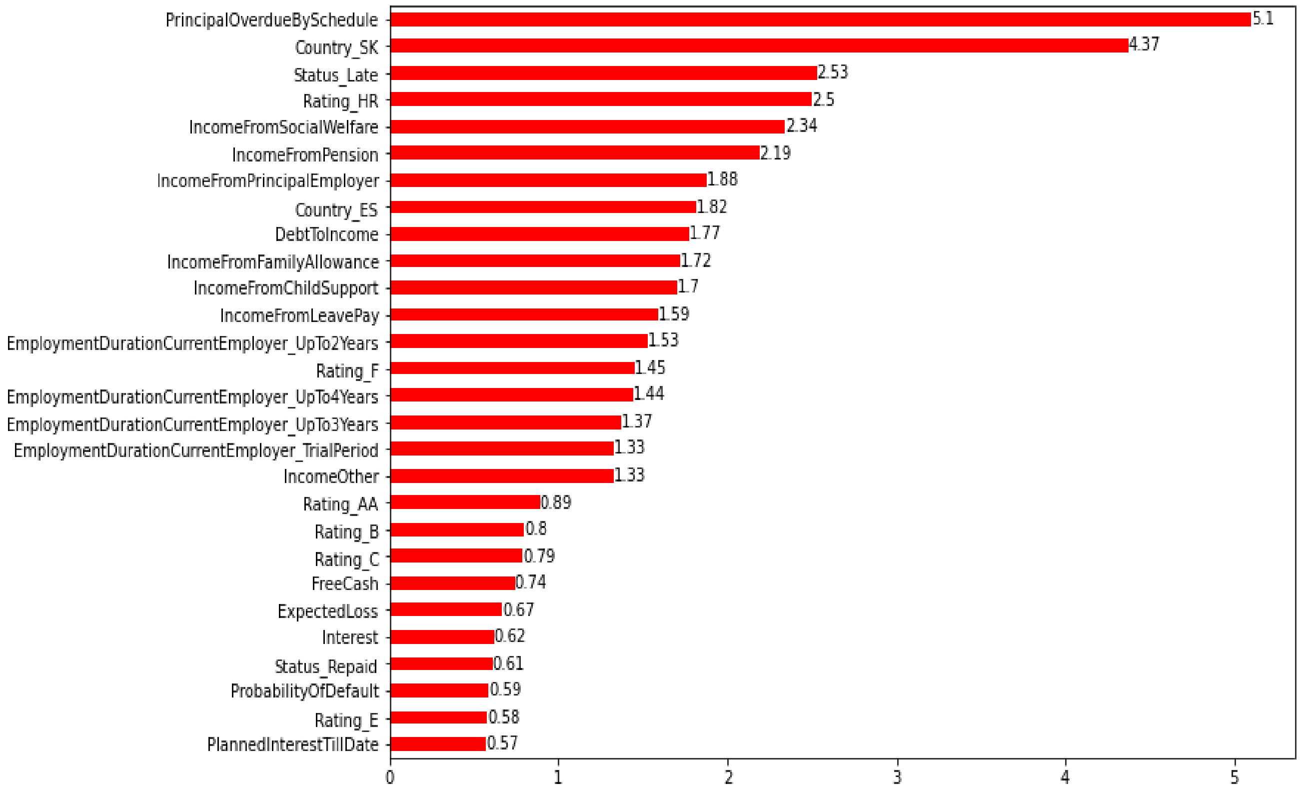


Exhibit 30: Important Features Coefficients Difference Between Classes Naïve Bayes/Best Model Following Tuning



5.2.2 Best Model Parameters

Based on the results of the tuning, the highest mean CV score of 0.838 (Exhibit 28) was obtained with the best values of hyperparameters noted on Exhibit 26. The best model was evaluated on the test dataset using these best model parameters. The results from this evaluation indicate that precision, recall, accuracy, F_1 score were all lower than 0.9 (between 0.7 and 0.9) and were lower than the other models evaluated in this study (Exhibit 27). The area under the curve of the receiver operating characteristic curve was 0.818 (Exhibit 29), which indicates that the model is less effective than the other evaluated models in separating the target class between 0 and 1.

The model provides estimates of the probability that a feature predicts a class 0 and a class 1 based on its values. Exhibit 30 depicts estimates of the absolute difference between these values for the features (for estimated values greater than or equal to 0.5) used in the modeling. Higher values of these estimates can be used an indicator of the relative importance of the feature in this model for separating the result for the target into it two disparate classes (0 or 1).

5.3 Decision Tree

5.3.1 Model Overview and Results

Decision Tree is a Supervised learning algorithm that is used for classification. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

Decision tree classifiers use either Gini Impurity Index or Information Gain (entropy) at a given node to create a split in the decision tree. Features that have the lowest Gini Impurity Index or highest Information Gain are placed at a given node.

sklearn's Decisiontree Classifier module was used to model the logistic regression on the final dataset (sklearn-c). The modeling was as follows:

class sklearn.tree.DecisionTreeClassifier (*criterion*, *max_depth*)

The noted hyperparameters were tuned per Grid Search CV with 5-fold cross validation per Exhibit 31. Results are provided on Exhibits 32-35.

Exhibit 31: Decision Tree Model Hyperparameters

Hyper- -parameter	Range	Best Value
Criterion	Gini and Entropy	Entropy
Max_Depth	5,10,20	20

Exhibit 32: Decision Tree Grid Search CV Results

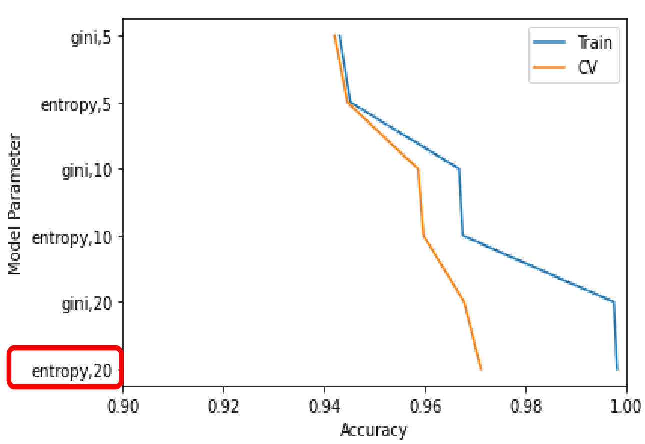


Exhibit 33: Performance Evaluation: Decision Tree

Confusion Matrix, Test Dataset Following Tuning:

	Predicted No	**Predicted Yes**
Actual No	26,663	554
Actual Yes	591	14,024

Parameter	**Value Following Tuning**
RMSE	0.166
Precision	0.962
Accuracy	0.973
Recall	0.960
F1_Score	0.961

Exhibit 34: ROC Curve: Decision Tree/Best Model Following Tuning

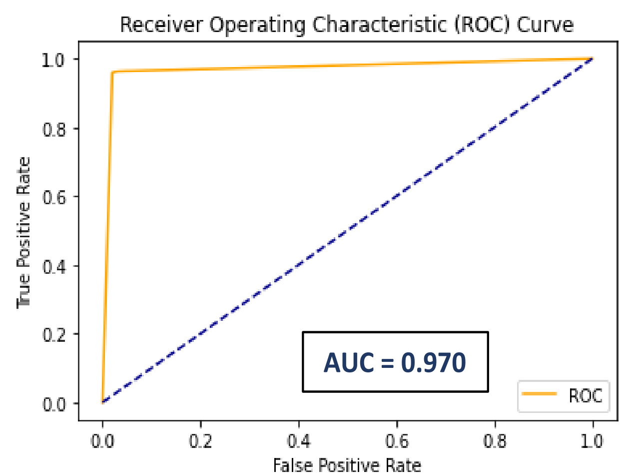
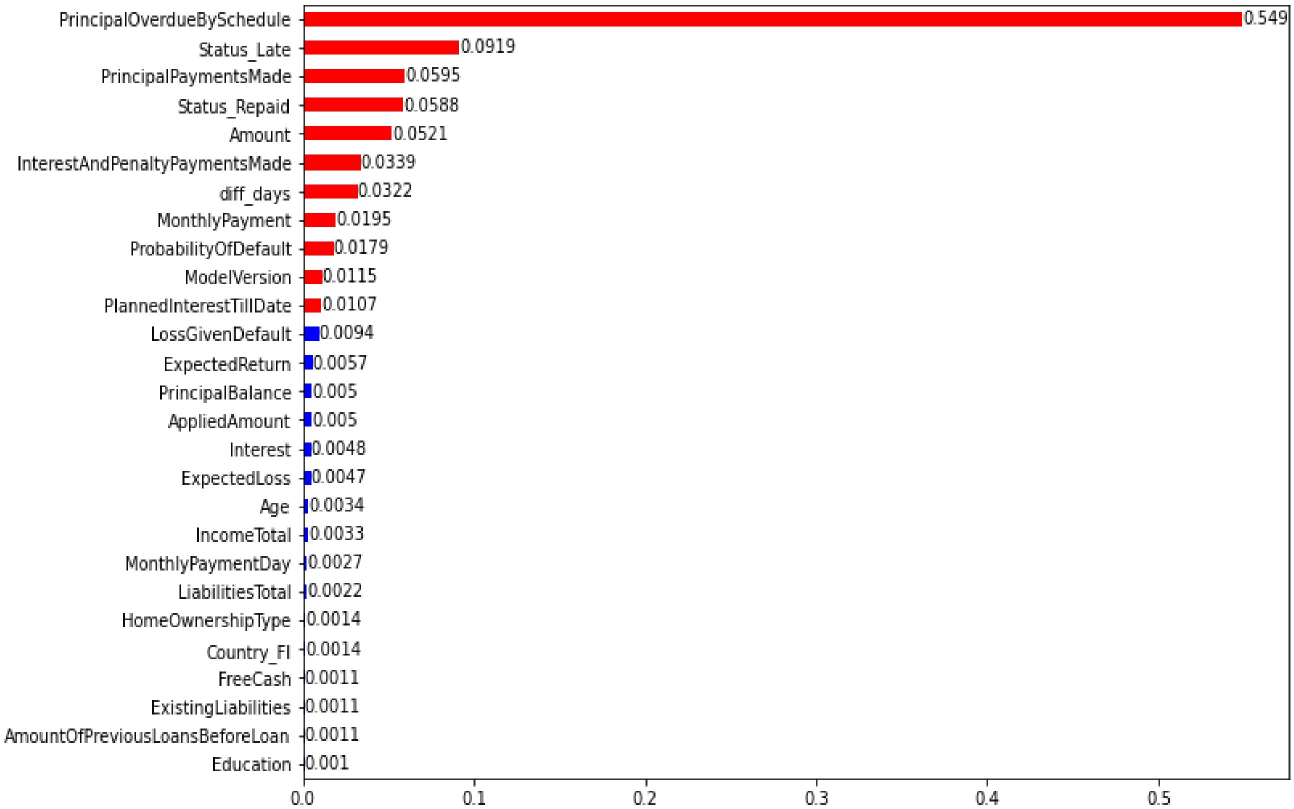


Exhibit 35: Features Importance Decision Tree/Best Model Following Tuning



5.3.2 Best Model Parameters

Based on the results of the tuning, the highest mean CV score of 0.971 (Exhibit 32) was obtained with the best values of hyperparameters noted on Exhibit 31. The best model was evaluated on the test dataset using these best model parameters. The results from this evaluation indicate that precision, recall, accuracy, F_1 score were all higher than 0.9 (Exhibit 33). The area under the curve of the receiver operating characteristic curve was 0.970 (Exhibit 34), which indicates that the model is effective in separating the target class between 0 and 1.

The five features with the most importance to model prediction were *PrincipalOverduebySchedule*, *StatusLate*, PrincipalPaymentsMade, *StatusRepaid*, and *loan amount* (see Exhibit 35). Exhibit 35 can be used for interpretation of the best "decision tree" model and to identify the features that drove the classification prediction in this model. Note that this exhibit only shows those features that have a variable importance of greater than or equal to 0.001.

5.4 Ensemble Forest

5.4.1 Model Overview and Results

Ensemble AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

For our analysis, the Ensemble Model was built on a base estimator of a Decision Tree Classifier with a maximum depth of 1. The Decision Tree Classifier is considered a weak classifier as it only has a maximum depth of 1. In this study, sklearn's Adaboot classifier that implements the algorithm known as AdaBoost-SAMME is utilized (Zhu, H., 2009). Despite the classifier much weaker than the Decision Tree Classifier (max_depth of 20 in Section 5.3), the results of this model do not suffer much in comparison.

sklearn's ensemble AdaBoost Classifier module was used to model the logistic regression on the final dataset (sklearn-d). The modeling was as follows:

class sklearn.ensemble.AdaBoostClassifier (*n_estimators, learning_rate*)

The noted hyperparameters were tuned per Grid Search CV with 5-fold cross validation per Exhibit 36. Results are provided on Exhibits 37-40.

Exhibit 36: Ensemble Forest Model Hyperparameters

Hyper- -parameter	Range	Best Value
N_estimators	5,10,20, 50,100	100
L_rate	.1, .5, 1.0, 5.0,10.0	1.0

Exhibit 38: Performance Evaluation: Ensemble Forest

Confusion Matrix, Test Dataset Following Tuning:

	Predicted No	Predicted Yes
Actual No	26,238	949
Actual Yes	1,276	13,339

Parameter	Value Following Tuning
RMSE	0.231
Precision	0.934
Accuracy	0.947
Recall	0.913
F1_Score	0.923

Exhibit 37: Ensemble Forest Grid Search CV Results

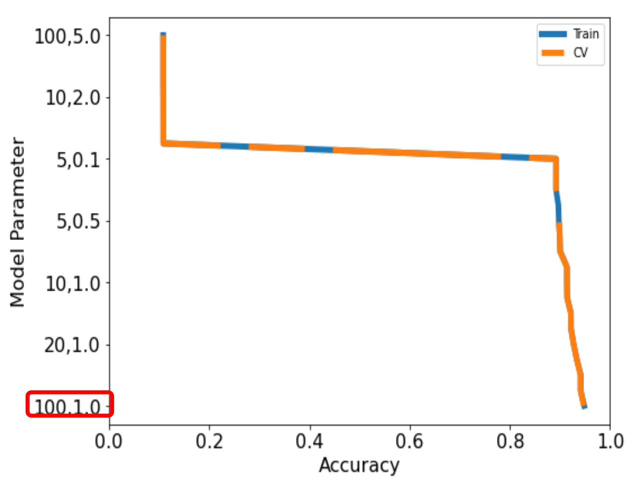


Exhibit 39: ROC Curve: Ensemble Forest/Best Model Following Tuning

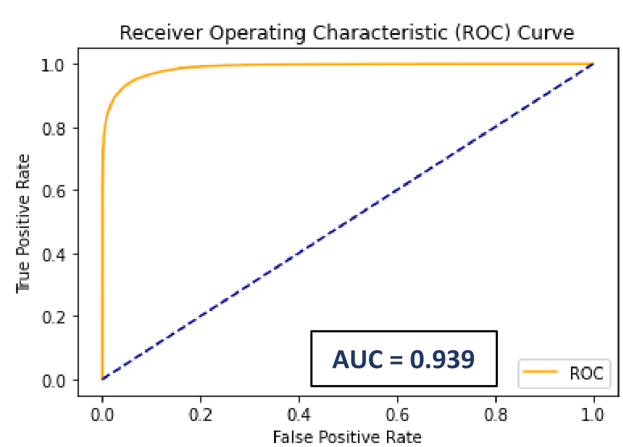
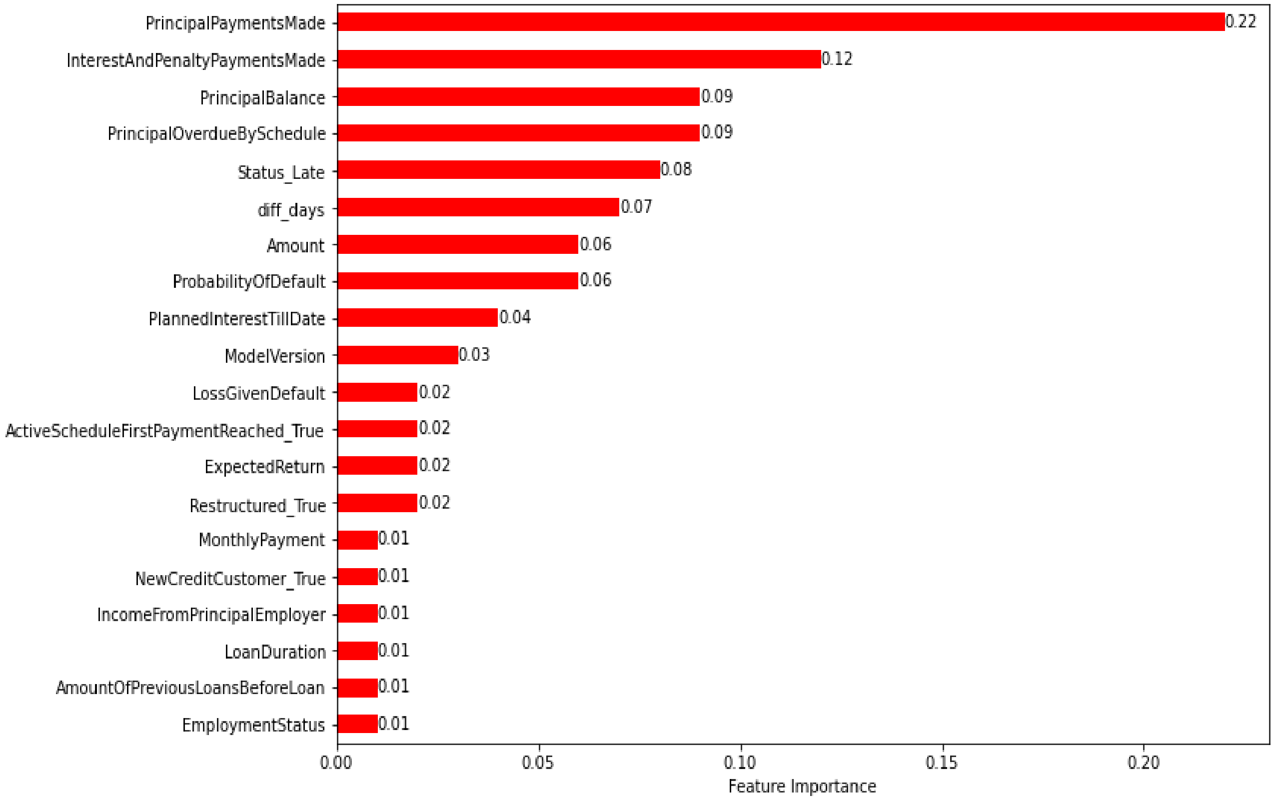


Exhibit 40: Features Importance Ensemble Forest/Best Model Following Tuning



5.4.2 Best Model Parameters

Based on the results of the tuning, the highest mean CV score of 0.947 (Exhibit 37) was obtained with the best values of hyperparameters noted on Exhibit 36. The best model was evaluated on the test dataset using these best model parameters. The results from this evaluation indicate that precision, recall, accuracy, F_1 score were marginally lower than the stronger and unboosted Decision Tree Classifier, but were all higher than 0.9 (Exhibit 38). The area under the curve of the receiver operating characteristic curve was 0.939 (Exhibit 39), which indicates that the model is effective in separating the target class between 0 and 1.

Despite the fact that this model boosted a much weaker Decision Tree Classifier than that utilized in Section 5.3, model results were comparable. It is worth noting that the strength of the weak Decision Tree Classifier boosted by this algorithm is much lower on the lower end for some hyperparameters (mean CV score of less than 0.2) when compared to the best model with l_rate of 1.0 and number of estimators of 100.

The five features with the most importance to model prediction were PrincipalPaymentsMade, *InterestandPenaltyPayment*Made, *PrincipalBalance*, *PrincipalOver DueBy Schedule*, and *StatusLate* (see Exhibit 40). Exhibit 40 can be used for interpretation of the best “ada-boost” model and to identify the features that drove the classification prediction in this model. Note that this exhibit only shows those features that have a variable importance of greater than or equal to 0.001.

5.5 Random Forest

5.5.1 Model Overview and Results

Random forests or **random decision forests** is an ensemble learning method for classification that operates by constructing a multitude of decision trees at training time. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

sklearn's ensemble RandomForest Classifier module was used to model the classification on the final dataset (sklearn-e). Default max_depth was utilized, which allows the nodes to expand until all leaves are pure or until all leaves contain less than 2 samples required to split an internal node.

The modeling was conducted as follows:

class sklearn.ensemble.RandomForestClassifier(*n_estimators, criterion, max_features)*

The noted hyperparameters were tuned per Grid Search CV with 5-fold cross validation per Exhibit 41. Results are provided on Exhibits 42-45.

Exhibit 41: Random Forest Model Hyperparameters

Hyper--parameter	Range	Best Value
N_estimators	50,100,200	200
Criterion	Gini, entropy	entropy
Max_features	sqrt, log2	sqrt

Exhibit 43: Performance Evaluation: Random Forest

Confusion Matrix, Test Dataset Following Tuning:

	Predicted No	Predicted Yes
Actual No	26,854	333
Actual Yes	826	13,789

Parameter	Value Following Tuning
RMSE	0.163
Precision	0.976
Accuracy	0.972
Recall	0.943
F1_Score	0.960

Exhibit 42: Random Forest Grid Search CV Results

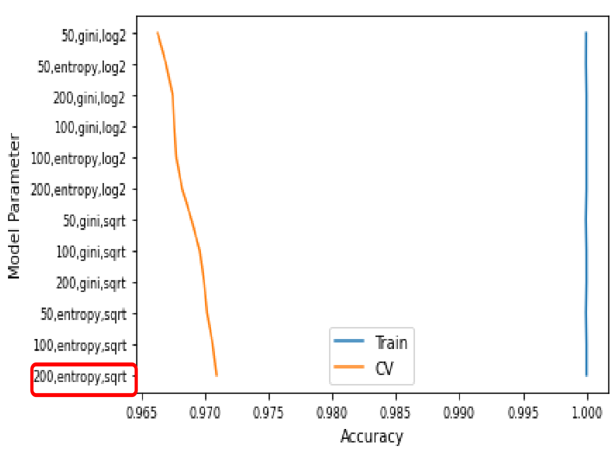


Exhibit 44: ROC Curve: Random Forest/Best Model Following Tuning

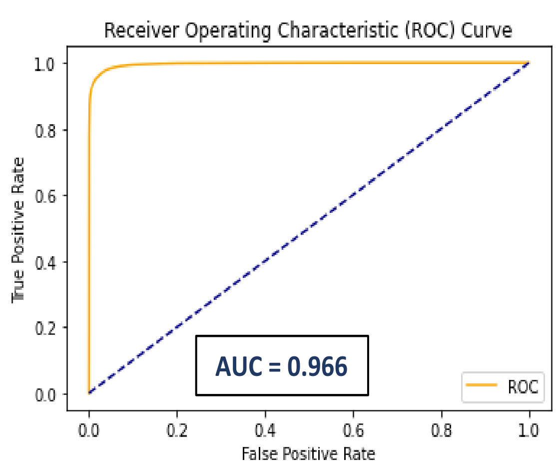
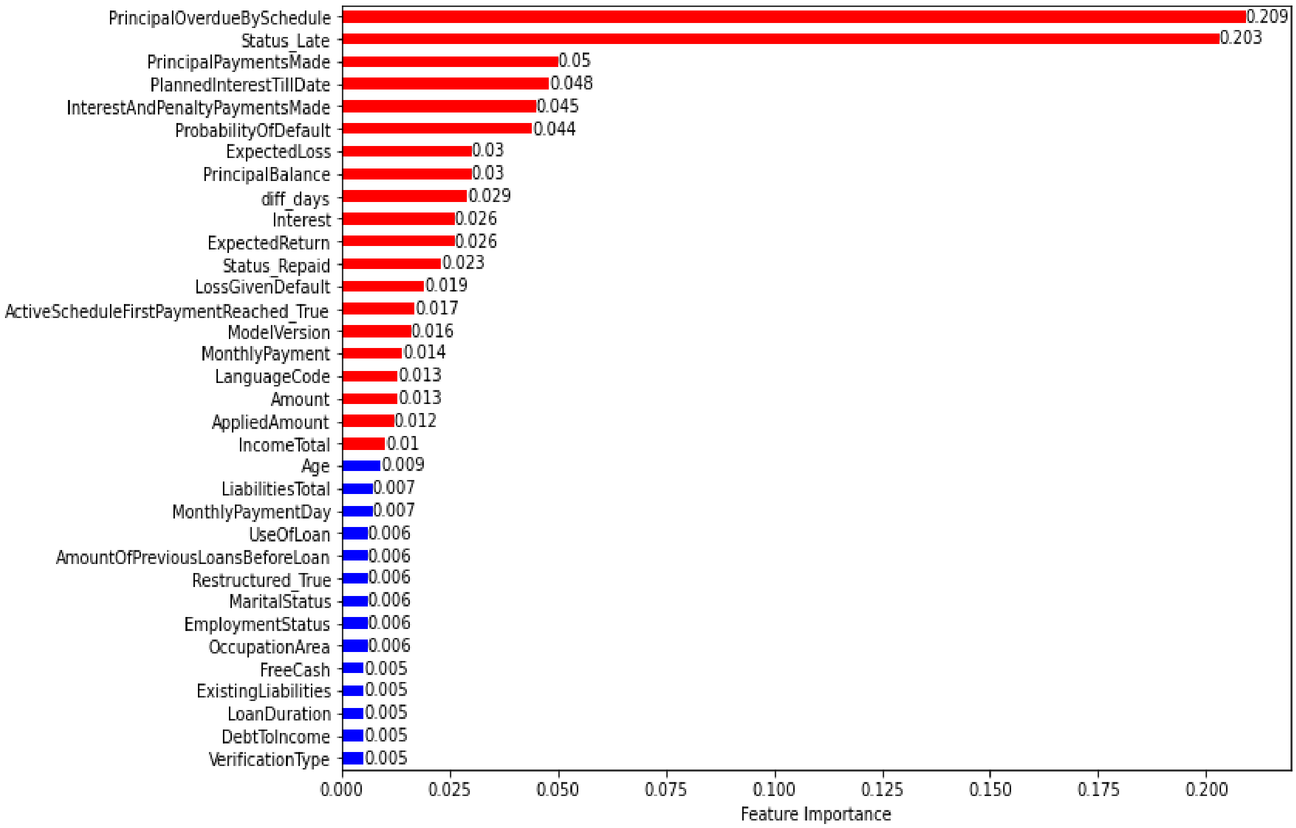


Exhibit 45: Important Features Importance Random Forest/Best Model Following Tuning



5.5.2 Best Model Parameters

Based on the results of the tuning, the highest mean CV score of 0.971 (Exhibit 42) was obtained with the best values of hyperparameters noted on Exhibit 41. The best model was evaluated on the test dataset using these best model parameters. The results from this evaluation indicate that precision, recall, accuracy, F_1 score were all higher than 0.9 (Exhibit 43). The area under the curve of the receiver operating characteristic curve was 0.966 (Exhibit 44), which indicates that the model is effective in separating the target class between 0 and 1.

The five features with the most importance to model prediction were *PrincipalOverduebySchedule*, *StatusLate*, PrincipalPaymentsMade, *PlannedInterestsTillDate*, and *InterestandPenaltyPaymentsMade* (see Exhibit 45). Exhibit 45 can be used for interpretation of the best "random forest" model and to identify the features that drove the classification prediction in this model. Note that this exhibit only shows those features that have a variable importance of greater than or equal to 0.005.

5.6 Bayesian Optimization

5.6.1 Overview

Bayesian optimization is an elegant solution for optimization of the objective functions (such as accuracy, RMSE, etc.) required for hyperparameter tuning of machine learning models (Jones, 2001). It is being used in academia to solve problems in numerous areas (Shahriari, 2016).

For our study, this approach of optimization was evaluated on three of the models, decision tree, ensemble forest, and random forest, for comparison with the grid search optimization outlined in sections 5.3, 5.4, and 5.5.

Bayesian optimization algorithm utilizes gaussian process and maintains a posterior distribution for this function as observations are made (Snoek, 2012). A range of hyperparameter values are optimized in this process and each subsequent value is picked to allow for optimization of the current best result of the objective function (see Exhibit 46 for the algorithm).

1. For n = 1,2,…. do
2. Select new x_{n+1} by optimizing objective function α (in this case 1 – Accuracy)

   x_{n+1} = argmin α (x, Dn) where Dn is surrogate function, (in this case a Gaussian Process)
3. Query the objective function to obtain y_{n+1}
4. augment data Dn+1 = {Dn, (x_{n+1} , y_{n+1})}
5. update statistical model
6. end for

Exhibit 46: Bayesian Optimization, Algorithm

The three main ingredients of the Bayesian optimization process used in our study are the following:

1) **Search space of Hyperparameters**: A range and type of values of the hyperparameters for the three models that were optimized using this algorithm are presented in sections 5.6.2, 5.6.3, and 5.6.4. The prior distributions for the hyperparameters assigned in our study are uniform and log-uniform for integer and real parameter values and a list of values for categorical hyperparameter values.
2) **Objective function**: The objective function is the main evaluator used in the optimization. For our study, the set of hyperparameters are optimized to obtain the lowest mean 5-fold cross validation scores of 1-accuracy score for each iteration.
3) **Surrogate function** and **Acquisition function**: sklearn's gaussian process was utilized as the surrogate function. The surrogate function can be interpreted as an approximation of the objective function. It is used to propose parameter sets to the objective function that likely yield an improvement in terms of accuracy score.
   The hyperparameters that are put forward for evaluation by the objective function are selected by applying a criterion to the surrogate function. This criterion is defined by a acquisition function.
   Sklearn's gp_minimize function is used with the default gp_hedge parameter for the acquisition function. Gp_hedge probabilistically chooses one of the above three acquisition functions at every iteration. These are:

   - "LCB" for lower confidence bound;
   - "EI" for negative expected improvement; and
   - "PI" for negative probability of improvement.

Reference to a github repository for explanation of the acquisition functions is provided as reference (skopt-a). Default values for the rest of the skopt's gp_minimize were retained for this study.

5.6.2 Decision Tree

Bayesian optimization algorithm using the process outlined above was applied to the Decision Tree classifier. Default values were retained for this model, except for the hyperparameters that were tuned. The search spaces for the hyperparameters that were tuned are presented in Table 7 below.

Three hyperparameters were tuned in this study. Criterion had a categorical search space while the other two had integer search spaces. Results (cross validated mean(1-accuracy score)) and the selected hyperparameter values are provided on Exhibit 47.

Table 7: Search Space Decision Tree

Search Space	
Criterion:	[Gini, Entropy]
Min_samples_split:	[2,5]
Max_Depth:	[5,20]

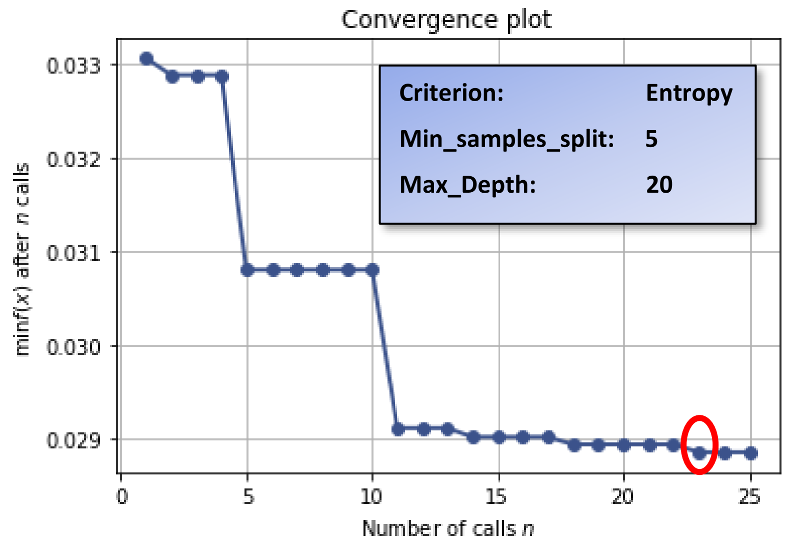


Exhibit 47: Bayesian Optimization Convergence, Decision Tree

5.6.3 Ensemble Forest

Bayesian optimization algorithm using the process outlined above was applied to the adaboost's ensemble forest classifier. Similar to the process used in Section 5.4, a weak decision tree classifier was boosted by the adaboost model.

The search spaces for the hyperparameters that were tuned are presented in Table 8 below.

Two hyperparameters were tuned in this study. N_estimators had an integer search space while the l_rate had real value search space. Results (cross validated mean (1-accuracy score)) and the selected hyperparameter values are provided on Exhibit 48.

Table 8: Search Space Ensemble Forest

Search Space	
N_estimators:	[5,200]
l_rate:	[0.1, 5]

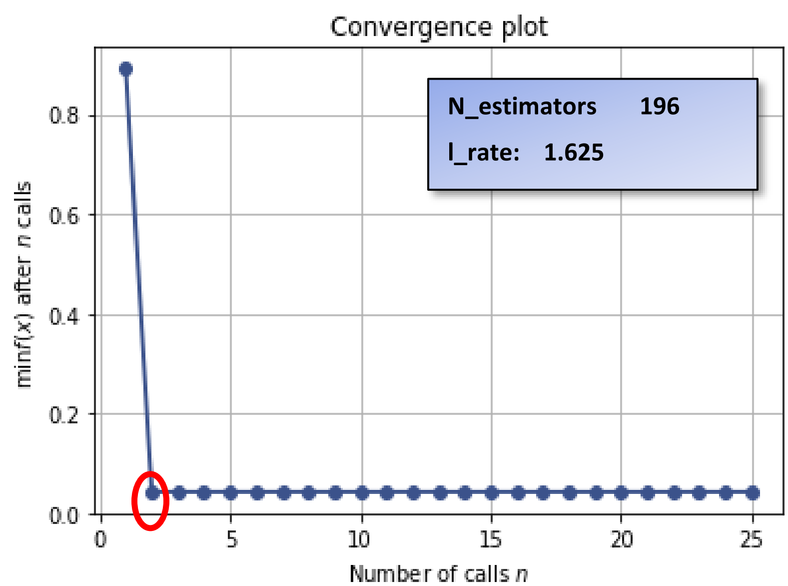


Exhibit 48: Bayesian Optimization Convergence, Ensemble Forest

5.6.4 Random Forest

Bayesian optimization algorithm using the process outlined above was applied to the random forest classifier. Aside from the hyperparameters outlined below, the remainder of the parameters used in the model were identical to those used in the modeling conducted in Section 5.5.

The search spaces for the hyperparameters that were tuned are presented in Table 9 below.

Three hyperparameters were tuned in this study. Criterion and max_features had categorical search spaces while n_estimators had an integer search space. Results (cross validated mean (1-accuracy score)) and the selected hyperparameter values are provided on Exhibit 49.

Table 9: Search Space Random Forest

Search Space	
Criterion:	**[Gini, Entropy]**
Max_Features:	**[sqrt, log2]**
N_estimators:	**[50,300]**

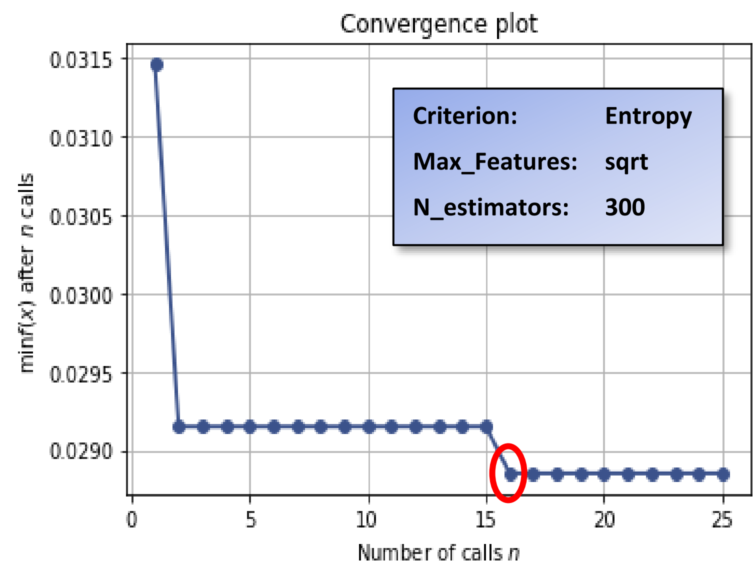


Exhibit 49: Bayesian Optimization Convergence, Random Forest

5.6.5 Model Results

Each of the tuned classifier models were fitted with the selected hyperparameters on the training portion of the final dataset. Following this fitting, the fitted models were evaluated on the test dataset and the results are provided on Exhibit 50.

For the decision tree and the random forest classifier, the results were similar to those obtained from the grid search optimization. Selected hyperparameters were identical for decision tree and for random forest, the only difference was in the no_of trees of 300 relative to the 200 in the grid search optimization.

Adaboost showed an improvement for all of the metrics relative to grid search optimization.

Exhibit 50:

Bayesian Optimization, Summary of Results

Metric	Decision Tree BO	AdaBoost BO	Random Forest BO
RMSE	0.166	0.199	0.164
Precision	0.962	0.950	0.976
Accuracy	0.973	0.960	0.972
Recall	0.960	0.936	0.944
F_1 Score	0.961	0.943	0.960
AUC	0.970	0.954	0.966

Note: Final results identical for Decision Tree and also for random forest within tolerance bounds relative to grid search optimization. Adaboost scores showed improvement.

5.7 Deep Neural Network with Tensorflow/Keras

5.7.1 Model Overview and Results

Deep neural network model was developed using Tensorflow/Keras to train, validate, and test the final dataset. The architecture for the neural network was as follows:

1) Input layer with 71 neurons corresponding to 71 predictor variables.
2) 3 Hidden layers: Layer 1 with 100 neurons; Layer 2 with 50 neurons, and Layer 3 with 25 neurons. Each accepts the sum of the products of linear input of weights and input values and the output activation of each layer is set to be RELU.
3) 1 output layer with 1 neuron with a sigmoid activation.

The neural network was first trained on the entire final dataset, with a 80% train and 20% test split. Training was conducted using default parameters noted on Exhibit 51.

Following this initial preliminary run, the Tensorflow/Keras model was subjected to 3-Fold cross validation. skLearn's GridSearch CV was utilized to perform hyperparameter tuning during this phase. Exhibit 52 identifies the various hyperparameters chosen during this study and the results of the analyses. Note that because of the significant time complexity of this phase of the modeling, only a 10% fraction of the final dataset was used for training, validation, and testing. This fraction was then split into 80% train (and validation) and test components. The noted hyperparameters were tuned per Grid Search CV with 5-fold cross validation per Exhibit 52. Results are provided on Exhibits 53-56.

Exhibits 57 and 58, show AUC for the receiver operating characteristic curves, for the default and the best "tuned" model, respectively.

Exhibit 51: Performance Evaluation: Keras/Tensorflow, Default Parameters

Confusion Matrix, Test Dataset:

	Predicted No Default	**Predicted Yes Default**
Actual No Default	26,101	1,086
Actual Yes Default	1,768	12,847

Parameter	**Value**
RMSE	0.261
Precision	0.922
Accuracy	0.931
Recall	0.879
F1_Score	0.900

Exhibit 52: Keras/Tensorflow Model Hyperparameters

Hyper- -parameter	**Range**	**Best Value**
Optimizer	rmsprop, adam	adam
Inits	glorot_uniform, normal, unform	glorot_uniform
Epochs	50,100,150	150
Batches	5,20	5
Default: Only Change: Inits: random_normal; No Batch; Early Stopping Allowed		

Exhibit 53: Keras/Tensorflow Training Errors, Best Tuned Model Retraining

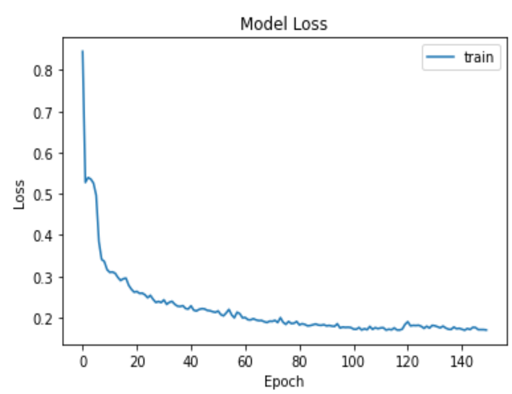


Exhibit 54: Keras/Tensorflow Training Accuracy, Best Model Retraining

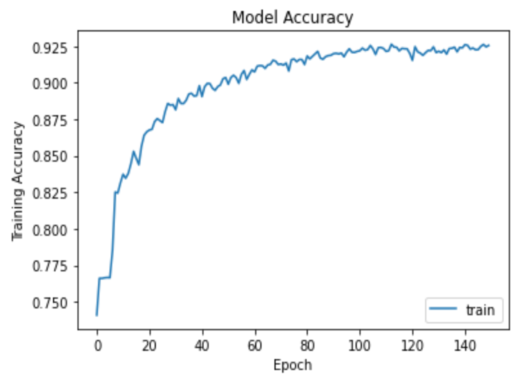


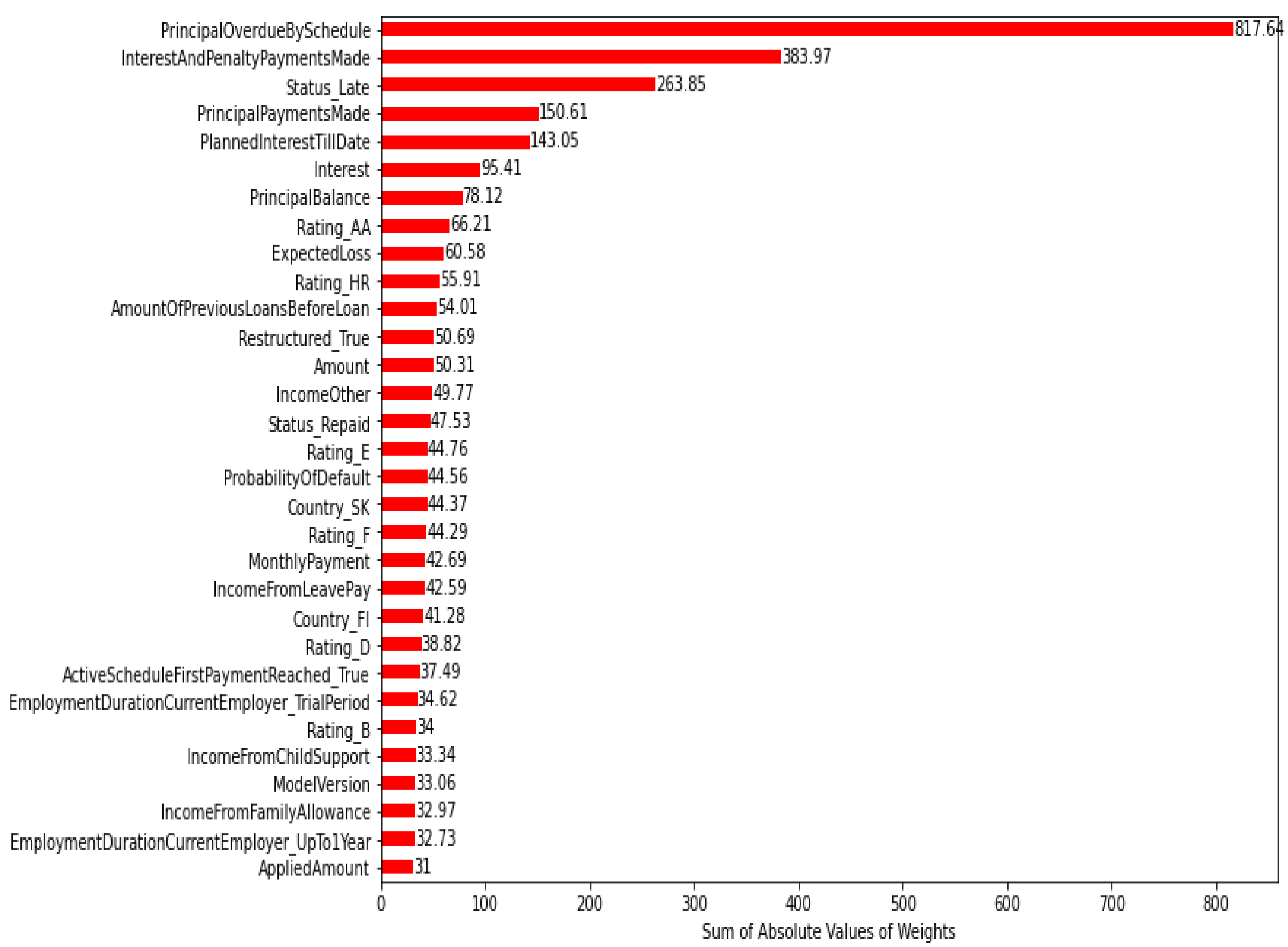
Exhibit 55: Performance Evaluation: Keras, Best Model Following Tuning

Confusion Matrix, Test Dataset Following Tuning (10% of Dataset):

	Predicted No	**Predicted Yes**
Actual No	630	308
Actual Yes	44	3,018

Parameter	**Value**
RMSE	0.249
Precision	0.907
Accuracy	0.912
Recall	0.986
F1_Score	0.945

Exhibit 56: Important Features Weights Neural Net/Best Model Following Tuning



5.7.2 Best Model Parameters

Based on the results of the tuning, best hyperparameters were selected (see Exhibit 52). The best model was evaluated on the test dataset using these best model parameters. The results from this evaluation indicate that precision, recall, accuracy, and F_1 score were all higher than 0.9 (Exhibit 55). The area under the curve of the receiver operating characteristic curve was 0.980 (Exhibit 58), which is the highest of all the models evaluated during this study.

Note that the top (10 pct of dataset) rows from the final dataset were chosen for the training and testing. The distribution of the target class within this segment of the dataset was different from the overall distribution. Despite this, the AUC for the ROC curve was the highest for this model and its performance relative to other performance metrics were similar to the best "tree" models – decision tree and random forest.

It is worth noting that the performance of the neural network on the entire dataset using the default model was also reasonable. The AUC for the ROC curve on the test dataset for this model was also 0.98 (Exhibit 57). The precision, accuracy, F_1 score were greater than or equal to 0.9, and recall was marginally below 0.9. With hyperparameter tuning, it is conceivable that the results of the modeling on the entire dataset will likely be similar to those obtained from the 10% of the final dataset.

Features that had the highest final weights (for values greater than or equal to 30) assigned to them on the best tuned model is presented in descending order of weights on Exhibit 56. The five features with the highest weights were *PrincipalOverduebySchedule*, *InterestandPenaltyPaymentsMade, StatusLate*, PrincipalPaymentsMade, and *PlannedInterestTillDate* (see Exhibit 54)

Exhibit 57: ROC Curve:

TensorFlow/Keras Default

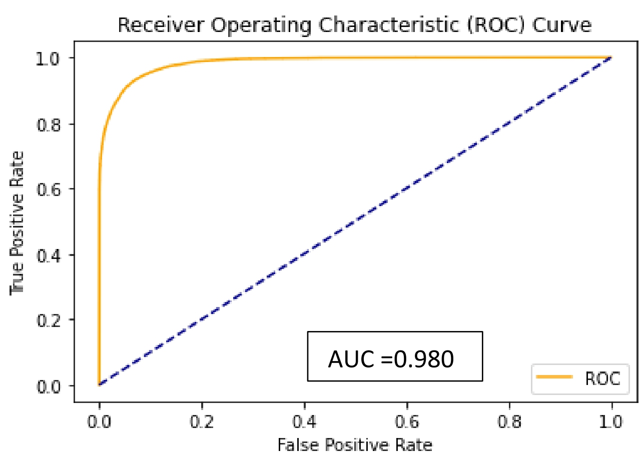
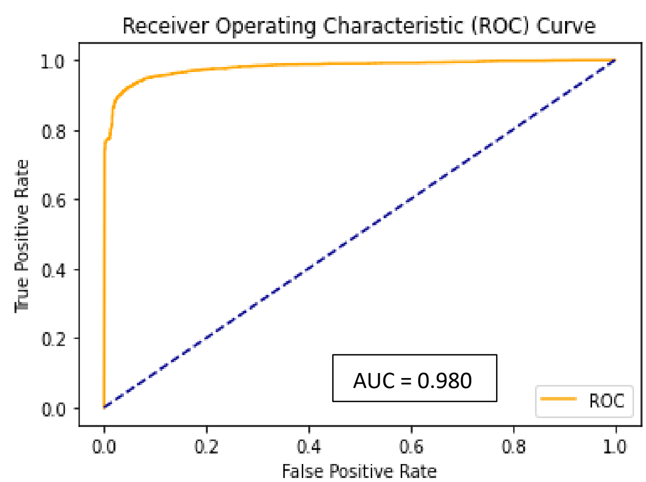


Exhibit 58: ROC Curve: Tensor Flow/Keras/Best Model Following Tuning



5.8 Federated Machine Learning with PyTorch and PySft

5.8.1 What is Federated Machine Learning and Why is it Relevant?

The traditional AI algorithms require centralizing data on a single machine or a server. The limitation of this approach is that all the data collected is sent back to the central server for processing before sending it back to the devices.

Federated Learning is a centralized server first approach. It is a distributed ML approach where multiple users collaboratively train a model. The concept of federated learning was first introduced in Google AI's 2017 blog. Here, remote raw data is distributed without being moved to a single server or data center. The central server selects a few remote nodes and sends the initialized version containing model parameters of an ML model to all the remote nodes. Each remote node now executes the model, trains the model on their local data, and has a local version of the model at each node. Once trained the models are then sent to the centralized server for aggregation and model evaluation.

Federated Learning (FL) leverages techniques from multiple research areas such as distributed systems, machine learning, and privacy. FL is best applied in situations where the on-device data is more relevant than the data that exists on servers. FL provides edge devices with state of the art ML without centralizing the data and privacy by default. Thus, it handles the unbalanced and non-Independent and Identically Distributed (IID) data of the features in mobile devices. A lot of data is generated from smartphones that can be used locally at the edge with on-device inference. Since the server does not need to be in the loop for every interaction with the locally generated data, this enables fast working with battery saving and better data privacy.

For this study, Facebook's PyTorch with a PySyft wrapper was utilized to perform a "test" run for the execution of federated ML. Process and connection layouts are depicted on Exhibits 59 and 60, respectively.

Exhibit 59: Federated ML Process Layout

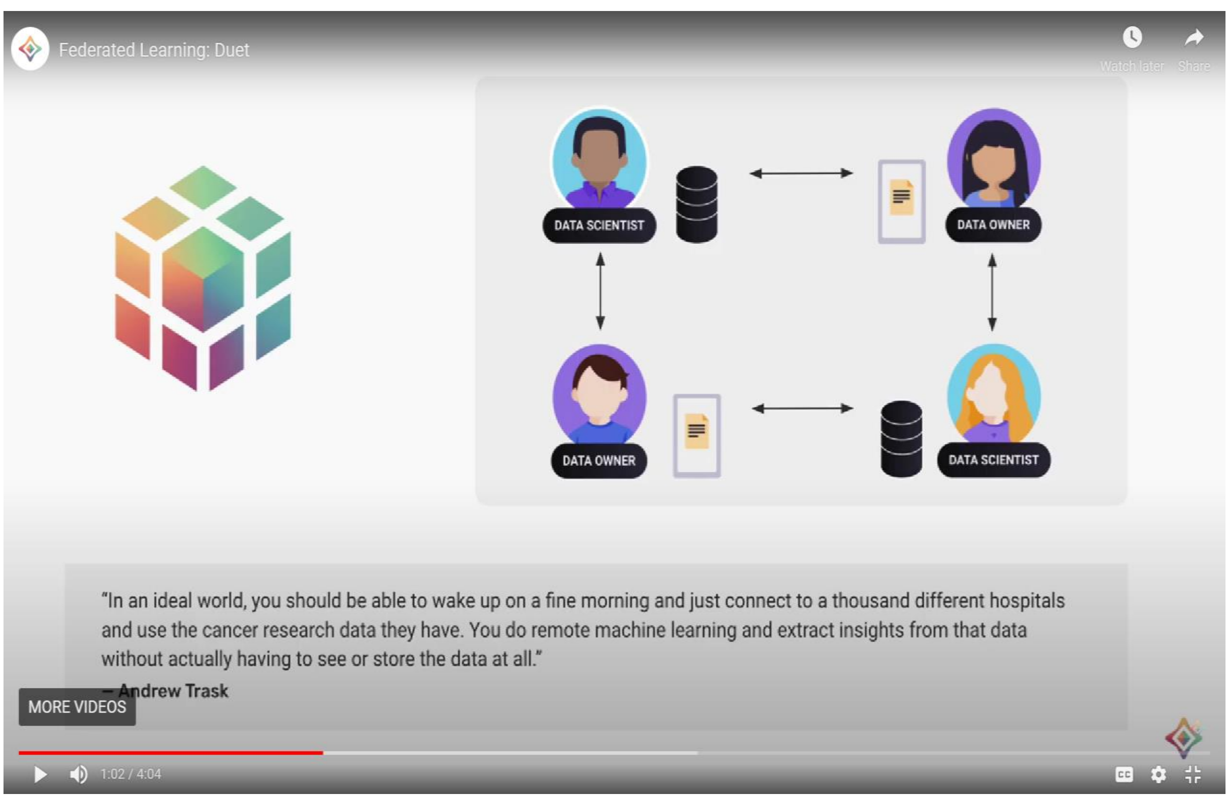
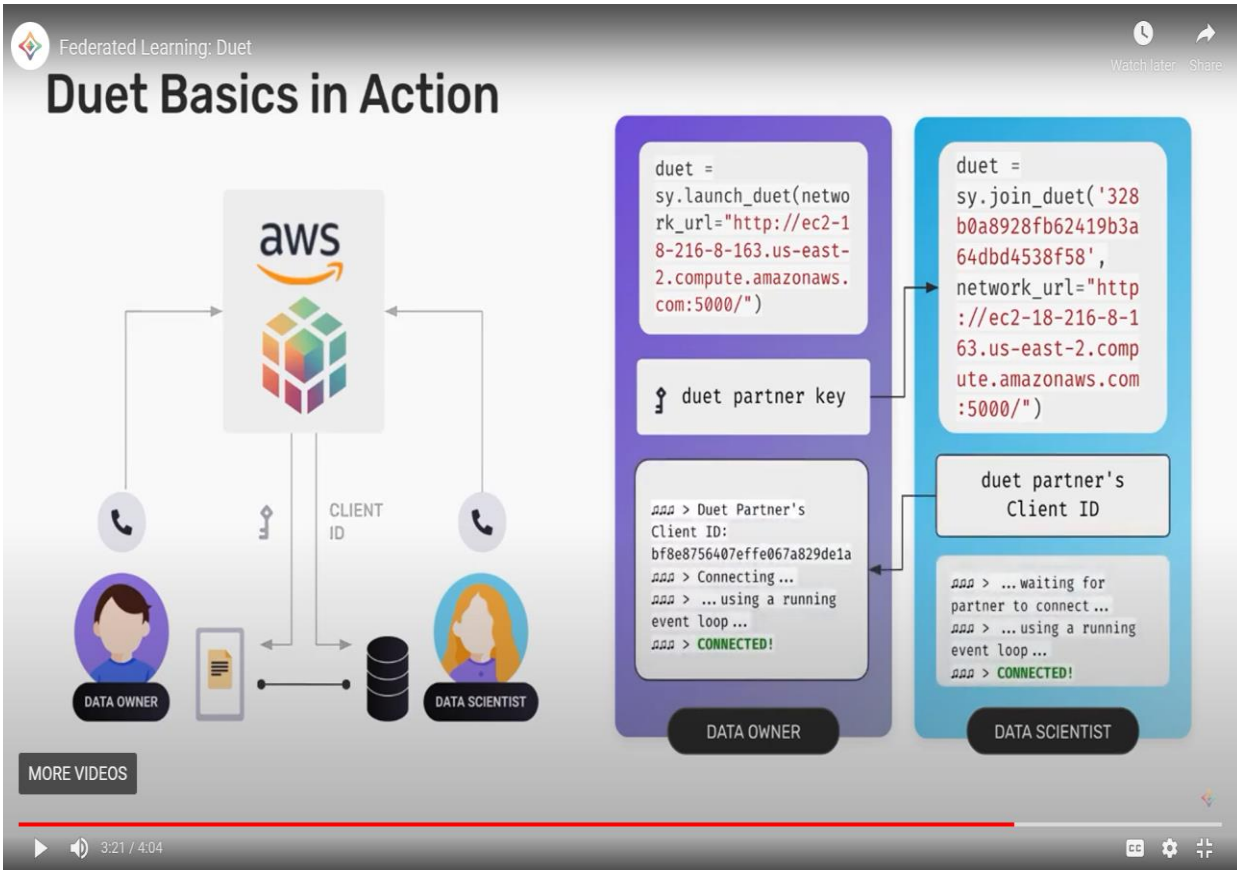


Exhibit 60: Federated ML Connection Layout



5.8.2 Modeling Steps

The steps used for the remote federated ML in this study are provided below. 2 Jupyter notebooks were developed; one for the data owner and a second one for the data scientist to simulate the federated ML.

The focus of PyTorch and PySft modeling effort was to identify the process to be used to train, build, and test the model on remote dataset and to evaluate its effectiveness in achieving results that are comparable to the other models. Accordingly, to reduce the time required to run the models, 5% of the final dataset was used in the modeling effort. Similar to the workflow for the other models, this fraction of the final dataset was split into train (80%) and test (20%) components.

The steps followed were as follows:

1) Data Owner/Data Scientist interacted via PySyft and PyGrid/Amazon Web Service (see Exhibit 59)
2) Data Owner sent data to Data Scientist upon request from Data Scientist
3) Data Scientist made requests via Pysft to Data Owner
4) Data Scientist created the neural network model architecture
5) Data Scientist sent the model to Owner
6) Training occurred on the Remote Server

7) Model Sent to Data Scientist Once Trained

8) Data Scientist Tested the Model using test set data – Scikit Learn Packages

5.8.3 Model Architecture

The neural network model architecture and model parameters were as follows:

1) 3 Hidden Layers: 100, 50, and 25 Neurons, RELU Activation

2) 1 Output Layer, 2 Neurons, Log_soft_max Activation

3) 300 Epochs

4) Optimizer: Adam

5) learning_rate = .01

6) nn.functional.nll_loss

5.8.4 Model Results

Results of the modeling are depicted on Exhibits 61 to 63. Model results indicated that the precision, accuracy, recall, and F_1 scores all exceeded 0.85, and the AUC score was 0.966. The model results indicate the viability of this application for the classification on the loan dataset. Further fine tuning and optimization and testing on the full final dataset should yield results comparable to the best performing models in this study.

Exhibit 61: Performance Model, PyTorch and PySft

	Predicted No	Predicted Yes
Actual No	1,262	99
Actual Yes	97	632

Parameter	**Value**
RMSE	0.306
Precision	0.865
Accuracy	0.906
Recall	0.867
F1_Score	0.867

Exhibit 62: Federated ML Training Errors

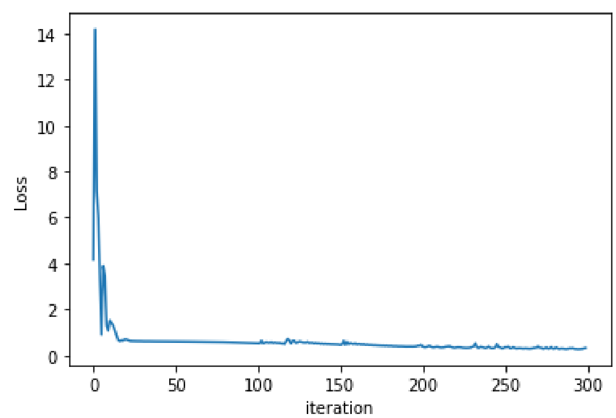
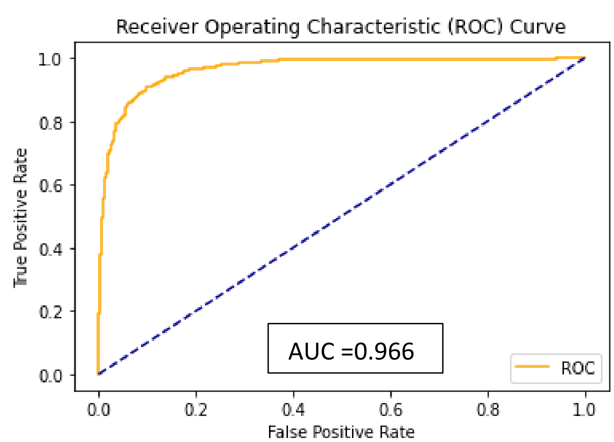


Exhibit 63: Federated ML ROC Curve



5.9 Summary of Model Evaluations

5.9.1 Summaries of Models with Full Feature Set

A comparison of the performance of the models presented in this study relative to the various performance metrics is presented in Exhibit 64 below. For the decision tree and random forest, scores from the grid search optimization are presented, but for the ensemble forest, scores from the Bayesian optimization are presented.

1) Accuracy and F_1 scores were highest for the decision tree model.
2) Recall was the highest for Tensorflow/Keras neural network model.
3) RMSE was the lowest for the Random forest model.
4) Precision was the highest for the Random forest model.
5) Better tuning of the Random forest model, which has a high time complexity, should allow it to outperform the Decision Tree model.
6) AUC was the highest for Tensorflow/Keras neural network model.
7) Ensemble forest which boosted a weak decision tree classifier performed only marginally poorer than the stronger Decision Tree Classifier.
8) Remote ML with PyTorch/PySft provided results that were comparable to other models.

Exhibit 64: Overall Models Performance Evaluation

Parameter	RMSE	Precision /Recall	Accuracy/ F_1 Score	AUC
Logistic Regression	0.209	0.938/0.936	0.956/0.937	0.951
Multinomial Bayes	0.399	0.789/0.743	0.841/0.765	0.818
Decision Tree	0.166	0.962/0.960	**0.973**/**0.961**	0.970
Ensemble Forest	0.199	0.950/0.936	0.960/0.943	0.954
Random Forest	**0.163**	**0.976**/0.943	0.972/0.960	0.966
Tensor Flow/Keras NN	0.249	0.907/**0.986**	0.912/0.945	**0.980**
PyTorch/PySft	0.306	0.865/0.867	0.906/0.867	0.966

5.9.2 Model Prediction Power, Limited Feature Set

In order to evaluate whether models that contain a smaller subset of input features can retain the prediction power of the "full" feature set models evaluated in this study, a focused evaluation using the random forest classifier model tuned with the hyperparameters selected from the Bayesian optimization process on a smaller subset of input features was conducted.

The perturbation approach utilized and the results from this evaluation are presented in sections 5.9.2.1 and 5.9.2.2, respectively.

5.9.2.1 Perturbation Approach

As part of this approach, the feature importance scores (see Exhibit 45) for all 71 input features from the random forest model (i.e., the baseline model) were sorted in descending order. Five separate modeling runs were conducted using the following subset of features:

1. Top 20 features from the baseline model (see Exhibit 65);
2. Top 10 features from the baseline model (see Exhibit 66);
3. Top 6 features from the baseline model (see Exhibit 67);
4. Top 4 features from the baseline model (see Exhibit 68); and
5. Top 2 features from the baseline model (see Exhibit 69).

Note that for several of the baseline models (i.e., logistic regression, decision tree, ensemble forest, and random forest classifiers), aside from some marginal differences, the top 20 features were similar, with 70 to 75 % of the features being identical across the classifiers (see Exhibit 71).

The random forest model trained with 5-fold cross validation mean accuracy scores with Bayesian optimization were fitted to these subsets of features in the train portion of the dataset and the fitted model was tested on the test portion of the dataset.

5.9.2.2 Results

Model evaluations on the test dataset for the various performance metrics and the feature importance scores for the smaller subset models are presented on Exhibits 65-69. The performance metric scores were compared with the baseline model that contained all 71 predictor features. Where model performance was better than the baseline model, the result is highlighted in blue and where it is poorer than the baseline model, it is highlighted in red.

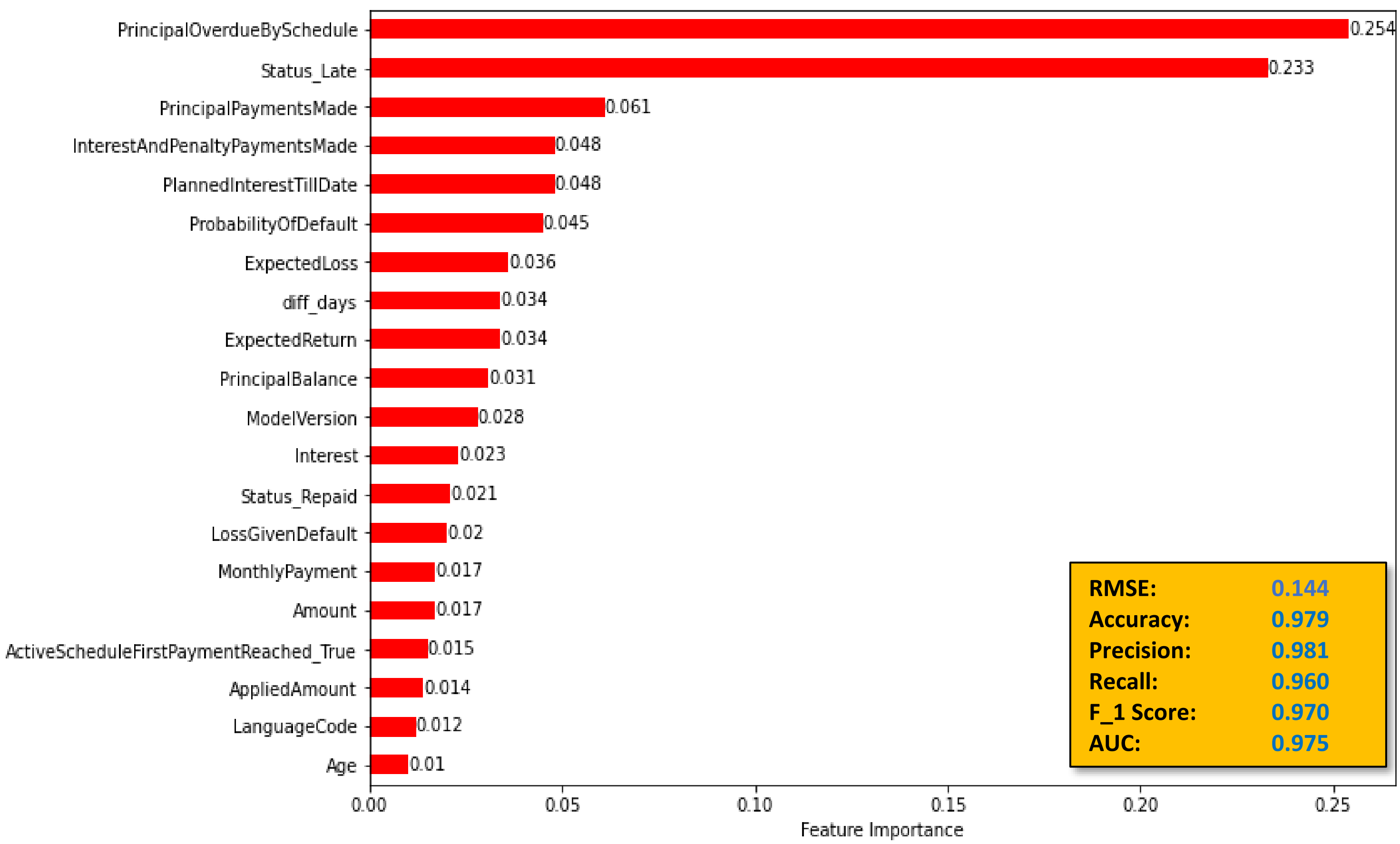


Exhibit 65: 20 Feature Set Mix/Performance Evaluation

For the 20 and the 10 input feature set models, model performance on the test dataset exhibited some improvement over the baseline model (see Exhibits 65 and 66). For the 6, 4, and 2 input feature set models, model performance declined relative to the baseline model (see Exhibits 67-69).

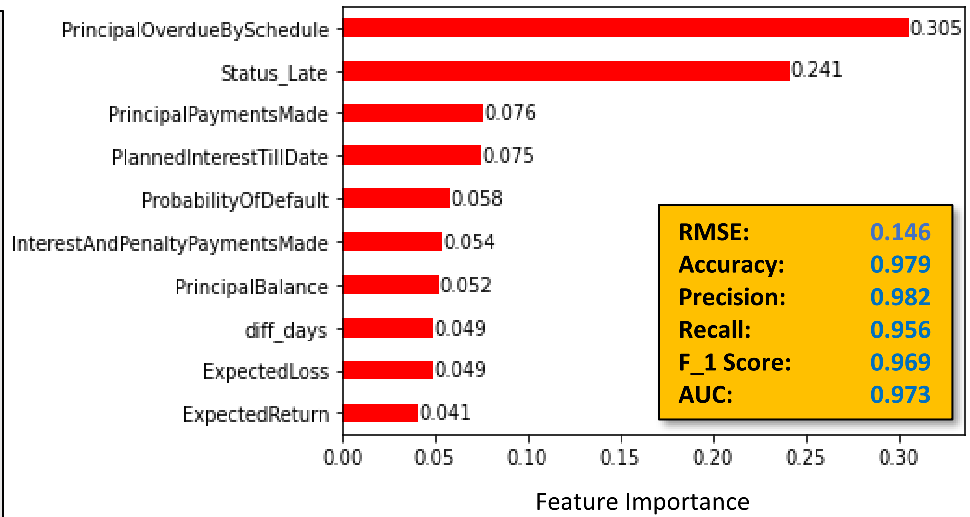


Exhibit 66: 10 Feature Set Mix/Performance Evaluation

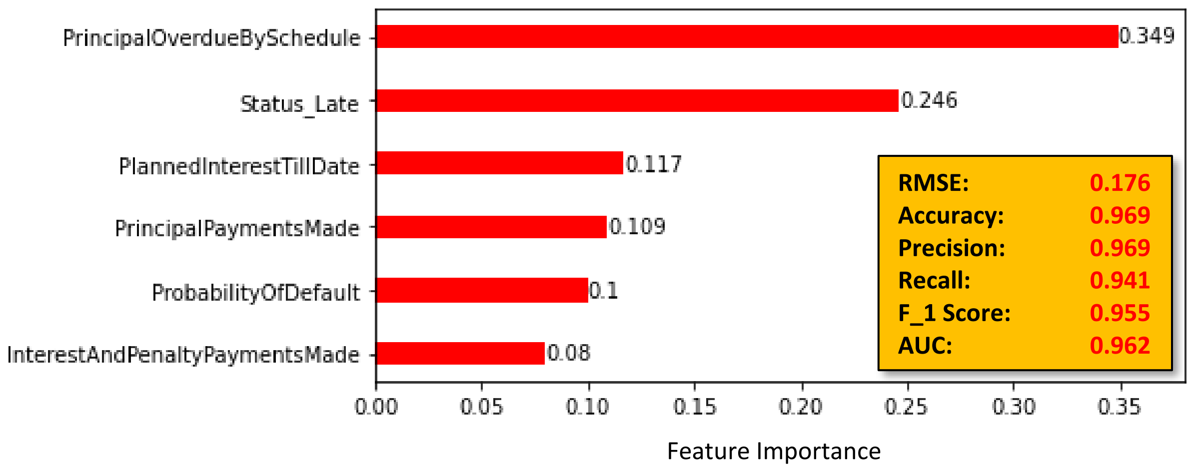


Exhibit 67: 6 Feature Set Mix/Performance Evaluation

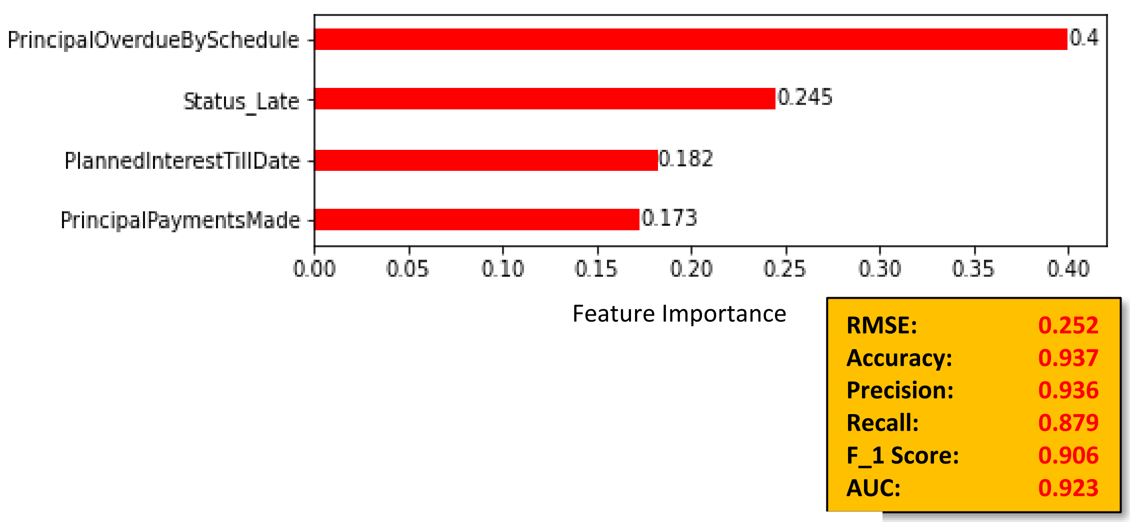
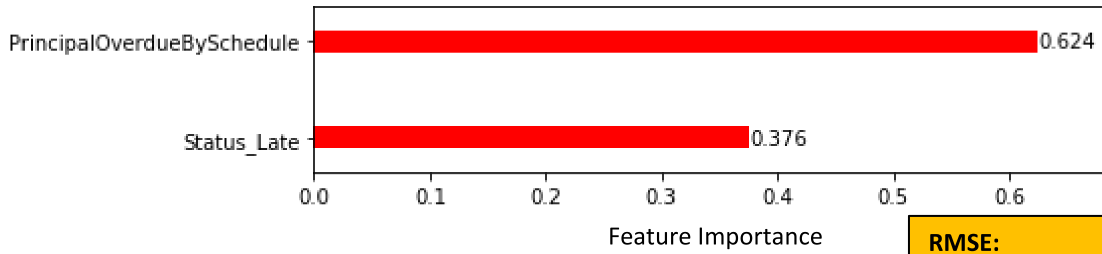


Exhibit 68: 4 Feature Set Mix/Performance Evaluation



RMSE:	**0.296**
Accuracy:	**0.912**
Precision:	**0.921**
Recall:	**0.819**
F_1 Score:	**0.867**
AUC:	**0.891**

Exhibit 69: 2 Feature Set Mix/Performance Evaluation

5.9.2.3 Summary of Evaluations

Exhibit 70 shows model performance vs no. of predictor features. Models with 10 or more features perform better than the baseline model and those below 10 features perform poorer than the baseline model.

The percentage of common features between classifiers such as logistic regression, decision tree, and ensemble forest classifiers included in this study and the random forest classifier decline as the total number of features drop below 20, with only 60% of features common for top 10 input features for logistic regression and decision tree classifiers (see Exhibit 71).

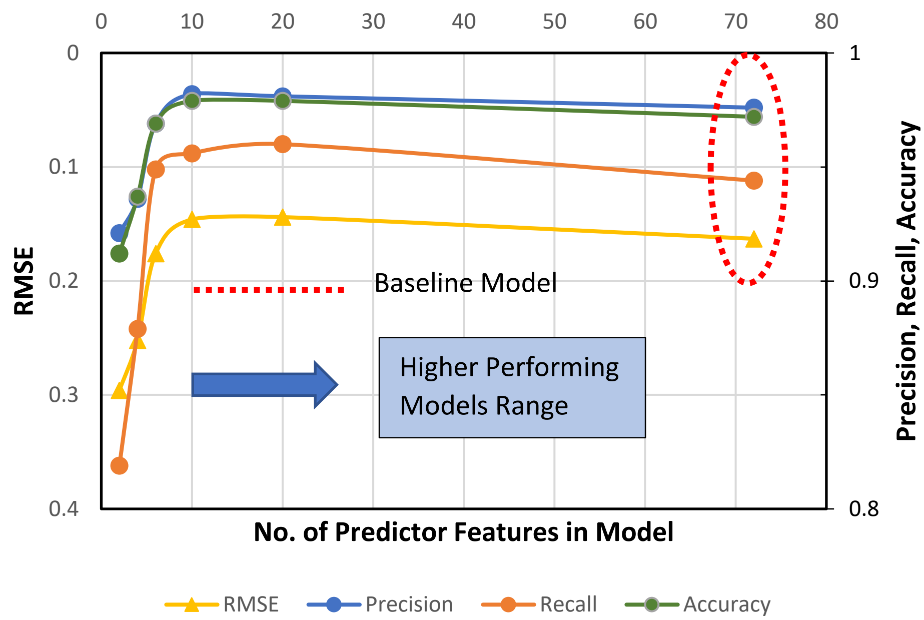


Exhibit 70: Model Performance Metrics vs No. of Predictor Features

ID	Similarity Measure Relative to Random Forest	Logistic Regression	Decision Tree	Ensemble Forest
Top 20 Features	No. of Features	6 out of 10	6 out of 10	8 out of 10
	% of Total	60%	60%	80%
Top 10 Features	No. of Features	15 out of 20	14 out of 20	15 out of 20
	% of Total	75%	70%	75%

Exhibit 71: Model Performance Metrics vs No. of Predictor Features

Care should be taken to ensure a balanced tradeoff between model simplicity and better interpretability (smaller number of input features) and robustness in model predictability and consistency in feature selection/importance (larger number of input features) while selecting the optimum subset of features to be utilized for classification.

A ROC curve for the 10 feature set model that shows marginal improvement over the baseline model is depicted on Exhibit 72. The predicted AUC of 0.973 is higher than the AUC of 0.966 for the baseline model.

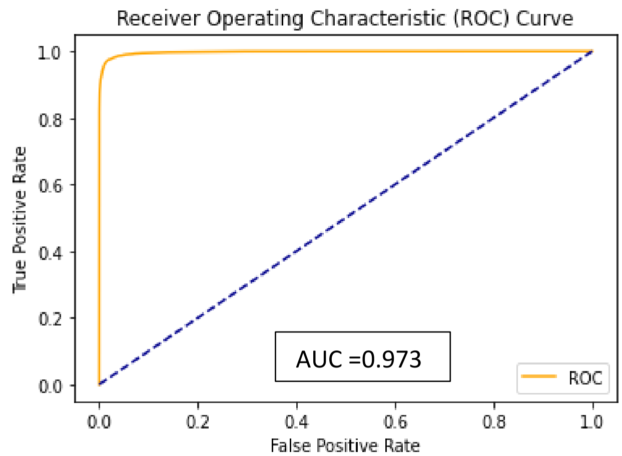


Exhibit 72: ROC Curve, 10 Feature Set Model

6.0 Conclusions

All the machine learning models, except Naïve Bayes provided consistent results. Precision, accuracy, recall, F1_scores were all above 0.85, and above 0.9 for all models, except remote ML performed by PyTorch/PySft.

Models with reduced feature sets (10 and 20 features) performed favorably relative to the full (71) feature set models. However, models with less than 10 features did not perform as well as the full feature set models.

If PyTorch/PySft model has a better architecture and undergoes tuning it should result in results comparable to the other models. Remote ML performed by PyTorch/PySft, which was only performed on a small fraction of the dataset (5 pct of the total) and was not tuned for hyperparameters still showed results that were comparable to other models. Remote ML models, when performed by PyTorch/PySft, can be trained remotely on multiple distributed systems and results can be aggregated and tested on the central server.

7.0 References

Bandora dataset:	Loan Dataset file from https://www.bondora.com/en/public-reports
Heaton, J., 2022	Applications of Deep Neural Networks with Keras, Jeff Heaton, Spring 2022.0
Heaton, J, 2022a:	Refer to Section 2.2.2 Encoding Categorical Variables as dummies, Applications of Deep Neural Networks with Keras, Jeff Heaton, Fall 2022.0
James G, 2017:	Introduction to Statistical Learning in R
Hastie T, 2017:	The Elements of Statistical Learning
Jones D.R., 2001	A taxonomy of global optimization methods based on response surfaces. Journal of Global Optimization
Shariari B, 2016	Taking the human out of the Loop: A review of Bayesian Optimization
Snoek J, 2012	Practical Bayesian Optimization of Machine Learning Algorithms
sklearn-a:	sklearn.linear_model.LogisticRegression — scikit-learn 1.1.1 documentation
sklearn-b:	sklearn.naive_bayes.MultinomialNB — scikit-learn 1.1.1 documentation
sklearn-c:	sklearn.tree.DecisionTreeClassifier — scikit-learn 1.1.1 documentation
Zhu, H. 2009:	Zhu, H. Zou , S. Rosset, T. Hastie, "Multi-class AdaBoost", 2009.
sklearn-d:	sklearn.ensemble.AdaBoostClassifier — scikit-learn 1.1.1 documentation
sklearn-e:	sklearn.ensemble.RandomForestClassifier — scikit-learn 1.1.1 documentation
skopt-a:	https://scikit-optimize.github.io/stable/auto_examples/bayesian-optimization.html
Zhu, H. 2009:	Zhu, H. Zou , S. Rosset, T. Hastie, "Multi-class AdaBoost", 2009.

APPENDICES

Appendix A: List of Feature Names

Feature No	Feature Name
1	ReportAsOfEOD
2	LoanId
3	LoanNumber
4	ListedOnUTC
5	BiddingStartedOn
6	BidsPortfolioManager
7	BidsApi
8	BidsManual
9	PartyId
10	NewCreditCustomer
11	LoanApplicationStartedDate
12	LoanDate
13	ContractEndDate
14	FirstPaymentDate
15	MaturityDate_Original
16	MaturityDate_Last
17	ApplicationSignedHour
18	ApplicationSignedWeekday
19	VerificationType
20	LanguageCode
21	Age
22	DateOfBirth
23	Gender
24	Country
25	AppliedAmount
26	Amount
27	Interest
28	LoanDuration
29	MonthlyPayment
30	County
31	City
32	UseOfLoan
33	Education
34	MaritalStatus
35	NrOfDependants
36	EmploymentStatus
37	EmploymentDurationCurrentEmployer
38	EmploymentPosition
39	WorkExperience
40	OccupationArea
41	HomeOwnershipType
42	IncomeFromPrincipalEmployer
43	IncomeFromPension
44	IncomeFromFamilyAllowance
45	IncomeFromSocialWelfare
46	IncomeFromLeavePay

47	IncomeFromChildSupport
48	IncomeOther
49	IncomeTotal
50	ExistingLiabilities
51	LiabilitiesTotal
52	RefinanceLiabilities
53	DebtToIncome
54	FreeCash
55	MonthlyPaymentDay
56	ActiveScheduleFirstPaymentReached
57	PlannedPrincipalTillDate
58	PlannedInterestTillDate
59	LastPaymentOn
60	CurrentDebtDaysPrimary
61	DebtOccuredOn
62	CurrentDebtDaysSecondary
63	DebtOccuredOnForSecondary
64	ExpectedLoss
65	LossGivenDefault
66	ExpectedReturn
67	ProbabilityOfDefault
68	PrincipalOverdueBySchedule
69	PlannedPrincipalPostDefault
70	PlannedInterestPostDefault
71	EAD1
72	EAD2
73	PrincipalRecovery
74	InterestRecovery
75	RecoveryStage
76	StageActiveSince
77	ModelVersion
78	Rating
79	EL_V0
80	Rating_V0
81	EL_V1
82	Rating_V1
83	Rating_V2
84	Status
85	Restructured
86	ActiveLateCategory
87	WorseLateCategory
88	CreditScoreEsMicroL
89	CreditScoreEsEquifaxRisk
90	CreditScoreFiAsiakasTietoRiskGrade
91	CreditScoreEeMini
92	PrincipalPaymentsMade
93	InterestAndPenaltyPaymentsMade
94	PrincipalWriteOffs

95	InterestAndPenaltyWriteOffs
96	PrincipalBalance
97	InterestAndPenaltyBalance
98	NoOfPreviousLoansBeforeLoan
99	AmountOfPreviousLoansBeforeLoan
100	PreviousRepaymentsBeforeLoan
101	PreviousEarlyRepaymentsBefoleLoan
102	PreviousEarlyRepaymentsCountBeforeLoan
103	GracePeriodStart
104	GracePeriodEnd
105	NextPaymentDate
106	NextPaymentNr
107	NrOfScheduledPayments
108	ReScheduledOn
109	PrincipalDebtServicingCost
110	InterestAndPenaltyDebtServicingCost
111	ActiveLateLastPaymentCategory
112	Target Class: Defaulted

Appendix B: Python code as pdf

```
# -*- coding: utf-8 -*-
"""
Created on Sat Jun  4 17:58:28 2022

@author: ramra
"""

# PROCESSES LOAN DATASET
# CREATES CORRELATION MATRIX
# CREATES AN INITIAL FILE FOR LOADING
# BUT IT REQUIRED PROCESSING, SEE PROJECT1.PY

# This is Ramkishore Rao's DSA 5900 practicuum project

import pandas as pd
import numpy as np
import psycopg2
import csv
from random import seed
from csv import reader
import random
import matplotlib.pyplot as plt
from math import exp
from math import pi
from math import sqrt
from random import random
import seaborn as sns
import csv

seed(500)

# Reading Loan Dataset File

filename = 'LoanData.csv'

df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

# Print First 5 Rows of Dataframe

print(df.head())

# Now cleaning the dataframe
# Remove Unnecessary Columns

df.drop(columns= ['ReportAsOfEOD', 'LoanId', 'LoanNumber',
                  'BiddingStartedOn', 'BidsPortfolioManager', 'BidsApi',
                  'PartyId', 'ApplicationSignedHour', 'ApplicationSignedWeekday',
```

```
                  'County', 'City', 'EmploymentPosition', 'EL_V0', 'Rating_V0'],
inplace=True)

print(df.head())

df[['DefaultDate']] = df[['DefaultDate']].fillna(value=0)

df.loc[df['DefaultDate'] != 0, 'DefaultDate'] = 1

check_missing_df = df.isna()

# checks the dataframe to see of there are missing values or no

check_missing_df.to_csv("datamiss.csv")

number_missing = df.isnull().sum()

# this tells us number missing in each column

number_missing.to_csv("datamiss1.csv")

result = df.isna().mean()

result.to_csv("missingresult.csv")

print(result)

df_consol = df.loc[: , result < .1]

# dropping additional unneeded columns

df_consol.drop(['BidsManual', 'ListedOnUTC', 'LoanApplicationStartedDate',
'MaturityDate_Original'], axis=1, inplace = True  )

# now let us check for dummy encoding for categorical variables

dummies = pd.get_dummies(df_consol['NewCreditCustomer'], prefix =
'NewCreditCustomer', drop_first = True)

df_consol = pd.concat([df_consol, dummies] , axis = 1)

df_consol.drop('NewCreditCustomer', axis = 1, inplace =True)

dummies1 = pd.get_dummies(df_consol['Country'], prefix = 'Country', drop_first =
True)

df_consol = pd.concat([df_consol, dummies1] , axis = 1)

df_consol.drop('Country', axis = 1, inplace =True)
```

```
# Unique Values in EmploymentDurationCurrentEmployer

Cur_empl_duration = list(df['EmploymentDurationCurrentEmployer'].unique())

print(Cur_empl_duration)

dummies2 = pd.get_dummies(df_consol['EmploymentDurationCurrentEmployer'], prefix =
'EmploymentDurationCurrentEmployer',
                           dummy_na = True, drop_first = True)

df_consol = pd.concat([df_consol, dummies2] , axis = 1)

df_consol.drop('EmploymentDurationCurrentEmployer', axis = 1, inplace =True)


dummies3 = pd.get_dummies(df_consol['ActiveScheduleFirstPaymentReached'], prefix =
'ActiveScheduleFirstPaymentReached',
                           dummy_na = True, drop_first = True
                            )

df_consol = pd.concat([df_consol, dummies3] , axis = 1)

dummies4 = pd.get_dummies(df_consol['Rating'], prefix = 'Rating',
                           dummy_na = True, drop_first = True
                            )

df_consol = pd.concat([df_consol, dummies4] , axis = 1)

dummies5 = pd.get_dummies(df_consol['Status'], prefix = 'Status',
                           dummy_na = True, drop_first = True
                            )

df_consol = pd.concat([df_consol, dummies5] , axis = 1)

dummies6 = pd.get_dummies(df_consol['Restructured'], prefix = 'Restructured',
                           dummy_na = True, drop_first = True
                            )

df_consol = pd.concat([df_consol, dummies6] , axis = 1)

df_consol.drop(['ActiveScheduleFirstPaymentReached', 'Rating', 'Status',
'Restructured'], axis = 1, inplace =True)

# convert strings to datatime object datatype

# check the reason for coercion for the MaturityDate_Last Column

df_consol['LoanDate'] = pd.to_datetime(df_consol['LoanDate'], format = '%Y-%m-%d')

df_consol['MaturityDate_Last'] = pd.to_datetime(df_consol['MaturityDate_Last'],
```

```
                                                                errors = 'coerce', format
='%Y-%m-%d')

df_consol['diff_days'] = (df_consol['MaturityDate_Last'] - df_consol['LoanDate']) /
np.timedelta64(1, 'D')

df_consol.drop(['LoanDate', 'MaturityDate_Last'], axis = 1, inplace =True)

print(df_consol.dtypes)

print(df_consol.head(10))

df_consol.to_csv("dataconsol.csv")


# print(df_consol.head())

# Number Missing in consolidated dataframe

number_missing = df_consol.isnull().sum()

missing_df = pd.DataFrame(number_missing)

missing_df.columns = ['Missing_Number']

#missing_df = pd.DataFrame(missing_df, columns = column_name)

number_missing.to_csv("datamiss2.csv")

#print(missing_df.head(60))

# Missing Values Bar Chart
# plot only if missing

only_miss_df = missing_df[missing_df['Missing_Number'] != 0]

only_miss_df.to_csv("onlymiss.csv")

ax = only_miss_df.plot.barh(figsize=(12, 8))

ax.bar_label(ax.containers[0])

# Now next steps are to check multi collinearity and correlation matrices
# Question is how to check if column values are real and not categorical without
looking at the data?
# Not sure

df_for_correl = df_consol

df_for_correl.drop(['VerificationType','ActiveScheduleFirstPaymentReached_nan',
```

```
                    'Rating_nan', 'Status_nan', 'Restructured_nan',
                    'LanguageCode', 'Age', 'Gender', 'IncomeTotal',
'EmploymentDurationCurrentEmployer_nan'], axis = 1, inplace =True)

df_for_correl.corr().to_csv("corr_matrix.csv")

print(df_for_correl.corr())

corr_dict =df_for_correl.corr().to_dict('dict')

#print(corr_dict)


def iterate_nest_Dict(data_dict):


    for key, value in data_dict.items():

        if isinstance(value, dict):

            for key_value in  iterate_nest_Dict(value):
                yield (key, *key_value)
        else:

            yield (key, value)

# now let us attempt to iterate through the correlation matrix dictionary
# only prints correlation coefficients that exceed 0.75.

list1 =[]

for key_value in iterate_nest_Dict(corr_dict):
    if key_value[0] != key_value[1]:
        if key_value[2] > 0.75:
            list1.append([key_value[0], key_value[1], key_value[2]])

#print(list1)

print(len(list1))

print("Variable 1", "," , "Variable 2", "," , "Corr_Coefficient")
print("______________________________________________")

for i in range(len(list1)):
    print(list1[i][0], "," , list1[i][1], ",", round(list1[i][2],3))


for i in range(len(list1)):
    list1[i][2] = str(round(list1[i][2],3))
```

```
print(list1)

rows = list1

# prints high correlated values to csv file

filename = 'corr_file.csv'

fields = ['Variable_1', 'Variable_2', 'Value']

with open(filename, 'w') as csvfile:
    csvwriter = csv.writer(csvfile)
    csvwriter.writerow(fields)
    csvwriter.writerows(rows)


# Next step is to fill missing values in the consolidated dataframe columns

# Columns in DataFrame with Missing Values are !
# they are the rows of only_miss_df

initial_model_df = df_consol.dropna()

count_target0 = 0

for i in initial_model_df.index:
    if (initial_model_df['DefaultDate'][i] == 0):
        count_target0 += 1

print(count_target0)

print(len(initial_model_df))

initial_model_df.to_csv("initialmodel.csv")
```

CODE FOR MINMAX SCALING AND ONE HOT DUMMY ENCODING

```
# -*- coding: utf-8 -*-
"""
Created on Sat Jun 18 17:02:02 2022

@author: ramra
"""

# This is Ramkishore Rao's DSA 5900 practicuum project

# THIS DOES FURTHER CLEANUP FROM PROJECT.PY
# AND CREATES A FILE THAT IS LOADED INTO ML MODELS

import pandas as pd
import numpy as np
import psycopg2
import csv
from random import seed
from csv import reader
import random
import matplotlib.pyplot as plt
from math import exp
from math import pi
from math import sqrt
from random import random
import seaborn as sns
import csv
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.compose import make_column_selector as selector
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error

seed(500)

# Reading Loan Dataset File

filename = 'LoanData.csv'
```

```
df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

# Print First 5 Rows of Dataframe

print(df.head())

# Now cleaning the dataframe
# Remove Unnecessary Columns

df.drop(columns= ['ReportAsOfEOD', 'LoanId', 'LoanNumber',
                  'BiddingStartedOn', 'BidsPortfolioManager', 'BidsApi',
                  'PartyId', 'ApplicationSignedHour', 'ApplicationSignedWeekday',
                  'County', 'City', 'EmploymentPosition', 'EL_V0', 'Rating_V0'],
inplace=True)

print(df.head())

df[['DefaultDate']] = df[['DefaultDate']].fillna(value=0)

df.loc[df['DefaultDate'] != 0, 'DefaultDate'] = 1

#check_missing_df = df.isna()

#check_missing_df.to_csv("datamiss.csv")

#number_missing = df.isnull().sum()

#number_missing.to_csv("datamiss.csv")

result = df.isna().mean()

df_consol = df.loc[: , result < .1]

# dropping additional unneeded columns

df_consol.drop(['BidsManual', 'ListedOnUTC', 'LoanApplicationStartedDate',
'MaturityDate_Original'], axis=1, inplace = True)


df_1 = df_consol

df_1_target_popped = df_1.pop('DefaultDate')

df_1['Defaulted'] = df_1_target_popped

df.drop(['FirstPaymentDate', 'LastPaymentOn'] , axis = 1, inplace =True)

max1 = df_1['UseOfLoan'].max() + 1
```

```
max2 = df_1['Education'].max() + 1
max3 = df_1['MaritalStatus'].max() + 1
max4 = df_1['EmploymentStatus'].max() + 1
max5 = df_1['OccupationArea'].max() + 1
max6 = df_1['HomeOwnershipType'].max() + 1

print(max1, max2, max3, max4, max5, max6)

df_1.loc[df_1['UseOfLoan'] < -0.5, 'UseOfLoan'] = 9
df_1.loc[df_1['Education'] < 0, 'Education'] = max2
df_1.loc[df_1['MaritalStatus'] < 0, 'MaritalStatus'] = max3
df_1.loc[df_1['EmploymentStatus'] < 0, 'EmploymentStatus'] = max4
df_1.loc[df_1['OccupationArea'] < 0, 'OccupationArea'] = max5
df_1.loc[df_1['HomeOwnershipType'] < 0, 'HomeOwnershipType'] = max6

df_1 = df_1.dropna()

#df_1.drop(columns = 'UseOfLoan')

df_1.to_csv("initialmodel1.csv")

target_name = "Defaulted"

df2= df_1.drop(columns=[target_name])

sc = MinMaxScaler()

# check the reason for coercion for the MaturityDate_Last Column

df2['LoanDate'] = pd.to_datetime(df2['LoanDate'], format = '%Y-%m-%d')

df2['MaturityDate_Last'] = pd.to_datetime(df2['MaturityDate_Last'],
                                                         errors = 'coerce', format
='%Y-%m-%d')

df2['diff_days'] = (df2['MaturityDate_Last'] - df2['LoanDate']) / np.timedelta64(1,
'D')

df2.drop(['LoanDate', 'MaturityDate_Last'], axis = 1, inplace =True)

df2.drop(['FirstPaymentDate', 'LastPaymentOn'], axis = 1, inplace =True)

print(df2.head())

# get numeric data

cols = ['AppliedAmount', 'Amount', 'Interest', 'LoanDuration', 'MonthlyPayment',
'IncomeFromPrincipalEmployer',
          'IncomeFromPension' , 'IncomeFromFamilyAllowance' ,
'IncomeFromSocialWelfare', 'IncomeFromLeavePay',
```

```
          'IncomeFromChildSupport', 'IncomeOther', 'IncomeTotal',
'ExistingLiabilities', 'LiabilitiesTotal',
          'DebtToIncome', 'FreeCash', 'MonthlyPaymentDay', 'PlannedInterestTillDate'
, 'ExpectedLoss',
          'LossGivenDefault', 'ExpectedReturn', 'ProbabilityOfDefault',
'PrincipalOverdueBySchedule',
          'PrincipalPaymentsMade', 'InterestAndPenaltyPaymentsMade',
'PrincipalBalance', 'AmountOfPreviousLoansBeforeLoan',
          'Age', 'diff_days']

num_d = df2[cols]

# update the cols with their normalized values
df2[num_d.columns] = sc.fit_transform(num_d)

df2['Defaulted'] = df_1_target_popped

print(df2.head())

# now let us check for dummy encoding for categorical variables

dummies = pd.get_dummies(df2['NewCreditCustomer'], prefix = 'NewCreditCustomer',
drop_first = True)

df2 = pd.concat([df2, dummies] , axis = 1)

df2.drop('NewCreditCustomer', axis = 1, inplace =True)

dummies1 = pd.get_dummies(df2['Country'], prefix = 'Country', drop_first = True)

df2 = pd.concat([df2, dummies1] , axis = 1)

df2.drop('Country', axis = 1, inplace =True)

# Unique Values in EmploymentDurationCurrentEmployer

Cur_empl_duration = list(df2['EmploymentDurationCurrentEmployer'].unique())

print(Cur_empl_duration)

dummies2 = pd.get_dummies(df2['EmploymentDurationCurrentEmployer'], prefix =
'EmploymentDurationCurrentEmployer',
                           dummy_na = True, drop_first = True)

df2 = pd.concat([df2, dummies2] , axis = 1)

df2.drop('EmploymentDurationCurrentEmployer', axis = 1, inplace =True)


dummies3 = pd.get_dummies(df2['ActiveScheduleFirstPaymentReached'], prefix =
```

```
'ActiveScheduleFirstPaymentReached',
                           dummy_na = True, drop_first = True
                            )

df2 = pd.concat([df2, dummies3] , axis = 1)

dummies4 = pd.get_dummies(df2['Rating'], prefix = 'Rating',
                           dummy_na = True, drop_first = True
                            )

df2 = pd.concat([df2, dummies4] , axis = 1)

dummies5 = pd.get_dummies(df2['Status'], prefix = 'Status',
                           dummy_na = True, drop_first = True
                            )

df2 = pd.concat([df2, dummies5] , axis = 1)

dummies6 = pd.get_dummies(df2['Restructured'], prefix = 'Restructured',
                           dummy_na = True, drop_first = True
                            )

df2 = pd.concat([df2, dummies6] , axis = 1)

df2.drop(['ActiveScheduleFirstPaymentReached', 'Rating', 'Status', 'Restructured'],
axis = 1, inplace =True)


print(df2.head())

df2.to_csv("initialmodel2.csv")
```

PYTHON FILES: MODULAR
SEPARATE FILES CREATED FOR INITIAL
PREPROCESSING, PCA ANALYSIS, EACH MODEL TYPE

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jun 19 18:36:00 2022

@author: ramra
"""

# -*- coding: utf-8 -*-
"""
Created on Sat Jun  4 17:58:28 2022

@author: ramra
"""

# CODE TO PRINT ATTRIBUTE NAMES

# This is Ramkishore Rao's DSA 5900 practicuum project

import pandas as pd
import csv
from csv import reader
import csv

# Reading Loan Dataset File

filename = 'LoanData.csv'

df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

df.rename(columns = {'DefaultDate':'Defaulted'}, inplace = True)

df1 = df.pop('Defaulted')

df['Target Class: Defaulted'] = df1

count = 1

print("Feature No", "Feature Name", sep = '\t')

print("")

for i in df.columns:
    print(count, i, sep='\t\t\t')
    count += 1
```

UNSCALED - FILE TO MAKE CORRELATION WITH TARGET CLASS

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jun  9 19:18:47 2022

@author: ramra
"""

import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats

filename = 'initialmodel.csv'

df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

df1 = df.pop('DefaultDate')

df['Defaulted'] = df1

df.drop(['Unnamed: 0', 'FirstPaymentDate', 'LastPaymentOn'] , axis = 1, inplace
=True)

print(df.head())

a = df[df.columns[0:]].corr()['Defaulted'][:]

a.to_csv("correlation.csv")
```

PRINCIPAL COMPONENT
ANALYSIS -1
USED STANDARD SCALER

```
# -*- coding: utf-8 -*-
"""
Created on Sat Jun 18 17:02:02 2022

@author: ramra
"""

# PCA-1

# This is Ramkishore Rao's DSA 5900 practicuum project

import pandas as pd
import numpy as np
import psycopg2
import csv
from random import seed
from csv import reader
import random
import matplotlib.pyplot as plt
from math import exp
from math import pi
from math import sqrt
from random import random
import seaborn as sns
import csv
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.compose import make_column_selector as selector
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error

seed(500)

# Reading Loan Dataset File

filename = 'LoanData.csv'
```

```
df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

# Print First 5 Rows of Dataframe

print(df.head())

# Now cleaning the dataframe
# Remove Unnecessary Columns

df.drop(columns= ['ReportAsOfEOD', 'LoanId', 'LoanNumber',
                  'BiddingStartedOn', 'BidsPortfolioManager', 'BidsApi',
                  'PartyId', 'ApplicationSignedHour', 'ApplicationSignedWeekday',
                  'County', 'City', 'EmploymentPosition', 'EL_V0', 'Rating_V0'],
inplace=True)

print(df.head())

df[['DefaultDate']] = df[['DefaultDate']].fillna(value=0)

df.loc[df['DefaultDate'] != 0, 'DefaultDate'] = 1

#check_missing_df = df.isna()

#check_missing_df.to_csv("datamiss.csv")

#number_missing = df.isnull().sum()

#number_missing.to_csv("datamiss.csv")

result = df.isna().mean()

df_consol = df.loc[: , result < .1]

# dropping additional unneeded columns

df_consol.drop(['BidsManual', 'ListedOnUTC', 'LoanApplicationStartedDate',
'MaturityDate_Original'], axis=1, inplace = True)


df_1 = df_consol

df_1_target_popped = df_1.pop('DefaultDate')

df_1['Defaulted'] = df_1_target_popped

df.drop(['FirstPaymentDate', 'LastPaymentOn'] , axis = 1, inplace =True)

max1 = df_1['UseOfLoan'].max() + 1
```

```
max2 = df_1['Education'].max() + 1
max3 = df_1['MaritalStatus'].max() + 1
max4 = df_1['EmploymentStatus'].max() + 1
max5 = df_1['OccupationArea'].max() + 1
max6 = df_1['HomeOwnershipType'].max() + 1

print(max1, max2, max3, max4, max5, max6)

df_1.loc[df_1['UseOfLoan'] < -0.5, 'UseOfLoan'] = 9 #not sure what is happening here
yet!
df_1.loc[df_1['Education'] < 0, 'Education'] = max2
df_1.loc[df_1['MaritalStatus'] < 0, 'MaritalStatus'] = max3
df_1.loc[df_1['EmploymentStatus'] < 0, 'EmploymentStatus'] = max4
df_1.loc[df_1['OccupationArea'] < 0, 'OccupationArea'] = max5
df_1.loc[df_1['HomeOwnershipType'] < 0, 'HomeOwnershipType'] = max6

df_1 = df_1.dropna()

#df_1.drop(columns = 'UseOfLoan')

#df_1.to_csv("initialmodel1.csv")

target_name = "Defaulted"

#df2= df_1

df2 = df_1.reset_index(drop = True)    # Apply reset_index function

sc = StandardScaler()

# get numeric data

cols = ['AppliedAmount', 'Amount', 'Interest', 'LoanDuration', 'MonthlyPayment',
'IncomeFromPrincipalEmployer',
          'IncomeFromPension' , 'IncomeFromFamilyAllowance' ,
'IncomeFromSocialWelfare', 'IncomeFromLeavePay',
          'IncomeFromChildSupport', 'IncomeOther', 'IncomeTotal',
'ExistingLiabilities', 'LiabilitiesTotal',
          'DebtToIncome', 'FreeCash', 'MonthlyPaymentDay', 'PlannedInterestTillDate'
, 'ExpectedLoss',
          'LossGivenDefault', 'ExpectedReturn', 'ProbabilityOfDefault',
'PrincipalOverdueBySchedule',
          'PrincipalPaymentsMade', 'InterestAndPenaltyPaymentsMade',
'PrincipalBalance', 'AmountOfPreviousLoansBeforeLoan', 'Age', target_name ]


df2 = df2.iloc[0:5000]

print(df2)
```

```
cols1 = ['AppliedAmount', 'Amount', 'Interest', 'LoanDuration', 'MonthlyPayment',
'IncomeFromPrincipalEmployer',
          'IncomeFromPension' , 'IncomeFromFamilyAllowance' ,
'IncomeFromSocialWelfare', 'IncomeFromLeavePay',
          'IncomeFromChildSupport', 'IncomeOther', 'IncomeTotal',
'ExistingLiabilities', 'LiabilitiesTotal',
          'DebtToIncome', 'FreeCash', 'MonthlyPaymentDay', 'PlannedInterestTillDate'
, 'ExpectedLoss',
          'LossGivenDefault', 'ExpectedReturn', 'ProbabilityOfDefault',
'PrincipalOverdueBySchedule',
          'PrincipalPaymentsMade', 'InterestAndPenaltyPaymentsMade',
'PrincipalBalance', 'AmountOfPreviousLoansBeforeLoan', 'Age']


num_d = df2[cols1]

print(type(num_d))

# update the cols with their normalized values
num_d[num_d.columns] = sc.fit_transform(num_d)

#df2['Defaulted'] = df_1_target_popped

print(num_d)

pca = PCA(n_components=3)
principalComponents = pca.fit_transform(num_d)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2',
'principal component 3'])

print(principalDf)

finalDf = pd.concat([principalDf, df2[target_name]], axis = 1)

print(finalDf)

fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('Principal Component 1', fontsize = 10)
ax.set_ylabel('Principal Component 2', fontsize = 10)
ax.set_zlabel('Principal Component 3', fontsize = 10)
ax.set_title('3 component PCA', fontsize = 20)
targets = [0, 1]
colors = ['r', 'g']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Defaulted'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , finalDf.loc[indicesToKeep, 'principal component 3']
```

```
                , c = color
                , alpha=0.8
                , s = 50)
ax.legend(targets)
ax.grid()

print(pca.explained_variance_ratio_)
```

PRINCIPAL COMPONENT
ANALYSIS -2
USED STANDARD SCALER

```
# -*- coding: utf-8 -*-
"""
Created on Sat Jun 18 17:02:02 2022

@author: ramra
"""

# PCA-2

# This is Ramkishore Rao's DSA 5900 practicuum project

import pandas as pd
import numpy as np
import psycopg2
import csv
from random import seed
from csv import reader
import random
import matplotlib.pyplot as plt
from math import exp
from math import pi
from math import sqrt
from random import random
import seaborn as sns
import csv
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.compose import make_column_selector as selector
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error

seed(500)

# Reading Loan Dataset File

filename = 'LoanData.csv'
```

```
df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

# Print First 5 Rows of Dataframe

print(df.head())

# Now cleaning the dataframe
# Remove Unnecessary Columns

df.drop(columns= ['ReportAsOfEOD', 'LoanId', 'LoanNumber',
                  'BiddingStartedOn', 'BidsPortfolioManager', 'BidsApi',
                  'PartyId', 'ApplicationSignedHour', 'ApplicationSignedWeekday',
                  'County', 'City', 'EmploymentPosition', 'EL_V0', 'Rating_V0'],
inplace=True)

print(df.head())

df[['DefaultDate']] = df[['DefaultDate']].fillna(value=0)

df.loc[df['DefaultDate'] != 0, 'DefaultDate'] = 1

#check_missing_df = df.isna()

#check_missing_df.to_csv("datamiss.csv")

#number_missing = df.isnull().sum()

#number_missing.to_csv("datamiss.csv")

result = df.isna().mean()

df_consol = df.loc[: , result < .1]

# dropping additional unneeded columns

df_consol.drop(['BidsManual', 'ListedOnUTC', 'LoanApplicationStartedDate',
'MaturityDate_Original'], axis=1, inplace = True)


df_1 = df_consol

df_1_target_popped = df_1.pop('DefaultDate')

df_1['Defaulted'] = df_1_target_popped

df.drop(['FirstPaymentDate', 'LastPaymentOn'] , axis = 1, inplace =True)

max1 = df_1['UseOfLoan'].max() + 1
```

```
max2 = df_1['Education'].max() + 1
max3 = df_1['MaritalStatus'].max() + 1
max4 = df_1['EmploymentStatus'].max() + 1
max5 = df_1['OccupationArea'].max() + 1
max6 = df_1['HomeOwnershipType'].max() + 1

print(max1, max2, max3, max4, max5, max6)

df_1.loc[df_1['UseOfLoan'] < -0.5, 'UseOfLoan'] = 9 #not sure what is happening here
yet!
df_1.loc[df_1['Education'] < 0, 'Education'] = max2
df_1.loc[df_1['MaritalStatus'] < 0, 'MaritalStatus'] = max3
df_1.loc[df_1['EmploymentStatus'] < 0, 'EmploymentStatus'] = max4
df_1.loc[df_1['OccupationArea'] < 0, 'OccupationArea'] = max5
df_1.loc[df_1['HomeOwnershipType'] < 0, 'HomeOwnershipType'] = max6

df_1 = df_1.dropna()

#df_1.drop(columns = 'UseOfLoan')

#df_1.to_csv("initialmodel1.csv")

target_name = "Defaulted"

#df2= df_1

df2 = df_1.reset_index(drop = True)    # Apply reset_index function

sc = StandardScaler()

# get numeric data

cols = ['AppliedAmount', 'Amount', 'Interest', 'LoanDuration', 'MonthlyPayment',
'IncomeFromPrincipalEmployer',
          'IncomeFromPension' , 'IncomeFromFamilyAllowance' ,
'IncomeFromSocialWelfare', 'IncomeFromLeavePay',
          'IncomeFromChildSupport', 'IncomeOther', 'IncomeTotal',
'ExistingLiabilities', 'LiabilitiesTotal',
          'DebtToIncome', 'FreeCash', 'MonthlyPaymentDay', 'PlannedInterestTillDate'
, 'ExpectedLoss',
          'LossGivenDefault', 'ExpectedReturn', 'ProbabilityOfDefault',
'PrincipalOverdueBySchedule',
          'PrincipalPaymentsMade', 'InterestAndPenaltyPaymentsMade',
'PrincipalBalance', 'AmountOfPreviousLoansBeforeLoan', 'Age', target_name ]


df2 = df2.iloc[0:5000]

print(df2)
```

```
cols1 = ['AppliedAmount', 'Amount', 'Interest', 'LoanDuration', 'MonthlyPayment',
'IncomeFromPrincipalEmployer',
          'IncomeFromPension' , 'IncomeFromFamilyAllowance' ,
'IncomeFromSocialWelfare', 'IncomeFromLeavePay',
          'IncomeFromChildSupport', 'IncomeOther', 'IncomeTotal',
'ExistingLiabilities', 'LiabilitiesTotal',
          'DebtToIncome', 'FreeCash', 'MonthlyPaymentDay', 'PlannedInterestTillDate'
, 'ExpectedLoss',
          'LossGivenDefault', 'ExpectedReturn', 'ProbabilityOfDefault',
'PrincipalOverdueBySchedule',
          'PrincipalPaymentsMade', 'InterestAndPenaltyPaymentsMade',
'PrincipalBalance', 'AmountOfPreviousLoansBeforeLoan', 'Age']


num_d = df2[cols1]

print(type(num_d))

# update the cols with their normalized values
num_d[num_d.columns] = sc.fit_transform(num_d)

#df2['Defaulted'] = df_1_target_popped

print(num_d)

pca = PCA(n_components=5)
principalComponents = pca.fit_transform(num_d)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2',
'principal component 3',
                          'principal component 4', 'principal component 5'])

print(principalDf)

finalDf = pd.concat([principalDf, df2[target_name]], axis = 1)

print(finalDf)

print(pca.explained_variance_ratio_)

x_axis = [1, 2, 3, 4, 5 ]

plt.figure(figsize=(4, 4))
plt.plot(x_axis ,  pca.explained_variance_ratio_)

plt.xlabel('Number of Components')
plt.ylabel('Explained Variance')
plt.show()
```

```
plt.figure(figsize=(11,11))

def myplot(score,coeff,labels=None):
    xs = score[:,0]
    ys = score[:,1]
    n = coeff.shape[0]
    scalex = .6/(xs.max() - xs.min())
    scaley =.8/(ys.max() - ys.min())
    plt.scatter(xs * scalex,ys * scaley,s=5)
    for i in range(n):
        plt.arrow(0, 0, coeff[i,0], coeff[i,1],color = 'r',alpha = 0.5)
        if labels is None:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, "Var"+str(i+1), color =
'green', ha = 'center', va = 'center')
        else:
            plt.text(coeff[i,0]* 1.15, coeff[i,1] * 1.15, labels[i], color = 'g', ha
= 'center', va = 'center')

    plt.xlabel("PC{}".format(1))
    plt.ylabel("PC{}".format(2))
    plt.grid()

myplot(principalComponents[:,0:2],np.transpose(pca.components_[0:2,
:]),list(num_d.columns))
plt.show()
```

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun  8 16:09:20 2022

@author: ramra
"""
# Next series of python files
# present the model coding in python

# LOGISTIC REGRESSION
# WITH TUNING

import pandas as pd
import numpy as np
import psycopg2
import csv
from random import seed
from csv import reader
import random
import matplotlib.pyplot as plt
from math import exp
from math import pi
from math import sqrt
from random import random
import seaborn as sns
import csv

#sklearn Imports

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

filename = 'initialmodel2.csv'
```

```
df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

df1 = df.pop('Defaulted')

df['Defaulted'] = df1

df.drop(['Unnamed: 0'] , axis = 1, inplace =True)

#df.drop(['FirstPaymentDate', 'LastPaymentOn'], axis = 1, inplace =True)

df = df.dropna()

print(df.head())

# Split dataframe into X and y

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

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

print(X_train.head(), len(X_train))

print(X_test.head(), len(X_test))

print(y_train.head(), len(y_train))

print(y_test.head(), len(y_test))

# Split train into train1 and val1

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state
= 1, test_size = 0.15)

print(X_train1.head(), len(X_train1))

print(X_val1.head(), len(X_val1))

print(y_train1.head(), len(y_train1))

print(y_val1.head(), len(y_val1))

# Logistic Regression Model Fitting with GridSearchCV
# GridSearch with varying alpha values
```

```
# 5 fold cross validation is being checked

lr_Pipeline = Pipeline([('lr', LogisticRegression(max_iter = 200, random_state =
42))])

param_grid = [

    {'lr__penalty' : ['l1', 'l2', 'elasticnet'],
    'lr__C' : [1, 5, 10],
    'lr__solver' : ['lbfgs', 'liblinear', 'saga'],
    'lr__l1_ratio': [0.2, 0.6]

    }

]

gs_lr = GridSearchCV(lr_Pipeline, param_grid, cv = 5, return_train_score = True,
verbose =2)

gs_lr = gs_lr.fit(X_train, y_train)

print(gs_lr.estimator.get_params())

print(gs_lr.best_index_)

print(gs_lr.best_params_)

cv_results = gs_lr.cv_results_

# print results of cross validation training

results_df = pd.DataFrame(
                             {
                             'rank_cv' : cv_results['rank_test_score'],
                             'params': cv_results['params'],
                             'cv_score(mean_cv)' : cv_results['mean_test_score'],
                             'cv_score(std_cv)': cv_results['std_test_score'],
                             'cv_score(mean_train)' : cv_results['mean_train_score'],
                             'cv_score(std_train)' : cv_results['std_train_score']
                             }
                             )

list1 =[]

for i in results_df.index:
    list1.append(str(results_df['params'][i]['lr__penalty']) + ',' +
str(results_df['params'][i]['lr__C'])
                 + ',' + str(results_df['params'][i]['lr__solver']) + ',' +
str(results_df['params'][i]['lr__l1_ratio']))
```

```
results_df = results_df.join(pd.DataFrame({'params1': list1}))

results_df = results_df.sort_values(by = ['rank_cv'], ascending = True)

results_df.to_csv("TRLRResultsCV.csv")

pd.set_option('display.max_colwidth', 100)

print(results_df)

plt.figure(figsize=(8,8))

plt.plot(results_df['cv_score(mean_train)'], results_df['params1'], label="Train",
linewidth = 5)

plt.plot(results_df['cv_score(mean_cv)'], results_df['params1'], label = "CV",
linewidth = 3, dashes=[2, 2])

plt.xlabel('Accuracy')
plt.ylabel('Model Parameter')

plt.legend(loc="upper right")

plt.show()

results_df.to_csv("TRLRResultsCV.csv")

best_gs_lr_test_score = gs_lr.score(X_test,  y_test)

print(best_gs_lr_test_score)

print(gs_lr.best_index_)

print(gs_lr.best_params_)

y_predict2 = gs_lr.predict(X_test)
mse2 = mean_squared_error(y_predict2, y_test, squared=False)

print(mse2)

accuracy1 = accuracy_score(y_test, y_predict2)
precision1 = precision_score(y_test, y_predict2)
recall1 = recall_score(y_test, y_predict2)
F1_score = f1_score(y_test, y_predict2)
confusion_mat_test = confusion_matrix(y_test, y_predict2)

print(accuracy1, precision1, recall1, F1_score)
print(confusion_mat_test)
```

```
auc= roc_auc_score(y_test, y_predict2)

print(auc)

# function for ROC Curve Plotting

def plot_roc_curve(fper, tper):
    plt.plot(fper, tper, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()

probs = gs_lr.predict_proba(X_test)
probs = probs[:, 1]
fper, tper, thresholds = roc_curve(y_test, probs)
plot_roc_curve(fper, tper)

log_reg = LogisticRegression(max_iter = 200, random_state = 42,
                             penalty = 'l1', C =5, solver = 'liblinear' )

log_reg.fit(X_train, y_train)

y_predict3 = log_reg.predict(X_test)
mse3 = mean_squared_error(y_predict3, y_test, squared=False)

print(mse3)

names = X_train.columns

# Simple function to evaluate the coefficients of a regression

from IPython.display import display, HTML

def report_coef(names,coef,intercept):
    r = pd.DataFrame( { 'coef': coef, 'positive': coef>=0  }, index = names )
    r = r.sort_values(by=['coef'])
    r.to_csv("TRLogRegCoefficients.csv")
    display(r)
    print(f"Intercept: {intercept}")
    data_range = r[ ((r['coef'] >= 1.00 ) | (r['coef'] <= -1.00))]
    ax = data_range['coef'].plot(kind='barh', color=data_range['positive'].map(
        {True: 'r', False: 'b'}), figsize=(12, 8))

    for container in ax.containers:
        ax.bar_label(container)

coeff_array = np.round(log_reg.coef_.ravel(),2)
```

```
report_coef(
  names,
  coeff_array,
  log_reg.intercept_)
```

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun  8 16:09:20 2022

@author: ramra
"""
# Next series of python files
# present the model coding in python

# MULTINOMIAL NAIVE BAYES
# WITH TUNING

import pandas as pd
import numpy as np
import psycopg2
import csv
from random import seed
from csv import reader
import random
import matplotlib.pyplot as plt
from math import exp
from math import pi
from math import sqrt
from random import random
import seaborn as sns
import csv

#sklearn Imports

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

filename = 'initialmodel2.csv'

df = pd.read_csv(
```

```
        filename, on_bad_lines="skip", engine="python"
    )

df1 = df.pop('Defaulted')

df['Defaulted'] = df1

df.drop(['Unnamed: 0'] , axis = 1, inplace =True)

df = df.dropna()

#df.drop(['FirstPaymentDate', 'LastPaymentOn'], axis = 1, inplace =True)

print(df.head())

# Split dataframe into X and y

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

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

print(X_train.head(), len(X_train))

print(X_test.head(), len(X_test))

print(y_train.head(), len(y_train))

print(y_test.head(), len(y_test))

# MNB Model Fitting with GridSearchCV
# GridSearch with varying alpha values
# 5 fold cross validation is being checked

mnb_Pipeline = Pipeline([('mnb', MultinomialNB())])

param_grid = {'mnb__alpha': [1e-4, 1e-3, 1e-2, 1e-1, 1]}

gs_mnb = GridSearchCV(mnb_Pipeline, param_grid, cv = 5, return_train_score = True,
verbose =2)

gs_mnb = gs_mnb.fit(X_train, y_train)

print(gs_mnb.estimator.get_params())

print(gs_mnb.best_index_)
```

```
print(gs_mnb.best_params_)

cv_results = gs_mnb.cv_results_

# print results of cross validation training

results_df = pd.DataFrame(
                            {
                             'rank_cv' : cv_results['rank_test_score'],
                             'params': cv_results['params'],
                             'cv_score(mean_cv)' : cv_results['mean_test_score'],
                             'cv_score(std_cv)': cv_results['std_test_score'],
                             'cv_score(mean_train)' :
cv_results['mean_train_score'],
                             'cv_score(std_train)' : cv_results['std_train_score']
                             }
                            )

pd.set_option('display.max_colwidth', 100)

list1 = []

for i in results_df.index:
    list1.append(str(results_df['params'][i]['mnb__alpha']))

results_df = results_df.join(pd.DataFrame({'params1': list1}))

results_df = results_df.sort_values(by = ['rank_cv'], ascending = True)

results_df.to_csv("MNBResultsCV.csv")

plt.plot(results_df['cv_score(mean_train)'], results_df['params1'], label="Train")

plt.plot(results_df['cv_score(mean_cv)'], results_df['params1'], label = "CV")

plt.xlabel('Accuracy')

plt.ylabel('Model Parameter')

plt.legend(loc="upper right")

plt.xlim(0.8372,0.8378)


plt.show()

print(results_df)

best_gs_mnb_test_score = gs_mnb.score(X_test,  y_test)
```

```
print(best_gs_mnb_test_score)

y_predict2 = gs_mnb.predict(X_test)
mse2 = mean_squared_error(y_predict2, y_test, squared=False)

print(mse2)

accuracy1 = accuracy_score(y_test, y_predict2)
precision1 = precision_score(y_test, y_predict2)
recall1 = recall_score(y_test, y_predict2)
F1_score = f1_score(y_test, y_predict2)
confusion_mat_test = confusion_matrix(y_test, y_predict2)

print(accuracy1, precision1, recall1, F1_score)
print(confusion_mat_test)

auc= roc_auc_score(y_test, y_predict2)

print(auc)

# function for ROC Curve Plotting

def plot_roc_curve(fper, tper):
    plt.plot(fper, tper, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()

probs = gs_mnb.predict_proba(X_test)
probs = probs[:, 1]
fper, tper, thresholds = roc_curve(y_test, probs)
plot_roc_curve(fper, tper)

#Model with Best Params, this warrants a recheck.
# I picked the best model from GridSearchCV and retrained on the same set as I was
not able to retrieve from GridSearchCV

mnb_clf = MultinomialNB(alpha= 1.0)

mnb_clf.fit(X_train, y_train)

y_predict3 = mnb_clf.predict(X_test)
mse3 = mean_squared_error(y_predict3, y_test, squared=False)

print(mse3)
```

```
from IPython.display import display, HTML

coeff_array = mnb_clf.feature_log_prob_

names = X_train.columns

print(coeff_array)

list1 =[]

for i in range(len(coeff_array[0])):
    list1.append(abs(coeff_array[0][i] - coeff_array[1][i]))
    if coeff_array[0][i] >=coeff_array[1][i]:
        print(names[i], coeff_array[0][i], "Class_0_dominates")

    else:
        print(names[i], coeff_array[1][i], "Class_1_dominates")


def report_coef(names, coef):
    r = pd.DataFrame( { 'coef': coef, 'more_imp': coef>=0.01, 'names': names  },
index = names )

    r = r.sort_values(by=['coef'])
    r.to_csv("MNBImp.csv")
    display(r)

    data_range = r[(r['coef'] >= 0.5 )]

    data_range = data_range[data_range['names'].str.contains("nan") == False]

    data_range.drop(['names'], axis = 1)

    ax = data_range['coef'].plot(kind='barh', color=data_range['more_imp'].map(
        {True: 'r', False: 'b'}), figsize=(11, 8))

    for container in ax.containers:
        ax.bar_label(container)

array1 = np.asarray(list1)

array1 = np.round(array1, 2)

report_coef(
  names,
  array1)
```

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun  8 16:09:20 2022

@author: ramra
"""
# Next series of python files
# present the model coding in python

# DECISION TREE
# WITH TUNING

import pandas as pd
import numpy as np
import psycopg2
import csv
from random import seed
from csv import reader
import random
import matplotlib.pyplot as plt
from math import exp
from math import pi
from math import sqrt
from random import random
import seaborn as sns
import csv

#sklearn Imports

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
```

```
filename = 'initialmodel2.csv'

df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

df1 = df.pop('Defaulted')

df['Defaulted'] = df1

df.drop(['Unnamed: 0'] , axis = 1, inplace =True)

df = df.dropna()

#df.drop(['FirstPaymentDate', 'LastPaymentOn'], axis = 1, inplace =True)

print(df.head())

# Split dataframe into X and y

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

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

print(X_train.head(), len(X_train))

print(X_test.head(), len(X_test))

print(y_train.head(), len(y_train))

print(y_test.head(), len(y_test))

# Split train into train1 and val1

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state
= 1, test_size = 0.15)

print(X_train1.head(), len(X_train1))

print(X_val1.head(), len(X_val1))

print(y_train1.head(), len(y_train1))

print(y_val1.head(), len(y_val1))
```

```
# Decision Tree Model Fitting with GridSearchCV
# GridSearch with varying alpha values
# 5 fold cross validation is being checked

dectree_Pipeline = Pipeline([('dt', DecisionTreeClassifier())])


param_grid = [

    {'dt__criterion' : ['gini', 'entropy'],
    'dt__max_depth' : [5, 10, 20],
    }

]

gs_dt = GridSearchCV(dectree_Pipeline, param_grid, cv = 5, return_train_score =
True, verbose =2)

gs_dt = gs_dt.fit(X_train, y_train)


print(gs_dt.best_index_)

print(gs_dt.best_params_)

#print(gs_dt.estimator.get_params())

cv_results = gs_dt.cv_results_

results_df = pd.DataFrame(
                            {
                             'rank_cv' : cv_results['rank_test_score'],
                             'params': cv_results['params'],
                             'cv_score(mean_cv)' : cv_results['mean_test_score'],
                             'cv_score(std_cv)': cv_results['std_test_score'],
                             'cv_score(mean_train)' :
cv_results['mean_train_score'],
                             'cv_score(std_train)' : cv_results['std_train_score']
                             }
                            )

pd.set_option('display.max_colwidth', 100)

print(results_df)

list1 = []

for i in results_df.index:
    list1.append(str(results_df['params'][i]['dt__criterion']) + ',' +
str(results_df['params'][i]['dt__max_depth']))
```

```
results_df = results_df.join(pd.DataFrame({'params1': list1}))

results_df = results_df.sort_values(by = ['rank_cv'], ascending = True)

results_df.to_csv("DTResultsCV.csv")

plt.plot(results_df['cv_score(mean_train)'], results_df['params1'], label="Train")

plt.plot(results_df['cv_score(mean_cv)'], results_df['params1'], label = "CV")

plt.xlabel('Accuracy')
plt.ylabel('Model Parameter')

plt.legend(loc="upper right")

plt.xlim(0.9,1.0)

plt.show()

best_gs_dt_test_score = gs_dt.score(X_test,  y_test)

print(best_gs_dt_test_score)

y_predict2 = gs_dt.predict(X_test)
mse2 = mean_squared_error(y_predict2, y_test, squared=False)

print(mse2)

accuracy1 = accuracy_score(y_test, y_predict2)
precision1 = precision_score(y_test, y_predict2)
recall1 = recall_score(y_test, y_predict2)
F1_score = f1_score(y_test, y_predict2)
confusion_mat_test = confusion_matrix(y_test, y_predict2)

print(accuracy1, precision1, recall1, F1_score)
print(confusion_mat_test)

auc= roc_auc_score(y_test, y_predict2)

print(auc)

# function for ROC Curve Plotting

def plot_roc_curve(fper, tper):
    plt.plot(fper, tper, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
```

```
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()

probs = gs_dt.predict_proba(X_test)
probs = probs[:, 1]
fper, tper, thresholds = roc_curve(y_test, probs)
plot_roc_curve(fper, tper)

#Model with Best Params, this warrants a recheck.
# I picked the best model from GridSearchCV and retrained on the same set as I was
not able to retrieve from GridSearchCV

tree_clf = DecisionTreeClassifier(criterion='entropy', max_depth = 20)

tree_clf.fit(X_train, y_train)

y_predict3 = tree_clf.predict(X_test)
mse3 = mean_squared_error(y_predict3, y_test, squared=False)

print(mse3)

from IPython.display import display, HTML

feature_array = np.round(tree_clf.feature_importances_.ravel(),4)

names = X_train.columns

print(type(tree_clf.feature_importances_))

def report_coef(names, coef):
    r = pd.DataFrame( { 'coef': coef, 'more_imp': coef>=0.01  }, index = names )
    r = r.sort_values(by=['coef'])
    r.to_csv("FeatureDTImp.csv")
    display(r)

    data_range = r[(r['coef'] >= 0.001 )]
    ax = data_range['coef'].plot(kind='barh', color=data_range['more_imp'].map(
        {True: 'r', False: 'b'}), figsize=(11, 8))

    for container in ax.containers:
        ax.bar_label(container)


report_coef(
  names,
  feature_array)
```

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun  8 16:09:20 2022

@author: ramra
"""
# Next series of python files
# present the model coding in python

# ENSEMBLE FOREST
# WITH TUNING

import pandas as pd
import numpy as np
import psycopg2
import csv
from random import seed
from csv import reader
import random
import matplotlib.pyplot as plt
from math import exp
from math import pi
from math import sqrt
from random import random
import seaborn as sns
import csv

#sklearn Imports

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
```

```
filename = 'initialmodel2.csv'

df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

df1 = df.pop('Defaulted')

df['Defaulted'] = df1

df.drop(['Unnamed: 0'] , axis = 1, inplace =True)

df = df.dropna()

#df.drop(['FirstPaymentDate', 'LastPaymentOn'], axis = 1, inplace =True)

print(df.head())

# Split dataframe into X and y

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

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

print(X_train.head(), len(X_train))

print(X_test.head(), len(X_test))

print(y_train.head(), len(y_train))

print(y_test.head(), len(y_test))

# Split train into train1 and val1

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state
= 1, test_size = 0.15)

print(X_train1.head(), len(X_train1))

print(X_val1.head(), len(X_val1))

print(y_train1.head(), len(y_train1))

print(y_val1.head(), len(y_val1))
```

```
# Split dataframe into X and y

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

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

print(X_train.head(), len(X_train))

print(X_test.head(), len(X_test))

print(y_train.head(), len(y_train))

print(y_test.head(), len(y_test))

# Split train into train1 and val1

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state
= 1, test_size = 0.15)

print(X_train1.head(), len(X_train1))

print(X_val1.head(), len(X_val1))

print(y_train1.head(), len(y_train1))

print(y_val1.head(), len(y_val1))

# AdaBoost Model Fitting with GridSearchCV
# GridSearch with varying alpha values
# 5 fold cross validation is being checked

adaboost_Pipeline = Pipeline([('ada', AdaBoostClassifier())])

param_grid = [

    {'ada__n_estimators' : [5, 10, 20, 50, 100],
    'ada__learning_rate' : [0.1, 0.5, 1.0, 2.0, 5.0],
    }

]

gs_ada = GridSearchCV(adaboost_Pipeline, param_grid, cv = 5, return_train_score =
True, verbose =2)
```

```
gs_ada  = gs_ada.fit(X_train, y_train)

print(gs_ada.estimator.get_params())

print(gs_ada.best_index_)

print(gs_ada.best_params_)

cv_results = gs_ada.cv_results_

# print results of cross validation training

results_df = pd.DataFrame(
                            {
                             'rank_cv' : cv_results['rank_test_score'],
                             'params': cv_results['params'],
                             'cv_score(mean_cv)' : cv_results['mean_test_score'],
                             'cv_score(std_cv)': cv_results['std_test_score'],
                             'cv_score(mean_train)' :
cv_results['mean_train_score'],
                             'cv_score(std_train)' : cv_results['std_train_score']
                              }
                              )


pd.set_option('display.max_colwidth', 100)

list1 = []

for i in results_df.index:
    list1.append(str(results_df['params'][i]['ada__n_estimators']) + ',' +
str(results_df['params'][i]['ada__learning_rate']))


results_df = results_df.join(pd.DataFrame({'params1': list1}))

results_df = results_df.sort_values(by = ['rank_cv'], ascending = True)

results_df.to_csv("EFResultsCV.csv")

plt.plot(results_df['cv_score(mean_train)'], results_df['params1'], label="Train")

plt.plot(results_df['cv_score(mean_cv)'], results_df['params1'], label = "CV")

plt.xlabel('Accuracy')
plt.ylabel('Model Parameter')

plt.legend(loc="upper right")

plt.xlim(0,1)
```

```
yticks = plt.gca().yaxis.get_major_ticks()
for i in range(len(yticks)):
    if i % 4 != 0:
        yticks[i].set_visible(False)

plt.xticks(fontsize=16)

plt.yticks(fontsize=16)


plt.show()

print(results_df)

best_gs_ada_test_score =gs_ada.score(X_test,  y_test)

print(best_gs_ada_test_score)

y_predict2 = gs_ada.predict(X_test)
mse2 = mean_squared_error(y_predict2, y_test, squared=False)

print(mse2)

accuracy1 = accuracy_score(y_test, y_predict2)
precision1 = precision_score(y_test, y_predict2)
recall1 = recall_score(y_test, y_predict2)
F1_score = f1_score(y_test, y_predict2)
confusion_mat_test = confusion_matrix(y_test, y_predict2)

print(accuracy1, precision1, recall1, F1_score)
print(confusion_mat_test)

auc= roc_auc_score(y_test, y_predict2)

print(auc)

# function for ROC Curve Plotting

def plot_roc_curve(fper, tper):
    plt.plot(fper, tper, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()

probs = gs_ada.predict_proba(X_test)
probs = probs[:, 1]
```

```
fper, tper, thresholds = roc_curve(y_test, probs)
plot_roc_curve(fper, tper)

#Model with Best Params, this warrants a recheck.
# I picked the best model from GridSearchCV and retrained on the same set as I was
not able to retrieve from GridSearchCV

ada_clf = AdaBoostClassifier(learning_rate = 1.0, n_estimators = 100)

ada_clf.fit(X_train, y_train)

y_predict3 = ada_clf.predict(X_test)
mse3 = mean_squared_error(y_predict3, y_test, squared=False)

print(mse3)

from IPython.display import display, HTML

feature_array = np.round(ada_clf.feature_importances_.ravel(),4)

names = X_train.columns

print(type(ada_clf.feature_importances_))

def report_coef(names, coef):
    r = pd.DataFrame( { 'coef': coef, 'more_imp': coef>=0.01  }, index = names )
    r = r.sort_values(by=['coef'])
    r.to_csv("FeatureImpADABoost.csv")
    display(r)

    data_range = r[(r['coef'] >= 0.001 )]
    ax = data_range['coef'].plot(kind='barh', color=data_range['more_imp'].map(
        {True: 'r', False: 'b'}), figsize=(11, 8))

    for container in ax.containers:
        ax.bar_label(container)


report_coef(
  names,
  feature_array)
```

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Jun  8 16:09:20 2022

@author: ramra
"""
# Next series of python files
# present the model coding in python

# RANDOM FOREST
# WITH TUNING

import pandas as pd
import numpy as np
import psycopg2
import csv
from random import seed
from csv import reader
import random
import matplotlib.pyplot as plt
from math import exp
from math import pi
from math import sqrt
from random import random
import seaborn as sns
import csv

#sklearn Imports

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
```

```
filename = 'initialmodel2.csv'

df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

df1 = df.pop('Defaulted')

df['Defaulted'] = df1

df.drop(['Unnamed: 0'] , axis = 1, inplace =True)

df = df.dropna()

#df.drop(['FirstPaymentDate', 'LastPaymentOn'], axis = 1, inplace =True)

print(df.head())

# Split dataframe into X and y

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

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

print(X_train.head(), len(X_train))

print(X_test.head(), len(X_test))

print(y_train.head(), len(y_train))

print(y_test.head(), len(y_test))

# Split train into train1 and val1

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state
= 1, test_size = 0.15)

print(X_train1.head(), len(X_train1))

print(X_val1.head(), len(X_val1))

print(y_train1.head(), len(y_train1))
```

```
print(y_val1.head(), len(y_val1))

# Split dataframe into X and y

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

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

print(X_train.head(), len(X_train))

print(X_test.head(), len(X_test))

print(y_train.head(), len(y_train))

print(y_test.head(), len(y_test))

# Split train into train1 and val1

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state
= 1, test_size = 0.15)

print(X_train1.head(), len(X_train1))

print(X_val1.head(), len(X_val1))

print(y_train1.head(), len(y_train1))

print(y_val1.head(), len(y_val1))

# AdaBoost Model Fitting with GridSearchCV
# GridSearch with varying alpha values
# 5 fold cross validation is being checked

rf_Pipeline = Pipeline([('rf', RandomForestClassifier())])

param_grid = [

    {'rf__n_estimators' : [50, 100, 200],

    'rf__criterion' : ['gini', 'entropy'],

    'rf__max_features' : ['sqrt', 'log2'],


    }
```

```
]

gs_rf = GridSearchCV(rf_Pipeline, param_grid, cv = 5, return_train_score = True,
verbose =2)

gs_rf  = gs_rf.fit(X_train, y_train)

print(gs_rf.estimator.get_params())

print(gs_rf.best_index_)

print(gs_rf.best_params_)

cv_results = gs_rf.cv_results_

# print results of cross validation training

results_df = pd.DataFrame(
                            {'rank_cv' : cv_results['rank_test_score'],
                            'params': cv_results['params'],
                            'cv_score(mean_cv)' : cv_results['mean_test_score'],
                            'cv_score(std_cv)': cv_results['std_test_score'],
                            'cv_score(mean_train)' : cv_results['mean_train_score'],
                            'cv_score(std_train)' : cv_results['std_train_score']}
                            )

list1 =[]

for i in results_df.index:
    list1.append(str(results_df['params'][i]['rf__n_estimators']) + ',' +
str(results_df['params'][i]['rf__criterion'])
                 + ',' + str(results_df['params'][i]['rf__max_features']))


results_df = results_df.join(pd.DataFrame({'params1': list1}))

results_df = results_df.sort_values(by = ['rank_cv'], ascending = True)

results_df.to_csv("RFResultsCV.csv")

plt.plot(results_df['cv_score(mean_train)'], results_df['params1'], label="Train")

plt.plot(results_df['cv_score(mean_cv)'], results_df['params1'], label = "CV")

plt.xlabel('Accuracy')
plt.ylabel('Model Parameter')

plt.legend(loc="upper right")
```

```
plt.show()

pd.set_option('display.max_colwidth', 100)

print(results_df)

results_df.to_csv("RandomForestResults.csv")

best_gs_rf_test_score =gs_rf.score(X_test,  y_test)

print(best_gs_rf_test_score)

y_predict2 = gs_rf.predict(X_test)
mse2 = mean_squared_error(y_predict2, y_test, squared=False)

print(mse2)

accuracy1 = accuracy_score(y_test, y_predict2)
precision1 = precision_score(y_test, y_predict2)
recall1 = recall_score(y_test, y_predict2)
F1_score = f1_score(y_test, y_predict2)
confusion_mat_test = confusion_matrix(y_test, y_predict2)

print(accuracy1, precision1, recall1, F1_score)
print(confusion_mat_test)

auc= roc_auc_score(y_test, y_predict2)

print(auc)

#Model with Best Params, this warrants a recheck.
# I picked the best model from GridSearchCV and retrained on the same set as I was
not able to retrieve from GridSearchCV

rf_clf = RandomForestClassifier(criterion='entropy', n_estimators = 200,
max_features = 'auto')

rf_clf.fit(X_train, y_train)

y_predict3 = rf_clf.predict(X_test)
mse3 = mean_squared_error(y_predict3, y_test, squared=False)

print(mse3)

from IPython.display import display, HTML

feature_array = np.round(rf_clf.feature_importances_.ravel(),3)

names = X_train.columns
```

```
print(type(rf_clf.feature_importances_))

def report_coef(names, coef):
    r = pd.DataFrame( { 'coef': coef, 'more_imp': coef>=0.01  }, index = names )
    r = r.sort_values(by=['coef'])
    r.to_csv("RFFeatureImp.csv")
    display(r)

    data_range = r[(r['coef'] >= 0.005 )]
    ax = data_range['coef'].plot(kind='barh', color=data_range['more_imp'].map(
        {True: 'r', False: 'b'}), figsize=(11, 8))

    for container in ax.containers:
        ax.bar_label(container)


report_coef(
  names,
  feature_array)
```

TENSOR FLOW/KERAS FILES
NEURAL NET IMPLEMENTATION

```
# -*- coding: utf-8 -*-
"""ProjectTFFile.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1LPP9p-DQC9Q9zQBOk_3ED3gLEjFmx0od

This is Ramkishore Rao's Project - Application of Tensor Flow and Keras for Loan
Dataset
"""

# TENSORFLOW/KERAS
# DEFAULT

import tensorflow.keras
from tensorflow . keras .models import Sequential
from tensorflow . keras . layers import Dense , Activation
from tensorflow . keras . callbacks import EarlyStopping
from sklearn . model_selection import train_test_split

import numpy as np
import pandas as pd
from sklearn import metrics
import sklearn
from sklearn.model_selection import train_test_split
import io
import requests
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

df = pd.read_csv('/content/initialmodel2.csv', on_bad_lines="skip", engine="python")

df.head()
len(df)

"""# New Section"""

df1 = df.pop('Defaulted')

df['Defaulted'] = df1

df.drop(['Unnamed: 0'] , axis = 1, inplace =True)

df = df.dropna()
print(df.head())

"""Split dataframe into X and y"""
```

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

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

"""Split train into train1 and val1"""

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state
= 1, test_size = 0.15)

model = Sequential()
model.add(Dense(100, input_dim=X_train1.shape[1], activation='relu',
                kernel_initializer='random_normal'))
model.add(Dense(50,activation='relu',kernel_initializer='random_normal'))
model.add(Dense(25,activation='relu',kernel_initializer='random_normal'))
model.add(Dense(1,activation='sigmoid',kernel_initializer='random_normal'))
model.compile(loss='binary_crossentropy',
              optimizer=tensorflow.keras.optimizers.Adam(),
              metrics =['accuracy'])
monitor = EarlyStopping(monitor='val_loss', min_delta=1e-3,
    patience=5, verbose=1, mode='auto', restore_best_weights=True)

model.fit(X_train1,y_train1,validation_data=(X_test,y_test),
          callbacks=[monitor],verbose=2,epochs=1000)

pred = model.predict(X_test)
pred

mse2 = mean_squared_error(pred, y_test, squared=False)

mse2

pred1 = np.round(pred) # this takes continues output and transforms to binary values
of 0 and 1

pred1 # this is the output target value array for the test dataset

from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
```

```
accuracy1 = accuracy_score(y_test, pred1)
precision1 = precision_score(y_test, pred1)
recall1 = recall_score(y_test, pred1)
F1_score = f1_score(y_test, pred1)
confusion_mat_test = confusion_matrix(y_test, pred1)

confusion_mat_test

accuracy1

precision1

recall1

F1_score

auc= roc_auc_score(y_test, pred)

print(auc)

# Plot an ROC. pred - the predictions, y - the expected output.
def plot_roc_curve(fper, tper):
    plt.plot(fper, tper, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()

pred = model.predict(X_test)
fper, tper, thresholds = roc_curve(y_test, pred)
plot_roc_curve(fper, tper)
```

```
# -*- coding: utf-8 -*-
"""ProjectTFFile_With_CrossValdation.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/19mZU60uRewDaf3fEcVfxop_-ggxNfpu1

This is Ramkishore Rao's Project - Application of Tensor Flow and Keras for Loan
Dataset
"""

# TENSOR FLOW/KERAS
# CROSS VALIDATION ONLY, NOT USED IN REPORT

import tensorflow.keras
from tensorflow . keras .models import Sequential
from tensorflow . keras . layers import Dense , Activation
from tensorflow . keras . callbacks import EarlyStopping
from sklearn . model_selection import train_test_split

import numpy as np
import pandas as pd
from sklearn import metrics
import sklearn
from sklearn.model_selection import train_test_split
import io
import requests
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.model_selection import StratifiedKFold

!pip install theano

df = pd.read_csv('/content/initialmodel2.csv', on_bad_lines="skip", engine="python")

df.head()
len(df)

df1 = df.pop('Defaulted')

df['Defaulted'] = df1

df.drop(['Unnamed: 0'] , axis = 1, inplace =True)

df = df.dropna()
print(df.head())
```

```
"""Split dataframe into X and y"""

df = df.iloc[0:20000]

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

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

"""Split train into train1 and val1"""

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state
= 1, test_size = 0.15)

X = X_train.to_numpy()
y = y_train.to_numpy()

len(X)

len(y)

KFold = StratifiedKFold(n_splits =5, shuffle = True)
cvscores = []

for train, test in KFold.split(X, y):
  model = Sequential()
  model.add(Dense(100, input_dim=X_train1.shape[1], activation='relu',
                kernel_initializer='random_normal'))
  model.add(Dense(50,activation='relu',kernel_initializer='random_normal'))
  model.add(Dense(25,activation='relu',kernel_initializer='random_normal'))
  model.add(Dense(1,activation='sigmoid',kernel_initializer='random_normal'))
  model.compile(loss='binary_crossentropy',
              optimizer=tensorflow.keras.optimizers.Adam(),
              metrics =['accuracy'])
  model.fit(X[train], y[train], epochs = 150, batch_size = 10, verbose = 0)
  scores = model.evaluate(X[test], y[test], verbose = 0)
  print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
  cvscores.append(scores[1]*100)

print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))

pred = model.predict(X_test)
pred

mse2 = mean_squared_error(pred, y_test, squared=False)
```

```
mse2

pred1 = np.round(pred) # this takes continues output and transforms to binary values
of 0 and 1
pred1.shape

pred1 # this is the output target value array for the test dataset

from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score

accuracy1 = accuracy_score(y_test, pred1)
precision1 = precision_score(y_test, pred1)
recall1 = recall_score(y_test, pred1)
F1_score = f1_score(y_test, pred1)
confusion_mat_test = confusion_matrix(y_test, pred1)

confusion_mat_test

accuracy1

precision1

recall1

F1_score

auc= roc_auc_score(y_test, pred)

print(auc)

def plot_roc_curve(fper, tper):
    plt.plot(fper, tper, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()

probs = model.predict(X_test)
fper, tper, thresholds = roc_curve(y_test, probs)
plot_roc_curve(fper, tper)
```

```
# -*- coding: utf-8 -*-
"""ProjectTFFile_With_TuningLatest.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1PuQFSnb_P3iAXjYFPsyvQlo2Fg5BFrif

This is Ramkishore Rao's Project - Application of Tensor Flow and Keras for Loan
Dataset.  This one includes sckit learn's gridsearchCV for
hyperparameter tuning
"""

# TENSORFLOW/KERAS
# TUNED FOR HYPERPARAMETERS, 10 PCT OF DATASET

import tensorflow.keras
from tensorflow . keras .models import Sequential
from tensorflow . keras . layers import Dense , Activation
from tensorflow . keras . callbacks import EarlyStopping
from sklearn . model_selection import train_test_split

import numpy as np
import pandas as pd
from sklearn import metrics
import sklearn
from sklearn.model_selection import train_test_split
import io
import requests
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
from sklearn.model_selection import StratifiedKFold
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV

!pip install theano

df = pd.read_csv('/content/initialmodel2.csv', on_bad_lines="skip", engine="python")

df.head()
len(df)

df1 = df.pop('Defaulted')

df['Defaulted'] = df1

df.drop(['Unnamed: 0'] , axis = 1, inplace =True)
```

```
df = df.dropna()
print(df.head())

"""Split dataframe into X and y"""

df = df.iloc[0:20000] # let's take a subset of the dataset as the neural net takes
long to execute

X = df.iloc[:, :-1]
X1= df.iloc[:, :-1]

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

"""Split train into train1 and val1"""

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state
= 1, test_size = 0.15)

X = X_train.to_numpy()
y = y_train.to_numpy()

len(X)

len(y)

def create_model (optimizer = 'rmsprop', init = 'glorot_uniform'):
  model = Sequential()
  model.add(Dense(100, input_dim=X.shape[1], activation='relu',
                kernel_initializer=init))
  model.add(Dense(50,activation='relu',kernel_initializer= init))
  model.add(Dense(25,activation='relu',kernel_initializer=init))
  model.add(Dense(1,activation='sigmoid',kernel_initializer=init))

# Compile Model

  model.compile(loss='binary_crossentropy',
              optimizer= optimizer,
              metrics =['accuracy'])
  return model

# create model

model = KerasClassifier(build_fn = create_model, verbose = 0)

# grid search, epochs, batch size and optimizer with sckitlearn's gridsearchCV
```

```
optimizers = ['rmsprop', 'adam']
inits = ['glorot_uniform', 'normal', 'uniform']
epochs = [50, 150]
batches = [5, 20]

param_grid = dict(optimizer = optimizers, epochs = epochs, batch_size = batches,
init = inits)
grid = GridSearchCV(estimator = model, param_grid = param_grid, cv =3)
grid_result = grid.fit(X,y)

# summarize results

print("Best %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']

for mean, stdev, param in zip (means, stds, params):
  print("%f (%f) with %r" % (mean, stdev, param))

# run model with best parameters from sckit learn's gridsearchCV

best_model = Sequential()
best_model.add(Dense(100, input_dim=X.shape[1], activation='relu',
                  kernel_initializer='glorot_uniform'))
best_model.add(Dense(50,activation='relu',kernel_initializer= 'glorot_uniform'))
best_model.add(Dense(25,activation='relu',kernel_initializer='glorot_uniform'))
best_model.add(Dense(1,activation='sigmoid',kernel_initializer='glorot_uniform'))

best_model.compile(loss='binary_crossentropy',
                optimizer= tensorflow.keras.optimizers.Adam(),
                metrics =['accuracy'])

best_model.fit(X ,  y, epochs = 150, batch_size = 5)

"""Predictions from Best Model Provided Below"""

pred = best_model.predict(X_test)
pred

mse2 = mean_squared_error(pred, y_test, squared=False)

mse2

pred1 = np.round(pred) # this takes continues output and transforms to binary values
of 0 and 1
pred1.shape

pred1 # this is the output target value array for the test dataset
```

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score

accuracy1 = accuracy_score(y_test, pred1)
precision1 = precision_score(y_test, pred1)
recall1 = recall_score(y_test, pred1)
F1_score = f1_score(y_test, pred1)
confusion_mat_test = confusion_matrix(y_test, pred1)

confusion_mat_test

accuracy1

precision1

recall1

F1_score

auc= roc_auc_score(y_test, pred)

print(auc)

def plot_roc_curve(fper, tper):
    plt.plot(fper, tper, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()

probs = best_model.predict(X_test)
fper, tper, thresholds = roc_curve(y_test, probs)
plot_roc_curve(fper, tper)

# Save Neural Network to JSON File

from keras.models import model_from_json

# Serialize model to JSON

best_tuned_model_json = best_model.to_json()
```

```
with open("model.json" , "w") as json_file:
  json_file.write(best_tuned_model_json)

# Serialize weights to HDF5

best_model.save_weights("best_model.h5")
print("Saved Model to Disk")

# Load json and create model

json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()

loaded_model = model_from_json(loaded_model_json)


# Load weights into new Model


loaded_model.load_weights("best_model.h5")
print("Loaded model from disk")

# Evaluate Loaded Model on Test Data

loaded_model.compile(loss='binary_crossentropy',
                optimizer= tensorflow.keras.optimizers.Adam(),
                metrics =['accuracy'])

score = loaded_model.evaluate(X_test, y_test, verbose = 0)
print("%s: %.2f%%" % (loaded_model.metrics_names[1], score[1]*100))

!pip install matplotlib --upgrade

print(loaded_model.layers[0].weights[0].shape)

list1 = []

for x in loaded_model.layers[0].weights[0]:
  a = (np.sum(abs(x)))
  a = np.round(a,2)
  list1.append(a)

array1 = np.array(list1)

X1 = X1.columns
X1 = X1.tolist()
list2 =[]

for i in range(len(X1)):
```

```
  list2.append((list1[i], X1[i]))

print (list2)



def report_coef(names, coef):
    r = pd.DataFrame( { 'coef': coef, 'more_imp': coef>=30  }, index = names )
    r = r.sort_values(by=['coef'])
    r.to_csv("BestModelNeuralNet.csv")
    display(r)

    data_range = r[(r['coef'] >=30 )]
    ax = data_range['coef'].plot(kind='barh', color=data_range['more_imp'].map(
        {True: 'r', False: 'b'}), figsize=(11, 8))

    for container in ax.containers:
        ax.bar_label(container)

    plt.xlabel("Sum of Absolute Values of Weights")


report_coef(
  X1,
  array1)

# Let's look at history of training errors in the best_model retraining

!pip install plot_keras_history
from plot_keras_history import show_history, plot_history
print(best_model.history)

print(best_model.history.history.keys())

plt.plot(best_model.history.history['accuracy'])
plt.xlabel("Epoch")
plt.ylabel("Training Accuracy")
plt.legend(['train'], loc = 'lower right')
plt.title("Model Accuracy")

plt.plot(best_model.history.history['loss'])
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend(['train'], loc = 'upper right')
plt.title("Model Loss")
```

BREAKOUT OF TRAIN AND TEST SUBSET - 5% OF DATASET FOR FEDERATED ML IMPLEMENTATION

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jun 26 12:42:39 2022

@author: ramra
"""

# Train, Test Creation for Federated ML

import pandas as pd
import numpy as np

import csv

from random import seed
from csv import reader

filename = 'initialmodel2.csv'

df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

df1 = df.pop('Defaulted')

df['Defaulted'] = df1

df.drop(['Unnamed: 0'] , axis = 1, inplace =True)

df = df.dropna()

print(df.head())


df_sub = df.sample(frac = 0.05, random_state=2)

print(len(df_sub))

count = 0

for i in df_sub.index:
    if df_sub['Defaulted'][i] == 0:
        count += 1

print(count/len(df_sub))


df_train = df_sub.sample(frac = 0.80, random_state=2)

df_test = pd.concat([df_sub, df_train])
```

```
df_test = df_test.drop_duplicates(keep=False)

count = 0

for i in df_train .index:
    if df_train['Defaulted'][i] == 0:
        count += 1

print(count/len(df_train))

count = 0

for i in df_test .index:
    if df_test['Defaulted'][i] == 0:
        count += 1

print(count/len(df_test))

df3 = df_train.merge(df_test, how = 'inner' ,indicator=False)

print(df3)

df_train.to_csv("Train.csv")

df_test.to_csv("Test.csv")
```

FEDERATED ML
SIMULATES REMOTE MACHINE
NOTEBOOK FOR
DATA OWNER

In [1]:
```
import syft as sy
```

Part 1: Launch a Duet Server

In [2]:
```
duet = sy.launch_duet(loopback=True)
```

```
🎤  🎸  ♪♪♪ Starting Duet ♫♫♫  🎻  🎹

♫♫♫ > DISCLAIMER: Duet is an experimental feature currently in beta.
♫♫♫ > Use at your own risk.


    > ❤️ Love Duet? Please consider supporting our community!
    > https://github.com/sponsors/OpenMined

♫♫♫ > Punching through firewall to OpenGrid Network Node at:
♫♫♫ > http://ec2-18-218-7-180.us-east-2.compute.amazonaws.com:5000
♫♫♫ >
♫♫♫ > ...waiting for response from OpenGrid Network...
♫♫♫ > DONE!

♫♫♫ > STEP 1: Send the following code to your Duet Partner!

import syft as sy
duet = sy.join_duet(loopback=True)

♫♫♫ > Connecting...

♫♫♫ > CONNECTED!

data: DUET LIVE STATUS  *  Objects: 0  Requests: 0   Messages: 0  Request Handlers: 0
`searchable` is deprecated please use `pointable` in futurees: 0  Request Handlers: 0
♫♫♫ > DUET LIVE STATUS  -  Objects: 9  Requests: 0   Messages: 47748  Request Handler
s: 1
```

Part 2: Upload data to Duet Server

Let's say the data owner has a dataset of Iris flowers. He will upload the data to the duet server for other data scientists to use.

In [3]:
```
import pandas as pd
import numpy as np
import csv
from random import seed
from csv import reader
import torch
```

In [5]:
```
project_train = pd.read_csv("C:/Data Science and Analytics/DSA 5900/Final Deliverable/
project_train.drop(['Unnamed: 0'] , axis = 1, inplace =True)
```

```
project_train = project_train.dropna()
project_train.head()
```

Out[5]:

	VerificationType	LanguageCode	Age	Gender	AppliedAmount	Amount	Interest	LoanDurat
0	4.0	1	0.807692	0.0	0.040555	0.040555	0.084342	0.282
1	4.0	4	0.730769	0.0	0.039415	0.039415	0.053725	0.179
2	4.0	4	0.153846	0.0	0.088707	0.088707	0.033690	0.487
3	1.0	4	0.461538	0.0	0.192041	0.192041	0.173468	0.487
4	4.0	3	0.384615	1.0	0.454744	0.454744	0.039175	0.487

5 rows × 72 columns

In [6]:

```
X_train = project_train.loc[:, project_train.columns != "Defaulted"]
y_train = project_train["Defaulted"]
```

In [7]:

```
X_train = torch.FloatTensor(np.array(X_train))
y_train = torch.LongTensor(np.array(y_train))
```

In [8]:

```
print("data:")
print(X_train[0:5])
```

```
tensor([[4.0000e+00, 1.0000e+00, 8.0769e-01, 0.0000e+00, 4.0555e-02, 4.0555e-02,
         8.4342e-02, 2.8205e-01, 1.0116e-02, 9.0000e+00, 1.0000e+00, 6.0000e+00,
         7.0000e+00, 2.0000e+01, 1.0000e+01, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 3.7539e-04, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 4.8437e-04, 2.5806e-01, 3.5454e-02,
         1.1207e-01, 7.0716e-01, 5.8218e-01, 1.8430e-01, 4.8012e-02, 5.0000e+00,
         1.8463e-03, 1.2813e-03, 4.8003e-02, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0980e+03],
        [4.0000e+00, 4.0000e+00, 7.3077e-01, 0.0000e+00, 3.9415e-02, 3.9415e-02,
         5.3725e-02, 1.7949e-01, 1.1053e-02, 9.0000e+00, 1.0000e+00, 6.0000e+00,
         7.0000e+00, 2.0000e+01, 3.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.1856e-03, 5.0000e-02,
         8.6282e-06, 0.0000e+00, 0.0000e+00, 4.8437e-04, 6.7742e-01, 6.1697e-03,
         6.9775e-02, 7.9705e-01, 5.6003e-01, 8.9197e-02, 0.0000e+00, 6.0000e+00,
         8.6748e-03, 1.7489e-03, 4.0046e-02, 1.0000e+00, 1.7127e-02, 0.0000e+00,
         1.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 7.1100e+02],
        [4.0000e+00, 4.0000e+00, 1.5385e-01, 0.0000e+00, 8.8707e-02, 8.8707e-02,
         3.3690e-02, 4.8718e-01, 1.0395e-02, 9.0000e+00, 5.0000e+00, 6.0000e+00,
         7.0000e+00, 2.0000e+01, 3.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 2.2232e-03, 1.0000e-01,
         2.1774e-05, 0.0000e+00, 0.0000e+00, 4.8437e-04, 9.6774e-02, 1.5807e-03,
         5.6476e-02, 7.8429e-01, 5.5078e-01, 7.3485e-02, 0.0000e+00, 6.0000e+00,
         2.2225e-03, 1.1839e-03, 9.5313e-02, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         1.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
         0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.8190e+03],
        [1.0000e+00, 4.0000e+00, 4.6154e-01, 0.0000e+00, 1.9204e-01, 1.9204e-01,
         1.7347e-01, 4.8718e-01, 4.5083e-02, 9.0000e+00, 5.0000e+00, 6.0000e+00,
         7.0000e+00, 2.0000e+01, 3.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 2.8704e-03, 5.0000e-02,
         2.6036e-05, 0.0000e+00, 0.0000e+00, 4.8437e-04, 3.2258e-02, 2.5870e-02,
         3.0646e-01, 0.0000e+00, 5.9693e-01, 4.2349e-01, 3.4130e-02, 6.0000e+00,
         7.2047e-03, 4.1642e-02, 1.9266e-01, 1.0000e+00, 4.6779e-02, 0.0000e+00,
         1.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 1.0000e+00,
         0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 2.0060e+03],
        [4.0000e+00, 3.0000e+00, 3.8462e-01, 1.0000e+00, 4.5474e-01, 4.5474e-01,
         3.9175e-02, 4.8718e-01, 5.7208e-02, 9.0000e+00, 4.0000e+00, 6.0000e+00,
         7.0000e+00, 2.0000e+01, 5.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.4821e-03, 7.5000e-02,
         2.1885e-05, 0.0000e+00, 0.0000e+00, 4.8437e-04, 7.4194e-01, 0.0000e+00,
         4.2633e-02, 5.4232e-01, 5.5899e-01, 8.0498e-02, 0.0000e+00, 6.0000e+00,
         2.0260e-02, 4.8405e-02, 4.3977e-01, 3.0000e+00, 8.4247e-02, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
         0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 2.5640e+03]])
```

In [9]:
```python
print("target:")
print(y_train)
```

```
target:
tensor([1, 0, 0,  ..., 0, 0, 0])
```

In [10]:
```python
print("Length of dataset:", len(X_train))
```

```
Length of dataset: 8360
```

In [11]:
```python
print(type(X_train))
```

```
<class 'torch.Tensor'>
```

In [12]:
```python
print(type(y_train))
```

```
<class 'torch.Tensor'>
```

For doing machine learning using torch, we need the data to be converted to FloatTensors. Here, the data owner is explicitly doing the conversion before uploading the data. If he doesn't do that, it has to be converted in the data scientist's end as you've seen in the previous exercise.

In [13]:
```python
X_train = X_train.tag("Loan-data")
y_train = y_train.tag("Loan-target")

X_train = X_train.describe(
    "This is a train dataset for Credit Default classification."

)
y_train = y_train.describe("Labels for Defaulted: No, Yes")
```

In [14]:
```python
data_pointer = X_train .send(duet, searchable=True)
target_pointer = y_train.send(duet, searchable=True)
```

```
`searchable` is deprecated please use `pointable` in future
```

In [15]:
```python
# Once uploaded, the data owner can see the object stored in the tensor
duet.store
```

Out[15]:
```
[<syft.proxy.torch.TensorPointer object at 0x0000021CEF00EB50>, <syft.proxy.torch.Ten
sorPointer object at 0x0000021CEEF93070>]
```

In [16]:
```python
# To see it in a human-readable format, data owner can also pretty-print the tensor ir
duet.store.pandas
```

Out[16]:

	ID	Tags	Description	object_type
0	<UID: f9a561460bb34141bb5228d6ed1c9300>	[Loan-data]	This is a train dataset for Credit Default cla...	<class 'torch.Tensor'>
1	<UID: c308d3fe1ee14fc0a350e0305f5d2e63>	[Loan-target]	Labels for Defaulted: No, Yes	<class 'torch.Tensor'>

Part 3: Response to requests coming from Data Scientist

The data owner can add requests to be accepted or denied by adding them to request handlers. If he doesn't specify a `name`, then all the requests will be accepted.

In [17]:
```
duet.requests.add_handler(action="accept")
```

```
Exception in callback Transaction.__retry()
handle: <TimerHandle when=1847.828 Transaction.__retry()>
Traceback (most recent call last):
  File "C:\Users\ramra\anaconda3\lib\asyncio\events.py", line 80, in _run
    self._context.run(self._callback, *self._args)
  File "C:\Users\ramra\AppData\Roaming\Python\Python39\site-packages\aioice\stun.py", line 306, in __retry
    self.__future.set_exception(TransactionTimeout())
  File "C:\Users\ramra\anaconda3\lib\asyncio\futures.py", line 270, in set_exception
    raise exceptions.InvalidStateError(f'{self._state}: {self!r}')
asyncio.exceptions.InvalidStateError: FINISHED: <Future finished result=(Message(messa...b5 k+\\\x0e^'), ('10.0.0.91', 61610))>
```

Checkpoint 1 : Well done!

FEDERATED ML
SIMULATES LOCAL SERVER
NOTEBOOK FOR
DATA SCIENTIST

In [1]:
```
import syft as sy
```

Part 1: Join the Duet Server the Data Owner connected to

In [2]:
```
duet = sy.join_duet(loopback=True)
```

```
🎤  🎸  ♪♪♪ Joining Duet ♫♫♫  🎻  🎹

♫♫♫ > DISCLAIMER: Duet is an experimental feature currently in beta.
♫♫♫ > Use at your own risk.


    > ❤️ Love Duet? Please consider supporting our community!
    > https://github.com/sponsors/OpenMined

♫♫♫ > Punching through firewall to OpenGrid Network Node at:
♫♫♫ > http://ec2-18-218-7-180.us-east-2.compute.amazonaws.com:5000
♫♫♫ >
♫♫♫ > ...waiting for response from OpenGrid Network...
♫♫♫ > DONE!

♫♫♫ > CONNECTED!
```

Checkpoint 0 : Now STOP and run the Data Owner notebook until Checkpoint 1.

Part 2: Search for Available Data

In [3]:
```
# The data scientist can check the list of searchable data in Data Owner's duet store
duet.store.pandas
```

Out[3]:

	ID	Tags	Description	object_type
0	<UID: f9a561460bb34141bb5228d6ed1c9300>	[Loan-data]	This is a train dataset for Credit Default cla...	<class 'torch.Tensor'>
1	<UID: c308d3fe1ee14fc0a350e0305f5d2e63>	[Loan-target]	Labels for Defaulted: No, Yes	<class 'torch.Tensor'>

Data Scientist wants to use the Bank dataset. (S)He needs a pointer to the data and a pointer to the target for prediction.

In [4]:
```
data_ptr = duet.store[0]
target_ptr = duet.store[1]
```

`data_ptr` is a reference to the iris dataset remotely available on data owner's server.
`target_ptr` is a reference to the iris dataset LABELS remotely available on data owner's server

In [5]:
```
print(data_ptr)
print(target_ptr)
```

```
<syft.proxy.torch.TensorPointer object at 0x0000029812DD1430>
<syft.proxy.torch.TensorPointer object at 0x0000029812DD1670>
```

Part 3: Perform Logistic Regression on Bank dataset

Now the data scientist can perform machine learning on the data that is in the Data Owner's duet server, without the owner having to share his/her data.

Basic analysis

First the data scientist needs to know some basic information about the dataset.

1. The length of the dataset
2. The input dimension
3. The output dimension

These information have to be explicitly shared by the Data Owner. Let's try to find them in the data description.

In [6]:
```
print(duet.store.pandas["Description"][0])
print()
print(duet.store.pandas["Description"][1])
```

```
This is a train dataset for Credit Default classification.

Labels for Defaulted: No, Yes
```

Train model

In [7]:
```
import pandas as pd
import numpy as np
import csv
from random import seed
from csv import reader
import torch
```

In [8]:
```
in_dim = 71
out_dim = 2
n_samples = 8360
```

First, let's create our model for `Logistic Regression`. If you are already familiar with PyTorch, you will notice that the model is built almost the exact same way as you do in PyTorch.

The main difference is that here we inherit from `sy.Module` instead of `nn.Module`. We also need to pass in a variable called `torch_ref` which we will use internally for any calls that you would normally make to torch.

In [88]:
```
class SyNet(sy.Module):
    def __init__(self, torch_ref):
        super(SyNet, self).__init__(torch_ref=torch_ref)
        self.layer1 = self.torch_ref.nn.Linear(in_dim, 100)
        self.layer2 = self.torch_ref.nn.Linear(100, 50)
        self.layer3 = self.torch_ref.nn.Linear(50,25)
        self.out = self.torch_ref.nn.Linear(25, out_dim)

    def forward(self, x):
        x = self.torch_ref.nn.functional.relu(self.layer1(x))
        x = self.torch_ref.nn.functional.relu(self.layer2(x))
        x = self.torch_ref.nn.functional.relu(self.layer3(x))
        output = self.torch_ref.nn.functional.log_softmax(self.out(x), dim=1)
        return output
```

Now we can create a local model by passing our local copy of torch.

In [89]:
```
local_model = SyNet(torch)
```

Now we will send the local copy of the model to our partner's duet server.

In [90]:
```
remote_model = local_model.send(duet)
```

Let's create an alias for our partner's torch called `remote_torch` so we can refer to the local torch as torch and any operation we want to do remotely as `remote_torch`. Remember, the return values from `remote_torch` are Pointers, not the real objects. They mostly act the same when using them with other Pointers but they cannot be mixed with local torch objects.

In [91]:
```
remote_torch = duet.torch
```

We will get a pointer to our remote model parameters. Then we will set our optimizer. Here, we will be using `Adam optimizer`. `params` is a pointer to the list of parameters. `optim` is a reference to the Adam optimizer which can be used to optimize the remote model.

In [92]:
```
params = remote_model.parameters()
optim = remote_torch.optim.Adam(params=params, lr=0.01)
print("params:", params)
print("optim:", optim)
```

```
params: <syft.proxy.syft.lib.python.ListPointer object at 0x0000029819DCDFA0>
optim: <syft.proxy.torch.optim.AdamPointer object at 0x0000029812DE9250>
```

Now we will create our `train` function. It will take few parameters, like the `remote_model`, `torch_ref`, `optim` and `data_ptr` and `target_ptr`.

In [93]:
```
def train(iterations, model, torch_ref, optim, data_ptr, target_ptr):

    losses = []
```

```
    for i in range(iterations):

        optim.zero_grad()

        output = model(data_ptr)

        # nll_loss = negative log-liklihood loss
        loss = torch_ref.nn.functional.nll_loss(output, target_ptr.long())

        loss_item = loss.item()

        loss_value = loss_item.get(
            reason="To evaluate training progress", request_block=True, timeout_secs=5
        )

        if i % 10 == 0:
            print("Epoch", i, "loss", loss_value)

        losses.append(loss_value)

        loss.backward()

        optim.step()

    return losses
```

In [94]:

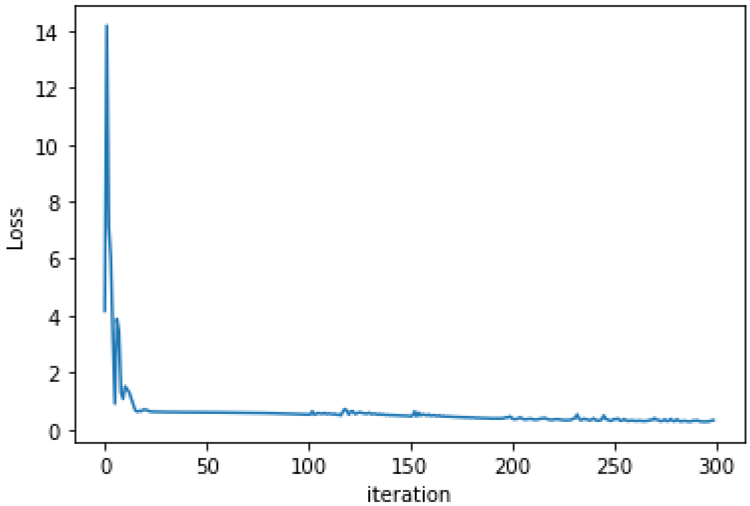
```
iteration = 300
losses = train(iteration, remote_model, remote_torch, optim, data_ptr, target_ptr)
```

```
Epoch 0 loss 4.16221284866333
Epoch 10 loss 1.513914704322815
Epoch 20 loss 0.691838800907135
Epoch 30 loss 0.6071500778198242
Epoch 40 loss 0.6033958792686462
Epoch 50 loss 0.597043514251709
Epoch 60 loss 0.5903545022010803
Epoch 70 loss 0.5813936591148376
Epoch 80 loss 0.5682870745658875
Epoch 90 loss 0.5486694574356079
Epoch 100 loss 0.5230823159217834
Epoch 110 loss 0.5581798553466797
Epoch 120 loss 0.5119012594223022
Epoch 130 loss 0.5870299935340881
Epoch 140 loss 0.492244690656662
Epoch 150 loss 0.4553671181201935
Epoch 160 loss 0.4724063575267792
Epoch 170 loss 0.44078072905540466
Epoch 180 loss 0.4117482304573059
Epoch 190 loss 0.38145458698272705
Epoch 200 loss 0.3809301257133484
Epoch 210 loss 0.354858156700134
Epoch 220 loss 0.33596453070640564
Epoch 230 loss 0.3753527104854584
Epoch 240 loss 0.3869413137435913
Epoch 250 loss 0.361598402261734
Epoch 260 loss 0.30032283067703247
Epoch 270 loss 0.3950543701648712
Epoch 280 loss 0.29164212942123413
Epoch 290 loss 0.3004149794578552
```

In [95]:
```python
import matplotlib.pyplot as plt
```

In [96]:
```python
plt.plot(range(iteration), losses)
plt.ylabel("Loss")
plt.xlabel("iteration")
```

Out[96]:
```
Text(0.5, 0, 'iteration')
```



In [108…
```python
iteration = 100
losses = train(iteration, remote_model, remote_torch, optim, data_ptr, target_ptr)
```

```
Epoch 0 loss 0.34083032608032227
Epoch 10 loss 0.32014209032058716
Epoch 20 loss 0.2577557861804962
Epoch 30 loss 0.23923061788082123
Epoch 40 loss 0.23766805231571198
Epoch 50 loss 0.2314949929714203
Epoch 60 loss 0.23056352138519287
Epoch 70 loss 0.26219597458839417
Epoch 80 loss 0.2306206077337265
Epoch 90 loss 0.23540011048316956
```

Download model

In [109…
```python
def get_local_model(model):
    if not model.is_local:
        local_model = model.get(
            request_block=True,
            reason="To run test and inference locally",
            timeout_secs=5,
        )
    else:
        local_model = model

    return local_model


local_model = get_local_model(remote_model)
```

Test on local data

In [45]:
```
import torch
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score
```

In [46]:
```
project_test = pd.read_csv("C:/Data Science and Analytics/DSA 5900/Final Deliverable/T
project_test.drop(['Unnamed: 0'] , axis = 1, inplace =True)

project_test = project_test.dropna()
project_test.head()
```

Out[46]:

	VerificationType	LanguageCode	Age	Gender	AppliedAmount	Amount	Interest	LoanDurat
0	4.0	1	0.442308	1.0	0.292715	0.292715	0.061933	0.282
1	4.0	1	0.576923	1.0	0.414284	0.414284	0.140012	1.000
2	4.0	3	0.442308	1.0	0.192136	0.192136	0.076600	0.487
3	4.0	6	0.519231	2.0	0.141229	0.141229	0.197744	0.487
4	4.0	4	0.346154	0.0	0.039415	0.039415	0.053530	0.487

5 rows × 72 columns

In [47]:
```
X_test = project_test.loc[:, project_test.columns != "Defaulted"]
y_test = project_test["Defaulted"]
```

In [48]:
```
X_test = torch.FloatTensor(np.array(X_test))
y_test = torch.LongTensor(np.array(y_test))
```

In [110...
```
preds = []
preds1 = []
probs1 = []
with torch.no_grad():
    for i in range(len(X_test)):
        sample = X_test[i]
        y_hat = local_model(sample.unsqueeze(0))
        preds1.append(y_hat)
        pred = y_hat.argmax().item()
        probs1.append(torch.max(torch.exp(y_hat)))
        #print(f"Prediction: {pred} Ground Truth: {y_test[i]}")
        preds.append(pred)
```

In [111...
```
acc = accuracy_score(y_test, preds)
print("Overall test accuracy", acc * 100)
```

```
Overall test accuracy 90.622009569378
```

In [112...
```
type(preds[0])
```

Out[112]:
```
int
```

In [113...
```
# the below code converts the log_softmax to softmax to estimate probabilities of the

probs = []
```

```
for i in range(len(preds1)):
    probs.append(torch.exp(preds1[i]))
```

In [129…
```
#probs
```

In [114…
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error
```

In [115…
```
precision1 = precision_score(y_test, preds)
recall1 = recall_score(y_test, preds)
F1_score = f1_score(y_test, preds)
confusion_mat_test = confusion_matrix(y_test, preds)
```

In [116…
```
precision1
```

Out[116]:
```
0.8645690834473324
```

In [117…
```
recall1
```

Out[117]:
```
0.8669410150891632
```

In [118…
```
F1_score
```

Out[118]:
```
0.8657534246575342
```

In [119…
```
confusion_mat_test
```

Out[119]:
```
array([[1262,   99],
       [  97,  632]], dtype=int64)
```

In [58]:
```
result = []
for i in range (len(probs)):
    result.append(probs[i].numpy())

result[0]
```

Out[58]:
```
array([[0.75392073, 0.24607928]], dtype=float32)
```

In [120…
```
result[0][0][0]
```

Out[120]:
```
0.75392073
```

In [121…
```
# the below code finds the probability for the positive class
resultprobs =[]
for i in range(len(result)):
    resultprobs.append(result[i][0][1])
```

In [122…
```
auc= roc_auc_score(y_test, resultprobs)
```

In [123…

```
auc
```

Out[123]:

```
0.9571887450625852
```

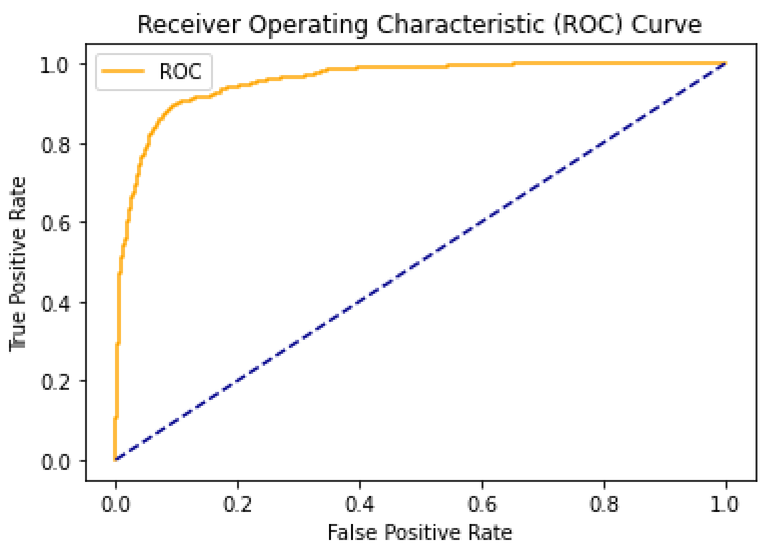
In [124…

```
from sklearn.metrics import roc_curve, auc
```

In [125…

```
def plot_roc_curve(fper, tper):
    plt.plot(fper, tper, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()

fper, tper, thresholds = roc_curve(y_test, resultprobs)
plot_roc_curve(fper, tper)
```



In [126…

```
mse2 = mean_squared_error(y_test, preds, squared=False) # this is after conversion to
```

In [127…

```
mse2
```

Out[127]:

```
0.306235047481865
```

In [128…

```
type(probs)
```

Out[128]:

```
list
```

In [ ]:

In [ ]:

In [ ]:

BAYESIAN OPTIMIZATION

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun  8 16:09:20 2022

@author: ramra
"""
# Next series of python files
# present the model coding in python

# RANDOM FOREST
# WITH TUNING

from numpy import mean
from sklearn.datasets import make_blobs
from sklearn.model_selection import cross_val_score

from skopt.space import Integer
from skopt.space import Categorical
from skopt.utils import use_named_args
from skopt import gp_minimize
from skopt.plots import plot_convergence

import pandas as pd
import numpy as np
import psycopg2
import csv
from random import seed
from csv import reader
import random
import matplotlib.pyplot as plt
from math import exp
from math import pi
from math import sqrt
from random import random
import seaborn as sns
import csv

#sklearn Imports

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
```

```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV


filename = 'initialmodel2.csv'

df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

df1 = df.pop('Defaulted')

df['Defaulted'] = df1

df.drop(['Unnamed: 0'] , axis = 1, inplace =True)

df = df.dropna()

#df.drop(['FirstPaymentDate', 'LastPaymentOn'], axis = 1, inplace =True)

print(df.head())

# Split dataframe into X and y

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

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

print(X_train.head(), len(X_train))

print(X_test.head(), len(X_test))

print(y_train.head(), len(y_train))

print(y_test.head(), len(y_test))

# Split train into train1 and val1
```

```
X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state
= 1, test_size = 0.15)

print(X_train1.head(), len(X_train1))

print(X_val1.head(), len(X_val1))

print(y_train1.head(), len(y_train1))

print(y_val1.head(), len(y_val1))

# Split dataframe into X and y

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

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

print(X_train.head(), len(X_train))

print(X_test.head(), len(X_test))

print(y_train.head(), len(y_train))

print(y_test.head(), len(y_test))

# Split train into train1 and val1

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state
= 1, test_size = 0.15)

print(X_train1.head(), len(X_train1))

print(X_val1.head(), len(X_val1))

print(y_train1.head(), len(y_train1))

print(y_val1.head(), len(y_val1))



# define the model
model = RandomForestClassifier()
# define the space of hyperparameters to search
search_space = [Integer(50, 300, name='n_estimators'),
```

```
Categorical(["sqrt","log2"],name="max_features"),
               Categorical(["gini","entropy"],name="criterion")]

# define the function used to evaluate a given configuration
@use_named_args(search_space)
def evaluate_model(**params):
        # something
        model.set_params(**params)
        # calculate 5-fold cross validation
        result = cross_val_score(model, X_train, y_train, cv=5, n_jobs=-1,
scoring='accuracy')
        # calculate the mean of the scores
        estimate = mean(result)
        return 1.0 - estimate

# perform optimization
result = gp_minimize(evaluate_model, search_space,  n_calls=25)

print(result.items())



# summarizing finding:
print('Best Accuracy: %.3f' % (1.0 - result.fun))
print('Best Parameters: criterion = %s, max_features=%s, n_estimators=%d' %
(result.x[2], result.x[1], result.x[0]))

plot_convergence(result)
```

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun  8 16:09:20 2022

@author: ramra
"""
# Next series of python files
# present the model coding in python

# RANDOM FOREST
# WITH TUNING

from numpy import mean
from sklearn.datasets import make_blobs
from sklearn.model_selection import cross_val_score

from skopt.space import Integer
from skopt.space import Real
from skopt.space import Categorical
from skopt.utils import use_named_args
from skopt import gp_minimize
from skopt.plots import plot_convergence

import pandas as pd
import numpy as np
import psycopg2
import csv
from random import seed
from csv import reader
import random
import matplotlib.pyplot as plt
from math import exp
from math import pi
from math import sqrt
from random import random
import seaborn as sns
import csv

#sklearn Imports

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score
```

```
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV


filename = 'initialmodel2.csv'

df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

df1 = df.pop('Defaulted')

df['Defaulted'] = df1

df.drop(['Unnamed: 0'] , axis = 1, inplace =True)

df = df.dropna()

#df.drop(['FirstPaymentDate', 'LastPaymentOn'], axis = 1, inplace =True)

print(df.head())

# Split dataframe into X and y

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

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

print(X_train.head(), len(X_train))

print(X_test.head(), len(X_test))

print(y_train.head(), len(y_train))

print(y_test.head(), len(y_test))
```

```
# Split train into train1 and val1

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state
= 1, test_size = 0.15)

print(X_train1.head(), len(X_train1))

print(X_val1.head(), len(X_val1))

print(y_train1.head(), len(y_train1))

print(y_val1.head(), len(y_val1))

# Split dataframe into X and y

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

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

print(X_train.head(), len(X_train))

print(X_test.head(), len(X_test))

print(y_train.head(), len(y_train))

print(y_test.head(), len(y_test))

# Split train into train1 and val1

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state
= 1, test_size = 0.15)

print(X_train1.head(), len(X_train1))

print(X_val1.head(), len(X_val1))

print(y_train1.head(), len(y_train1))

print(y_val1.head(), len(y_val1))



# define the model
model = AdaBoostClassifier()
# define the space of hyperparameters to search
```

```
search_space = [Integer(5, 200, name='n_estimators'), Real(0.1, 5.0,
name='learning_rate')]

# define the function used to evaluate a given configuration
@use_named_args(search_space)
def evaluate_model(**params):
        # something
        model.set_params(**params)
        # calculate 5-fold cross validation
        result = cross_val_score(model, X_train, y_train, cv=5, n_jobs=-1,
scoring='accuracy')
        # calculate the mean of the scores
        estimate = mean(result)
        return 1.0 - estimate

# perform optimization
result = gp_minimize(evaluate_model, search_space,  n_calls=25)

print(result.items())


# summarizing finding:
print('Best Accuracy: %.3f' % (1.0 - result.fun))
print('Best Parameters: n_estimators = %d, learning_rate=%.3f' % (result.x[0],
result.x[1]))

plot_convergence(result)
```

```
# example of bayesian optimization with scikit-optimize
from numpy import mean
from sklearn.datasets import make_blobs
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from skopt.space import Integer
from skopt.space import Categorical
from skopt.utils import use_named_args
from skopt import gp_minimize
from skopt.plots import plot_convergence
from skopt.plots import plot_gaussian_process

import pandas as pd

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

filename = 'initialmodel2.csv'

df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

df1 = df.pop('Defaulted')

df['Defaulted'] = df1

df.drop(['Unnamed: 0'] , axis = 1, inplace =True)

df = df.dropna()

#df.drop(['FirstPaymentDate', 'LastPaymentOn'], axis = 1, inplace =True)

print(df.head())
```

```
# Split dataframe into X and y

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

Y = df.iloc[:,-1].astype(int)

# Split into train and test

X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state = 1,
test_size = 0.2)

print(X_train.head(), len(X_train))

print(X_test.head(), len(X_test))

print(y_train.head(), len(y_train))

print(y_test.head(), len(y_test))

# Split train into train1 and val1

X_train1, X_val1, y_train1, y_val1 = train_test_split(X_train, y_train, random_state
= 1, test_size = 0.15)

print(X_train1.head(), len(X_train1))

print(X_val1.head(), len(X_val1))

print(y_train1.head(), len(y_train1))

print(y_val1.head(), len(y_val1))


# define the model
model = DecisionTreeClassifier()
# define the space of hyperparameters to search
search_space = [Integer(5, 20, name='max_depth'), Integer(2, 5,
name='min_samples_split'),
               Categorical(["gini","entropy"],name="criterion")]

# define the function used to evaluate a given configuration
@use_named_args(search_space)
def evaluate_model(**params):
        # something
        model.set_params(**params)
        # calculate 5-fold cross validation
        result = cross_val_score(model, X_train, y_train, cv=5, n_jobs=-1,
scoring='accuracy')
        # calculate the mean of the scores
```

```
        estimate = mean(result)
        return 1.0 - estimate

# perform optimization
result = gp_minimize(evaluate_model, search_space, n_calls=10)

print(result.items())


# summarizing finding:
print('Best Accuracy: %.3f' % (1.0 - result.fun))
print('Best Parameters: criterion = %s, max_depth=%d, min_samples_split=%d' %
(result.x[2], result.x[0], result.x[1]))

plot_convergence(result)

print(result)
```

K MEANS CLUSTERING

```
# -*- coding: utf-8 -*-
"""
Created on Wed Jun  8 16:09:20 2022

@author: ramra
"""
# Next series of python files
# present the model coding in python

# KS MEANS

import pandas as pd
import numpy as np
import psycopg2
import csv
from random import seed
from csv import reader
import random
import matplotlib.pyplot as plt
from math import exp
from math import pi
from math import sqrt
from random import random
import seaborn as sns
import csv
from matplotlib.ticker import FormatStrFormatter

#sklearn Imports

import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.cluster import KMeans

filename = 'initialmodel2-1.csv'
```

```
df = pd.read_csv(
        filename, on_bad_lines="skip", engine="python"
    )

df1 = df.pop('Defaulted')

df['Defaulted'] = df1

df.drop(['Unnamed: 0'] , axis = 1, inplace =True)

df = df.dropna()

new_df = df

#df.drop(['FirstPaymentDate', 'LastPaymentOn'], axis = 1, inplace =True)

print(df.head())

# Split dataframe into X and y

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

Y = df.iloc[:,-1].astype(int)

WCSS = []
K = range(1,13)
for k in K:
    kmeanModel = KMeans(n_clusters=k)
    kmeanModel.fit(X)
    WCSS.append(kmeanModel.inertia_)

plt.figure(figsize=(8,8))

plt.rc('font', size = 20 )

plt.plot(K, WCSS, 'bx-')
plt.xlim(0,15)
plt.xlabel('No. of Clusters')
plt.ylabel('WCSS')
plt.title('The Elbow Method showing the optimal clusters')
plt.show()

kmeans_model = KMeans(n_clusters=6, random_state=42)

kmeans_predict = kmeans_model.fit_predict(X)

centroids = kmeans_model.cluster_centers_

print(type(centroids))
```

```
vals  = []

for i in range(len(centroids[0])):
   max1 = 0
   min1 = 10000
   for j in range(centroids.shape[0]):
       if centroids[j][i] >max1:
           max1 = centroids[j][i]
       if centroids[j][i] <min1:
           min1 = centroids[j][i]

   vals.append((max1, min1))


spread = []

for i in range(len(vals)):
    spread.append(vals[i][0] -vals[i][1])


print(spread)

r = pd.DataFrame( { 'spread': spread, 'names': X.columns})

r = r.sort_values(by=['spread'], ascending = False)
r.to_csv("spread.csv")

print(type(kmeans_predict))

print(kmeans_predict)

new_df['pred'] = kmeans_predict

print(new_df.head())

df1 = new_df[new_df["pred"] == 0]
df2 = new_df[new_df["pred"] == 1]
df3 = new_df[new_df["pred"] == 2]
df4 = new_df[new_df["pred"] == 3]
df5 = new_df[new_df["pred"] == 4]
df6 = new_df[new_df["pred"] == 5]

plt.rc('font', size = 20 )

plt.scatter(df6.PrincipalOverdueBySchedule, df6.ProbabilityOfDefault, c = 'gray',
label = '1', alpha = 0.5)
plt.scatter(df1.PrincipalOverdueBySchedule, df1.ProbabilityOfDefault, c = 'blue',
label = '2', alpha = 0.5)
plt.scatter(df2.PrincipalOverdueBySchedule, df2.ProbabilityOfDefault, c = 'green',
label = '3',  alpha = 0.5)
```

```
plt.scatter(df3.PrincipalOverdueBySchedule, df3.ProbabilityOfDefault, c = 'orange',
label = '4', alpha = 0.5)
plt.scatter(df4.PrincipalOverdueBySchedule, df4.ProbabilityOfDefault, c = 'red',
label = '5', alpha = 0.5)
plt.scatter(df5.PrincipalOverdueBySchedule, df5.ProbabilityOfDefault, c = 'yellow',
label = '6', alpha = 0.5)
plt.xlabel("PrincipalOverdueBySchedule")
plt.ylabel('ProbabilityOfDefault')
plt.title('K Means Visualization')
plt.legend()

plt.show()
```