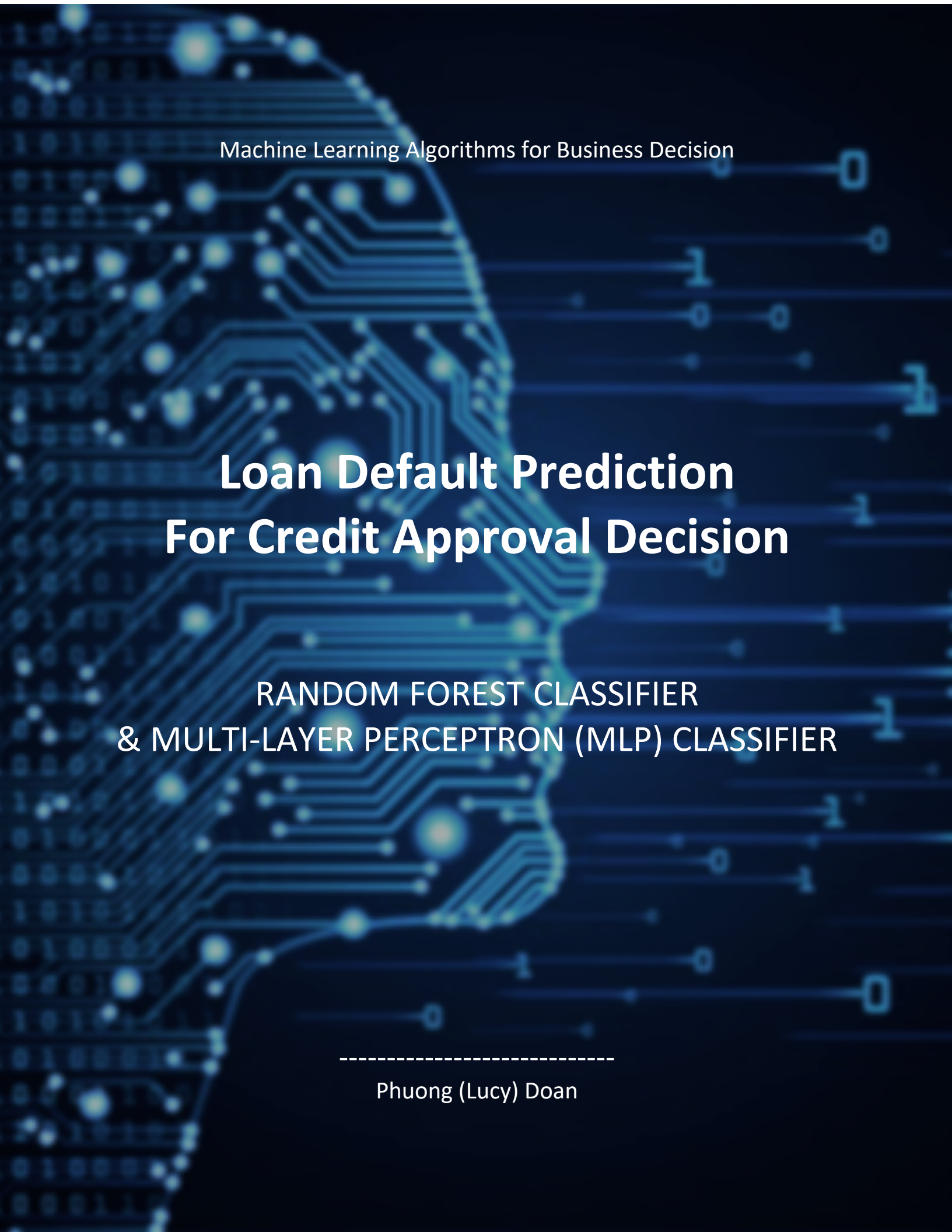


Machine Learning Algorithms for Business Decision

Loan Default Prediction For Credit Approval Decision

RANDOM FOREST CLASSIFIER & MULTI-LAYER PERCEPTRON (MLP) CLASSIFIER

Phuong (Lucy) Doan

Table of Contents

I. Problem Description

1.1. Business Objective

Financial institutions, such as banks, lend money to qualified borrowers and expect them to repay the loan amount in installments by a certain period of time. Irrespective of the size of the bank, loans contribute a major portion of the bank's total asset. For large commercial banks in the United States, such as Wells Fargo, JP Morgan and Capital One, loans contribute more than a third of their total assets. Therefore, a large percentage of funds deposited to the banks are used to issue different types of loans (e.g., credit card loans, mortgage loans, auto loans) and earn a profit by collecting interest on the loaned amount. However, there is also a huge risk involved because a percentage of the total approved loans may result in bad loans (debts that are not recovered on time). Bad loans are one of the most common causes of bank revenue losses and sometimes bankruptcy. Many national and international banks suffered huge losses due to late payment or loan default. As a result, the financial stability of the banks is at risk. In the late 2000's, banks issued a large sum of money towards home and personal loans without considering the risk of bad loans. This was one of the major drives of the financial crisis, also the US recession, in 2009 and could have been avoided if the banks had carefully chosen their borrowers. Nowadays, many banks invest largely into risk management or credit appraisal to obtain better, scientific and data-driven input for lending decisions.

1.2. Data Mining Objective

The objective of this case study is analyzing existing loan applicant data to predict risk type (1 = high risk, 0 = low risk) of forthcoming applicants using machine learning algorithms. Moreover, the data used in this case is a real-world dataset obtained from a bank in the US. The raw data contains 100,000 samples. A detailed description of the data is as following.

Variable Name	Description	Type
Age	Age of the borrower (in years)	Integer
Debt Ratio	The ratio of monthly debt payments to monthly gross income	Continuous between 0 and 1
LOC	Number of open loans and lines of credit	Integer
Income	Monthly income of the borrower	Continuous
MREL	Number of mortgage and real estate loans	Integer
Dependents	Number of dependents of the borrower	Integer
Utilization	The ratio of total balance on lines of credit to the total credit limits	Continuous between 0 and 1
30Day	Number of times the borrower has been 30-59 days past the due date in the last two years	Integer
60Day	Number of times the borrower has been 60-89 days past the due date in the last two years	Integer
90Day	Number of times the borrower has been equal to or more than 90 days past the due date	Integer
Risk	The risk associated with the borrower	Binary

II. Analytical Questions

2.1. Variable and problem identification

- **Variable Identification**

The dependent variable is Risk, and independent variables are 10 remaining variables, including ones about customer's demographics (Age, Income, Dependents), their financial health (Debt Ratio, MREL), and credit-using behaviors (LOC, Utilization, 30Day, 60Day, 90Day).

- **Problem Identification**

This is a supervised learning – binary classification problem, applying machine learning algorithm to predict the probability of 2 output classes, low risk and high risk of forthcoming credit customers.

2.2. Data Cleaning

- **Handling User Input Errors**

  - Checking column names

Besides grammatical mistakes, sometimes column names, after being read, have blank spaces that may make data extraction a hassle.

```
#Check column names for any typos
LoanData.columns
```

```
Index(['Age', 'DebtRatio', 'LOC', 'Income', 'MREL', 'Dependents ',
       'Utilization', '30Day', '60Day', '90Day', 'Risk'],
      dtype='object')
```

```
#Delete blank spaces in column names
LoanData = LoanData.rename(columns={'Dependents ':'Dependents'})
```

```
LoanData.columns
```

```
Index(['Age', 'DebtRatio', 'LOC', 'Income', 'MREL', 'Dependents',
       'Utilization', '30Day', '60Day', '90Day', 'Risk'],
      dtype='object')
```

After right-trimming the name of Dependents variable, now all our variable names are ready for analysis

  - Handling input errors in data types

Each business domain has its own rules of metrics that data collected must comply with. Specifically, in this financial banking case, the debt ratio and utilization must be continuous and bounded between 0 and 1. Also, the count variables, including Age, ROC, MREL, Dependents, 30Day, 60Day, 90Day, should be in integer data type.

```
#Data summary
LoanData.describe()
```

	Age	DebtRatio	LOC	Income	MREL	Dependents	Utilization	30Day	60Day	90Day	10
count	100000.000000	100000.000000	100000.000000	8.009800e+04	100000.000000	97416.000000	100000.000000	100000.000000	100000.00000	100000.000000	
mean	52.279570	359.624790	8.424920	6.689975e+03	1.016750	0.760358	6.115302	0.418810	0.23807	0.265630	
std	14.742859	2339.815864	5.121764	1.637884e+04	1.127538	1.117592	255.249894	4.178854	4.14121	4.156193	
min	0.000000	0.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	
25%	41.000000	0.175075	5.000000	3.400000e+03	0.000000	0.000000	0.030106	0.000000	0.00000	0.000000	
50%	52.000000	0.366529	8.000000	5.400000e+03	1.000000	0.000000	0.154768	0.000000	0.00000	0.000000	
75%	63.000000	0.874209	11.000000	8.264750e+03	2.000000	1.000000	0.560238	0.000000	0.00000	0.000000	
max	109.000000	329664.000000	58.000000	3.008750e+06	54.000000	20.000000	50708.000000	98.000000	98.00000	98.000000	

Since maximum values of Debt Ratio and Utilization variables are bigger than 1 while they should not, there must be input errors. To deal with this issue, the author chooses to convert these bigger-than-1 values into null values.

```
#Convert any bigger-than-1 value of DebtRatio and Utilization variables into null values
for i in range(0,len(LoanData)):
    for f in ['DebtRatio','Utilization']:
        if LoanData[f][i] > 1:
            LoanData[f][i] = np.nan

LoanData.describe()
```

	Age	DebtRatio	LOC	Income	MREL	Dependents	Utilization	30Day	60Day	90Day	10
count	100000.000000	76507.000000	100000.000000	8.009800e+04	100000.000000	97416.000000	97807.000000	100000.000000	100000.00000	100000.000000	
mean	52.279570	0.302661	8.424920	6.689975e+03	1.016750	0.760358	0.304342	0.418810	0.23807	0.265630	
std	14.742859	0.226218	5.121764	1.637884e+04	1.127538	1.117592	0.338206	4.178854	4.14121	4.156193	
min	0.000000	0.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	
25%	41.000000	0.125853	5.000000	3.400000e+03	0.000000	0.000000	0.028822	0.000000	0.00000	0.000000	
50%	52.000000	0.273918	8.000000	5.400000e+03	1.000000	0.000000	0.144825	0.000000	0.00000	0.000000	
75%	63.000000	0.437171	11.000000	8.264750e+03	2.000000	1.000000	0.521300	0.000000	0.00000	0.000000	
max	109.000000	1.000000	58.000000	3.008750e+06	54.000000	20.000000	1.000000	98.000000	98.00000	98.000000	

Now let's check data types.

```
#Check data types
LoanData.dtypes
```

```
Age               int64
DebtRatio       float64
LOC               int64
Income          float64
MREL              int64
Dependents      float64
Utilization     float64
30Day             int64
60Day             int64
90Day             int64
Risk              int64
dtype: object
```

As shown above, Dependents variable is identified as float type while it should be integer. We could doubt that there is decimal values in this variable. Let's check it.

```
#Since Dependents variable should be integer, let's check if there is any decimal value in this variable
a=0
b=0
for i in range(0,len(LoanData)):
    if np.isnan(LoanData['Dependents'][i]):
        b+=1
    elif LoanData['Dependents'][i] %1 != 0:
        a+=1
print("There are", a, "decimal values and", b, "null values")
```

```
There are 0 decimal values and 2584 null values
```

There is no decimal value in Dependent variable; however, with null values we cannot convert the variable into integer at this point. We will do that after handling missing values.

- **Handling Missing Values**

```
#Count missing values in each variable
LoanData.isnull().sum()
```

```
Age                0
DebtRatio      23493
LOC                0
Income         19902
MREL               0
Dependents      2584
Utilization     2193
30Day              0
60Day              0
90Day              0
Risk               0
dtype: int64
```

Null values show up in 4 variables, DebtRatio, Income, Dependents and Utilization. In order to deal with missing values, this report will implement 2 methods, discarding incomplete rows (i), and discarding variable containing more than 10% missing values and substituting remaining missing values with the median of that variable (ii), resulting in 2 cleaned datasets, LoanData1 and LoanData2, respectively.

  - Alternative #1: Discard incomplete rows

```
#Discard incomplete rows to create the first cleaned dataset named "LoanData1"
LoanData1 = LoanData.dropna()
```

```
LoanData1.isnull().sum()
```

```
Age            0
DebtRatio      0
LOC            0
Income         0
MREL           0
Dependents     0
Utilization    0
30Day          0
60Day          0
90Day          0
Risk           0
dtype: int64
```

Now that every variable gets rid of missing values, it's time for converting data type of Dependents variable from float to integer.

```
#Change Dependents variable from float type to integer type
LoanData1["Dependents"] = LoanData1["Dependents"].astype("int64")
```

```
LoanData1.dtypes
```

```
Age              int64
DebtRatio      float64
LOC              int64
Income         float64
MREL             int64
Dependents       int64
Utilization    float64
30Day            int64
60Day            int64
90Day            int64
Risk             int64
dtype: object
```

- Alternative #2: Discard variable containing more than 10% missing values and substitute remaining missing values with the median of that variable

First, obtain the LoanData2 dataset by copying the LoanData dataset. Then, we identify the percentage accounted by missing values in each variable

```
#Calculate the percentage of missing values in each variable
print("The Debt Ratio variable contains",
      round((LoanData2.Age.count() - LoanData2.DebtRatio.count())/ LoanData2.Age.count() *100),
      "%","missing values")
print("The Income variable contains",
      round((LoanData2.Age.count() - LoanData2.Income.count())/ LoanData2.Age.count() *100),
      "%","missing values")
print("The Dependents variable contains",
      round((LoanData2.Age.count() - LoanData2.Dependents.count())/ LoanData2.Age.count() *100),
      "%","missing values")
print("The Utilization variable contains",
      round((LoanData2.Age.count() - LoanData2.Utilization.count())/ LoanData2.Age.count() *100),
      "%","missing values")
```

```
The Debt Ratio variable contains 23.0 % missing values
The Income variable contains 20.0 % missing values
The Dependents variable contains 3.0 % missing values
The Utilization variable contains 2.0 % missing values
```

On the one hand, since Debt Ratio and Income variables contain more than 10% missing values, they will be discarded.

```
#Delete the Debt Ratio and Income variable because it has more than 10% missing values
LoanData2 = LoanData2.drop(["DebtRatio","Income"], axis=1)
```

On the other hand, Dependents and Utilization variables have lower than 10% missing values; therefore, these missing values will be replaced with median of their variables.

```
#For Dependents and Utilization variables, impute missing values with their median value
LoanData2["Dependents"] = LoanData2.fillna(LoanData2["Dependents"].median())
LoanData2["Utilization"] = LoanData2.fillna(LoanData2["Utilization"].median())
```

Now the LoanData2 is completely free of missing values. As a side note, final LoanData2 has 8 independent variables, which is 2 variables fewer than LoanData1.

```
LoanData2.isnull().sum()
```

```
Age            0
LOC            0
MREL           0
Dependents     0
Utilization    0
30Day          0
60Day          0
90Day          0
Risk           0
dtype: int64
```

Last but not least, we convert Dependents variable's data type into integer.

```
#Change Dependents variable from float type to integer type
LoanData2["Dependents"] = LoanData2["Dependents"].astype("int64")
```

```
LoanData2.dtypes
```

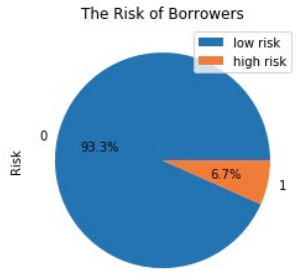
```
Age              int64
LOC              int64
MREL             int64
Dependents       int64
Utilization    float64
30Day            int64
60Day            int64
90Day            int64
Risk             int64
dtype: object
```

- **Handling Imbalanced Dataset**

Imbalance happens when one class of dependent variable extremely outnumbers the other classes of that variable. Imbalanced dataset can affect algorithm learning and produce misleading results.

```
#The pie chart shows imbalanced dataset with a ratio of 1(high risk):15(low risk)
LoanData2.Risk.value_counts().plot.pie(autopct='%1.1f%%', title="The Risk of Borrowers")
plt.legend(("low risk","high risk"))
```

```
<matplotlib.legend.Legend at 0x1a453c23d0>
```



The LoanData2 dataset has extremely imbalanced dependent variable with 93% classified as "high risk" (coded 0). With this situation, the machine learning algorithm can classify all the rows as 0 (without learning any useful information) and still get an accuracy rate at 93%. To overcome this drawback of imbalanced dataset, the report's author will employ oversampling method, using SMOTE - Synthetic Minority Oversampling Technique, to increase the number 1 in the 'Risk' column to achieve a 1:5 ratio – For every 1 high risk, there will be 5 low risk. The dataset cleaned this way is LoanData3. The reason to choose

oversampling, instead of downsampling, is that downsampling could result in losing some important characteristics/ features of the original sample.

```
#Upsampling, using SMOTE library, by increasing the minority class (1 - high risk)
X = pd.DataFrame(LoanData2.iloc[:,:-1])
y = pd.DataFrame(LoanData2.iloc[:,-1])

sm = SMOTE(random_state=8810, sampling_strategy = 0.2) #Ratio of minority:majority = 1:5
X_res, y_res = sm.fit_sample(X,y)
```

```
#Check the ratio after resampling to see if we obtain the desired ratio
print("For every 1 high risk, there will be",
      len(y_res[y_res.Risk == 0]) / len(y_res[y_res.Risk == 1]), "low risk")
```

```
For every 1 high risk, there will be 5.0 low risk
```

After achieving the desired ratio, we combine the dependent variable and independent variables parts to obtain LoanData3. Note that, before oversampling, LoanData3 was copied from LoanData2, thus having no missing value.

```
#Create LoanData3 by merging horizontally 2 dataframes X_res and y_res
LoanData3 = pd.concat([X_res,y_res], axis=1)
```

```
LoanData3.isnull().sum()
```

```
Age            0
LOC            0
MREL           0
Dependents     0
Utilization    0
30Day          0
60Day          0
90Day          0
Risk           0
dtype: int64
```

2.3. Splitting Data into Training and Testing Datasets

```
#For LoanData1 dataset
X1 = pd.DataFrame(LoanData1.iloc[:,:-1])
y1 = pd.DataFrame(LoanData1.iloc[:,-1])

X_train1, X_test1, y_train1, y_test1=train_test_split(X1,y1, test_size=0.3, random_state=8810)
```

```
#For LoanData2 dataset
X2 = pd.DataFrame(LoanData2.iloc[:,:-1])
y2 = pd.DataFrame(LoanData2.iloc[:,-1])

X_train2, X_test2, y_train2, y_test2=train_test_split(X2,y2, test_size=0.3, random_state=8810)
```

```
#For LoanData3 dataset
X3 = pd.DataFrame(LoanData3.iloc[:,:-1])
y3 = pd.DataFrame(LoanData3.iloc[:,-1])

X_train3, X_test3, y_train3, y_test3=train_test_split(X3,y3, test_size=0.3, random_state=8810)
```

By this way, each of three datasets is split into training and testing datasets with the 7:3 ratio – 70% training data and 30% testing data.
LoanData1 → X_train1, X_test1, y_train1, y_test1
LoanData2 → X_train2, X_test2, y_train2, y_test2
LoanData3 → X_train3, X_test3, y_train3, y_test3

2.4.Learning from Datasets and Making Predictions

- **Learning from Datasets**

  - Random Forest Classifier

Initialize the Random Forest Classifier then let the model learn from 3 training datasets

```
#For LoanData1 dataset
RF1_class = RandomForestClassifier(n_estimators=50)
RF1_class = RF1_class.fit(X_train1, y_train1)

#For LoanData2 dataset
RF2_class = RandomForestClassifier(n_estimators=50)
RF2_class = RF2_class.fit(X_train2, y_train2)

#For LoanData3 dataset
RF3_class = RandomForestClassifier(n_estimators=50)
RF3_class = RF3_class.fit(X_train3, y_train3)
```

  - Multi-Layer Perceptron (MLP) Classifier

MLP Classifier algorithm requires us to identify some parameters that will affect the accuracy of the predicting model. Among them, the number of hidden layers and the number of nodes in each hidden layer should draw our attention.

In his research, Huang (2003) found out that in the two-hidden-layer case, with m output neurons, the number of hidden nodes that are enough to learn N samples with negligibly small error is given by:

$$2\sqrt{(m+2)N} \qquad (1)$$

The number of nodes in the first hidden layer is:

$$\sqrt{N(m+2)} + 2\sqrt{N/(m+2)} \qquad (2)$$

And the number of nodes in the second hidden layer will be:

$$m\sqrt{N/(m+2)} \qquad (3)$$

In another research, Berry and Linoff suggested (1997) "How large should the hidden layer be? One rule of thumb is that it should never be more than twice as large as the input layer."

In this case, there are one input neuron, at least 8 output neurons and more than 70,000 observations for each testing dataset. Therefore, the author will consider ether one or two hidden layers; It should exceed 3 layers to avoid complexity. Regarding the number of nodes in each hidden layer, on the one hand, applying the formulas (2) and (3), we will have 5 and 1 nodes in the first and second hidden layers accordingly; on the other hand, following Berry and Linoff's advice will result in 20 nodes at most. For these reasons, the

author decides to take into account the ranges of 1-20 node(s) and 1-5 node(s) for the first and second hidden layers, respectively.

We will conduct "trial and error" method to find out the local optimum option, which results in the best accuracy score among all options tested. Refer to Appendix 1 for the Accuracy table of this method.

```
#Determine number of hidden layers and number of hidden nodes in each layer, based on Accuracy Score, using LoanData1
df = pd.DataFrame(columns=["firstLayer_N","2ndLayer_0N","2ndLayer_1N","2ndLayer_2N","2ndLayer_3N","2ndLayer_4N"
                           ,"2ndLayer_5N"])

for i in range(7):           #i is the number of nodes in second hidden layer
    for j in range(1,21):    #j is the number of nodes in first hidden layer
        if i == 0:
            df[df.columns[i]] = range(1,21)
        elif i == 1:
            MLP1_class = MLPClassifier(hidden_layer_sizes=(j), solver='lbfgs', alpha=1e-05, random_state=8810
                                       , max_iter=500).fit(X_train1, y_train1)
            y_pred_MLP1 = MLP1_class.predict(X_test1)
            Accuracy_MLP1 = metrics.accuracy_score(y_test1, y_pred_MLP1)
            df[df.columns[i]][j-1] = Accuracy_MLP1
        else:
            MLP1_class = MLPClassifier(hidden_layer_sizes=(j,i), solver='lbfgs', alpha=1e-05, random_state=8810
                                       , max_iter=500).fit(X_train1, y_train1)
            y_pred_MLP1 = MLP1_class.predict(X_test1)
            Accuracy_MLP1 = metrics.accuracy_score(y_test1, y_pred_MLP1)
            df[df.columns[i]][j-1] = Accuracy_MLP1
```

Based on the Accuracy table, the local maximum score, 0.93874, shows up in a few pairs, one of which is (5,2). This is close to the numbers resulted from Huang's formulas.

After picking the option of 2 hidden layers, with 5 nodes and 2 nodes on each layer, we initialize the MLP Classifier then let the model learn from 3 training datasets.

```
#Train 3 datasets with the MLP Classifier

#For LoanData1 dataset
MLP1_class = MLPClassifier(hidden_layer_sizes=(5,2), solver='lbfgs', alpha=1e-05, random_state=8810, max_iter=500)
MLP1_class = MLP1_class.fit(X_train1, y_train1)

#For LoanData2 dataset
MLP2_class = MLPClassifier(hidden_layer_sizes=(5,2), solver='lbfgs', alpha=1e-05, random_state=8810, max_iter=500)
MLP2_class = MLP2_class.fit(X_train2, y_train2)

#For LoanData3 dataset
MLP3_class = MLPClassifier(hidden_layer_sizes=(5,2), solver='lbfgs', alpha=1e-05, random_state=8810, max_iter=500)
MLP3_class = MLP3_class.fit(X_train3, y_train3)
```

- **Making Predictions using two algorithms on all 3 testing datasets**

```
y_pred_RF1 = RF1_class.predict(X_test1)
y_pred_MLP1 = MLP1_class.predict(X_test1)

y_pred_RF2 = RF2_class.predict(X_test2)
y_pred_MLP2 = MLP2_class.predict(X_test2)

y_pred_RF3 = RF3_class.predict(X_test3)
y_pred_MLP3 = MLP3_class.predict(X_test3)
```

2.5.Evaluating Random Forest and MLP Classification models

- **Confusion Matrix**

```
cm_RF1 = confusion_matrix(y_test1, y_pred_RF1)
cm_MLP1 = confusion_matrix(y_test1, y_pred_MLP1)

cm_RF2 = confusion_matrix(y_test2, y_pred_RF2)
cm_MLP2 = confusion_matrix(y_test2, y_pred_MLP2)

cm_RF3 = confusion_matrix(y_test3, y_pred_RF3)
cm_MLP3 = confusion_matrix(y_test3, y_pred_MLP3)
```

Please refer to Appendix 2 for 6 confusion matrices of 2 algorithms on 3 datasets.

- **Classification Report**

```
cr_RF1 = pd.DataFrame(metrics.classification_report(y_test1, y_pred_RF1, output_dict=True))
cr_MLP1 = pd.DataFrame(metrics.classification_report(y_test1, y_pred_MLP1, output_dict=True))

cr_RF2 = pd.DataFrame(metrics.classification_report(y_test2, y_pred_RF2, output_dict=True))
cr_MLP2 = pd.DataFrame(metrics.classification_report(y_test2, y_pred_MLP2, output_dict=True))

cr_RF3 = pd.DataFrame(metrics.classification_report(y_test3, y_pred_RF3, output_dict=True))
cr_MLP3 = pd.DataFrame(metrics.classification_report(y_test3, y_pred_MLP3, output_dict=True))
```

Classification report provides us with 4 performance measurements, precision, recall, f-1 score and accuracy score. Please refer to Appendix 3 for 6 classification reports.

- **Performance Measurement Table**

For convenient comparison, we gather all performance measurements of 3 approaches (technique i, technique ii or technique ii + SMOTE) and 2 algorithms. This is a crosstab with 6 columns, RF1, RF2, RF3, MLP1, MLP2, MLP3, and 4 rows of measurements.

First, extract 3 measures, precision, recall and f-1 score, from classification reports, then populate them into a table. Note that the author chooses macro averages, instead of weighted averages, because the predicting variable, credit risk, in first and second datasets, are highly skewed towards low risks, thus distorting weighted average measures.

```
#Extract Precision, Recall and f-1 score from classification report of each algorithm and data cleaning approach
measure_RF1 = pd.DataFrame(cr_RF1.iloc[:-1,3])
measure_RF2 = pd.DataFrame(cr_RF2.iloc[:-1,3])
measure_RF3 = pd.DataFrame(cr_RF3.iloc[:-1,3])
measure_MLP1 = pd.DataFrame(cr_MLP1.iloc[:-1,3])
measure_MLP2 = pd.DataFrame(cr_MLP2.iloc[:-1,3])
measure_MLP3 = pd.DataFrame(cr_MLP3.iloc[:-1,3])
```

```
#Combine them into a table
measure_table = pd.concat([measure_RF1,measure_RF2,measure_RF3,measure_MLP1,measure_MLP2,measure_MLP3], axis=1)
measure_table.columns=["RF1","RF2","RF3","MLP1","MLP2","MLP3"]
measure_table
```

	RF1	RF2	RF3	MLP1	MLP2	MLP3
precision	0.737085	0.665736	0.774561	0.505091	0.032667	0.084675
recall	0.561991	0.588636	0.699736	0.500112	0.500000	0.500000
f1-score	0.589687	0.612253	0.726946	0.485359	0.061327	0.144824

Similarly, extract accuracy score from classification reports and populate them into another table.

```
#Extract Accuracy score from classification report of each algorithm and data cleaning approach
accuracy=(cr_RF1.iloc[0,2],cr_RF2.iloc[0,2], cr_RF3.iloc[0,2], cr_MLP1.iloc[0,2], cr_MLP2.iloc[0,2], cr_MLP3.iloc[0,2])
accuracy = pd.DataFrame(accuracy, columns=["accuracy"]).T
accuracy.columns=["RF1","RF2","RF3","MLP1","MLP2","MLP3"]
accuracy
```

	RF1	RF2	RF3	MLP1	MLP2	MLP3
accuracy	0.939601	0.926933	0.866508	0.937653	0.065333	0.16935

Afterwards, combine two tables vertically to obtain a 4-measure table.

```
#Combine the above 2 tables to have 4 performance measures (Precision, Recall, f-1 score and accuracy score)
fourmeasure_table = measure_table.append(accuracy, sort=False)
fourmeasure_table
```

	RF1	RF2	RF3	MLP1	MLP2	MLP3
precision	0.761509	0.678089	0.777438	0.505091	0.032667	0.084675
recall	0.561976	0.595423	0.701694	0.500112	0.500000	0.500000
f1-score	0.590857	0.620986	0.729249	0.485359	0.061327	0.144824
accuracy	0.940870	0.928533	0.867669	0.937653	0.065333	0.169350

Finally, we calculate 6 specificity scores.

```
#Calculate Specificity measure from Confusion matrix
Specificity_RF1 = cm_RF1[1,1]/(cm_RF1[1,0] + cm_RF1[1,1])
Specificity_RF2 = cm_RF2[1,1]/(cm_RF2[1,0] + cm_RF2[1,1])
Specificity_RF3 = cm_RF3[1,1]/(cm_RF3[1,0] + cm_RF3[1,1])
Specificity_MLP1 = cm_MLP1[1,1]/(cm_MLP1[1,0] + cm_MLP1[1,1])
Specificity_MLP2 = cm_MLP2[1,1]/(cm_MLP2[1,0] + cm_MLP2[1,1])
Specificity_MLP3 = cm_MLP3[1,1]/(cm_MLP3[1,0] + cm_MLP3[1,1])

print("Specificity score of RF1 is ", Specificity_RF1)
print("Specificity score of RF2 is ", Specificity_RF2)
print("Specificity score of RF3 is ", Specificity_RF3)
print("Specificity score of MLP1 is ", Specificity_MLP1)
print("Specificity score of MLP2 is ", Specificity_MLP2)
print("Specificity score of MLP3 is ", Specificity_MLP3)
```

```
Specificity score of RF1 is  0.1301775147928994
Specificity score of RF2 is  0.21224489795918366
Specificity score of RF3 is  0.4507116499736426
Specificity score of MLP1 is  0.0014792899408284023
Specificity score of MLP2 is  1.0
Specificity score of MLP3 is  1.0
```

Then we gather all 6 specificity scores into one table called “specificity”

```
#Populate all Specificity scores of each algorithm and data cleaning approach into one table
specificity =(Specificity_RF1, Specificity_RF2, Specificity_RF3, Specificity_MLP1, Specificity_MLP2, Specificity_MLP3)
specificity = pd.DataFrame(specificity, columns=["specificity"]).T
specificity.columns=["RF1","RF2","RF3","MLP1","MLP2","MLP3"]
specificity
```

	RF1	RF2	RF3	MLP1	MLP2	MLP3
specificity	0.130178	0.212245	0.450712	0.001479	1.0	1.0

Finally, we combine specificity table to the above 4-measure table to obtain a complete performance measurement table as below:

```
#Combine the above 4-measure table with the specificity table to have all 5 performance measures
Measure_Table = fourmeasure_table.append(specificity, sort=False)
Measure_Table
```

	RF1	RF2	RF3	MLP1	MLP2	MLP3
precision	0.761509	0.678089	0.777438	0.505091	0.032667	0.084675
recall	0.561976	0.595423	0.701694	0.500112	0.500000	0.500000
f1-score	0.590857	0.620986	0.729249	0.485359	0.061327	0.144824
accuracy	0.940870	0.928533	0.867669	0.937653	0.065333	0.169350
specificity	0.130178	0.212245	0.450712	0.001479	1.000000	1.000000

As shown above, regarding Random Forest Classifier, LoanData3 dataset produces the best performance measurements except for accuracy score. Although LoanData1 has the highest accuracy score, it is closely followed by those of the LoanData2 then LoanData3 datasets. Look closely into confusion matrix on Appendix 2, we can explain that LoanData1 has better accuracy score because its random forest model predicts very few false negatives compared to the other datasets.

For MLP Classifier, LoanData1 dataset performs the best on 4 out of 5 measurements, which is understandable because this dataset was used for "trial-and-error" to pick the optimum number of hidden layers and number of nodes on each hidden layer. However, MLP Classifier models on LoanData2 and LoanData3 result in maximum specificity scores at 1, because they predict all credit risk are high risks (class 1).

In a comparison between Random Forest Classifier and MLP Classifier, generally Random Forest Classifier performs better in 4 out of 5 measurements. The only measures that MLP Classifier is superior is specificity on LoanData2 and LoanData3 datasets

- **Variable Importance Plot**

Please refer to Appendix 4 for reference on variable importance plots of random forest classification model on 3 datasets.

In a quick look, top 3 most important variables of LoanData2 and LoanData3 are the same while they are totally different from those from LoanData1. An explanation is that 2 out of 3 most important variables in LoanData1, Income and Debt Ratio, are already removed from the other two datasets. Besides that, Age and Utilization, which is the ratio of total balance on lines of credit to the total credit limits, also fall out of top importance in LoanData2 and LoanData3 dataset while both rank top in LoanData1 dataset. In the author's opinion, the reason could be all 4 variables, Income, Debt Ratio, Age and Utilization are correlated to each other. For example, the higher income a person has, the lower debt ratio (the ratio of monthly debt payments to monthly gross income) that person has. Therefore, the presence of one variable may increase the importance of its related variables. Finally, LOC (the number of open loans and lines of credit), despite ranked 5th in sample 1, is number one in the samples 2 and 3.

III. Conclusions and Recommendations

3.1. Conclusions

We have 6 models in total, created from 2 algorithms, Random Forest classifier and MLP classifier, on 3 datasets, LoanData1, LoanData2 and LoanData3. Overall, Random Forest model on LoanData3 could be regarded as the best because it results in highest precision, recall and f-1 scores, while its accuracy score follows very closely to the first and second ranks. What makes difference between the samples 2-3 and sample 1 is they keep the rows having missing values. By discarding the rows of missing values, sample 1 (LoanData1 dataset) could lose some importance characteristics presented in these rows.

In a thought about two algorithms, Random Forest Classifier performs much better than MLP Classifier in almost all of the performance measurements, except for specificity. This is reasonable because when doing "trial-and-error" to select hidden layer's number and node's numbers, only LoanData1 dataset was used. It also explains why MLP Classifier performs better on LoanData1 than the other two datasets.

For variable importance resulted from Random Forest Classifier, the LoanData1 gives more weights on borrower's demographical information, age and income, while LoanData2 and LoanData3 focus more on borrower's behaviors of using credits.

3.2. Recommendations

Most of the performance metrics prove that LoanData2 and LoanData3 datasets have more advantages than LoanData1 because they do not remove the rows having missing values, thus not losing important characteristics. Especially, the data sample 3 reduces the classification imbalance in predicting variable, by oversampling, resulting in better performance than the other two samples. The data preparation method applied on sample 3 could be considered for other analysis on datasets with similar domain or features.

For algorithms selection, it is early to conclude that Random Forest Classifier does better job than MLP Classifier in binary classification. The first reservation is although MLP Classifier models perform the worst in 4 out of 5 measurements, their specificity is highest. This case is about predicting credit risk in which more attention is drawn to the high risk (class 1), the specificity, also known as true negative rate, deserves more weight. Another argument is, the MLP Classifier parameters used for all 3 datasets are the best for only data sample 1 and accurately speaking, they are local optimum with limited options tested.

Last but not least, all 3 Random Forest models give certain importance weights on major factors given by the original datasets. In a reference to a survey on 126 banks in the US cited in a publication by John M. Chapman (1940), the bankers put quite high importance score on income, the variable already discarded on data samples 2 and 3. Therefore, in future, we may consider raise the bar, e.g. only discarding variables having more than 20-25% missing values, in order to avoid losing major contributing factors. According to the survey, the bankers also thought high on stability of employment and characters and reputation of credit applicants. We may consider adding these factors in future research.

APPENDICES

Appendix 1. "Trial-and-error" table for accuracy test

	firstLayer_N	2ndLayer_0N	2ndLayer_1N	2ndLayer_2N	2ndLayer_3N	2ndLayer_4N	2ndLayer_5N
0	1	0.93874	0.93874	0.93874	0.93874	0.93874	0.93874
1	2	0.0612596	0.0625283	0.93874	0.93874	0.937472	0.93874
2	3	0.937608	0.93874	0.93874	0.937336	0.0614862	0.937925
3	4	0.93874	0.93874	0.93874	0.0632533	0.0612596	0.938469
4	5	0.938514	0.937653	0.0612596	0.93874	0.93874	0.0612596
5	6	0.937653	0.93874	0.93874	0.937426	0.880788	0.0623924
6	7	0.0613049	0.114318	0.937517	0.93874	0.938695	0.937608
7	8	0.0622565	0.0612596	0.938695	0.0623924	0.937653	0.936565
8	9	0.0639329	0.937608	0.93874	0.937381	0.93874	0.936022
9	10	0.938695	0.937608	0.93874	0.313638	0.0623924	0.937744
10	11	0.93874	0.937653	0.93874	0.0612596	0.937653	0.843453
11	12	0.93874	0.938061	0.0612596	0.934572	0.937608	0.939375
12	13	0.935976	0.0612596	0.0626189	0.0612596	0.939148	0.938831
13	14	0.0612143	0.937517	0.901087	0.0612596	0.93874	0.93874
14	15	0.802583	0.0612596	0.938106	0.794155	0.935433	0.127367
15	16	0.937562	0.0613049	0.93874	0.938695	0.93874	0.937744
16	17	0.937608	0.795333	0.761169	0.93498	0.0623924	0.930267
17	18	0.938015	0.937608	0.938695	0.93874	0.93874	0.935569
18	19	0.938061	0.93874	0.936565	0.0612596	0.0623471	0.93865
19	20	0.761667	0.93874	0.938876	0.0612596	0.936656	0.937109

Appendix 2. Confusion Matrices for 2 algorithms on 3 datasets

```
cm_RF1_df = pd.DataFrame(cm_RF1, columns=["Predicted Class " + str(class_name) for class_name in [0,1]],
                         index = ["Actual Class " + str(class_name) for class_name in [0,1]])
cm_RF2_df = pd.DataFrame(cm_RF2, columns=["Predicted Class " + str(class_name) for class_name in [0,1]],
                         index = ["Actual Class " + str(class_name) for class_name in [0,1]])
cm_RF3_df = pd.DataFrame(cm_RF3, columns=["Predicted Class " + str(class_name) for class_name in [0,1]],
                         index = ["Actual Class " + str(class_name) for class_name in [0,1]])
cm_MLP1_df = pd.DataFrame(cm_MLP1, columns=["Predicted Class " + str(class_name) for class_name in [0,1]],
                         index = ["Actual Class " + str(class_name) for class_name in [0,1]])
cm_MLP2_df = pd.DataFrame(cm_MLP2, columns=["Predicted Class " + str(class_name) for class_name in [0,1]],
                         index = ["Actual Class " + str(class_name) for class_name in [0,1]])
cm_MLP3_df = pd.DataFrame(cm_MLP3, columns=["Predicted Class " + str(class_name) for class_name in [0,1]],
                         index = ["Actual Class " + str(class_name) for class_name in [0,1]])

print("Confusion Matrix of Random Forest Classifier on sample 1:\n", cm_RF1_df)
print("\nConfusion Matrix of Random Forest Classifier on sample 2:\n", cm_RF2_df)
print("\nConfusion Matrix of Random Forest Classifier on sample 3:\n", cm_RF3_df)
print("\nConfusion Matrix of MLP Classifier on sample 1:\n", cm_MLP1_df)
print("\nConfusion Matrix of MLP Classifier on sample 2:\n", cm_MLP2_df)
print("\nConfusion Matrix of MLP Classifier on sample 3:\n", cm_MLP3_df)
```

```
Confusion Matrix of Random Forest Classifier on sample 1:
                 Predicted Class 0  Predicted Class 1
Actual Class 0               20589                129
Actual Class 1                1176                176

Confusion Matrix of Random Forest Classifier on sample 2:
                 Predicted Class 0  Predicted Class 1
Actual Class 0               27440                600
Actual Class 1                1544                416

Confusion Matrix of Random Forest Classifier on sample 3:
                 Predicted Class 0  Predicted Class 1
Actual Class 0               26593               1321
Actual Class 1                3126               2565

Confusion Matrix of MLP Classifier on sample 1:
                 Predicted Class 0  Predicted Class 1
Actual Class 0               20692                 26
Actual Class 1                1350                  2

Confusion Matrix of MLP Classifier on sample 2:
                 Predicted Class 0  Predicted Class 1
Actual Class 0                   0              28040
Actual Class 1                   0               1960

Confusion Matrix of MLP Classifier on sample 3:
                 Predicted Class 0  Predicted Class 1
Actual Class 0                   0              27914
Actual Class 1                   0               5691
```

Appendix 3. Classification Report for 2 algorithms on 3 datasets

```
#Classification report
cr_RF1 = pd.DataFrame(metrics.classification_report(y_test1, y_pred_RF1, output_dict=True))
cr_MLP1 = pd.DataFrame(metrics.classification_report(y_test1, y_pred_MLP1, output_dict=True))

cr_RF2 = pd.DataFrame(metrics.classification_report(y_test2, y_pred_RF2, output_dict=True))
cr_MLP2 = pd.DataFrame(metrics.classification_report(y_test2, y_pred_MLP2, output_dict=True))

cr_RF3 = pd.DataFrame(metrics.classification_report(y_test3, y_pred_RF3, output_dict=True))
cr_MLP3 = pd.DataFrame(metrics.classification_report(y_test3, y_pred_MLP3, output_dict=True))

print("Classification Report of Random Forest Classifier on sample 1:\n", cr_RF1)
print("\nClassification Report of Random Forest Classifier on sample 2:\n", cr_RF2)
print("\nClassification Report of Random Forest Classifier on sample 3:\n", cr_RF3)
print("\nClassification Report of MLP Classifier on sample 1:\n", cr_MLP1)
print("\nClassification Report of MLP Classifier on sample 2:\n", cr_MLP2)
print("\nClassification Report of MLP Classifier on sample 3:\n", cr_MLP3)
```

```
Classification Report of Random Forest Classifier on sample 1:
                     0            1  accuracy     macro avg  weighted avg
precision      0.945968     0.577049   0.94087      0.761509      0.923368
recall         0.993774     0.130178   0.94087      0.561976      0.940870
f1-score       0.969282     0.212432   0.94087      0.590857      0.922917
support    20718.000000  1352.000000   0.94087  22070.000000  22070.000000

Classification Report of Random Forest Classifier on sample 2:
                     0            1  accuracy     macro avg  weighted avg
precision      0.946729     0.409449  0.928533      0.678089      0.911627
recall         0.978602     0.212245  0.928533      0.595423      0.928533
f1-score       0.962402     0.279570  0.928533      0.620986      0.917790
support    28040.000000  1960.000000  0.928533  30000.000000  30000.000000

Classification Report of Random Forest Classifier on sample 3:
                     0            1  accuracy     macro avg  weighted avg
precision      0.894815     0.660062  0.867669      0.777438      0.855059
recall         0.952676     0.450712  0.867669      0.701694      0.867669
f1-score       0.922839     0.535658  0.867669      0.729249      0.857270
support    27914.000000  5691.000000  0.867669  33605.000000  33605.000000


Classification Report of MLP Classifier on sample 1:
                     0            1  accuracy     macro avg  weighted avg
precision      0.938753     0.071429  0.937653      0.505091      0.885621
recall         0.998745     0.001479  0.937653      0.500112      0.937653
f1-score       0.967820     0.002899  0.937653      0.485359      0.908710
support    20718.000000  1352.000000  0.937653  22070.000000  22070.000000

Classification Report of MLP Classifier on sample 2:
                     0            1  accuracy     macro avg  weighted avg
precision      0.0      0.065333  0.065333      0.032667      0.004268
recall         0.0      1.000000  0.065333      0.500000      0.065333
f1-score       0.0      0.122653  0.065333      0.061327      0.008013
support    28040.0  1960.000000  0.065333  30000.000000  30000.000000

Classification Report of MLP Classifier on sample 3:
                     0            1  accuracy     macro avg  weighted avg
precision      0.0      0.169350   0.16935      0.084675      0.028679
recall         0.0      1.000000   0.16935      0.500000      0.169350
f1-score       0.0      0.289648   0.16935      0.144824      0.049052
support    27914.0  5691.000000   0.16935  33605.000000  33605.000000
```

Appendix 4. Variable Importance Plots for Random Forest algorithm on 3 datasets

```
#Variable importances for LoanData1 dataset
RF1class_VarImportance = pd.Series(RF1_class.feature_importances_, index = X1.columns)

#Visualize with bar chart
RF1class_VarImportance.nlargest(10).plot(kind="bar", rot=35, color="orange")
plt.title("Variable Importance Plot for Random Forest on Sample 1", size=15, weight="bold")
```

```
Text(0.5, 1.0, 'Variable Importance Plot for Random Forest on Sample 1')
```

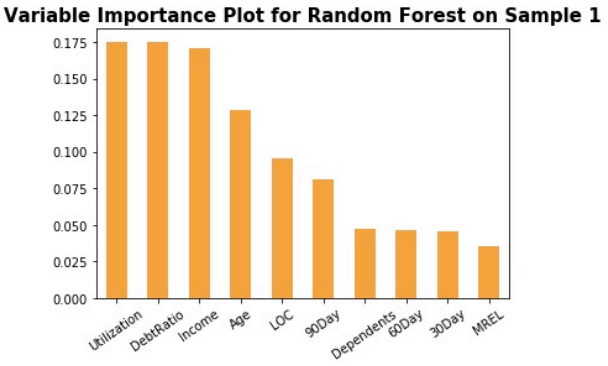


Figure 2.1. Variable Importance Plot for Random Forest on LoanData1 sample

```
#Variable importances for LoanData2 dataset
RF2class_VarImportance = pd.Series(RF2_class.feature_importances_, index = X2.columns)

#Visualize with bar chart
RF2class_VarImportance.nlargest(10).plot(kind="bar", rot=35, color="lightgray")
plt.title("Variable Importance Plot for Random Forest on Sample 2", size=15, weight="bold")
```

```
Text(0.5, 1.0, 'Variable Importance Plot for Random Forest on Sample 2')
```

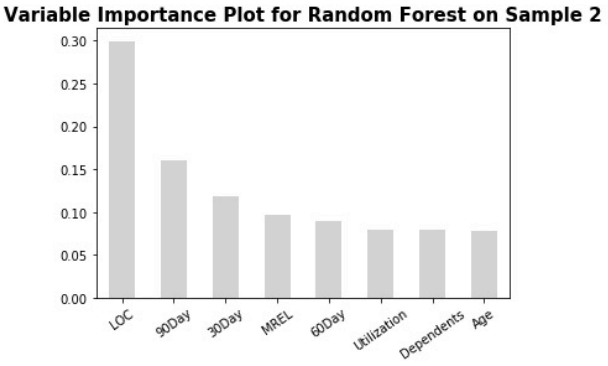


Figure 2.2. Variable Importance Plot for Random Forest on LoanData2 sample

```
#Variable importances for LoanData3 dataset
RF3class_VarImportance = pd.Series(RF3_class.feature_importances_, index = X3.columns)

#Visualize with bar chart
RF3class_VarImportance.nlargest(10).plot(kind="bar", rot=35, color="darkcyan")
plt.title("Variable Importance Plot for Random Forest on Sample 3", size=15, weight="bold")
```

```
Text(0.5, 1.0, 'Variable Importance Plot for Random Forest on Sample 3')
```

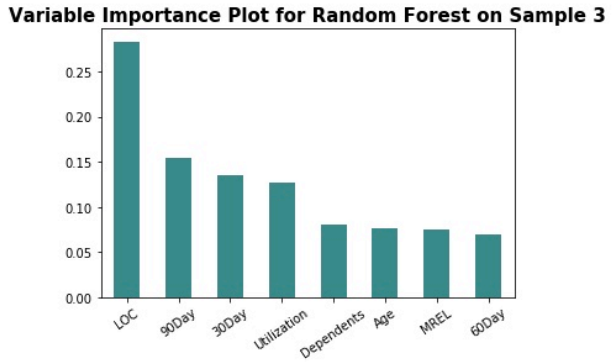


Figure 2.3. Variable Importance Plot for Random Forest on LoanData3 sample

REFERENCES

Berry, M.J.A., and Linoff, G. (1997), Data Mining Techniques, NY: John Wiley & Sons.

Huang, G.B. (2003), Learning capability and storage capacity of two-hidden-layer feedforward networks. IEEE Transactions on Neural Networks, 14, pp. 274–281

John, M.C. et al (1940), Factors Affecting Credit Risk in Personal Lending. National Bureau of Economic Research, pp. 137–139