Integrated Machine Learning Approaches for Credit Score Analysis: A Multi-Faceted Study Using Classification, Regression, Clustering, and Association Rule Mining

Rajat R Belgundi

Department of Computer Science, Virginia Tech

CS 5805: Machine Learning 1

Dr. Reza Jafari

November 28, 2023

Table of Contents

Sr. No	Content	Page No.
1.	Abstract	4
2.	Introduction	5
3.	Dataset Description	6
4.	Phase 1: Feature Engineering and EDA	7
5.	Phase 2: Regression Analysis	30
6.	Phase 3: Classification Analysis	39
7.	Phase 4: Clustering & Association Rule Mining	85
8.	Recommendations	89
9.	Future Work	94
11.	Appendix	95
12.	References	138

Table of Figures

Sr. No	Figure	Page No.
1.	*Figure 1: Target Feature Credit_Score*	6
2.	Figure 2: Payment of Min Amount	13
3.	Figure 3: Random Forest Feature Selection	18
4.	Figure 4: Random Forest Feature Selection	19
5.	Figure 5: PCA for Regression	20
6.	Figure 6: PCA for Classification	22
7.	Figure 7: SVD for Regression	24
8.	Figure 8: SVD for Classification	25
9.	Figure 9: Data Balancing	28
10.	Figure 10: Correlation Analysis	29
11.	Figure 11: Covariance Analysis	30
12.	Figure 12: Linear Regression Result	33
13.	Figure 13: Prediction Interval	35
14.	Figure 14: ROC-AUC of various classifiers	39-82
15.	Figure 15: Silhouette Analysis	86
16.	Figure 16: WCSS	86

Abstract

This study presents a comprehensive analysis of credit score through integrated machine learning (ML) approaches. The project explores the multifaceted application of classification, regression, clustering, and association rule mining to gain a nuanced understanding of individuals' credit scores. This project consists of four phases – 1) Feature Engineering and EDA 2) Regression Analysis 3) Classification Analysis 4) Clustering and Association Rule Mining Analysis.

In the feature engineering & EDA phase, techniques such as data cleaning, feature selection, feature transformation, discretization, correlation, covariance and outlier analysis and removal have been employed to ensure quality data is being used in the subsequent phases. In the classification phase, various classifiers are used to classify the customers into 3 categories of credit scores – good, standard and poor. The classifiers are provided with credit-related features like outstanding debt, number of loans, interest rate, delay from due date and many more to categorize the customers. The classifiers employed for this analysis include Decision TreeLogistic Regression, KNN, SVM, Ensemble Techniques (Stacking, Bagging, and Boosting) and Neural Network (MLP Classifier). In the regression phase, t-test and f-test analysis have been used in conjunction with stepwise regression analysis (OLS). Multiple Linear Regression has been used to predict outstanding debt of customers based on the credit-related data. Clustering techniques reveal distinct groups of customers with similar credit profiles, shedding light on diverse financial behaviors within the dataset. Association rule mining uncovers intricate relationships between various financial indicators, offering insights into hidden patterns.

Introduction

In financial institutions like banks, credit related information about the customers like number of loans, number of credit inquiries, outstanding loan, type of loan taken, interest rate etc. is being collected. One such dataset is being used for this project.

In this project, comprehensive machine learning approaches are used to perform credit score analysis. The project spans over four phases – Phase1, Phase 2, Phase 3, and Phase 4. The phase 1 of the project implements data cleaning, feature engineering and exploratory data analysis phases to ensure proper data quality, appropriate features for modeling. Comprehensive data analysis techniques like correlation, covariance analysis with outlier analysis and removal have been employed in this project. Feature transformation techniques like standardization, one-hot encoding have been implemented to make the data ready for modeling. Numerous feature selection techniques like- Random Forest, PCA (Principal Component Analysis), SVD (Singular Value Decomposition), VIF (Variance Inflation Factor) help in selecting the most relevant features from the dataset. The phase 2 includes regression analysis that is used to predict outstanding debt which can help the bank to flag the customers who have really high loan amounts, also see which customers make their monthly payments on time etc. Multiple Linear Regression with F-Test and T-Test Analysis is used to predict the outstanding debt of a customer. The phase 3 includes classification techniques that are used to classify the customers into 3 brackets – good, standard and poor credit scores. We will make use of classifiers like: 1) Decision Tree 2) Logistic Regression 3) KNN 4) SVM Classifiers (Linear, Poly, RBF kernels) 5) Naïve Bayes Classifier 6) Random Forest (Stacking, Bagging, Boosting) 7) Multi-layer perceptron. The phase 4 implements clustering and association rule mining analysis that will help uncover additional insights from this credit related dataset.

Dataset Description

The dataset is titled, ‘Credit Score Classification’ which is taken from the Kaggle website. The dataset includes credit-related information of customers with features like Outstanding Debt, Interest Rate, Number of Loans, Type of Loans, Credit History Age, Delay from due date, Credit_Score etc. This data can be used to place customers into various credit score brackets and take some important decisions like curating offers to specific customers, deciding whether new loan should be offered or not and many more like this.

- The dataset consists of 100,000 instances with 17 numerical features, 10 categorical features based on the information provided. Based on the analysis in Phase 1, we will fix the number of numerical and categorical features and will handle data types accordingly. The number of observations will also reduce after data cleaning.
- For the regression dataset, our target feature is ‘Outstanding Debt’. For the classification analysis, our target feature is ‘Credit Score’ which is a multi-class classification problem.
- With the regression analysis, wherein we predict the outstanding debt of the customer, the bank or institution can identify the customers with really high loan amounts due, the customers which pay their monthly installments on time, the customers which might need loan in the near future. The customers with a great credit score will be offered loan with less interest rates thus prompting the customer to take up a loan. The bank can also come up with marketing strategies for each customer bracket based on credit score which can be used to widen their customer base.
- Credit score classification has many more applications in the banking industry when it comes to approving loan, credit cards or other investment schemes. The bank can also use the association rules to identify customer behaviors and plan accordingly.

Phase 1: Feature Engineering and EDA

Phase 1 starts with data cleaning. We have general information columns like Name, ID and SSN of customers. These columns will be dropped as they do not add value to our analysis.

On checking for duplicates, there were no duplicates found. We then proceed with checking the imbalance in the target feature for Classification analysis. Along with this we will encode our categorical target column Credit_Score as: {'Poor': 0, 'Standard': 1, 'Good': 2}. Phase 1 also includes preparation of datasets for classification, regression and association rule mining tasks. We will have separate CSV files for each of the above mentioned phases.

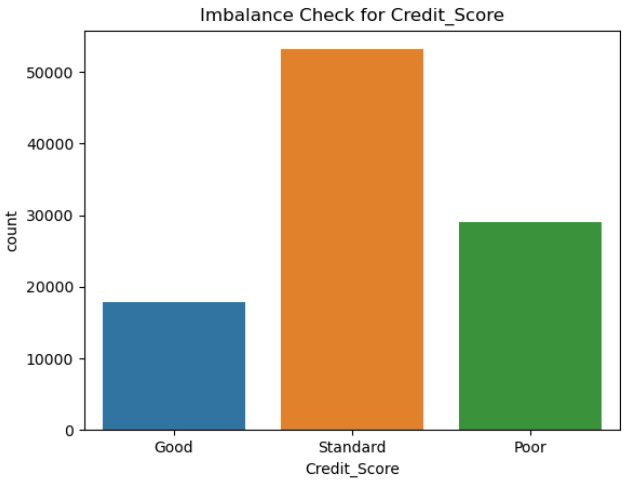


Figure 2: Target Feature Credit_Score

Data cleaning of numerical columns:

We will work in 2 phases here:

1) Cleaning Garbage values along with leading and trailing underscores (‘_’).
2) Handling missing values(NaN values)

Cleaning Garbage Values:

In the numerical columns like Age, Outstanding_Debt, Changed_Credit_Limit etc. there are underscores (‘_’) either as a part of the record or as the record itself. We will clean the data by replacing them with empty strings with a method as follows:

```
1 usage
def clean_numeric_data(data):
    try:
        return float(data.replace("_", ""))
    except:
        return np.nan
```

```
Column Name: Amount_invested_monthly
**********************************************
__10000__            4305
0.0                   169
80.41529543900253       1
36.66235139442514       1
89.7384893604547        1
                      ...
36.541908593249026      1
93.45116318631192       1
140.80972223052834      1
38.73937670100975       1
167.1638651610451       1
Name: Amount_invested_monthly, Length: 91049, dtype: int64
END ----------------------------------------
```

```
Column Name: Amount_invested_monthly
**********************************************
NaN             4479
10000.000000    4305
0.000000         169
36.662351          1
89.738489          1
                 ...
36.541909          1
93.451163          1
140.809722         1
38.739377          1
167.163865         1
Name: Amount_invested_monthly, Length: 91050, dtype: int64
END ----------------------------------------
```

```
Column Name: Outstanding_Debt
**********************************************
1360.45     24
460.46      23
1151.7      23
1109.03     23
467.7       16
            ..
245.46_      1
645.77_      1
174.79_      1
1181.13_     1
1013.53_     1
Name: Outstanding_Debt, Length: 13178, dtype: int64
```

```
Column Name: Changed_Credit_Limit
**********************************************
_                    2091
8.22                  133
11.5                  127
11.32                 126
7.35                  121
                      ...
-1.84                   1
0.8899999999999999      1
28.06                   1
1.5599999999999996      1
21.17                   1
Name: Changed_Credit_Limit, Length: 4384, dtype: int64
END ----------------------------------------
```

After cleaning this data, our data looks like:

```
Column Name: Outstanding_Debt
**********************************************
1109.03    24
1151.70    24
1360.45    24
460.46     24
1058.13    16
           ..
4230.04     8
641.99      8
98.61       8
2614.48     8
502.38      8
Name: Outstanding_Debt, Length: 12203, dtype: int64
END ----------------------------------------
```

```
Column Name: Changed_Credit_Limit
**********************************************
 NaN      2091
 8.22      133
 11.50     127
 11.32     126
 7.35      121
           ...
-1.84        1
 0.89        1
 28.06       1
 1.56        1
 21.17       1
Name: Changed_Credit_Limit, Length: 4384, dtype: int64
END ----------------------------------------
```

We see that the values with underscores are no longer popping up in the value counts of that column, as they have been cleaned.

We will fill the NaN values in numerical columns with average of that column with respect to the customer-id.

```
# 1. Monthly_Inhand_Salary
# We can replace with mean of Customer_ID
df['Monthly_Inhand_Salary'] = df.groupby('Customer_ID')['Monthly_Inhand_Salary'].transform(mean_fn)

# 2. Num_of_Delayed_Payment
# We replace with mean of Customer_ID
df['Num_of_Delayed_Payment'] = df.groupby('Customer_ID')['Num_of_Delayed_Payment'].transform(mean_fn)
```

After cleaning remaining NaN values we have zero missing values

```
After cleaning remaining NaN values:
                          0
Customer_ID               0
Month                     0
Age                       0
Occupation                0
Annual_Income             0
Monthly_Inhand_Salary     0
Num_Bank_Accounts         0
Num_Credit_Card           0
Interest_Rate             0
Num_of_Loan               0
Delay_from_due_date       0
Num_of_Delayed_Payment    0
Changed_Credit_Limit      0
Num_Credit_Inquiries      0
Credit_Mix                0
Outstanding_Debt          0
Credit_Utilization_Ratio  0
Credit_History_Age        0
Payment_of_Min_Amount     0
Total_EMI_per_month       0
Amount_invested_monthly   0
Payment_Behaviour         0
Monthly_Balance           0
Credit_Score              0
```

Data Cleaning in Categorical columns:

```
-----------------------------------------------------
         Data Cleaning in Object type columns
-----------------------------------------------------
Index(['Customer_ID', 'Month', 'Occupation', 'Type_of_Loan', 'Credit_Mix',
       'Credit_History_Age', 'Payment_of_Min_Amount', 'Payment_Behaviour'],
      dtype='object')
```

For categorical columns, the missing values and garbage values are replaced with the mode of that column grouped by Customer_ID.

The dataset consists of a column called Customer_ID which has 8 unique values which are equally distributed in the data.

For example, cleaning Occupation column

The garbage values are first replaced with NaN to make sure that all values to be replaced are only NaN.

The mode of the Occupation column grouped on customer_id is used to replace the missing values in the column

```
# 2. Occupation
# Replace garbage value(_______) with mode based on customer_id
print('------------------------Occupation------------------------')
print('Before')
print(df['Occupation'].value_counts())
# Replace with mode w.r.t Customer_ID
df['Occupation'] = df['Occupation'].replace('_______', np.nan)
mode_fn = lambda x: x.mode().iat[0]
df['Occupation'] = df['Occupation'].fillna(df.groupby('Customer_ID')['Occupation'].transform(mode_fn))
print('After')
print(df['Occupation'].value_counts())
plt.figure(figsize=(24, 12))
sns.countplot(data=df, x='Occupation', hue='Credit_Score')
plt.title('Occupation wise Credit Score')
plt.show()
```

```
Before
_______          7062
Lawyer           6575
Architect        6355
Engineer         6350
Scientist        6299
Mechanic         6291
Accountant       6271
Developer        6235
Media_Manager    6232
Teacher          6215
Entrepreneur     6174
Doctor           6087
Journalist       6085
Manager          5973
Musician         5911
Writer           5885
Name: Occupation, dtype: int64
```

```
After
Lawyer           7096
Engineer         6864
Architect        6824
Mechanic         6776
Scientist        6744
Accountant       6744
Developer        6720
Media_Manager    6720
Teacher          6672
Entrepreneur     6648
Doctor           6568
Journalist       6536
Manager          6432
Musician         6352
Writer           6304
Name: Occupation, dtype: int64
```

For Credit Mix column

```
------------------- Credit_Mix -------------------
Before Replace
Standard    32491
Good        19325
Bad         18989
_           17787
Name: Credit_Mix, dtype: int64
After replace
Standard    40736
Good        24088
Bad         23768
Name: Credit_Mix, dtype: int64
```

For Payment_of_Min_Amount column:

```
------------------- Payment_of_Min_Amount -------------------
Yes    49515
No     28372
NM     10705
Name: Payment_of_Min_Amount, dtype: int64
0
Yes    56264
No     32328
```

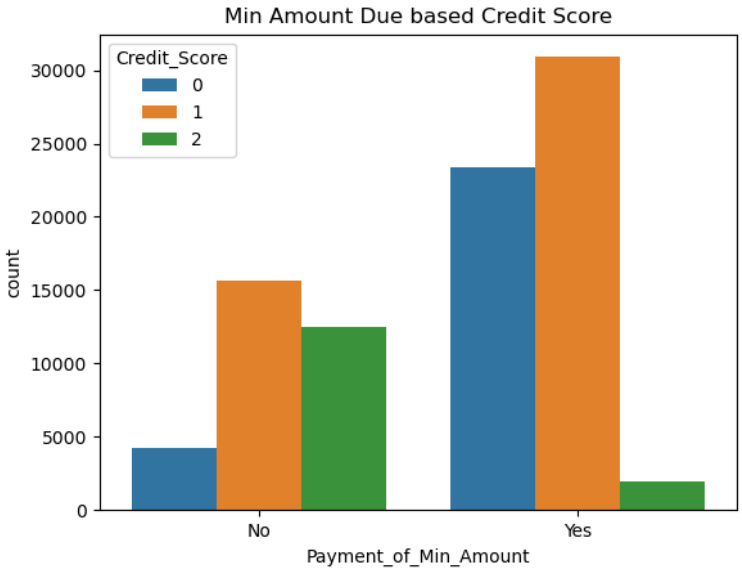


Figure 2: Payment_of_Min_Amount

For other categorical columns, we will replace the missing values with the mode of that column grouped by Customer_ID.

Outlier Analysis and Removal

Let’s examine the columns which possibly have outliers before we decide to implement the outlier removal strategy.

```
# Observations
# 1. Age: Min value = -500, Max value = 867  clean up age
# 2. Num_Bank_Accounts: min value = -1 we can make it 0
# 3. Num_Credit_Card: Outlier Detection
# 4. Interest_Rate:  Outlier Detection
# 5. Num_of_Loan: Min value = -100
# 6. Delay_from_due_date: Min value = -5
# 7. Num_of_Delayed_Payment: Min value = -1
# 8. Changed_Credit_Limit: Min value = -1.07 (Explore Later)
# 9. Num_Credit_Inquiries: Min value = 0.0  Max value: 600
# 10. Outstanding_Debt: Min Value = 0.23  Max value = 4998
# 11. Credit_Utilization_Ratio: Min value = 20  Max value = 49.5
# 12. Credit_History_Age: Min value = 4.5  Max value = 400
# 13. Total_EMI_per_month: Min value = 4.46  Max value = 82331
# 14. Amount_invested_monthly: Min value = 15.2  Max value = 1313
# 15. Monthly_Balance: Min value = large -ve  Max value = 1313, garbage value = 3333333333
# 16. Annual_Income: Min value = 7005   Max value = large +ve
# 17. Monthly_Inhand_Salary: Min value = 303.6  Max value = 15,204
```

From the above observations we can see that there a quite a few columns which contain outliers or anomalies.

We will implement the Interquartile Range Method for outlier removal.

We see that Age has negative values and also extremely high positive values such as 867.

We will keep values between 0 and 100 for Age column

```
# 1. Age
print('-----------------------------------------')
print('                  Age                    ')
print('-----------------------------------------')
# Handle negative values and values greater than 100
df = df[(df['Age'] > 0) & (df['Age'] < 100)]
print(df.shape)
imbalance_check(df, col: 'Credit_Score')
```

For example, number of bank accounts contain negative values and some extremely high positive values.

```
# 2. Num_Bank_Accounts
# Replace -1 with 0
df.loc[df['Num_Bank_Accounts'] < 0, 'Num_Bank_Accounts'] = 0
sns.boxplot(data=df, x='Num_Credit_Card')
plt.title('Number of Bank Accounts with Outliers')
plt.show()
# print('Outlier instances in Num_Bank_Accounts: ', df[df['Num_Bank_Accounts'] > 8 & df['Num_Bank_Account
q25 = df['Num_Bank_Accounts'].quantile(0.25)
q75 = df['Num_Bank_Accounts'].quantile(0.75)
iqr = q75 - q25
upper = q75 + 1.5 * iqr
# lower = q25 - 1.5 * iqr
lower = 0
df = df[(df['Num_Bank_Accounts'] >= lower) & (df['Num_Bank_Accounts'] <= upper)]
print('After Removing Outliers from # of Bank Accounts')
print(df.shape)
```

Few more examples of outlier analysis and removal

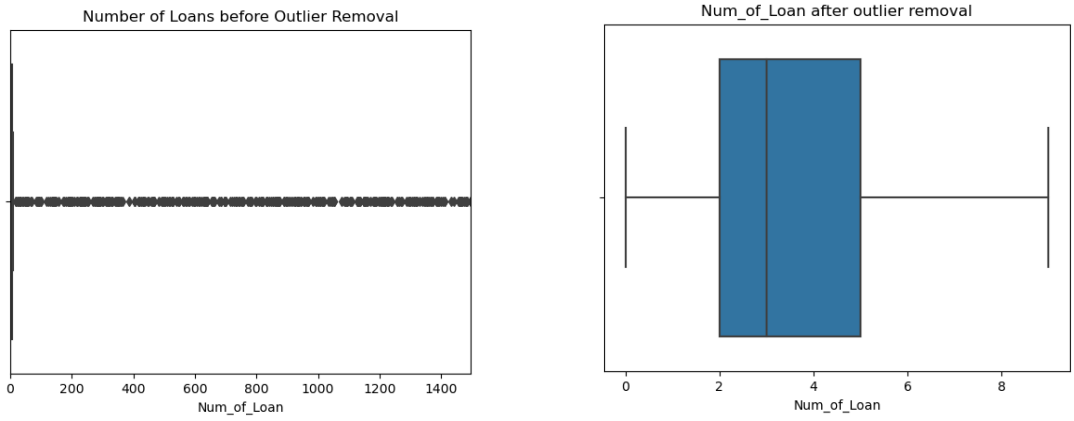


Figure 3: Num_of_Loan Outlier Analysis and Removal

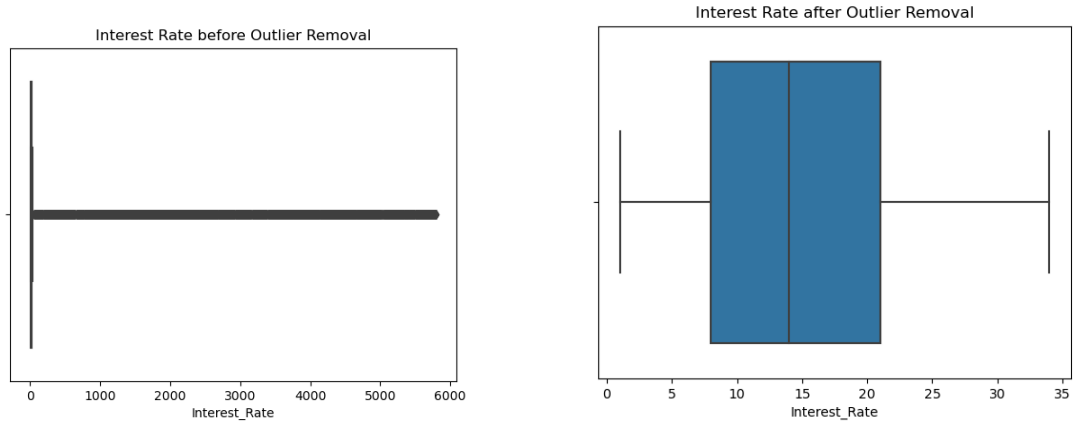


Figure 4: Interest Rate Outlier Analysis and Removal

Standardization:

```
def standardize_data(df):
    standardized_df = (df - df.mean(numeric_only=True)) / (df.std(numeric_only=True))
    return standardized_df
```

One-Hot Encoding:

In this section, we use pandas get dummies function to on hot encode the categorical features.

We avoid the dummy variable trap by setting drop_first = True.

```
--------------------------------------------------
                One-Hot Encoding
--------------------------------------------------
-----Encoding in progress-----
Columns: Index(['Payment_of_Min_Amount_Yes', 'Credit_Mix_Good', 'Credit_Mix_Standard',
       'Occupation_Architect', 'Occupation_Developer', 'Occupation_Doctor',
       'Occupation_Engineer', 'Occupation_Entrepreneur',
       'Occupation_Journalist', 'Occupation_Lawyer', 'Occupation_Manager',
       'Occupation_Mechanic', 'Occupation_Media_Manager',
       'Occupation_Musician', 'Occupation_Scientist', 'Occupation_Teacher',
       'Occupation_Writer',
       'Payment_Behaviour_High_spent_Large_value_payments',
       'Payment_Behaviour_High_spent_Medium_value_payments',
       'Payment_Behaviour_High_spent_Small_value_payments',
       'Payment_Behaviour_Low_spent_Large_value_payments',
       'Payment_Behaviour_Low_spent_Medium_value_payments',
       'Payment_Behaviour_Low_spent_Small_value_payments', 'Month_August',
       'Month_February', 'Month_January', 'Month_July', 'Month_June',
       'Month_March', 'Month_May'],
```

We can see that for the column Credit_Mix which had 3 categories [Good, Standard, Poor] is now split into 2 columns Credit_Mix _Good, Credit_Mix _Standard which avoid dummy variable trap. Similarly we can see other columns being encoded.

Feature Selection and Dimensionality Reduction:

1. Random Forest for feature selection:

   Random Forest based feature selection shows which features are being selected based on the set threshold. This makes it easy to explain or provide reason behind modeling our data using the classifiers. Knowing the features selected also can be verified by domain experts indicating that we are proceeding in the right direction before we employ classification algorithms to make predictions.

   We apply Random Forest feature selection for Regression and Classification tasks, once using the Regressor module and once the Classifier module

   a. Regression Dataset

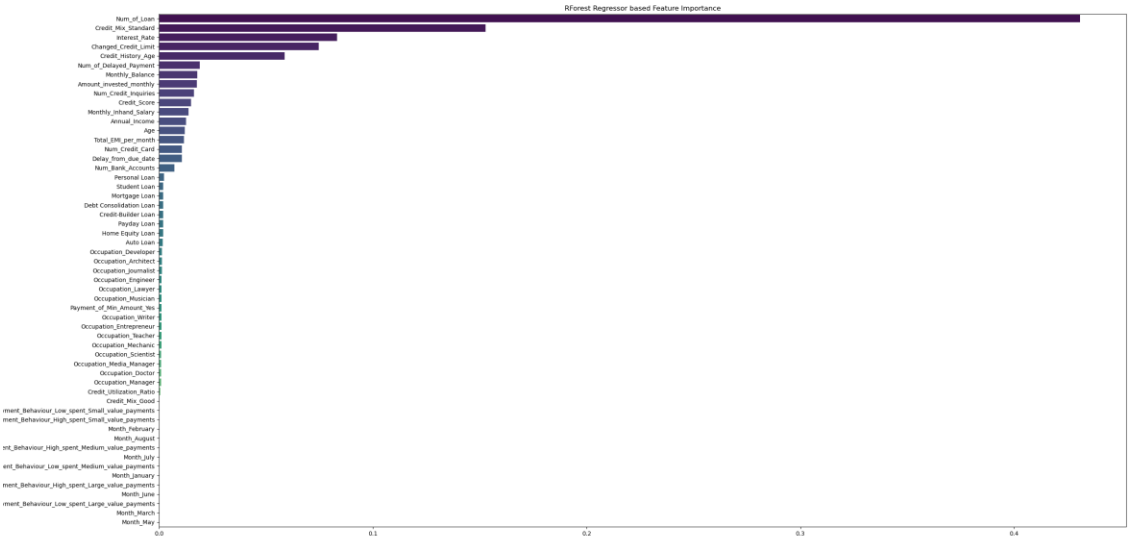


Figure 5: Random Forest Regression Feature Selection

```
Random Forest based feature selection for Regression Dataset
------------------METHOD 1: RANDOM_FOREST REGRESSION-----------------
No. of features selected by RForest Regressor Analysis: 16
(RForest Regressor Analysis) Selected Features with threshold = 0.01:
['Num_of_Loan', 'Credit_Mix_Standard', 'Interest_Rate', 'Changed_Credit_Limit', 'Credit_History_Age', 'Num_of_Delayed_Payment', 'Monthly_Balance', 'Amount_invested_m
(RForest Regressor Analysis) Eliminated Features with threshold = 0.01: ['Num_Bank_Accounts', 'Personal Loan', 'Student Loan', 'Mortgage Loan', 'Debt Consolidation L
```

Features selected from Regression dataset

['Num_of_Loan', 'Credit_Mix_Standard', 'Interest_Rate', 'Changed_Credit_Limit', 'Credit_History_Age', 'Num_of_Delayed_Payment', 'Amount_invested_monthly', 'Monthly_Balance', 'Num_Credit_Inquiries', 'Credit_Score', 'Monthly_Inhand_Salary', 'Annual_Income', 'Age', 'Num_Credit_Card', 'Total_EMI_per_month', 'Delay_from_due_date']

b. Classification Dataset

We set a threshold of 0.016 to select the best features in our dataset and use these features in our classification phase.

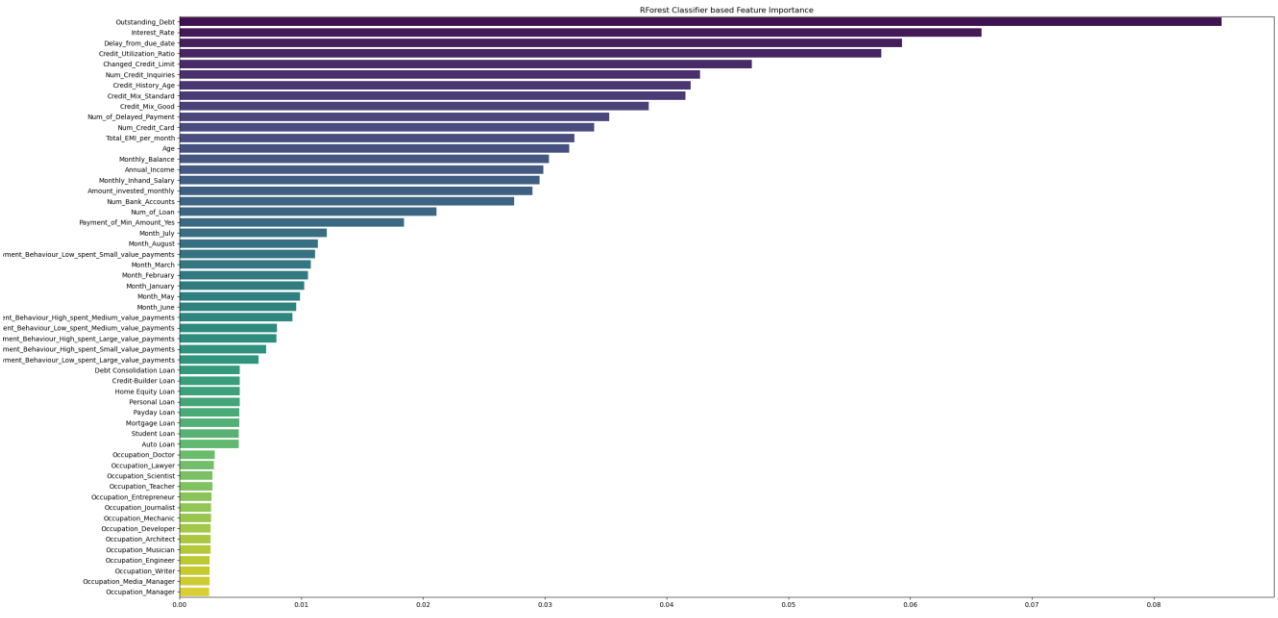


Figure 6: Random Forest Classification Feature Selection

```
Random Forest based feature selection for Classification Dataset
-----------------METHOD 1: RANDOM_FOREST CLASSIFICATION-----------------
No. of features selected by RForest Analysis: 20
(RForest Classifier Analysis) Selected Features with threshold = 0.016:
['Outstanding_Debt', 'Interest_Rate', 'Delay_from_due_date', 'Credit_Utilization_Ratio', 'Changed_Credit_Limit', 'Num_Credit_Inquiries', 'Credit_History_Ag
(RForest Classifier Analysis) Eliminated Features with threshold = 0.016: ['Month_July', 'Month_August', 'Payment_Behaviour_Low_spent_Small_value_payments'
```

Selected features:

['Outstanding_Debt', 'Interest_Rate', 'Delay_from_due_date', 'Credit_Utilization_Ratio', 'Changed_Credit_Limit', 'Credit_Mix_Standard', 'Credit_History_Age', 'Num_Credit_Inquiries', 'Num_of_Delayed_Payment', 'Credit_Mix_Good', 'Num_Credit_Card', 'Total_EMI_per_month', 'Age', 'Monthly_Balance', 'Monthly_Inhand_Salary', 'Amount_invested_monthly', 'Annual_Income', 'Num_Bank_Accounts', 'Num_of_Loan', 'Payment_of_Min_Amount_Yes']

2. Principal Component Analysis:

   Principal Component Analysis (PCA) for dimensionality reduction is a technique used to reduce the number of features (dimensions) in a dataset while retaining the most important information and minimizing information loss.

   a. Regression Dataset

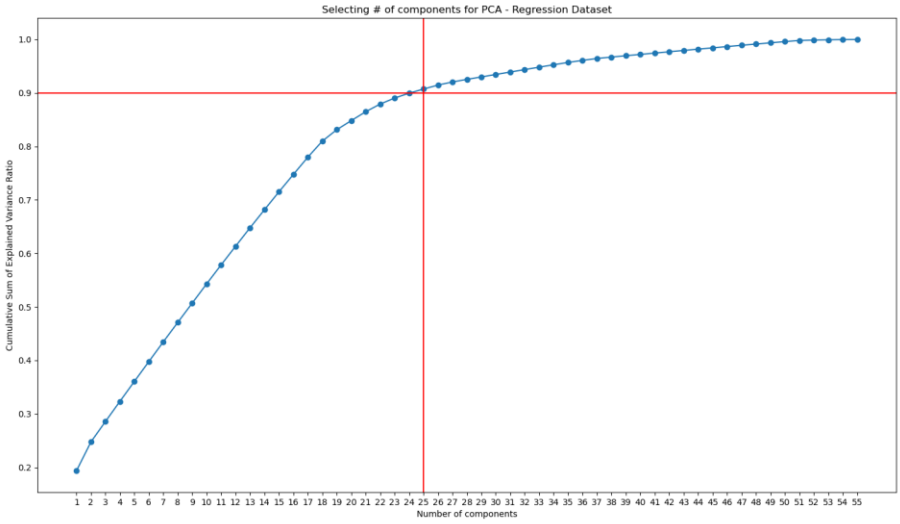


Figure 7: PCA for Regression

Condition Number with PCA:

```
Condition Number for Regression Dataset
 ------------------------------
        Condition Number
 ------------------------------
Condition Number (pre PCA): 1205.6665300386383
Condition Number (post PCA) and 25 components: 62.65817475918088
```

Observations:

1. We set threshold of explained variance to 90%.
2. N-components will be selected based on explained_variance > 90%.
3. From the above graph, we can see that the horizontal red line is the line of 90% variance.
4. At n-components = 25, we are getting explained variance > 90%.
5. We cannot determine specifically which features are being chosen as relevant to the target feature – Outstanding Debt
6. The condition number for Regression data, reduced from 1205 to 62, indicating reduction in collinearity in the dataset.

b. Classification Dataset

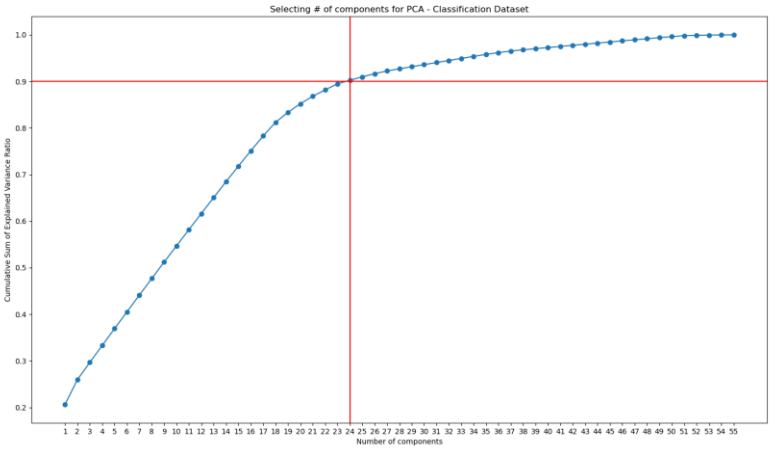


Figure 8: PCA for Classification

Condition Number with PCA:

```
Condition Number for Classification Dataset
 ------------------------------
        Condition Number
 ------------------------------
Condition Number (pre PCA): 1303.4004302906787
Condition Number (post PCA) and 25 components: 68.62529560831062
```

Observations:

1. We set the threshold of 90% explained variance.
2. We look for n_components where explained variance > 90%. N-components = 24 shows the point where we attain 90% explained variance.
3. From the above graph, we can see that it is 25 components for the classification dataset. (Target feature: Credit_Score).
4. PCA does not tell us which features are being selected as it transforms the features to different axes (along Eigen vectors). Thus, even though PCA is a powerful dimensionality reduction technique, it is unable to tell us which features are most relevant to our target feature.
5. The condition number for classification dataset has reduced from 1303 to 68 which indicates that after the use of PCA we are successful in reducing the collinearity.

3. Singular Value Decomposition (SVD)

   Singular Value Decomposition (SVD) is a matrix factorization technique that can be used for dimensionality reduction. In the context of dimensionality reduction, SVD is often employed to approximate a high-dimensional matrix by retaining only its most important components.

   a. Regression Dataset

      i. We can see that on including all features, we have 99.9% explained variance ratio.

```
METHOD 3: Singular Value Decomposition (SVD) for Regression
------------------------------------------------------------------------
          Singular Value Decomposition (SVD) for Regression
------------------------------------------------------------------------
Original Data Shape: (50000, 55)
Explained Variance Ratio: 0.9998056846396535
```

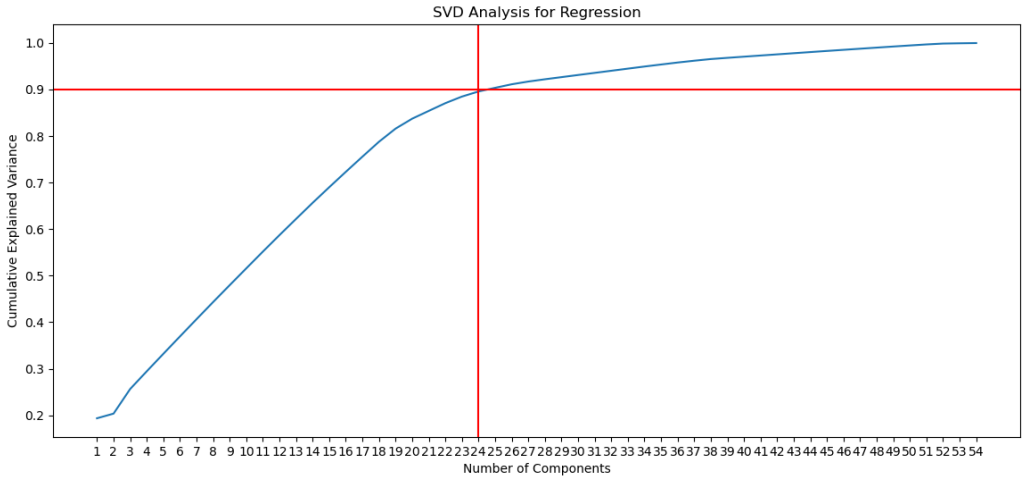


Figure 9: SVD for Regression

Observations:

1. We set the threshold of 90% explained variance for SVD Analysis.
2. From the graph we see that, n-components for explained variance > 90% is 25. At n-components = 24, we reach the 90% mark of explained variance.
3. Thus, n-components based on SVD Analysis = 25.

b. Classification Dataset

   i. We can see that on including all features, we have 99.9% explained variance ratio.

```
METHOD 3: Singular Value Decomposition (SVD) for Classification
------------------------------------------------------------------------
          Singular Value Decomposition (SVD) for Classification
------------------------------------------------------------------------
Original Data Shape: (77993, 55)
Explained Variance Ratio: 0.9998080916854398
```

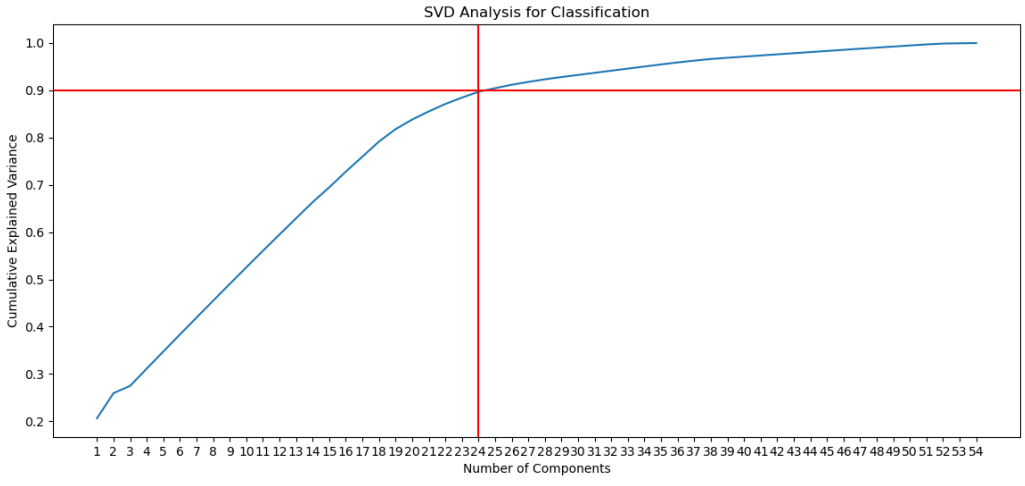


Figure 10: SVD for Classification

Observations:

1. We set the threshold of 90% explained variance for SVD Analysis.
2. From the graph we see that, n-components for explained variance > 90% is 25. At n-components = 24, we reach the 90% mark of explained variance.

4. Variance Inflation Factor (VIF)

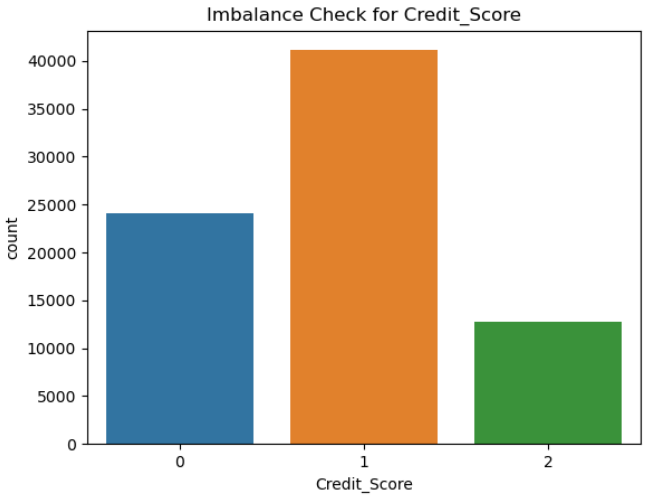
   VIF is a feature selection technique which helps tackle the multi-collinearity in regression analysis. By setting threshold for VIF, features can be eliminated to make sure that collinearity does not exist or is reduced by a large extent.

```
METHOD 4: VIF (Variance Inflation Factor)
                   Variable       VIF
6        Credit_History_Age  2.174652
9       Delay_from_due_date  2.070946
4      Changed_Credit_Limit  1.427728
5       Credit-Builder Loan  1.224757
8   Debt Consolidation Loan  1.213936
3                 Auto Loan  1.208368
0                       Age  1.092961
7  Credit_Utilization_Ratio  1.040465
1   Amount_invested_monthly  1.030896
2             Annual_Income  1.001795
```

   Observations:

   1. The above results show the top 10 features with highest VIF values.
   2. We can see that all the features have VIF less than 5 indicating that collinearity has been reduced.

Data Imbalance Check:

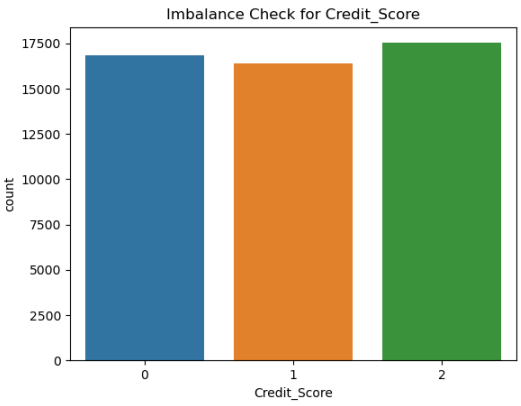


As we can clearly see that the data is imbalanced, we need to balance it.

For balancing the data, we use

1. SMOTE Over Sampling – Use SMOTE to balance the classes to have fairly equal count.
2. Imblearn based Under Sampling – Down-sample the data.
3. TomekLinks Under Sampling – Down sample the data to remove data points which confuse the model.

After implementing the above strategy for balancing the dataset, we see the following results:




```
1    0.527291
0    0.308989
2    0.163720
Name: Credit_Score, dtype: float64
Oversampled X: (123375, 55)
Balance the dataset
2    0.345035
0    0.331476
1    0.323489
Name: Credit_Score, dtype: float64
```

Figure 11: Data Balancing

Correlation Analysis

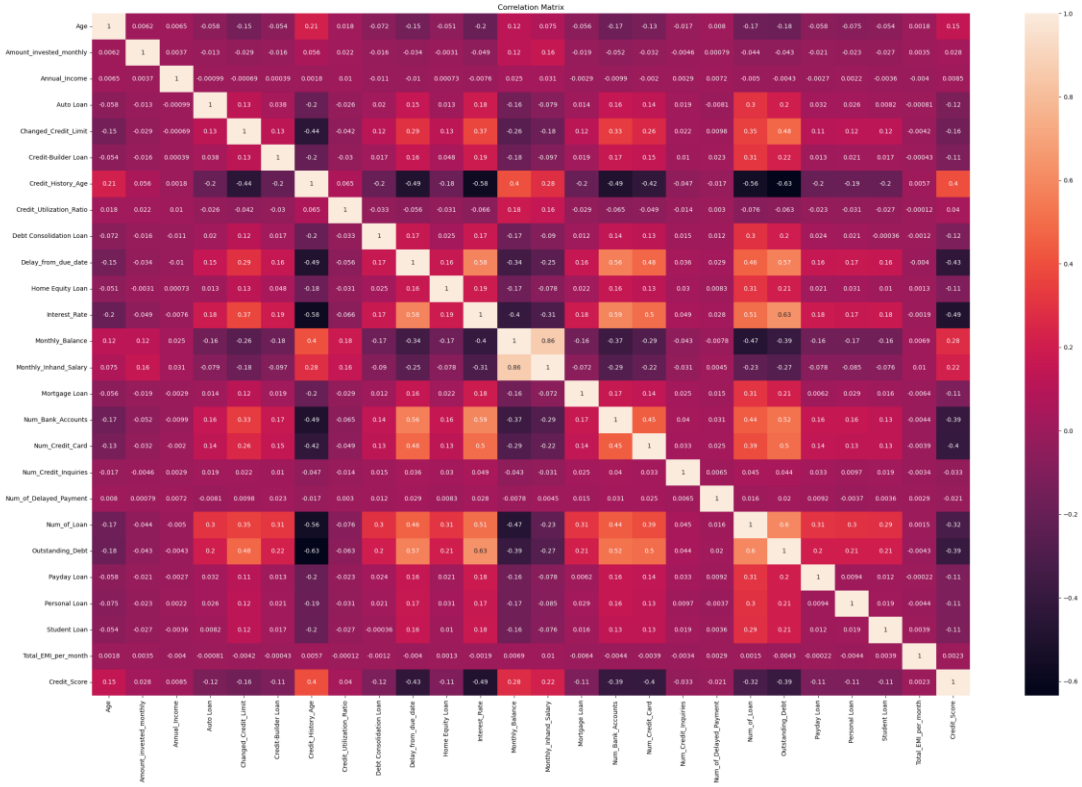


Figure 12: Correlation Analysis

Observations:

1. We can see that Monthly Salary and Monthly Balance have high correlation 0.86. Thus we have to drop one of them.
2. The next highest correlation is between Interest Rate and Outstanding Debt -0.63.
3. Interest Rate and Credit_History_Age have correlation of -0.58.
4. Credit_History_Age and Num_of_Loan have a correlation of -0.56.

Covariance Analysis

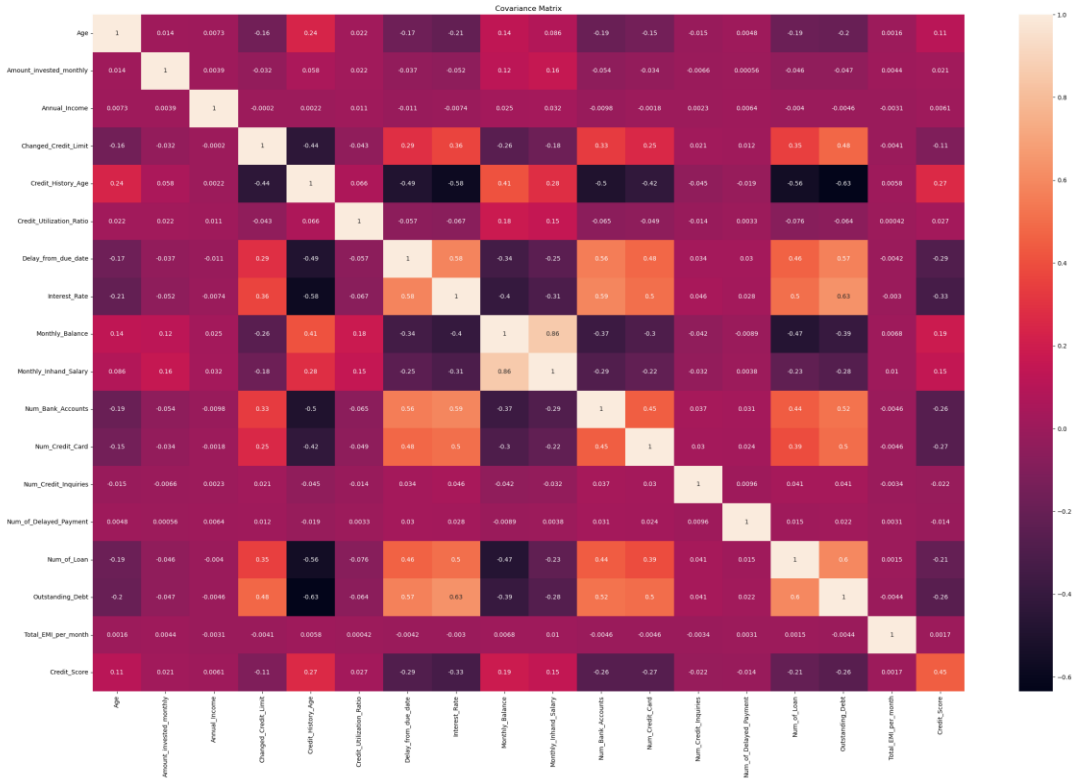


Figure 13: Covariance Analysis

Observations:

1. We can see that Monthly Balance and Monthly_Inhand_Salary have high covariance - 0.86. Thus, they tend to increase with one another.
2. Outstanding Debt and Credit_History_Age have covariance of -0.63. Thus, when one increases, the other decreases.

Phase 2: Regression Analysis

In this phase we will predict the outstanding debt based on the credit related information we have in the dataset. Target = 'Outstanding_Debt'. This phase includes Stepwise Regression Analysis with T-test and F-test Analysis for selecting features to perform Multiple Linear Regression. Regression analysis also involves confidence interval analysis.

Stepwise Regression Analysis OLS:

1. F- Test Analysis
    a. The F-test in linear regression is used to assess the overall significance of a regression model. It helps determine whether the model, with all its predictor variables, explains a significant amount of variability in the response variable.
    b. Higher F-statistic value is always preferred, indicating higher significance of the model.

    Before we start stepwise regression analysis, we can check the f-statistic value and max-pvalue.

```
F-value: 1648.362
Max p-value: 0.968
Current Adjusted R-squared: 0.694
```

After we perform regression analysis, final values of f-statistic and max-pvalue.

```
Feature to be dropped: Occupation_Media_Manager
F-value: 3485.909
Max p-value: 0.001
Current Adjusted R-squared: 0.694
Regression Analysis complete
```

2. Stepwise Regression Analysis OLS

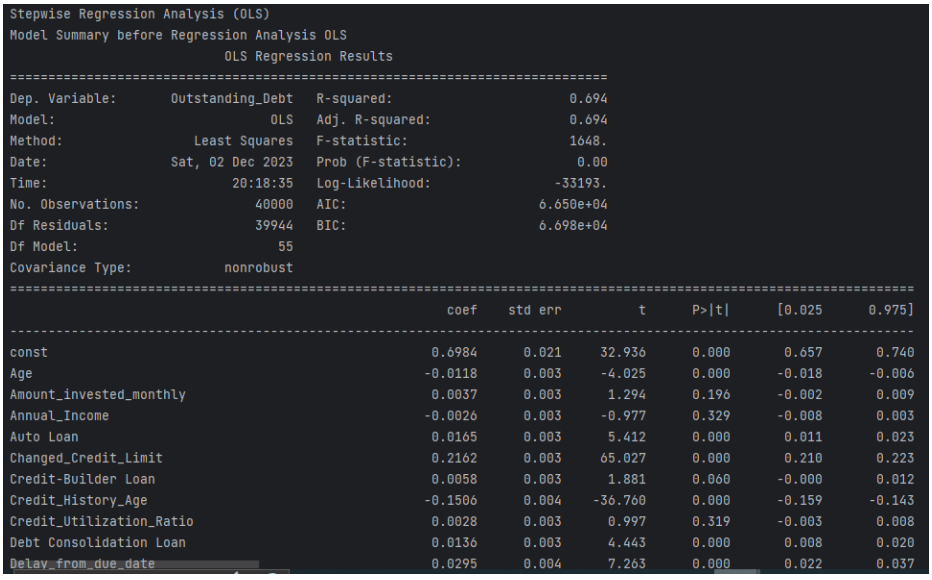
We are implementing backward stepwise regression to select significant features for our linear regression model.

a. The T-test analysis involves setting a threshold to the p-value and eliminate the features with a p-value greater than the threshold.

b. Here we set the threshold to 0.01.

c. Thus all features having p-value > 0.01 will be dropped.

We start with all 55 features and perform OLS on these features, we are left with 28 features.

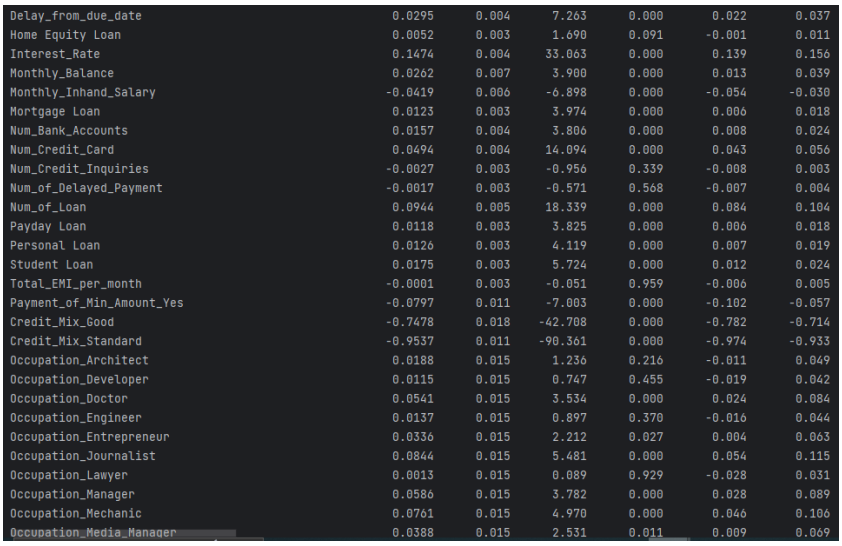


Stepwise Regression Analysis (OLS)
Model Summary before Regression Analysis OLS

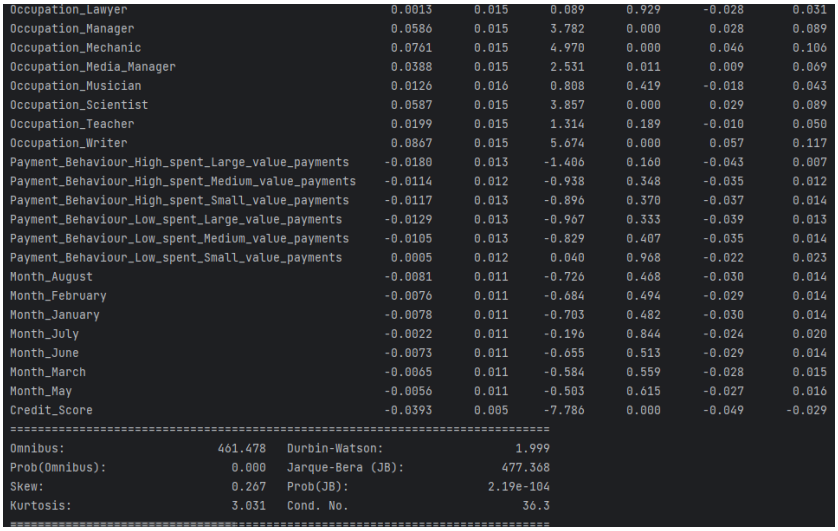
OLS Regression Results

Dep. Variable:	Outstanding_Debt	R-squared:	0.694
Model:	OLS	Adj. R-squared:	0.694
Method:	Least Squares	F-statistic:	1648.
Date:	Sat, 02 Dec 2023	Prob (F-statistic):	0.00
Time:	20:18:35	Log-Likelihood:	-33193.
No. Observations:	40000	AIC:	6.650e+04
Df Residuals:	39944	BIC:	6.698e+04
Df Model:	55		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.6984	0.021	32.936	0.000	0.657	0.740
Age	-0.0118	0.003	-4.025	0.000	-0.018	-0.006
Amount_invested_monthly	0.0037	0.003	1.294	0.196	-0.002	0.009
Annual_Income	-0.0026	0.003	-0.977	0.329	-0.008	0.003
Auto Loan	0.0165	0.003	5.412	0.000	0.011	0.023
Changed_Credit_Limit	0.2162	0.003	65.027	0.000	0.210	0.223
Credit-Builder Loan	0.0058	0.003	1.881	0.060	-0.000	0.012
Credit_History_Age	-0.1506	0.004	-36.760	0.000	-0.159	-0.143
Credit_Utilization_Ratio	0.0028	0.003	0.997	0.319	-0.003	0.008
Debt Consolidation Loan	0.0136	0.003	4.443	0.000	0.008	0.020
Delay_from_due_date	0.0295	0.004	7.263	0.000	0.022	0.037



Delay_from_due_date	0.0295	0.004	7.263	0.000	0.022	0.037
Home Equity Loan	0.0052	0.003	1.690	0.091	-0.001	0.011
Interest_Rate	0.1474	0.004	33.063	0.000	0.139	0.156
Monthly_Balance	0.0262	0.007	3.900	0.000	0.013	0.039
Monthly_Inhand_Salary	-0.0419	0.006	-6.898	0.000	-0.054	-0.030
Mortgage Loan	0.0123	0.003	3.974	0.000	0.006	0.018
Num_Bank_Accounts	0.0157	0.004	3.806	0.000	0.008	0.024
Num_Credit_Card	0.0494	0.004	14.094	0.000	0.043	0.056
Num_Credit_Inquiries	-0.0027	0.003	-0.956	0.339	-0.008	0.003
Num_of_Delayed_Payment	-0.0017	0.003	-0.571	0.568	-0.007	0.004
Num_of_Loan	0.0944	0.005	18.339	0.000	0.084	0.104
Payday Loan	0.0118	0.003	3.825	0.000	0.006	0.018
Personal Loan	0.0126	0.003	4.119	0.000	0.007	0.019
Student Loan	0.0175	0.003	5.724	0.000	0.012	0.024
Total_EMI_per_month	-0.0001	0.003	-0.051	0.959	-0.006	0.005
Payment_of_Min_Amount_Yes	-0.0797	0.011	-7.003	0.000	-0.102	-0.057
Credit_Mix_Good	-0.7478	0.018	-42.708	0.000	-0.782	-0.714
Credit_Mix_Standard	-0.9537	0.011	-90.361	0.000	-0.974	-0.933
Occupation_Architect	0.0188	0.015	1.236	0.216	-0.011	0.049
Occupation_Developer	0.0115	0.015	0.747	0.455	-0.019	0.042
Occupation_Doctor	0.0541	0.015	3.534	0.000	0.024	0.084
Occupation_Engineer	0.0137	0.015	0.897	0.370	-0.016	0.044
Occupation_Entrepreneur	0.0336	0.015	2.212	0.027	0.004	0.063
Occupation_Journalist	0.0844	0.015	5.481	0.000	0.054	0.115
Occupation_Lawyer	0.0013	0.015	0.089	0.929	-0.028	0.031
Occupation_Manager	0.0586	0.015	3.782	0.000	0.028	0.089
Occupation_Mechanic	0.0761	0.015	4.970	0.000	0.046	0.106
Occupation_Media_Manager	0.0388	0.015	2.531	0.011	0.009	0.069



Occupation_Lawyer	0.0013	0.015	0.089	0.929	-0.028	0.031
Occupation_Manager	0.0586	0.015	3.782	0.000	0.028	0.089
Occupation_Mechanic	0.0761	0.015	4.970	0.000	0.046	0.106
Occupation_Media_Manager	0.0388	0.015	2.531	0.011	0.009	0.069
Occupation_Musician	0.0126	0.016	0.808	0.419	-0.018	0.043
Occupation_Scientist	0.0587	0.015	3.857	0.000	0.029	0.089
Occupation_Teacher	0.0199	0.015	1.314	0.189	-0.010	0.050
Occupation_Writer	0.0867	0.015	5.674	0.000	0.057	0.117
Payment_Behaviour_High_spent_Large_value_payments	-0.0180	0.013	-1.406	0.160	-0.043	0.007
Payment_Behaviour_High_spent_Medium_value_payments	-0.0114	0.012	-0.938	0.348	-0.035	0.012
Payment_Behaviour_High_spent_Small_value_payments	-0.0117	0.013	-0.896	0.370	-0.037	0.014
Payment_Behaviour_Low_spent_Large_value_payments	-0.0129	0.013	-0.967	0.333	-0.039	0.013
Payment_Behaviour_Low_spent_Medium_value_payments	-0.0105	0.013	-0.829	0.407	-0.035	0.014
Payment_Behaviour_Low_spent_Small_value_payments	0.0005	0.012	0.040	0.968	-0.022	0.023
Month_August	-0.0081	0.011	-0.726	0.468	-0.030	0.014
Month_February	-0.0076	0.011	-0.684	0.494	-0.029	0.014
Month_January	-0.0078	0.011	-0.703	0.482	-0.030	0.014
Month_July	-0.0022	0.011	-0.196	0.844	-0.024	0.020
Month_June	-0.0073	0.011	-0.655	0.513	-0.029	0.014
Month_March	-0.0065	0.011	-0.584	0.559	-0.028	0.015
Month_May	-0.0056	0.011	-0.503	0.615	-0.027	0.016
Credit_Score	-0.0393	0.005	-7.786	0.000	-0.049	-0.029

Omnibus:	461.478	Durbin-Watson:	1.999
Prob(Omnibus):	0.000	Jarque-Bera (JB):	477.368
Skew:	0.267	Prob(JB):	2.19e-104
Kurtosis:	3.031	Cond. No.	36.3

Note:

1. F-statistic: We have a value of 1648 to start with when we include all the features.
2. The condition number also starts at 36.3 when we start with all the features.
3. The max p-value is also at 0.968 for the first linear model.

After completing regression analysis:

```
Regression Analysis complete
                            OLS Regression Results
==============================================================================
Dep. Variable:       Outstanding_Debt   R-squared:                       0.694
Model:                            OLS   Adj. R-squared:                  0.694
Method:                 Least Squares   F-statistic:                     3240.
Date:                Mon, 27 Nov 2023   Prob (F-statistic):               0.00
Time:                        15:37:23   Log-Likelihood:                -33214.
No. Observations:               40000   AIC:                         6.649e+04
Df Residuals:                   39971   BIC:                         6.674e+04
Df Model:                          28
Covariance Type:            nonrobust
================================================================================================
                                   coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------------------------
const                            0.7026      0.014     50.029      0.000       0.675       0.730
Age                             -0.0120      0.003     -4.160      0.000      -0.018      -0.006
Auto Loan                        0.0152      0.003      4.969      0.000       0.009       0.021
Changed_Credit_Limit             0.2130      0.003     64.053      0.000       0.207       0.220
Credit-Builder Loan              0.0093      0.003      3.023      0.003       0.003       0.015
Credit_History_Age              -0.1513      0.004    -36.983      0.000      -0.159      -0.143
Debt Consolidation Loan          0.0176      0.003      5.727      0.000       0.012       0.024
Delay_from_due_date              0.0350      0.004      8.649      0.000       0.027       0.043
Home Equity Loan                 0.0085      0.003      2.761      0.006       0.002       0.014
Interest_Rate                    0.1442      0.004     32.504      0.000       0.135       0.153
Monthly_Balance                  0.0248      0.007      3.720      0.000       0.012       0.038
Monthly_Inhand_Salary           -0.0408      0.006     -6.783      0.000      -0.053      -0.029
Mortgage Loan                    0.0177      0.003      5.747      0.000       0.012       0.024
```

```
Mortgage Loan                    0.0177      0.003      5.747      0.000       0.012       0.024
Num_Bank_Accounts                0.0122      0.004      2.939      0.003       0.004       0.020
Num_Credit_Card                  0.0506      0.003     14.469      0.000       0.044       0.057
Num_of_Loan                      0.0835      0.005     16.365      0.000       0.074       0.094
Payday Loan                      0.0113      0.003      3.705      0.000       0.005       0.017
Personal Loan                    0.0144      0.003      4.739      0.000       0.008       0.020
Student Loan                     0.0187      0.003      6.128      0.000       0.013       0.025
Payment_of_Min_Amount_Yes       -0.0677      0.011     -6.043      0.000      -0.090      -0.046
Credit_Mix_Good                 -0.7445      0.018    -42.521      0.000      -0.779      -0.710
Credit_Mix_Standard             -0.9577      0.011    -89.794      0.000      -0.979      -0.937
Occupation_Doctor                0.0419      0.011      3.717      0.000       0.020       0.064
Occupation_Journalist            0.0553      0.011      4.838      0.000       0.033       0.078
Occupation_Manager               0.0413      0.012      3.527      0.000       0.018       0.064
Occupation_Mechanic              0.0577      0.011      5.042      0.000       0.035       0.080
Occupation_Scientist             0.0540      0.011      4.786      0.000       0.032       0.076
Occupation_Writer                0.0556      0.011      4.915      0.000       0.033       0.078
Credit_Score                    -0.0380      0.005     -7.565      0.000      -0.048      -0.028
==============================================================================
Omnibus:                      419.386   Durbin-Watson:                   2.015
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              432.471
Skew:                           0.254   Prob(JB):                     1.23e-94
Kurtosis:                       3.030   Cond. No.                         19.7
```

Features Dropped:

Drop Feature	AIC	BIC	AdjR2	p-value	f-statistic
Payment_Behaviour_Low_spent_Small_value_payments	66498.532	66979.943	0.694	0.968	1648.362
Total_EMI_per_month	66496.533	66969.348	0.694	0.959	1678.929
Occupation_Lawyer	66494.536	66958.754	0.694	0.929	1710.65
Month_July	66492.544	66948.165	0.694	0.845	1743.59
Month_May	66490.582	66937.607	0.694	0.639	1777.82
Month_March	66488.802	66927.231	0.694	0.667	1813.408
Month_June	66486.988	66916.819	0.694	0.672	1850.45
Month_February	66485.167	66906.403	0.694	0.701	1889.036
Month_January	66483.315	66895.954	0.694	0.726	1929.267
Month_August	66481.438	66885.48	0.694	0.738	1971.248
Num_of_Delayed_Payment	66479.55	66874.995	0.694	0.566	2015.096
Occupation_Developer	66477.88	66864.728	0.694	0.41	2060.92
Occupation_Musician	66476.56	66854.812	0.694	0.52	2108.85
Occupation_Engineer	66474.974	66844.629	0.694	0.523	2159.082
Num_Credit_Inquiries	66473.383	66834.442	0.694	0.347	2211.766
Occupation_Architect	66472.267	66824.729	0.694	0.34	2267.044
Occupation_Teacher	66471.179	66815.044	0.694	0.359	2325.155
Credit_Utilization_Ratio	66470.021	66805.29	0.694	0.317	2386.331
Annual_Income	66469.024	66795.696	0.694	0.317	2450.799
Payment_Behaviour_Low_spent_Medium_value_payments	66468.024	66786.1	0.694	0.23	2518.849
Payment_Behaviour_High_spent_Small_value_payments	66467.465	66776.944	0.694	0.342	2590.747
Payment_Behaviour_High_spent_Medium_value_payments	66466.369	66767.251	0.694	0.417	2666.925
Payment_Behaviour_Low_spent_Large_value_payments	66465.029	66757.315	0.694	0.451	2747.745
Payment_Behaviour_High_spent_Large_value_payments	66463.597	66747.286	0.694	0.245	2833.625
Amount_invested_monthly	66462.952	66738.045	0.694	0.187	2924.962
Home Equity Loan	66462.698	66729.193	0.694	0.092	3022.347
Credit-Builder Loan	66463.537	66721.436	0.694	0.1	3126.324
Occupation_Entrepreneur	66464.237	66713.54	0.694	0.055	3237.744
Occupation_Media_Manager	66465.929	66706.635	0.694	0.028	3357.298

Final Adj R-squared: 0.694
Final F-statistic: 3485.909

Multiple Linear Regression Results:

Multiple Linear Regression Result							
Model	R-squared	AIC	BIC	Adj_R2	MSE	RMSE	MAE
Multiple Linear Regression	0.694	66468.731	66700.84	0.694	434095.002	658.859	527.816
Multiple Linear Regression(RForest)	0.675	68886.411	69109.924	0.675	461433.529	679.289	541.695

Based on this, we can say that the stepwise regression analysis with OLS performs better, we will thus select this method to build linear regression model.

We can plot train, test and predicted values in one plot to analyze the modeling results of our final regression model.

We need to reverse transform the test and predicted variables to include original values.

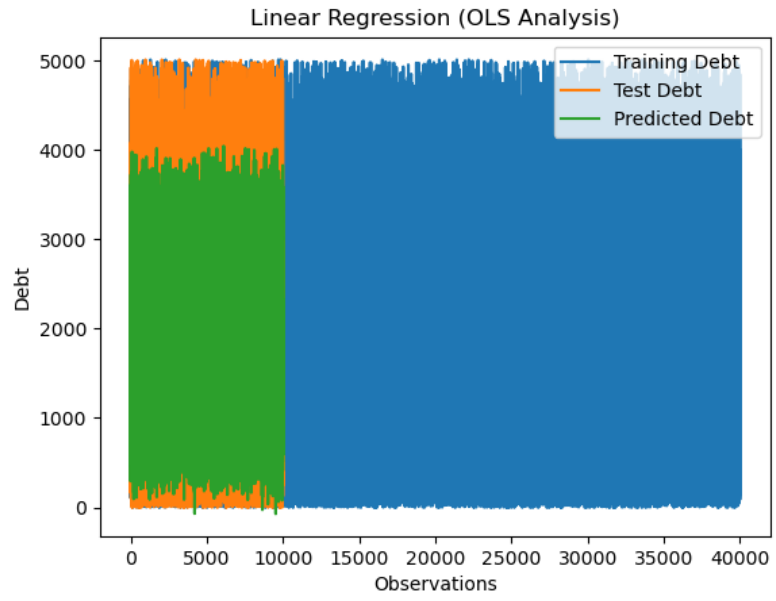


Figure 14: Linear Regression Predicted, Train, Test

Confidence Interval Analysis:

```
--------------------Prediction Intervals--------------------
           mean   mean_se  mean_ci_lower  mean_ci_upper  obs_ci_lower  \
19616 -0.079302  0.012515      -0.103832      -0.054773     -1.168001
27166 -1.008259  0.014018      -1.035735      -0.980782     -2.097028
7373   1.770842  0.017465       1.736610       1.805074      0.681882
37034 -0.575665  0.015243      -0.605543      -0.545788     -1.664498
3372  -0.765431  0.012034      -0.789017      -0.741844     -1.854109

       obs_ci_upper
19616      1.009397
27166      0.080511
7373       2.859803
37034      0.513167
3372       0.323248
```

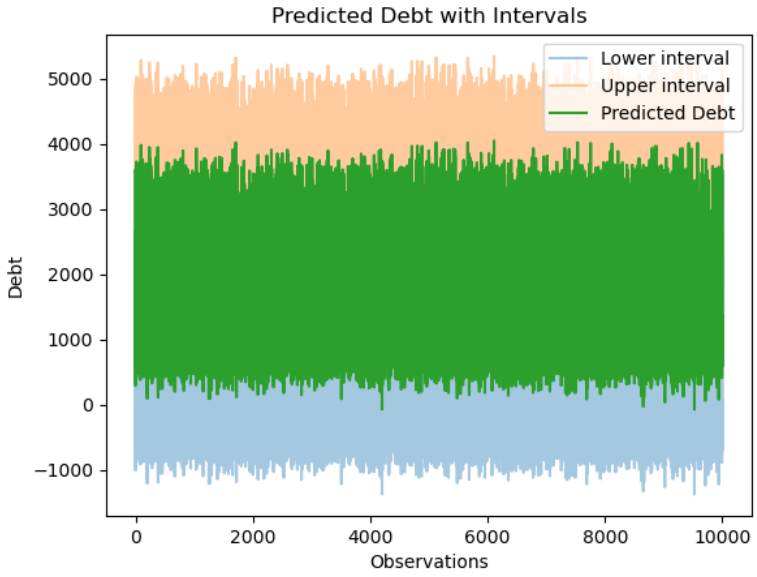


Figure 15: Prediction Interval

--------Confidence Interval Analysis----------
Confidence Intervals:

	0	1
const	0.673742	0.728803
Age	-0.017664	-0.006221
Auto Loan	0.009078	0.020752
Changed_Credit_Limit	0.209649	0.222661
Credit_History_Age	-0.158962	-0.142924
Debt Consolidation Loan	0.006028	0.017752
Delay_from_due_date	0.021860	0.037746
Interest_Rate	0.139080	0.156509
Monthly_Balance	0.010475	0.036374
Monthly_Inhand_Salary	-0.052305	-0.028952
Mortgage Loan	0.004668	0.016408
Num_Bank_Accounts	0.007627	0.023780
Num_Credit_Card	0.042589	0.056313
Num_of_Loan	0.090030	0.108452
Payday Loan	0.004102	0.015800
Personal Loan	0.004831	0.016511
Student Loan	0.009816	0.021520
Payment_of_Min_Amount_Yes	-0.101862	-0.057280
Credit_Mix_Good	-0.781695	-0.713153
Credit_Mix_Standard	-0.974482	-0.933171
Occupation_Doctor	0.015198	0.059753
Occupation_Journalist	0.045324	0.090306
Occupation_Manager	0.018765	0.064216
Occupation_Mechanic	0.037045	0.081555
Occupation_Scientist	0.019719	0.063857
Occupation_Writer	0.047778	0.092186
Credit_Score	-0.049430	-0.029711

Phase 3: Classification Analysis

In this phase, we will predict Credit_Score into 3 categories: Good, Standard and Poor.

The results obtained in this report are based on dataset containing 50806 instances. The code files attached will contain code for down-sampled data for approximately 12k records which completes execution in approximately 20 minutes. (As suggested by professor)

We will implement the following classifiers:

1) Decision Tree Classifier
2) Logistic Regression
3) KNN
4) SVM (Linear, Poly, RBF)
5) Naïve Bayes Classifier
6) Random Forest (Stacking, Bagging, Boosting)
7) Neural Network (MLP Classifier)

As we predict credit score in three categories, we will evaluate the classifiers with One v/s One and the One v/s Rest approaches.

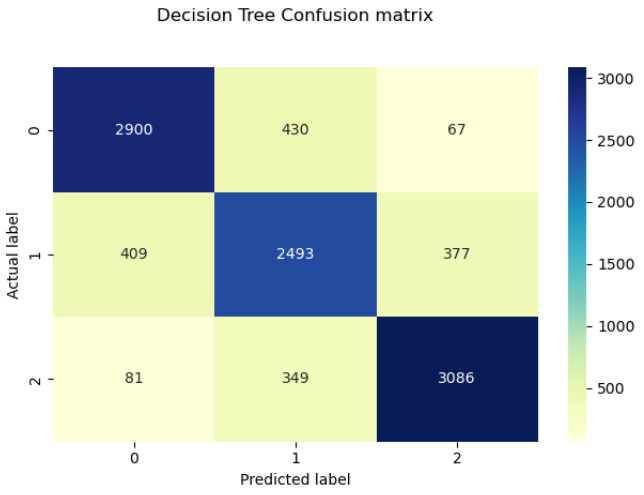
We will evaluate these with the following metrics:

1) Confusion Matrix
2) Precision
3) Recall or Sensitivity
4) Specificity
5) F1-score
6) ROC-AUC
7) Stratified K-Fold Cross Validation

Decision Tree Classifier

```
-----------------Simple Decision Tree-----------------
Accuracy of train: 1
Accuracy of test: 0.83
-------------- Evaluating for model Decision Tree --------------
Confusion Matrix for Decision Tree is:
[[2900  430   67]
 [ 409 2493  377]
 [  81  349 3086]]
Macro Precision for DecisionTreeClassifier(random_state=5805) is: 0.831
Micro Precision for DecisionTreeClassifier(random_state=5805) is: 0.832

Macro Recall for DecisionTreeClassifier(random_state=5805) is: 0.831
Micro Recall for DecisionTreeClassifier(random_state=5805) is: 0.832

Macro F1 for DecisionTreeClassifier(random_state=5805) is: 0.831
Micro F1 for DecisionTreeClassifier(random_state=5805) is: 0.832

Macro-specificity for Decision Tree is: 0.916
Macro-averaged One v/s Rest ROC AUC score: 0.873
Micro-averaged One v/s Rest ROC AUC score: 0.874
Macro AUC (OvO) for model Decision Tree is: 0.873
```

```
---------- Stratified K-Fold CV for Decision Tree ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for Decision Tree = 0.827
Macro averaged Recall Score for Decision Tree = 0.827
Macro averaged F1 Score for Decision Tree = 0.827
```



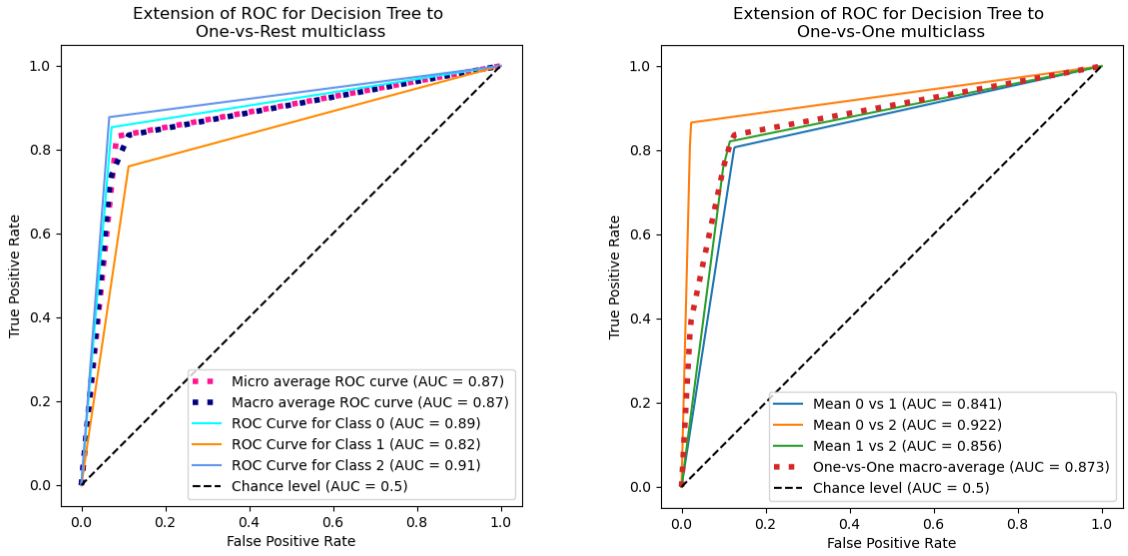


Figure 16: Decision Tree ROC-AUC

Pre-Pruned Decision Tree Classifier

The best_params = {max_depth = 15, max_features='sqrt', min_samples_leaf=5, min_samples_split =15}

For decision tree having tried for max features 5, 10, 15 the results of AUC score and other metrics were not improving much. For sqrt as max features the AUC score has improved.

```
-----------------Decision Tree Pre-Pruned-----------------
------------- Evaluating for model Pre-pruned Decision Tree -------------
Confusion Matrix for Pre-pruned Decision Tree is:
[[2839  368  190]
 [ 548 2236  495]
 [ 144  437 2935]]
Macro Precision for DecisionTreeClassifier(max_depth=15, max_features='sqrt', min_samples_leaf=5,
                      min_samples_split=15, random_state=5805) is: 0.783
Micro Precision for DecisionTreeClassifier(max_depth=15, max_features='sqrt', min_samples_leaf=5,
                      min_samples_split=15, random_state=5805) is: 0.786

Macro Recall for DecisionTreeClassifier(max_depth=15, max_features='sqrt', min_samples_leaf=5,
                      min_samples_split=15, random_state=5805) is: 0.784
Micro Recall for DecisionTreeClassifier(max_depth=15, max_features='sqrt', min_samples_leaf=5,
                      min_samples_split=15, random_state=5805) is: 0.786

Macro F1 for DecisionTreeClassifier(max_depth=15, max_features='sqrt', min_samples_leaf=5,
                      min_samples_split=15, random_state=5805) is: 0.784
Micro F1 for DecisionTreeClassifier(max_depth=15, max_features='sqrt', min_samples_leaf=5,
                      min_samples_split=15, random_state=5805) is: 0.786

Macro-specificity for Pre-pruned Decision Tree is: 0.893
Macro-averaged One v/s Rest ROC AUC score: 0.897
Micro-averaged One v/s Rest ROC AUC score: 0.903
Macro AUC (OvO) for model Pre-pruned Decision Tree is: 0.897
```

```
---------- Stratified K-Fold CV for Pre-pruned Decision Tree ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for Pre-pruned Decision Tree = 0.782
Macro averaged Recall Score for Pre-pruned Decision Tree = 0.782
Macro averaged F1 Score for Pre-pruned Decision Tree = 0.781
```

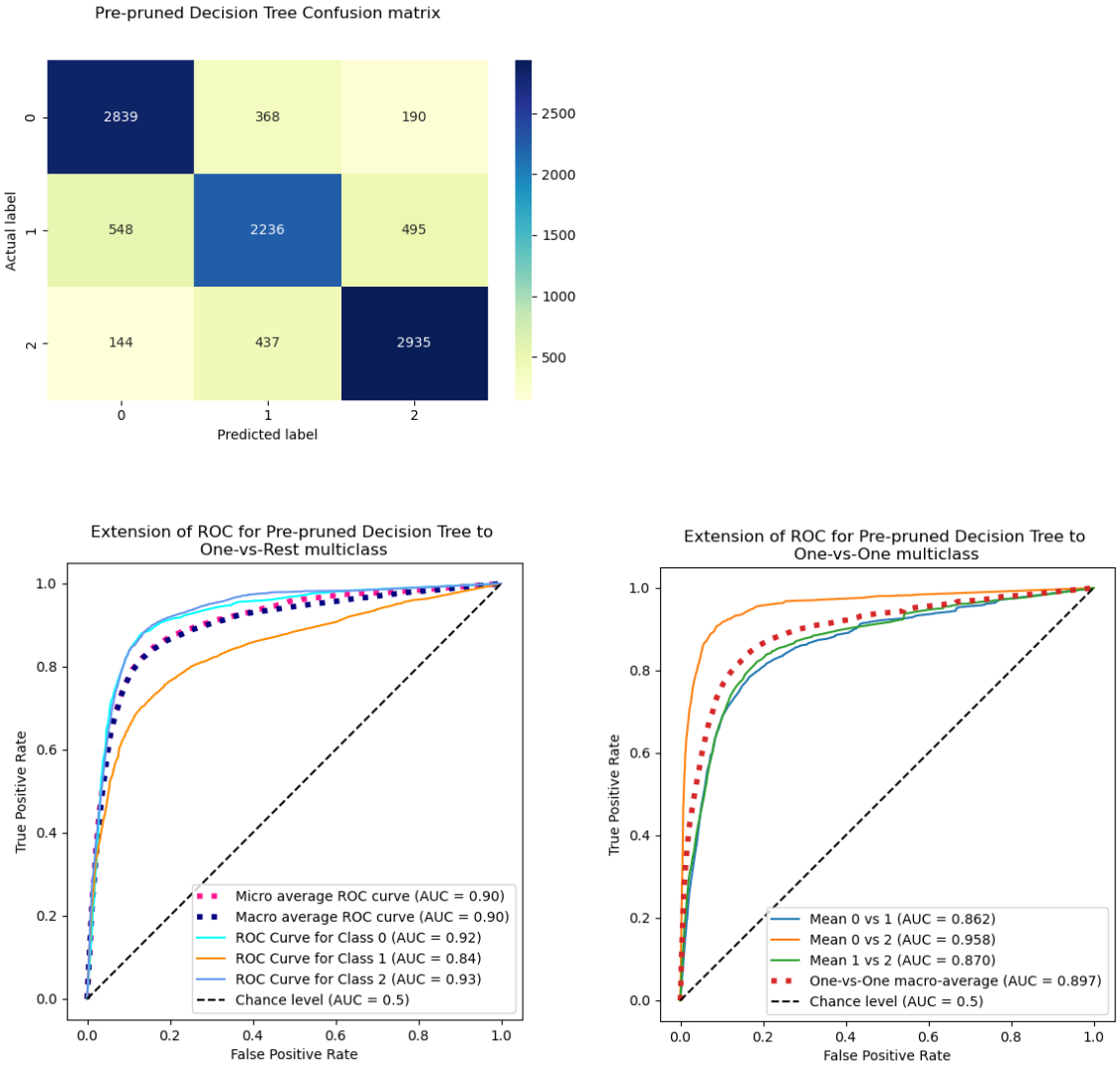


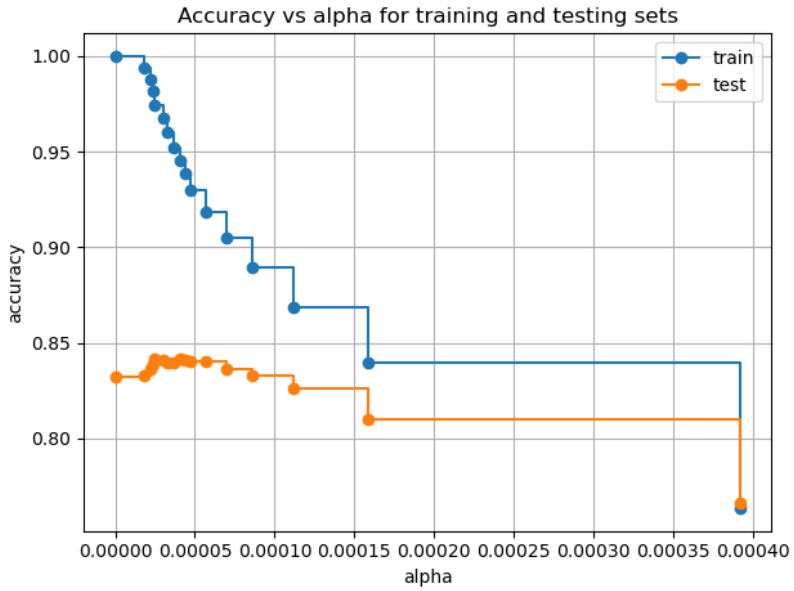
Figure 17: Pre-pruned Decision Tree Classifier ROC-AUC

Post-pruned Decision Tree Classifier

Optimal alpha: 0.0000325

Experiment with alphas

Overall alphas for optimum alpha selection
Selecting best alpha by iterating in steps of 200:




```
-------------- Evaluating for model Post-Pruned Decision Tree --------------
Confusion Matrix for Post-Pruned Decision Tree is:
[[2960  362   75]
 [ 426 2477  376]
 [  80  294 3142]]
Macro Precision for DecisionTreeClassifier(ccp_alpha=3.25e-05, random_state=5805) is: 0.84
Micro Precision for DecisionTreeClassifier(ccp_alpha=3.25e-05, random_state=5805) is: 0.842

Macro Recall for DecisionTreeClassifier(ccp_alpha=3.25e-05, random_state=5805) is: 0.84
Micro Recall for DecisionTreeClassifier(ccp_alpha=3.25e-05, random_state=5805) is: 0.842

Macro F1 for DecisionTreeClassifier(ccp_alpha=3.25e-05, random_state=5805) is: 0.84
Micro F1 for DecisionTreeClassifier(ccp_alpha=3.25e-05, random_state=5805) is: 0.842

Macro-specificity for Post-Pruned Decision Tree is: 0.921
Macro-averaged One v/s Rest ROC AUC score: 0.882
Micro-averaged One v/s Rest ROC AUC score: 0.888
Macro AUC (OvO) for model Post-Pruned Decision Tree is: 0.881
```

```
---------- Stratified K-Fold CV for Post-pruned Decision Tree ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for Post-pruned Decision Tree = 0.835
Macro averaged Recall Score for Post-pruned Decision Tree = 0.836
Macro averaged F1 Score for Post-pruned Decision Tree = 0.835
```

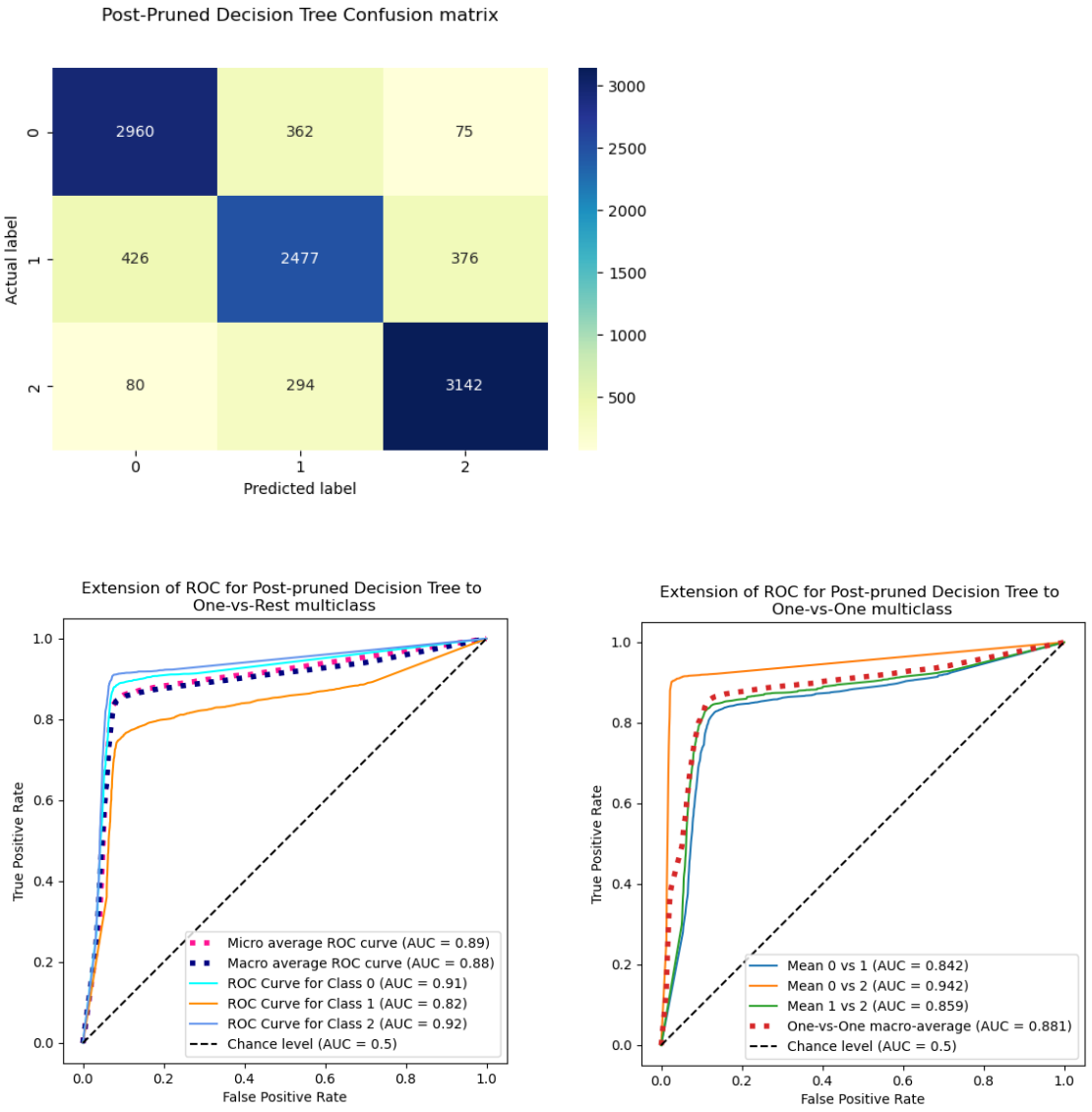


Figure 18: Post-pruned Decision Tree ROC-AUC

ROC-AUC of One v/s Rest approach:

AUC of Class 0: 0.91

ROC-AUC of One v/s One approach:

Mean 0 v/s 1: 0.842

AUC of Class 1: 0.82

AUC of Class 2: 0.92

Micro average AUC: 0.89

Macro average AUC: 0.88

Mean 0 v/s 2: 0.942

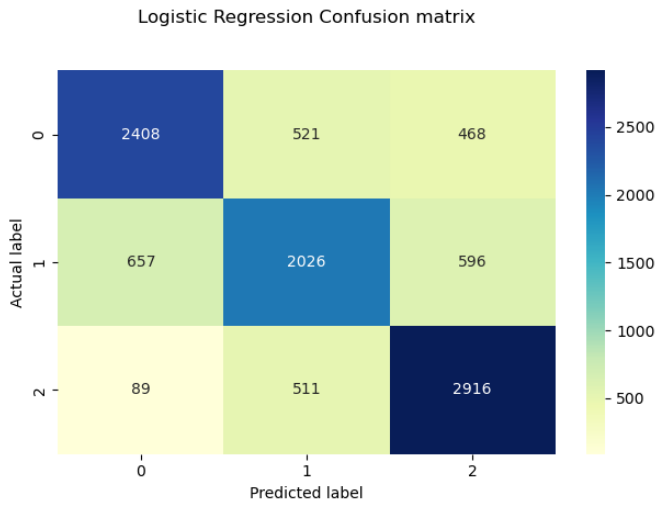
Mean 1 v/s 2: 0.859

Macro One v/s One: 0.881

Logistic Regression

```
----------------Logistic Regression-----------------
------------- Evaluating for model Logistic Regression -------------
Confusion Matrix for Logistic Regression is:
[[2408  521  468]
 [ 657 2026  596]
 [  89  511 2916]]
Macro Precision for LogisticRegression(multi_class='ovr', random_state=5805) is: 0.72
Micro Precision for LogisticRegression(multi_class='ovr', random_state=5805) is: 0.721

Macro Recall for LogisticRegression(multi_class='ovr', random_state=5805) is: 0.719
Micro Recall for LogisticRegression(multi_class='ovr', random_state=5805) is: 0.721

Macro F1 for LogisticRegression(multi_class='ovr', random_state=5805) is: 0.719
Micro F1 for LogisticRegression(multi_class='ovr', random_state=5805) is: 0.721

Macro-specificity for Logistic Regression is: 0.861
Macro-averaged One v/s Rest ROC AUC score: 0.839
Micro-averaged One v/s Rest ROC AUC score: 0.845
Macro AUC (OvO) for model Logistic Regression is: 0.838
```

```
---------- Stratified K-Fold CV for Logistic Regression ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for Logistic Regression = 0.717
Macro averaged Recall Score for Logistic Regression = 0.716
Macro averaged F1 Score for Logistic Regression = 0.715
```



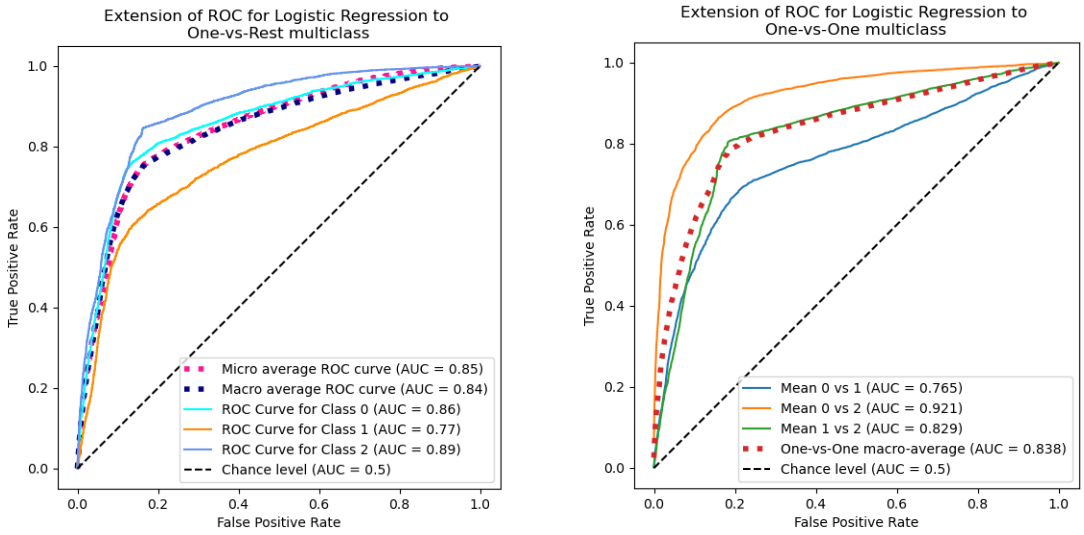


Figure 19: Logistic Regression ROC-AUC

ROC-AUC of One v/s Rest approach:

AUC of Class 0: 0.86

AUC of Class 1: 0.77

AUC of Class 2: 0.89

Micro average AUC: 0.85

Macro average AUC: 0.84

ROC-AUC of One v/s One approach:

Mean 0 v/s 1: 0.765

Mean 0 v/s 2: 0.921

Mean 1 v/s 2: 0.829

Macro One v/s One: 0.838

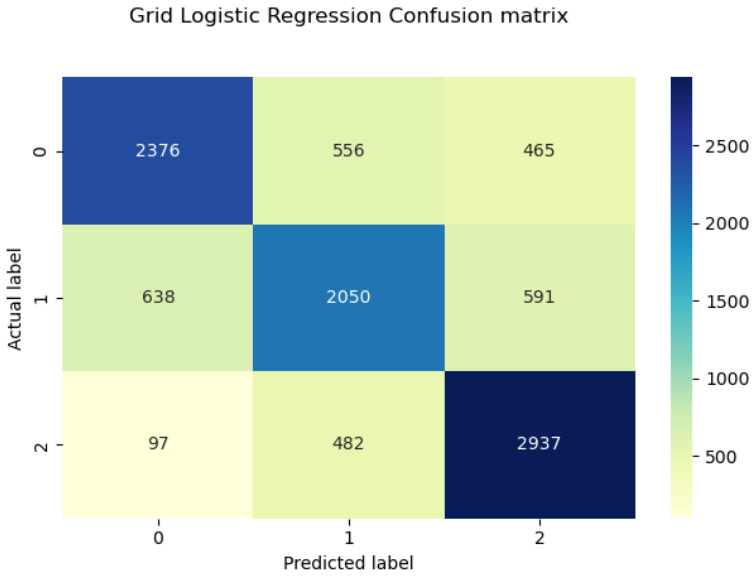
Grid Search Logistic Regression

```
Best Parameters for Logistic Regression: {'C': 2.0, 'solver': 'lbfgs'}
```

```
------------- Evaluating for model Grid Logistic Regression -------------
Confusion Matrix for Grid Logistic Regression is:
[[2376  556  465]
 [ 638 2050  591]
 [  97  482 2937]]
Macro Precision for LogisticRegression(random_state=5805) is: 0.721
Micro Precision for LogisticRegression(random_state=5805) is: 0.722

Macro Recall for LogisticRegression(random_state=5805) is: 0.72
Micro Recall for LogisticRegression(random_state=5805) is: 0.722

Macro F1 for LogisticRegression(random_state=5805) is: 0.72
Micro F1 for LogisticRegression(random_state=5805) is: 0.722

Macro-specificity for Grid Logistic Regression is: 0.861
Macro-averaged One v/s Rest ROC AUC score: 0.844
Micro-averaged One v/s Rest ROC AUC score: 0.849
Macro AUC (OvO) for model Grid Logistic Regression is: 0.843
```

```
---------- Stratified K-Fold CV for Grid Logistic Regression ----------
-----------------Stratified K-Fold CV Results-----------------
Macro averaged Precision Score for Grid Logistic Regression = 0.719
Macro averaged Recall Score for Grid Logistic Regression = 0.718
Macro averaged F1 Score for Grid Logistic Regression = 0.717
```

After Grid Search, we see that we have performance improvements in the k-fold cross validation for logistic regression.



We can see in the confusion matrix, we have a rise in the correctly classified labels for Class 1 and Class 2.

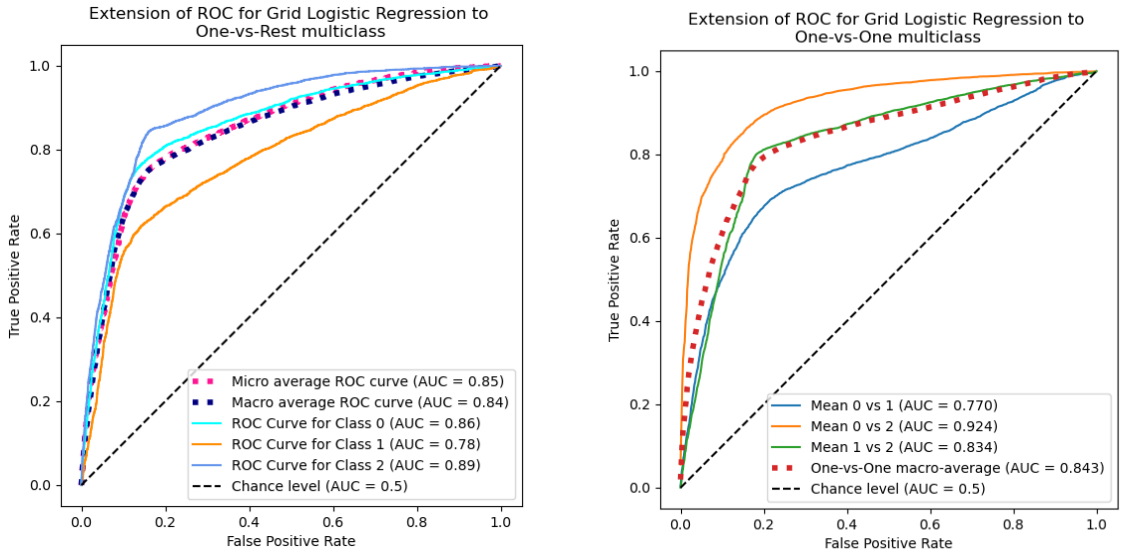


Figure 20: Grid Search Logistic Regression ROC-AUC

ROC-AUC of One v/s Rest approach:

AUC of Class 0: 0.86

AUC of Class 1: 0.78

AUC of Class 2: 0.89

Micro average AUC: 0.85

Macro average AUC: 0.84

ROC-AUC of One v/s One approach:

Mean 0 v/s 1: 0.770

Mean 0 v/s 2: 0.924

Mean 1 v/s 2: 0.834

Macro One v/s One: 0.843

We can see that after Grid Search, Logistic Regression is able to perform better for class 1 in One v/s Rest approach, also improvements can be seen in the One v/s One approach.

KNN (K-Nearest-Neighbors)

```
-----------------K-Nearest Neighbors(KNN)------------------
--------------Results before Optimal K--------------
-------------- Evaluating for model KNN --------------
Confusion Matrix for KNN is:
[[2943  312  142]
 [ 605 2078  596]
 [  86  230 3200]]
Macro Precision for KNeighborsClassifier() is: 0.805
Micro Precision for KNeighborsClassifier() is: 0.807

Macro Recall for KNeighborsClassifier() is: 0.803
Micro Recall for KNeighborsClassifier() is: 0.807

Macro F1 for KNeighborsClassifier() is: 0.803
Micro F1 for KNeighborsClassifier() is: 0.807

Macro-specificity for KNN is: 0.903
Macro-averaged One v/s Rest ROC AUC score: 0.918
Micro-averaged One v/s Rest ROC AUC score: 0.922
Macro AUC (OvO) for model KNN is: 0.917
```

```
---------- Stratified K-Fold CV for KNN ----------
-----------------Stratified K-Fold CV Results-----------------
Macro averaged Precision Score for KNN = 0.806
Macro averaged Recall Score for KNN = 0.804
Macro averaged F1 Score for KNN = 0.801
```

As our dataset is balanced, macro and micro averages of the evaluation metrics, have almost similar values.

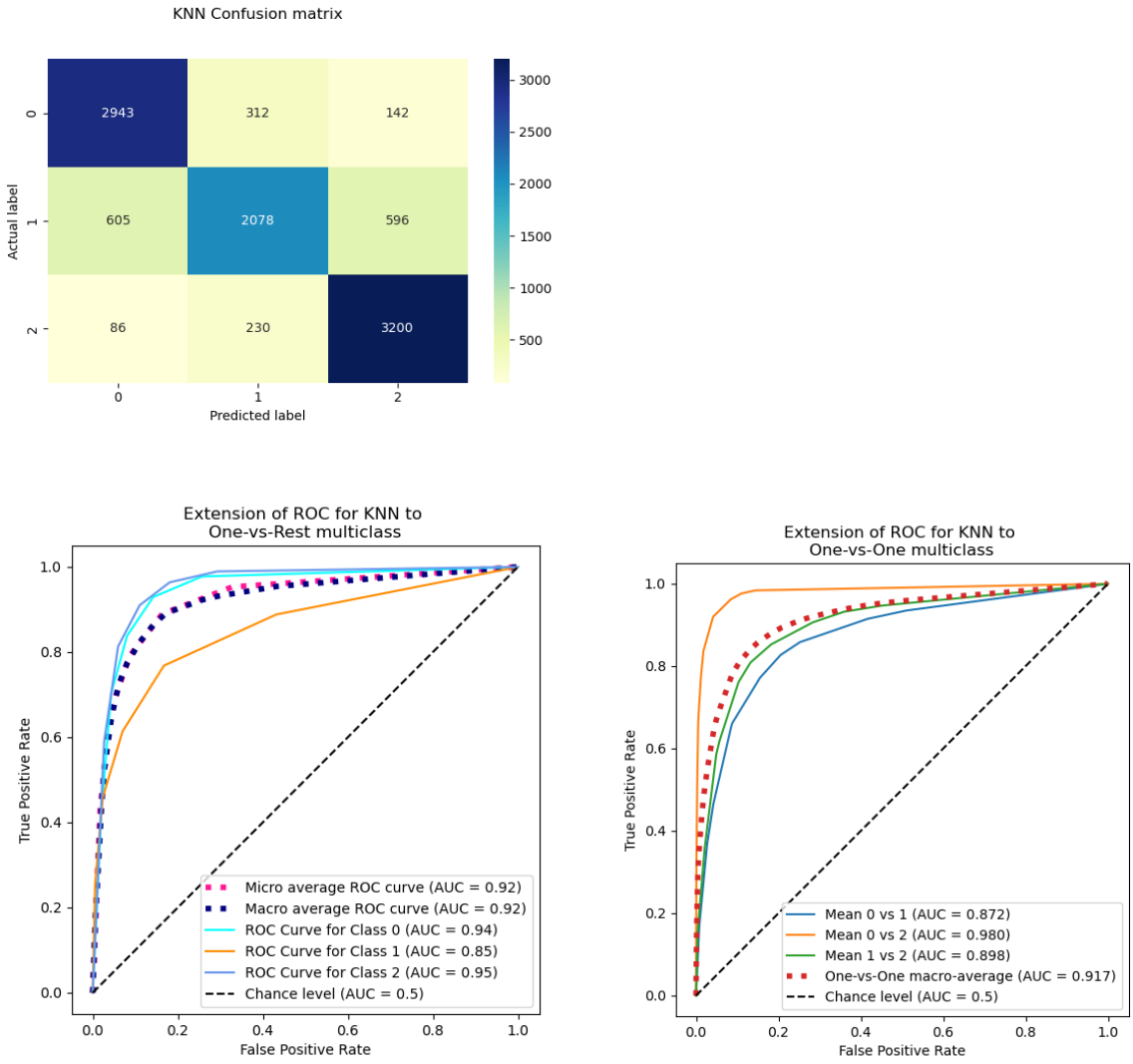


Figure 21: KNN ROC-AUC

ROC-AUC of One v/s Rest approach:

AUC of Class 0: 0.94

AUC of Class 1: 0.85

AUC of Class 2: 0.95

Micro average AUC: 0.92

Macro average AUC: 0.92

ROC-AUC of One v/s One approach:

Mean 0 v/s 1: 0.872

Mean 0 v/s 2: 0.980

Mean 1 v/s 2: 0.898

Macro One v/s One: 0.917

KNN with Optimal K

Grid Search for K

```
----------Grid Search for KNN Classifier----------
Fitting 2 folds for each of 19 candidates, totalling 38 fits
The best k after grid search is : {'n_neighbors': 3}
```

Evaluating for Optimal K:

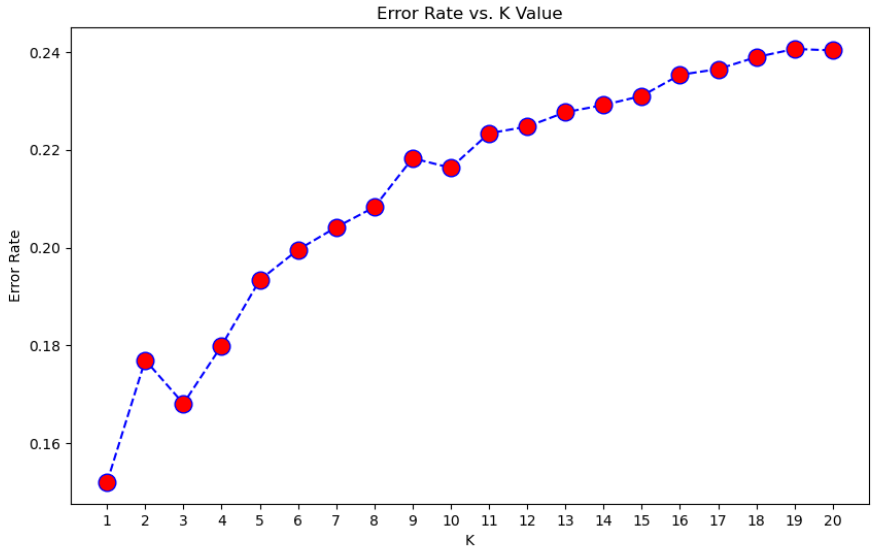


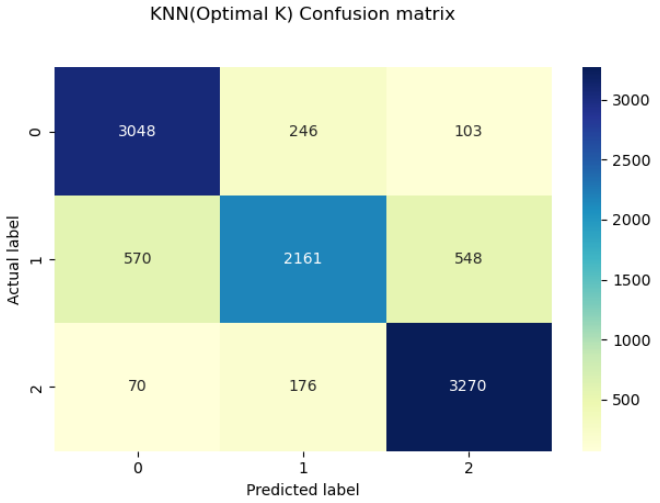
Figure 22: KNN Elbow Method

Based on the Grid Search and Elbow method for finding optimal K, we select K=3.

Based on the results below, we can see an improvement in the model performance by selecting optimal K = 3.

```
-----------------Results after selecting Optimal K-----------------
Optimal K: 3
------------- Evaluating for model KNN(Optimal K) -------------
Confusion Matrix for KNN(Optimal K) is:
[[3048  246  103]
 [ 570 2161  548]
 [  70  176 3270]]
Macro Precision for KNeighborsClassifier(n_neighbors=3) is: 0.832
Micro Precision for KNeighborsClassifier(n_neighbors=3) is: 0.832

Macro Recall for KNeighborsClassifier(n_neighbors=3) is: 0.829
Micro Recall for KNeighborsClassifier(n_neighbors=3) is: 0.832

Macro F1 for KNeighborsClassifier(n_neighbors=3) is: 0.829
Micro F1 for KNeighborsClassifier(n_neighbors=3) is: 0.832

Macro-specificity for KNN(Optimal K) is: 0.916
Macro-averaged One v/s Rest ROC AUC score: 0.918
Micro-averaged One v/s Rest ROC AUC score: 0.922
Macro AUC (OvO) for model KNN(Optimal K) is: 0.918
```

```
---------- Stratified K-Fold CV for KNN(Optimal K) Classifier ----------
-----------------Stratified K-Fold CV Results-----------------
Macro averaged Precision Score for KNN(Optimal K) Classifier = 0.825
Macro averaged Recall Score for KNN(Optimal K) Classifier = 0.823
Macro averaged F1 Score for KNN(Optimal K) Classifier = 0.819
```



The number of correctly classified labels have increased for each class after selecting optimal K for KNN.

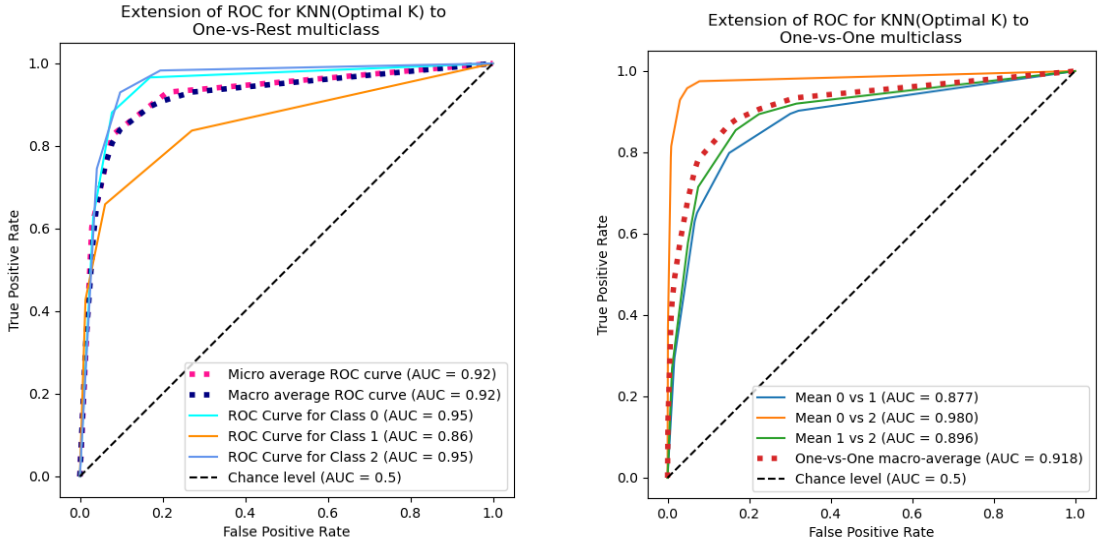


Figure 23: KNN (Optimal K) with ROC-AUC

ROC-AUC of One v/s Rest approach:

AUC of Class 0: 0.94

AUC of Class 1: 0.86

AUC of Class 2: 0.95

Micro average AUC: 0.92

Macro average AUC: 0.92

ROC-AUC of One v/s One approach:

Mean 0 v/s 1: 0.877

Mean 0 v/s 2: 0.980

Mean 1 v/s 2: 0.896

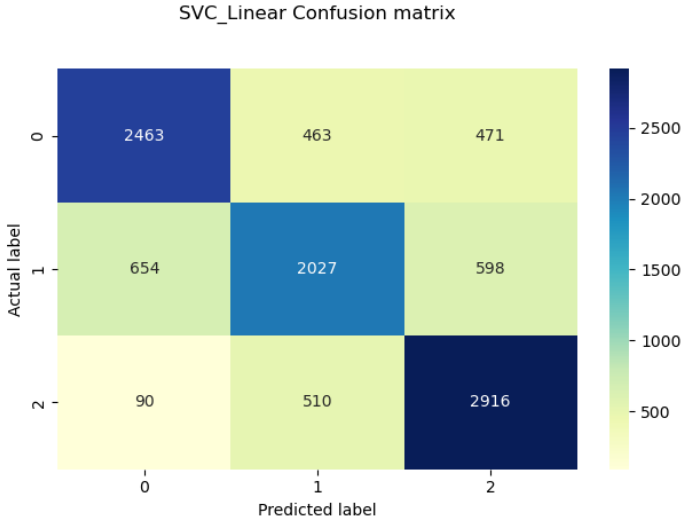
Macro One v/s One: 0.918

We can see that KNN classifier improves after selecting optimal K.

SVM (Linear Kernel Classifier)

```
-------------- Evaluating for model SVC_Linear --------------
Confusion Matrix for SVC_Linear is:
[[2463  463  471]
 [ 654 2027  598]
 [  90  510 2916]]
Macro Precision for SVC(C=0.2, kernel='linear', probability=True, random_state=5805) is: 0.725
Micro Precision for SVC(C=0.2, kernel='linear', probability=True, random_state=5805) is: 0.727

Macro Recall for SVC(C=0.2, kernel='linear', probability=True, random_state=5805) is: 0.724
Micro Recall for SVC(C=0.2, kernel='linear', probability=True, random_state=5805) is: 0.727

Macro F1 for SVC(C=0.2, kernel='linear', probability=True, random_state=5805) is: 0.724
Micro F1 for SVC(C=0.2, kernel='linear', probability=True, random_state=5805) is: 0.727

Macro-specificity for SVC_Linear is: 0.863
Macro-averaged One v/s Rest ROC AUC score: 0.826
Micro-averaged One v/s Rest ROC AUC score: 0.832
Macro AUC (OvO) for model SVC_Linear is: 0.825
```

```
---------- Stratified K-Fold CV for SVC_Linear ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for SVC_Linear = 0.721
Macro averaged Recall Score for SVC_Linear = 0.721
Macro averaged F1 Score for SVC_Linear = 0.72
```



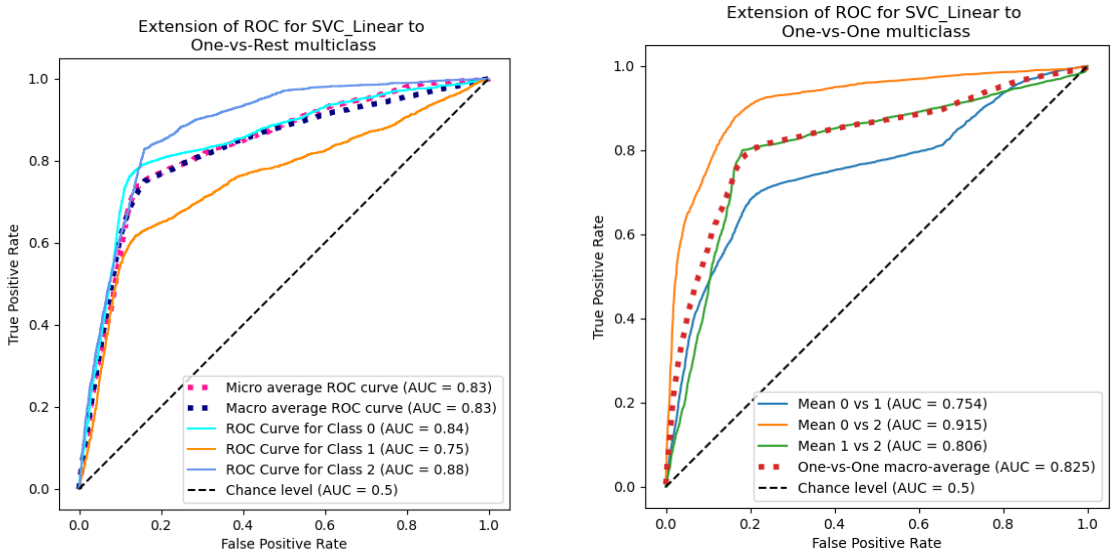


Figure 24: SVM Linear ROC-AUC

ROC-AUC of One v/s Rest approach:

AUC of Class 0: 0.84

AUC of Class 1: 0.75

AUC of Class 2: 0.88

Micro average AUC: 0.83

Macro average AUC: 0.83

ROC-AUC of One v/s One approach:

Mean 0 v/s 1: 0.877

Mean 0 v/s 2: 0.980

Mean 1 v/s 2: 0.896

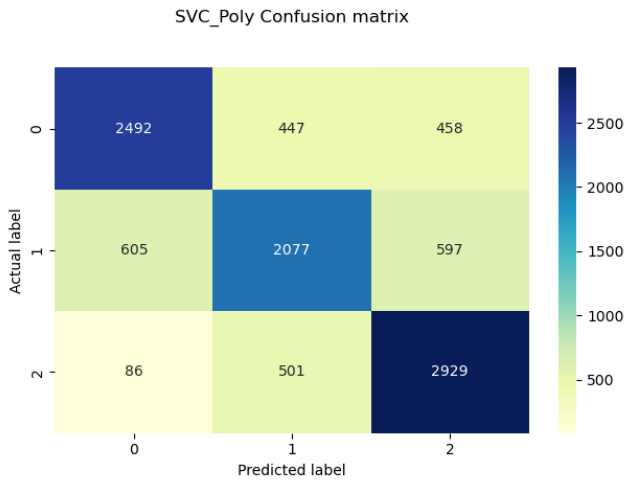
Macro One v/s One: 0.918

SVM (Polynomial Kernel Classifier)

The SVC with polynomial kernel is able to better divide the data into the three categories in comparison with SVC with linear kernel.

```
------------- Evaluating for model SVC_Poly -------------
Confusion Matrix for SVC_Poly is:
[[2492  447  458]
 [ 605 2077  597]
 [  86  501 2929]]
Macro Precision for SVC(C=0.1, kernel='poly', probability=True, random_state=5805) is: 0.735
Micro Precision for SVC(C=0.1, kernel='poly', probability=True, random_state=5805) is: 0.736

Macro Recall for SVC(C=0.1, kernel='poly', probability=True, random_state=5805) is: 0.733
Micro Recall for SVC(C=0.1, kernel='poly', probability=True, random_state=5805) is: 0.736

Macro F1 for SVC(C=0.1, kernel='poly', probability=True, random_state=5805) is: 0.733
Micro F1 for SVC(C=0.1, kernel='poly', probability=True, random_state=5805) is: 0.736

Macro-specificity for SVC_Poly is: 0.868
Macro-averaged One v/s Rest ROC AUC score: 0.854
Micro-averaged One v/s Rest ROC AUC score: 0.861
Macro AUC (OvO) for model SVC_Poly is: 0.853
```

```
---------- Stratified K-Fold CV for SVC_Poly ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for SVC_Poly = 0.733
Macro averaged Recall Score for SVC_Poly = 0.732
Macro averaged F1 Score for SVC_Poly = 0.731
```



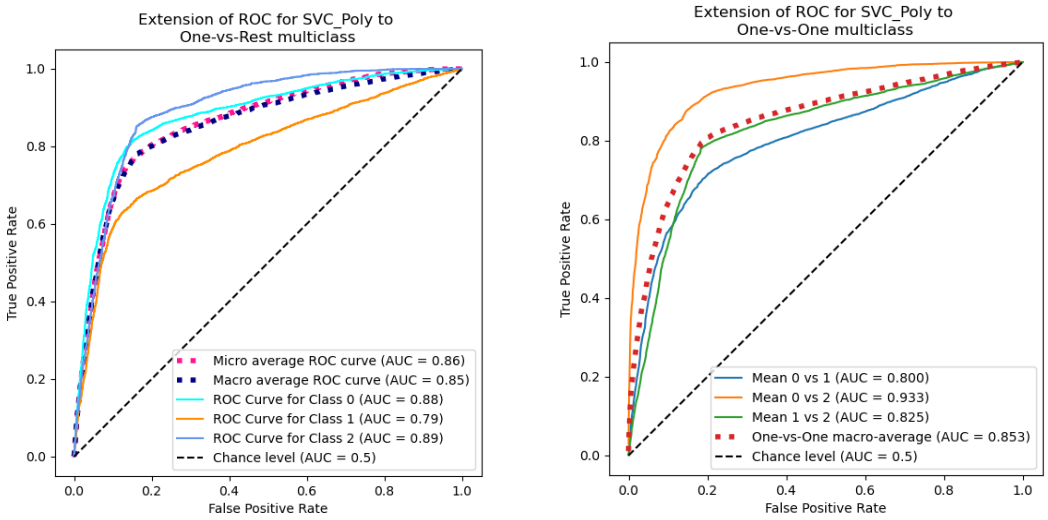


Figure 25: SVM Poly ROC-AUC

ROC-AUC of One v/s Rest approach:

AUC of Class 0: 0.88

AUC of Class 1: 0.79

AUC of Class 2: 0.89

Micro average AUC: 0.86

Macro average AUC: 0.85

ROC-AUC of One v/s One approach:

Mean 0 v/s 1: 0.800

Mean 0 v/s 2: 0.933

Mean 1 v/s 2: 0.835

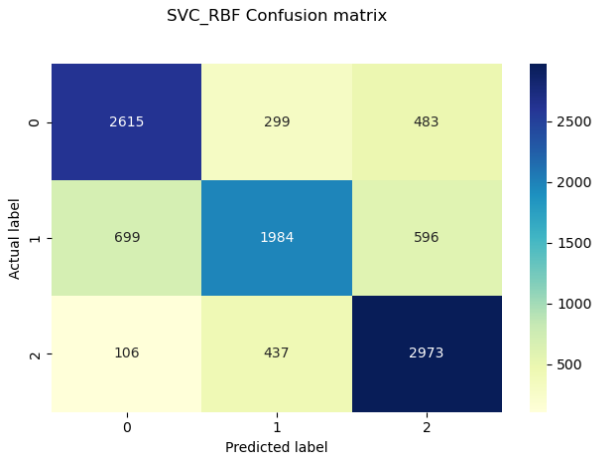
Macro One v/s One: 0.853

SVM (RBF Kernel Classifier)

The SVC with RBF kernel, performs better than both linear and polynomial kernel classifiers.

```
-------------- Evaluating for model SVC_RBF --------------
Confusion Matrix for SVC_RBF is:
[[2615  299  483]
 [ 699 1984  596]
 [ 106  437 2973]]
Macro Precision for SVC(C=0.1, probability=True, random_state=5805) is: 0.743
Micro Precision for SVC(C=0.1, probability=True, random_state=5805) is: 0.743

Macro Recall for SVC(C=0.1, probability=True, random_state=5805) is: 0.74
Micro Recall for SVC(C=0.1, probability=True, random_state=5805) is: 0.743

Macro F1 for SVC(C=0.1, probability=True, random_state=5805) is: 0.74
Micro F1 for SVC(C=0.1, probability=True, random_state=5805) is: 0.743

Macro-specificity for SVC_RBF is: 0.871
Macro-averaged One v/s Rest ROC AUC score: 0.862
Micro-averaged One v/s Rest ROC AUC score: 0.868
Macro AUC (OvO) for model SVC_RBF is: 0.861
```

```
---------- Stratified K-Fold CV for SVC_RBF ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for SVC_RBF = 0.739
Macro averaged Recall Score for SVC_RBF = 0.737
Macro averaged F1 Score for SVC_RBF = 0.735
```



The classification of the data points into their correct labels has also increased with RBF kernel of SVM.

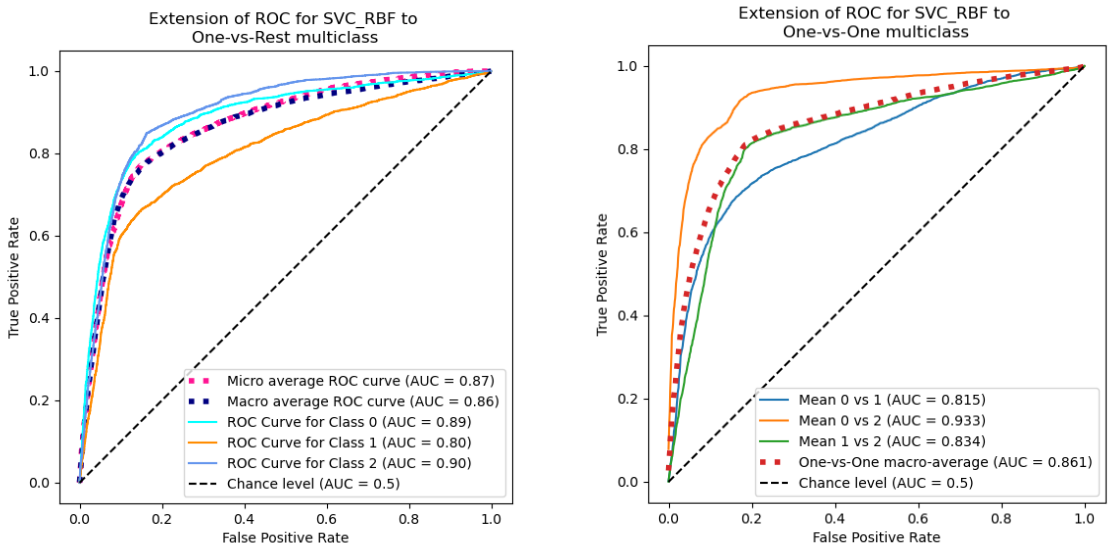


Figure 26: SVM RBF ROC-AUC

ROC-AUC of One v/s Rest approach:

AUC of Class 0: 0.89

AUC of Class 1: 0.80

AUC of Class 2: 0.90

Micro average AUC: 0.87

Macro average AUC: 0.87

ROC-AUC of One v/s One approach:

Mean 0 v/s 1: 0.815

Mean 0 v/s 2: 0.933

Mean 1 v/s 2: 0.834

Macro One v/s One: 0.861

Grid Search SVM

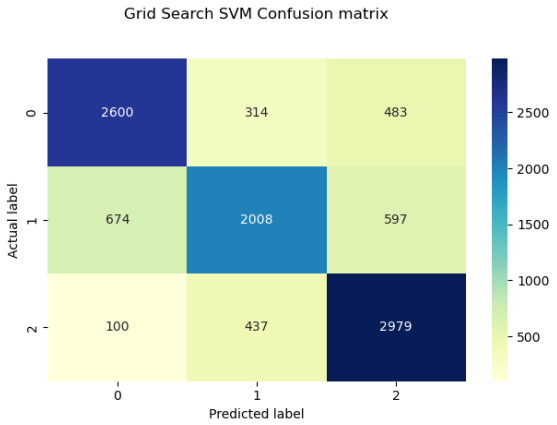
Best Parameter of Grid Search for SVM: (C=0.2, kernel=rbf)

The code will contain limited/fixed search criteria for grid search for execution of code in reasonable time frame.

The SVM Classifier is able to better classify the data into the classes, with increase in performance for predicting class 1 and class 0.

```
------------- Evaluating for model Grid Search SVM -------------
Confusion Matrix for Grid Search SVM is:
[[2600  314  483]
 [ 674 2008  597]
 [ 100  437 2979]]
Macro Precision for SVC(C=0.2, probability=True, random_state=5805) is: 0.744
Micro Precision for SVC(C=0.2, probability=True, random_state=5805) is: 0.744

Macro Recall for SVC(C=0.2, probability=True, random_state=5805) is: 0.742
Micro Recall for SVC(C=0.2, probability=True, random_state=5805) is: 0.744

Macro F1 for SVC(C=0.2, probability=True, random_state=5805) is: 0.742
Micro F1 for SVC(C=0.2, probability=True, random_state=5805) is: 0.744

Macro-specificity for Grid Search SVM is: 0.872
Macro-averaged One v/s Rest ROC AUC score: 0.866
Micro-averaged One v/s Rest ROC AUC score: 0.872
Macro AUC (OvO) for model Grid Search SVM is: 0.866
```

```
---------- Stratified K-Fold CV for Grid Search SVM  ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for Grid Search SVM  = 0.741
Macro averaged Recall Score for Grid Search SVM  = 0.739
Macro averaged F1 Score for Grid Search SVM  = 0.737
```



We can see that the performance has improved for class 1 and 2 for SVM with best parameters.

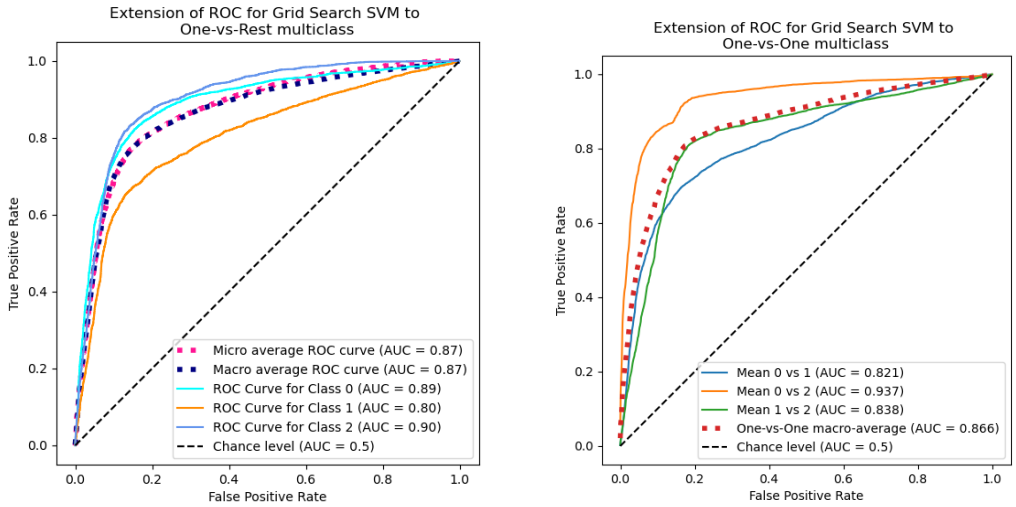


Figure 27: Grid SVM ROC-AUC

ROC-AUC of One v/s Rest approach:

AUC of Class 0: 0.89

AUC of Class 1: 0.80

AUC of Class 2: 0.90

Micro average AUC: 0.87

Macro average AUC: 0.87

ROC-AUC of One v/s One approach:

Mean 0 v/s 1: 0.821

Mean 0 v/s 2: 0.937

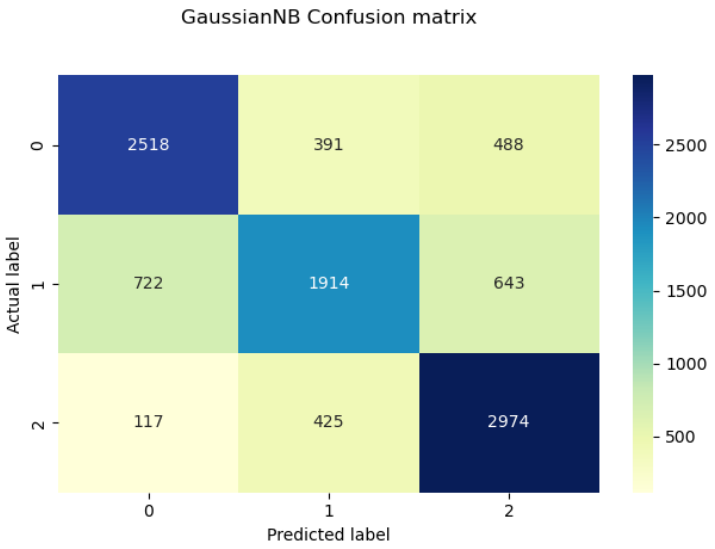
Mean 1 v/s 2: 0.838

Macro One v/s One: 0.866

Naïve Bayes Classifier (Gaussian NB)

```
-------------- Evaluating for model GaussianNB --------------
Confusion Matrix for GaussianNB is:
[[2518  391  488]
 [ 722 1914  643]
 [ 117  425 2974]]
Macro Precision for GaussianNB() is: 0.725
Micro Precision for GaussianNB() is: 0.727

Macro Recall for GaussianNB() is: 0.724
Micro Recall for GaussianNB() is: 0.727

Macro F1 for GaussianNB() is: 0.724
Micro F1 for GaussianNB() is: 0.727

Macro-specificity for GaussianNB is: 0.863
Macro-averaged One v/s Rest ROC AUC score: 0.817
Micro-averaged One v/s Rest ROC AUC score: 0.827
Macro AUC (OvO) for model GaussianNB is: 0.816
```

```
---------- Stratified K-Fold CV for GaussianNB ----------
-----------------Stratified K-Fold CV Results-----------------
Macro averaged Precision Score for GaussianNB = 0.722
Macro averaged Recall Score for GaussianNB = 0.721
Macro averaged F1 Score for GaussianNB = 0.718
```



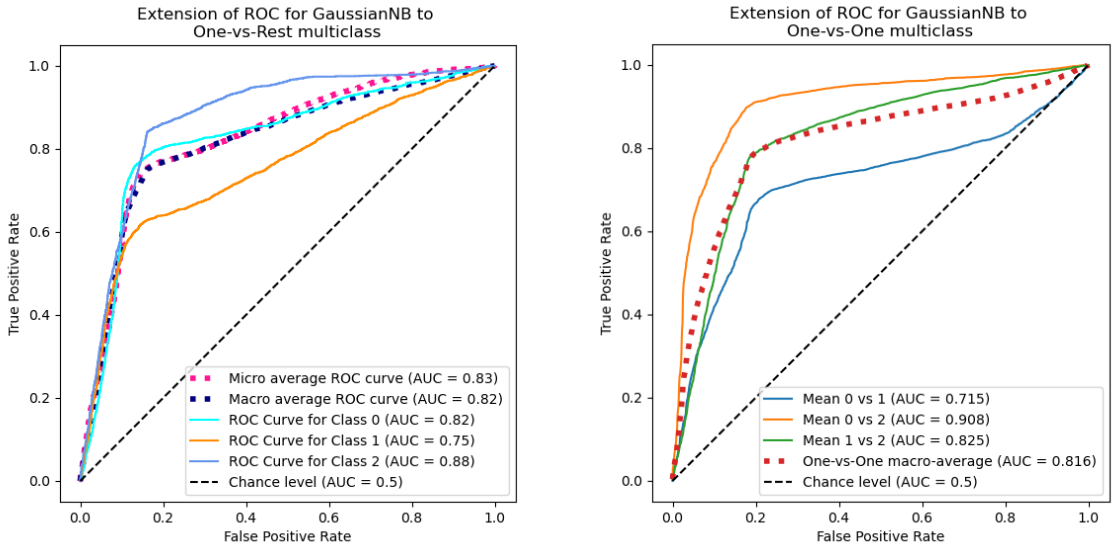


Figure 28: Naive Bayes ROC-AUC

ROC-AUC of One v/s Rest approach:

AUC of Class 0: 0.82

AUC of Class 1: 0.75

AUC of Class 2: 0.88

Micro average AUC: 0.83

Macro average AUC: 0.82

ROC-AUC of One v/s One approach:

Mean 0 v/s 1: 0.715

Mean 0 v/s 2: 0.908

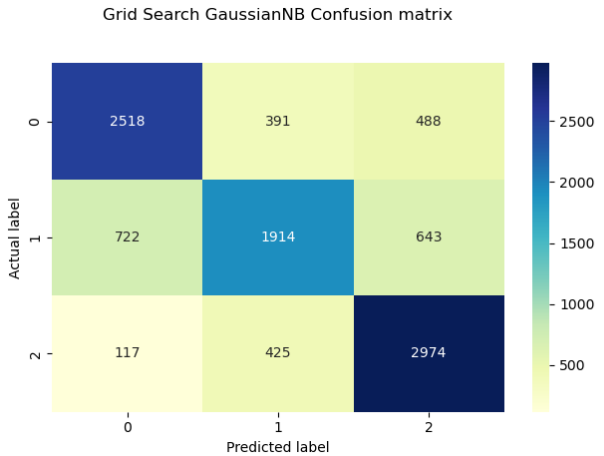
Mean 1 v/s 2: 0.825

Macro One v/s One: 0.816

Grid Search Naïve Bayes Classifier

```
-------------- Evaluating for model Grid Search GaussianNB --------------
Confusion Matrix for Grid Search GaussianNB is:
[[2518  391  488]
 [ 722 1914  643]
 [ 117  425 2974]]
Macro Precision for GaussianNB(var_smoothing=4.328761281083062e-05) is: 0.725
Micro Precision for GaussianNB(var_smoothing=4.328761281083062e-05) is: 0.727

Macro Recall for GaussianNB(var_smoothing=4.328761281083062e-05) is: 0.724
Micro Recall for GaussianNB(var_smoothing=4.328761281083062e-05) is: 0.727

Macro F1 for GaussianNB(var_smoothing=4.328761281083062e-05) is: 0.724
Micro F1 for GaussianNB(var_smoothing=4.328761281083062e-05) is: 0.727

Macro-specificity for Grid Search GaussianNB is: 0.863
Macro-averaged One v/s Rest ROC AUC score: 0.817
Micro-averaged One v/s Rest ROC AUC score: 0.827
Macro AUC (OvO) for model Grid Search GaussianNB is: 0.816
---------- Stratified K-Fold CV for Grid Search GaussianNB ----------
-----------------Stratified K-Fold CV Results-----------------
Macro averaged Precision Score for Grid Search GaussianNB = 0.722
Macro averaged Recall Score for Grid Search GaussianNB = 0.721
Macro averaged F1 Score for Grid Search GaussianNB = 0.718
```

```
---------- Stratified K-Fold CV for Grid Search GaussianNB ----------
-----------------Stratified K-Fold CV Results-----------------
Macro averaged Precision Score for Grid Search GaussianNB = 0.722
Macro averaged Recall Score for Grid Search GaussianNB = 0.721
Macro averaged F1 Score for Grid Search GaussianNB = 0.718
```



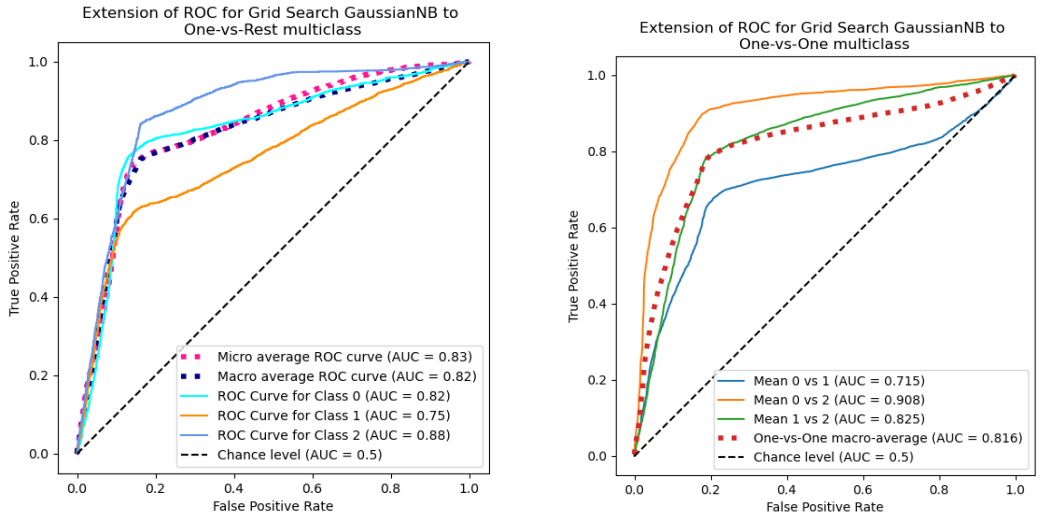


Figure 29: Grid Naive Bayes ROC-AUC

ROC-AUC of One v/s Rest approach:

AUC of Class 0: 0.82

AUC of Class 1: 0.75

AUC of Class 2: 0.90

Micro average AUC: 0.83

Macro average AUC: 0.82

ROC-AUC of One v/s One approach:

Mean 0 v/s 1: 0.715

Mean 0 v/s 2: 0.908

Mean 1 v/s 2: 0.825

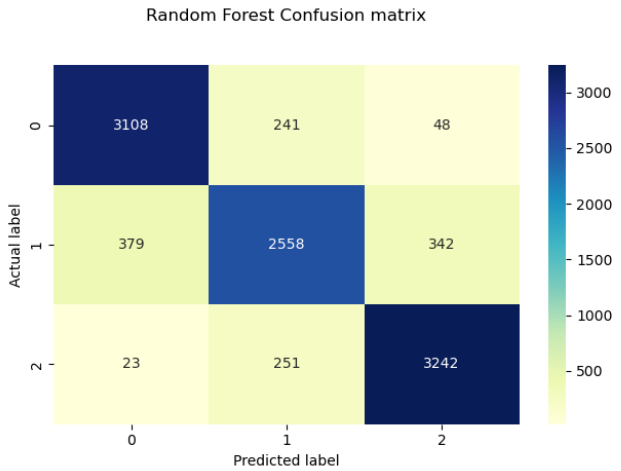
Macro One v/s One: 0.816

Random Forest Classifier

The Random Forest is a combination of multiple weak learners which learn from the data and then classify the points into their classes.

```
-------------- Evaluating for model Random Forest --------------
Confusion Matrix for Random Forest is:
[[3108  241   48]
 [ 379 2558  342]
 [  23  251 3242]]
Macro Precision for RandomForestClassifier(n_estimators=50, random_state=5805) is: 0.872
Micro Precision for RandomForestClassifier(n_estimators=50, random_state=5805) is: 0.874

Macro Recall for RandomForestClassifier(n_estimators=50, random_state=5805) is: 0.872
Micro Recall for RandomForestClassifier(n_estimators=50, random_state=5805) is: 0.874

Macro F1 for RandomForestClassifier(n_estimators=50, random_state=5805) is: 0.872
Micro F1 for RandomForestClassifier(n_estimators=50, random_state=5805) is: 0.874

Macro-specificity for Random Forest is: 0.937
Macro-averaged One v/s Rest ROC AUC score: 0.953
Micro-averaged One v/s Rest ROC AUC score: 0.958
Macro AUC (OvO) for model Random Forest is: 0.952
```

```
---------- Stratified K-Fold CV for Random Forest Classifier ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for Random Forest Classifier = 0.868
Macro averaged Recall Score for Random Forest Classifier = 0.869
Macro averaged F1 Score for Random Forest Classifier = 0.868
```



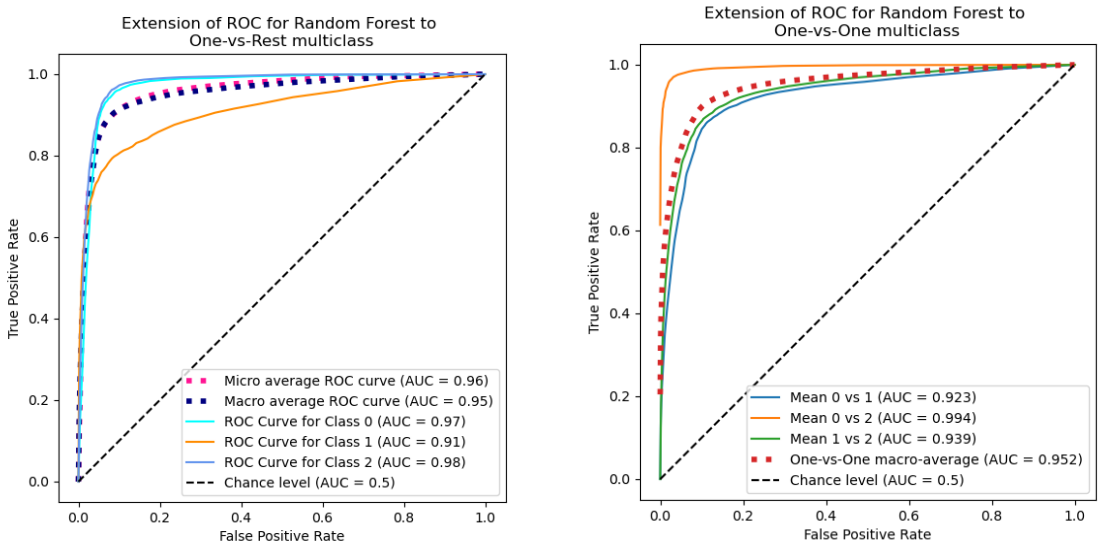


Figure 30: Random Forest ROC-AUC

ROC-AUC of One v/s Rest approach:

AUC of Class 0: 0.97

AUC of Class 1: 0.91

AUC of Class 2: 0.98

Micro average AUC: 0.96

Macro average AUC: 0.95

ROC-AUC of One v/s One approach:

Mean 0 v/s 1: 0.923

Mean 0 v/s 2: 0.994

Mean 1 v/s 2: 0.939

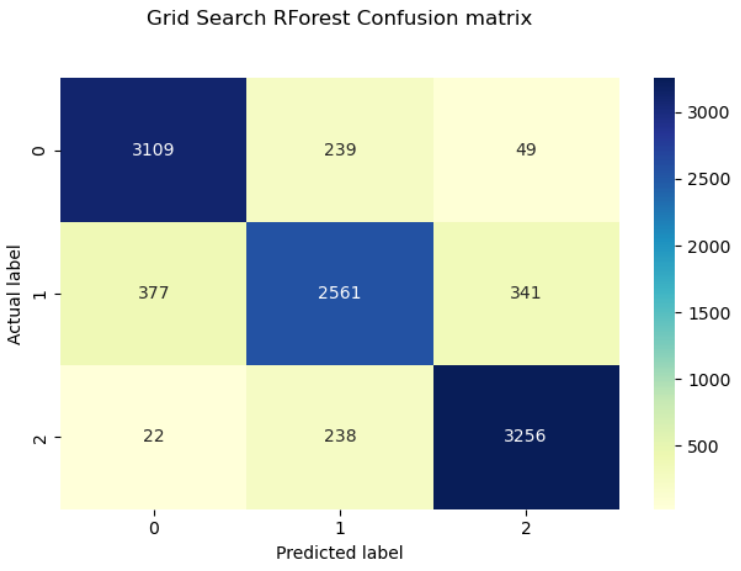
Macro One v/s One: 0.952

Grid Search Random Forest

Best parameters are: {criterion: entropy, n_estimators: 125}

```
-----------------------Grid Search Random Forest Classifier-----------------------
R-Forest Classifier best parameters: {'criterion': 'entropy', 'n_estimators': 125}
------------- Evaluating for model Grid Search RForest -------------
Confusion Matrix for Grid Search RForest is:
[[3109  239   49]
 [ 377 2561  341]
 [  22  238 3256]]
Macro Precision for RandomForestClassifier(criterion='entropy', n_estimators=125, random_state=5805) is: 0.874
Micro Precision for RandomForestClassifier(criterion='entropy', n_estimators=125, random_state=5805) is: 0.876

Macro Recall for RandomForestClassifier(criterion='entropy', n_estimators=125, random_state=5805) is: 0.874
Micro Recall for RandomForestClassifier(criterion='entropy', n_estimators=125, random_state=5805) is: 0.876

Macro F1 for RandomForestClassifier(criterion='entropy', n_estimators=125, random_state=5805) is: 0.874
Micro F1 for RandomForestClassifier(criterion='entropy', n_estimators=125, random_state=5805) is: 0.876

Macro-specificity for Grid Search RForest is: 0.938
Macro-averaged One v/s Rest ROC AUC score: 0.955
Micro-averaged One v/s Rest ROC AUC score: 0.961
Macro AUC (OvO) for model Grid Search RForest is: 0.954
```

```
---------- Stratified K-Fold CV for Grid Search RForest ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for Grid Search RForest = 0.87
Macro averaged Recall Score for Grid Search RForest = 0.87
Macro averaged F1 Score for Grid Search RForest = 0.87
```



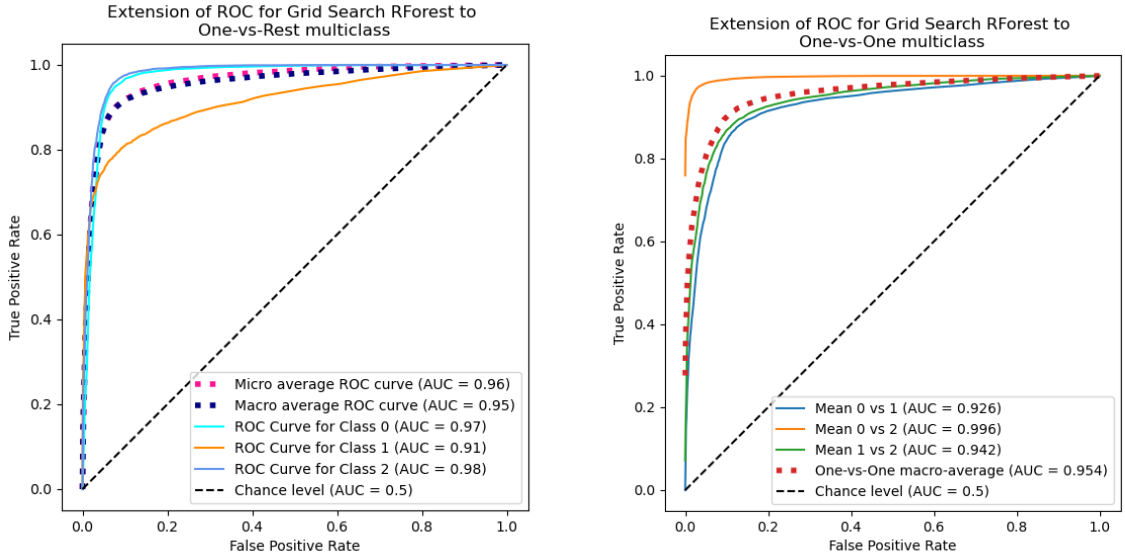


Figure 31: Grid Search Random Forest ROC-AUC

The Grid Search Random Forest Classifier is able to show better performance than the baseline Random Forest Classifier.

Bagging Random Forest Classifier

```
--------------------Bagging with Random Forest Classifier--------------------
-------------- Evaluating for model Bagging Random Forest --------------
Confusion Matrix for Bagging Random Forest is:
[[3125  206   66]
 [ 403 2513  363]
 [  35  234 3247]]
Macro Precision for BaggingClassifier(estimator=RandomForestClassifier(random_state=5805),
                  n_jobs=-1, random_state=5805) is: 0.87
Micro Precision for BaggingClassifier(estimator=RandomForestClassifier(random_state=5805),
                  n_jobs=-1, random_state=5805) is: 0.872

Macro Recall for BaggingClassifier(estimator=RandomForestClassifier(random_state=5805),
                  n_jobs=-1, random_state=5805) is: 0.87
Micro Recall for BaggingClassifier(estimator=RandomForestClassifier(random_state=5805),
                  n_jobs=-1, random_state=5805) is: 0.872

Macro F1 for BaggingClassifier(estimator=RandomForestClassifier(random_state=5805),
                  n_jobs=-1, random_state=5805) is: 0.87
Micro F1 for BaggingClassifier(estimator=RandomForestClassifier(random_state=5805),
                  n_jobs=-1, random_state=5805) is: 0.872

Macro-specificity for Bagging Random Forest is: 0.936
Macro-averaged One v/s Rest ROC AUC score: 0.953
Micro-averaged One v/s Rest ROC AUC score: 0.959
Macro AUC (OvO) for model Bagging Random Forest is: 0.953
```

```
---------- Stratified K-Fold CV for Bagging Random Forest ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for Bagging Random Forest = 0.864
Macro averaged Recall Score for Bagging Random Forest = 0.863
Macro averaged F1 Score for Bagging Random Forest = 0.863
```

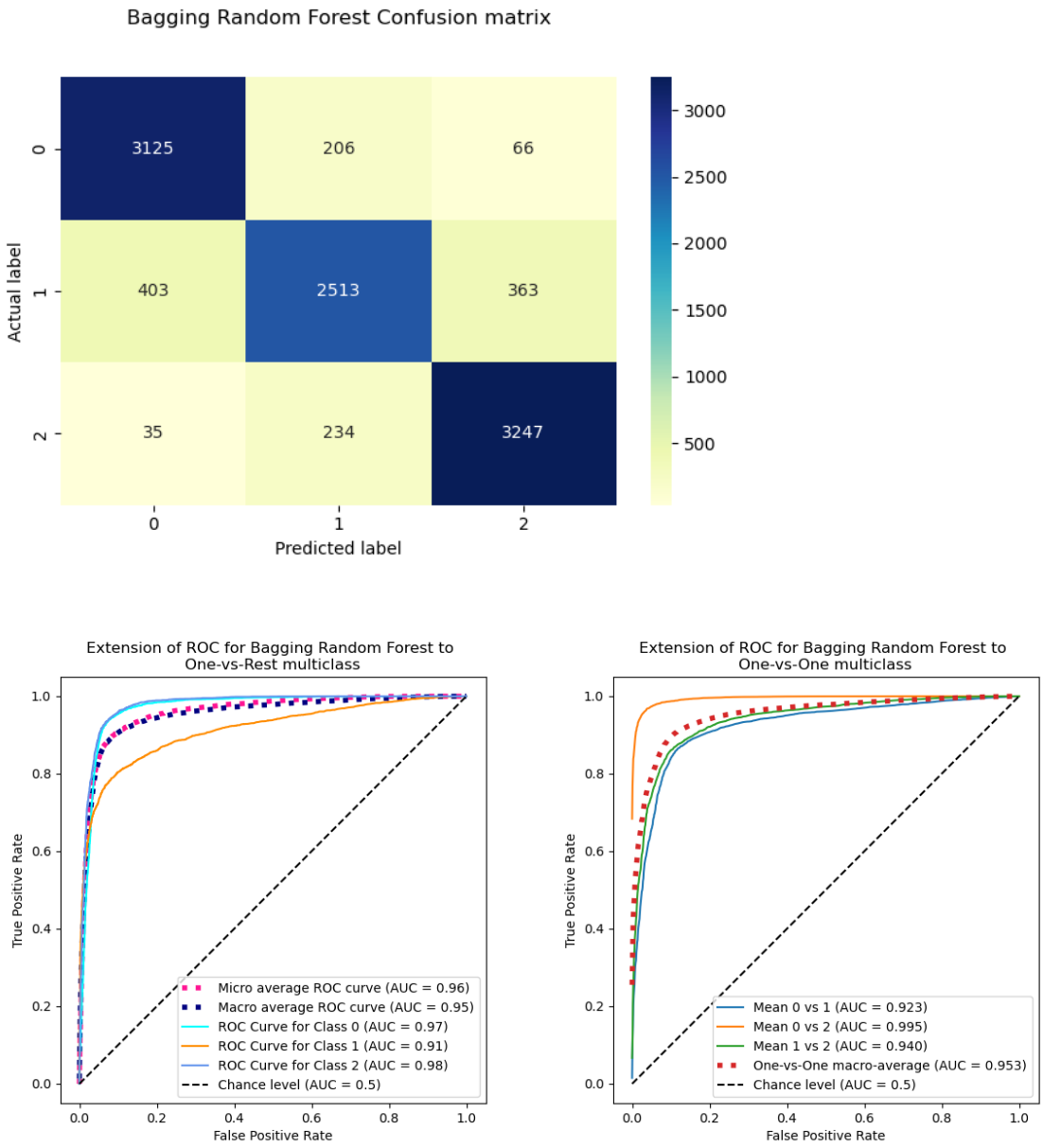


Figure 32: Bagging Random Forest ROC-AUC

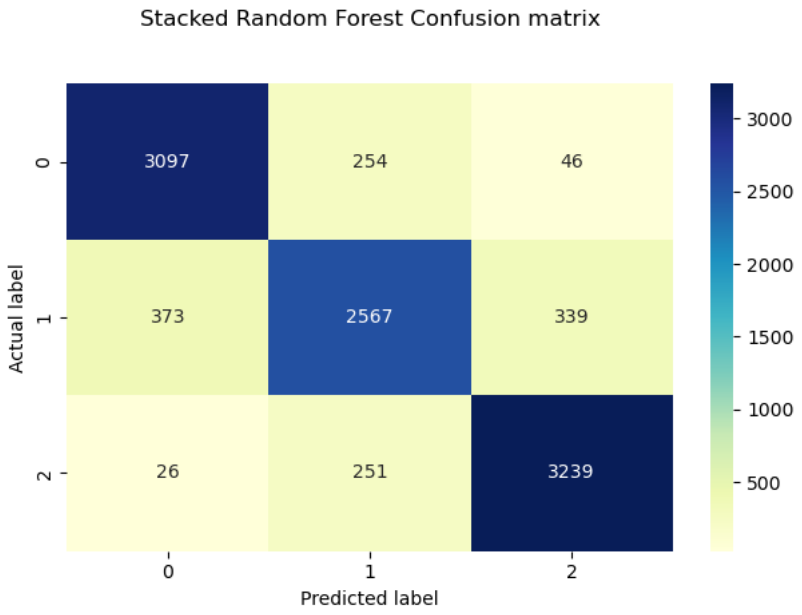
Stacked Random Forest Classifier

```
--------------------Stacking with Random Forest Classifier--------------------
------------- Evaluating for model Stacked Random Forest -------------
Confusion Matrix for Stacked Random Forest is:
[[3097  254   46]
 [ 373 2567  339]
 [  26  251 3239]]
Macro Precision for StackingClassifier(estimators=[('rf',
                                RandomForestClassifier(random_state=5805)),
                               ('rf2',
                                RandomForestClassifier(random_state=5805))],
                   final_estimator=LogisticRegression(multi_class='ovr',
                                                      random_state=5805),
                   n_jobs=-1) is: 0.872
Micro Precision for StackingClassifier(estimators=[('rf',
                                RandomForestClassifier(random_state=5805)),
                               ('rf2',
                                RandomForestClassifier(random_state=5805))],
                   final_estimator=LogisticRegression(multi_class='ovr',
                                                      random_state=5805),
                   n_jobs=-1) is: 0.874
```

```
Macro Recall for StackingClassifier(estimators=[('rf',
                                RandomForestClassifier(random_state=5805)),
                               ('rf2',
                                RandomForestClassifier(random_state=5805))],
                   final_estimator=LogisticRegression(multi_class='ovr',
                                                      random_state=5805),
                   n_jobs=-1) is: 0.872
Micro Recall for StackingClassifier(estimators=[('rf',
                                RandomForestClassifier(random_state=5805)),
                               ('rf2',
                                RandomForestClassifier(random_state=5805))],
                   final_estimator=LogisticRegression(multi_class='ovr',
                                                      random_state=5805),
                   n_jobs=-1) is: 0.874

Macro F1 for StackingClassifier(estimators=[('rf',
                                RandomForestClassifier(random_state=5805)),
                               ('rf2',
                                RandomForestClassifier(random_state=5805))],
                   final_estimator=LogisticRegression(multi_class='ovr',
                                                      random_state=5805),
                   n_jobs=-1) is: 0.872
```

```
Micro F1 for StackingClassifier(estimators=[('rf',
                                RandomForestClassifier(random_state=5805)),
                               ('rf2',
                                RandomForestClassifier(random_state=5805))],
                   final_estimator=LogisticRegression(multi_class='ovr',
                                                      random_state=5805),
                   n_jobs=-1) is: 0.874

Macro-specificity for Stacked Random Forest is: 0.937
Macro-averaged One v/s Rest ROC AUC score: 0.954
Micro-averaged One v/s Rest ROC AUC score: 0.962
Macro AUC (OvO) for model Stacked Random Forest is: 0.954
```

```
---------- Stratified K-Fold CV for Stacked Random Forest ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for Stacked Random Forest = 0.869
Macro averaged Recall Score for Stacked Random Forest = 0.87
Macro averaged F1 Score for Stacked Random Forest = 0.869
```



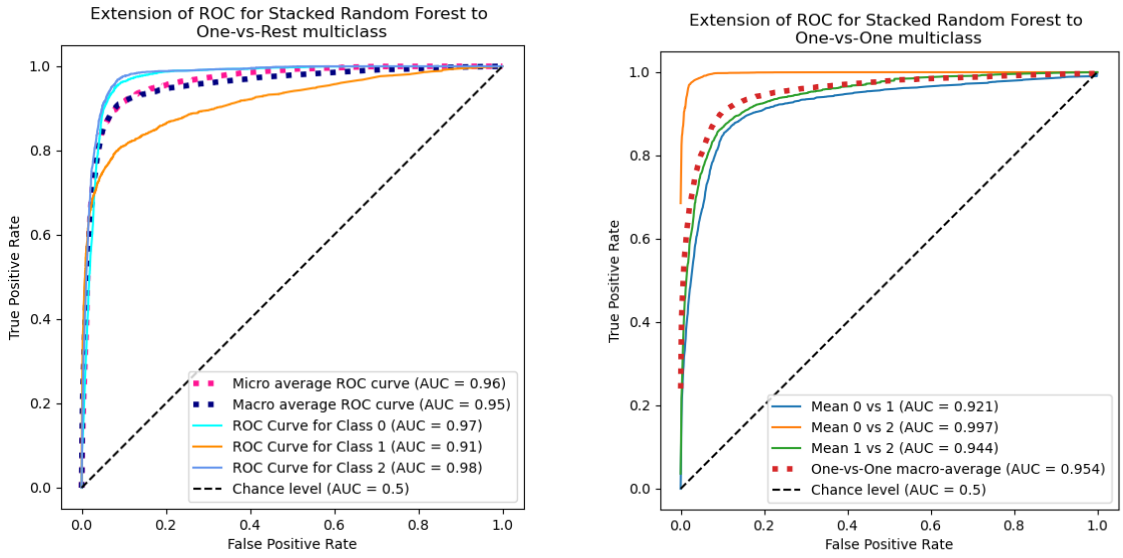


Figure 33: Stacked Random Forest ROC-AUC

Boosting Random Forest Classifier

```
--------------------Boosting with Random Forest Classifier--------------------
------------- Evaluating for model Boosting Random Forest -------------
Confusion Matrix for Boosting Random Forest is:
[[3099  246   52]
 [ 375 2563  341]
 [  24  241 3251]]
Macro Precision for AdaBoostClassifier(estimator=RandomForestClassifier(n_estimators=125,
                                                random_state=5805),
                   random_state=5805) is: 0.873
Micro Precision for AdaBoostClassifier(estimator=RandomForestClassifier(n_estimators=125,
                                                random_state=5805),
                   random_state=5805) is: 0.875

Macro Recall for AdaBoostClassifier(estimator=RandomForestClassifier(n_estimators=125,
                                                random_state=5805),
                   random_state=5805) is: 0.873
Micro Recall for AdaBoostClassifier(estimator=RandomForestClassifier(n_estimators=125,
                                                random_state=5805),
                   random_state=5805) is: 0.875

Macro F1 for AdaBoostClassifier(estimator=RandomForestClassifier(n_estimators=125,
                                                random_state=5805),
                   random_state=5805) is: 0.873
Micro F1 for AdaBoostClassifier(estimator=RandomForestClassifier(n_estimators=125,
                                                random_state=5805),
                   random_state=5805) is: 0.875
```

```
Macro-specificity for Boosting Random Forest is: 0.937
Macro-averaged One v/s Rest ROC AUC score: 0.954
Micro-averaged One v/s Rest ROC AUC score: 0.959
Macro AUC (OvO) for model Boosting Random Forest is: 0.954
```

```
---------- Stratified K-Fold CV for Boosting Random Forest ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for Boosting Random Forest = 0.87
Macro averaged Recall Score for Boosting Random Forest = 0.871
Macro averaged F1 Score for Boosting Random Forest = 0.87
```

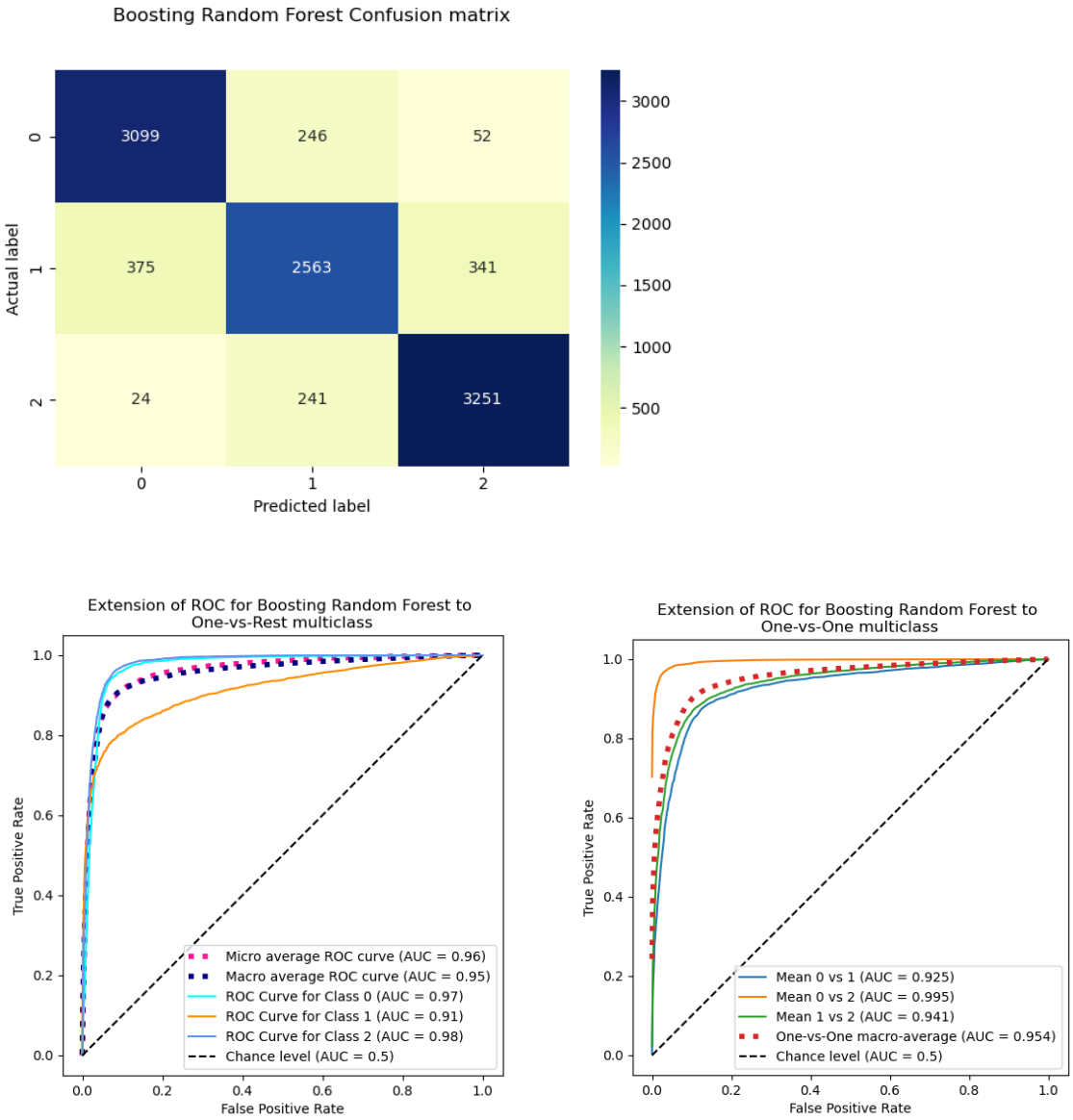


Figure 34: Boosting Random Forest ROC-AUC

The boosting random forest classifier performs the best among the three variants (Stacking, Bagging and Boosting).

Neural Network (MLP Classifier)

```
-------------- Evaluating for model MLP Classifier --------------
Confusion Matrix for MLP Classifier is:
[[2669  413  315]
 [ 622 2072  585]
 [ 114  423 2979]]
Macro Precision for MLPClassifier(hidden_layer_sizes=(20, 20), max_iter=1000, random_state=5805) is: 0.755
Micro Precision for MLPClassifier(hidden_layer_sizes=(20, 20), max_iter=1000, random_state=5805) is: 0.757

Macro Recall for MLPClassifier(hidden_layer_sizes=(20, 20), max_iter=1000, random_state=5805) is: 0.755
Micro Recall for MLPClassifier(hidden_layer_sizes=(20, 20), max_iter=1000, random_state=5805) is: 0.757

Macro F1 for MLPClassifier(hidden_layer_sizes=(20, 20), max_iter=1000, random_state=5805) is: 0.755
Micro F1 for MLPClassifier(hidden_layer_sizes=(20, 20), max_iter=1000, random_state=5805) is: 0.757

Macro-specificity for MLP Classifier is: 0.879
Macro-averaged One v/s Rest ROC AUC score: 0.883
Micro-averaged One v/s Rest ROC AUC score: 0.89
Macro AUC (OvO) for model MLP Classifier is: 0.882
```

```
---------- Stratified K-Fold CV for MLP Classifier ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for MLP Classifier = 0.754
Macro averaged Recall Score for MLP Classifier = 0.754
Macro averaged F1 Score for MLP Classifier = 0.753
```

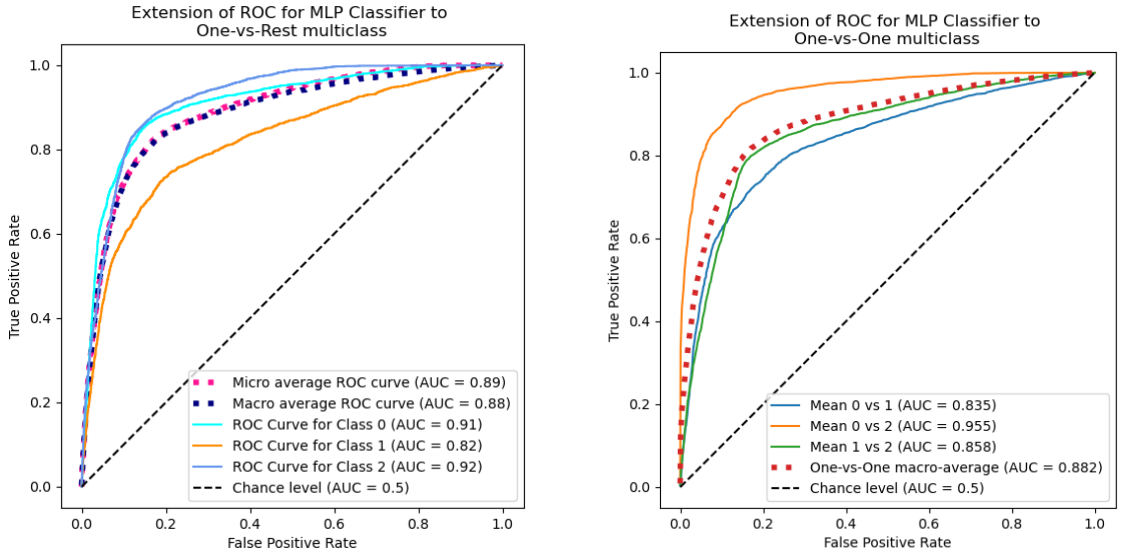


Figure 35: MLP Classifier ROC-AUC

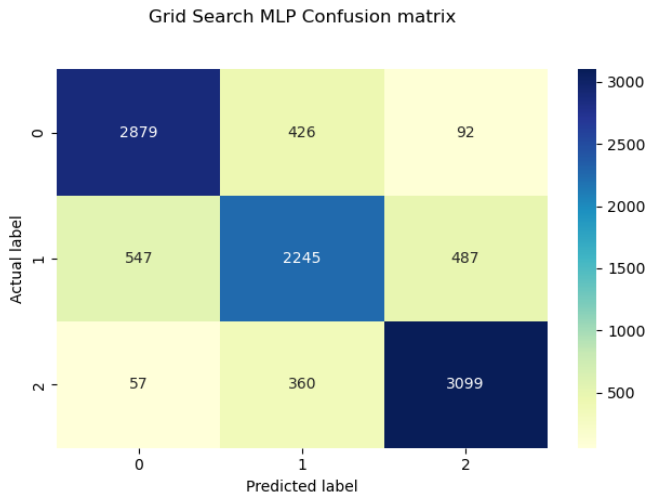
Grid Search Neural Network MLP Classifier

Best parameters = {‘activation’: ‘tanh’, hidden_layer_sizes: (50, 50), max_iter: 1000}

```
-----------------------Grid Search MLP Classifier-----------------------
Best Parameters for MLP Classifier: {'activation': 'tanh', 'hidden_layer_sizes': (50, 50), 'max_iter': 1000}
------------- Evaluating for model Grid Search MLP -------------
Confusion Matrix for Grid Search MLP is:
[[2879  426   92]
 [ 547 2245  487]
 [  57  360 3099]]
Macro Precision for MLPClassifier(activation='tanh', hidden_layer_sizes=(50, 50), max_iter=1000,
              random_state=5805) is: 0.803
Micro Precision for MLPClassifier(activation='tanh', hidden_layer_sizes=(50, 50), max_iter=1000,
              random_state=5805) is: 0.807

Macro Recall for MLPClassifier(activation='tanh', hidden_layer_sizes=(50, 50), max_iter=1000,
              random_state=5805) is: 0.805
Micro Recall for MLPClassifier(activation='tanh', hidden_layer_sizes=(50, 50), max_iter=1000,
              random_state=5805) is: 0.807

Macro F1 for MLPClassifier(activation='tanh', hidden_layer_sizes=(50, 50), max_iter=1000,
              random_state=5805) is: 0.805
Micro F1 for MLPClassifier(activation='tanh', hidden_layer_sizes=(50, 50), max_iter=1000,
              random_state=5805) is: 0.807

Macro-specificity for Grid Search MLP is: 0.904
Macro-averaged One v/s Rest ROC AUC score: 0.912
Micro-averaged One v/s Rest ROC AUC score: 0.921
Macro AUC (OvO) for model Grid Search MLP is: 0.911
```

Stratified K-Fold Cross Validation (k=5)

```
---------- Stratified K-Fold CV for Grid Search MLP ----------
------------------Stratified K-Fold CV Results------------------
Macro averaged Precision Score for Grid Search MLP = 0.792
Macro averaged Recall Score for Grid Search MLP = 0.794
Macro averaged F1 Score for Grid Search MLP = 0.792
```



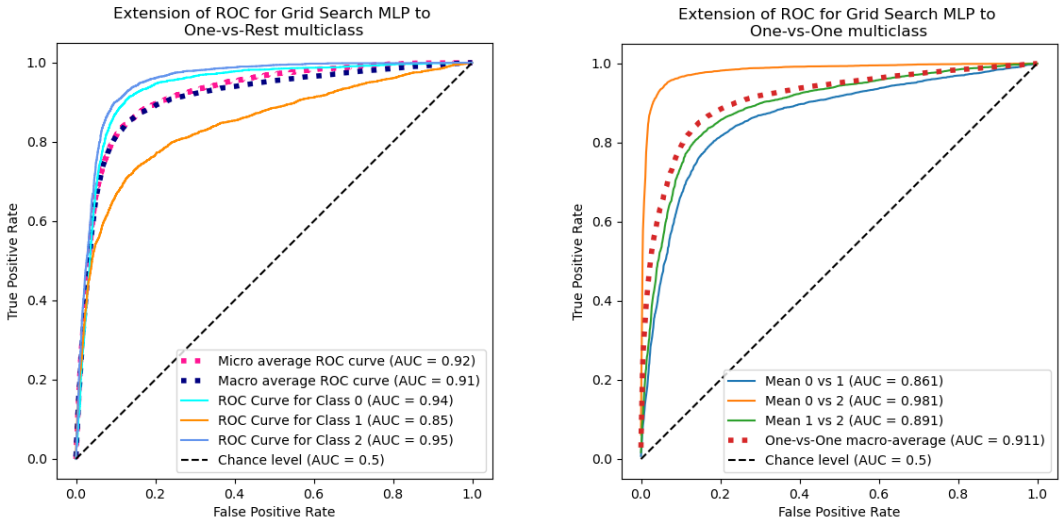


Figure 36: Grid Search MLP ROC-AUC

NOTE: For the downsampled code the code returns convergence warnings as neural network needs more data to converge.

Result Table (Evaluation)

The following results are obtained with 50806 instances or records in the dataset. The executable python file will contain down-sampled data (approx. 12000 records) for faster execution time. (As suggested by professor). Results may differ due to down-sampling of the data. Overall Random Forest Classifier shows the best performance for original and down-sampled data.

Evaluation of various classifiers

Classifier	Confusion_Matrix	Macro-Precision	Macro-Recall	Macro-F1	Macro-Specificity	Macro ROC-AUC(OvR)	Macro ROC-AUC(OvO)
Decision Tree	[[2900 430 67] [409 2493 377] [81 349 3086]]	0.831	0.831	0.831	0.916	0.873	0.873
Pre-pruned Decision Tree	[[2839 368 190] [548 2236 495] [144 437 2935]]	0.783	0.784	0.784	0.893	0.897	0.897
Post-Pruned Decision Tree	[[2960 362 75] [426 2477 376] [80 294 3142]]	0.84	0.84	0.84	0.921	0.882	0.881
Logistic Regression	[[2408 521 468] [657 2026 596] [89 511 2916]]	0.72	0.719	0.719	0.861	0.839	0.838
Grid Logistic Regression	[[2376 556 465] [638 2050 591] [97 482 2937]]	0.721	0.72	0.72	0.861	0.844	0.843
KNN	[[2943 312 142] [605 2078 596] [86 230 3200]]	0.805	0.803	0.803	0.903	0.918	0.917
KNN(Optimal K)	[[3048 246 103] [570 2161 548] [70 176 3270]]	0.832	0.829	0.829	0.916	0.918	0.918
SVC_Linear	[[2463 463 471] [654 2027 598] [90 510 2916]]	0.725	0.724	0.724	0.863	0.826	0.825
SVC_Poly	[[2492 447 458] [605 2077 597] [86 501 2929]]	0.735	0.733	0.733	0.868	0.854	0.853
SVC_RBF	[[2615 299 483] [699 1984 596] [106 437 2973]]	0.743	0.74	0.74	0.871	0.862	0.861
Grid Search SVM	[[2600 314 483] [674 2008 597] [100 437 2979]]	0.744	0.742	0.742	0.872	0.866	0.866
GaussianNB	[[2518 391 488] [722 1914 643] [117 425 2974]]	0.725	0.724	0.724	0.863	0.817	0.816
Grid Search GaussianNB	[[2518 391 488] [722 1914 643] [117 425 2974]]	0.725	0.724	0.724	0.863	0.817	0.816
Random Forest	[[3108 241 48] [379 2558 342] [23 251 3242]]	0.872	0.872	0.872	0.937	0.953	0.952
Grid Search RForest	[[3109 239 49] [377 2561 341] [22 238 3256]]	0.874	0.874	0.874	0.938	0.955	0.954

Stacked Random Forest	[[3097 254 46] [373 2567 339] [26 251 3239]]	0.872	0.872	0.872	0.937	0.954	0.954
Bagging Random Forest	[[3125 206 66] [403 2513 363] [35 234 3247]]	0.87	0.87	0.87	0.936	0.953	0.953
Boosting Random Forest	[[3099 246 52] [375 2563 341] [24 241 3251]]	0.873	0.873	0.873	0.937	0.954	0.954
MLP Classifier	[[2669 413 315] [622 2072 585] [114 423 2979]]	0.755	0.755	0.755	0.879	0.883	0.882
Grid Search MLP	[[2879 426 92] [547 2245 487] [57 360 3099]]	0.803	0.805	0.805	0.904	0.912	0.911

NOTE: All the hyper-parameters for decision tree, KNN and other classifiers might change for the down-sampled data.

All the best parameters obtained in the above results are for 51k records.

Phase 4: Clustering and Association Rule Mining

In this phase, we will implement the K-Means clustering algorithm which helps us divide the customers into various clusters. We will determine the clusters with two methods and choose the best k clusters based on these. For the association rule mining we will implement the Apriori algorithm to find frequent item sets and develop association rules to find what sort of customer behavior is related to the type of loan, payment of minimum amount, and credit mix.

The K-Means Clustering will be implemented with the down-sampled data of 12k records taken from the original dataset. The Association rule mining also takes a slice of the dataframe of approximately 10k records for faster execution.

K-Means Clustering

a. Silhouette Analysis

i. We will select the K with the highest silhouette score.

ii. In this case it is k=3 clusters.

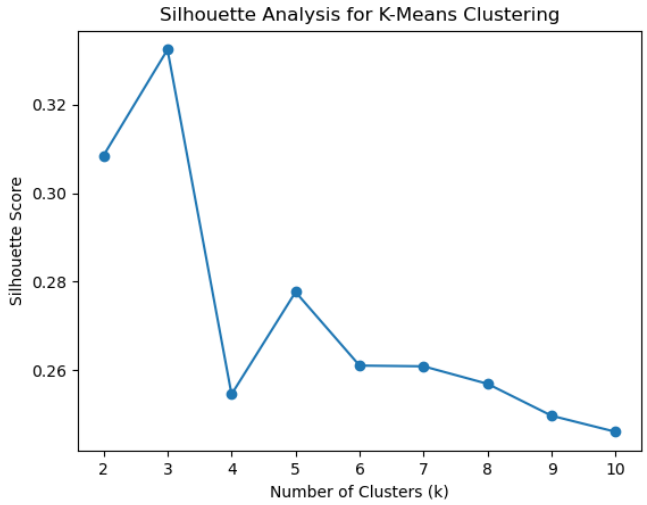


Figure 37: Silhouette Analysis

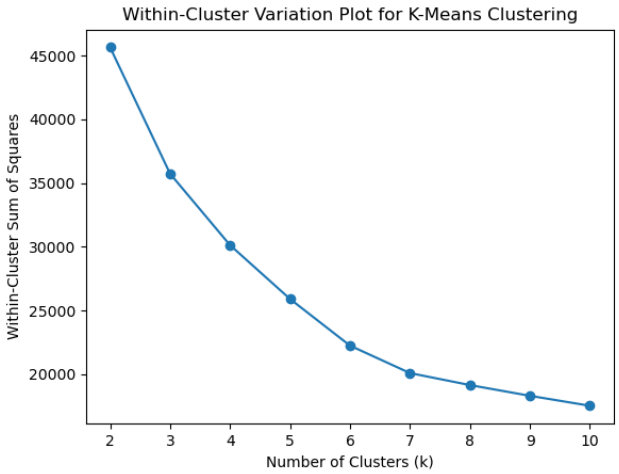


Figure 38: WCSS Analysis

Among the two methods implemented we will choose the silhouette analysis method to choose the optimal k.

Association Rule Mining

We aim to find association between Credit Mix, Payment of Min Amount, Type of Loan and Occupation. We take a slice of 10k records of the original dataset for this phase.

Item sets are formed here with minimum support set to 0.2.

```
---------- Results of Association Rule Mining ----------
Processing 3 combinations | Sampling itemset size 3
    support                                        itemsets
0    0.3469                         (Debt Consolidation Loan)
1    0.3447                                     (Auto Loan)
2    0.3603                                (Credit-Builder Loan)
3    0.3620                                   (Home Equity Loan)
4    0.3513                                      (Personal Loan)
5    0.3465                                       (Student Loan)
6    0.2687                                     (Credit_Mix_Bad)
7    0.2694                                    (Credit_Mix_Good)
8    0.4619                                (Credit_Mix_Standard)
9    0.3578                           (Payment_of_Min_Amount_No)
10   0.6422                          (Payment_of_Min_Amount_Yes)
11   0.2613  (Payment_of_Min_Amount_Yes, Debt Consolidation...
12   0.2602            (Payment_of_Min_Amount_Yes, Auto Loan)
13   0.2711   (Payment_of_Min_Amount_Yes, Credit-Builder Loan)
14   0.2732      (Payment_of_Min_Amount_Yes, Home Equity Loan)
15   0.2625         (Personal Loan, Payment_of_Min_Amount_Yes)
16   0.2567          (Payment_of_Min_Amount_Yes, Student Loan)
17   0.2687        (Credit_Mix_Bad, Payment_of_Min_Amount_Yes)
18   0.2694        (Credit_Mix_Good, Payment_of_Min_Amount_No)
19   0.3735   (Credit_Mix_Standard, Payment_of_Min_Amount_Yes)
```

The association rules are shown below with confidence threshold set to 0.7.

antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
(Credit_Mix_Good)	(Payment_of_Min_Amount_No)	0.2694	0.3578	0.2694	1.000000	2.794857	0.173009	inf	0.879004
(Credit_Mix_Bad)	(Payment_of_Min_Amount_Yes)	0.2687	0.6422	0.2687	1.000000	1.557147	0.096141	inf	0.489266
(Credit_Mix_Standard)	(Payment_of_Min_Amount_Yes)	0.4619	0.6422	0.3735	0.808617	1.259135	0.076868	1.869545	0.382464
(Auto Loan)	(Payment_of_Min_Amount_Yes)	0.3447	0.6422	0.2602	0.754859	1.175427	0.038834	1.459570	0.227751
(Home Equity Loan)	(Payment_of_Min_Amount_Yes)	0.3620	0.6422	0.2732	0.754696	1.175173	0.040724	1.458599	0.233639
(Debt Consolidation Loan)	(Payment_of_Min_Amount_Yes)	0.3469	0.6422	0.2613	0.753243	1.172910	0.038521	1.450010	0.225723
(Payment_of_Min_Amount_No)	(Credit_Mix_Good)	0.3578	0.2694	0.2694	0.752935	2.794857	0.173009	2.957112	1.000000
(Credit-Builder Loan)	(Payment_of_Min_Amount_Yes)	0.3603	0.6422	0.2711	0.752429	1.171642	0.039715	1.445239	0.229009
(Personal Loan)	(Payment_of_Min_Amount_Yes)	0.3513	0.6422	0.2625	0.747225	1.163539	0.036895	1.415486	0.216669
(Student Loan)	(Payment_of_Min_Amount_Yes)	0.3465	0.6422	0.2567	0.740837	1.153592	0.034178	1.380598	0.203738

Rules:

Credit Mix Good → Payment of Min Amount No,

Credit Mix Bad → Payment of Min Amount Yes,

Credit Mix Standard → Payment of Min Amount Yes,

Auto Loan → Payment of Min Amount Yes

Payment of Min Amount No → Credit Mix Good

Home Equity Loan → Payment of Min Amount Yes

Recommendations

A. Key Learnings:

1. Various data cleaning techniques to clean the data and understood the importance and impact of clean data and good quality data by ensuring no duplicates in the data.
2. Filling the missing values in the dataset, by aggregating on specific features.
3. Understanding and implementing various feature selection techniques in depth and evaluating each one of them based on their pros and cons. We proceed with one feature selection technique in the project and build models on the data selected from it.
4. Random Forest based feature selection, enables us to select features based on threshold and gives us a list of features relevant to the problem to be solved.
5. Techniques such as PCA, SVD with the concept of condition number showed the capabilities of these techniques in dimensionality reduction and reducing the collinearity
6. Variance Inflation Factor added a new perspective to feature selection for Regression Analysis by handling the collinearity problem.
7. Correlation Analysis and Covariance Analysis helped in providing better understanding of the dataset and linear relationships amongst the features.
8. The project also involved, balancing the imbalanced data. This enabled me to explore techniques like SMOTE, Tomek-Links for balancing the data.
9. In the regression phase, we implement multiple linear regression by analyzing the f-statistic, p-value for T-test and feature elimination using stepwise regression OLS. This helped to understand the importance of each of these statistical parameters and their impact on the regression analysis.

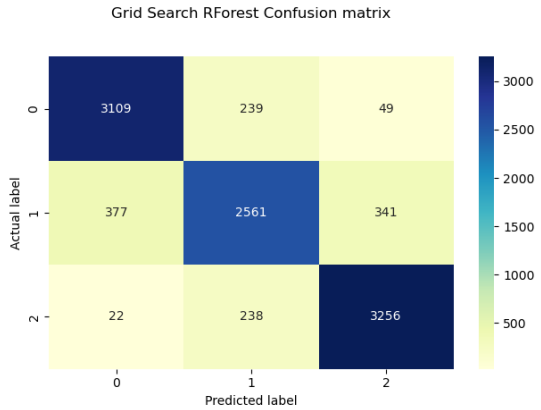
10. In the classification phase, we implement a variety of classifiers to categorize the credit score into three classes. The focus on improving the performance of each classifier helped us explore each classifier in depth via various techniques like Grid Search, Elbow Method (KNN) for improving the performance.
11. The various evaluation metrics calculated for each classifier help in comparing one classifier with the other thus enabling us to choose the best classifier amongst all.
12. The classification problem being solved in this project is a multi-class classification problem. This helped me learn evaluating each classifier in one v/s rest and one v/s one approach. Plotting ROC-AUC curves for both the approaches helps us to evaluate the classifier's performance for various combinations of the target labels.
13. The clustering and association rule mining phase, helps us discover deeper insights or relations between the customers in this credit related dataset. Apriori algorithm helps us to discover association rules in the data. K-Means Clustering helps us divide the customers into clusters.
14. Overall, the project gives in-depth knowledge of various machine learning techniques and the necessary background work to provide quality data to the machine learning models.

B. Selecting the Best Classifier

The table in the results section shows various metrics like precision, recall, f1, specificity, roc-auc (OvR), roc-auc (OvO). The dataset has been balanced hence we can verify that micro and macro averages for precision, recall, f1 are very much similar to each other (check in output section of each classifier).

Based on the results, we can say that broadly, the Random Forest Classifier performs the best. Now, in the implementation of the Random Forest Classifier, we have variants like Stacked, Bagging, Boosting and Grid Search Random Forest Classifier. The Grid Search Random Forest Classifier and the Boosting Random Forest Classifier are the top 2 classifiers with best results.The Grid Search Random Forest Classifier slightly outperforms the Boosting Random Forest Classifier. Grid Search Random Forest Classifier shows the best results:

Model	Macro Precision	Macro Recall	Macro F1	Macro Specificity	Macro ROC-AUC(OvR)	Macro ROC-AUC(OvO)
Grid Search Random Forest	0.874	0.874	0.874	0.938	0.955	0.954



As we can verify with the results table in the classification phase, Grid Search Random Forest Classifier performs the best for all evaluation metrics.

C. Selecting Relevant Features

    a. We have chosen the Random Forest for feature selection.

    b. Regression Analysis:

        i. Features selected: ['Num_of_Loan', 'Credit_Mix_Standard', 'Interest_Rate', 'Changed_Credit_Limit', 'Credit_History_Age', 'Num_of_Delayed_Payment', 'Amount_invested_monthly', 'Monthly_Balance', 'Num_Credit_Inquiries', 'Credit_Score', 'Monthly_Inhand_Salary', 'Annual_Income', 'Age', 'Num_Credit_Card', 'Total_EMI_per_month', 'Delay_from_due_date']

        ii. Feature selected with OLS: ['Age', 'Auto Loan', 'Changed_Credit_Limit', 'Credit_History_Age', 'Debt Consolidation Loan', 'Delay_from_due_date', 'Interest_Rate', 'Monthly_Balance', 'Monthly_Inhand_Salary', 'Mortgage Loan', 'Num_Bank_Accounts', 'Num_Credit_Card', 'Num_of_Loan', 'Payday Loan', 'Personal Loan', 'Student Loan', 'Payment_of_Min_Amount_Yes', 'Credit_Mix_Good', 'Credit_Mix_Standard', 'Occupation_Doctor', 'Occupation_Journalist', 'Occupation_Manager', 'Occupation_Mechanic', 'Occupation_Scientist', 'Occupation_Writer', 'Credit_Score']

    c. Classification Analysis:

        i. Features Selected: ['Outstanding_Debt', 'Interest_Rate', 'Delay_from_due_date', 'Credit_Utilization_Ratio',

'Changed_Credit_Limit', 'Credit_Mix_Standard', 'Credit_History_Age', 'Num_Credit_Inquiries', 'Num_of_Delayed_Payment', 'Credit_Mix_Good', 'Num_Credit_Card', 'Total_EMI_per_month', 'Age', 'Monthly_Balance', 'Monthly_Inhand_Salary', 'Amount_invested_monthly', 'Annual_Income', 'Num_Bank_Accounts', 'Num_of_Loan', 'Payment_of_Min_Amount_Yes']

D. Based on Clustering Analysis: Our data can be clustered into three clusters.

Future Work

The performance of the classifiers seems to be good, but there is always scope for improvement. Due to limitations of computation power, we have limited the grid search parameters for hyper-parameter tuning. We can see that even after using fewer parameters in the grid search we have achieved performance improvement. Thus, by exploring more number of hyper-parameters for each classifier, we can achieve better performance. As we can see that, random forest classifier is the best performing classifier, we can explore stacking, bagging and boosting classifiers with various combinations of models in the stack or bag. In boosting classifiers, we can explore gradient boosting or XGBoost (extreme gradient boosting) which are powerful algorithms being used by many practitioners to improve classification performance. The last classifier we implemented is the MLP Classifier which is a neural network. We can experiment with more combinations of architectures of hidden layers and number of nodes in each layer. The neural network can thus learn better, which can lead to improving the performance of the classifier. The feature selection can be varied by setting various thresholds and based on experimentation we can evaluate the classifiers for best performance. The data cleaning part, especially outlier analysis and removal can be done using other techniques apart from IQR, which might improve model performance.

Appendix

A. main.py: This python file contains code for feature engineering and EDA phase (Phase 1). Three data files are created from the original dataset – one for each phase.

   1) Regression Dataset (Debt_Regression_Cleaned.csv)

   2) Classification Dataset (credit_classification_cleaned.csv)

   3) Association Rule Mining Dataset (Association.csv)

   4) Random Forest selected features data (rf_regression.csv)

B. Regression.py: This python file contains code for the Regression Analysis.

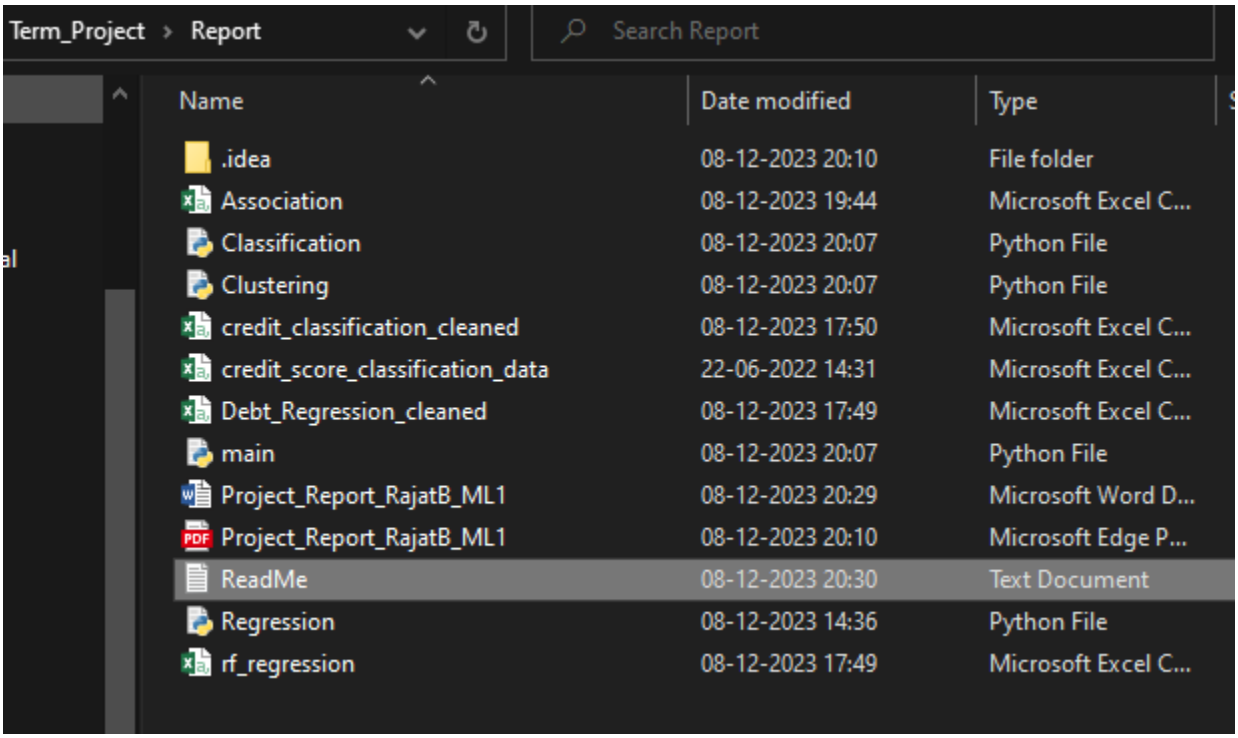
C. Classification.py: This python file contains code for implementing various classifiers listed in the term project requirements.

D. Clustering.py: This python file contains code for Clustering and Association Rule Mining with K-Means and Apriori algorithm.

NOTE: All the datasets will be provided in the submission of the report and python files. The code to create the above 4 CSV files will be commented in the actual python files.

NOTE: Please rename the CSV files to the following to ensure smooth execution of the code.

1. Association.csv
2. credit_classification_cleaned.csv
3. credit_score_classification_data.csv
4. credit_classification_cleaned.csv
5. Debt_Regression_cleaned.csv
6. rf_regression.csv



main.py - Phase 1

```
# ----------------------------------------------------------------
#           Term Project - CS 5805 Machine Learning 1
# ----------------------------------------------------------------
# Author: Rajat Belgundi

# -----------------Project is conducted in four phases----------------------
# Phase 1: Feature Engineering and Exploratory Data Analysis
# Phase 2: Regression Analysis [on selected continuous numerical feature]
# Phase 3: Classification Analysis
# Phase 4: Association Rule Mining and Clustering
# ------------------------------------------------------------------------
-


# Imports here
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier,RandomForestRegressor
import pickle
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import TruncatedSVD
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,accuracy_score, f1_score,
```

```
recall_score, precision_score,classification_report
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler, TomekLinks
from statsmodels.stats.outliers_influence import variance_inflation_factor
import time
pd.set_option('display.max_columns', 100)

# import phase1  # Phase1
# Basic EDA
def basic_eda(data):
    # Display first 5 rows
    print(data.head())
    # Inspect numerical columns
    print(data.describe())
    # Inspect memory usage
    print(data.info(memory_usage='deep'))
    print('Data Description')
    print(data.describe().T)
    print('Shape of Data')
    print('Rows: {}'.format(data.shape[0]))
    print('Columns: {}'.format(data.shape[1]))


def missing_value_check(data):
    # Check missing values
    # print(self.data.isna().sum())
    missing_data = pd.DataFrame(data.isna().sum())
    print(missing_data)


def imbalance_check(data, col):
    print(data[col].value_counts(normalize=True))  # Count in %
    sns.countplot(x=data[col])
    plt.title('Imbalance Check for '+col)
    plt.show()


def clean_object_data(data):
    # Strip the additional underscores
    char_remove = ['!@9#%8', '_']
    for char in char_remove:
        data = data.replace(char, "")
    return data


def clean_numeric_data(data):
    try:
        return float(data.replace("_", ""))
    except:
        return np.nan


def credit_history_age(age):
    # 15 Years and 11 Months is the format
    if pd.notnull(age):
```

```
        age_str = age.split()
        return round(int(age_str[0]) * 12 + int(age_str[3]),2)


def standardize_data(df):
    standardized_df = (df - df.mean(numeric_only=True)) /
(df.std(numeric_only=True))
    return standardized_df


def conduct_rforest_selection(X, y, kind):
    if kind == 'C':
        print('------------------METHOD 1: RANDOM_FOREST CLASSIFICATION------
------------')
        clf_model = RandomForestClassifier(n_estimators=100,
random_state=5805)
        clf_model.fit(X, y)
        # Save model as pickle
        # model_file = 'rf_model.pkl'
        # with open(model_file, 'wb') as file:
        #     pickle.dump(model, file)

        # Load from pickle
        # with open(model_file, 'rb') as file:
        #     model = pickle.load(file)
        importances = clf_model.feature_importances_
        # Check Feature Importance
        feature_importance = pd.Series(clf_model.feature_importances_,
index=X.columns).sort_values(ascending=False)
        plt.figure(figsize=(30, 16))
        sns.barplot(x=feature_importance, y=feature_importance.index,
palette='viridis')
        plt.title('RForest Classifier based Feature Importance')
        plt.show()
        selected_features = []
        eliminated_features = []
        threshold = 0.016
        # feature_importance series has the mapping of features and
importances
        # Iterate through series using items function to get index and value
        for feature, importance in feature_importance.items():
            if importance > threshold:
                selected_features.append(feature)
            else:
                eliminated_features.append(feature)
        print('No. of features selected by RForest Analysis:',
len(selected_features))
        print('(RForest Classifier Analysis) Selected Features with threshold
= {}:'.format(threshold))
        print(selected features)
        print('(RForest Classifier Analysis) Eliminated Features with
threshold = {}:'.format(threshold),
              eliminated_features)
        print('------------------------------------------------------------
-------------------------------')
```

```
        return selected_features

    if kind == 'R':
        print('------------------METHOD 1: RANDOM_FOREST REGRESSION----------
--------')
        reg_model = RandomForestRegressor(n_estimators=100,
random_state=5805)
        reg_model.fit(X_reg, y_reg)
        importances = reg_model.feature_importances_
        # Check Feature Importance
        feature_importance = pd.Series(reg_model.feature_importances_,
index=X_reg.columns).sort_values(ascending=False)
        plt.figure(figsize=(30, 16))
        sns.barplot(x=feature_importance, y=feature_importance.index,
palette='viridis')
        plt.title('RForest Regressor based Feature Importance')
        plt.show()
        selected_features = []
        eliminated_features = []
        threshold = 0.01
        # feature_importance series has the mapping of features and
importances
        # Iterate through series using items function to get index and value
        for feature, importance in feature_importance.items():
            if importance > threshold:
                selected_features.append(feature)
            else:
                eliminated_features.append(feature)
        print('No. of features selected by RForest Regressor Analysis:',
len(selected_features))
        print('(RForest Regressor Analysis) Selected Features with threshold
= {}:'.format(threshold))
        print(selected_features)
        print('(RForest Regressor Analysis) Eliminated Features with
threshold = {}:'.format(threshold),eliminated_features)
        print('----------------------------------------------------------------
--------------------------------')
        return selected_features


def conduct_PCA(X, y, kind):
    if kind == 'C':
        print('----------------------------------------------------------------
---------')
        print('          Principal Component Analysis (PCA) for
Classification        ')
        print('----------------------------------------------------------------
---------')
        pca = PCA(random_state=5805)
        pca.fit(X, y)
        print('Shape of X: ', X.shape)
        print('Columns of X:', X.columns)
        # print('Explained variance:',pca.explained_variance_)
        # print('Cumulative sum of explained variance
ratio',np.cumsum(pca.explained_variance_ratio_))
```

```
        n_comp = [i for i in range(1, X.shape[1] + 1)]
        cum_exp_variance = np.cumsum(pca.explained_variance_ratio_)
        print('Explained Variance:', cum_exp_variance)
        plt.figure(figsize=(18, 10))
        print('N-comp:', len(n_comp))
        print('Exp:', len(cum_exp_variance))
        plt.plot(n_comp, np.cumsum(pca.explained_variance_ratio_),
marker='o', linestyle='-')
        plt.axhline(y=0.9, linestyle='-', color='red')
        # print(x=)
        plt.axvline(x=len([var for var in cum_exp_variance if var < 0.9]) +
1, linestyle='-', color='red')
        # plt.plot(y=0.9)
        plt.xticks(n_comp)  # x-ticks start from 1
        plt.xlabel('Number of components')
        plt.ylabel('Cumulative Sum of Explained Variance Ratio')
        plt.title('Selecting # of components for PCA - Classification
Dataset')
        plt.show()
        return pca

    if kind == 'R':
        print('----------------------------------------------------------------
------')
        print('          Principal Component Analysis (PCA) for Regression
')
        print('----------------------------------------------------------------
------')
        pca_reg = PCA(random_state=5805)
        pca_reg.fit(X,y)
        print('Shape of X: ', X.shape)
        print('Columns of X:', X.columns)
        # print('Explained variance:',pca.explained_variance_)
        # print('Cumulative sum of explained variance
ratio',np.cumsum(pca.explained_variance_ratio_))
        n_comp = [i for i in range(1, X.shape[1] + 1)]
        cum_exp_variance = np.cumsum(pca_reg.explained_variance_ratio_)
        print('Explained Variance:', cum_exp_variance)
        plt.figure(figsize=(18, 10))
        print('N-comp:', len(n_comp))
        print('Exp:', len(cum_exp_variance))
        plt.plot(n_comp, np.cumsum(pca_reg.explained_variance_ratio_),
marker='o', linestyle='-')
        plt.axhline(y=0.9, linestyle='-', color='red')
        # print(x=)
        plt.axvline(x=len([var for var in cum_exp_variance if var < 0.9]) +
1, linestyle='-', color='red')
        # plt.plot(y=0.9)
        plt.xticks(n_comp)  # x-ticks start from 1
        plt.xlabel('Number of components')
        plt.ylabel('Cumulative Sum of Explained Variance Ratio')
        plt.title('Selecting # of components for PCA - Regression Dataset')
        plt.show()
        return pca_reg
```

```
def get_condition_num(X, y, pca, n):
    print(' ------------------------------ ')
    print('        Condition Number        ')
    print(' ------------------------------ ')
    pca_selected = PCA(n_components=n, random_state=5805)
    pca_selected.fit(X,y)
    cov_matrix = pca.get_covariance()
    cov_selected = pca_selected.get_covariance()
    # Condition number
    condition_num = np.linalg.cond(cov_matrix)
    c_num_selected = np.linalg.cond(cov_selected)
    print('Condition Number (pre PCA):', condition_num)
    print('Condition Number (post PCA) and {} components:'.format(n),
c_num_selected)


def truncated_svd(X, kind):
    if kind == 'C':
        print('-----------------------------------------------------------------
------')
        print('          Singular Value Decomposition (SVD) for
Classification     ')
        print('-----------------------------------------------------------------
------')
        svd = TruncatedSVD(n_components=(X.shape[1]-1))
        svd.fit(X)

        # Original data shape
        print("Original Data Shape:", X.shape)

        # Reduced data shape
        # print("Reduced Data Shape:", data_reduced.shape)
        n_comp = [i for i in range(1, X.shape[1])]
        # Explained variance ratio
        print("Explained Variance Ratio:",
svd.explained_variance_ratio_.sum())

        # Visualize explained variance
        cum_exp_variance = np.cumsum(svd.explained_variance_ratio_)
        plt.figure(figsize=(14, 6))
        plt.plot(n_comp, np.cumsum(svd.explained_variance_ratio_))
        plt.axhline(y=0.9, linestyle='-', color='red')
        # print(x=)
        plt.axvline(x=len([var for var in cum_exp_variance if var < 0.9])+1,
linestyle='-', color='red')
        plt.xlabel('Number of Components')
        plt.ylabel('Cumulative Explained Variance')
        plt.title('SVD Analysis for Classification')
        plt.xticks(n comp)
        # plt.grid()
        plt.show()
    if kind == 'R':
        print('-----------------------------------------------------------------
------')
```

```
        print('          Singular Value Decomposition (SVD) for Regression
')
        print('------------------------------------------------------------------
------')
        svd = TruncatedSVD(n_components=(X.shape[1] - 1))
        svd.fit(X)

        # Original data shape
        print("Original Data Shape:", X.shape)

        # Reduced data shape
        # print("Reduced Data Shape:", data_reduced.shape)
        n_comp = [i for i in range(1, X.shape[1])]
        # Explained variance ratio
        print("Explained Variance Ratio:",
svd.explained_variance_ratio_.sum())

        # Visualize explained variance
        cum_exp_variance = np.cumsum(svd.explained_variance_ratio_)
        plt.figure(figsize=(14, 6))
        plt.plot(n_comp, np.cumsum(svd.explained_variance_ratio_))
        plt.axhline(y=0.9, linestyle='-', color='red')
        # print(x=)
        plt.axvline(x=len([var for var in cum_exp_variance if var < 0.9]),
linestyle='-', color='red')
        plt.xlabel('Number of Components')
        plt.ylabel('Cumulative Explained Variance')
        plt.title('SVD Analysis for Regression')
        plt.xticks(n_comp)
        # plt.grid()
        plt.show()


# compute the vif for all given features
def compute_vif(considered_features):
    X = df[considered_features]
    # the calculation of variance inflation requires a constant
    #     X = add_constant(X)

    # create dataframe to store vif values
    vif = pd.DataFrame()
    vif["Variable"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in
range(X.shape[1])]
    vif = vif[vif['Variable'] != 'intercept']
    return vif


if __name__ == "__main__":
    start = time.time()
    # Read dataset as Pandas DataFrame
    df = pd.read_csv('credit_score_classification_data.csv',
low_memory=False)

    # ----------------------------OBSERVATIONS----------------------------
```

```
---
    # 1. We will inspect the features (numeric and categorical)
    # 2. We will check for missing values
    # 3. We will drop columns like ID, Name, SSN (Contain NaN values and no
impact on the classification task)
    # 4. Dataset has no duplicates - as checked below
    # 5. We see that Customer_ID has 12500 unique values that occur 8 times
each. (Groupby to fill missing values)
    # Candidates for Regression - Outstanding Debt, Monthly Balance
    # ------------------------------------------------------------------------
----
    # Basic EDA
    basic_eda(df)

    # Missing Value Analysis
    print('Before dropping general info columns')
    missing_value_check(df)
    print('After dropping general info columns')

    # Drop general info columns by passing list of columns to drop
    df.drop(['Name', 'ID', 'SSN'],axis=1,inplace=True)
    missing_value_check(df)
    print(df.columns)
    # -------------------------------------------------
    #                 Drop NaN values
    # -------------------------------------------------
    # df.dropna(how='any',inplace=True,axis=0)
    # print('Dropped NaN values')
    # Check duplicates
    print('-----------Check Duplicates-------------')
    df.duplicated().value_counts()
    print('No duplicates found')
    print('----------------------------------------')

    # Check for imbalance for Target feature Credit_Score
    imbalance_check(df, 'Credit_Score')
    print('--------------------Encoding Target Feature--------------------')
    df['Credit_Score'].replace({'Good': 2, 'Standard': 1, 'Poor': 0},
inplace=True)

    # Check if Customer_ID is unique
    # We will use this column to fill missing values
    # print(df['Customer_ID'].value_counts())

    # ---------------------------------------------------
    #             Data Cleaning - Numerical columns
    # ---------------------------------------------------
    print(' ---------------------------------------------------------')
    print('             Data Cleaning - Numerical columns           ')
    print(' ---------------------------------------------------------')

    num_cols_to_fix = ['Age', 'Annual_Income', 'Num_of_Loan',
'Num_of_Delayed_Payment', 'Changed_Credit_Limit',
                       'Outstanding_Debt', 'Amount_invested_monthly',
'Monthly_Balance']
```

```
    for num_col in num_cols_to_fix:
        print('Column Name: ' + num_col)
        print("**" * 24)
        print(df[num_col].value_counts())
        print('END', "--" * 22, '\n')
    # We wil replace if any string like '_' with empty string but with np.nan
    for i in num_cols_to_fix:
        df[i] = df[i].apply(clean_numeric_data)
    print('------------After Cleaning Numerical columns of object type-------
---------')
    for num_col in num_cols_to_fix:
        print('Column Name: ' + num_col)
        print("**" * 24)
        print(df[num_col].value_counts(dropna=False))
        print('END', "--" * 22, '\n')
    # print(df[df['Amount_invested_monthly']])
    print(df.info())

    # # Check unique values in each numerical column
    # # numeric = df.select_dtypes(exclude='object')
    # print('Numerical features described:')
    # print(df_cleaned.describe(exclude='object').T)
    # print('Categorical features described:')
    # print(df_cleaned.describe(exclude=np.number).T)
    # print('Missing Value Check')
    # missing_value_check(df_cleaned)

    # We are now left with the following object type columns
    # -----------------------------------------------------------
    #         Fix Strange values in Object type columns
    # -----------------------------------------------------------
    print('-------------------------------------------------------------')
    print('         Data Cleaning in Object type columns      ')
    print('-------------------------------------------------------------')
    # Fix strange values in Object type columns
    object_cols = df.select_dtypes(include='object').columns
    print(object_cols)
    for col in object_cols:
        print('Column Name: ' + col)
        print("**" * 24)
        print(df[col].value_counts(dropna=False))
        print('END', "--" * 22, '\n')

    #
['Month','Occupation','Type_of_Loan','Credit_Mix','Credit_History_Age','Payme
nt_of_Min_Amount']
    # ---------------------------------------------------------------
    #         Replace strange values in the object type columns
    # ---------------------------------------------------------------
    print('-------------------------------------------------------------------')
    print('        Replace strange values in the object type columns    ')
    print('-------------------------------------------------------------------')
    # 1. Month
    # print(df['Month'].value_counts(dropna=False))
    print(df['Month'].dtypes)
```

```
    # month_tab = df.groupby('Credit_Score')['Month'].value_counts()

    # 2. Occupation
    # Replace garbage value(_______) with mode based on customer_id
    print('-------------------------Occupation-------------------------')
    print('Before')
    print(df['Occupation'].value_counts())
    # Replace with mode w.r.t Customer_ID
    df['Occupation'] = df['Occupation'].replace('_______', np.nan)
    mode_fn = lambda x: x.mode().iat[0]
    df['Occupation'] =
df['Occupation'].fillna(df.groupby('Customer_ID')['Occupation'].transform(mod
e_fn))
    print('After')
    print(df['Occupation'].value_counts())
    plt.figure(figsize=(24, 12))
    sns.countplot(data=df, x='Occupation', hue='Credit_Score')
    plt.title('Occupation wise Credit Score')
    plt.show()

    # # 3. Type_of_Loan
    print('-------------------------Type of Loan-------------------------')
    print('Before')
    print(df['Type_of_Loan'].value_counts().head(10))
    # # We see that we have comma separated values which we can convert to
columns
    # # We can look at the Types of Loans as categorical columns
    # # We have types - Credit Builder Loan, Personal Loan, Mortgage Loan,
Student Loan
    # # We will now assign 0 or 1 to each column of unique loan types
    # # We also have a category Not Specified which we can either include or
drop
    # # Now replacing Type of Loan could have a great effect on credit score
hence replacing would be counterproductive
    # # Hence we include only types other than Not Specified and we split
them up into columns
    # # tl = ''
    # # print(type('kadka'))
    # # for i in df['Type_of_Loan'].values:
    # #
    # #     if type(i) == str:
    # #         tl += i + ','
    # # print(len(tl))
    # # uniqueLoanTypes = set(tl.split(","))
    # # print(uniqueLoanTypes)
    #
    uloantypes = ['Debt Consolidation Loan', 'Student Loan', 'Mortgage Loan',
'Auto Loan',
                  'Personal Loan', 'Not Specified', 'Payday Loan', 'Student
Loan', 'Mortgage Loan',
                  'Home Equity Loan', 'Debt Consolidation Loan', 'Student
Loan', 'Credit-Builder Loan',
                  'Payday Loan', 'Auto Loan', 'Mortgage Loan', 'Personal
Loan', 'Auto Loan', 'Debt Consolidation Loan',
                  'Credit-Builder Loan', 'Home Equity Loan', 'Not Specified',
```

```
'Not Specified', 'Credit-Builder Loan',
                  'Payday Loan', 'Personal Loan', 'Home Equity Loan']
    unique_loan_types = set(uloantypes)
    # print('We have {} unique loan types: ',len(unique_loan_types))
    # print(unique_loan_types)
    # Handle Type of Loan =  Not Specified
    # df['Type_of_Loan'].dropna(how='any', axis=0, inplace=True)
    df.dropna(how='any', axis=0, subset=['Type_of_Loan'], inplace=True)
    #
print(df.groupby('Type_of_Loan')['Customer_ID'].value_counts(dropna=False))
    # print(df[df['Type_of_Loan'].isna()]['Customer_ID'].head())
    # One-Hot Encoding of Type_of_Loan column
    print('One-Hot Encoding of Type_of_Loan column')
    for i in unique_loan_types:
        typeLoan = []
        for value in df['Type_of_Loan']:
            if type(value) == str:
                typeLoan.append(1 if i in value else 0)
        df[i] = typeLoan
    print('After handling Type of Loan column:', df.columns)
    print(df.tail(10))
    # We can now drop the column Type_of_Loan column
    df.drop('Type_of_Loan', axis=1, inplace=True)
    # Avoid Dummy variable trap
    df.drop('Not Specified',axis=1,inplace=True)
    # # df['Type_of_Loan'].value_counts()
    # # for i in df['Type_of_Loan'].values:
    # #     flag = 0
    # #     if(type(i) == str):
    # #         for typeLoan in uloantypes:
    # #             if typeLoan in i:
    # #                 flag = 1
    # #         if flag == 0:
    # #             print(i)

    # Working
    # df = pd.concat([df, df['Type_of_Loan'].str.split(', ', expand=True)],
axis=1)
    # print('After handling Type of Loan column:', df.columns)
    # print(df.tail())

    # 4. Credit_Mix
    print('------------------- Credit_Mix -------------------')
    print('Before Replace')
    print(df['Credit_Mix'].value_counts())
    # Replace with mode
    df['Credit_Mix'] = df['Credit_Mix'].replace('_', np.nan)
    mode_fn = lambda x: x.mode().iat[0]
    df['Credit_Mix'] =
df['Credit Mix'].fillna(df.groupby('Customer ID')['Credit Mix'].transform(mod
e_fn))
    # df['Credit_Mix'].replace({'_': 'Standard'}, inplace=True)  # replace _
with mode = Standard
    print('After replace')
    print(df['Credit_Mix'].value_counts())
```

```
    # # 5. Credit_History_Age
    print('-------------------- Credit_History_Age --------------------')
    # # It is in the form of years and months we will convert to numeric
values in the form of Months
    # # We have NaN values as well here
    # # We will replace with mean of Months
    # # 15 Years and 11 Months is the format
    df['Credit_History_Age'] =
df['Credit_History_Age'].apply(credit_history_age)
    print(df['Credit_History_Age'].head(10))
    # Fill NaN values
    mean_fn = lambda x: x.mean()
    df['Credit_History_Age'] =
df.groupby('Customer_ID')['Credit_History_Age'].transform(mean_fn)
    print('Misisng Value Check after filling NaN values')
    print('There are {} missing values after
cleaning:'.format(df['Credit_History_Age'].isna().sum()))
    print(df['Credit_History_Age'].head(10))

    # Payment_of_Min_Amount
    print('-------------------- Payment_of_Min_Amount --------------------')
    # If payment of min amount due has been done or not
    print(df['Payment_of_Min_Amount'].value_counts(dropna=False))
    print(df['Payment_of_Min_Amount'].isna().sum())
    mode_fn = lambda x: x.mode().iat[0]
    df['Payment_of_Min_Amount'].replace('NM', np.nan, inplace=True)
    df['Payment_of_Min_Amount'] =
df.groupby('Customer_ID')['Payment_of_Min_Amount'].transform(mode_fn)
    print(df['Payment_of_Min_Amount'].value_counts(dropna=False))
    sns.countplot(data=df, x='Payment_of_Min_Amount', hue='Credit_Score')
    plt.title('Min Amount Due based Credit Score')
    plt.show()

    # Check df
    print(df.shape)
    missing_value_check(df)
    print(df.info())

    # ------------------------------------------------------------
    #               Handle remaining NaN values
    # ------------------------------------------------------------

    # 1. Monthly_Inhand_Salary
    # We can replace with mean of Customer_ID
    df['Monthly_Inhand_Salary'] =
df.groupby('Customer_ID')['Monthly_Inhand_Salary'].transform(mean_fn)

    # 2. Num_of_Delayed_Payment
    # We replace with mean of Customer ID
    df['Num_of_Delayed_Payment'] =
df.groupby('Customer_ID')['Num_of_Delayed_Payment'].transform(mean_fn)

    # 3. Changed_Credit_Limit
    # Replace with mean of Customer_ID
```

```
    df['Changed_Credit_Limit'] =
df.groupby('Customer_ID')['Changed_Credit_Limit'].transform(mean_fn)

    # 4. Num_Credit_Inquiries
    df['Num_Credit_Inquiries'] =
df.groupby('Customer_ID')['Num_Credit_Inquiries'].transform(mean_fn)

    # 5. Amount_invested_monthly
    df['Amount_invested_monthly'] =
df.groupby('Customer_ID')['Amount_invested_monthly'].transform(mean_fn)

    # 6. Monthly_Balance
    df['Monthly_Balance'] =
df.groupby('Customer_ID')['Monthly_Balance'].transform(mean_fn)

    # Missing value check
    print('After cleaning remaining NaN values:')
    missing_value_check(df)

    # Check Numerical columns for wrong values
    # Example: Age in negative, Monthly_Balance in negative
    # use describe to check
    # we can ignore the encoded columns like types of loan
    numeric_df = df.select_dtypes(exclude='object')
    print(numeric_df.describe().T.head(17))

    # Observations
    # 1. Age: Min value = -500, Max value = 867  clean up age
    # 2. Num_Bank_Accounts: min value = -1 we can make it 0
    # 3. Num_Credit_Card: Outlier Detection
    # 4. Interest_Rate:  Outlier Detection
    # 5. Num_of_Loan: Min value = -100
    # 6. Delay_from_due_date: Min value = -5
    # 7. Num_of_Delayed_Payment: Min value = -1
    # 8. Changed_Credit_Limit: Min value = -1.07 (Explore Later)
    # 9. Num_Credit_Inquiries: Min value = 0.0  Max value: 600
    # 10. Outstanding_Debt: Min Value = 0.23  Max value = 4998
    # 11. Credit_Utilization_Ratio: Min value = 20  Max value = 49.5
    # 12. Credit_History_Age: Min value = 4.5  Max value = 400
    # 13. Total_EMI_per_month: Min value = 4.46  Max value = 82331
    # 14. Amount_invested_monthly: Min value = 15.2  Max value = 1313
    # 15. Monthly_Balance: Min value = large -ve  Max value = 1313, garbage
value = 3333333333
    # 16. Annual_Income: Min value = 7005   Max value = large +ve
    # 17. Monthly_Inhand_Salary: Min value = 303.6  Max value = 15,204

    # -------------------------------------------------
    #                 OUTLIER DETECTION
    # -------------------------------------------------

    # 1. Age
    print('-----------------------------------------')
    print('                 Age                     ')
    print('-----------------------------------------')
    # Handle negative values and values greater than 100
```

```
    df = df[(df['Age'] > 0) & (df['Age'] < 100)]
    print(df.shape)
    imbalance_check(df,'Credit_Score')

    # 2. Num_Bank_Accounts
    # Replace -1 with 0
    df.loc[df['Num_Bank_Accounts'] < 0, 'Num_Bank_Accounts'] = 0
    sns.boxplot(data=df, x='Num_Credit_Card')
    plt.title('Number of Bank Accounts with Outliers')
    plt.show()
    # print('Outlier instances in Num_Bank_Accounts: ',
df[df['Num_Bank_Accounts'] > 8 & df['Num_Bank_Accounts']].shape[0])
    q25 = df['Num_Bank_Accounts'].quantile(0.25)
    q75 = df['Num_Bank_Accounts'].quantile(0.75)
    iqr = q75 - q25
    upper = q75 + 1.5 * iqr
    # lower = q25 - 1.5 * iqr
    lower = 0
    df = df[(df['Num_Bank_Accounts'] >= lower) & (df['Num_Bank_Accounts'] <=
upper)]
    print('After Removing Outliers from # of Bank Accounts')
    print(df.shape)

    # 3. Num_Credit_Card
    print('-----------------------------------------------')
    print('              Num_Credit_Card                  ')
    print('-----------------------------------------------')
    # Detect Outliers we can use IQR range to detect outliers
    sns.boxplot(data=df, x='Num_Credit_Card')
    plt.title('Number of Credit Cards with Outliers')
    plt.show()
    print('Outlier instances in Num_Credit_Card:
',df[df['Num_Credit_Card']>8].shape[0])
    q25 = df['Num_Credit_Card'].quantile(0.25)
    q75 = df['Num_Credit_Card'].quantile(0.75)
    iqr = q75 - q25
    upper = q75 + 1.5 * iqr
    # lower = q25 - 1.5 * iqr
    lower = 0
    df = df[(df['Num_Credit_Card'] >= lower) & (df['Num_Credit_Card'] <=
upper)]
    print('After Removing Outliers from # of Credit Cards')
    print(df.shape)
    sns.boxplot(data=df, x='Interest_Rate')
    plt.title('Interest Rate after Outlier Removal')
    plt.show()

    # Interest_Rate
    # Detect Outliers after checking distribution
    print('-----------------------------------------------')
    print('              Interest_Rate                    ')
    print('-----------------------------------------------')
    sns.boxplot(data=df, x='Interest_Rate')
    plt.title('Interest Rate before Outlier Removal')
    plt.show()
```

```
    q25 = df['Interest_Rate'].quantile(0.25)
    q75 = df['Interest_Rate'].quantile(0.75)
    iqr = q75 - q25
    print('25th',q25)
    print('75th',q75)
    print('IQR of Interest Rate',iqr)
    upper = q75 + 1.5 * iqr
    # lower = q25 - 1.5 * iqr
    lower = 0.0  # Interest Rate low is 0.0
    print('Interest Rate IQR')
    print('Upper:', upper)
    print('Lower:', lower)
    print('After Removing Outliers from Interest_Rate')
    df = df[(df['Interest_Rate'] >= lower) & (df['Interest_Rate'] <= upper)]
    print(df.shape)
    sns.boxplot(data=df, x='Interest_Rate')
    plt.title('Interest Rate after Outlier Removal')
    plt.show()

    # Num_of_Loan
    print('-----------------------------------------')
    print('                Num_of_Loan              ')
    print('-----------------------------------------')
    # We have -ve values which is not right, we will set lower limit to 0
    df.loc[df['Num_of_Loan']<0, 'Num_of_Loan'] = 0
    sns.boxplot(data=df, x='Num_of_Loan')
    plt.title('Num_of_Loan before Outlier Removal')
    plt.xlim(min(df['Num_of_Loan']), max(df['Num_of_Loan']))
    plt.title('Number of Loans before Outlier Removal')
    plt.show()
    q25 = df['Num_of_Loan'].quantile(0.25)
    q75 = df['Num_of_Loan'].quantile(0.75)
    iqr = q75 - q25
    print('25th:', q25)
    print('75th:', q75)
    print('IQR:', iqr)
    upper = q75 + 1.5 * iqr
    # lower = q25 - 1.5 * iqr
    lower = 0.0
    print('# of Loans IQR')
    print('Upper:', upper)
    print('Lower:', lower)
    print('After Removing Outliers from # of Loans')
    df = df[(df['Num_of_Loan'] >= lower) & (df['Num_of_Loan'] <= upper)]
    df['Num_of_Loan'] = df['Num_of_Loan'].astype(int)
    # Check credit score of very high Num_of_Loan
    sns.boxplot(data=df,x='Num_of_Loan',hue='Credit_Score')
    plt.title('Num_of_Loan after outlier removal')
    plt.show()
    print(df[(df['Num of Loan'] >= lower) & (df['Num of Loan'] <=
upper)].shape)
    print('Credit Score based on Num of Loans!!!')
    # loans_mean = df.groupby('Credit_Score')['Num_of_Loan'].mean()
    # num_loan_tab = pd.pivot_table(data=df,
values=['Credit_Score','Num_of_Loan'], index='Credit_Score')
```

```
    # pd.crosstab(index=df['Credit_Score'],columns=loans_mean))
    print(df[['Num_of_Loan', 'Credit_Score']].groupby('Credit_Score').mean())

    # sns.countplot(data=df,x='Num_of_Loan',hue='Credit_Score')

    # Delay_from_due_date
    print('-------------------------------------------')
    print('            Delay_from_due_date            ')
    print('-------------------------------------------')
    sns.boxplot(data=df, x='Delay_from_due_date')
    plt.title('Delay_from_due_date before outlier removal')
    plt.show()
    print('No. of negative values:', df[df['Delay_from_due_date'] <
0].shape[0])
    print('Filtering out negative values from Delay_from_due_date')
    print('After removing outliers from Delay_from_due_date')
    df = df[df['Delay_from_due_date'] >= 0]
    sns.boxplot(data=df, x='Delay_from_due_date')
    plt.title('Delay_from_due_date after outlier removal')
    plt.show()
    # print(df[df['Delay_from_due_date'] < 0])
    print('No.of records:', df.shape[0])
    print('-----Delay_from_due_date vs Credit Score-----')
    print(df[['Delay_from_due_date',
'Credit_Score']].groupby('Credit_Score').mean())

    # Num_of_Delayed_Payment
    print('-------------------------------------------')
    print('           Num_of_Delayed_Payment          ')
    print('-------------------------------------------')
    print('Min:', min(df['Num_of_Delayed_Payment']))
    print('Max:', max(df['Num_of_Delayed_Payment']))
    q25 = df['Num_of_Delayed_Payment'].quantile(0.25)
    q75 = df['Num_of_Delayed_Payment'].quantile(0.75)
    iqr = q75 - q25
    print('IQR:',iqr)
    upper = q75 + 1.5 * iqr
    print("Upper:", upper)
    print('-----Num_of_Delayed_Payment vs Credit Score-----')
    print(df[['Num_of_Delayed_Payment',
'Credit_Score']].groupby('Credit_Score').mean())

    # Changed_Credit_Limit
    print('-------------------------------------------')
    print('            Changed_Credit_Limit           ')
    print('-------------------------------------------')
    print(df['Changed_Credit_Limit'].value_counts(dropna=False))
    print('Min:',min(df['Changed_Credit_Limit']))
    print('Max:',max(df['Changed_Credit_Limit']))
    # We are capturing the change so keep it above zero
    q25 = df['Changed_Credit_Limit'].quantile(0.25)
    q75 = df['Changed_Credit_Limit'].quantile(0.75)
    iqr = q75 - q25
    print('IQR:', iqr)
    upper = q75 + 1.5 * iqr
```

```
    print("Upper:", upper)
    print('-----Monthly Balance vs Credit Score-----')
    print(df[['Monthly_Balance',
'Credit_Score']].groupby('Credit_Score').mean())

    # Monthly_Balance
    print('-------------------------------------------')
    print('             Monthly_Balance               ')
    print('-------------------------------------------')
    print('# of Negative values are:', df[df['Monthly_Balance'] <
0].shape[0])
    # Include only positive values
    df = df[df['Monthly_Balance'] > 0]
    # We are capturing the change so keep it above zero
    q25 = df['Monthly_Balance'].quantile(0.25)
    q75 = df['Monthly_Balance'].quantile(0.75)
    iqr = q75 - q25
    print('IQR:', iqr)
    upper = q75 + 1.5 * iqr
    print("Upper:", upper)
    print('We will include only positive(>0) values')
    print('-----Monthly Balance vs Credit Score-----')
    # print(df[df['Monthly_Balance']==-
333333333333333333333333333].sample(2))
    print(df[['Monthly_Balance',
'Credit_Score']].groupby('Credit_Score').mean())

    # # Outstanding_Debt
    print('-------------------------------------------')
    print('             Outstanding_Debt              ')
    print('-------------------------------------------')
    q25 = df['Outstanding_Debt'].quantile(0.25)
    q75 = df['Outstanding_Debt'].quantile(0.75)
    iqr = q75 - q25
    print('IQR:', iqr)
    upper = q75 + 1.5 * iqr
    print("Upper:", upper)

    # Basic EDA
    # basic_eda(df)

    # ----------------------Split into X & y---------------------
    X, y = df.drop('Credit_Score', axis=1), df['Credit_Score']
    reg_target_org = df['Outstanding_Debt'].head(50000).values

    assoc_df = X[['Occupation','Credit_Mix','Payment_of_Min_Amount','Debt
Consolidation Loan','Auto Loan',
                  'Credit-Builder Loan','Home Equity Loan',
                   'Personal Loan','Student
Loan']].sample(10000,random state=5805)
    print(assoc_df.columns)
    assoc_df.to_csv('Association.csv',index=False)
    print('Association Rule Mining Dataset is ready!')
    # ------------------------------------------------------------
    #                      Standardization
```

```
    # ------------------------------------------------------------

    df_standardized = standardize_data(X)

    # ------------------------------------------------------------------
    #                   One-Hot Encoding
    # ------------------------------------------------------------------
    print('------------------------------------------------------')
    print('                  One-Hot Encoding                    ')
    print('------------------------------------------------------')
    object_df = df.select_dtypes(include='object')
    # print(object_df.head())
    # ['Payment_of_Min_Amount', 'Credit_Mix', 'Occupation',
'Payment_Behaviour','Month']
    print('-----Encoding in progress-----')
    one_hot_df = pd.get_dummies(data=object_df[['Payment_of_Min_Amount',
'Credit_Mix', 'Occupation', 'Payment_Behaviour', 'Month']], drop_first=True)
    # Payment_of_Min_Amount: {'No':0, 'Yes':1}
    # print(one_hot_df.head())
    print('Columns:', one_hot_df.columns)
    # Concat one_hot_df with original df
    df = pd.concat([df_standardized, one_hot_df], axis=1)
    df = pd.concat([df, y], axis=1)
    df.drop(['Payment_of_Min_Amount', 'Credit_Mix', 'Occupation',
'Payment_Behaviour', 'Month'], axis=1, inplace=True)

    # Now that we have used Customer_ID as reference for filling our missing
values and data cleaning
    # We can now drop Customer_ID
    df.drop('Customer_ID', axis=1, inplace=True)
    # basic_eda(df)
    # print(df.head())
    print('Shape of data before X-y split:',df.shape)

    X, y = df.drop('Credit_Score', axis=1), df['Credit_Score']
    # ------------------------------------------------
    #             Prepare Regression Dataset
    # ------------------------------------------------
    X_reg,y_reg = df.drop('Outstanding_Debt', axis=1), df['Outstanding_Debt']
    # ----------------------------------------------------------
    #                   Feature Importance
    # ----------------------------------------------------------
    # print('------------------------------------------------------------------
')
    # print('                  Feature Selection Techniques
')
    print('------------------------------------------------------------------')
    print('Random Forest based feature selection for Classification Dataset')
    clf_selected_features = conduct_rforest_selection(X, y, kind='C')
    print('Random Forest based feature selection for Regression Dataset')
    reg_selected_features = conduct_rforest_selection(X_reg, y_reg, kind='R')
    rf_reg = X_reg[reg_selected_features]
    rf_reg = rf_reg.head(50000)

    # # df_reg = X_reg[reg_selected_features]
```

```
    df_reg = X_reg
    df_reg = df_reg.head(50000)
    # df_reg = df_reg.head(50000)
    # df_target = pd.DataFrame(reg_target_org)
    df_target = pd.DataFrame(y_reg)
    rf_reg = pd.concat([rf_reg, df_target.head(50000)], axis=1)
    print('Original Regression Dependent Mean:', np.mean(reg_target_org))
    df_target = df_target.head(50000)
    df_reg = pd.concat([df_reg, df_target], axis=1)
    df_reg['Target_Original'] = reg_target_org
    print('Regression Data shape (after feature selection):', df_reg.shape)
    rf_reg.to_csv('rf_regression.csv', index=False)
    print('Regression Data Random Forest Features dataset ready')
    df_reg.to_csv('Debt_Regression_cleaned.csv',index=False)
    print('Regression Data csv file ready!')

    # ------------------------------------------------------------------------
-
    #           Principal Component Analysis (PCA) and Condition Number
    # ------------------------------------------------------------------------
-
    print('METHOD 2: Principal Component Analysis (PCA) for Regression')
    X_reg = X_reg.head(50000)
    y_reg = y_reg.head(50000)
    pca_reg = conduct_PCA(X_reg, y_reg, kind='R')
    print('Condition Number for Regression Dataset')
    get_condition_num(X_reg, y_reg, pca_reg, n=25)
    print('METHOD 2: Principal Component Analysis (PCA) for Classification')
    pca_clf = conduct_PCA(X, y, kind='C')
    print('Condition Number for Classification Dataset')
    get_condition_num(X, y, pca_clf, n=25)

    # ------------------------------------------------------
    #          Singular Value Decomposition(SVD)
    # ------------------------------------------------------
    # print('SVD reading is in process')
    print('METHOD 3: Singular Value Decomposition (SVD) for Classification')
    truncated_svd(X, kind='C')
    print('METHOD 3: Singular Value Decomposition (SVD) for Regression')
    truncated_svd(X_reg, kind='R')

    ## ------------------------------------------------------
    ##              Variance Inflation Factor(VIF)
    ## ------------------------------------------------------
    print('METHOD 4: VIF (Variance Inflation Factor)')
    features = list(X_reg.columns)
    df_vif = compute_vif(features)
    print(df_vif.head(10).sort_values(by='VIF',ascending=False))

    # ------------------------------------------------------
    #              Balancing the dataset
    # ------------------------------------------------------
    # Check imbalance
    imbalance_check(df, col='Credit_Score')
    # We can see that we have imbalanced dataset, we will use SMOTE
```

```
oversampling to balance the dataset
    over_sampling = SMOTE(sampling_strategy='auto', random_state=5805)
    X_columns = X.columns
    X, y = over_sampling.fit_resample(X, y)
    print('Oversampled X:', X.shape)
    down_sample = RandomUnderSampler(sampling_strategy={0: 18000, 1: 18000,
2: 18000}, random_state=5805)
    X, y = down_sample.fit_resample(X, y)
    enn_down_sample = TomekLinks(sampling_strategy='all', n_jobs=3)
    X, y = enn_down_sample.fit_resample(X, y)
    df1 = pd.DataFrame(X, columns=list(X_columns))
    df2 = pd.DataFrame(y,columns=['Credit_Score'])
    df3 = pd.concat([df1, df2], axis=1)
    # Check the balance
    print('Balance the dataset')
    imbalance_check(df3, col='Credit_Score')
    # print(df3.columns)
    # selected_features = ['Outstanding_Debt', 'Interest_Rate',
'Delay_from_due_date', 'Credit_Utilization_Ratio', 'Changed_Credit_Limit',
'Credit_Mix_Standard', 'Num_Credit_Inquiries', 'Credit_History_Age',
'Num_of_Delayed_Payment', 'Credit_Mix_Good', 'Num_Credit_Card',
'Total_EMI_per_month', 'Age', 'Monthly_Balance', 'Annual_Income',
'Amount_invested_monthly', 'Monthly_Inhand_Salary', 'Num_Bank_Accounts',
'Num_of_Loan', 'Payment_of_Min_Amount_Yes', 'Month_July', 'Month_August',
'Payment_Behaviour_Low_spent_Small_value_payments', 'Month_March',
'Month_February', 'Month_January']
    print('Shape of cleaned df:', df3[clf_selected_features].shape)
    df3.drop_duplicates(inplace=True)
    print('Shape of cleaned df:', df3[clf_selected_features].shape)
    # df4 = pd.concat([df3[clf_selected_features],df2])
    # #Save preprocessed and balanced dataset into new csv for Classification
dataset
    clf_selected_features.append('Credit_Score')
    df3[clf_selected_features].to_csv('Credit_Classification_cleaned.csv',
index=False)
    print('Classification Data csv file ready!')


    ## Association Rule Mining Dataset Preparation
    assoc_df = X[['Credit_Mix_Good',
                  'Credit_Mix_Standard','Payment_of_Min_Amount_Yes','Debt
Consolidation Loan','Auto Loan','Credit-Builder Loan','Home Equity Loan',
                  'Personal Loan','Student
Loan']].sample(8000,random_state=5805)
    assoc_df.to_csv('Association.csv', index=False)
    #
'Occupation_Scientist','Occupation_Engineer','Occupation_Lawyer','Occupation_
Architect','Occupation_Teacher'
    #
,'Occupation Doctor','Occupation Developer','Occupation Entrepreneur','Occupa
tion_Journalist','Occupation_Musician',
    #
'Occupation_Manager','Occupation_Media_Manager','Occupation_Mechanic','Occupa
tion_Writer'
    # # ------------------------------------------------
```

```
    # #               Correlation Analysis
    # # ------------------------------------------------------
    print('------------------------------------------------------')
    print('            Correlation Analysis                   ')
    print('------------------------------------------------------')
    temp_df = df.drop(list(one_hot_df.columns), axis=1)
    loan_list = []
    for i in uloantypes:
        if i in list(temp_df.columns):
            loan_list.append(i)
    temp_df.drop(loan_list,axis=1,inplace=True)
    corr = temp_df.corr()
    # # print(temp_df.columns)
    plt.figure(figsize=(24,18))
    plt.title('Correlation Matrix')
    sns.heatmap(corr,annot=True)
    plt.show()

    # # ------------------------------------------------------
    # #               Covariance Analysis
    # # ------------------------------------------------------
    print('------------------------------------------------------')
    print('            Covariance Analysis                    ')
    print('------------------------------------------------------')
    # temp_df = df.drop(list(one_hot_df.columns),axis=1)
    cov = temp_df.cov()
    plt.figure(figsize=(24, 18))
    plt.title('Covariance Matrix')
    sns.heatmap(cov, annot=True)
    plt.show()

    end = time.time()
    print('Time taken in minutes:', round((end-start)/60,2))
```

Regression.py – Regression Analysis Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
import statsmodels.api as sm
from prettytable import PrettyTable
import warnings
warnings.simplefilter(action='ignore')
```

```
pd.set_option('display.max_columns', 100)


def reverse_standardize_data(df, y):
    original_df = (df * y.std()) + y.mean()
    # print('Mean(original):',y.mean())
    # print('STD:',y.std())
    return original_df


# Read Data
df = pd.read_csv('Debt_Regression_cleaned.csv')
# print(df.head())
print('Shape:', df.shape)
# print(df.columns)
# Drop additional columns
# df.drop(['Unnamed: 0'], axis=1, inplace=True)
y_org = df['Target_Original']
print('Y-Original Mean:', np.mean(y_org))
df.drop('Target_Original', axis=1, inplace=True)
X, y = df.drop('Outstanding_Debt', axis=1), df['Outstanding_Debt']
print('Shape X:', X.shape)
# print(X.columns)
# X = X.head(40000)
# y = y.head(40000)
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=5805)

# Result Table:
result_table = PrettyTable(field_names=['Model', 'R-squared', 'AIC', 'BIC',
'Adj_R2','MSE'])
result_table.title = 'Multiple Linear Regression Result'
# Regression Analysis using OLS
# We have list of selected and eliminated features
print('Stepwise Regression Analysis (OLS)')

OLS_features = list(X_train.columns)
OLS_features_eliminated = []
table = PrettyTable(field_names=['Drop Feature','AIC','BIC','AdjR2','p-
value','f-statistic'])
best_adj_R2 = 0
first = 0
while len(OLS_features) > 0:
    # print('Entered')
    X_train = sm.add_constant(X_train[OLS_features])
    model = sm.OLS(y_train, X_train).fit()
    if first == 0:
        print('Model Summary before Regression Analysis OLS')
        print(model.summary())
        first += 1
    print('F-value:', round(model.fvalue,3))
    curr_adj_R2 = model.rsquared_adj
    max_p_value = model.pvalues.max()
    print('Max p-value:', round(max_p_value,3))
```

```
    print('Current Adjusted R-squared:', round(curr_adj_R2,3))
    max_p_value_index = model.pvalues.idxmax()
    p_values = model.pvalues[0:]
    threshold = 0.01
    if max_p_value > threshold:
        # print('Entered if')
        # if curr_adj_R2 >= best_adj_R2:
        drop_feature = max_p_value_index
        OLS_features_eliminated.append(drop_feature)
        OLS_features.remove(drop_feature)
        print('Feature to be dropped:', drop_feature)

table.add_row([drop_feature,round(model.aic,3),round(model.bic,3),round(model
.rsquared_adj,3),round(max_p_value, 3),
                       round(model.fvalue,3)])
        # table = table.append({'Drop Feature': drop_feature, 'AIC':
model.aic,
        #                       'BIC': model.bic, 'AdjR2':
model.rsquared_adj,
        #                       'p-value': round(max_p_value, 3), 'f-
statistic': round(model.fvalue,3),
        #                       }, ignore_index=True)
        best_adj_R2 = model.rsquared_adj
        # else:
        #     print('Adj-Rsquared saturation point achieved')
        #     print(model.summary())
        #     break
    else:
        print('Regression Analysis complete')
        print(model.summary())
        break
print(table)
# table.to_csv('Regression_Analysis_OLS.csv')
print('Final Adj R-squared:', round(model.rsquared_adj, 3))
print('Final F-statistic:',round(model.fvalue, 3))
# Drop features based on correlation matrix
# OLS_features.remove('Credit_History_Age')
# OLS_features.remove('Monthly_Inhand_Salary')
print('Eliminated Features:', OLS_features_eliminated)
print('Selected features:', OLS_features)
print(f'After Regression {len(OLS_features)} features are selected')

# Update Result table
# result_table.add_row(['Multiple Linear Regression',mlr_model.])
# Create Train and Test Sets based on selected features
X_train = X_train[OLS_features]
X_test = X_test[OLS_features]
# mlr_OLS = sm.OLS(sm.add_constant(X_train),y_train).fit()
mlr_model = LinearRegression()
mlr_model.fit(X_train, y_train)
y_pred_reg = mlr_model.predict(X_test)
# y_pred_reg = mlr_OLS.predict(sm.add_constant(X_test))
# Our predictions are standardized, so we will reverse standardize them based
on y_original
y_pred_org = reverse_standardize_data(y_pred_reg, y_org)
```

```
y_test_org = reverse_standardize_data(y_test, y_org)
y_train_org = reverse_standardize_data(y_train, y_org)
print('y-train shape:',y_train_org.shape)
print('y-test shape:',y_test_org.shape)
print('y-pred shape:',y_pred_org.shape)

# Mean Squared Error
mse = metrics.mean_squared_error(y_test, y_pred_reg)
# mae = metrics.mean_absolute_error(y_test, y_pred_reg)
print('Mean Squared Error:', mse.round(3))
# print('Root Mean Squared Error:', round(np.sqrt(mse), 3))
# print('Mean Absolute Error:', round(mae, 3))
# Update Result table
result_table.add_row(['Multiple Linear Regression', round(model.rsquared, 3),
round(model.aic,3),
                      round(model.bic, 3), round(model.rsquared_adj, 3),
round(mse, 3)])
print('Results of Linear Regression')
print(result_table)

# ------------------------------------------------------------------
#       Random Forest Selected Features Linear Regression
# ------------------------------------------------------------------
print('--------Random Forest feature selection based Linear Regression-------
-')
df_rf = pd.read_csv('rf_regression.csv')
rf_cols = df_rf.columns
X_train_rf, X_test_rf, y_train_rf, y_test_rf =
train_test_split(df_rf.drop('Outstanding_Debt',axis=1),

df_rf['Outstanding_Debt'], random_state=5805)
X_train_rf = sm.add_constant(X_train_rf)
model_rf_ols = sm.OLS(y_train, X_train).fit()
y_pred_rf = model_rf_ols.predict(X_test)
y_pred_rf_org = reverse_standardize_data(y_pred_rf, y_org)
y_train_rf_org = reverse_standardize_data(y_train_rf, y_org)
# MSE
mse_rf = metrics.mean_squared_error(y_test, y_pred_rf)
# mae_rf = metrics.mean_absolute_error(y_test, y_pred_rf)
result_table.add_row(['Multiple Linear Regression(RForest)',
round(model_rf_ols.rsquared, 3), round(model_rf_ols.aic,3),
                      round(model_rf_ols.bic, 3),
round(model_rf_ols.rsquared_adj, 3), round(mse_rf, 3)])

# ------------------------------------------------------
#       Evaluation of Linear Regression Model
# ------------------------------------------------------
print('Evaluation of Linear Regression Model')
print(result_table)
# plot result df = pd.DataFrame({'Original test set': y test org, 'Predicted
Debt': y_pred_org,
#                                'Training Debt': y_train_org})
# x_range = [i for i in range(len(plot_result_df['Original test set']))]
train_range = [i for i in range(len(y_train_org))]
test_range = [i for i in range(y_test.shape[0])]
```

```
plt.plot(train_range, y_train_org, label='Training Debt')
plt.plot(test_range, y_test_org, label='Test Debt')
plt.plot(test_range, y_pred_org, label='Predicted Debt')
plt.xlabel('Observations')
plt.ylabel('Debt')
plt.legend(loc='best')
plt.title('Linear Regression (OLS Analysis)')
plt.show()


# ------------------------------------------------------
#               Prediction Intervals
# ------------------------------------------------------
# X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.2,
random_state=5805,shuffle=True)
# X_test = X_test[OLS_features]
sm_pred = model.get_prediction(sm.add_constant(X_test)) \
          .summary_frame(alpha=0.05)
print('--------------------Prediction Intervals--------------------')
print(sm_pred.head())
# obs_ci_lower and obs_ci_upper are prediction intervals
lower_interval = sm_pred['obs_ci_lower']
upper_interval = sm_pred['obs_ci_upper']
y_test_original = reverse_standardize_data(sm_pred['mean'], y_org)
lower_interval_original = reverse_standardize_data(lower_interval, y_org)
upper_interval_original = reverse_standardize_data(upper_interval, y_org)
#print('Upper Interval:',upper_interval_original.head())
x_range = [i for i in range(len(y_test_original))]
plt.plot(x_range, lower_interval_original, alpha=0.4, label='Lower interval')
plt.plot(x_range, upper_interval_original, alpha=0.4, label='Upper interval')
plt.plot(x_range, y_test_original, alpha=1.0, label='Predicted Debt')
plt.title('Predicted Debt with Intervals')
plt.ylabel('Debt')
plt.xlabel('Observations')
plt.legend()
plt.show()
```

Classification.py - Phase 3 Classification Analysis code

```
# ------------------------------------------------------------
#               Phase 3: Classification Analysis
# ------------------------------------------------------------
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelBinarizer
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, StackingClassifier,
BaggingClassifier, GradientBoostingClassifier, AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn import metrics
from sklearn.metrics import RocCurveDisplay
from sklearn.model_selection import StratifiedKFold
import time
from itertools import cycle, combinations
from prettytable import PrettyTable
from imblearn.under_sampling import RandomUnderSampler
import warnings
warnings.filterwarnings(action='ignore')


result_table = PrettyTable(field_names=['Classifier', 'Confusion_Matrix',
'Macro-Precision', 'Macro-Recall', 'Macro-F1', 'Macro-Specificity'
                                        , 'Macro ROC-AUC(OvR)', 'Macro ROC-
AUC(OvO)'])
result_table.title = 'Evaluation of various classifiers'


def evaluate_model(model, y_true, y_pred, y_proba, model_name):
    """
    :param model:  Classifier to evaluate
    :param y_true: target class labels
    :param y_pred: predicted class labels
    :param model_name: Classifier name in string (passed as string in method
call)
    """
    print('-------------- Evaluating for model {} --------------
'.format(str(model_name)))
    # 1. Confusion Matrix
    cnf_matrix = metrics.confusion_matrix(y_true, y_pred)
    print('Confusion Matrix for {} is:'.format(model_name))
    print(cnf_matrix)
    sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu", fmt='g',
                xticklabels=np.unique(y_test),
                yticklabels=np.unique(y_test))
    plt.title('{} Confusion matrix'.format(model_name), y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
    plt.show()
    # 2. Precision (Macro and micro)
    mac_precision = metrics.precision_score(y_true, y_pred, average='macro')
    mic_precision = metrics.precision_score(y_true, y_pred, average='micro')
    print('Macro Precision for {} is: {}'.format(str(model),
round(mac_precision, 3)))
```

```
    print('Micro Precision for {} is: {}'.format(str(model),
round(mic_precision, 3)))
    print()
    # 3. Recall (Macro & Micro)
    mac_recall = metrics.recall_score(y_true, y_pred, average='macro')
    mic_recall = metrics.recall_score(y_true, y_pred, average='micro')
    print('Macro Recall for {} is: {}'.format(str(model), round(mac_recall,
3)))
    print('Micro Recall for {} is: {}'.format(str(model), round(mic_recall,
3)))
    print()
    # 4. F1 Score (Macro & Micro)
    mac_f1 = metrics.recall_score(y_true, y_pred, average='macro')
    mic_f1 = metrics.recall_score(y_true, y_pred, average='micro')
    print('Macro F1 for {} is: {}'.format(str(model), round(mac_f1, 3)))
    print('Micro F1 for {} is: {}'.format(str(model), round(mic_f1, 3)))
    print()

    # 5. Specificity (TN / TN + FP)
    FP = cnf_matrix.sum(axis=0) - np.diag(cnf_matrix)
    FN = cnf_matrix.sum(axis=1) - np.diag(cnf_matrix)
    TP = np.diag(cnf_matrix)
    TN = cnf_matrix.sum() - (FP + FN + TP)
    FP = FP.astype(float)
    FN = FN.astype(float)
    TP = TP.astype(float)
    TN = TN.astype(float)
    # labels=[0,1,2]
    # for i in labels:
    # Specificity
    #     label_specificities = [specificity(y_true[:, i].ravel(), y_pred[:,
i].ravel()) for i in range(y_true.shape[i])]
    #     # Calculate micro specificity
    #     micro_specificity = np.sum(label_specificities) /
len(label_specificities)
    #     # Calculate macro specificity
    #     macro_specificity = np.mean(label_specificities)
    #     print('Micro Specificity for model {} is: {}'.format(str(model),
micro_specificity))
    #     print('Macro Specificity for model {} is: {}'.format(str(model),
macro_specificity))
    # print(f"Macro Specificity: {macro_specificity}")
    specificity = TN / (TN + FP)
    mac_specificity = np.mean(specificity)
    print('Macro-specificity for {} is: {}'.format(model_name,
round(mac_specificity,3)))
    # print('Specificity for class 0:', round(specificity[0], 3))
    # print('Specificity for class 1:', round(specificity[1], 3))
    # print('Specificity for Class 2:', round(specificity[2], 3))

    # 6 Macro and Micro ROC_AUC score (OvR)
    macro_auc_ovr = metrics.roc_auc_score(y_true, y_proba,
labels=model.classes_, multi_class='ovr', average='macro')
    micro_auc_ovr = metrics.roc_auc_score(y_true, y_proba,
labels=model.classes_, multi_class='ovr', average='micro')
```

```
    print('Macro-averaged One v/s Rest ROC AUC score:
{}'.format(round(macro_auc_ovr, 3)))
    print('Micro-averaged One v/s Rest ROC AUC score:
{}'.format(round(micro_auc_ovr, 3)))

    # # 7. Macro and Micro ROC_AUC score (OvO)
    macro_auc_ovo = metrics.roc_auc_score(y_true, y_proba,
labels=model.classes_, multi_class='ovo', average='macro')
    # micro_auc_ovo = metrics.roc_auc_score(y_true, y_proba,
labels=model.classes_, multi_class='ovo', average='micro')
    print('Macro AUC (OvO) for model {} is: {}'.format(model_name,
round(macro_auc_ovo, 3)))
    # print('Micro AUC for(OvO) model {} is: {}'.format(str(model),
round(micro_auc_ovo, 3)))

    # Append result to table
    # field_names=['Classifier', 'Confusion_Matrix', 'Macro-Precision',
'Macro-Recall', 'Macro-F1', 'Macro-Specificity'
    #                                         , 'Macro ROC-AUC(OvR)', 'Macro
ROC-AUC(OvO)'])
    result_table.add_row([model_name, cnf_matrix, round(mac_precision,3),
round(mac_recall,3), round(mac_f1,3),
                          round(mac_specificity,3), round(macro_auc_ovr,3),
                          round(macro_auc_ovo,3)])


def plot_roc_ovr(y_train, y_test, y_proba, model_name):
    # We need macro and micro averages of roc_auc, along with class-wise
roc_auc
    label_binarizer = LabelBinarizer().fit(y_train)
    y_onehot_test = label_binarizer.transform(y_test)

    # Compute micro AUC-ROC
    fpr, tpr, roc_auc = {}, {}, {}
    fpr['micro'], tpr['micro'], _ = metrics.roc_curve(y_onehot_test.ravel(),
y_proba.ravel())
    roc_auc["micro"] = metrics.auc(fpr["micro"], tpr["micro"])

    # Compute macro AUC-ROC
    n_classes = len(np.unique(y_train))
    for i in range(n_classes):
        fpr[i], tpr[i], _ = metrics.roc_curve(y_onehot_test[:, i], y_proba[:,
i])
        roc_auc[i] = metrics.auc(fpr[i], tpr[i])
    fpr_grid = np.linspace(0.0,1.0,1000)
    # Interpolate all ROC curves at these points
    mean_tpr = np.zeros_like(fpr_grid)

    for i in range(n_classes):
        mean_tpr += np.interp(fpr_grid, fpr[i], tpr[i])   # Linear
Interpolation

    # Take average and compute AUC
    mean_tpr /= n_classes
    fpr['macro'] = fpr_grid
```

```
    tpr['macro'] = mean_tpr
    roc_auc['macro'] = metrics.auc(fpr['macro'], tpr['macro'])

    # Plot all OvR ROC curves together
    fig, ax = plt.subplots(figsize=(6, 6))

    plt.plot(
        fpr['micro'], tpr['micro'],
        label=f"Micro average ROC curve (AUC = {roc_auc['micro']:.2f})",
        color='deeppink',
        linestyle=':',
        linewidth=4)
    plt.plot(
        fpr['macro'], tpr['macro'],
        label=f"Macro average ROC curve (AUC = {roc_auc['macro']:.2f})",
        color='navy',
        linestyle=':',
        linewidth=4)

    colors = cycle(['aqua', 'darkorange', 'cornflowerblue'])
    for class_id, color in zip(range(n_classes), colors):
        RocCurveDisplay.from_predictions(
            y_onehot_test[:, class_id],
            y_proba[:, class_id],
            name='ROC Curve for Class {}'.format(class_id),
            color=color,
            ax=ax,
            plot_chance_level=(class_id==2),
        )
    plt.axis('square')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Extension of ROC for {} to \nOne-vs-Rest
multiclass'.format(model_name))
    plt.legend()
    plt.show()


def plot_roc_ovo(y_train, y_test, y_proba, model_name):
    label_binarizer = LabelBinarizer().fit(y_train)
    pair_list = list(combinations(np.unique(y_train), 2))
    pair_scores = []
    mean_tpr = dict()
    fpr_grid = np.linspace(0.0, 1.0, 1000)
    for ix, (label_a, label_b) in enumerate(pair_list):
        a_mask = y_test == label_a
        b_mask = y_test == label_b
        ab_mask = np.logical_or(a_mask, b_mask)

        a_true = a_mask[ab_mask]
        b_true = b_mask[ab_mask]

        idx_a = np.flatnonzero(label_binarizer.classes_ == label_a)[0]
        idx_b = np.flatnonzero(label_binarizer.classes_ == label_b)[0]
```

```
        fpr_a, tpr_a, _ = metrics.roc_curve(a_true, y_proba[ab_mask, idx_a])
        fpr_b, tpr_b, _ = metrics.roc_curve(b_true, y_proba[ab_mask, idx_b])

        mean_tpr[ix] = np.zeros_like(fpr_grid)
        mean_tpr[ix] += np.interp(fpr_grid, fpr_a, tpr_a)
        mean_tpr[ix] += np.interp(fpr_grid, fpr_b, tpr_b)
        mean_tpr[ix] /= 2
        mean_score = metrics.auc(fpr_grid, mean_tpr[ix])
        pair_scores.append(mean_score)

    ovo_tpr = np.zeros_like(fpr_grid)
    macro_roc_auc_ovo = metrics.roc_auc_score(
        y_test,
        y_proba,
        multi_class="ovo",
        average="macro",
    )
    fig, ax = plt.subplots(figsize=(6, 6))
    for ix, (label_a, label_b) in enumerate(pair_list):
        ovo_tpr += mean_tpr[ix]
        plt.plot(
            fpr_grid,
            mean_tpr[ix],
            label=f"Mean {label_a} vs {label_b} (AUC =
{pair_scores[ix]:.3f})",
        )

    ovo_tpr /= sum(1 for pair in enumerate(pair_list))

    plt.plot(
        fpr_grid,
        ovo_tpr,
        label=f"One-vs-One macro-average (AUC = {macro_roc_auc_ovo:.3f})",
        linestyle=":",
        linewidth=4,
    )
    plt.plot([0, 1], [0, 1], "k--", label="Chance level (AUC = 0.5)")
    plt.axis("square")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(f"Extension of ROC for {model_name} to \nOne-vs-One
multiclass")
    plt.legend()
    plt.show()


def stratified_kfold_cv(X, y, model, model_name):
    print(f'---------- Stratified K-Fold CV for {model_name} ----------')
    skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=5805)
    prec, rec, f1 = [], [], []
    for train_index, test_index in skf.split(X, y):
        X_train, X_test, y_train, y_test = X.iloc[train_index],
X.iloc[test_index], y.iloc[train_index], y.iloc[test_index]
        if 'KNN' in model_name:
            # Fit the model
```

```
            model.fit(X_train.values, y_train.values)
            # Predict the credit score
            y_pred = model.predict(X_test.values)
            y_pred_proba = model.predict_proba(X_test.values)
        else:
            # Fit the model
            model.fit(X_train, y_train)
            # Predict the credit score
            y_pred = model.predict(X_test)
            y_pred_proba = model.predict_proba(X_test)
        # Evaluate model
        prec += [metrics.precision_score(y_test, y_pred, average='macro')]
        rec += [metrics.recall_score(y_test, y_pred, average='macro')]
        f1 += [metrics.f1_score(y_test, y_pred, average='macro')]
        # auc += [metrics.roc_auc_score(y_test, y_pred_proba,
multi_class='ovr',average='macro')]
    print('------------------Stratified K-Fold CV Results------------------')
    print('Macro averaged Precision Score for {} = {}'.format(model_name,
round(np.mean(prec), 3)))
    print('Macro averaged Recall Score for {} = {}'.format(model_name,
round(np.mean(rec), 3)))
    print('Macro averaged F1 Score for {} = {}'.format(model_name,
round(np.mean(f1), 3)))
    # print('Macro averaged ROC-AUC score for {} = {}'.format(model_name))


if __name__ == "__main__":
    start = time.time()
    # ------------------------------
    #       Read Cleaned Data
    # ------------------------------
    df = pd.read_csv('credit_classification_cleaned.csv')
    print(df.info())
    print(df.head())
    print(df.columns)
    print('Shape:', df.shape)
    # Drop additional columns
    # df.drop(['Unnamed: 0'], axis=1, inplace=True)

    # Split into X & y
    X = df.drop('Credit_Score', axis=1)
    y = df['Credit_Score']
    # Downsample the data
    down_sample = RandomUnderSampler(sampling_strategy={0: 4000, 1: 4000, 2:
4000}, random_state=5805)
    X, y = down_sample.fit_resample(X, y)
    # ---------------------------------
    #       Train Test Split
    # ---------------------------------
    X train, X test, y train, y test = train test split(X, y, test size=0.2,
random_state=5805)

    #
================================================================================
    #                   ONE V/S ALL OR ONE V/S REST CLASSIFICATION
```

```
    #
================================================================================
    # ------------------------------------------------------------
    #           Classifier 1: Decision Tree
    # ------------------------------------------------------------
    print('-----------------Simple Decision Tree------------------')
    dt_model = DecisionTreeClassifier(random_state=5805)
    dt_model.fit(X_train, y_train)
    # # #
    # # # # Predict credit_score
    y_pred_train = dt_model.predict(X_train)
    y_pred_dt = dt_model.predict(X_test)
    y_pred_dt_proba = dt_model.predict_proba(X_test)
    # # # # #
    # # # # # Show metrics like accuracy of test and train set to check
overfitting
    train_acc = metrics.accuracy_score(y_train, y_pred_train)
    test_acc = metrics.accuracy_score(y_test, y_pred_dt)
    print('Accuracy of train:', round(train_acc, ))
    print('Accuracy of test:', round(test_acc, 2))
    evaluate_model(dt_model, y_test, y_pred_dt, y_pred_dt_proba,
model_name='Decision Tree')
    plot_roc_ovr(y_train, y_test,y_pred_dt_proba, model_name='Decision Tree')
    plot_roc_ovo(y_train, y_test, y_pred_dt_proba, model_name='Decision
Tree')
    stratified_kfold_cv(X, y, dt_model, 'Decision Tree')
    # # # #
    # # # # # Grid Search CV for parameters of decision tree classifier
    # # start_time = time.time()
    # parameters = {'max_depth': [i for i in range(5, 16, 5)],
    #               'min_samples_split': [i for i in range(5, 16, 5)],
    #               'min_samples_leaf': [i for i in range(5, 16, 5)],
    #               'max_features': [5, 10, 15, 'sqrt','log2'],
    #               'splitter': ['best'],
    #               'criterion': ['gini', 'entropy']}
    # grid_model = GridSearchCV(estimator=dt_model, param_grid=parameters,
scoring='accuracy',n_jobs=-1)
    # grid_model.fit(X_train, y_train)
    print('After Grid Search best parameters are:')
    # print(grid_model.best_params_)
    # best_parameters = grid_model.best_params_
    best_parameters = {'criterion': 'gini', 'max_depth': 15, 'max_features':
'sqrt', 'min_samples_leaf': 5, 'min_samples_split': 15, 'splitter': 'best'}
    print('Best Parameters Decision Tree:', best_parameters)
    # # # #     # grid_model.best_params_)
    # end_time = time.time()
    # # time_spent = end_time - start_time
    # # print('Time (mins) for Grid Search:', round(time_spent/60, 2))
    print('-----------------Decision Tree Pre-Pruned------------------')
    dt_model_pre = DecisionTreeClassifier(**best_parameters,
random_state=5805)
    dt_model_pre.fit(X_train, y_train)
    y_pred_dt = dt_model_pre.predict(X_test)
    y_pred_proba_dt = dt_model_pre.predict_proba(X_test)
    # # # # # Evaluate Model
```

```
    evaluate_model(dt_model_pre, y_test, y_pred_dt, y_pred_proba_dt,
model_name='Pre-pruned Decision Tree')
    plot_roc_ovr(y_train, y_test, y_pred_proba_dt, model_name='Pre-pruned
Decision Tree')
    plot_roc_ovo(y_train, y_test, y_pred_proba_dt, model_name='Pre-pruned
Decision Tree')
    stratified_kfold_cv(X, y, dt_model_pre, 'Pre-pruned Decision Tree')
    #
    # # # # ------------------------------------------------------------
    # # # #           Post-Pruning (Optimum alpha)
    # # # # ------------------------------------------------------------
    path = dt_model.cost_complexity_pruning_path(X_train, y_train)
    alphas = path['ccp_alphas']
    # print(alphas)
    print('No.of alphas:',len(alphas))
    # print(alphas)
    # # ============================
    # # Grid search for best alpha
    # # ============================
    accuracy_train, accuracy_test = [], []
    for i in range(0, len(alphas), 200):
        dt_model = DecisionTreeClassifier(ccp_alpha=alphas[i],
random_state=5805)
        dt_model.fit(X_train, y_train)
        y_train_pred = dt_model.predict(X_train)
        y_test_pred = dt_model.predict(X_test)
        accuracy_train.append(metrics.accuracy_score(y_train, y_train_pred))
        accuracy_test.append(metrics.accuracy_score(y_test, y_test_pred))
    alpha_range = [alphas[i] for i in range(0,len(alphas),200)]
    fig, ax = plt.subplots()
    ax.set_xlabel('alpha')
    ax.set_ylabel('accuracy')
    ax.set_title("Accuracy vs alpha for training and testing sets")
    ax.plot(alpha_range, accuracy_train, marker="o", label="train",
            drawstyle="steps-post")
    ax.plot(alpha_range, accuracy_test, marker="o", label="test",
            drawstyle="steps-post")
    ax.legend()
    plt.grid()
    plt.tight_layout()
    plt.show()
    if X.shape[1] < 50000:
        opt_alpha = 0.00021
    else:
        opt_alpha = 0.0000325

    dt_model = DecisionTreeClassifier(ccp_alpha=opt_alpha, random_state=5805)
    dt_model.fit(X_train, y_train)
    # # # # Predict
    y_pred_dt = dt_model.predict(X_test)
    y_pred_proba_dt = dt_model.predict_proba(X_test)
    evaluate_model(dt_model, y_test, y_pred_dt, y_pred_proba_dt,
model_name='Post-Pruned Decision Tree')
    plot_roc_ovr(y_train, y_test, y_pred_proba_dt, model_name='Post-pruned
Decision Tree')
```

```
    plot_roc_ovo(y_train, y_test, y_pred_proba_dt, model_name='Post-pruned
Decision Tree')
    stratified_kfold_cv(X, y, dt_model, 'Post-pruned Decision Tree')
    # # # Experiments: 100-200, 200-300, 900-1000,1500-1600, 3000-3100,5000-
5200
    # #
    # # # # ------------------------------------------------------------------
--------
    # # # #           Classifier 2: Logistic Regression
    # # # # ------------------------------------------------------------------
--------
    print('-----------------Logistic Regression------------------')
    multi_logistic = LogisticRegression(C=0.5, multi_class='ovr',
random_state=5805)  # OvR Classifier
    multi_logistic.fit(X_train, y_train)
    # #
    # # # Make Prediction
    y_pred_log = multi_logistic.predict(X_test)
    y_pred_proba_log = multi_logistic.predict_proba(X_test)
    # # # # # Evaluate OvR
    evaluate_model(multi_logistic, y_test, y_pred_log, y_pred_proba_log,
model_name='Logistic Regression')
    plot_roc_ovr(y_train, y_test, y_pred_proba_log, model_name='Logistic
Regression')
    plot_roc_ovo(y_train,y_test,y_pred_proba_log, model_name='Logistic
Regression')
    stratified_kfold_cv(X, y, multi_logistic, 'Logistic Regression')
    #
    # # # ------------Grid Search CV for Logistic Regression----------------
    parameters = {'C': [1.0, 2.0],
                  'solver': ['lbfgs', 'liblinear']}
    grid_log = GridSearchCV(estimator=multi_logistic, param_grid=parameters,
scoring='f1_macro')
    grid_log.fit(X_train, y_train)
    # best_parameters = grid_log.best_params_
    best_parameters = {'C': 2.0, 'solver': 'lbfgs'}
    print('Best Parameters for Logistic Regression:', best_parameters)
    grid_log = LogisticRegression(**best_parameters,random_state=5805)
    grid_log.fit(X_train, y_train)
    # #
    y_pred_log = grid_log.predict(X_test)
    y_pred_proba_log = grid_log.predict_proba(X_test)
    # # Evaluate the model
    evaluate_model(grid_log, y_test, y_pred_log, y_pred_proba_log,
model_name='Grid Logistic Regression')
    plot_roc_ovr(y_train, y_test, y_pred_proba_log, model_name='Grid Logistic
Regression')
    plot_roc_ovo(y_train, y_test, y_pred_proba_log, model_name='Grid Logistic
Regression')
    stratified_kfold_cv(X, y, grid_log, 'Grid Logistic Regression')
    #
    #
    # #
    # # # ------------------------------------------------------------------
------
```

```
    # # #           Classifier 3: K-Nearest Neighbors(KNN)
    # # # ------------------------------------------------------------------
------
    print('-----------------K-Nearest Neighbors(KNN)------------------')
    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train.values, y_train.values)
    y_pred_knn = knn.predict(X_test.values)
    y_pred_knn_proba = knn.predict_proba(X_test.values)
    print('--------------Results before Optimal K--------------')
    evaluate_model(knn, y_test, y_pred_knn, y_pred_knn_proba,
model_name='KNN')
    plot_roc_ovr(y_train, y_test, y_pred_knn_proba, model_name='KNN')
    plot_roc_ovo(y_train, y_test, y_pred_knn_proba, model_name='KNN')
    stratified_kfold_cv(X, y, knn, 'KNN')
    #
    #
    ## Select Best K
    print('----------Grid Search for KNN Classifier----------')
    k_options = [i for i in range(1, 12)]
    error_rate = []
    for k in k_options:
        knn = KNeighborsClassifier(n_neighbors=k)
        knn.fit(X_train.values,y_train.values)
        y_pred = knn.predict(X_test.values)
        # Get average error per record for each knn model
        error_rate.append(np.mean(y_test.values != y_pred))
    # ## Plot Error Rate vs K
    plt.figure(figsize=(10, 6))
    # k_range = [i for i in range(1,21,1)]
    plt.plot(k_options, error_rate, color='blue', linestyle='dashed',
marker='o', markerfacecolor='red',markersize=12)
    plt.title('Error Rate vs. K Value')
    plt.xlabel('K')
    plt.ylabel('Error Rate')
    plt.xticks(ticks=k_options)
    plt.show()
    knn = KNeighborsClassifier()
    from sklearn.model_selection import GridSearchCV

    # k_range = list(range(3, 11))
    # param_grid = dict(n_neighbors=k_range)
    # grid = GridSearchCV(knn, param_grid, cv=2, scoring='f1_macro',
    #                     return_train_score=True, verbose=1)
    # # fitting the model for grid search
    # grid_search = grid.fit(X_train.values, y_train.values)
    # print(f'The best k after grid search is : {grid_search.best_params_}')
    # #
    print('------------------Results after selecting Optimal K---------------
---')
    optimal_k = 9  # Used Elbow Method to select K
    print('Optimal K:', optimal_k)
    knn = KNeighborsClassifier(n_neighbors=optimal_k)
    knn.fit(X_train.values, y_train.values)
    y_pred_knn = knn.predict(X_test.values)
    y_pred_knn_proba = knn.predict_proba(X_test.values)
```

```
    # # # # print(y_pred_knn_proba)
    evaluate_model(knn, y_test, y_pred_knn, y_pred_knn_proba,
model_name='KNN(Optimal K)')
    plot_roc_ovr(y_train, y_test, y_pred_knn_proba, model_name='KNN(Optimal
K)')
    plot_roc_ovo(y_train, y_test, y_pred_knn_proba, model_name='KNN(Optimal
K)')
    stratified_kfold_cv(X, y, knn, 'KNN(Optimal K) Classifier')
    # #
    # # # # ------------------------------------------------------------------
--------
    # # # #           Classifier 4: Support Vector Machine(SVM)
    # # # # ------------------------------------------------------------------
--------
    print('-----------------Support Vector Machine(SVM)------------------')
    svc_linear = SVC(C=0.1, kernel='linear', random_state=5805,
probability=True)
    svc_poly = SVC(C=0.1, kernel='poly', random_state=5805, probability=True)
    svc_rbf = SVC(C=0.1, kernel='rbf', random_state=5805,probability= True)
    # # # #
    # # # # # Fit the models
    print('-----------------Linear SVM Classifier(Fit in progress)----------
-------')
    svc_linear.fit(X_train, y_train)
    print('-----------------Poly SVM Classifier(Fit in progress)------------
-----')
    svc_poly.fit(X_train, y_train)
    print('-----------------RBF SVM Classifier(Fit in progress)--------------
----')
    svc_rbf.fit(X_train, y_train)
    # # # #
    # # # # # Predicting the target
    y_pred_svc_linear = svc_linear.predict(X_test)
    y_pred_svc_poly = svc_poly.predict(X_test)
    y_pred_svc_rbf = svc_rbf.predict(X_test)
    # # # #
    # # # # # Predicting the probability
    y_proba_linear = svc_linear.predict_proba(X_test)
    y_proba_poly = svc_poly.predict_proba(X_test)
    y_proba_rbf = svc_rbf.predict_proba(X_test)
    # # # # #
    # # # # # ------Evaluate the SVC models------
    evaluate_model(svc_linear, y_test, y_pred_svc_linear, y_proba_linear,
model_name='SVC_Linear')
    plot_roc_ovr(y_train, y_test, y_proba_linear, model_name='SVC_Linear')
    plot_roc_ovo(y_train, y_test, y_proba_linear, model_name='SVC_Linear')
    stratified_kfold_cv(X, y, svc_linear, 'SVC_Linear')
    # # #
    evaluate_model(svc_poly, y_test, y_pred_svc_poly, y_proba_poly,
model_name='SVC_Poly')
    plot_roc_ovr(y_train, y_test, y_proba_poly, model_name='SVC_Poly')
    plot_roc_ovo(y_train, y_test, y_proba_poly, model_name='SVC_Poly')
    stratified_kfold_cv(X, y, svc_poly, 'SVC_Poly')
    # # #
    evaluate_model(svc_rbf, y_test, y_pred_svc_rbf, y_proba_rbf,
```

```
model_name='SVC_RBF')
    plot_roc_ovr(y_train, y_test, y_proba_rbf, model_name='SVC_RBF')
    plot_roc_ovo(y_train, y_test, y_proba_rbf, model_name='SVC_RBF')
    stratified_kfold_cv(X, y, svc_rbf, 'SVC_RBF')
    # #
    # # # # ------------Grid Search CV for SVM----------------
    print('------------Grid Search CV for SVM (One v/s Rest)----------------
')
    svc = SVC(random_state=5805, probability=True)
    # svc_params = {'C': [0.1, 0.2],
    #               'kernel': ['poly', 'rbf']}
    # grid_svm = GridSearchCV(estimator=svc, param_grid=svc_params,
scoring='f1_macro', n_jobs=-1)
    # grid_svm.fit(X_train, y_train)
    print('After Grid Search best parameters are:')
    # best_params_svc = grid_svm.best_params_
    best_params_svc = {'C': 0.2, 'kernel': 'rbf'}
    print(best_params_svc)
    # # # best_params_svc = {'C': 0.2, 'kernel': 'rbf'}
    grid_svm = SVC(**best_params_svc, random_state=5805, probability=True)
    grid_svm.fit(X_train, y_train)
    y_best_svc_pred = grid_svm.predict(X_test)
    y_best_svc_proba = grid_svm.predict_proba(X_test)
    evaluate_model(grid_svm, y_test, y_best_svc_pred, y_best_svc_proba,
model_name='Grid Search SVM')
    plot_roc_ovr(y_train, y_test, y_best_svc_proba, model_name='Grid Search
SVM')
    plot_roc_ovo(y_train, y_test, y_best_svc_proba, model_name='Grid Search
SVM')
    stratified_kfold_cv(X, y, grid_svm, 'Grid Search SVM ')
    # #
    # # # # ------------------------------------------------------------------
--------
    # # # #           Classifier 5: Naive Bayes Classifier
    # # # # ------------------------------------------------------------------
--------
    gnb = GaussianNB()
    # # # Fit the model
    gnb.fit(X_train, y_train)
    y_pred_gnb = gnb.predict(X_test)
    y_pred_gnb_proba = gnb.predict_proba(X_test)
    # # #
    # # # # Evaluate the models
    evaluate_model(gnb, y_test, y_pred_gnb, y_pred_gnb_proba, 'GaussianNB')
    plot_roc_ovr(y_train, y_test, y_pred_gnb_proba, model_name='GaussianNB')
    plot_roc_ovo(y_train, y_test, y_pred_gnb_proba, model_name='GaussianNB')
    stratified_kfold_cv(X, y, gnb, 'GaussianNB')
    params_NB = {'var_smoothing': np.logspace(0, -9, num=100)}
    gs_NB = GridSearchCV(estimator=gnb,
                         param_grid=params_NB,
                         scoring='f1_macro')
    gs_NB.fit(X_train, y_train)
    best_params_NB = gs_NB.best_params_
    print('Best Parameters of Grid Search NB Classifier are:',best_params_NB)
    gs_NB = GaussianNB(**best_params_NB)
```

```
    gs_NB.fit(X_train, y_train)
    y_pred_gnb = gs_NB.predict(X_test)
    y_pred_gnb_proba = gs_NB.predict_proba(X_test)
    evaluate_model(gs_NB, y_test, y_pred_gnb, y_pred_gnb_proba, 'Grid Search
GaussianNB')
    plot_roc_ovr(y_train, y_test, y_pred_gnb_proba, model_name='Grid Search
GaussianNB')
    plot_roc_ovo(y_train, y_test, y_pred_gnb_proba, model_name='Grid Search
GaussianNB')
    stratified_kfold_cv(X, y, gs_NB, 'Grid Search GaussianNB')
    #
    # # #
    # # # # ----------------------------------------------------------------
--------
    # # # #           Classifier 6: Random Forest Classifier
    # # # # # ----------------------------------------------------------------
----------
    rfc_model = RandomForestClassifier(n_estimators=50, random_state=5805)
    # # # Fit the model
    rfc_model.fit(X_train, y_train)
    # # # Predict credit_score
    y_pred_rfc = rfc_model.predict(X_test)
    y_pred_rfc_proba = rfc_model.predict_proba(X_test)
    # # #
    # # # # Evaluate Models
    evaluate_model(rfc_model, y_test, y_pred_rfc, y_pred_rfc_proba,
model_name='Random Forest')
    plot_roc_ovr(y_train, y_test, y_pred_rfc_proba, model_name='Random
Forest')
    plot_roc_ovo(y_train, y_test, y_pred_rfc_proba, model_name='Random
Forest')
    stratified_kfold_cv(X, y, rfc_model, 'Random Forest Classifier')
    # # #
    # # # Grid Search for Random Forest Classifier
    print('------------------------Grid Search Random Forest Classifier------
-------------------')
    # parameter_grid = {'n_estimators': [100, 125],
    #                   'criterion': ['gini', 'entropy']}
    # grid_rfc = GridSearchCV(estimator=rfc_model, param_grid=parameter_grid,
scoring='f1_macro',n_jobs=-1)
    # grid_rfc.fit(X_train, y_train)
    # best_params = grid_rfc.best_params_
    best_params = {'criterion': 'gini', 'n_estimators': 125}
    print('R-Forest Classifier best parameters:', best_params)
    grid_rfc = RandomForestClassifier(**best_params, random_state=5805)
    grid_rfc.fit(X_train, y_train)
    # # # # Predict Credit Score with grid_rfc
    y_pred_rfc = grid_rfc.predict(X_test)
    y_pred_rfc_proba = grid_rfc.predict_proba(X_test)
    # # # # Evaluate Model
    evaluate_model(grid_rfc, y_test, y_pred_rfc, y_pred_rfc_proba,
model_name='Grid Search RForest')
    plot_roc_ovr(y_train, y_test, y_pred_rfc_proba, model_name='Grid Search
RForest')
    plot_roc_ovo(y_train, y_test, y_pred_rfc_proba, model_name='Grid Search
```

```
RForest')
    stratified_kfold_cv(X, y, grid_rfc, 'Grid Search RForest')
    #
    # # ------------------------------------------------------------
    # #              Stacking Random Forest Classifier
    # # ------------------------------------------------------------
    print('--------------------Stacking with Random Forest Classifier--------
------------')
    # # # # We will use a random forest classifier with logistic regression
as final estimator
    base_estimators = [('rf', RandomForestClassifier(n_estimators=100,
random_state=5805)),
                       ('rf2', RandomForestClassifier(n_estimators=100,
random_state=5805))]
    stacked_model = StackingClassifier(estimators=base_estimators,
final_estimator=LogisticRegression(multi_class='ovr', random_state=5805),
                                       n_jobs=-1)
    # # #
    stacked_model.fit(X_train, y_train)
    y_pred_stacked = stacked_model.predict(X_test)
    y_pred_stacked_proba = stacked_model.predict_proba(X_test)
    # # # # Evaluate Model
    evaluate_model(stacked_model, y_test, y_pred_stacked,
y_pred_stacked_proba, model_name='Stacked Random Forest')
    plot_roc_ovr(y_train, y_test, y_pred_stacked_proba, model_name='Stacked
Random Forest')
    plot_roc_ovo(y_train, y_test, y_pred_stacked_proba, model_name='Stacked
Random Forest')
    stratified_kfold_cv(X, y, stacked_model, 'Stacked Random Forest')
    #
    # # ------------------------------------------------------------
    # #              Bagging Random Forest Classifier
    # # ------------------------------------------------------------
    print('--------------------Bagging with Random Forest Classifier---------
-----------')
    rfc = RandomForestClassifier(n_estimators=100, random_state=5805)
    bag_classifier = BaggingClassifier(estimator=rfc, n_estimators=10,
random_state=5805, n_jobs=-1)
    # # # # Fit the model
    bag_classifier.fit(X_train, y_train)
    # # #
    # # # # Predict the Credit_Score
    y_pred_bag = bag_classifier.predict(X_test)
    y_pred_bag_proba = bag_classifier.predict_proba(X_test)
    # # #
    # # # # Evaluate the model
    evaluate_model(bag_classifier, y_test, y_pred_bag, y_pred_bag_proba,
model_name='Bagging Random Forest')
    plot_roc_ovr(y_train, y_test, y_pred_bag_proba, model_name='Bagging
Random Forest')
    plot_roc_ovo(y_train, y_test, y_pred_bag_proba, model_name='Bagging
Random Forest')
    stratified_kfold_cv(X, y, bag_classifier, 'Bagging Random Forest')
    #
    # # ------------------------------------------------------------
```

```
    # #              Boosting Random Forest Classifier
    # # ------------------------------------------------------------
    print('--------------------Boosting with Random Forest Classifier--------
------------')
    rfc = RandomForestClassifier(n_estimators=125, random_state=5805)
    boost_classifier = AdaBoostClassifier(rfc, n_estimators=50,
random_state=5805)
    boost_classifier.fit(X_train, y_train)
    # # # Predict the Credit_Score
    y_pred_boost = boost_classifier.predict(X_test)
    y_pred_boost_proba = boost_classifier.predict_proba(X_test)
    # # #
    # # # # Evaluate the model
    evaluate_model(boost_classifier, y_test, y_pred_boost,
y_pred_boost_proba, model_name='Boosting Random Forest')
    plot_roc_ovr(y_train, y_test, y_pred_boost_proba, model_name='Boosting
Random Forest')
    plot_roc_ovo(y_train, y_test, y_pred_boost_proba, model_name='Boosting
Random Forest')
    stratified_kfold_cv(X, y, boost_classifier, 'Boosting Random Forest')

    # ------------------------------------------------------------
    #               Neural Network - MLP Classifier
    # ------------------------------------------------------------
    # 2 layers of 20 nodes each
    mlp_clf = MLPClassifier(hidden_layer_sizes=(20,)*2, activation='relu',
max_iter=1000, random_state=5805)
    # # Fit the model
    mlp_clf.fit(X_train, y_train)
    # # Predict the Credit Score
    y_pred_mlp = mlp_clf.predict(X_test)
    y_pred_mlp_proba = mlp_clf.predict_proba(X_test)
    # # Evaluate the models
    evaluate_model(mlp_clf, y_test, y_pred_mlp,
y_pred_mlp_proba,model_name='MLP Classifier')
    plot_roc_ovr(y_train, y_test, y_pred_mlp_proba, model_name='MLP
Classifier')
    plot_roc_ovo(y_train, y_test,y_pred_mlp_proba, model_name='MLP
Classifier')
    stratified_kfold_cv(X, y, mlp_clf, 'MLP Classifier')

    # Grid Search
    print('------------------------Grid Search MLP Classifier---------------
---------')
    # parameters = {'hidden_layer_sizes': [(50,)*2, (50,)*1],
    #               'activation': ['relu', 'tanh'],
    #               'max_iter': [1000, 1200]}
    # grid_mlp = GridSearchCV(param_grid=parameters,
estimator=MLPClassifier(random_state=5805), scoring='f1_macro',
    #                         n_jobs=-1)
    # grid_mlp.fit(X_train, y_train)
    # best_params = grid_mlp.best_params_
    best_params = {'activation': 'tanh', 'hidden_layer_sizes': (50, 50),
'max_iter': 1000}
    print('Best Parameters for MLP Classifier:', best_params)
```

```
    # # # for MLP Classifier: {'activation': 'tanh', 'hidden_layer_sizes':
(50, 50), 'max_iter': 500, 'solver': 'adam'}
    grid_mlp = MLPClassifier(**best_params, random_state=5805)
    grid_mlp.fit(X_train, y_train)
    y_pred_mlp = grid_mlp.predict(X_test)
    y_pred_mlp_proba = grid_mlp.predict_proba(X_test)
    # # Evaluate Grid Search MLP
    evaluate_model(grid_mlp, y_test, y_pred_mlp, y_pred_mlp_proba,
model_name='Grid Search MLP')
    plot_roc_ovr(y_train,y_test, y_pred_mlp_proba, model_name='Grid Search
MLP')
    plot_roc_ovo(y_train, y_test, y_pred_mlp_proba, model_name='Grid Search
MLP')
    stratified_kfold_cv(X, y, grid_mlp, 'Grid Search MLP')
    # ------------------------------------------------------------------------
-------------------------
    #                           Need to add K-fold cross validation (Code
Ready!!!!)
    # ------------------------------------------------------------------------
--------------------

    # Print Table
    print(result_table)
    end=time.time()
    time_spent = end-start
    print('Execution time:',round((time_spent/60),2))
```

Clustering.py - Phase 4 – code

```
# ------------------------------------------------------------------------
#               Phase 4: Clustering and Association Rule Mining
# ------------------------------------------------------------------------
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import collections
from mlxtend.frequent_patterns import apriori, association_rules
from imblearn.under_sampling import RandomUnderSampler
import warnings
warnings.filterwarnings(action='ignore')

# Read Dataset
df = pd.read_csv('credit_classification_cleaned.csv')
print(df.info())
print(df.head())
print(df.columns)
print('Shape:', df.shape)
X = df.drop('Credit_Score',axis=1)
y = df['Credit_Score']
```

```
# Downsample the data

down_sample = RandomUnderSampler(sampling_strategy={0: 4000, 1: 4000, 2:
4000}, random_state=5805)
X, y = down_sample.fit_resample(X, y)
# df = df.sample(18000, random_state=5805)
# X = df.drop('Credit_Score',axis=1)
# Prepare data X
X = X[['Outstanding_Debt','Delay_from_due_date','Changed_Credit_Limit',

'Credit_Utilization_Ratio','Num_of_Delayed_Payment','Payment_of_Min_Amount_Ye
s','Credit_Mix_Good']]

# -------------------Clustering Analysis--------------------

# Silhouette Scores
silhouette_scores = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=5805)
    cluster_labels = kmeans.fit_predict(X)
    silhouette_avg = silhouette_score(X, cluster_labels)
    silhouette_scores.append(silhouette_avg)

# Plotting the silhouette scores
plt.plot(range(2, 11), silhouette_scores, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Analysis for K-Means Clustering')
plt.show()

# Within cluster sum of squares (WCSS)
wcss = []

# Assume you want to try values of k from 2 to 10
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=5805)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plotting the within-cluster sum of squares
plt.plot(range(2, 11), wcss, marker='o')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Within-Cluster Sum of Squares')
plt.title('Within-Cluster Variation Plot for K-Means Clustering')
plt.show()


# # Association Rule Mining
# Prepare Data
# Which occupation is linked to Credit Mix of good
# Occupations - Scientist,
assoc_df = pd.read_csv('Association.csv')
print(assoc_df.columns)
one_hot_df=
pd.get_dummies(data=assoc_df[['Occupation','Credit_Mix','Payment_of_Min_Amoun
```

```
t']])
# one_hot_df.drop(['Occupation','Credit_Mix','Payment_of_Min_Amount'],axis=1,
inplace=True)
assoc_df = pd.concat([assoc_df,one_hot_df],axis=1)
assoc_df.drop(['Occupation','Credit_Mix','Payment_of_Min_Amount'],axis=1,
inplace=True)
print(assoc_df.columns)
print('---------- Results of Association Rule Mining ----------')
df = apriori(assoc_df,min_support=0.2, use_colnames=True, verbose=1)
print(df)
df_ar = association_rules(df,metric='confidence', min_threshold=0.7)
df_ar = df_ar.sort_values(['confidence','lift'], ascending=[False, False])
print(df_ar.to_string())


# X = assoc_df
#     # df)[['Outstanding_Debt', 'Interest_Rate', 'Delay_from_due_date',
'Credit_Utilization_Ratio',
#     #    'Changed_Credit_Limit']]
#         # 'Credit_Mix_Standard', 'Num_Credit_Inquiries',
'Credit_History_Age',
#         # 'Num_of_Delayed_Payment','Credit_Mix_Good', 'Num_Credit_Card',
'Total_EMI_per_month']]
```

References

[1] https://towardsdatascience.com/targeting-multicollinearity-with-python-3bd3b4088d0b

[2] https://analyticsindiamag.com/beginners-guide-to-truncated-svd-for-dimensionality-reduction/

[3] https://medium.com/@moussadoumbia_90919/elbow-method-in-supervised-learning-optimal-k-value-99d425f229e7

[4] https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

[5] https://scikit-learn.org/stable/modules/classes.html#module-sklearn.ensemble