MINI PROJECT

Virginia Polytechnic Institute and State University

CS 5804 Intro to Artificial Intelligence

Team:

Jyothi Sevakula
Sanjna Kumari
Muskaan Gupta
Sri Venkateshwara Swami Tumu

Table of contents

Table of figures

Abstract

Introduction

The project is based on diverse machine learning methodologies using a loan approval dataset to anticipate approvals of loans based on customer history. The process commences with meticulous preprocessing and feature engineering, ensuring the dataset's readiness for subsequent model construction. The process begins with thorough preprocessing and feature engineering to ensure that the dataset is prepared for subsequent model construction.

Various dimensionality reduction techniques, including PCA, RandomForest, SVD, and VIF, are implemented to streamline the feature space. The statistical significance of features is assessed through T-test and F-test analyses. Linear regression and backward stepwise regression are employed to predict the numerical feature 'Length.'

Moving to classification models, a range of techniques, such as decision tree, logistic regression, KNN, SVM, Naive Bayes, Random Forest, and neural networks, undergo grid search and stratified k-fold cross-validation. This process helps identify optimal hyperparameters, and model selection is based on a comprehensive evaluation of metrics.

Next, an unsupervised learning technique, KMeans clustering, is utilized to explore optimal clusters. Silhouette analysis and within-cluster sum of squares (WCSS) play crucial roles in determining the most suitable number of clusters.

The project concludes with the application of the Apriori algorithm, delving into insightful association rules among observations. This thorough analysis sheds light on intricate relationships within the dataset, providing valuable insights.

Description of Dataset

The Loan Prediction dataset consists of 251,999 instances and features 11 distinct attributes. The primary objective is to predict whether a given loan application will be approved or denied, utilizing information related to the scheduled application. The dataset includes the following features 'income', 'age', 'experience', 'married/single', 'house_ownership', 'car_ownership', 'profession', 'city', 'state', 'current_job_yrs', 'current_house_yrs', 'risk_flag'.

Each observation in the dataset is uniquely identified by an 'id' and encompasses details of an individual.The 'Income' attribute describes the income,'age' attribute describes the age,'experience' describes the experience of an individual,'married/single' attribute describes whether an individual is married or not, he is married, divorced or single, 'house_ownership describes whether a person has an own house or living in a rented basis. Car Ownership says weather an individual own a car or not, profession states the working class of an individual there are vast professions depicted in the dataset and city and state described where the person is from and current_house_yrs states since how many years he is living in the house and finally risk_flag out target variable which has 2 values 1 depicts the defaulters and 0 depicts that person had paid the loan.

In summary, the machine learning project on loan approval prediction offers tangible benefits in real-life scenarios by improving risk management, efficiency, objectivity, customer experience, resource allocation, and regulatory compliance in the lending process. It addresses

the evolving needs of financial institutions, fostering a more secure and customer-centric approach to lending practices.

There are totally 5 numerical and 6 categorical features: Income, Age, Experience, Current_job_yrs, current_house_yrs are numerical and Age, Experience, Married/single,House Age,Experience,Married/Single,House_Ownership,Profession,City,State are categorical features.which satisfies the criterion for this project having more than 50k observations and has at least two numerical features, at least two categorical features and there are at least two categories each of the categorical features.

Phase I: Feature Engineering and EDA

Data Preprocessing:

Data Cleaning:

There are no missing values in the dataset. There are also no nan values. The dataset is inherently clean .

Data Duplicates:

There are data duplicates in the dataset. I have performed df.drop_duplicates()

Dimensionality Reduction or Feature Selection

Random Forest Analysis

Conducting a Random Forest Analysis to assess feature importances is crucial for dimensionality reduction and pinpointing the most influential factors that affect the model's predictions. This process entails evaluating the relevance and contribution of each feature in predicting the target variable. By utilizing the 'feature_importances_' attribute inherent in the Random Forest classifier, the algorithm quantifies the significance of each feature in the predictive process. Analyzing these feature importances facilitates the identification of key variables that wield a substantial impact on the model's decision-making. This analysis is valuable for streamlining the feature space by prioritizing essential variables, potentially leading to dimensionality reduction and a more efficient model without compromising predictive performance. The identification and retention of the most impactful features play a pivotal role in optimizing model accuracy while minimizing computational complexity.

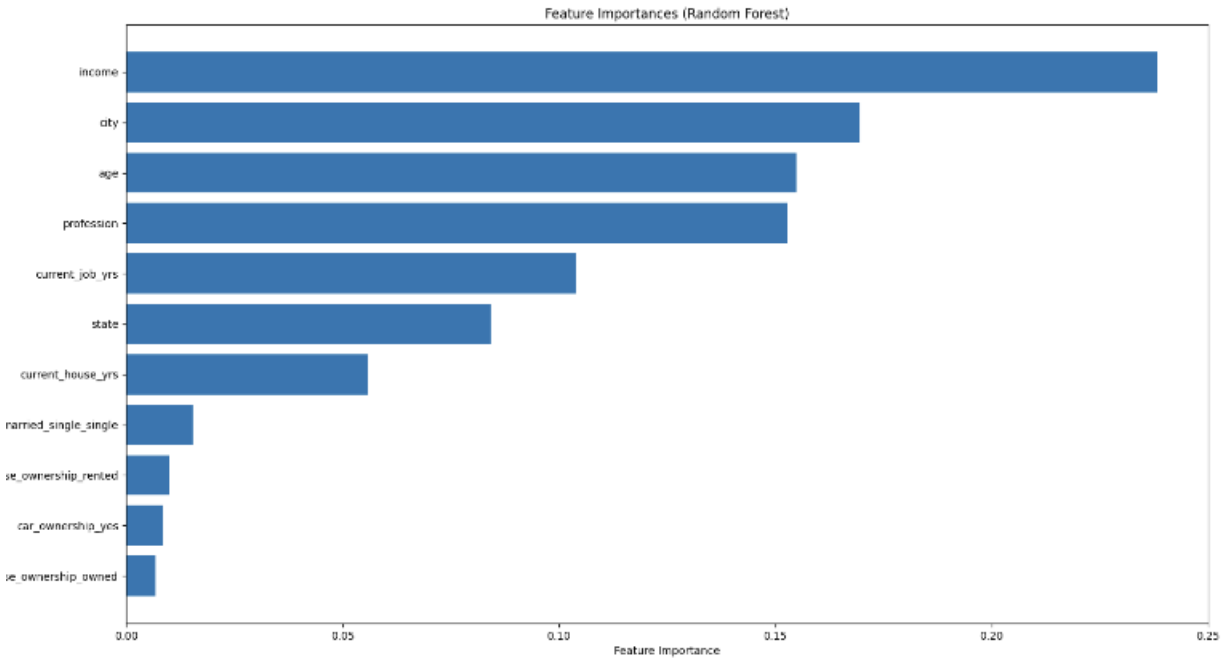

Fig 1

If we observe the feature importance graph there are four features which do not have a minimum importance of 0.05. I have taken the threshold as 0.05 and have dropped the features car_owned_yes ,married_single_single ,house_ownership_rented, house_ownership_owned

Principal Component Analysis (PCA)

Principal Component Analysis (PCA) stands out as a widely used unsupervised learning technique designed to reduce the dimensionality of expansive datasets. It strikes a balance by not only enhancing interpretability but also minimizing information loss. Positioned as a foundational method in data analysis, PCA excels in transforming high-dimensional data into a lower-dimensional form referred to as principal components. By capturing the most vital information from the original dataset, PCA effectively condenses the feature space while retaining crucial patterns and reducing redundancy. This dimensionality reduction not only facilitates easier visualization but also improves computational efficiency, often leading to enhanced performance in machine learning models.

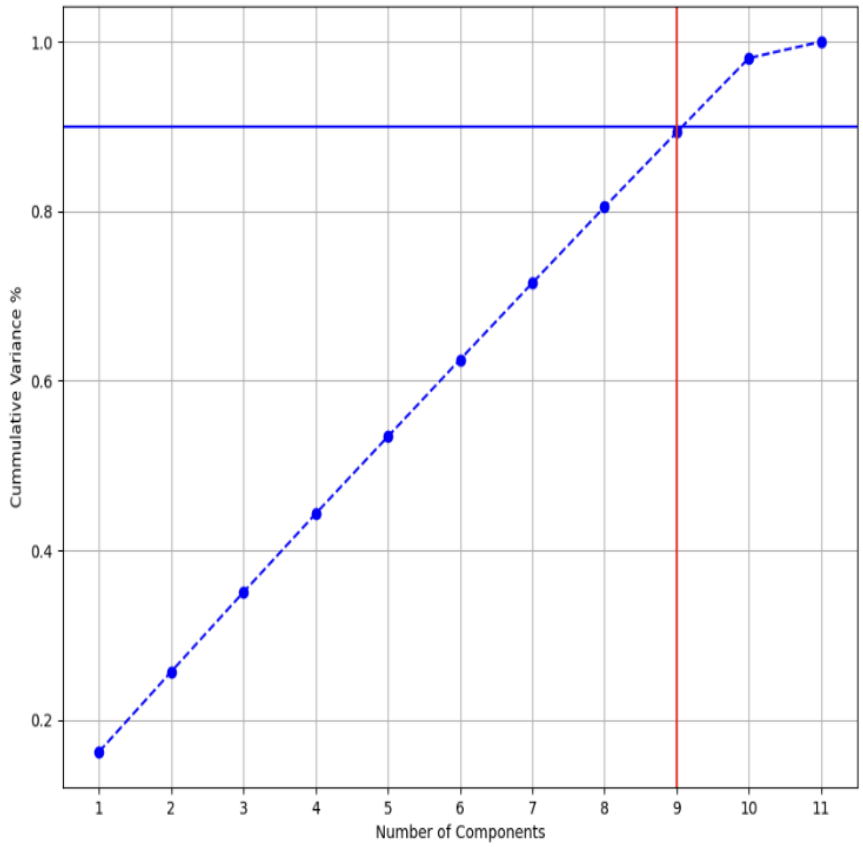

Fig 2

When we choose a threshold of 95% of total variance, we see that we are left with 9 components from PCA and these features are enough to explain 95% of total variance of the data.

Singular Value Decomposition (SVD

Singular Value Decomposition (SVD) Analysis emerges as a potent technique employed for dimensionality reduction in data analysis. This method involves breaking down a matrix into three constituent matrices, thereby unveiling inherent patterns and structures within the data. By conducting this decomposition, SVD isolates the most crucial components, facilitating the transformation of high-dimensional data into a condensed set of singular vectors. This

streamlined representation of the original data retains vital information, making it valuable for tasks like noise reduction, feature extraction, and optimizing computations in machine learning algorithms.

```
Feature income:
    Singular value: 352.58
    Variance ratio: 0.15
Feature age:
    Singular value: 269.04
    Variance ratio: 0.09
Feature profession:
    Singular value: 267.54
    Variance ratio: 0.09
Feature city:
    Singular value: 266.71
    Variance ratio: 0.09
Feature state:
    Singular value: 264.65
    Variance ratio: 0.08
Feature current_job_yrs:
    Singular value: 263.61
    Variance ratio: 0.08
Feature current_house_yrs:
    Singular value: 263.07
    Variance ratio: 0.08
Feature risk_flag:
    Singular value: 262.16
    Variance ratio: 0.08
```

Fig 3

With a 95% threshold of total explained variance, we see that we can reduce the dimensions from 11 to 9 by not losing much of the variance of the total data which also makes it easy for further analysis.

Variance Inflation Factor (VIF)

The Variance Inflation Factor (VIF) primarily serves the purpose of identifying multicollinearity among predictors in regression analysis. It quantifies the extent to which the variance of an estimated regression coefficient is inflated due to multicollinearity among predictors.

In certain instances, notably when VIF values are exceptionally high (typically surpassing 5 or 10), it signals the presence of strong multicollinearity. This suggests that specific predictors might be redundant or highly correlated. Eliminating such highly correlated features can result in a form of dimensionality reduction within the context of regression modeling.

Upon examining the provided VIF values for each feature, it becomes evident that the variables associated with 'house_owner_ship_single‘ and house_ownership_rented' exhibit robust multicollinearity with other predictors. This situation has the potential to impact the stability of coefficient estimates in regression models.

	Feature	VIF
0	income	1.000749
1	age	1.000526
2	profession	1.000595
3	city	1.001026
4	state	1.001063
5	current_job_yrs	1.000288
6	current_house_yrs	1.000221
7	risk_flag	1.000358
8	married_single_single	1.000411
9	house_ownership_owned	2.617638
10	house_ownership_rented	2.617245
11	car_ownership_yes	1.000874

Fig 4

Collinearity analysis

In our collinearity analysis, we opted to decrease the number of features using a random forest approach. This decision was based on the fact that random forest reduces a

substantial number of features, thereby mitigating collinearity, given that these features are also eliminated in the random forest process.

The features eliminated from random forest analysis are car_owned_yes, married_single_single ,house_ownership_rented ,house_ownership_owned and leaves us with 7 dimensions which is the least of all the dimensionality reduction techniques. Hence, I chose to follow random forest feature importances.

Discretization and Binarization:

Performed label encoding on city, state, profession features as they have large cardinality and one-hot encoding on 'married_single', 'house_ownership', 'car_ownership features and also avoided a dummy variable trap by dropping one of the columns of one category. Dataframe after one hot encoding is displayed below,

Variable transformation:

The dataframe undergoes standardization for all analyses, whether it be for training a model or implementing dimensionality reduction techniques. The numerical columns are standardized, resulting in data that is transformed uniformly.

	income	age	...	house_ownership_rented	car_ownership_yes
0	-1.277401	-1.579761	...	0.298286	-0.652676
1	0.896945	-0.583631	...	0.298286	-0.652676
2	-0.345349	0.939863	...	0.298286	-0.652676
3	0.439909	-0.525035	...	0.298286	1.532154
4	0.270842	-0.173460	...	0.298286	-0.652676

Fig 5

Outlier analysis and removal:

Outliers in the numerical column 'age' are identified and subsequently eliminated using the Interquartile Range (IQR) method. In this approach, the first quartile (Q1) and third quartile (Q3) of the dataset are determined. The interquartile range is then calculated, and lower and upper bounds are defined by subtracting 1.5 times the IQR from Q1 and adding 1.5 times the IQR to Q3. Any values falling outside these upper and lower bounds are treated as outliers and removed from the dataset.

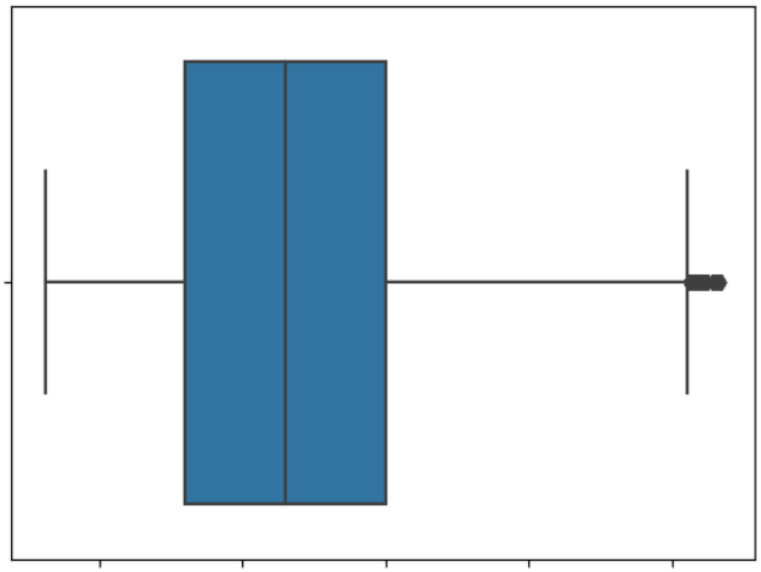

Fig 6

This box plot shows the age after outlier removal. The outliers currently present are with respect to the updated column where the IQR, Q1 and Q3 changed.

Covariance Matrix:

Below displays a heatmap of the sample covariance matrix of the dataset using all the features. From the plot, we can see that, experience' and 'current job years 'have larger covariance of 14 so they increase or decrease together so we can use any one feature, because Variability in the data can be drawn from one feature

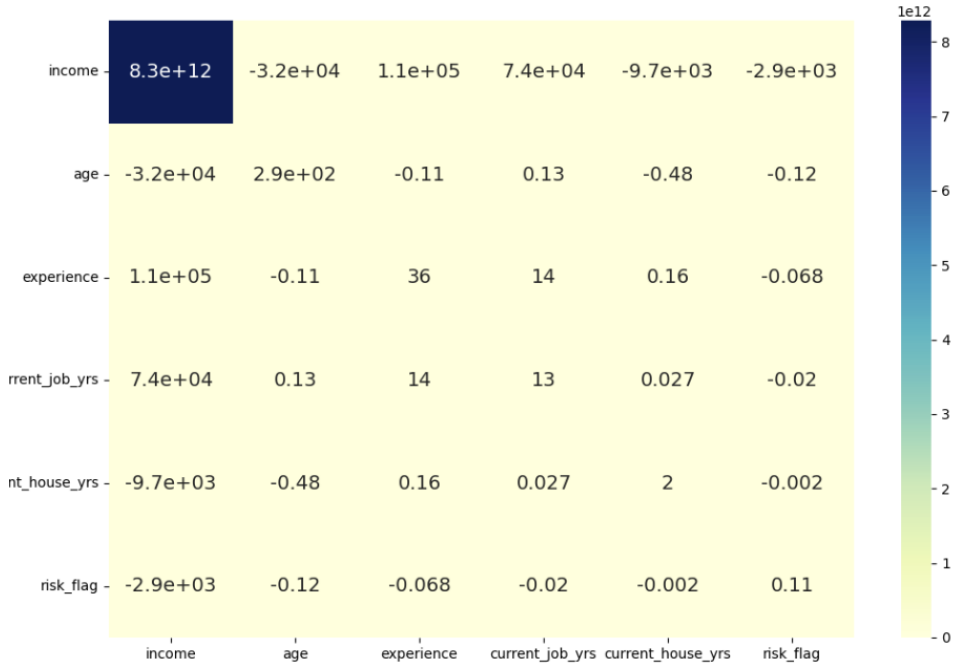

Fig 7

Pearson correlation matrix:

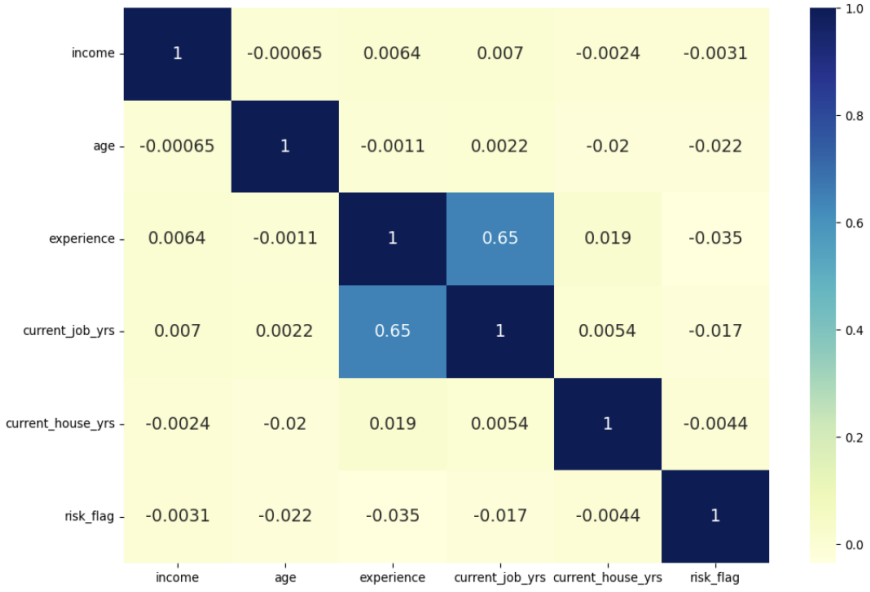

Fig 8

From the Pearson correlation matrix,, we can see that Correlation coefficient of current_job_yrs and Experience is high so We can drop any one,I chose experience to drop

Balanced or Imbalanced data:

The data is not balanced. So to balance the data i have performed oversampling .

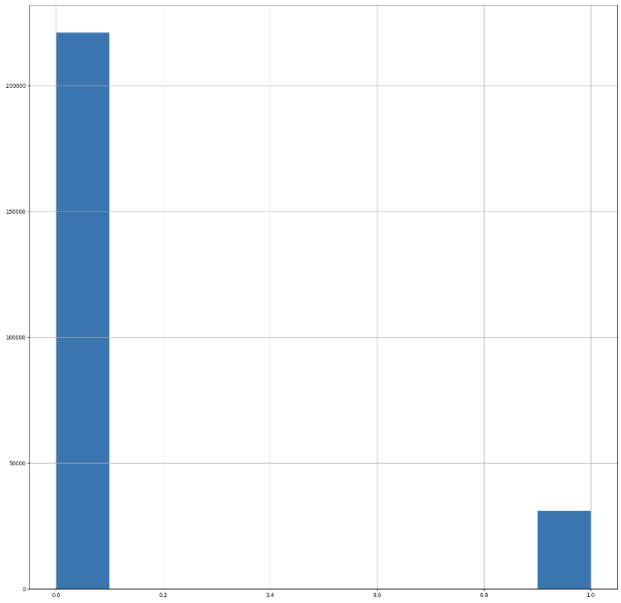

Fig 9

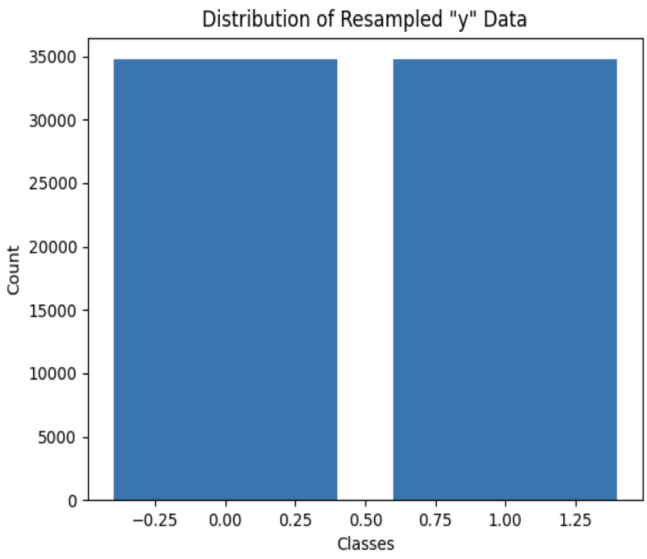

Fig 10

Phase II: Regression Analysis

Mine is a classification problem but to perform the regression analysis i have chose income as the target variable.

T-test analysis:

The t-test is a statistical technique utilized to ascertain whether there is a noteworthy difference between the means of two groups. It aims to determine if observed disparities between these groups result from genuine differences in the population or if they are merely outcomes of random chance. Among the various types of t-tests, the two-sample t-test is the most prevalent, comparing means of two independent groups to evaluate their statistical significance. This

involves analyzing the means and variances of the groups, calculating a t-statistic, and comparing it against a critical value from the t-distribution. The t-test evaluates both the magnitude of the difference between groups and the variability within each group to establish whether the observed difference is statistically significant. If the calculated t-value surpasses a critical value at a chosen significance level, typically 0.05, it suggests that the observed difference is unlikely due to random variability, indicating a statistically significant distinction between the groups.

Features such as married_single_single, house_ownership_rented , car_ownership_yes exhibit low p-values (close to 0), indicating strong significance in predicting income..

Age, profession, city,state, current_job_yrs, risk_flag, current_house_yrs, showcase higher p-values (above 0.05), suggesting they may not significantly contribute to predicting income based on this analysis.

F-test analysis:

The F-test is a statistical approach utilized to compare the variability among two or more groups, assessing whether they display significantly different variances. Specifically, the F-test examines whether the variances within these groups are equal or exhibit differences beyond what can be attributed to random variation. In the context of Analysis of Variance (ANOVA), the F-test is frequently employed to simultaneously compare variances across multiple groups. It

computes an F-statistic by dividing the variance between groups by the variance within groups. This ratio helps determine if the variability among group means is significantly greater than the variability within groups. The F-test aims to establish whether observed differences in variances between groups stem from actual disparities in population variances or if they are a result of random chance. By comparing the calculated F-value to a critical value from the F-distribution at a chosen significance level, typically 0.05, statisticians assess whether the observed differences in variances are statistically significant

Final regression model and prediction of dependent variable:

Linear regression serves as a fundamental statistical method employed to model the connection between a dependent variable (usually represented as 'y') and one or more independent variables (typically denoted as 'x'). The core aim of linear regression is to comprehend the impact of changes in the independent variable(s) on the dependent variable. The linear regression model operates under the assumption of a linear relationship between the independent variable(s) and the dependent variable..

.

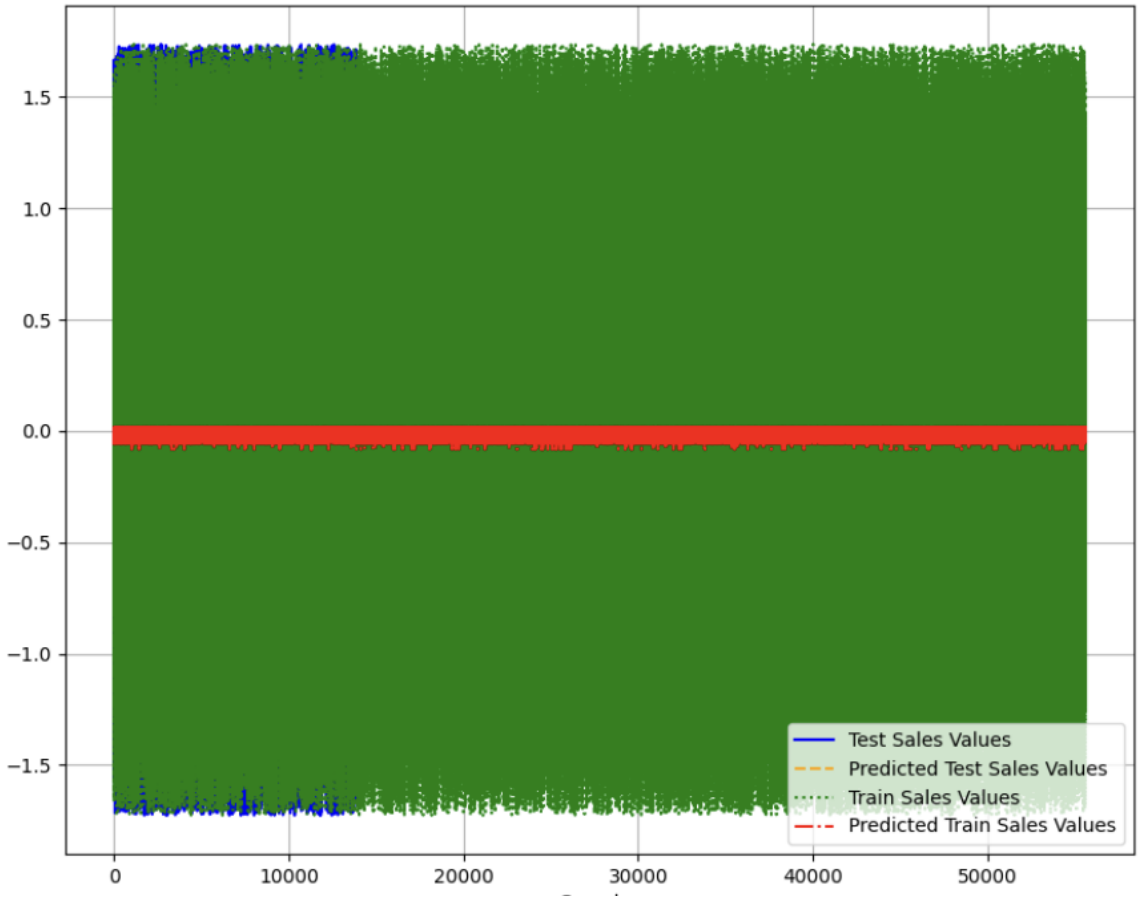

Fig 11

The training and predicted values can be seen in the above graph and the model has bias between predicted and training values.

Confidence interval analysis:

The below plot shows the confidence interval of the predicted values which has huge variance from the actual predicted values. This indicates that there is uncertainty in the estimated coefficients and this model is not reliable in predicting the 'Income'. This uncertainty is expected as our dataset is a classification problem and this is an independent variable and helps in predicting risk_flag rather than being predicted.

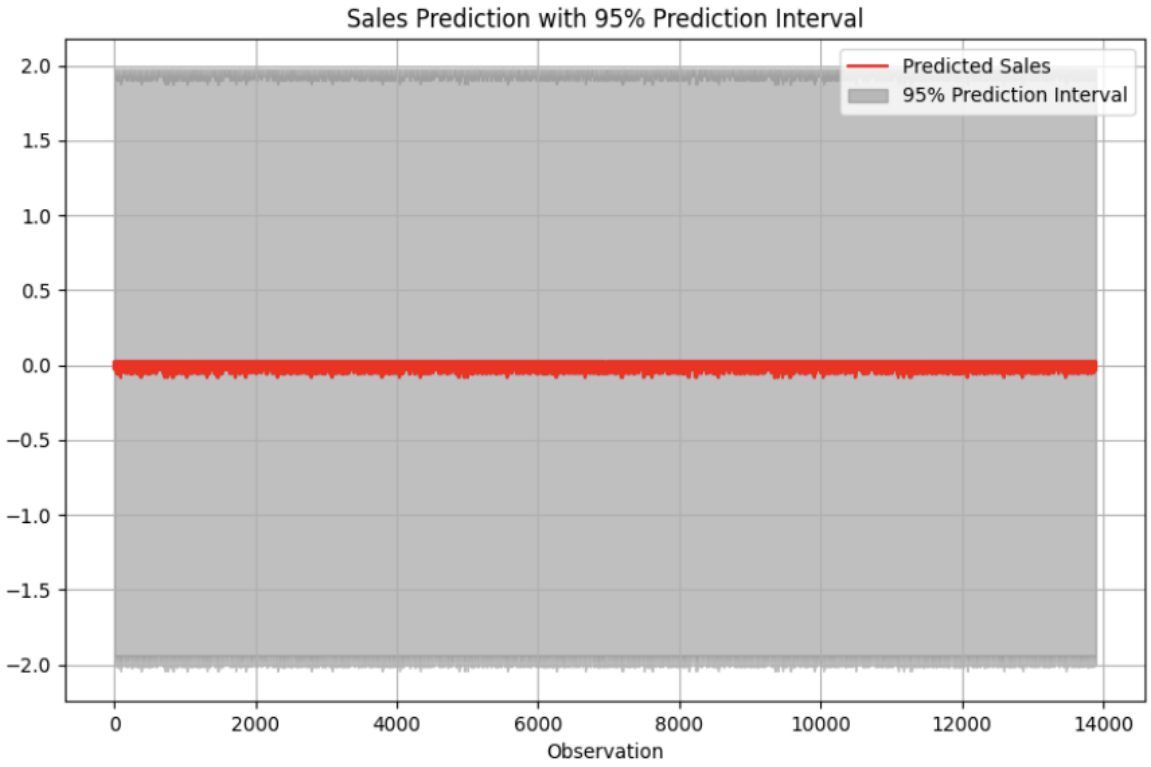

Fig 12

Stepwise regression and adjusted R-square analysis:

This approach entails initiating a model that includes all predictors and then iteratively removing the least significant variables until no further enhancement in the model's performance is discerned. The selection of backward stepwise regression is motivated by the presence of a few features exhibiting higher p-values for the T-test. The elimination process involves identifying the feature with the maximum p-value, surpassing the chosen threshold of 0.01. The goal is to arrive at a reduced set of features that adequately explains the variance of the target variable. Throughout each step, close attention is paid to the adjusted R-squared from the Ordinary Least Squares (OLS) summary to assess for any decline in value. If a drop is observed, the process halts, and the feature most recently removed is reconsidered for inclusion.

OLS Regression Results

Dep. Variable:	income	R-squared (uncentered):	0.001
Model:	OLS	Adj. R-squared (uncentered):	0.001
Method:	Least Squares	F-statistic:	10.49
Date:	Fri, 08 Dec 2023	Prob (F-statistic):	6.84e-07
Time:	20:50:16	Log-Likelihood:	-78880.
No. Observations:	55595	AIC:	1.578e+05
Df Residuals:	55592	BIC:	1.578e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
married_single_single	0.0088	0.004	2.065	0.039	0.000	0.017
house_ownership_rented	0.0070	0.004	1.648	0.099	-0.001	0.015
car_ownership_yes	-0.0211	0.004	-4.966	0.000	-0.029	-0.013

Omnibus:	50480.458	Durbin-Watson:	1.999
Prob(Omnibus):	0.000	Jarque-Bera (JB):	3351.922
Skew:	0.010	Prob(JB):	0.00
Kurtosis:	1.797	Cond. No.	1.01

Fig 13

Mean Squared Error: 1.0033678442927012

process Update	AIC	BIC	Adj R^2	p-value
const	157706.973	157814.083	0.001	0.806
house_ownership_owned	157705.033	157803.217	0.001	0.209
city	157704.613	157793.871	0.001	0.479
risk_flag	157703.114	157783.446	0.001	0.465
current_job_yrs	157701.648	157773.055	0.001	0.117
current_house_yrs	157702.105	157764.586	0.001	0.899
profession	157700.121	157753.676	0.001	0.275
age	157699.313	157743.942	0.001	0.17
state	157699.2	157734.903	0.001	0.192

Fig 14

If we see the Adj -R2 in my regression analysis it is very low i.e 0.001 as this is not a very rood regression analysis.

Both the analysis do not perform well explaining the target variable's variability. But in backward regression analysis there are less features.

Phase III: Classification Analysis

Decision Tree:

Initially Performed decision tree analysis without any pre or post pruning:

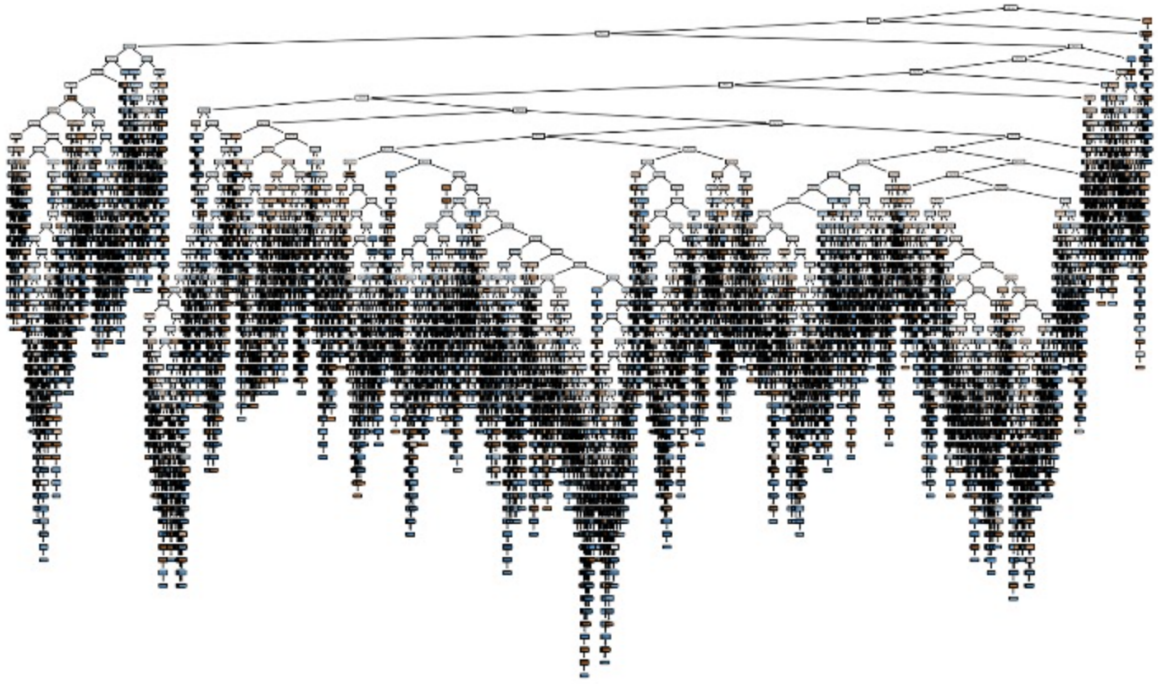

Fig 15

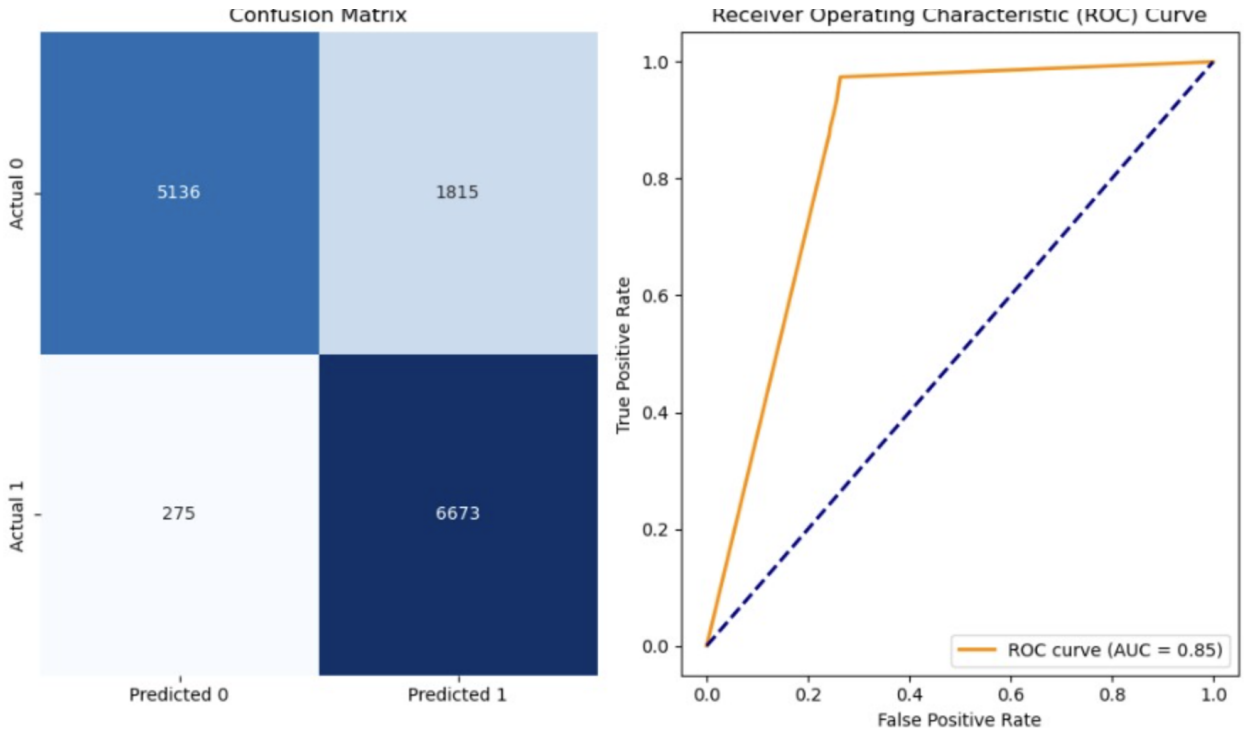
Confusion Matrix
Actual 0
Actual 1
5136
1815
275
6673
Predicted 0
Predicted 1
Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.85)

Fig 16

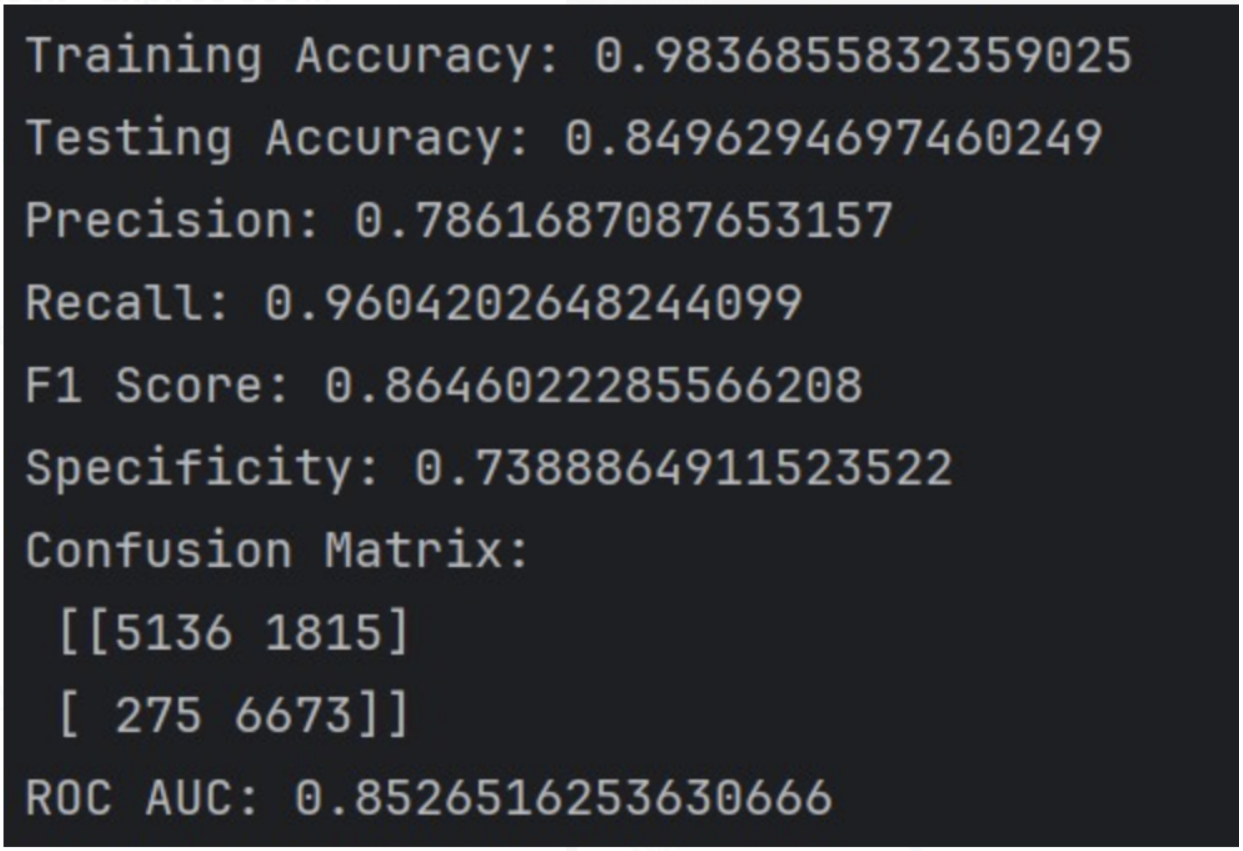

Fig 17

Classifier	Training Accuracy	Testing Accuracy	Precision	Recall	F1 Score	Specificity	ROC AUC
Decision Tree	0.9834877237161616	0.8479027268148788	0.7840188014101057	0.9602763385146805	0.8632423340665029	0.7355776147316933	0.8533513689501981

Fig 18

Pre-Pruning:

In the context of decision trees, pre-pruning techniques are employed to strike a balance between model complexity and predictive accuracy. Rather than permitting the tree to grow unrestrained until it precisely fits the training data, risking overfitting, pre-pruning techniques impose limitations on the tree's growth. A pre-pruned decision tree intentionally restricts its growth or depth during the construction phase, preventing it from expanding to the maximum

depth allowed by the algorithm. Prior to training the model, a grid search and stratified k-fold cross-validation are performed to identify the best hyperparameters

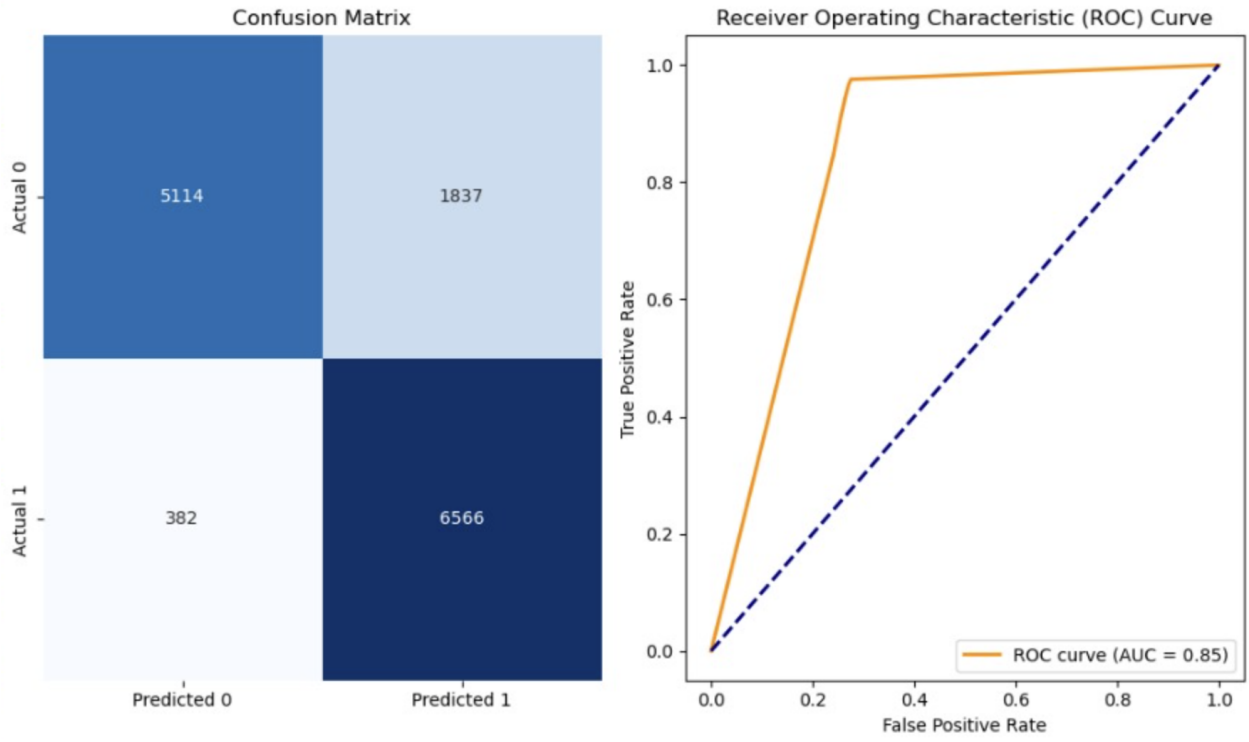

Fig 19

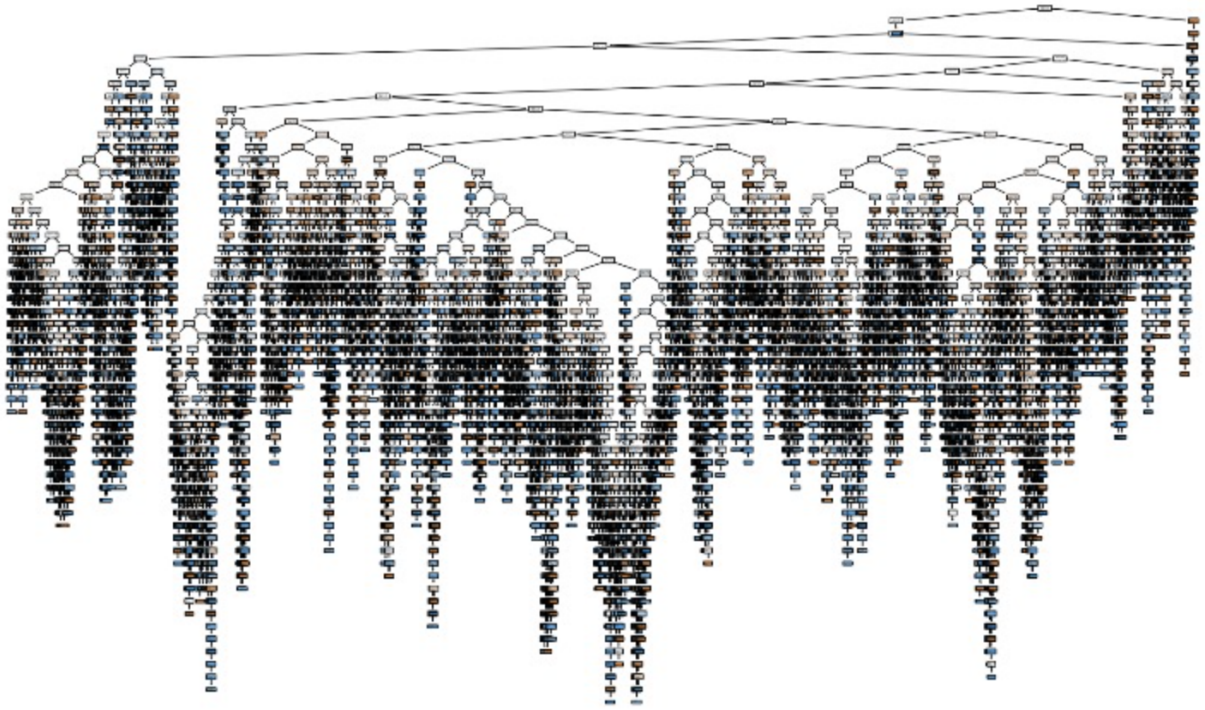

Fig 20

```
Best Parameters: {'criterion': 'entropy', 'max_depth': None, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 4, 'splitter': 'best'}
Training Accuracy: 0.9799442395898912
Testing Accuracy: 0.8403482264911145
Precision: 0.7813875996667857
Recall: 0.9450201496833621
F1 Score: 0.8554491564067488
Specificity: 0.7357214789238958
Confusion Matrix:
 [[5114 1837]
 [ 382 6566]]
ROC AUC: 0.8493849578019075
```

Fig 21

Classifier	Training Accuracy	Testing Accuracy	Precision	Recall	F1 Score	Specificity	ROC AUC
Descion Tree Prepruned	0.9796564439248134	0.8397726455140657	0.7813803790678269	0.9434369602763385	0.854795592358349	0.7361530715005036	0.85072413921051

Fig 22

Post pruned decision tree,

Post-pruning, alternatively termed tree pruning or cost complexity pruning, entails the elimination or collapsing of branches (subtrees) from a decision tree subsequent to its full construction. In contrast to pre-pruning, which involves imposing constraints during the tree-building phase, post-pruning takes place after the tree has been established. The primary aim of post-pruning is to diminish the tree's size, alleviate overfitting, and enhance the model's generalization capacity to unseen data. The parameter ccp_alpha governs the tree's complexity, with a larger ccp_alpha resulting in a more aggressively pruned tree. The optimal alpha value is determined to be 0.002.

Classifier	Training Accuracy	Testing Accuracy	Precision	Recall	F1 Score	Specificity	ROC AUC
ion Tree Postpruned	0.9834337620289595	0.8506367364558601	0.7859154929577464	0.9637305699481865	0.8657874321179209	0.7375917134225292	0.853307689975896

Fig 23

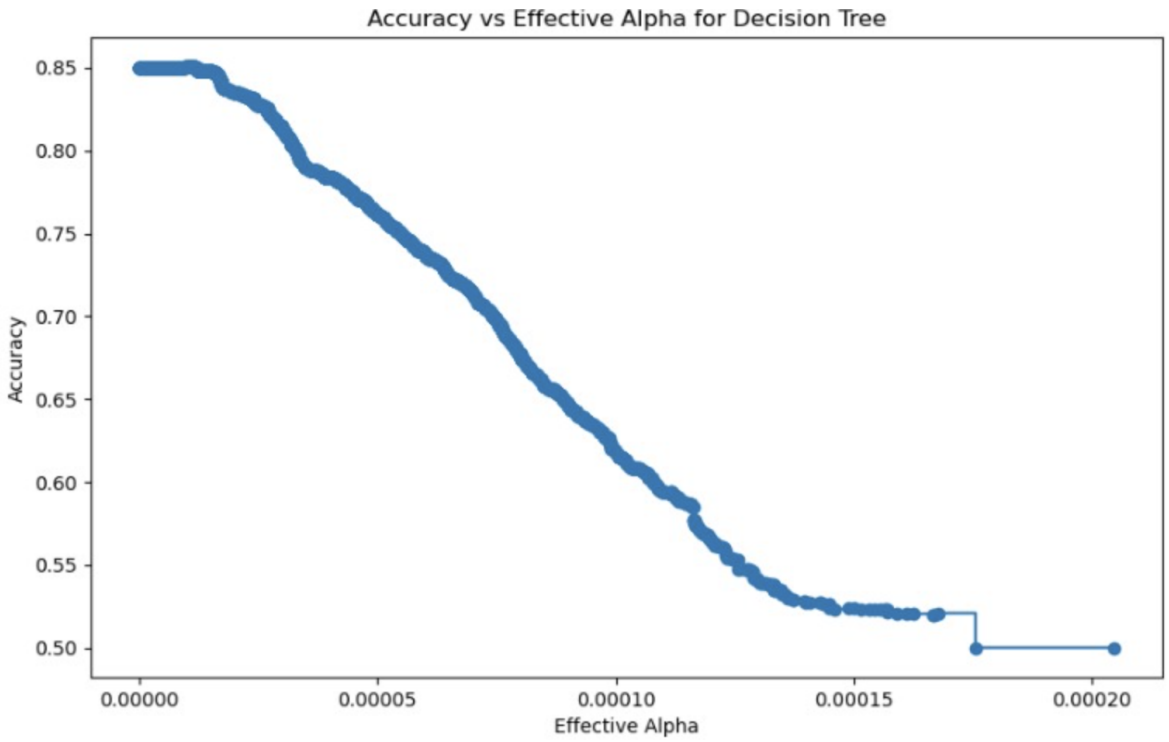

Fig 24

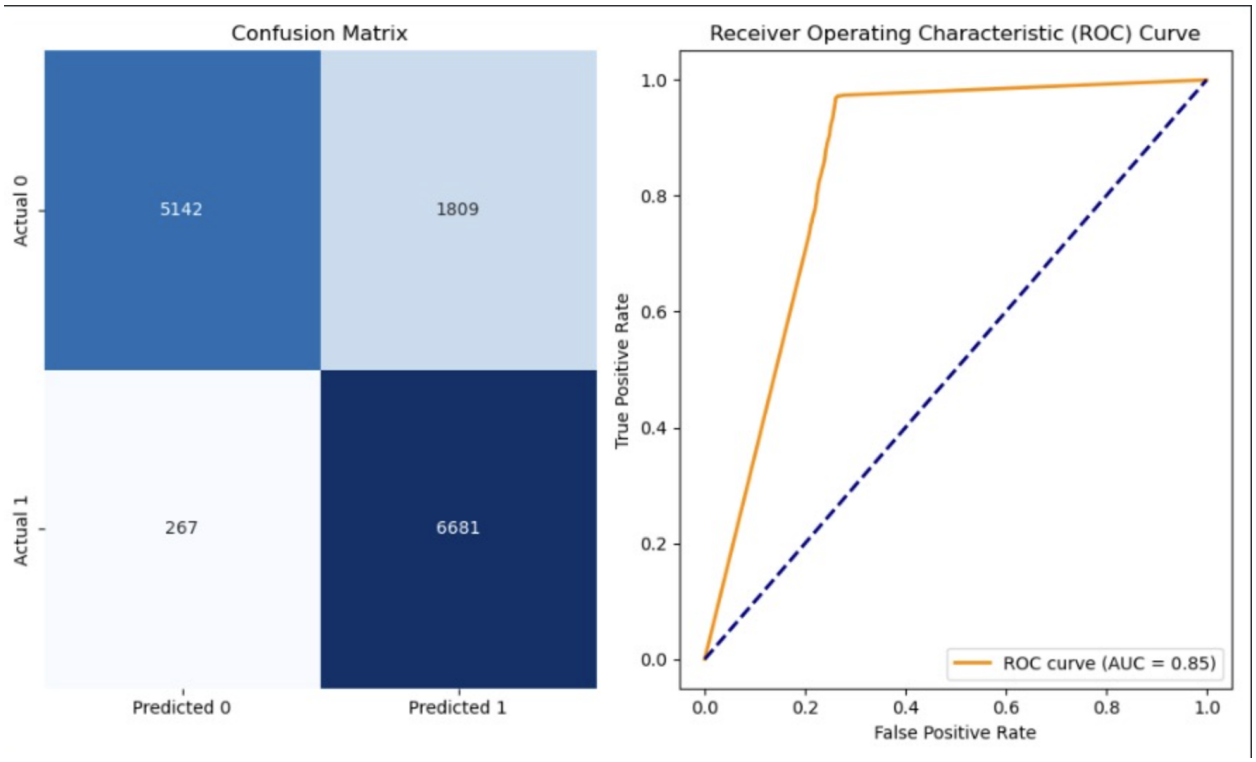
Confusion Matrix
Actual 0
Actual 1
5142
1809
267
6681
Predicted 0
Predicted 1
Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.85)

Fig 25

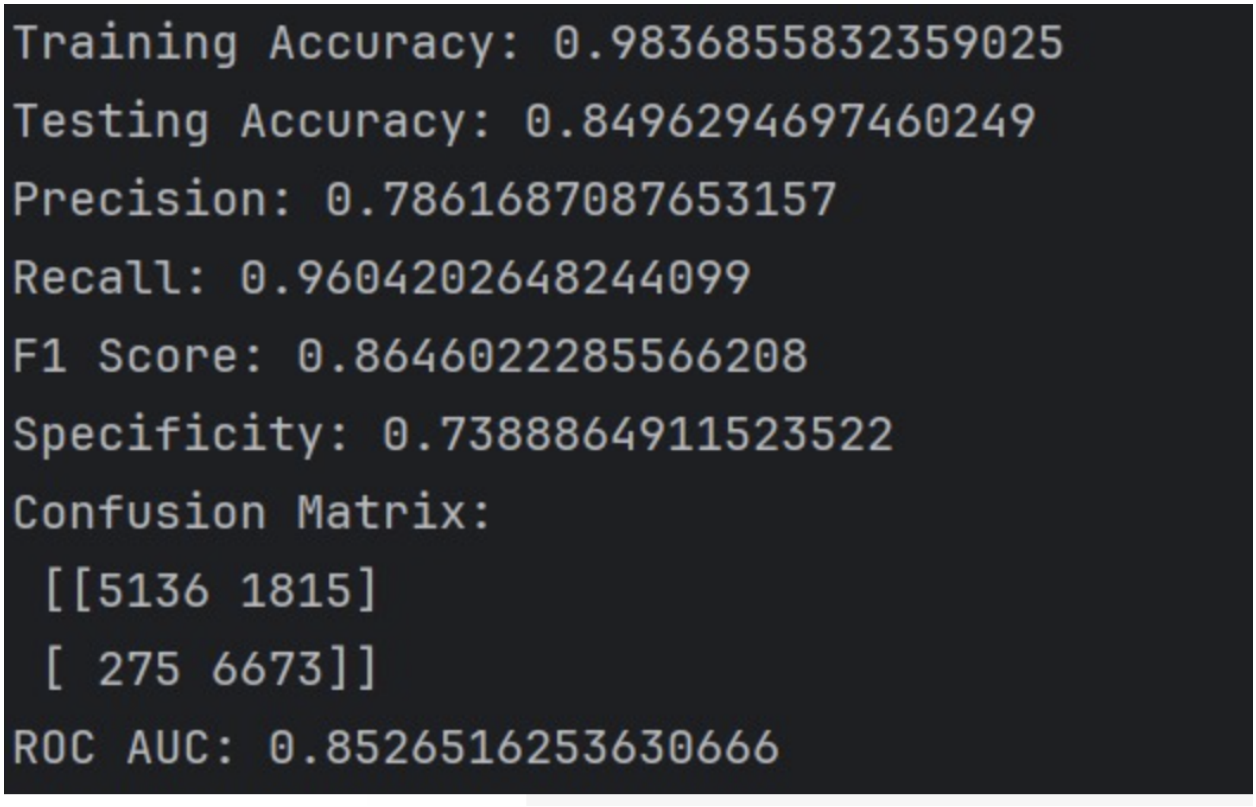

Fig 26

Logistic Regression:

Logistic Regression serves as a statistical technique employed for binary classification tasks, where it predicts the probability of an event occurring for a categorical dependent variable, typically represented as 0 or 1. A grid search was conducted on parameters 'C', 'Solver', and 'Penalty', alongside stratified k-fold cross-validation, resulting in the identification of the best hyperparameters.

Classifier	Training Accuracy	Testing Accuracy	Precision	Recall	F1 Score	Specificity	ROC AUC
Logistic Regression	0.5067721917438619	0.49766170228073964	0.49747549747549746	0.48215313759355521	0.4896944891097793	0.5131635735865343	0.497647702434187

Fig 27

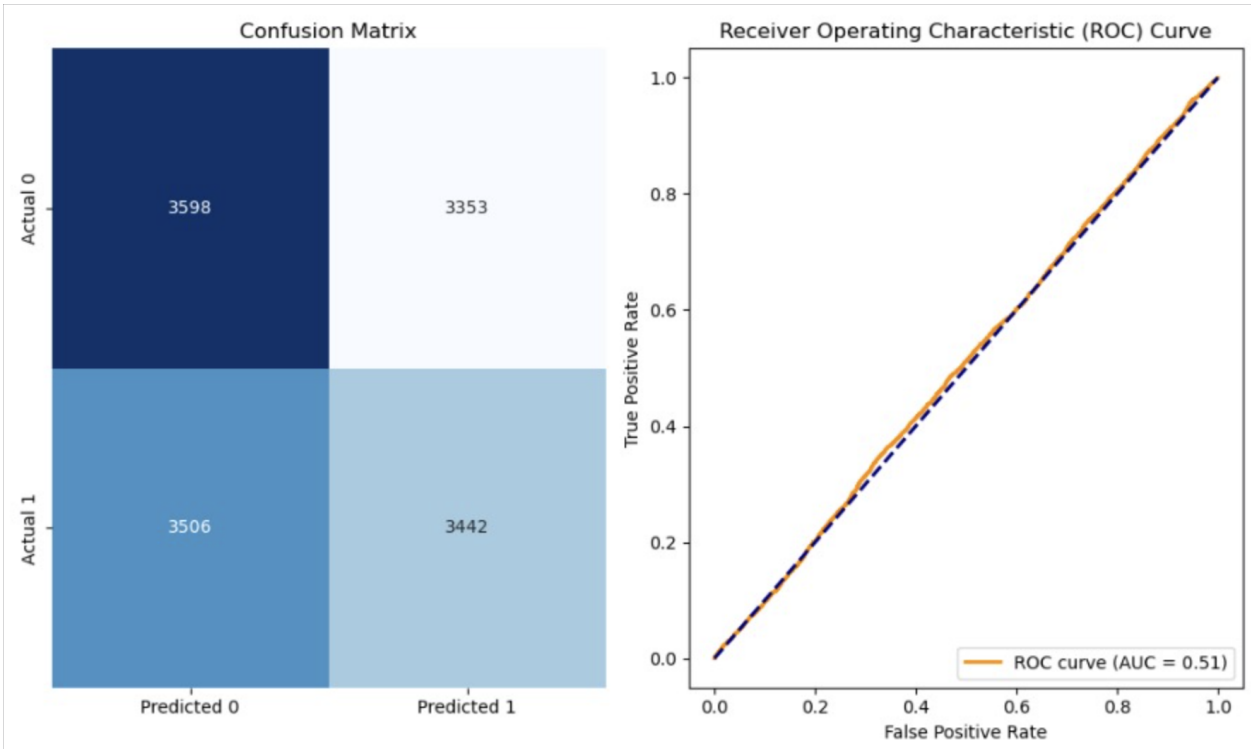

Fig 28

```
Best Hyperparameters: {'C': 0.001, 'penalty': 'l2', 'solver': 'newton-cg'}
Training Accuracy: 0.5043619030488353
Testing Accuracy: 0.5065112598028635
Precision: 0.5065489330389993
Recall: 0.4953943580886586
F1 Score: 0.5009095539547406
Specificity: 0.5176233635448136
Confusion Matrix:
 [[3598 3353]
 [3506 3442]]
ROC AUC: 0.5064803074602238
```

Fig 29

K Nearest Neighbors:

K-Nearest Neighbors (KNN) is a simple yet effective supervised learning algorithm used for both classification and regression tasks. The 'K' in KNN refers to the number of nearest neighbors used for prediction. When a new data point needs to be classified or predicted, the algorithm identifies its K nearest neighbors based on a chosen distance metric. Grid search and stratified k fold cross validation are used for best hyperparameters. The best value for k from error rate vs. k graph nearly matches with grid search result.

Classifier	Training Accuracy	Testing Accuracy	Precision	Recall	F1 Score	Specificity	ROC AUC
KNN	0.976310819318284	0.8450967695517663	0.7880571909167368	0.9440126655152562	0.8590138170388317	0.7462235649546828	0.8451181152349695

Fig 30

```
Optimal k: 1
Training Accuracy: 0.9755733429265222
Testing Accuracy: 0.8420030218001295
Precision: 0.78516562650024
Recall: 0.941565918249856
F1 Score: 0.8562827225130889
Specificity: 0.7424830959574162
Confusion Matrix:
 [[5161 1790]
 [ 406 6542]]
ROC AUC: 0.8420245071036361
```

Fig 31

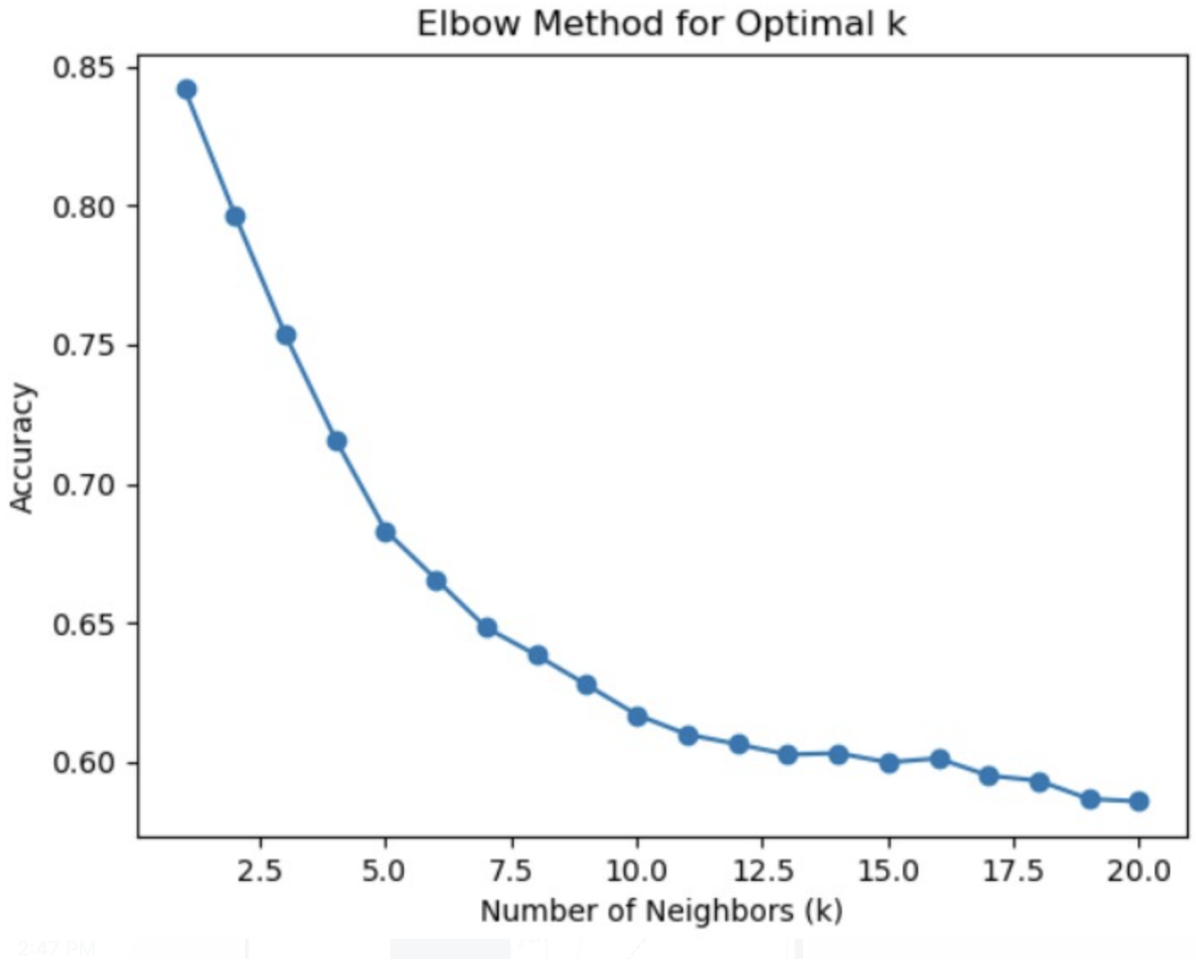
Elbow Method for Optimal k
Accuracy
0.85
0.80
0.75
0.70
0.65
0.60
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Number of Neighbors (k)

Fig 32

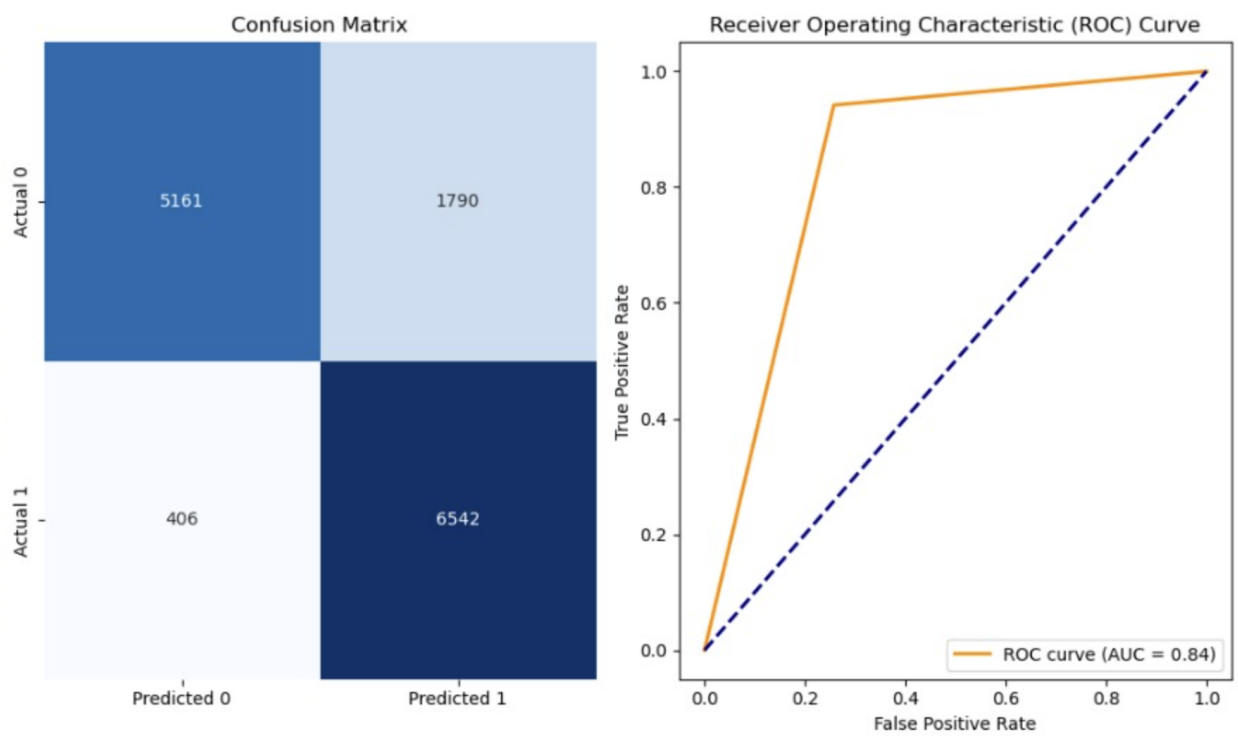

Fig 33

SVM:

SVM finds the hyperplane that maximizes the margin between the closest data points (support vectors) of different classes. Performed grid search with linear, poly and rbf kernel and stratified k fold cross validation on searching for the best kernel. Best kernel is found to be the radial basis function which is widely used for nonlinear problems.

```
Best Hyperparameters: {'kernel': 'rbf'}
```

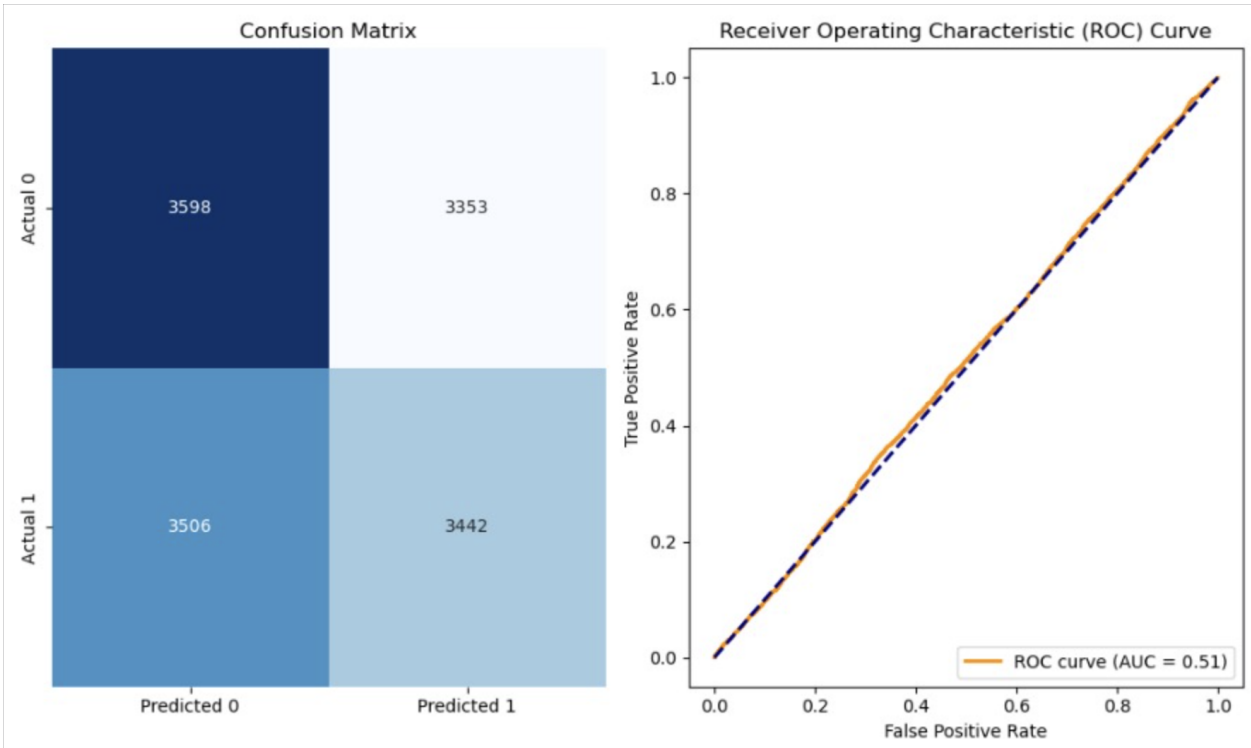

Fig 34

Naïve Bayes:

Naive Bayes is a popular and simple probabilistic machine learning algorithm used primarily for classification tasks. It's based on Bayes' theorem with an assumption of independence among predictors (features), known as the "naive" assumption. Grid search is not possible as there are no parameters. Stratified K fold approach followed in calculating the test accuracy.

```
Training Accuracy: 0.5043619030488353
Testing Accuracy: 0.5065112598028635
Precision: 0.5065489330389993
Recall: 0.4953943580886586
F1 Score: 0.5009095539547406
Specificity: 0.5176233635448136
Confusion Matrix:
 [[3598 3353]
 [3506 3442]]
ROC AUC: 0.5064803074602238
```

FIg 35

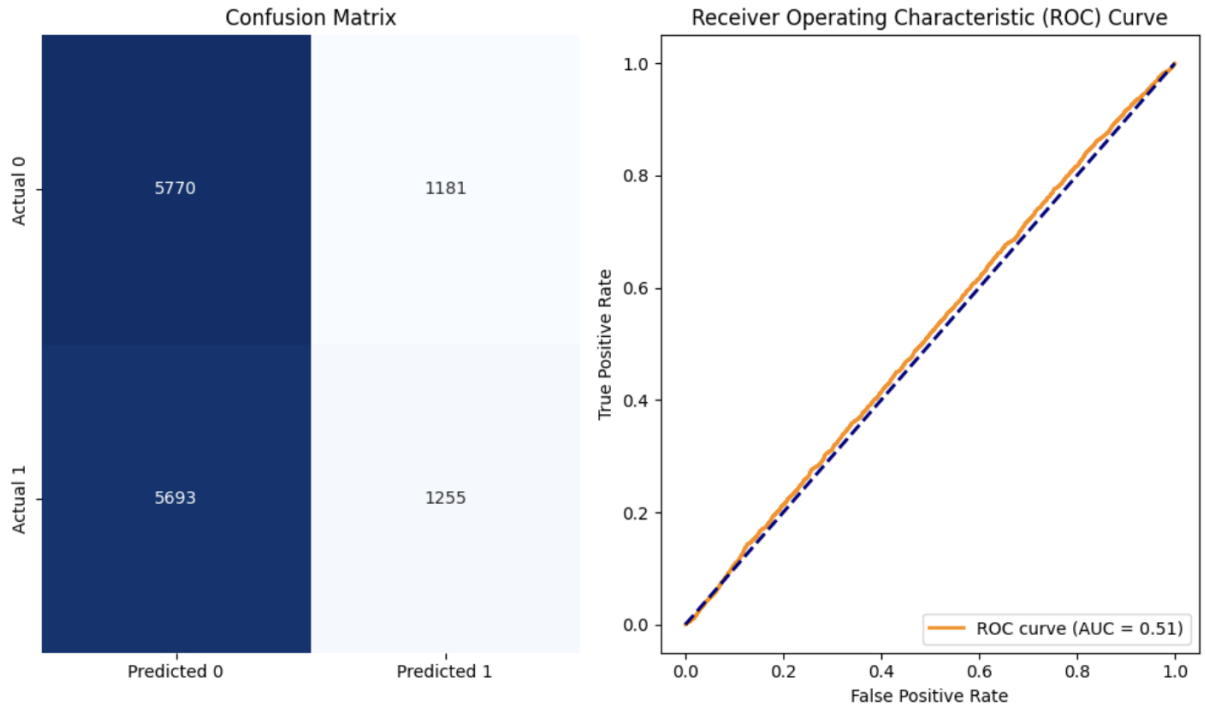

Fig 36

Random Forest:

Random Forest, a robust ensemble learning technique applicable to both classification and regression tasks, builds multiple decision trees during training. The model produces its final prediction by considering the mode of classes for classification or the mean prediction for regression from the individual trees. Best hyperparameters are determined through grid search and stratified k-fold cross-validation. The model exhibits commendable accuracy, with decent precision and good specificity, albeit showing lower recall and f-score.

Classifier	Training Accuracy	Testing Accuracy	Precision	Recall	F1 Score	Specificity	ROC AUC
Random Forest	0.9831459663638816	0.9351751924598892	0.9164026993527062	0.957685664939551	0.9365894855373356	0.9126744353330456	0.9609515456787032

Fig 37

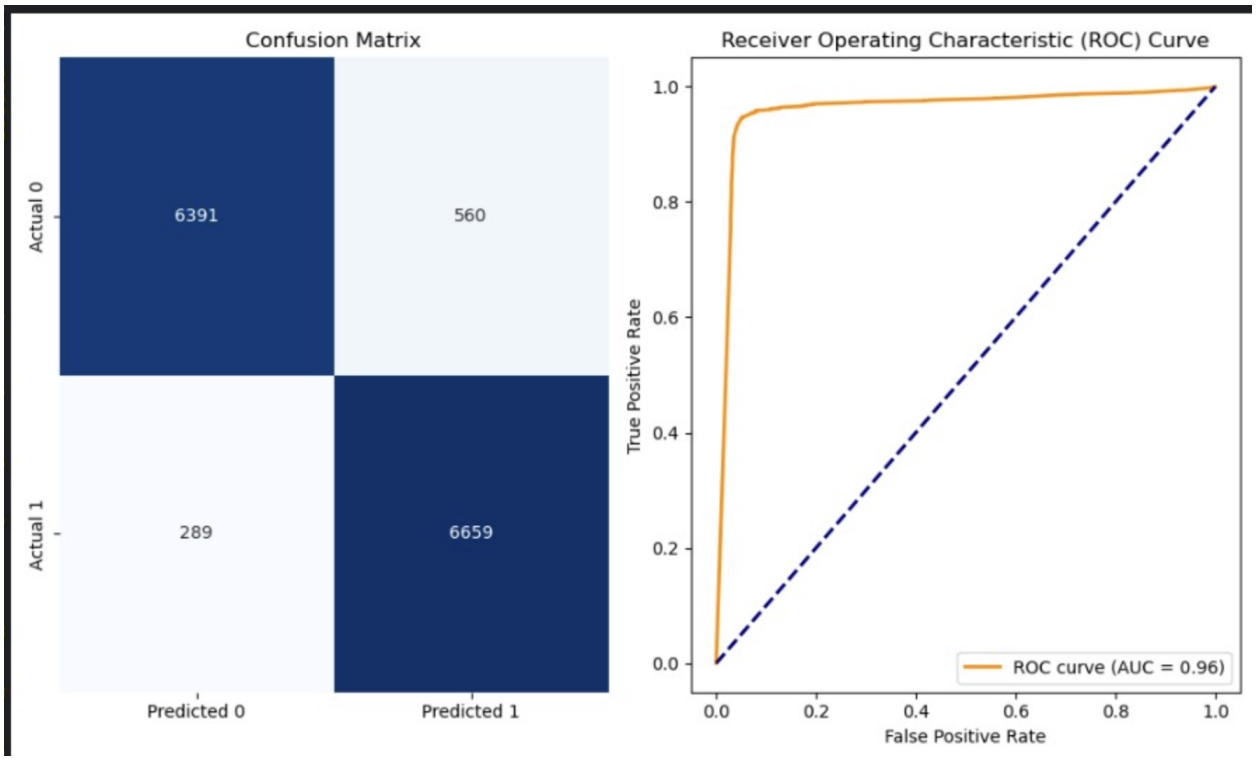

Fig 38

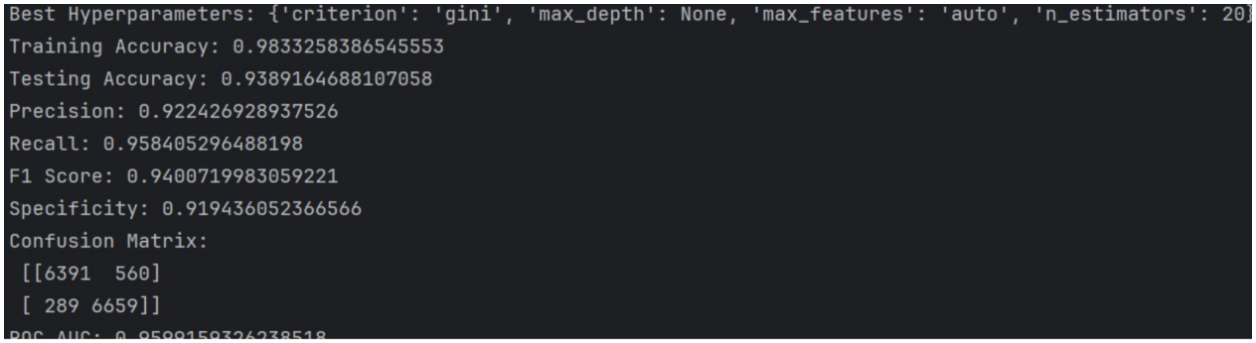

Fig 39

Multi layered Perceptron:

A Multi-Layer Perceptron (MLP) represents a variant of artificial neural networks characterized by multiple layers of nodes or neurons. Renowned for its capacity to grasp intricate associations between input and output data, this feedforward neural network adeptly maps sets of input data to corresponding outputs. To optimize its performance, a grid search and stratified k-fold cross-validation are employed to identify the best hyperparameters. While the model demonstrates commendable accuracy, precision, specificity, and Area Under the Curve (AUC), it does exhibit lower performance in terms of recall and f-score

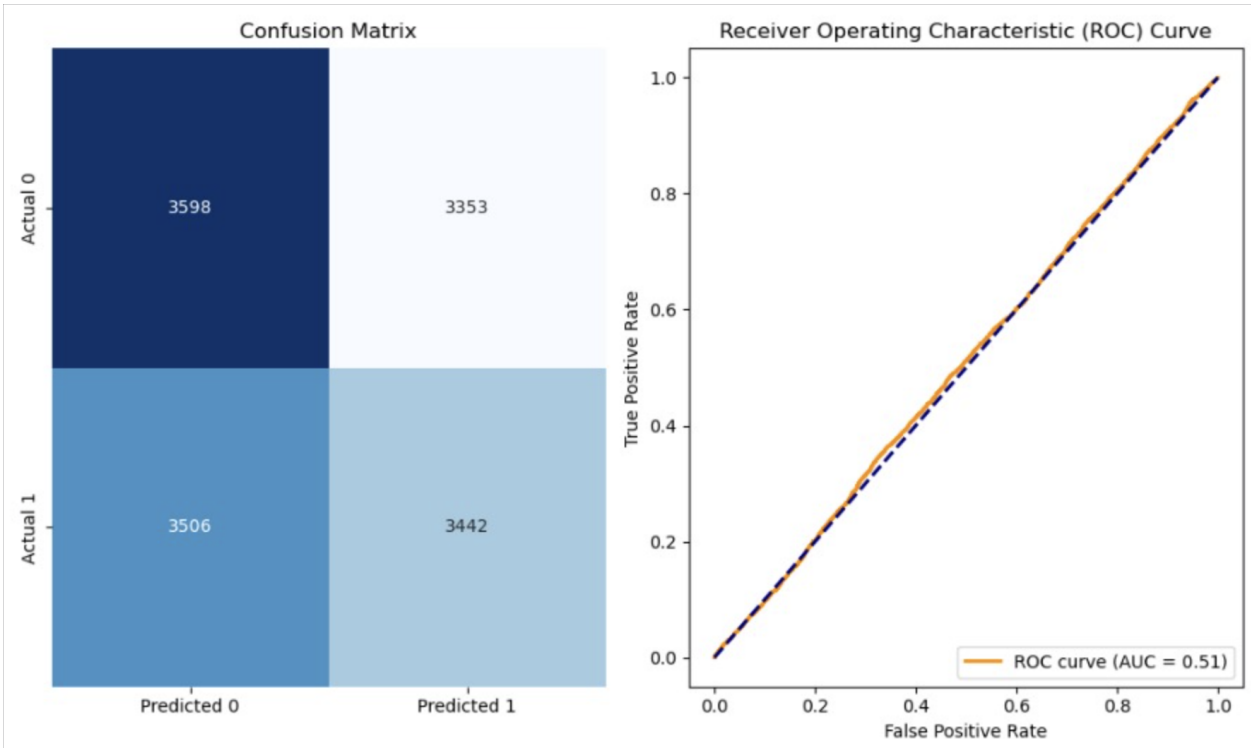

Fig 40

```
Training Accuracy: 0.5043619030488353
Testing Accuracy: 0.5065112598028635
Precision: 0.5065489330389993
Recall: 0.4953943580886586
F1 Score: 0.5009095539547406
Specificity: 0.5176233635448136
Confusion Matrix:
 [[3598 3353]
 [3506 3442]]
ROC AUC: 0.5064803074602238
```

Fig 41

KMeans Clustering:

K-means clustering is a popular unsupervised machine learning algorithm used for clustering similar data points into groups or clusters .Silhouette Score measures how

well-separated the clusters are. A higher silhouette score indicates better-defined clusters. From the graph, it is seen that the optimal number of clusters is 4.

Analysis of all classifiers:

If we observe all the parameters which decide the the efficiency of a model,Randoforest is has all the best outcomes of around 96% accuracy and 0.96 auc . so Randomforest classifier is the best classifier.

Figure 55

Silhouette analysis of KMeans clustering

Within-Cluster Sum of Squares (WCSS): Measures the compactness of clusters. Lower WCSS implies more compact clusters. We also see the optimal number of clusters from the WCSS plot as 2.

Figure 52

WCSS plot analysis for optimal k in kmeans

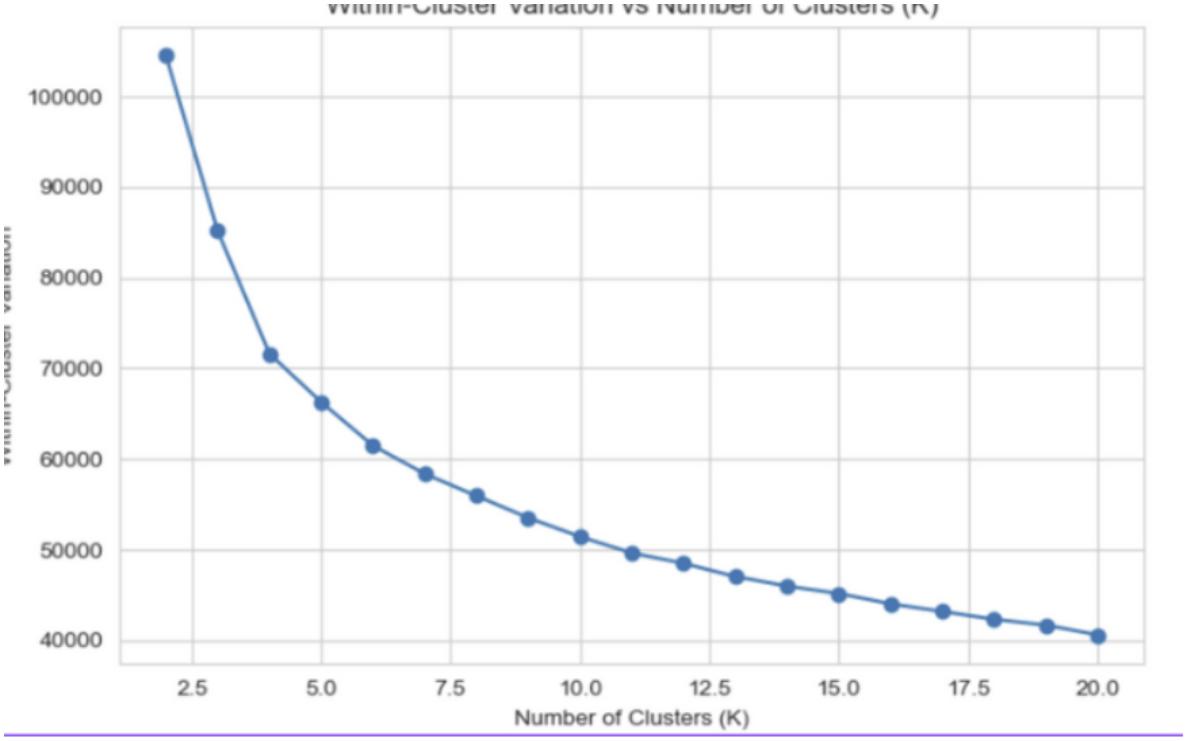
Within-Cluster Variation vs Number of Clusters (K)
Within-Cluster Variation
100000
90000
80000
70000
60000
50000
40000
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Number of Clusters (K)

Fig 42

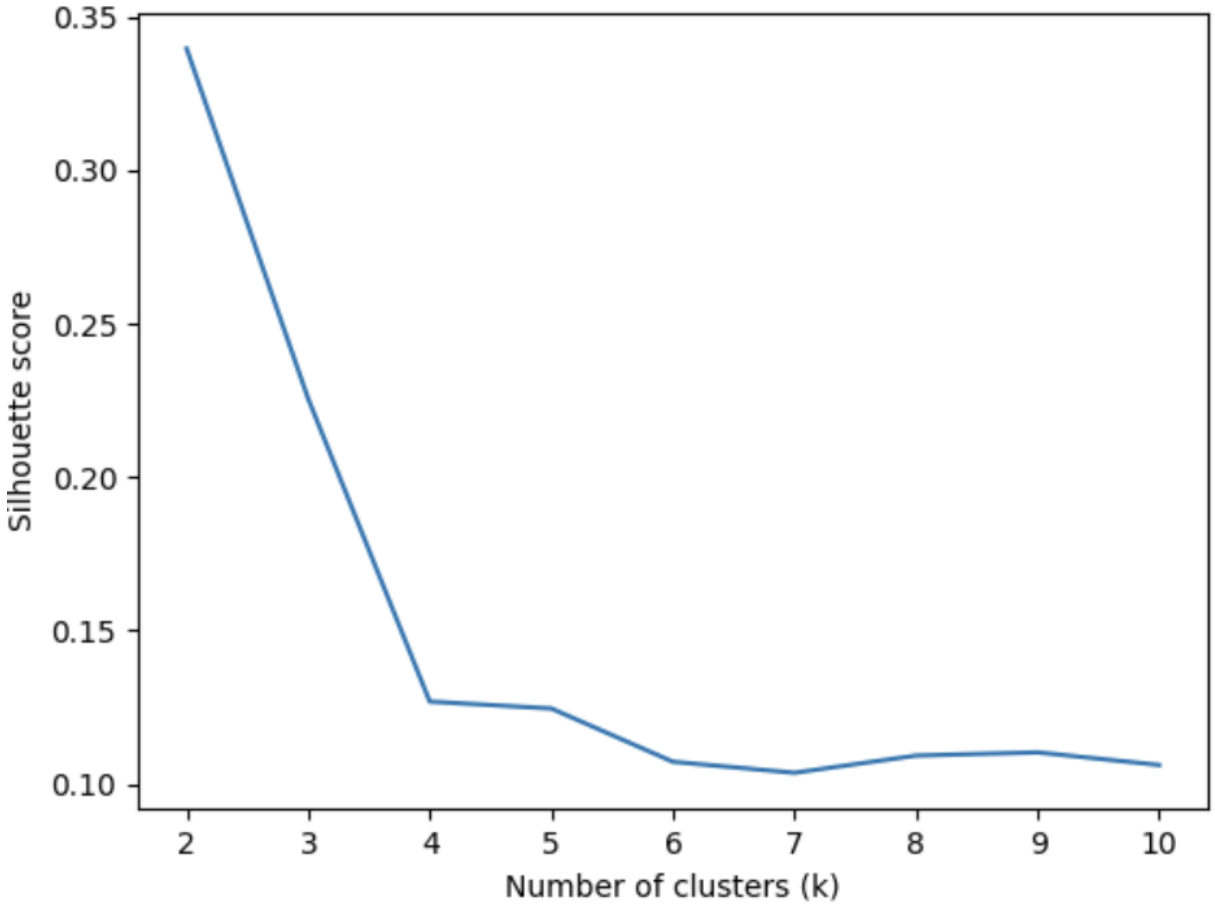
0.35
0.30
0.25
0.20
0.15
0.10
Silhouette score
2
3
4
5
6
7
8
9
10
Number of clusters (k)

Fig 43

Fig 44

Best k based on silhouette score: 2

Fig 45

Apriori algorithm:

The Apriori algorithm is a classic algorithm used in association rule mining, particularly for discovering frequent itemsets in transactional databases or datasets. I have kept the support and confidence of 0.000001 to generate the association rules if we clearly observe there arent any valid association rules being generated.

Rules generated from Apriori

```
[5 rows x 17 columns]
   _  a  b  c  e  g  h  i  j  m  n  o  r  s  t  u  y
0  0  0  0  1  1  0  0  1  0  1  1  1  0  0  0  0  0
1  0  1  0  0  1  1  0  0  0  0  0  0  0  0  0  0  0
2  1  0  1  1  1  1  0  0  0  1  0  1  1  1  1  1  1
3  1  0  0  1  1  1  0  1  0  0  0  1  1  1  1  1  1
4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
Processing 12 combinations | Sampling itemset size 12
             antecedents         consequents  antecedent support  consequent support   support  confidence          lift  leverage  conviction  zhangs_metric
319932     (n, t, r, y, h)           (_, u, c)            0.000014            0.000029  0.000014         1.0  34747.000000  0.000014         inf       0.999986
298284             (h, _)  (n, y, u, c, e, s)            0.000014            0.000029  0.000014         1.0  34747.000000  0.000014         inf       0.999986
298319  (n, t, y, u, h, e)              (_, c)            0.000014            0.000029  0.000014         1.0  34747.000000  0.000014         inf       0.999986
298318  (n, t, y, h, _, e)              (u, c)            0.000014            0.000029  0.000014         1.0  34747.000000  0.000014         inf       0.999986
298317  (n, t, y, h, _, u)              (e, c)            0.000014            0.000043  0.000014         1.0  23164.666667  0.000014         inf       0.999971
```

Fig 46

Recommendations

This project concludes with the practical application of diverse machine learning techniques on an airlines dataset to predict whether a person will be granted a loan. Throughout this endeavor, I have garnered valuable experience in implementing various models, fine-tuning hyperparameters, and ultimately identifying optimal models. This hands-on exploration not only equipped me with the skills to navigate real-world datasets but also emphasized the importance of efficient training through hyperparameter optimization in the process of model selection.

Looking ahead to the future of this project, I aim to augment this dataset with additional information from datasets containing practical features such as credit card score, fraud history, and the purpose of the loan. Additionally, I plan to explore the utilization of models like XGBoost to further enhance the training of robust models.

In terms of performance analysis, the RandomForest classification stands out as the best-performing model among all classifiers, achieving an accuracy of nearly 96%. The KMeans

analysis reveals that the optimal number of clusters is 2, which aligns with the silhouette score for clusters equal to 2. This suggests that data points are not well-differentiated into distinct clusters, with each data point appearing similar to every other point. This implies that a single cluster best encapsulates the dataset. Lastly, in association rule mining, despite setting the support and confidence as 0.00001, there are no relevant rules or associations between the features.

Appendix

Necessary Libraries:

```
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import statsmodels.api as sm
from scipy import stats
from IPython.core.interactiveshell import InteractiveShell
from imblearn.over_sampling import RandomOverSampler
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn import preprocessing
from prettytable import PrettyTable
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn import tree
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc_curve, auc
from sklearn.model_selection import StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.decomposition import TruncatedSVD
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import mean_squared_error
from mlxtend.preprocessing import TransactionEncoder
```

Defined a function evaluate:

```
def evaluate_model(model, X_train, y_train, X_test, y_test, classifier_name):
    # Train the model
    model.fit(X_train, y_train)

    # Predictions on the training set
    train_predictions = model.predict(X_train)

    # Predictions on the test set
    test_predictions = model.predict(X_test)

    # Evaluate performance metrics
    train_accuracy = accuracy_score(y_train, train_predictions)
    test_accuracy = accuracy_score(y_test, test_predictions)

    precision = precision_score(y_test, test_predictions)
    recall = recall_score(y_test, test_predictions)
    f1 = f1_score(y_test, test_predictions)

    confusion_mat = confusion_matrix(y_test, test_predictions)

    # ROC curve and AUC
    fpr, tpr, thresholds = roc_curve(y_test, model.predict_proba(X_test)[:, 1])
    roc_auc = auc(fpr, tpr)

    # Specificity
    specificity = specificity_score(y_test, test_predictions)
```

EDA analysis of different features:

```
#### Exploring Experience ########
print(pd.crosstab(df['risk_flag'], df['experience'], margins=True))
pd.crosstab(df['experience'], df['risk_flag']).plot.bar(width=0.9)
plt.title("Number of loan takers according to Experience")
plt.show()

# Factor plot
sns.catplot(x='experience', y='risk_flag', data=df, kind='point')
plt.title("Default rate vs Experience")
plt.show()

#### Exploring Current job years #####
print(pd.crosstab(df['risk_flag'],df['current_job_yrs']))

pd.crosstab(df['current_job_yrs'] ,df['risk_flag'] ).plot.bar(width=0.9)
plt.title("Number of loan takers according to Current Job")
plt.show()


sns.catplot(x='current_job_yrs', y='risk_flag', data=df, kind='point')
plt.title("Default rate vs Current Job Years")
plt.show()

#### Exploring Housing years ####
print(pd.crosstab(df['risk_flag'],df['current_house_yrs']))
```

Balancing The data:

```
# ### balancing the data ####
df["risk_flag"].hist(figsize=(20,20))
plt.show()
df.drop_duplicates(inplace=True)
### as we can see the data is not balanced ####
### we are oversampling ###
X = df.drop( labels: "risk_flag",axis = 1)
y = df["risk_flag"]
rs=RandomOverSampler()
X,y=rs.fit_resample(X,y)
y = pd.Series(y)
value_counts = y.value_counts()
plt.bar(value_counts.index, value_counts.values)
plt.xlabel('Classes')
plt.ylabel('Count')
plt.title('Distribution of Resampled "y" Data')
plt.show()
df=pd.concat( objs: [X,y],axis=1)
df_bal=df
print(df.risk_flag.value_counts())
```

Standardizing and label encoding and one-hot encoding:

```
####now the data is balance
##### as we know the state and city and profession features have a lot of cardinality so i will be using label encoder
label_encode_columns=['profession','city','state']
label_encoder = preprocessing.LabelEncoder()
for column in label_encode_columns:
    df[column] = label_encoder.fit_transform(df[column])

### one hot encoding for features with less cardinality #####
df_one_hot_encode=pd.get_dummies(df,columns=['married_single','house_ownership','car_ownership'],drop_first=True)
### features values with True and False to 1 and 0 ###
features_to_convert = ['married_single_single','house_ownership_owned', 'house_ownership_rented', 'car_ownership_yes']
for feature in features_to_convert:
    df_one_hot_encode[feature] = df_one_hot_encode[feature].astype(int)
print(df_one_hot_encode.columns)
non_encoded_features=df_one_hot_encode.select_dtypes(['int','float']).columns
print(non_encoded_features)

## standardizing and splitting the dataset
scaler=StandardScaler()
df_standard=df_one_hot_encode;
df_standard[non_encoded_features]=scaler.fit_transform(df_one_hot_encode[non_encoded_features])
df_stan=df_standard


df_train,df_test=train_test_split(
     *arrays: df_standard,test_size=0.20,shuffle=True,random_state=5805
)
```

PCA

```
### Performing PCA
df_standard=df_stan
X_scaled = df_standard.drop(columns='risk_flag')
pca = PCA()
X_pca = pca.fit_transform(X_scaled)
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance_ratio = np.cumsum(explained_variance_ratio)
n_components_range = np.arange(1, 11)
n_components_needed = np.argmax(cumulative_variance_ratio >= 0.85) + 1
print("Number of components needed to explain 85% of the variance:", n_components_needed)
plt.figure(figsize=(10,8))
x=np.arange(1,12,step=1)
plt.plot( *args: x,cumulative_variance_ratio,marker='o', linestyle='--', color='b')
plt.xlabel('Number of Components')
plt.ylabel('Cummulative Variance %')
plt.xticks(np.arange(1, 12, step=1))
plt.axhline(y=0.90, color='b', linestyle='-')
plt.axvline(x=9,color='r',linestyle='-')
plt.grid(True)
plt.show()
```

SVD and VIF:

```
###### performing SVD ###############
f_standard=df_stan
vd = TruncatedSVD(n_components=len(df_standard.columns))
vd_model = svd.fit(df_standard)
=0
or i, singular_value in enumerate(svd_model.singular_values_):
   variance_ratio = svd_model.explained_variance_ratio_[i]
   print(f"Feature {df_standard.columns[i]}:")
   print(f"\tSingular value: {singular_value:.2f}")
   print(f"\tVariance ratio: {variance_ratio:.2f}")

## Print the top 2 left singular vectors ####
rint(f"Top 2 left singular vectors:\n{pd.DataFrame(svd_model.components_.T[:2], colu

####### VIF #################
 Create an empty list to store VIFs
if_values = []
f_standard = df_stan

 Loop through each feature and calculate VIF
or i in range(len(df_standard.columns)):
   vif = variance_inflation_factor(df_standard.values, i)
   vif_values.append(vif)

 Create a DataFrame to store features and VIF values
if_df = pd.DataFrame({"Feature": df_standard.columns, "VIF": vif_values})
```

Feature Importance by Rnadomforest:

```
## performing regression analysis###

 ##  splitting the dataset into train and test
f_standard=df_stan
rint(df_standard.head())
rint(df_standard.columns)
f_train,df_test=train_test_split(
    *arrays: df_standard,test_size=0.20,shuffle=True,random_state=5805)

### Performing backward  Regression ###
ependent_variable='income'
ndependent_variables=[col for col in df_train.columns if col != dependent_variable]

_train=df_train[independent_variables]
_test=df_test[independent_variables]
_train=df_train[dependent_variable]
_test=df_test[dependent_variable]

## creating an array for droped features ###
ropped_features=[]

# adding constant
_model_train=sm.add_constant(X_train)
_test=sm.add_constant(X_test)

# training Model
odel=sm.OLS(y_train,X_model_train).fit()
rint(model.summary())
```

Regression Analysis:

```
######## Removing current house years ####
X_model_train.drop(columns=['current_house_yrs'],inplace=True)
dropped_features.append('current_house_yrs')
model=sm.OLS(y_train,X_model_train).fit()
p_value = model.pvalues['profession']
summary_table.add_row(['profession',round(model.aic,3),round(model.bic,3),round(model.rsquared_adj,3),round(p_value, 3)])
print(model.summary())
######## Removing Profession ####
X_model_train.drop(columns=['profession'],inplace=True)
dropped_features.append('prefession')
model=sm.OLS(y_train,X_model_train).fit()
p_value = model.pvalues['age']
summary_table.add_row(['age',round(model.aic,3),round(model.bic,3),round(model.rsquared_adj,3),round(p_value, 3)])
print(model.summary())

#### Removing age ####
X_model_train.drop(columns=['age'],inplace=True)
dropped_features.append('age')
model=sm.OLS(y_train,X_model_train).fit()
p_value = model.pvalues['state']
summary_table.add_row(['state',round(model.aic,3),round(model.bic,3),round(model.rsquared_adj,3),round(p_value, 3)])
print(model.summary())
#### removing state###
X_model_train.drop(columns=['state'],inplace=True)
dropped_features.append('state')
model=sm.OLS(y_train,X_model_train).fit()
```

Decision Tree:

```
df_standard=df_stan
stratified_kfold=StratifiedKFold(n_splits=5,shuffle=True,random_state=5805)
df_train,df_test=train_test_split(
     *arrays: df_standard,test_size=0.20,shuffle=True,random_state=5805)
dependent_variable='risk_flag'
independent_variables=[col for col in df_train.columns if col != dependent_variable]

X_train=df_train[independent_variables]
X_test=df_test[independent_variables]
y_train=df_train[dependent_variable]
y_test=df_test[dependent_variable]
#
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

evaluate_model(model, X_train, y_train, X_test, y_test, classifier_name: 'Decision Tree')

feature_importance = model.feature_importances_
feature_importance_dict = dict(zip(independent_variables, feature_importance))
sorted_feature_importance = sorted(feature_importance_dict.items(), key=lambda x: x[1], reverse=True)
print("Feature Importance:")
for feature, importance in sorted_feature_importance:
    print(f"{feature}: {importance:.2f}")
plt.figure(figsize=(20,12))
tree.plot_tree(model,rounded=True,filled=True)
plt.show()
```

Pre Pruned Decision Tree:

```
)est_params = grid_model.best_params_
)est_accuracy = grid_model.best_score_

)rint("Best Parameters:", best_params)
:valuate_model(grid_model, X_train, y_train, X_test, y_test, classifier_name: 'Descion Tree Prepruned')

)est_model = grid_model.best_estimator_
)lt.figure(figsize=(20, 12))
:ree.plot_tree(best_model, rounded=True, filled=True)
)lt.show()
iccuracies = []


)ath = model.cost_complexity_pruning_path(X_train, y_train)
:cp_alphas, impurities = path.ccp_alphas, path.impurities

for ccp_alpha in ccp_alphas:
    pruned_model = DecisionTreeClassifier(ccp_alpha=ccp_alpha, random_state=5805)
    pruned_model.fit(X_train, y_train)
    y_pred = pruned_model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracies.append(accuracy)
```

Logistic Regression:

```
df_standard=df_stan
df_train,df_test=train_test_split(
     *arrays: df_standard,test_size=0.20,shuffle=True,random_state=5805)
dependent_variable='risk_flag'
independent_variables=[col for col in df_train.columns if col != dependent_variable]

X_train=df_train[independent_variables]
X_test=df_test[independent_variables]
y_train=df_train[dependent_variable]
y_test=df_test[dependent_variable]
param_grid={
    "C": [0.001, 0.01, 0.1, 1, 10, 100],
    "solver": ["newton-cg", "lbfgs", "liblinear", "sag", "saga"],
    "penalty": ["none", "l1", "l2", "elasticnet"],
}
log_model=LogisticRegression(random_state=5805)
grid_model=GridSearchCV(log_model,param_grid,cv=stratified_kfold,scoring='accuracy')
grid_model.fit(X_train, y_train)
print("Best Hyperparameters:", grid_model.best_params_)


evaluate_model(grid_model, X_train, y_train, X_test, y_test, classifier_name: 'Logistic Regression')
"
```

Random Forest:

```
df_standard=df_stan
df_train,df_test=train_test_split(
     *arrays: df_standard,test_size=0.20,shuffle=True,random_state=5805)
dependent_variable='risk_flag'
independent_variables=[col for col in df_train.columns if col != dependent_variable]

X_train=df_train[independent_variables]
X_test=df_test[independent_variables]
y_train=df_train[dependent_variable]
y_test=df_test[dependent_variable]
param_grid={
    "n_estimators": [5,10,15,20],
    "max_features": ['auto', 'sqrt', 'log2'],
    "criterion": ['gini', 'entropy'],
    "max_depth":[None, 5, 10, 20, 50],
}
rf_model = RandomForestClassifier(class_weight="balanced", random_state=5805)
grid_model=GridSearchCV(rf_model,param_grid,cv=stratified_kfold,scoring='accuracy')
grid_model.fit(X_train, y_train)
print("Best Hyperparameters:", grid_model.best_params_)
evaluate_model(grid_model, X_train, y_train, X_test, y_test, classifier_name: 'Random Forest')
```

KNN :

```
# ### KNN algorithm ##########
df_standard=df_stan
df_train,df_test=train_test_split(
     *arrays: df_standard,test_size=0.20,shuffle=True,random_state=5805)
dependent_variable='risk_flag'
independent_variables=[col for col in df_train.columns if col != dependent_variable]

X_train=df_train[independent_variables]
X_test=df_test[independent_variables]
y_train=df_train[dependent_variable]
y_test=df_test[dependent_variable]


def train_knn_model(X_train, y_train, X_test, y_test, k):
    knn_model = KNeighborsClassifier(n_neighbors=k)
    knn_model.fit(X_train, y_train)
    y_pred = knn_model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    return accuracy
# Perform the elbow method to find optimal k
k_values = range(1, 21)  # Try different values of k
accuracies = []

for k in k_values:
    accuracy = train_knn_model(X_train, y_train, X_test, y_test, k)
    accuracies.append(accuracy)
# Plot the accuracy against different values of k
plt.plot( *args: k_values, accuracies, marker='o')
plt.title('Elbow Method for Optimal k')
```

Naive Bayes :

```
## Performing Naive Bayes #####
df_standard=df_stan
df_train,df_test=train_test_split(
    *arrays: df_standard,test_size=0.20,shuffle=True,random_state=5805)
dependent_variable='risk_flag'
independent_variables=[col for col in df_train.columns if col != dependent_variable]

X_train=df_train[independent_variables]
X_test=df_test[independent_variables]
y_train=df_train[dependent_variable]
y_test=df_test[dependent_variable]

NB_model = GaussianNB()

NB_model.fit(X_train, y_train)

evaluate_model(NB_model, X_train, y_train, X_test, y_test, classifier_name: 'Naive Bayes')
#
#
```

MLP:

```
#### Performing MLP classifier ######
df_standard=df_stan
df_train,df_test=train_test_split(
    *arrays: df_standard,test_size=0.20,shuffle=True,random_state=5805)
dependent_variable='risk_flag'
independent_variables=[col for col in df_train.columns if col != dependent_variable]

X_train=df_train[independent_variables]
X_test=df_test[independent_variables]
y_train=df_train[dependent_variable]
y_test=df_test[dependent_variable]
param_grid={
    "hidden_layer_sizes":  [(100, ), (100, 50), (100, 50, 25)],
    "activation": ['relu', 'tanh', 'logistic'],
    "solver": ['adam', 'sgd'],
    "alpha": [ 0.001, 0.01],
}
mlp_model = MLPClassifier(random_state=5805)
grid_model=GridSearchCV(mlp_model,param_grid,cv=stratified_kfold,scoring='accuracy')
grid_model.fit(X_train, y_train)
print("Best Hyperparameters:", grid_model.best_params_)
```

K-Means Clustering

```
# # ######### K Means  Clustering ###
df_standard=df_stan

def get_silhouette_score(df_standard, k):
    kmeans = KMeans(n_clusters=k, random_state=5805)
    kmeans.fit(df_standard)
    return silhouette_score(df_standard, kmeans.labels_)

# Calculate silhouette score for a range of k values
k_range = range(2, 11)
silhouette_scores = [get_silhouette_score(df_standard, k) for k in k_range]

## plotting the silhouette score###
plt.plot( *args: k_range, silhouette_scores)
plt.xlabel('Number of clusters (k)')
plt.ylabel('Silhouette score')
plt.show()

max_score_index = np.argmax(silhouette_scores)
best_k = k_range[max_score_index]
print(f"Best k based on silhouette score: {best_k}")
 # Replace with the determined optimal k
kmeans_final = KMeans(n_clusters=2, init='k-means++', random_state=5805)
kmeans_final.fit(df_standard)
clusters = kmeans_final.predict(df_standard)
```

Apriori Association analysis:

```
# # silhouette_score(df_standard, cluster_labels)
# # #
# # # ##### performing Apriori ###
df_balanced=df_bal
df_ap = df_balanced[['income', 'age', 'current_job_yrs', 'current_house_yrs']]
df_ap['income'] = pd.cut(df_ap['income'], bins=3, labels=['low', 'mid', 'high'])
df_ap['age'] = pd.cut(df_ap['age'], bins=3, labels=['young', 'mid_age', 'old'])
df_ap['current_job_yrs'] = pd.cut(df_ap['current_job_yrs'], bins=2, labels=['fresher', 'experienced'])
df_ap['current_house_yrs'] = pd.cut(df_ap['current_house_yrs'], bins=2, labels=['recent_move_in', 'stayed_long'])
te = TransactionEncoder()
te_ary = te.fit(df_ap).transform(df_ap)
df_ap = pd.DataFrame(te_ary, columns=te.columns_)
print(df_ap.head(5))
# convert the data into binary format
df_ap= df_ap.astype('int')
print(df_ap.head(5))
frequent_itemsets = apriori(df_ap, min_support=0.00001, use_colnames=True, verbose=1)
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=0.00001)
rules = rules.sort_values(['confidence'], ascending=False)
print(rules.head(5).to_string())
```

References

1) Loan Dataset to Risk_flag. www.kaggle.com. https://www.kaggle.com/datasets/subhamjain/loan-prediction-based-on-customer-behavior/data
2) scikit-learn. (2019). scikit-learn: machine learning in Python. Scikit-Learn.org. https://scikit-learn.org/stable/
3) Pandas. (2018). Python Data Analysis Library — pandas: Python Data Analysis Library. Pydata.org. https://pandas.pydata.org/