Hackathon On Developing a Predictive Model In GST

सत्यमेव जयते

Analytics Hackathon on Developing a predictive model in GST

Submitted by: -

Akankshya Jena

Sruti Shriya Sahu

Debasmita Jena

From: -

Sambalpur University Institute of Information Technology (SUIIT)

Jyoti Vihar, Burla, Odisha

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all those who supported and contributed to the successful completion of our project on **GST Prediction**.

First and foremost, we would like to thank our project guide, for their continuous guidance, invaluable feedback, and insightful suggestions throughout the course of this project. Their expertise and encouragement were instrumental in shaping our research and approach.

We also extend our appreciation to Indian goverment for providing us with the necessary resources and a conducive environment for conducting this study. Special thanks to our faculty members for their constructive criticism, which helped refine our work.

We are grateful to our peers and friends for their support and willingness to collaborate, which made the journey more enjoyable and fruitful. Finally, we acknowledge the teamwork and dedication of all the members of our group: Debasmita Jena, Sruti Shriya Sahu , Akankshya Jena. Their contributions and effort were crucial in completing this project.

We sincerely hope that our work adds value to the growing body of knowledge on GST prediction and serves as a foundation for further research in this field.

Thank you.

CONTENTS

ABSTRACT.......... 4

1. INTRODUCTION 4

  1.1 Background.......... 4

  1.2 Problem Statement.......... 5

2. OBJECTIVES.......... 5

3. METHODOLOGY 6

  3.1 Data Preprocessing 6

4. Model Selection and Evolution.......... 17

  4.1 Random Forest Classifier 17

  4.2 Logistic Regression.......... 23

  4.3 Boosting 24

  4.4 ANN.......... 30

  4.5 Decission Tree.......... 32

  4.6 Hyper Tuning and Cross-Validation.......... 33

5. Comparing all the Models.......... 34

6. Conclusion 35

7.Future Scope 35

ABSTRACT

The Goods and Services Tax (GST) is a vital component of the taxation system, but it faces significant challenges such as tax fraud, non-compliance, and inefficient taxpayer segmentation. This project aims to develop a machine learning-based predictive model to address these challenges, focusing on **fraud detection**, **taxpayer segmentation**, **anomaly detection**, and **compliance monitoring** within the GST framework. The proposed solution leverages advanced machine learning models, to enhance the accuracy and efficiency of fraud detection and anomaly identification. The Random Forest model is utilized to classify taxpayers into risk categories, aiding in the segmentation process and providing insights into taxpayer behaviour. The project also incorporates **predictive analysis** to forecast taxpayer behaviour and GST revenue trends, helping tax authorities make informed decisions and optimize their monitoring efforts. The results demonstrate the effectiveness of these models in identifying high-risk taxpayers, detecting fraud early, and improving compliance rates. The integration of these machine learning techniques offers a data-driven approach to tax administration, reducing revenue loss, increasing transparency, and promoting a more efficient and secure GST system.

1. INTRODUCTION

1.1 Background

The Goods and Services Tax (GST) is an indirect tax applied to the supply of goods and services in many countries, including India. It is a unified tax that has replaced various indirect taxes such as VAT, service tax, and excise duty, simplifying the tax structure. GST is structured into multiple layers, including Central GST (CGST), State GST (SGST), and Integrated GST (IGST), ensuring tax revenue is divided between central and state governments. Despite its benefits, the GST system faces multiple challenges. One of the primary concerns is ensuring **compliance**. Businesses are required to file accurate returns and make timely tax payments, but non-compliance, either intentional or due to errors, remains an issue. The complexity of GST filings often leads to mistakes in reporting, missed deadlines, and incorrect claims, which can result in substantial revenue losses. **Fraud detection** is another pressing concern, particularly with **input tax credit (ITC)** fraud, where businesses falsely claim ITC on the basis of fake invoices. This type of fraud is difficult to detect and results in massive tax evasion. Moreover, identifying suspicious patterns in the vast amount of GST return data requires advanced analytical tools. **Taxpayer segmentation** is equally crucial for efficient tax administration. It involves classifying taxpayers into risk categories based on their behaviour, industry, and compliance history. Manual methods of monitoring these taxpayers are inefficient and often overlook key indicators of non-compliance or fraud. Given the scale and complexity of the GST system, there is a critical need for **predictive models** that can automatically detect fraud, classify taxpayers, and identify anomalies. Machine learning models are particularly suited for these tasks, as they can analyse large datasets and uncover hidden patterns that might indicate fraud or non-compliance. These models can provide tax authorities with real-time insights, allowing them to optimize auditing efforts, allocate resources effectively, and take primitive actions to prevent revenue losses. By implementing predictive models such as **Random Forest** for taxpayer segmentation and **Isolation Forest** for anomaly detection, tax authorities can significantly improve the accuracy and efficiency of GST administration, ensuring higher compliance, reducing fraud, and optimizing tax collection processes.

1.2 Problem Statement

The current GST system faces significant challenges in ensuring compliance, detecting fraud, and efficiently segmenting taxpayers. Some of the key problems include:

1. **Fraud**: False claims of input tax credit (ITC), fake invoices, and deliberate tax evasion.
2. **Non-Compliance**: Businesses failing to file timely and accurate GST returns, leading to revenue losses.
3. **Inefficient Tax Collection**: Manual or rule-based methods of monitoring tax compliance often result in missed fraud cases and improper resource allocation.

These issues necessitate a predictive model that can automatically analyse taxpayer behaviour, detect fraud in real-time, and ensure more efficient monitoring of compliance. Traditional rule-based approaches are limited in their scope and scalability, making **machine learning models** critical for identifying patterns and anomalies in large GST datasets.

2. OBJECTIVES

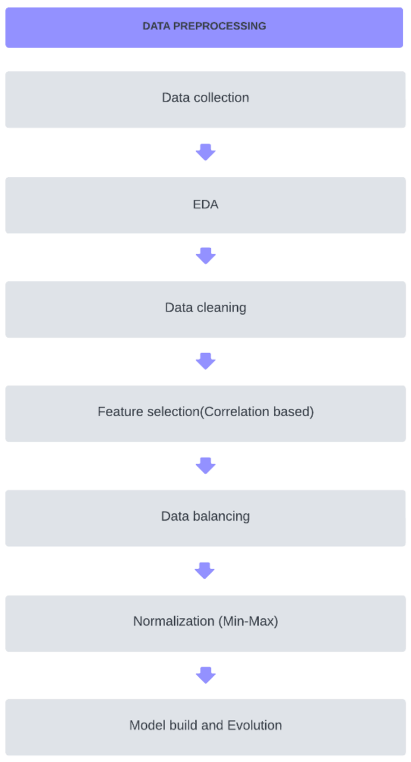
1. Fraud Detection and Prevention:
   - Develop a predictive system to identify fraudulent activities such as false ITC claims and fake invoicing by analysing historical and behavioural tax data.
2. Taxpayer Segmentation:
   - Classify taxpayers into risk-based categories using machine learning models, allowing authorities to prioritize high-risk cases for auditing and compliance checks.
3. Predictive Analysis for Tax Behaviour:

    - Forecast taxpayer behaviour and GST revenue trends to optimize tax collection and detect emerging risks in real-time.
4. Anomaly Detection and Compliance Monitoring:
    - Implement an anomaly detection model that can identify unusual patterns in tax returns, flagging potentially non-compliant taxpayers or those at risk of evasion, thus improving compliance monitoring efforts.

This project aims to enhance tax administration by developing a robust, data-driven framework that leverages machine learning to streamline GST compliance, detect fraud, and improve taxpayer segmentation.

3. METHODOLOGY

The methodology involves several steps from data collection to model training, testing, and evaluation. Here is a breakdown of the essential steps for your GST prediction project:



3.1 Data Preprocessing

3.1.1 Data Collection

The dataset used for the GST prediction system which is provided by the Government authorities are primarily includes taxpayer information, GST return filings, transaction data, and compliance records. These datasets are crucial for identifying patterns related to tax compliance, fraud detection, and anomaly detection.

3.1.2 EDA

EDA stands for Exploratory Data Analysis. It refers to the process of analysing datasets to summarize their main characteristics, often using visual methods. The primary purpose of EDA is to: Discover patterns, spot anomalies, check assumptions, and find relationships between variables. Identify irregularities that may affect analysis or model performance. Data visualizations, such as histograms, box plots, scatter plots, and correlation matrices, are essential tools used for visually exploring the relationships between variables and identifying patterns or anomalies. Additionally, data distribution checks are performed to assess the normality or skewness of the data, ensuring the dataset is well-suited for further analysis and modeling.

3.1.3 Data Cleaning

Loading the Dataset and Handling Missing Values

To begin with the data cleaning phase, the training dataset was loaded and thoroughly examined using summary statistics and data exploration techniques. This initial inspection aimed to check for any inconsistencies, missing values, and irregularities that could impact subsequent analysis. The dataset was checked for missing values using the is null() function. If any missing data was detected, those missing values were systematically replaced with a value of **0.** But in our dataset there is no missing values .

Assessing Data Distribution and Normality Testing

Once the dataset was checked, the next step was to evaluate whether the data followed a normal distribution. This is a crucial step in determining the appropriate statistical methods to use in the later stages of analysis. To assess normality, the following statistical tests were performed: The **Shapiro-Wilk Test** is a statistical test used to assess whether a given dataset is normally distributed. It tests the null hypothesis that the data is drawn from a normal distribution. It is widely used due to its high power for small sample sizes and its ability to detect deviations from normality in both tails of the distribution. This test was applied to determine whether the dataset significantly deviates from a normal distribution. The

D'Agostino's K-squared Test is a statistical test used to determine whether a dataset follows a normal distribution. Unlike the Shapiro-Wilk test, which assesses general normality, the D'Agostino's K-squared test specifically examines skewness and kurtosis, two key measures of the shape of a distribution. By doing this test we came to know that our data is highly positive skewed dataset. **Anderson-Darling Test**: This was employed to provide a more robust test for normality, especially for small samples. So, we have not used this process because our dataset is too large.

Identifying and Handling Skewed Columns

The dataset was then analysed for skewness , particularly focusing on columns that exhibited significant deviation from normality. Skewness refers to the asymmetry of the data distribution and can distort the results of many statistical models. Histograms were plotted for the skewed columns to visualize the degree of skewness. These plots provided insight into the nature of the data (positively or negatively skewed) and helped decide the appropriate transformation techniques to apply.

Outlier Detection Using Isolation Forest

Outliers, or extreme values, can severely impact the performance of predictive models. To identify outliers, the **Isolation Forest** algorithm was applied. The Isolation Forest algorithm flagged potential outliers by analyzing the dataset and isolating points that deviated significantly from the normal data points. The results of the outlier detection process were saved in a file named outliers.csv, allowing for further analysis and validation.

Outlier Detection Using the IQR Method

In addition to the Isolation Forest, the **Interquartile Range (IQR)** method was used to detect outliers. This statistical approach calculates the range between the first quartile (Q1) and third quartile (Q3) and defines any point outside 1.5 times the IQR as an outlier. Outliers were identified by calculating the lower bound as Q1 - 1.5 * IQR and the upper bound as Q3 + 1.5 * IQR. Data points falling outside these bounds were classified as outliers and marked for further processing.

Log Transformation for Outlier Detection

To handle skewness and reduce the impact of extreme values, **log transformation** was applied to the dataset. This transformation is effective in compressing the range of values and making the distribution more symmetric. **Hybrid Approach (Log Transformation + IQR)**: After applying log transformation, the IQR method was once again applied to detect outliers. This hybrid approach aimed to detect more subtle outliers that might not have been captured in the original scale of the data.

Outlier Removal and Replacement

Once the outliers were detected, the next step was to handle these outliers appropriately:

Replacing Outliers with the Median Value: For columns identified as highly skewed, the outliers were replaced with the **median** value of the respective column. This method ensures that the outlier values are not simply removed, but instead are replaced with a robust central tendency measure. **Replacement with 0 (Caution)**: In certain cases, the outliers were replaced with a value of **0**, but this was avoided for columns where 0 could introduce bias or was not appropriate for further analysis. **Hybrid Approach** :Using the **log transformation + IQR method**, outliers were replaced with the median value post-transformation, ensuring the dataset was free from extreme values that could distort model performance.

Verifying the Presence of Outliers in the Transformed Dataset

After performing the necessary transformations and outlier replacements, the IQR method was once again applied to the newly transformed dataset to ensure that no outliers remained.

Displaying Outlier Counts: The total number of outliers detected in each column was displayed. This step helps quantify the extent of outliers in the dataset, and the overall number of outliers detected in the entire dataset was recorded. **Boxplot Visualization**: To visually inspect the distribution and presence of outliers, **boxplots** were plotted for each column. Boxplots are an effective way to visualize the spread of data and the presence of outliers, as they clearly indicate points that fall outside the interquartile range.

Final Data Transformation

After all transformations and outlier handling steps, the final dataset was saved for model training and testing. This newly cleaned and processed data was free from missing values, skewness, and outliers, ensuring that it is well-suited for further analysis and machine learning.

3.1.4 **FEATURE SELECTION**

FEATURE IMPORTANCE

Feature importance refers to a score that indicates how useful or valuable each feature is in predicting the target variable in a given machine learning model. Models like decision trees, random forests, and gradient boosting machines can provide importance scores for features.

How Feature Importance Helps in Feature Selection?

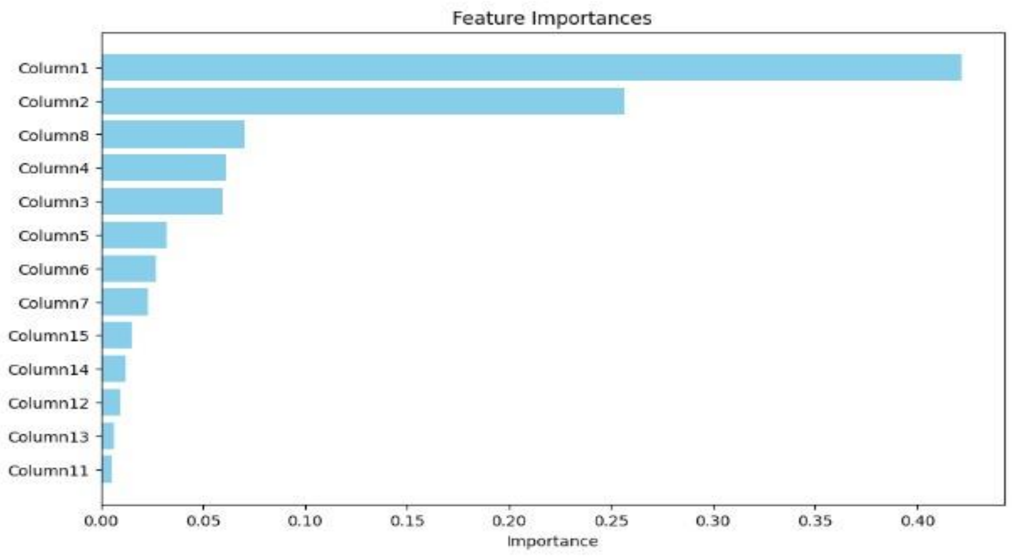
Feature importance can be used as a basis for feature selection:

1. Train a Model**:** You train a model that can compute feature importance (e.g., a random forest or XGBoost model).
2. Rank Features: The model assigns importance scores to features based on how much they contribute to the predictions.
3. Select the Most Important Features: You can select a subset of the most important features based on these scores. Features with very low importance scores can be discarded as they contribute little to the prediction.

Now in Feature Importance we have taken Random Forest Technique.

Random Forest

Feature importance in **Random Forest** is a powerful technique that helps to understand which features in your dataset are most influential in predicting the target variable. Random Forest is an ensemble of decision trees, and each decision tree provides a measure of how much a feature contributes to splitting the data and improving model accuracy. These contributions are then aggregated across all trees to give the overall feature importance score.



Now for Feature Selection we have used two techniques i.e., **VARIANCE THRESHOLD** and **RECURSIVE FEATURE ELIMINATION (RFE).**

Variance Threshold (Removing Low Variance Features)

The **Variance Threshold** is a feature selection technique that removes features with low variance. Features with little variance do not carry much useful information and may negatively affect the performance of the model by introducing noise. First, we will use the Variance Threshold function from the sklearn library to remove features that have variance below a certain threshold. This will remove low-variance features, retaining only the features with variance above the defined threshold.

Recursive Feature Elimination (RFE)

Recursive Feature Elimination (RFE) is a feature selection algorithm that works by recursively removing the least important features based on the model's weight (or coefficient) rankings. It trains the model multiple times, eliminating features in each iteration, and selects the most important ones. RFE can be applied with any machine learning model that has a way of ranking feature importance (e.g., logistic regression, decision trees, SVMs, etc.).

How RFE Improves Model Performance:

Elimination of Irrelevant Features

In many datasets, especially high-dimensional ones, not all features are informative or relevant to the target variable. Some features may be redundant, noisy, or even irrelevant, which can:

- **Increase the complexity of the model**: More features can lead to overfitting, where the model captures noise instead of true patterns.
- **Degrade performance**: Irrelevant features may confuse the model, leading to poorer predictive accuracy.

RFE helps by **efficiently eliminating irrelevant or less informative features**, ensuring that only the most valuable features are retained. This simplifies the model, reduces noise, and improves the overall quality of predictions.

Reduced Risk of Overfitting

Overfitting occurs when a model becomes too complex and starts learning the noise in the training data rather than the true underlying patterns. This results in poor generalization to new, unseen data.By removing non-informative or irrelevant features, RFE reduces the dimensionality of the dataset. Lower-dimensional models are simpler and less likely to overfit. RFE focuses the model on features that genuinely contribute to prediction, improving the model's ability to generalize to new data.

Enhanced Model Accuracy and Generalization

By focusing only on the most relevant features, RFE helps build models that perform better:

- **Accuracy**: When the noise and irrelevant features are removed, the model's predictive accuracy often increases because the model is no longer distracted by useless features.
- **Generalization**: A model that only includes important features is more likely to generalize well to new, unseen data. This means that it will maintain high performance across both training and testing sets.
-

Suitability for High-Dimensional Datasets

RFE is particularly suitable for high-dimensional datasets with many features because:

- In high-dimensional spaces, there is a higher likelihood of irrelevant or redundant features that could lead to overfitting.
- With datasets that have hundreds or thousands of features, manually selecting features becomes impractical. RFE automates this process, allowing the model to identify the most important features efficiently.

In high-dimensional datasets, RFE effectively reduces the dimensionality, making the machine learning model more efficient and faster to train.

Detecting the Number of Zeros in Each Column

To begin with, each column in the dataset needs to be evaluated to determine how many zero values are present. The presence of zeros can indicate missing or irrelevant data in certain cases, particularly in datasets where zeros do not hold any meaningful value.

Removing the Column with the Maximum Number of Zeros

After identifying the count of zeros, the next step is to remove the column that contains the maximum number of zeros. This is necessary because a column with too many zero values might distort the results of further analysis, such as correlation computation or machine learning. Removing columns with high zero counts helps in cleaning the data and removing noise that could affect the quality of correlation analysis or other feature-based analysis.

Specifying the List of Columns for Correlation Analysis

Once columns with excessive zeros have been removed, a subset of relevant columns is selected for correlation analysis. Correlation analysis helps identify relationships between features, which is crucial for tasks like feature selection, as highly correlated features may lead to redundancy in the dataset.

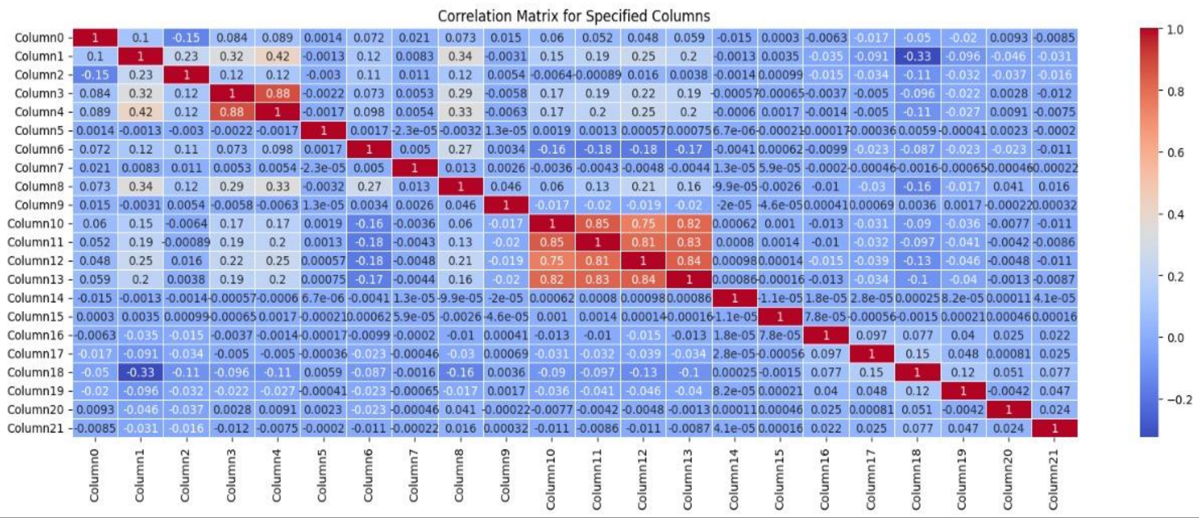
Calculating the Correlation Matrix for the Specified Columns

Now, compute the correlation matrix for the specified columns. The correlation matrix provides a measure of how strongly related two variables are, with values ranging from -1 to +1:

- **+1** indicates a strong positive correlation.
- **-1** indicates a strong negative correlation.
- **0** indicates no correlation.

Visualizing the Correlation Matrix

To make it easier to interpret, the correlation matrix is typically visualized using a heatmap. The color-coding of the heatmap provides a quick way to identify strongly correlated features. Use seaborn to create a heatmap that visualizes the correlation between the selected columns. Red shades will indicate strong positive correlations, blue shades will represent strong negative correlations, and lighter colors will show weak or no correlation. **Red Shades**: Indicate a strong positive correlation (closer to +1). This means that as one variable increases, the other also tends to increase. **Blue Shades**: Indicate a strong negative correlation (closer to -1). This means that as one variable increases, the other tends to decrease. **Lighter Colours (e.g., white)**: Indicate a weak or no correlation (close to 0), meaning the variables do not influence each other significantly.



	Column0	Column1	Column2	Column3	Column4	Column5	Column6	Column7	Column8	Column9	Column10	Column11	Column12	Column13	Column14	Column15	Column16	Column17	Column18	Column19	Column20	Column21
Column0	1	0.1	-0.15	0.084	0.089	0.0014	0.072	0.021	0.073	0.015	0.06	0.052	0.048	0.059	-0.015	0.0003	-0.0063	-0.017	-0.05	-0.02	0.0093	-0.0085
Column1	0.1	1	0.23	0.32	0.42	-0.0013	0.12	0.0083	0.34	-0.0031	0.15	0.19	0.25	0.2	-0.0013	0.0035	-0.035	-0.091	-0.33	-0.096	-0.046	-0.031
Column2	-0.15	0.23	1	0.12	0.12	-0.003	0.11	0.011	0.12	0.0054	-0.0064	-0.00089	0.016	0.0038	-0.0014	0.00099	-0.015	-0.034	-0.11	-0.032	-0.037	-0.016
Column3	0.084	0.32	0.12	1	0.88	-0.0022	0.073	0.0053	0.29	-0.0058	0.17	0.19	0.22	0.19	-0.00057	0.00065	-0.0037	-0.005	-0.096	-0.022	0.0028	-0.012
Column4	0.089	0.42	0.12	0.88	1	-0.0017	0.098	0.0054	0.33	-0.0063	0.17	0.2	0.25	0.2	-0.0006	0.0017	-0.0014	-0.005	-0.11	-0.027	0.0091	-0.0075
Column5	0.0014	-0.0013	-0.003	-0.0022	-0.0017	1	0.0017	-2.3e-05	-0.0032	1.3e-05	0.0019	0.0013	0.00057	0.00075	6.7e-06	-0.00021	-0.00017	-0.00036	0.0059	-0.00041	0.0023	-0.0002
Column6	0.072	0.12	0.11	0.073	0.098	0.0017	1	0.005	0.27	0.0034	-0.16	-0.18	-0.18	-0.17	-0.0041	0.00062	-0.0099	-0.023	-0.087	-0.023	-0.023	-0.011
Column7	0.021	0.0083	0.011	0.0053	0.0054	-2.3e-05	0.005	1	0.013	0.0026	-0.0036	-0.0043	-0.0048	-0.0044	1.3e-05	5.9e-05	-0.0002	-0.00046	-0.0016	-0.00065	-0.00046	-0.00022
Column8	0.073	0.34	0.12	0.29	0.33	-0.0032	0.27	0.013	1	0.046	0.06	0.13	0.21	0.16	-9.9e-05	-0.0026	-0.01	-0.03	-0.16	-0.017	0.041	0.016
Column9	0.015	-0.0031	0.0054	-0.0058	-0.0063	1.3e-05	0.0034	0.0026	0.046	1	-0.017	-0.02	-0.019	-0.02	-2e-05	-4.6e-05	0.00041	0.00069	0.0036	0.0017	-0.00022	0.00032
Column10	0.06	0.15	-0.0064	0.17	0.17	0.0019	-0.16	-0.0036	0.06	-0.017	1	0.85	0.75	0.82	0.00062	0.001	-0.013	-0.031	-0.09	-0.036	-0.0077	-0.011
Column11	0.052	0.19	-0.00089	0.19	0.2	0.0013	-0.18	-0.0043	0.13	-0.02	0.85	1	0.81	0.83	0.0008	0.0014	-0.01	-0.032	-0.097	-0.041	-0.0042	-0.0086
Column12	0.048	0.25	0.016	0.22	0.25	0.00057	-0.18	-0.0048	0.21	-0.019	0.75	0.81	1	0.84	0.00098	0.00014	-0.015	-0.039	-0.13	-0.046	-0.0048	-0.011
Column13	0.059	0.2	0.0038	0.19	0.2	0.00075	-0.17	-0.0044	0.16	-0.02	0.82	0.83	0.84	1	0.00086	-0.00016	-0.013	-0.034	-0.1	-0.04	-0.0013	-0.0087
Column14	-0.015	-0.0013	-0.0014	-0.00057	-0.0006	6.7e-06	-0.0041	1.3e-05	-9.9e-05	-2e-05	0.00062	0.0008	0.00098	0.00086	1	-1.1e-05	1.8e-05	2.8e-05	0.00025	8.2e-05	0.00011	4.1e-05
Column15	0.0003	0.0035	0.00099	0.00065	0.0017	-0.00021	0.00062	5.9e-05	-0.0026	-4.6e-05	0.001	0.0014	0.00014	-0.00016	-1.1e-05	1	7.8e-05	-0.00056	-0.0015	0.00021	0.00046	0.00016
Column16	-0.0063	-0.035	-0.015	-0.0037	-0.0014	-0.00017	-0.0099	-0.0002	-0.01	0.00041	-0.013	-0.01	-0.015	-0.013	1.8e-05	7.8e-05	1	0.097	0.077	0.04	0.025	0.022
Column17	-0.017	-0.091	-0.034	-0.005	-0.005	-0.00036	-0.023	-0.00046	-0.03	0.00069	-0.031	-0.032	-0.039	-0.034	2.8e-05	-0.00056	0.097	1	0.15	0.048	0.00081	0.025
Column18	-0.05	-0.33	-0.11	-0.096	-0.11	0.0059	-0.087	-0.0016	-0.16	0.0036	-0.09	-0.097	-0.13	-0.1	0.00025	-0.0015	0.077	0.15	1	0.12	0.051	0.077
Column19	-0.02	-0.096	-0.032	-0.022	-0.027	-0.00041	-0.023	-0.00065	-0.017	0.0017	-0.036	-0.041	-0.046	-0.04	8.2e-05	0.00021	0.04	0.048	0.12	1	-0.0042	0.047
Column20	0.0093	-0.046	-0.037	0.0028	0.0091	0.0023	-0.023	-0.00046	0.041	-0.00022	-0.0077	-0.0042	-0.0048	-0.0013	0.00011	0.00046	0.025	0.00081	0.051	-0.0042	1	0.024
Column21	-0.0085	-0.031	-0.016	-0.012	-0.0075	-0.0002	-0.011	-0.00022	0.016	0.00032	-0.011	-0.0086	-0.011	-0.0087	4.1e-05	0.00016	0.022	0.025	0.077	0.047	0.024	1

Identifying Strongly Correlated Features

Extracting Highly Correlated Pairs: From the correlation matrix, identify pairs of variables with absolute correlation values close to +1 or -1. These highly correlated features can be marked for further analysis or removed if they are redundant.

3.1.5 **Data Balancing**

Data balancing refers to techniques used to address the issue of imbalanced datasets, where the distribution of target classes is uneven. This imbalance can negatively impact machine learning models because most algorithms assume or perform better when classes are roughly equally represented. When a dataset is heavily imbalanced, the model may become biased toward the majority class and perform poorly on the minority class.

Why Data Balancing is Important?

When training models on imbalanced datasets, the model tends to predict the majority class more often, leading to high accuracy but poor performance on the minority class. This can lead to misleading model evaluation metrics (such as accuracy) because the model fails to correctly classify instances of the minority class. So, we have used SMOTE Technique for balancing the data. SMOTE (Synthetic Minority Over-sampling Technique) is a popular method to handle class imbalance:

- Oversamples minority class: Creates synthetic samples to balance classes.
- Preserves data distribution: Maintains original data characteristics.
- Reduces overfitting: Improves model generalization.

How SMOTE works?

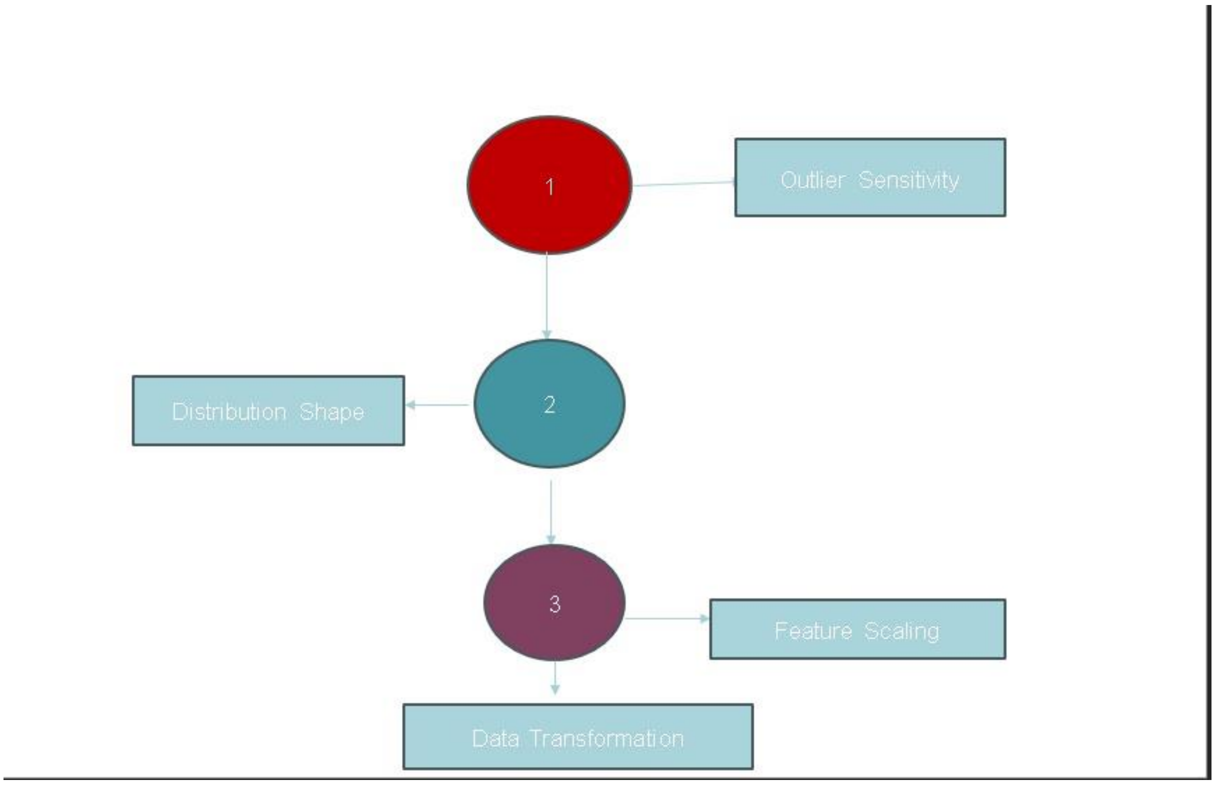
- Identify minority class: Determine the class with fewer instances.
- Select nearest neighbours: Find k-nearest neighbours for each minority class sample.
- Create synthetic samples: Interpolate between minority class samples and their neighbours.
- Balance classes: Repeat steps 2-3 until classes are balanced

Benefits of SMOTE

- Improves classification performance: Enhances accuracy and F1-score.
- Reduces bias: Mitigates majority class dominance.
- Handles high-dimensional data: Effective in feature-rich environments.

3.1.6 Data Normalization

Data normalization in machine learning is a process of converting all data features into a standardized format, ensuring that all variables in the data have a similar range. It is crucial in preparing the data for machine learning algorithms. Database normalization is significant in the data pre-processing phase as it improves the model performance by eliminating biases and inconsistencies in the data. The process speeds up model training, improves model convergence, and reduces the impact of outliers to avoid outcome biase.



Min-Max Scaling

This is a fundamental data normalization technique that is often simply called ‘normalization.’ This technique transforms features to a specific range, typically between 0 and 1. The formula for min-max scaling is:

Xnormalized = X – Xmin / Xmax – Xmin

Here,

X is a random feature value that needs to be normalized.

Xmin is the dataset’s minimum feature value and Xmax is the maximum.

When X is minimum, the numerator is 0 (Xmin- Xmin) the normalized value is 0.

When X is maximum, the numerator and denominator are equal (Xmax– Xmin) and the normalized value is 1.

When X is neither minimum nor maximum, the normalized value lies between 0 and 1.

4. Model Selection and Evolution

In model selection we have use different model to bring accuracy. Here are the models listed below which we have used: -

- Random Forest Classifier.
- Logistic Regression.
- Boosting.
- ANN
- Decission Tree

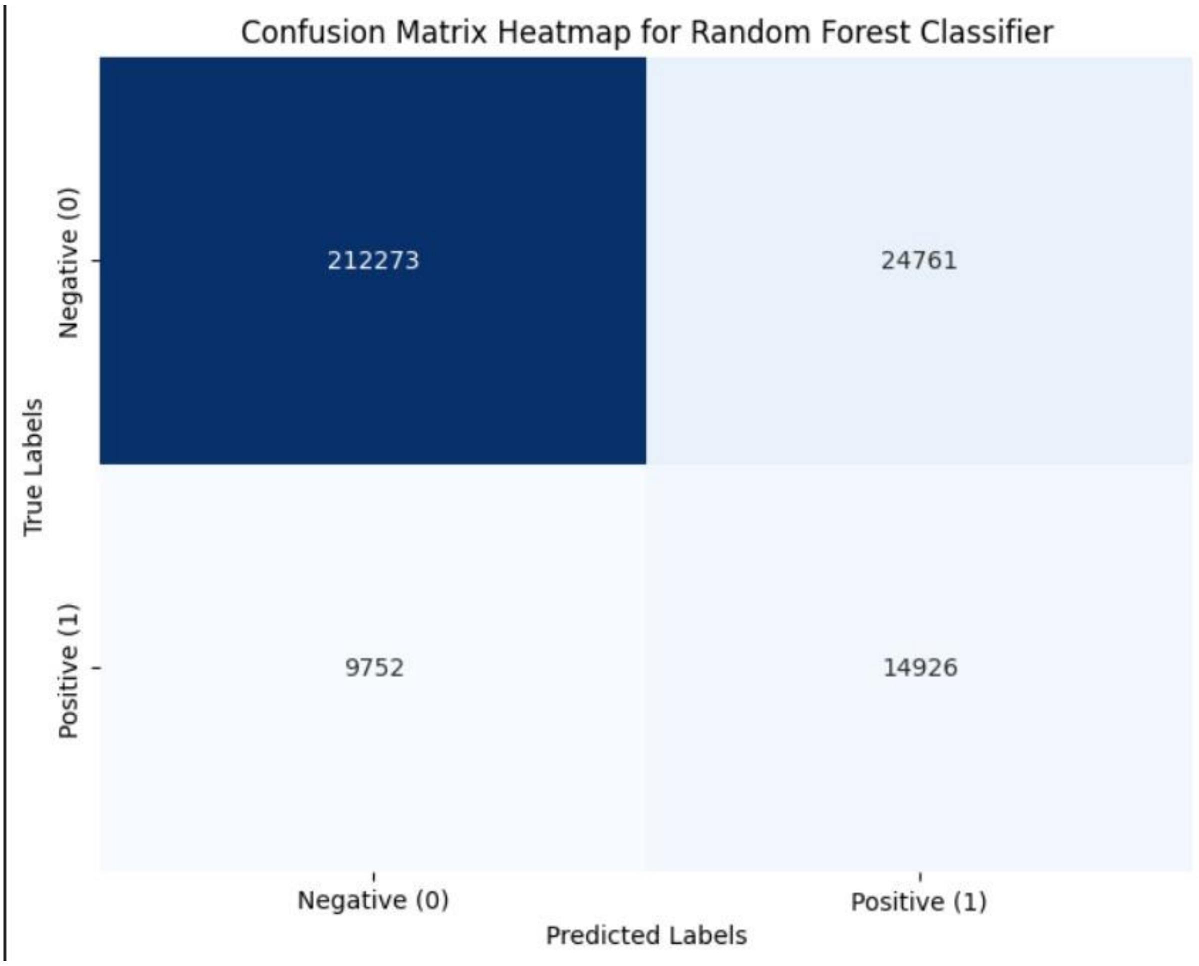
4.1 Random Forest Classifier

In this section, we provide a detailed evaluation of the performance of the **Random Forest** classifier on different dataset splits. We assess the model's behaviour when trained without splitting the dataset and when using various train-test splits (80-20, 70-30, and 60-40). The key performance metrics include **Accuracy**, **Precision**, **Recall**, **F1 Score**, and **AUC-ROC**.

4.1.1 ***Random Forest (Without Split)***

In the first experiment, the model was trained and tested on the entire dataset without performing any data split. This approach does not provide a true evaluation of the model's generalization ability, as there is no independent test set.

- **Accuracy**: 0.9135
- **Precision**: 0.5417
- **Recall**: 0.5374
- **F1 Score**: 0.5395
- **AUC-ROC**: 0.8966



Classification Report

	precision	recall	f1-score	support
0	**0.95**	**0.95**	**0.95**	**237034**
1	**0.54**	**0.54**	**0.54**	**24678**
accuracy			**0.91**	**261712**
macro avg	**0.75**	**0.75**	**0.75**	**261712**
weighted avg	**0.91**	**0.91**	**0.91**	**261712**

Analysis:

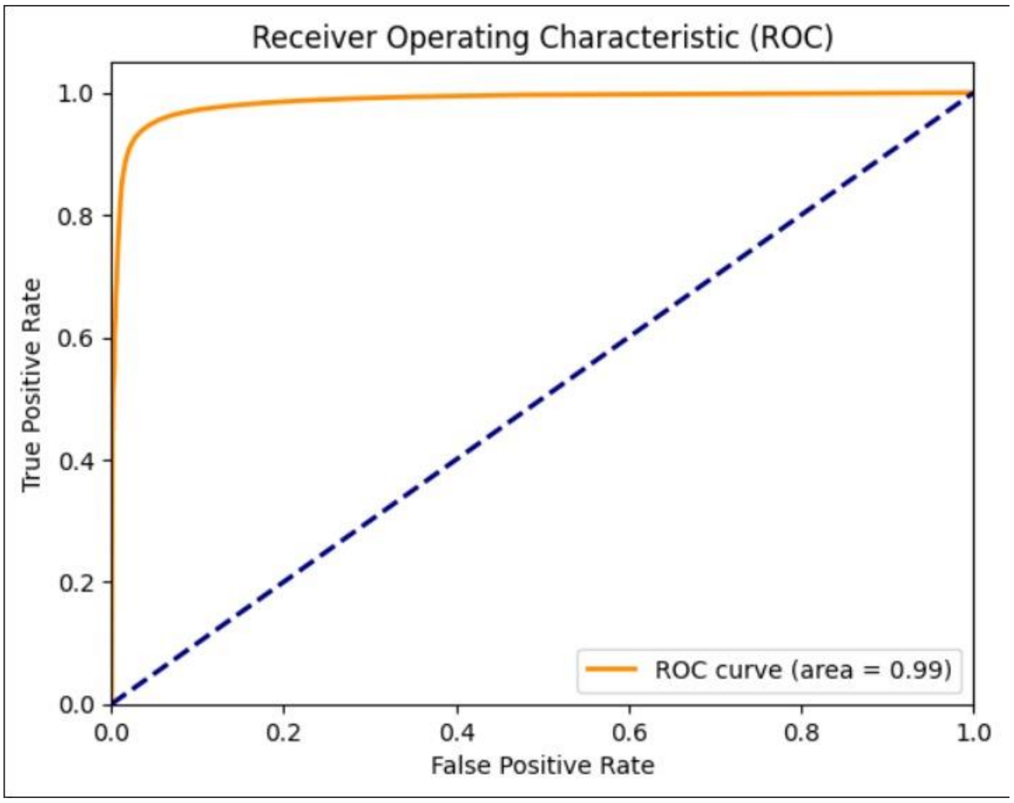
- The model achieved an overall accuracy of **91.35%**, which indicates high performance in predicting the majority class (class 0).
- However, the **precision** and **recall** for the minority class (class 1) are significantly lower, around **54%**, indicating that the model struggles to predict the minority class accurately.
- The **AUC-ROC** score of **0.8966** is relatively high, meaning the model has good discrimination between the classes, but the poor precision and recall suggest overfitting to the majority class.

While the overall accuracy is acceptable, the performance on the minority class is unsatisfactory, highlighting the need for a better validation strategy, such as splitting the dataset into training and test sets.

4.1.2 Random Forest (80-20 Split)

To better assess the model's generalization performance, we split the dataset into **80% training** and **20% testing** sets.

- **Accuracy**: 0.9515
- Precision: 0.9551
- **Recall**: 0.9479
- **F1 Score**: 0.9515
- **AUC-ROC**: 0.9861



Classification Report

	precision	recall	f1-score	support
0	**0.95**	**0.96**	**0.95**	**141872**
1	**0.96**	**0.95**	**0.95**	**142568**
accuracy			**0.95**	**284440**
macro avg	**0.95**	**0.95**	**0.95**	**284440**
weighted avg	**0.95**	**0.95**	**0.95**	**284440**

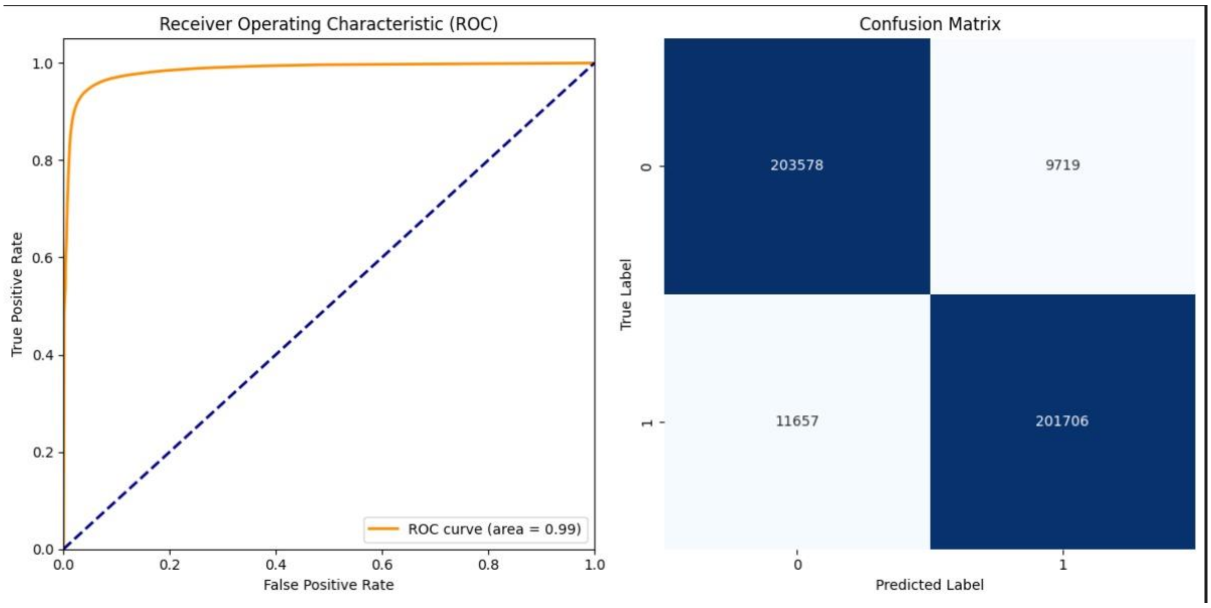
Analysis

- The accuracy improved to **95.15%**. The **precision** and **recall** are both around **95-96%**, showing that the model is able to correctly classify both the majority and minority classes more accurately.
- The **AUC-ROC** score of **0.9861** indicates excellent model performance, with minimal false positives and false negatives.

4.1.3 Random Forest (70-30 Split)

In this scenario, the dataset was split into **70% training** and **30% testing**.

- **Accuracy**: 0.9499
- Precision: 0.9540
- **Recall**: 0.9454
- **F1 Score**: 0.9497
- **AUC-ROC**: 0.9853



Classification Report

	precision	recall	f1-score	support
0	0.95	0.95	0.95	213297
1	0.95	0.95	0.95	213363
accuracy			0.95	426660
macro avg	0.95	0.95	0.95	426660
weighted avg	0.95	0.95	0.95	426660

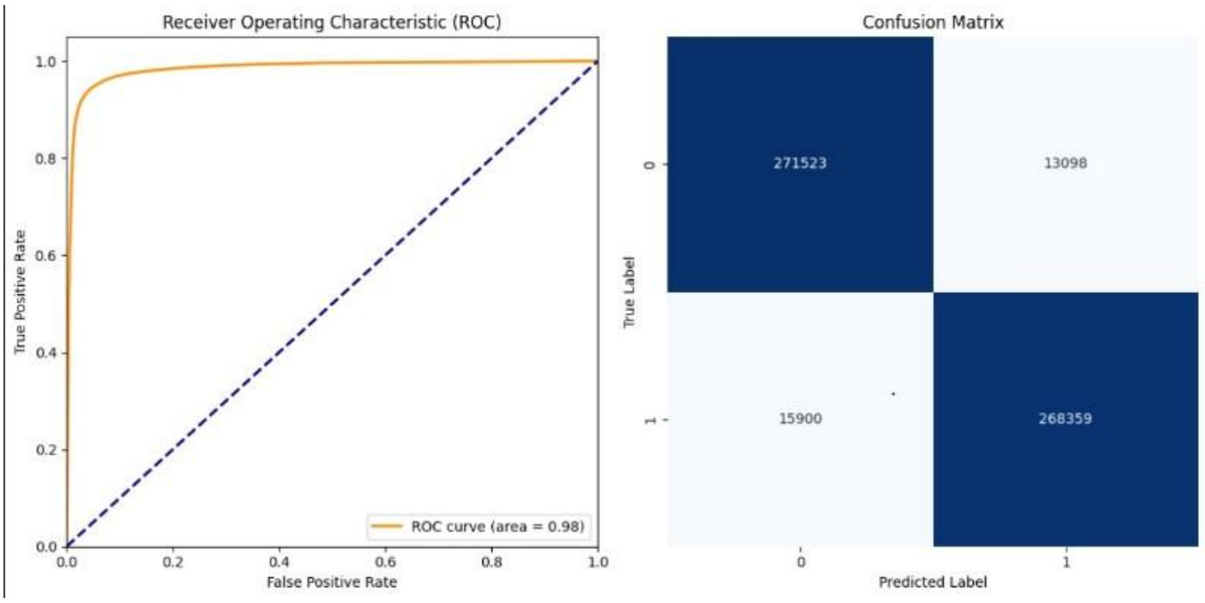
Analysis: -

- The model achieved an accuracy of **94.99%**, with **precision** and **recall** values close to **95%** for both classes.
- The **AUC-ROC** score of **0.9853** shows that the model can discriminate well between the classes, though it performs slightly lower than the 80-20 split.

4.1.4 Random Forest (60-40 Split)

In this case, the dataset was split into **60% training** and **40% testing**.

- **Accuracy**: 0.9490
- Precision: 0.9535
- **Recall**: 0.9441
- **F1 Score**: 0.9487
- **AUC-ROC**: 0.9843



Classification Report

	precision	recall	f1-score	support
0	0.94	0.95	0.95	284621
1	0.95	0.94	0.95	284259
accuracy			0.95	568880
macro avg	0.95	0.95	0.95	568880
weighted avg	0.95	0.95	0.95	568880

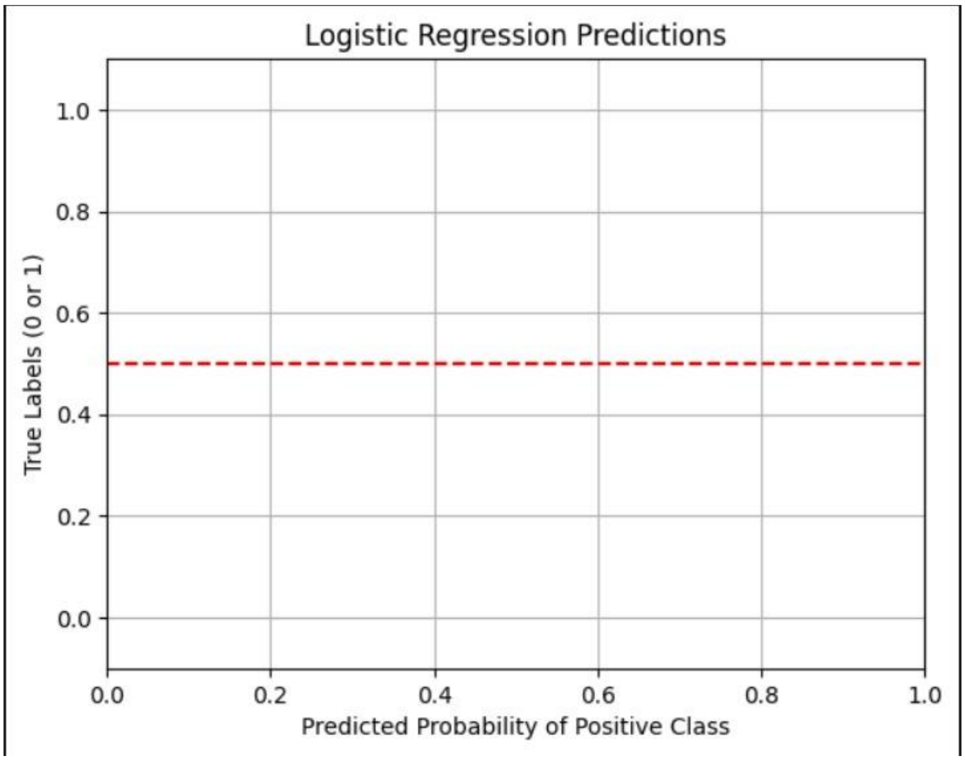
Analysis: -

- With **94.90%** accuracy, **precision**, and **recall** remaining at **94-95%**, the model's performance on this split is consistent with the previous splits.
- The **AUC-ROC** of **0.9843** remains high, but slightly lower than the other splits.

4.2 Logistic Regression

After evaluating the **Random Forest** model, we applied a **Logistic Regression** model to the dataset. Logistic Regression is a simpler, linear model that often serves as a baseline for binary classification tasks. Below are the performance metrics obtained from the Logistic Regression model:

- **Accuracy**: 0.6978
- Precision: 0.2257
- **Recall**: 0.9072
- **F1 Score**: 0.3615
- **AUC-ROC**: 0.8138



Classification Report: -

	precision	recall	f1-score	support
0	**0.99**	**0.68**	**0.80**	**237034**
1	**0.23**	**0.91**	**0.36**	**24678**
accuracy			**0.70**	**261712**
macro avg	**0.61**	**0.79**	**0.58**	**261712**
weighted avg	**0.91**	**0.70**	**0.76**	**261712**

Analysis: -

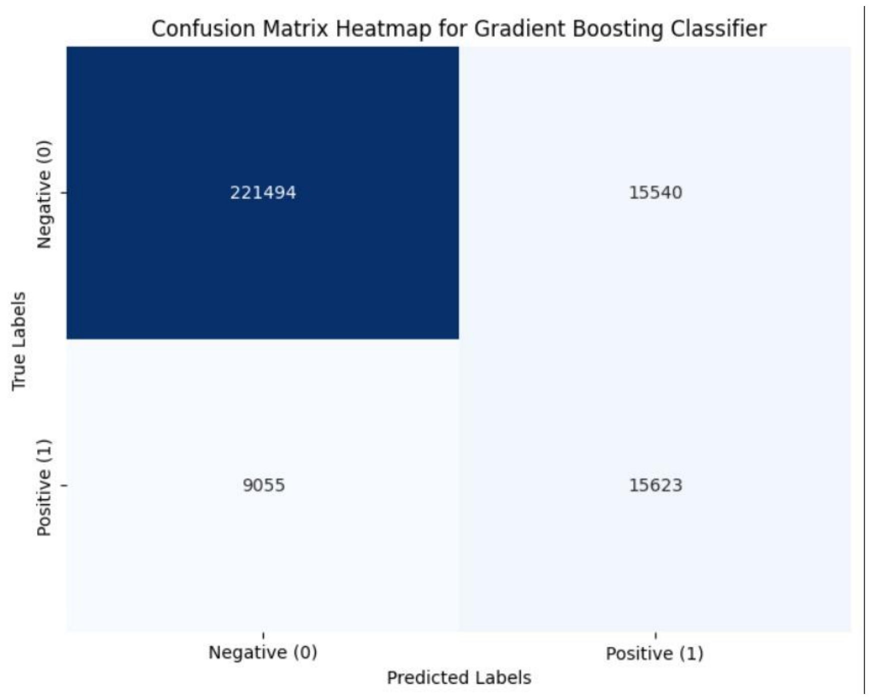
- The model achieved an overall **accuracy** of **69.78%**, which is lower compared to the Random Forest model.
- **Precision** for the minority class (class 1) is very low at **22.57%**, indicating that the model struggles with false positives.
- However, the **recall** for class 1 is significantly higher at **90.72%**, meaning the model identifies most of the actual positive cases. This high recall comes at the cost of precision, leading to a lower **F1 Score** of **0.3615**.
- The **AUC-ROC** of **0.8138** is moderately good, reflecting the model's ability to distinguish between the two classes, but it is lower compared to the Random Forest model.

4.3 Boosting

In addition to Random Forest and Logistic Regression, we applied a **Boosting Algorithm** to the dataset. Boosting is a powerful ensemble technique that combines multiple weak learners to create a strong classifier. Below are the results of the Boosting Algorithm applied

4.3.1 Without splitting the dataset:

- **Accuracy**: 0.9060
- Precision: 0.5013
- **Recall**: 0.6331
- **F1 Score**: 0.5596
- **AUC-ROC**: 0.9076



Classification Report

	precision	recall	f1-score	support
0	0.96	0.93	0.95	237034
1	0.50	0.63	0.56	24678
accuracy			0.91	261712
macro avg	0.73	0.78	0.75	261712
weighted avg	0.92	0.91	0.91	261712

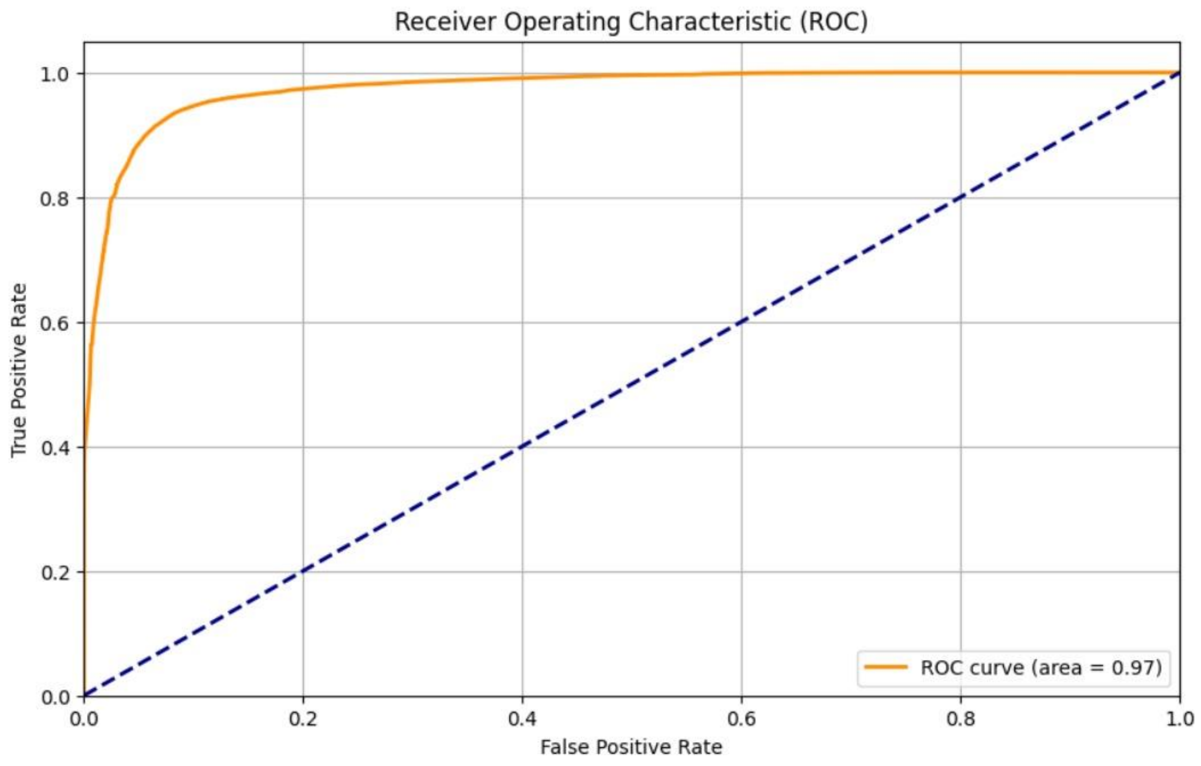
Analysis: -

- The model achieved an overall **accuracy** of **90.60%**, which is similar to the Random Forest model.
- The **precision** for the minority class (class 1) is **50.13%**, which is better than Logistic Regression but lower than Random Forest.
- The **recall** of **63.31%** for class 1 indicates that the model captures a moderate number of actual positive cases, but not as effectively as Logistic Regression.
- The **F1 Score** of **0.5596** represents a balance between precision and recall for class 1, and the **AUC-ROC** score of **0.9076** indicates strong model performance in distinguishing between the two classes.

4.3.2 With splitting the dataset:

In addition to applying the Boosting Algorithm without splitting the dataset, we also evaluated its performance after splitting the data into **80 % training and 20 % testing** . This allowed us to assess the model's generalization ability more effectively.

- **Accuracy**: 0.9256
- Precision: 0.9237
- **Recall**: 0.9283
- **F1 Score**: 0.9260
- **AUC-ROC**: 0.9748



Classification Report: -

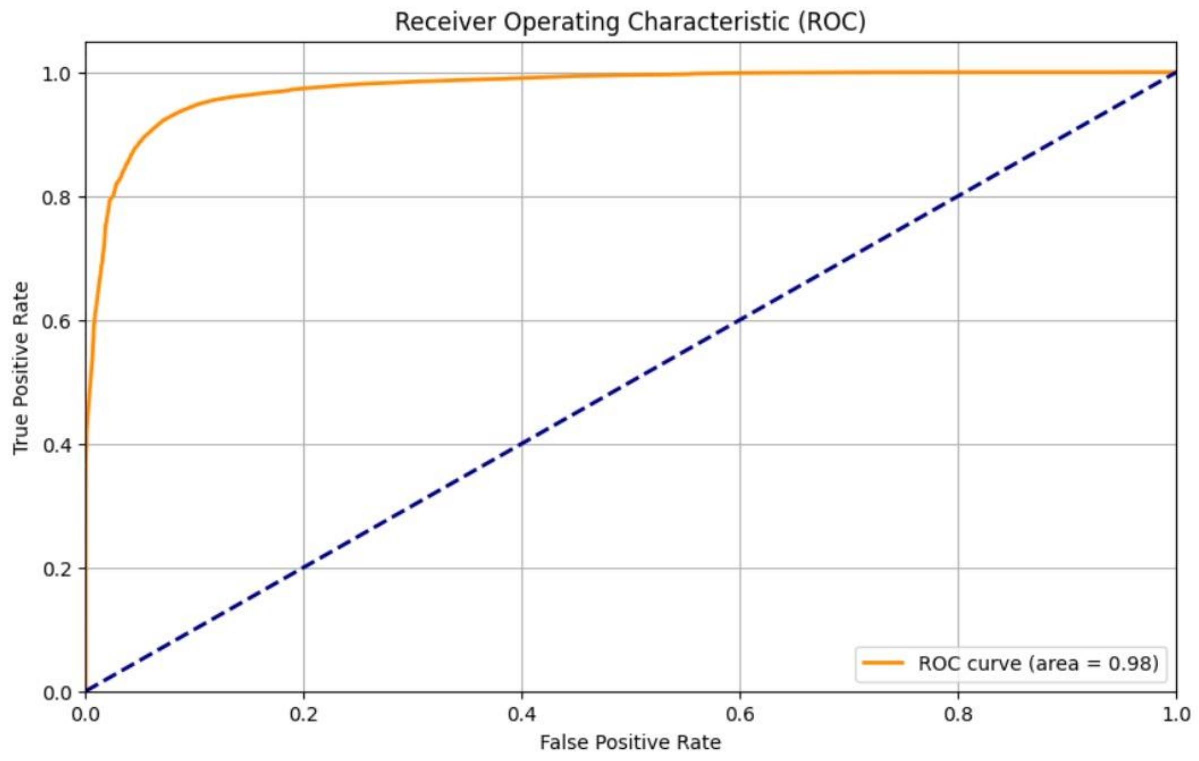
	precision	recall	f1-score	support
0	0.93	0.92	0.93	141872
1	0.92	0.93	0.93	142568
accuracy			0.93	284440
macro avg	0.93	0.93	0.93	284440
weighted avg	0.93	0.93	0.93	284440

Analysis: -

- With an **accuracy** of **92.56%**, the model performs robustly in both training and testing phases.
- The **precision** of **92.37%** and **recall** of **92.83%** demonstrate that the Boosting Algorithm maintains a balance between identifying true positives and avoiding false positives.
- The **F1 Score** of **0.9260** suggests a strong harmony between precision and recall.
- The **AUC-ROC** of **0.9748** reflects the high capability of the model to distinguish between classes.

To further evaluate the Boosting Algorithm, we also split the dataset into **70% training and 30 % testing**. This provides a comparison against the 80% training and 20% testing and assesses how the model performs with different proportions of training and testing data.

- **Accuracy**: 0.9254
- Precision: 0.9236
- **Recall**: 0.9275
- **F1 Score**: 0.9255
- **AUC-ROC**: 0.9753



Classification Report: -

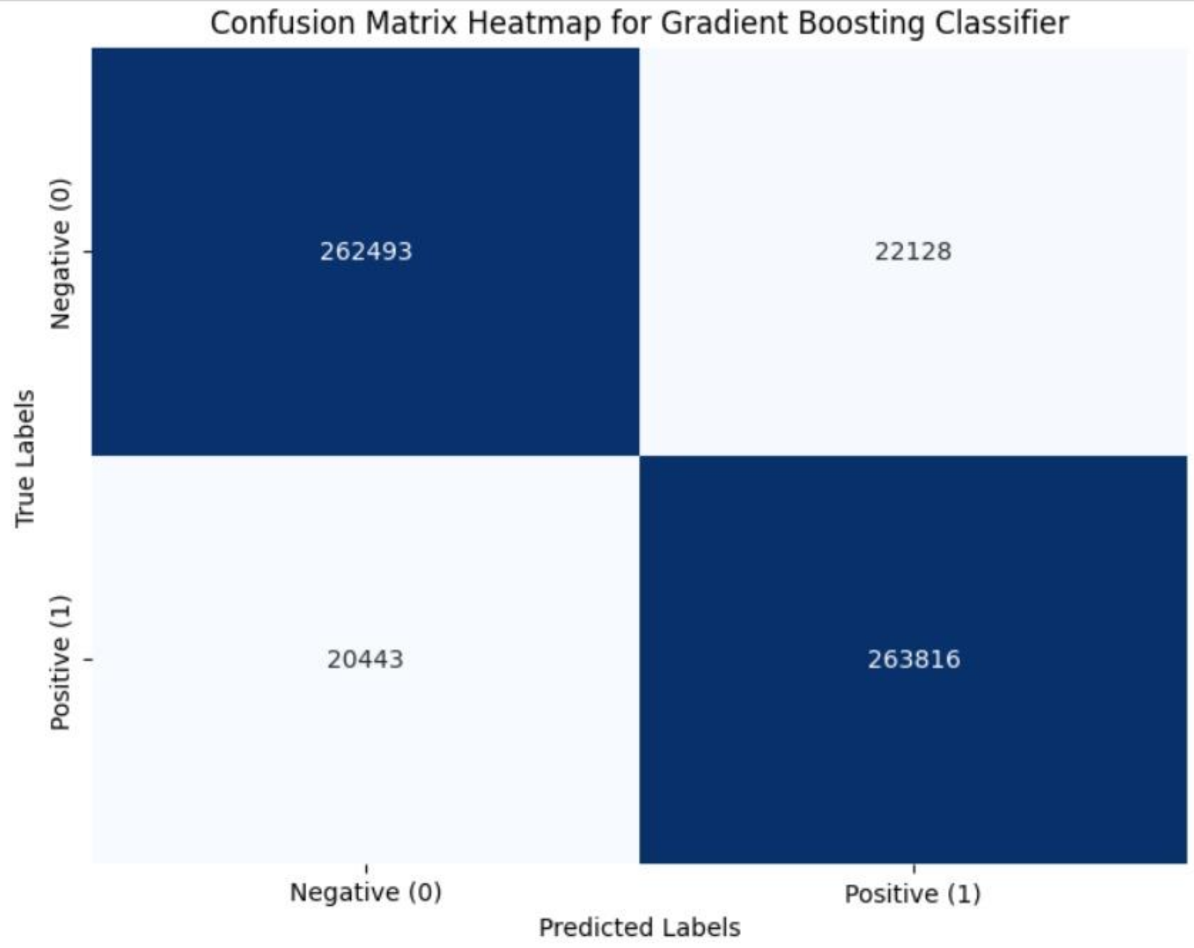
	precision	recall	f1-score	support
0	0.93	0.92	0.93	213297
1	0.92	0.93	0.93	213363
accuracy			0.93	426660
macro avg	0.93	0.93	0.93	426660
weighted avg	0.93	0.93	0.93	426660

Analysis: -

- The model achieved an **accuracy** of **92.54%**, which is consistent with the results of the 80-20 split.
- With **precision** at **92.36%** and **recall** at **92.75%**, the model effectively handles both false positives and false negatives.
- The **F1 Score** of **0.9255** continues to reflect a balanced trade-off between precision and recall.
- The **AUC-ROC** score of **0.9753** indicates strong discriminatory power, similar to the 80% training and 20% testing.

We further evaluated the performance of the Boosting Algorithm by splitting the dataset into **60% training and 40% testing**. This allows us to explore the model's behavior under yet another train-test split configuration.

- **Accuracy**: 0.9252
- Precision: 0.9226
- **Recall**: 0.9281
- **F1 Score**: 0.9253
- **AUC-ROC**: 0.9751



Classification Report

	precision	recall	f1-score	support
0	0.93	0.92	0.92	284621
1	0.92	0.93	0.93	284259
accuracy			0.93	568880
macro avg	0.93	0.93	0.93	568880
weighted avg	0.93	0.93	0.93	568880

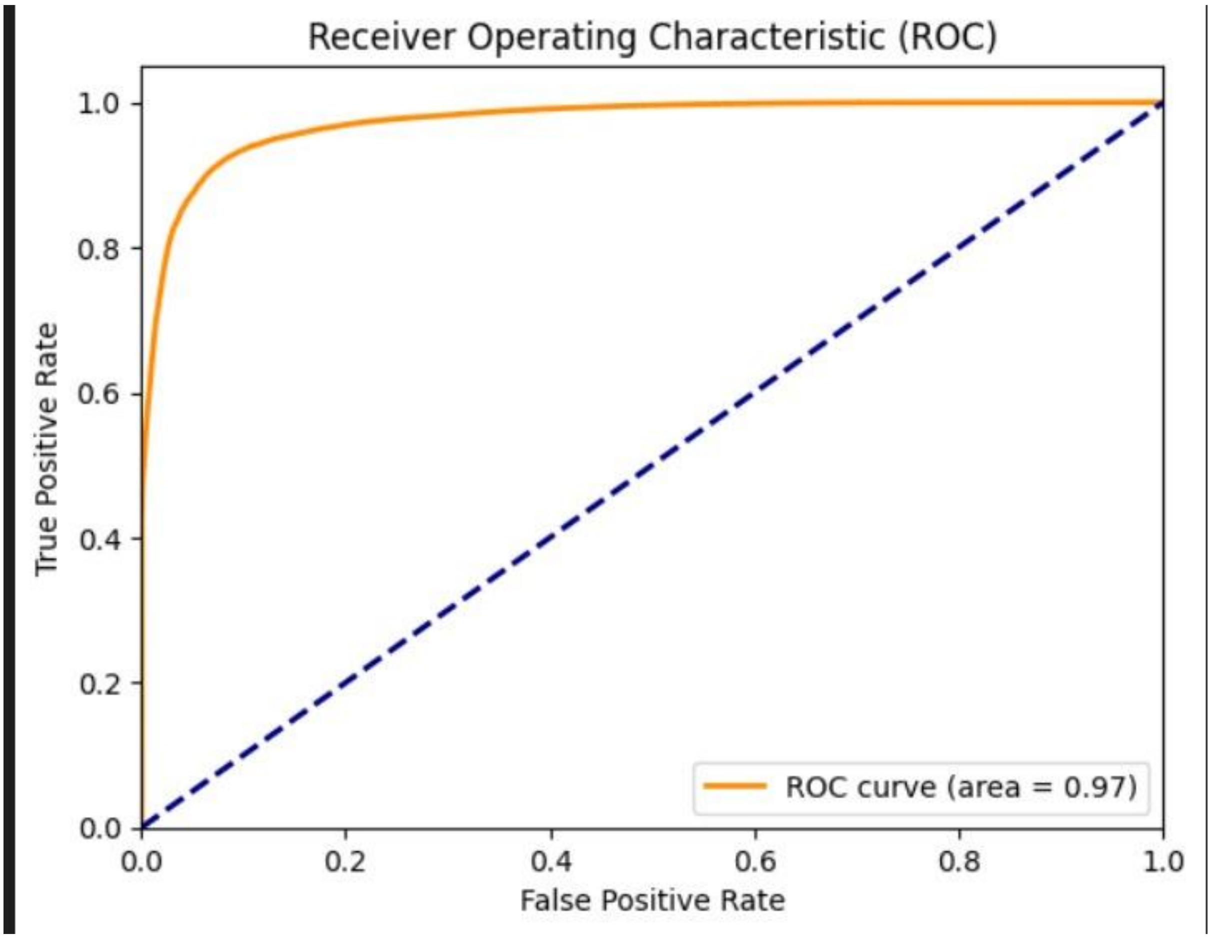
Analysis: -

- The **accuracy** of **92.52%** is consistent with previous splits, indicating the model's stability.
- With a **precision** of **92.26%** and **recall** of **92.81%**, the model balances the detection of true positives while maintaining a low false-positive rate.
- The **F1 Score** of **0.9253** confirms the model's reliability in both precision and recall.
- The **AUC-ROC** score of **0.9751** is in line with previous results, demonstrating strong class separability.

4.4 ANN

To further explore model performance, we trained an **Artificial Neural Network (ANN)** on the dataset. This deep learning approach provides a non-linear, hierarchical model that can capture complex patterns in the data.

- **Accuracy**: 0.9193
- Precision: 0.9144
- **Recall**: 0.9257
- **F1 Score**: 0.9200
- **AUC-ROC**: 0.9739



Classification Report: -

	precision	recall	f1-score	support
0	0.92	0.91	0.92	141872
1	0.91	0.93	0.92	142568
accuracy			0.92	284440
macro avg	0.92	0.92	0.92	28444
weighted avg	0.92	0.92	0.92	284440

Analysis: -

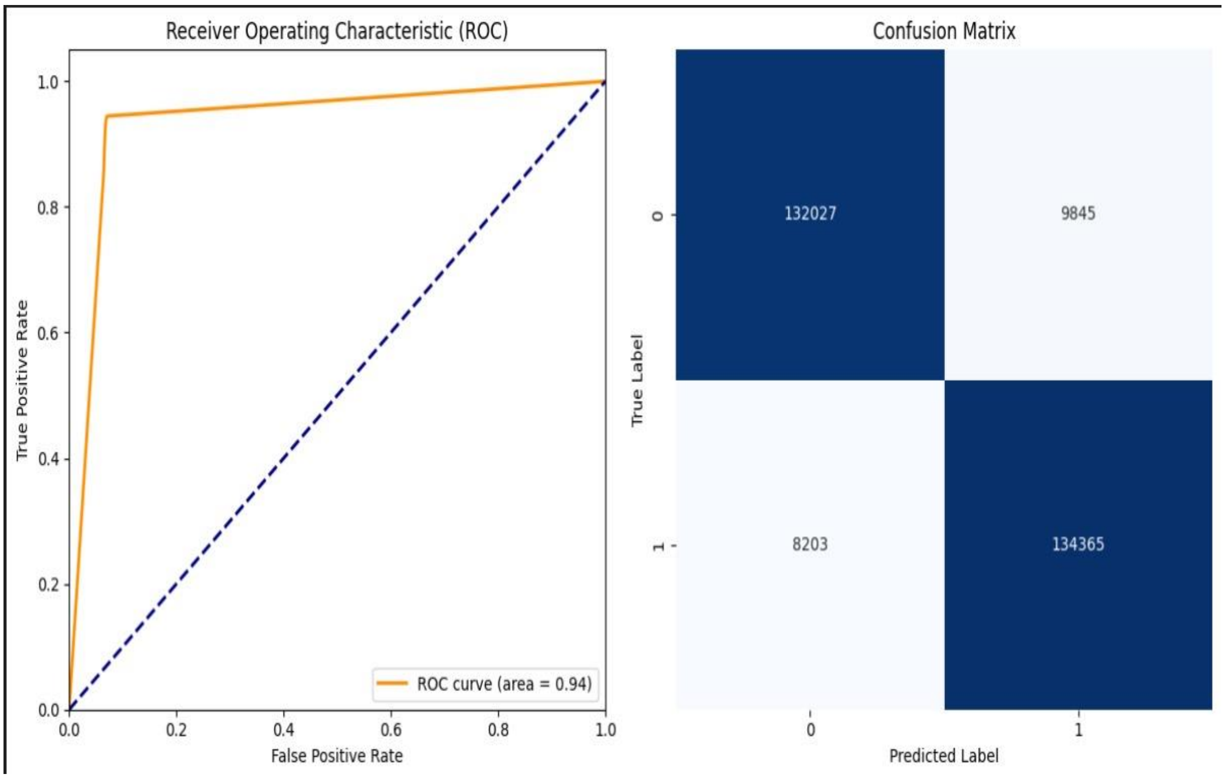
- The **accuracy** of **91.93%** reflects the ANN's capacity to perform well on both training and test data.
- A **precision** of **91.44%** and a **recall** of **92.57%** demonstrate that the model effectively handles both false positives and false negatives.

- The **F1 Score** of **0.9200** strikes a balance between precision and recall, ensuring reliability in prediction.
- The **AUC-ROC** score of **0.9739** indicates that the model distinguishes between classes with high accuracy.

4.5 Decission Tree

To further explore the model's performance, we applied the **Decision Tree algorithm** with an **80-20 data split**. Decision Trees provide a powerful method for classification, capturing non-linear decision boundaries through recursive partitioning.

- **Accuracy**: 0.9365
- Precision: 0.9317
- **Recall**: 0.9425
- **F1 Score**: 0.9371
- **AUC-ROC**: 0.9367



Classification Report: -

	precision	recall	f1-score	support
0	0.94	0.93	0.94	141872
1	0.93	0.94	0.94	142568
accuracy			0.94	284440
macro avg	0.94	0.94	0.94	28444
weighted avg	0.94	0.94	0.94	284440

Analysis: -

- The model achieved an **accuracy** of **93.65%**, which is highly competitive compared to other models.
- With a **precision** of **93.17%** and **recall** of **94.25%**, the Decision Tree handles both false positives and false negatives effectively.
- The **F1 Score** of **0.9371** confirms a good balance between precision and recall.
- The **AUC-ROC** score of **0.9367** indicates a strong ability to distinguish between the classes.

4.6 Hyper Tuning and Cross-Validation

To optimize model performance and prevent overfitting, k-fold cross-validation (k=5) was employed for all models. Additionally, Grid Search was utilized to perform hyperparameter tuning for each model. Specifically, the following models were evaluated:

- Random Forest: Tuned parameters included n_estimators, max_depth, and min_samples_split.
- Decision Tree: Tuned parameters included max_depth, min_samples_split, and criterion.
- Artificial Neural Network (ANN): Tuned parameters included hidden_layer_sizes, activation, and learning_rate_init.
- Logistic Regression: Tuned parameters included penalty, C, and max_iter.
- Boosting: Tuned parameters included learning_rate, n_estimators, and max_depth.

The optimal hyperparameters were selected based on the highest average cross-validation accuracy. This rigorous approach ensured robust and generalizable models.

- k-fold cross-validation
- Grid Search
- Hyperparameter tuning

- Model optimization
- Overfitting prevention
- Robust and generalizeable models
- Comparative analysis of models

N.B: - When we execute the code the output is partially accurate.

5. Comparing all the Models

Model Comparison

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**AUC-ROC**
Random Forest (No Split)	0.9135	0.5417	0.5374	0.5395	0.8966
Random Forest (80-20 Split)	0.9515	0.9551	0.9479	0.9515	0.9861
Random Forest (70-30 Split)	0.9499	0.9540	0.9454	0.9497	0.9853
Random Forest (60-40 Split)	0.9490	0.9535	0.9441	0.9487	0.9843
Logistic Regression	0.6978	0.2257	0.9072	0.3615	0.8138
Boosting (No Split)	0.9060	0.5013	0.6331	0.5596	0.9076
Boosting (80-20 Split)	0.9256	0.9237	0.9283	0.9260	0.9748
Boosting (70-30 Split)	0.9254	0.9236	0.9275	0.9255	0.9753
Boosting (60-40 Split)	0.9252	0.9226	0.9281	0.9253	0.9751
Artificial Neural Network (ANN)	0.9193	0.9144	0.9257	0.9200	0.9739
Decision Tree (80-20 Split)	0.9365	0.9317	0.9425	0.9371	0.9367

Based on the evaluation of various models, the **Random Forest with an 80-20 split** outperformed the others in terms of overall performance:

- **Accuracy**: 95.15%

- Precision: 95.51%
- **Recall**: 94.79%
- **F1 Score**: 95.15%
- **AUC-ROC**: 98.61%

While other models, such as **Boosting** and **ANN**, also performed well, the Random Forest with an 80-20 split had the highest combination of precision, recall, and AUC-ROC, indicating its superior ability to handle both false positives and false negatives while maintaining excellent class separability.

Therefore, **Random Forest with an 80-20 split** is recommended as the best model for this project due to its consistent and high performance across all key metrics.

6. Conclusion

In conclusion, the **Random Forest model** with the 80-20 data split provided the best overall performance for predicting GST compliance. This model is well-suited for use in GST prediction tasks due to its high accuracy and robustness in handling large datasets. It can serve as a reliable tool for regulatory authorities to detect potential tax defaulters efficiently and accurately. Implementing this model into real-world systems could enhance tax compliance monitoring, contributing to better enforcement and collection of GST.

7.Future Scope

The implementation of time series forecasting for GST fraud detection, anomaly detection, and taxpayer segmentation has shown promising results. To further enhance the effectiveness of this system, the following areas can be explored:

Short-term

- Integration with additional data sources: Incorporate data from other relevant sources, such as financial statements, tax returns, and external databases.
- Real-time data processing: Implement real-time data processing to enable timely fraud detection and prevention.
- Model refinement: Continuously monitor and refine the time series forecasting models to adapt to changing patterns and trends.
- Visualization dashboard: Develop an interactive visualization dashboard for stakeholders to easily interpret results.

Mid-term

- Deep learning techniques: Explore the application of deep learning techniques, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs).
- Transfer learning: Investigate the use of pre-trained models and transfer learning to improve model performance.
- Explainable AI (XAI): Implement XAI techniques to provide insights into model decision-making.
- Collaboration with other agencies: Collaborate with other tax authorities and law enforcement agencies to share knowledge and best practices.

Long-term

- Graph-based anomaly detection: Explore graph-based anomaly detection methods to identify complex relationships between entities.
- Predictive maintenance: Develop predictive maintenance models to forecast potential system failures and optimize resource allocation.
- Natural Language Processing (NLP): Integrate NLP techniques to analyze unstructured data, such as tax returns and financial statements.
- Blockchain integration: Investigate the use of blockchain technology to ensure data integrity and security.

Potential Research Directions

- Anomaly detection in multivariate time series
- Explainable AI for tax fraud detection
- Graph-based methods for taxpayer segmentation
- Deep learning for GST forecasting