https://dx.doi.org/10.4314/dujopas.v11i1a.31
ISSN (Print): 2476-8316
ISSN (Online): 2635-3490
Dutse Journal of Pure and Applied Sciences (DUJOPAS), *Vol. 11 No. 1a March 2025*

Identifying Financial Fraud Transactions Using Decision Tree Classifier Algorithm

Ayoade Akeem Owoade

Department of Computer Science
Tai Solarin University of Education,
Ijagun,
Ijebu Ode,
Nigeria.

Email: owoadeaa@tasued.edu.ng

Abstract

Fraud in financial transactions is a critical issue for businesses, governments, and consumers, causing substantial financial losses and eroding trust in financial systems. Rule-based systems and other conventional fraud detection techniques have trouble identifying complex fraudulent activity. This research applies various machine learning (ML) algorithms to detect fraud in financial transactions. The research compares the performance of supervised ML techniques, such as random forest, logistic regression, and decision tree classifier*, using a publicly available Kaggle dataset. It evaluates the models based on accuracy, precision, recall, and F1 scores. To address data imbalance in both undersampling and oversampling techniques, with a focus on oversampling techniques such as Synthetic Minority Over-Sampling Method (SMOTE) to enhance model performance. Results show that the Decision Tree Classifier, when used with oversampling, outperforms other models, achieving a 99% accuracy in detecting fraudulent transactions. This highlights the effectiveness of using ensemble and hybrid models in combination with oversampling* to enhance the identification of fraud. *The findings emphasize the importance of using advanced ML techniques and robust data preprocessing for* detection of financial fraud.
.

Keywords: Fraud Detection, Machine learning, Financial Transaction, Sampling, Oversampling

INTRODUCTION

In the ever-evolving financial world, fraudulent activities pose significant risks to businesses, governments, and individuals. These illegal practices not only result in huge financial losses but also erode confidence in the economy and threaten its stability. As digital transactions continue to proliferate, there is a need to develop effective and accurate methods to detect fraudulent activity to protect the integrity of financial systems. The advent of machine learning (ML) techniques has revolutionized fraud detection by providing sophisticated tools to identify and analyze complex patterns in financial transactions ML algorithms have demonstrated the skills that they are surprisingly demonstrated in modifying the evolving fraudulent methods. It is to examine its effectiveness in detecting fraud in financial transactions. Considering the pressing demand for effective and efficient fraud detection systems, the purpose of this study is to create an artificial intelligence-based approach to detect fraudulent in financial transactions by exploring advanced ML techniques possible

**Author for Correspondence*

hybrid methods and ways to overcome the semantic problems associated with data imbalance. This research minimizes fraud detection to reduce economic loss to rebuilding consumer confidence in financial services.

In recent years, financial fraud has become an increasingly prevalent issue, posing significant challenges to financial services, government and private customers. With the rapid development of digital technology and online transactions, fraudulent activity has continued to intensify, negatively affecting the economy and undermining consumers' trust in the economy (Chou et al., 2020). Conventional techniques for detecting fraud like rule-based systems and manual inspections, frequently do not detect complex and well-developed fraudulent behaviors (Ngai et al., 2011). Machine learning (ML) models have shown promising results in addressing the limitations of traditional fraud detection methods through complex pattern recognition, modification of evolving fraud techniques, and large amounts of data implementation control (Kou et al., 2021). However, in fraud-identification, ML is a suitable technique used. Traditionally, rule-based measures have been used to combat fraud in the financial sector, however, this system is inherently rigid and struggle to keep up with the active role of deception. Fraudulent individuals are always coming up with new ways to avoid detection, constructing conventional rule-based systems inadequate. In this context, machine learning has emerged as a beacon of hope due to its ability to identify complex patterns in large datasets. ML. This study seeks to look into the potential of ML methods in financial deception and fraud detection.

Several empirical research has shown the effectiveness of ML techniques in detecting fraudulent financial transactions (Sánchez et al. 2020) employed a hybrid approach integrating supervised and unsupervised learning to detect credit card fraud, achieving an F1 score of 0.991. Fu et al. 2020 suggested a framework based on deep learning, FD-GAN for financial fraud detection and outperformed traditional methods. In another study, Pumsirirat and Nukoolkit (2020) developed a real-time model for detecting credit card fraud using a one-class OC-SVM, or support vector machine and attained an accuracy of 99.75%.

Tsai et al. 2018 developed an ensemble model integrating Artificial neural networks, decision trees, and logistic regression, achieving better performance than individual models. The following review highlights key empirical findings from recent studies applying ML methods to address fraud detection challenges.

Sánchez et al. 2021 proposed a hybrid approach integrating supervised and unsupervised learning for credit card fraud detection, achieving an F1 score of 0.991. Fu et al. 2020 introduced FD-GAN, a deep learning-based framework for financial fraud detection, outperforming traditional ML methods.

In order to detect fraudulent financial transactions Tsai et al., 2018 created an ensemble model that combined Artificial neural networks, logistic regression, and decision trees outperforming individual models. Credit card fraud and other forms of cybercrime are becoming more and more of a worry since they may cause serious harm to companies of all kinds. The main aim of fraud is the actual card transaction; in recent instances, stolen data has been exploited. Automated fraud detection techniques are essential for safeguarding users' online safety and thwarting thieves. For many applications, machine learning has emerged as the industry standard; nonetheless, the quality of training data determines how well it performs (Strelcenia et al., 2023).

Financial fraud is the illegal and fraudulent use of money to gain financial benefits in various sectors, including insurance, banking, taxation, and corporate sectors. Despite efforts to reduce fraud, it continues to negatively impact the economy and society. Conventional techniques, such as manual detection, are expensive, time-consuming, and inaccurate. Machine learning and data mining are being used to identify fraudulent activity in the financial sector because to advancements in artificial intelligence (AI). The most often used techniques for identifying financial fraud are classification approaches. However, recent increases in fraud activities in health sectors have led to a need for more comprehensive approaches (Ali et al 2022).

Several important ideas need to be taken into account in order to use machine learning techniques to identify financial transaction fraud. To improve model performance, data preprocessing is essential, including feature selection and data balancing (Eryu, 2024). ML models' prediction capabilities can be further enhanced by ensemble learning and model stacking strategies (Kou et al., 2021).

METHODOLOGY

By analyzing large and varied datasets, the primary objective of this study is to determine and assess which machine learning methods are most effective in detecting fraudulent activity. Data gathering, data preprocessing, feature and model selection, training, validation, and assessment are some of the phases that make up this study. The target population for this study is the historical financial transaction on credit card, the source of the data set is kaggle.com, which is an online opensource platform to get dataset to train machine learning models. The sampling procedure involve randomly selecting a subset of the credit card transactions from the Kaggle dataset, with the goal of obtaining a sample size of approximately 10,000 or more transactions. A training set and a test set is created from the sample, with roughly 8,000 transactions (or 80% of the chosen data set) in the training set and 2,000 transactions (20% of the chosen data set) in the test set.

Method of data collection

The data for this work is obtained from the Kaggle dataset, which contains historical credit card transactions labeled as either fraudulent or legitimate. All data is sourced from publicly available datasets, and strict conformity to ethical principles and data laws pertaining to confidentiality of data is ensured throughout the data collection process.

Procedure for data analysis

The Logistic Regression technique, a method for supervised machine learning suitable for classification applications is used to examine the data. The following steps are included in the data analysis:

i. Importing necessary libraries
ii. Data Preprocessing: Raw data is subjected to a series of cleaning, transformation, and normalization procedures to ensure data quality and consistency. Missing values is imputed, and outliers is identified and handled accordingly.
iii. Model Selection: A range of machine learning algorithms, specifically logistic regression, is considered for the development of predictive models. The choice of algorithm is guided by its performance in similar applications and its suitability for processing the size and complexity of the dataset.
iv. Model Training and Validation: To evaluate the chosen model's performance, a subset of the dataset is used for training, and another subset is used for validation.

Techniques for cross-validation are used to guarantee stable model performance and reduce the possibility of overfitting.

v. Model Evaluation: Accuracy is the metric used to assess the produced model's performance

The Python programming language and well-known machine learning packages, including Scikit-learn, sklearn, pandas, numpy, and others, are used to analyze the data. Kaggle.com, a well-known online venue for data science competitions and collaboration provided the dataset used in this study. This dataset was selected because it is publicly available and pertinent to the study's goals.

Programming selection with python

Python is a high-level, object-oriented, interpreted programming language with dynamic semantics. Its high-level built-in data structures, dynamic typing, and dynamic binding make it very attractive for creating applications quickly and for use as a scripting or glue language to connect pre-existing components. Readability is given priority in Python's simple syntax, which reduces the cost of software maintenance. Python's support for modules and packages encourages software modularity and code reuse. The Python interpreter and the extensive standard library are freely distributable and accessible in source or binary form for all major systems. The following applications were also used in solving the problem:

Google collaboratory

Google Collaboratory is a free cloud-based platform that lets users utilize Jupyter notebooks to write, run, and share Python code. Without requiring complicated setup and preparation, Colab offers a collaborative environment for data scientists, researchers, and developers to work on machine learning and data science projects.

Jupyter notebook

A server-client application called the Jupyter Notebook App allows notepad entries to be changed and accessed from a web browser. The Jupyter Notebook App can be deployed on a remote server and accessible online, as this paper demonstrates, or it can be used locally without requiring web connectivity. In addition to displaying, editing, and executing note pad archives, the Jupyter Notebook App has a "Dashboard" (Notebook Dashboard), a "control board" that shows nearby records and lets you read note pad reports or close down their sections.

Numpy

NumPy is much the same as SciPy, Scikit-Learn, Pandas, and so forth, one of the bundles that you cannot miss when you are learning information science, principally in light of the fact that this library gives you a cluster information structure that holds a few advantages over Python records, for example, being increasingly reduced, quicker access in perusing and composing things, being progressively advantageous and increasingly productive. NumPy exhibits are somewhat similar to Python records, yet at the same time particularly unique in the meantime.

Pandas

A server-client application called the Jupyter Notebook App allows notepad entries to be changed and accessed from a web browser. The Jupyter Notebook App can be deployed on a remote server and accessible online, as this paper demonstrates, or it can be used locally without requiring web connectivity. In addition to displaying, editing, and executing note pad archives, the Jupyter Notebook App has a "Dashboard" (Notebook Dashboard), a "control

board" that shows nearby records and lets you read note pad reports or close down their sections.

Description of the data collected

The dataset used for the detection was collected from kaggle. The figures 1, 2 and 3 show the sample of the dataset used for the detection.

	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V
2	-0.5516	-0.6178	-0.99139	-0.31117	1.468177	-0.4704	0.207971	0.025791	0.403993	0.251412	-0.01831	0.277838	-0.11047	
3	1.612727	1.065235	0.489095	-0.14377	0.635558	0.463917	-0.1148	-0.18336	-0.14578	-0.06908	-0.22578	-0.63867	0.101288	
4	0.624501	0.066084	0.717293	-0.16595	2.345865	-2.89008	1.109969	-0.12136	-2.26186	0.52498	0.247998	0.771679	0.909412	
5	-0.22649	0.178228	0.507757	-0.28792	-0.63142	-1.05965	-0.68409	1.965775	-1.23262	-0.20804	-0.1083	0.005274	-0.19032	
6	-0.82284	0.538196	1.345852	-1.11967	0.175121	-0.45145	-0.23703	-0.03819	0.803487	0.408542	-0.00943	0.798278	-0.13746	
7	1.341262	0.359894	-0.35809	-0.13713	0.517617	0.401726	-0.05813	0.068653	-0.03319	0.084968	-0.20825	-0.55982	-0.0264	
8	-1.41691	-0.15383	-0.75106	0.167372	0.050144	-0.44359	0.002821	-0.61199	-0.04558	-0.21963	-0.16772	-0.27071	-0.1541	
9	-0.61947	0.291474	1.757964	-1.32387	0.686133	-0.07613	-1.22213	-0.35822	0.324505	-0.15674	1.943465	-1.01545	0.057504	
10	-0.70512	-0.11045	-0.28625	0.074355	-0.32878	-0.21008	-0.49977	0.118765	0.570328	0.052736	-0.07343	-0.26809	-0.20423	
11	1.017614	0.83639	1.006844	-0.44352	0.150219	0.739453	-0.54098	0.476677	0.451773	0.203711	-0.24691	-0.63375	-0.12079	
12	1.199644	-0.67144	-0.51395	-0.09505	0.23093	0.031967	0.253415	0.854344	-0.22137	-0.38723	-0.0093	0.313894	0.02774	
13	-0.25912	-0.32614	-0.09005	0.362832	0.928904	-0.12949	-0.80998	0.359985	0.707664	0.125992	0.049924	0.238422	0.00913	
14	0.227666	-0.24268	1.205417	-0.31763	0.725675	-0.81561	0.873936	-0.84779	-0.68319	-0.10276	-0.23181	-0.48329	0.084668	
15	-0.77366	0.323387	-0.01108	-0.17849	-0.65556	-0.19993	0.124005	-0.9805	-0.98292	-0.1532	-0.03688	0.074412	-0.07141	

creditcard (1)

Ready

Figure 1: Shows the sample of the dataset used for the prediction.

	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
2	0.025791	0.403993	0.251412	-0.01831	0.277838	-0.11047	0.066928	0.128539	-0.18911	0.133558	-0.02105	149.62	0
3	-0.18336	-0.14578	-0.06908	-0.22578	-0.63867	0.101288	-0.33985	0.16717	0.125895	-0.00898	0.014724	2.69	0
4	-0.12136	-2.26186	0.52498	0.247998	0.771679	0.909412	-0.68928	-0.32764	-0.1391	-0.05535	-0.05975	378.66	0
5	1.965775	-1.23262	-0.20804	-0.1083	0.005274	-0.19032	-1.17558	0.647376	-0.22193	0.062723	0.061458	123.5	0
6	-0.03819	0.803487	0.408542	-0.00943	0.798278	-0.13746	0.141267	-0.20601	0.502292	0.219422	0.215153	69.99	0
7	0.068653	-0.03319	0.084968	-0.20825	-0.55982	-0.0264	-0.37143	-0.23279	0.105915	0.253844	0.08108	3.67	0
8	-0.61199	-0.04558	-0.21963	-0.16772	-0.27071	-0.1541	-0.78006	0.750137	-0.25724	0.034507	0.005168	4.99	0
9	-0.35822	0.324505	-0.15674	1.943465	-1.01545	0.057504	-0.64971	-0.41527	-0.05163	-1.20692	-1.08534	40.8	0
10	0.118765	0.570328	0.052736	-0.07343	-0.26809	-0.20423	1.011592	0.373205	-0.38416	0.011747	0.142404	93.2	0
11	0.476677	0.451773	0.203711	-0.24691	-0.63375	-0.12079	-0.38505	-0.06973	0.094199	0.246219	0.083076	3.68	0
12	0.854344	-0.22137	-0.38723	-0.0093	0.313894	0.02774	0.500512	0.251367	-0.12948	0.04285	0.016253	7.8	0
13	0.359985	0.707664	0.125992	0.049924	0.238422	0.00913	0.99671	-0.76731	-0.49221	0.042472	-0.05434	9.99	0
14	-0.84779	-0.68319	-0.10276	-0.23181	-0.48329	0.084668	0.392831	0.161135	-0.35499	0.026416	0.042422	121.5	0
15	-0.9805	-0.98292	-0.1532	-0.03688	0.074412	-0.07141	0.104744	0.548265	0.104094	0.021491	0.021293	27.5	0

creditcard (1)

Ready

Figure 2: Shows continuation of the dataset used for the prediction

	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V
2	-0.5516	-0.6178	-0.99139	-0.31117	1.468177	-0.4704	0.207971	0.025791	0.403993	0.251412	-0.01831	0.277838	-0.11047	
3	1.612727	1.065235	0.489095	-0.14377	0.635558	0.463917	-0.1148	-0.18336	-0.14578	-0.06908	-0.22578	-0.63867	0.101288	
4	0.624501	0.066084	0.717293	-0.16595	2.345865	-2.89008	1.109969	-0.12136	-2.26186	0.52498	0.247998	0.771679	0.909412	
5	-0.22649	0.178228	0.507757	-0.28792	-0.63142	-1.05965	-0.68409	1.965775	-1.23262	-0.20804	-0.1083	0.005274	-0.19032	
6	-0.82284	0.538196	1.345852	-1.11967	0.175121	-0.45145	-0.23703	-0.03819	0.803487	0.408542	-0.00943	0.798278	-0.13746	
7	1.341262	0.359894	-0.35809	-0.13713	0.517617	0.401726	-0.05813	0.068653	-0.03319	0.084968	-0.20825	-0.55982	-0.0264	
8	-1.41691	-0.15383	-0.75106	0.167372	0.050144	-0.44359	0.002821	-0.61199	-0.04558	-0.21963	-0.16772	-0.27071	-0.1541	
9	-0.61947	0.291474	1.757964	-1.32387	0.686133	-0.07613	-1.22213	-0.35822	0.324505	-0.15674	1.943465	-1.01545	0.057504	
10	-0.70512	-0.11045	-0.28625	0.074355	-0.32878	-0.21008	-0.49977	0.118765	0.570328	0.052736	-0.07343	-0.26809	-0.20423	
11	1.017614	0.83639	1.006844	-0.44352	0.150219	0.739453	-0.54098	0.476677	0.451773	0.203711	-0.24691	-0.63375	-0.12079	
12	1.199644	-0.67144	-0.51395	-0.09505	0.23093	0.031967	0.253415	0.854344	-0.22137	-0.38723	-0.0093	0.313894	0.02774	
13	-0.25912	-0.32614	-0.09005	0.362832	0.928904	-0.12949	-0.80998	0.359985	0.707664	0.125992	0.049924	0.238422	0.00913	
14	0.227666	-0.24268	1.205417	-0.31763	0.725675	-0.81561	0.873936	-0.84779	-0.68319	-0.10276	-0.23181	-0.48329	0.084668	
15	-0.77366	0.323387	-0.01108	-0.17849	-0.65556	-0.19993	0.124005	-0.9805	-0.98292	-0.1532	-0.03688	0.074412	-0.07141	

creditcard (1)

Ready

Figure 3: Shows amount and class of the dataset used for the prediction.

Preprocessing

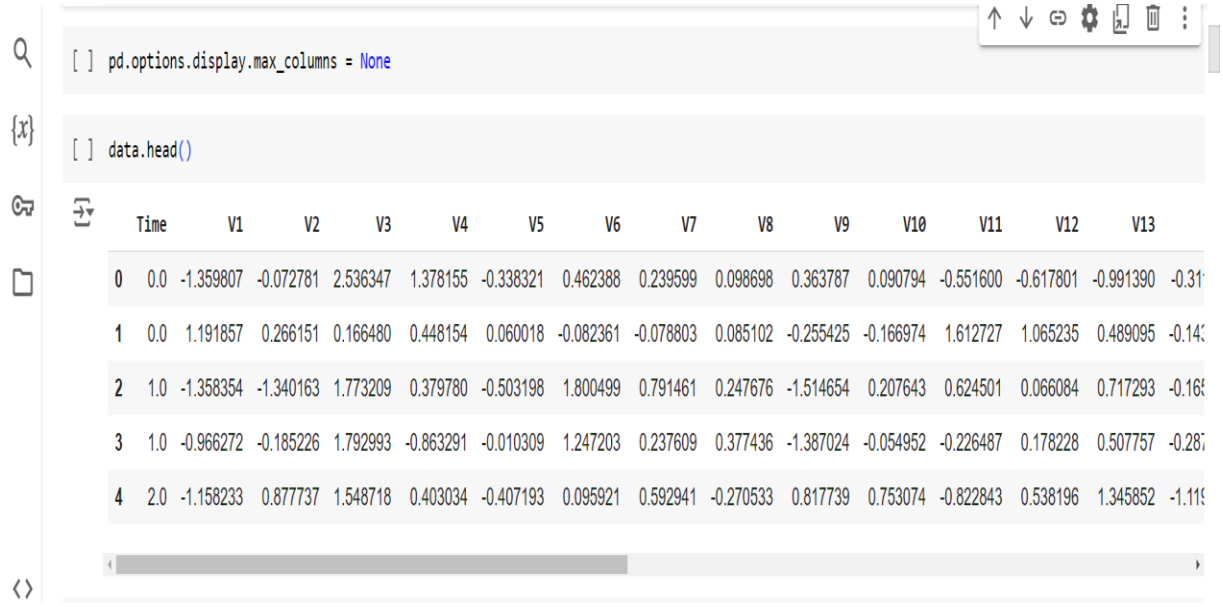
The pre-processed dataset was split into two groups with 80:20 ratios. The training set which represents 80% of all the data and testing which represents 20% of the data set. A comparison is made between the two techniques which are: Logistics Regression and Decision Tree Classifier. The first 5 data.head is displayed in figure 4. A DataFrame's first few rows are returned by Data.head(); by default, the first five rows are returned. It is useful for quickly checking the contents of a DataFrame, such as column names, data types, and a preview of the data in the dataset. While in figure 5, the last 5 data.tail of the dataset is diaplayed. A DataFrame's last few rows are returned by Data.tail(); by default, the last five rows are returned. This function is useful for quick checking the end of a DataFrame to identify any trailing data issues or ensure data integrity.

```
[ ] pd.options.display.max_columns = None
```

```
[ ] data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.31
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.14
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.16
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.28
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.11

Figure 4: Displays the first 5 data.head of the dataset.

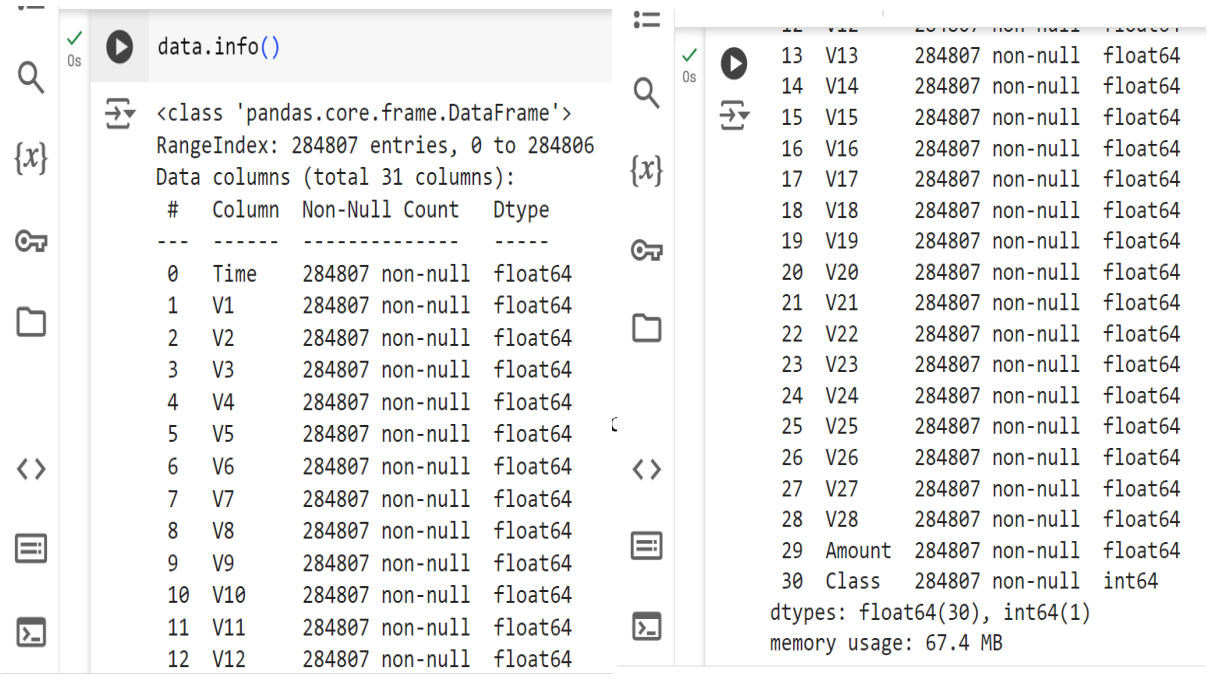


```
[ ] pd.options.display.max_columns = None

[ ] data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.31
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.14
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.16
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.28
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.11

Figure 5: Displays the last 5 data.tail of the dataset

The data.info () is displayed in figure 6. A DataFrame's data types, memory utilization, and number of non-null elements are all summarized in a succinct manner via Data.info (). It is helpful for rapidly comprehending the DataFrame's structure. While in figure 7 data.isnull ().sum() is displayed. Data.isnull(). Sum () provides a summary of the missing values in each column. It counts the number of null entries in each column, which is useful for data cleaning and preprocessing.



```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Figure 6: Checking the data type and any missing Values in the dataset

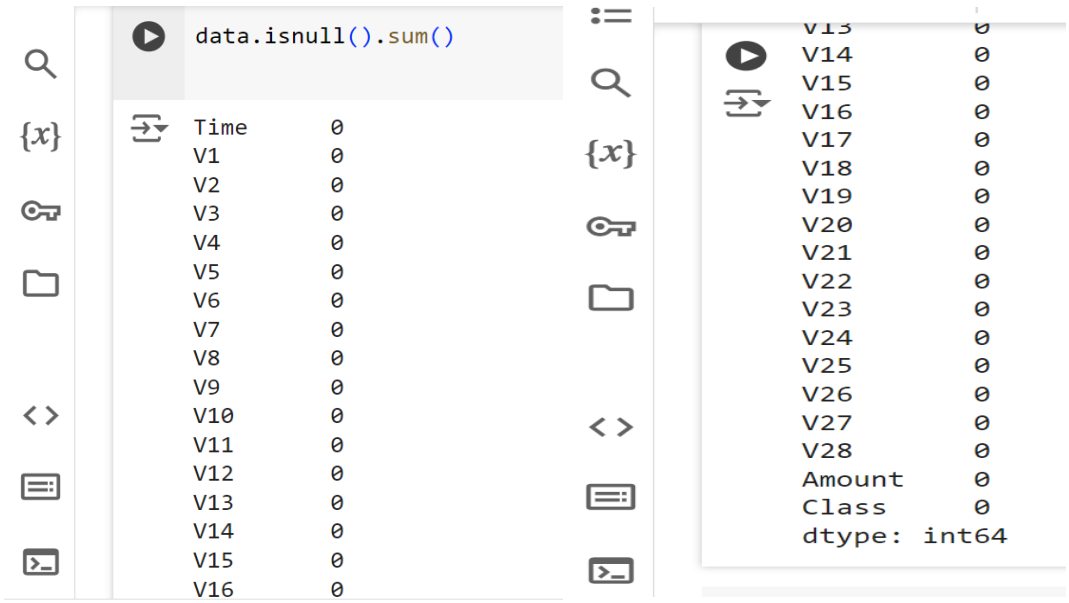
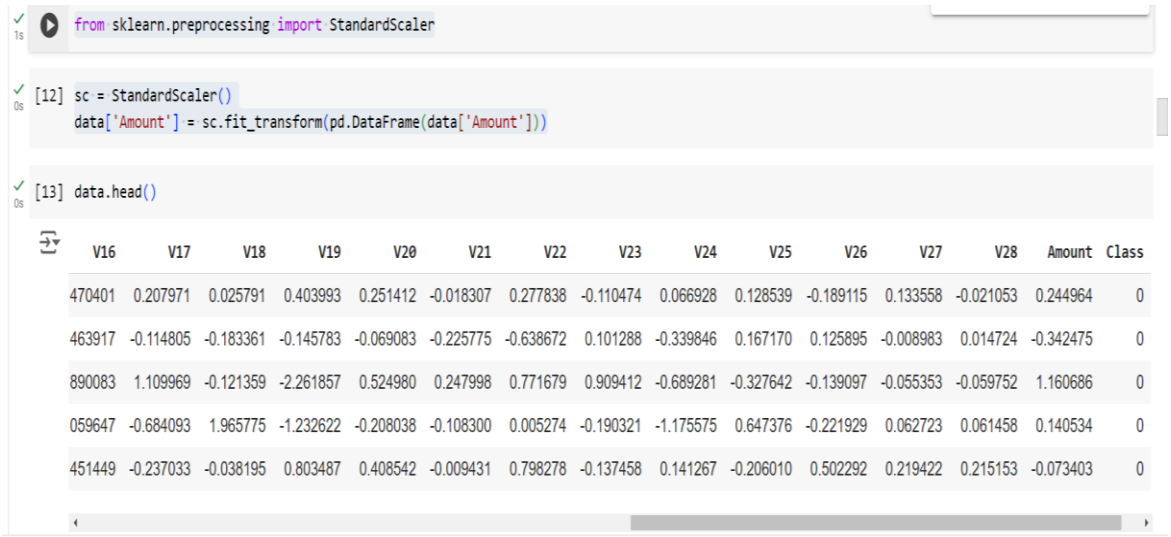


Figure 7: Displaying the data.isnull().sum() and data type

In figure 8, the provided code standardizes the Amount column in the DataFrame data using StandardScaler from sklearn.preprocessing. It first imports the necessary libraries, StandardScaler from sklearn.preprocessing and pandas for DataFrame manipulation. A sample DataFrame data is created with columns Name, Amount, and City. An instance of StandardScaler is initialized and assigned to the variable sc. The fit_transform method of the scaler is then applied to the Amount column. The fit_transform method expects a 2D array, so the column is converted to a DataFrame before applying the method. This method computes the mean and standard deviation for the Amount column, scales the data, and returns the transformed values, which replace the original values in the Amount column. The transformed Amount values will have a mean of 0 and a standard deviation of 1. The code then prints the updated DataFrame, showing the standardized Amount column. From sklearn.preprocessing import StandardScaler sc = StandardScaler() data['Amount'] = sc.fit_transform(pd.DataFrame(data['Amount']))



```
from sklearn.preprocessing import StandardScaler
```

```
[12] sc = StandardScaler()
     data['Amount'] = sc.fit_transform(pd.DataFrame(data['Amount']))
```

```
[13] data.head()
```

	V16	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
470401	0.207971	0.025791	0.403993	0.251412	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	0.244964		0
463917	-0.114805	-0.183361	-0.145783	-0.069083	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	-0.342475		0
890083	1.109969	-0.121359	-2.261857	0.524980	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	1.160686		0
059647	-0.684093	1.965775	-1.232622	-0.208038	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	0.140534		0
451449	-0.237033	-0.038195	0.803487	0.408542	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	-0.073403		0

Figure 8: Standardizing the amount column to fit to the other column data type

The train_test_split function from the sklearn.model selection module is used to divide a dataset into training and testing sets, as demonstrated in the code snippet supplied in figure 9. This is a description of the code.

1. Bringing in the Necessary Library: Sklearn.model_selection's train_test_split method is imported. Arrays or matrices can be divided into random train and test subsets with this function.
2. Dividing the Information: The feature matrix X and the target vector y are passed to the train_test_split function. Additionally, the function accepts two more parameters.
   i. test_size=0.2: This value indicates what percentage of the dataset should be used in the test split. In this case, the test set receives 20% of the data, while the training set receives the remaining 80%.
   ii. To make sure the split is repeatable, use the random_state=42 argument. Every time the code is executed with the same random_state value, the same split will be produced.
3. Assigning the Split Data: The function returns four values:
   i. X_train: The features' training set
   ii. X_test: The features' testing set.
   iii. y_train: The target's training set
   iv. y_test: The target's testing set

The corresponding variables are then assigned to these four returning values. By training a machine learning model on the training set and testing it on the test set to see if it can generalize to new data, this division is crucial for assessing the model's performance. Train_test_split is imported from sklearn.model_selection. train_test_split(X, y, test_size = 0.2, random_state = 42) = X_train, X_test, y_train, y_test.

Model simulation

Import sklearn.linear_model import numpy as np Importing logistic regression from sklearn.ensemble The RandomForestClassifier import from sklearn.tree The sklearn.metrics DecisionTreeClassifier program import recall_score, precision_score, f1_score, and accuracy_score. The popular Python library for numerical operations, numpy, is imported as np. It supports arrays, matrices, and a number of mathematical functions.

LogisticsFor binary classification issues, regression from sklearn.linear_model is imported. It simulates the likelihood that a specific input point is a member of a particular class. Imported is the RandomForestClassifier from sklearn.ensemble. This classifier is an ensemble learning technique that builds several decision trees during training and produces a class that is the mean prediction (regression) or the mode of the classes (classification) of the individual trees. It is renowned for its great accuracy, resilience, and capacity to manage sizable datasets with more dimensions.

Importing the DecisionTreeClassifier from sklearn.tree allows you to build a model that learns basic decision rules derived from the features of the data to forecast a target variable's value. It creates a decision tree-like model by separating the dataset into subsets based on each stage's most crucial aspect.

Precision_score, recall_score, f1_score, and accuracy_score from Sklearn.Imported measurements are used to assess how well categorization models function. The ratio of successfully predicted instances to total instances is computed by accuracy_score. Precision

and recall are balanced by the f1_score, which is the harmonic mean of the two metrics. Recall_score evaluates the classifier's capacity to identify every positive sample, whereas precision_score gauges the accuracy of the positive predictions. These imports set up the environment for building, training, and evaluating various machine learning models on a given dataset using metrics to assess their performance.

In figure 9 the code defines a dictionary classifier with two classifiers: LogisticRegression and DecisionTreeClassifier. It then iterates over this dictionary, fits each classifier to produce predictions on the test data (X_test) based on the training data (X_train, y_train). The accuracy, precision, recall, and F1 score performance metrics used to compare the predictions to the actual test labels (y_test) are printed for each classifier.

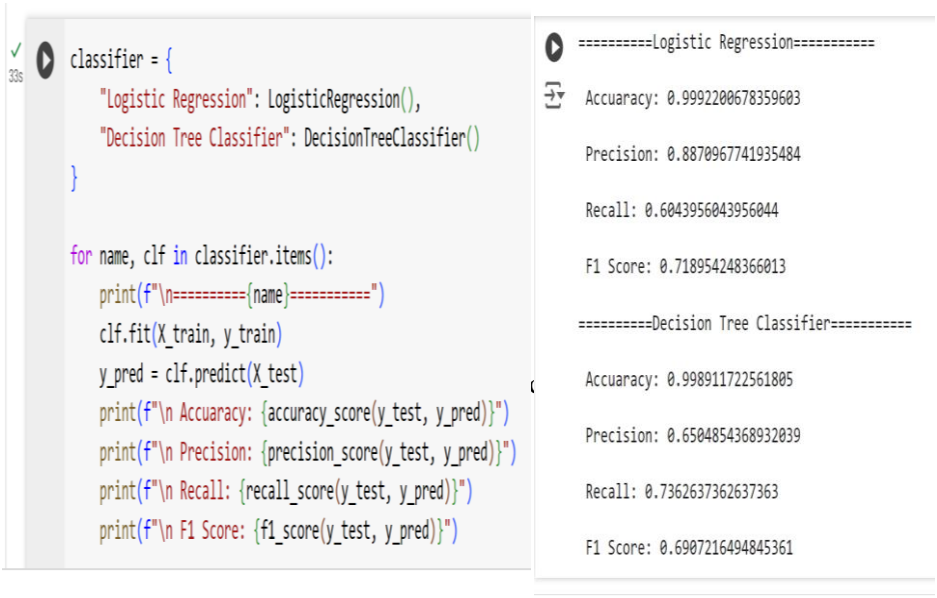


Figure 9: Shows the performance metrics score on each algorithm.

Undersampling

One method employed in data analysis, especially when dealing with unbalanced datasets, is undersampling. One class may substantially exceed other classes in a large number of real-world datasets. Biased models that perform badly on the minority class may result from this imbalance. This problem is solved by undersampling, which balances the distribution of classes by lowering the number of examples in the majority class. To achieve this, samples from the majority class can be randomly removed until there are roughly equal numbers of samples in each class. The objective is to enhance the learning algorithm's performance on the minority class while preventing it from becoming overloaded by the majority class. It is crucial to remember that undersampling may not always be the optimal strategy and can result in the loss of critical information, particularly if the dataset is tiny. Depending on the particular issue and dataset properties, other options like oversampling the minority class or applying sophisticated methods like SMOTE (Synthetic Minority Oversampling Technique) may also be taken into consideration. Figures 10 and 11 display the undersampled data and the outcome following undersampling.

.

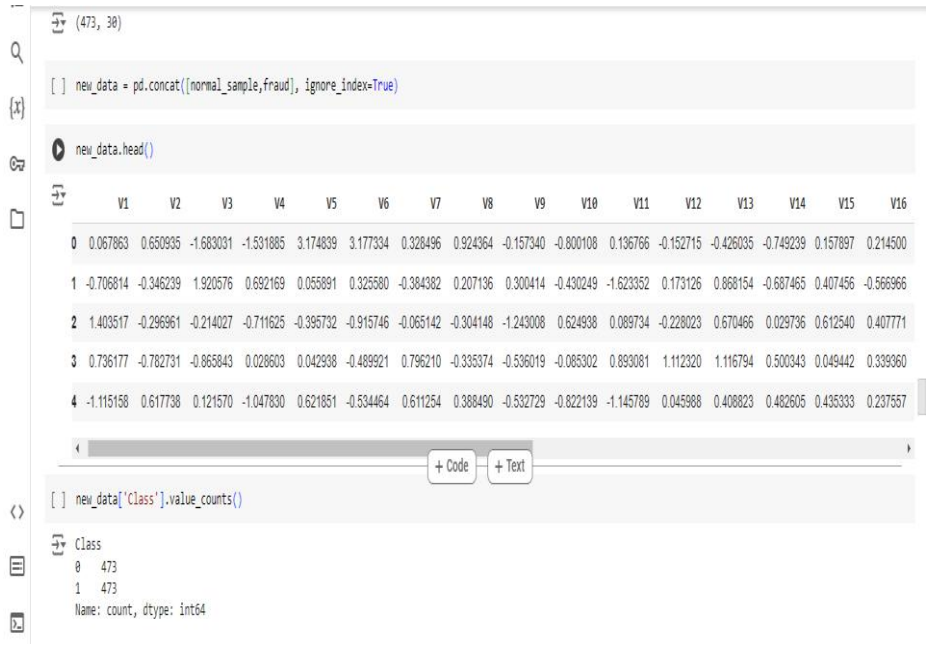


Figure 10: Shows the undersampled data

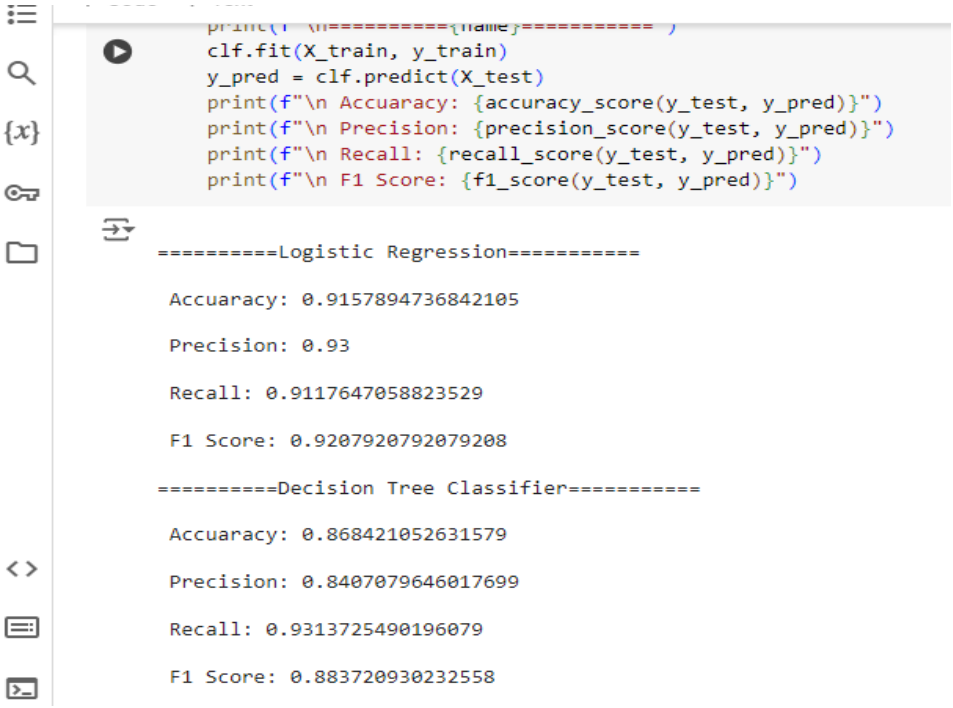
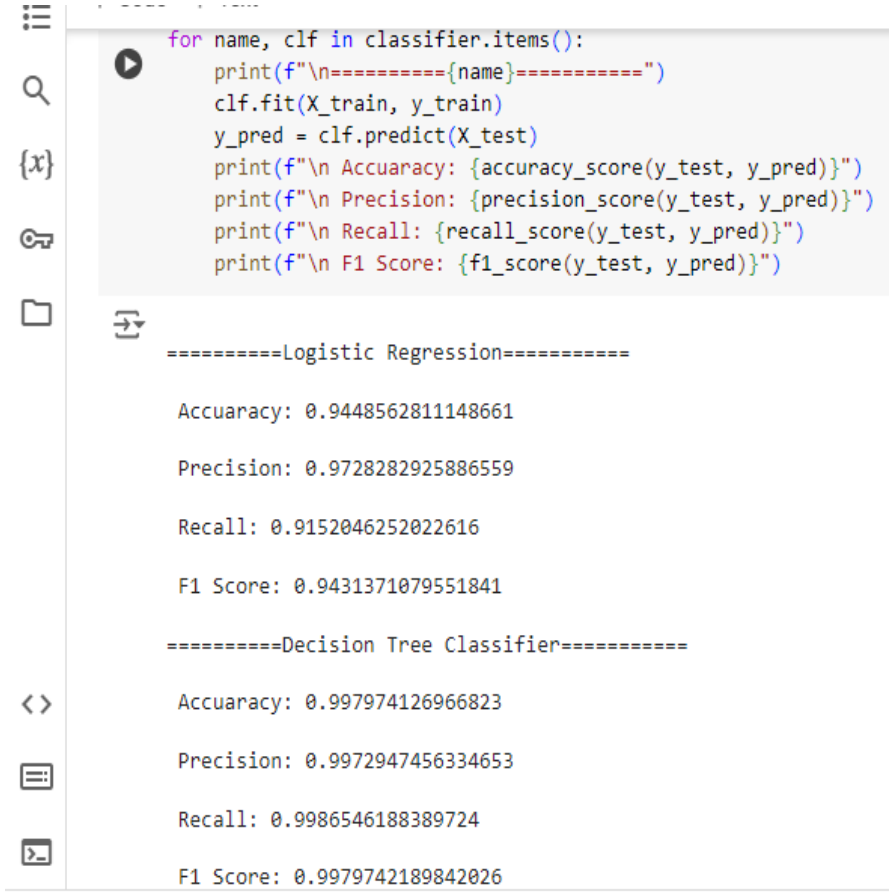


Figure 11: Shows the result after undersampling.

Oversampling

By expanding the minority class's number of instances, oversampling is a strategy used to address class imbalance in datasets. By doing this, the distribution of classes is balanced, and machine learning models that could otherwise favor the majority class perform better. Oversampling can be done in a number of ways. Duplicating examples from the minority class at random until the distribution of classes is balanced is known as random oversampling. Simple and efficient, it copies current data without adding additional information, which might result in overfitting. The Synthetic Minority Over-Sampling Technique (SMOTE) is an additional technique that creates synthetic examples instead of replicating preexisting ones. In order to create a new, synthetic instance that combines two or more similar examples from

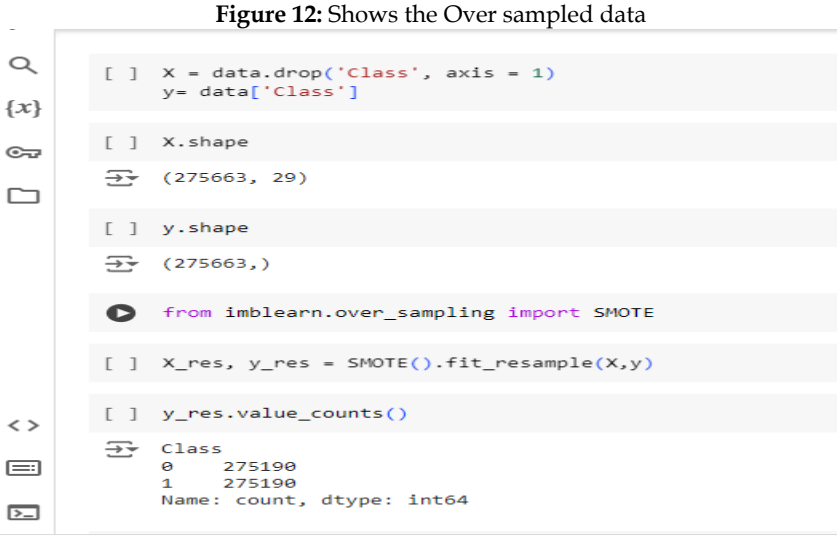
the minority class, SMOTE first chooses which instances to use. Compared to random oversampling, this lessens the chance of overfitting and helps to establish a more generic decision boundary. A variation of SMOTE called ADASYN (Adaptive Synthetic Sampling) creates synthetic data with an emphasis on cases that are challenging to learn. It makes sure that more synthetic data is produced for minority instances that are more difficult to categorize by adaptively determining how many synthetic examples to create for each minority instance based on the neighborhood's density. Figures 12 and 13 display the oversampled data and the outcome following oversampling.



```
for name, clf in classifier.items():
    print(f"\n=========={name}==========")
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(f"\n Accuaracy: {accuracy_score(y_test, y_pred)}")
    print(f"\n Precision: {precision_score(y_test, y_pred)}")
    print(f"\n Recall: {recall_score(y_test, y_pred)}")
    print(f"\n F1 Score: {f1_score(y_test, y_pred)}")
```

```
==========Logistic Regression==========

 Accuaracy: 0.9448562811148661

 Precision: 0.9728282925886559

 Recall: 0.9152046252022616

 F1 Score: 0.9431371079551841

==========Decision Tree Classifier==========

 Accuaracy: 0.997974126966823

 Precision: 0.9972947456334653

 Recall: 0.9986546188389724

 F1 Score: 0.9979742189842026
```

Figure 12: Shows the Over sampled data



```
[ ]  X = data.drop('Class', axis = 1)
     y= data['Class']

[ ]  X.shape

     (275663, 29)

[ ]  y.shape

     (275663,)

     from imblearn.over_sampling import SMOTE

[ ]  X_res, y_res = SMOTE().fit_resample(X,y)

[ ]  y_res.value_counts()

     Class
     0    275190
     1    275190
     Name: count, dtype: int64
```

Figure 13: Shows result after Oversampling.

Building a detecting system

dtc = DecisionTreeClassifier() initializes a new instance of the DecisionTreeClassifier from sklearn.tree. This classifier creates a decision tree to predict the target variable based on the input features. dtc.fit(X_res, y_res) trains the decision tree classifier using the resampled training data. Here, X_res represents the resampled feature matrix and y_res represents the resampled target vector. The fit method builds the decision tree by finding the optimal splits in the data to predict the target variable. The process of using decisiontreeClassifier as a detecting system is shown in figure 14.

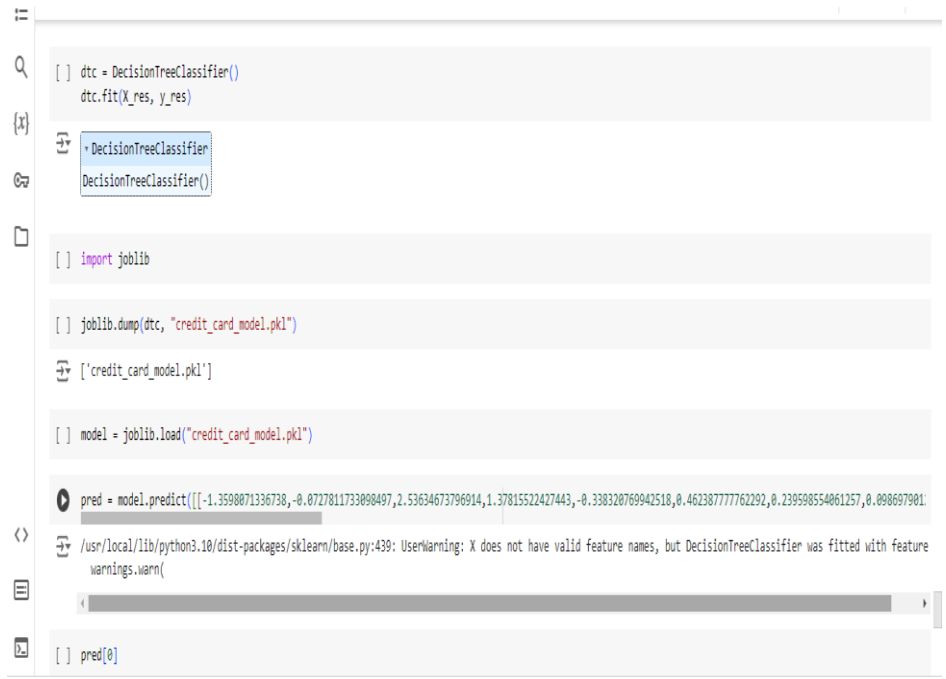


Figure 14: Shows the built system for detection

RESULTS AND DISCUSSION

This section summarizes the main findings of the study, including the accuracy and performance of the machine learning models and the relationships between the variables. Table 1 shows the results of all algorithms with their metric scores.

Table 1 All the algorithms used with their metric scores

Model	Accuracy	Precision	Recall	F1 Score
Logistics Regression	0.9992200678359603	0.8870967741935484	0.6043956043956044	0.718954248366013
Decision Tree Classifier	0.9989298605191084	0.66	0.7252747252747253	0.6910994764397906
Undersampling				
Logistics Regression	0.9157894736842105	0.93	0.9117647058823529	0.9207920792079208
Decision Tree Classifier	0.868421052631579	0.8407079646017699	0.9313725490196079	0.883720930232558
Oversampling				
Logistics Regression	0.9448562811148661	0.9728282925886559	0.9152046252022616	0.9431371079551841
Decision Tree Classifier	0.997974126966823	0.9972947456334653	0.9986546188389724	0.9979742189842026

Based on the above analysis, it is shows that Decision Tree after oversampling have the best accuracy of 99% across all the evaluation metrics, which make Decision tree the best model to build our system with. When a decision tree is used with unbalanced data, oversampling is frequently more effective than undersampling. Although undersampling lowers the number of cases in the majority class, it can also result in the loss of crucial information and impair the quality of the model. Without erasing any data from the majority class, oversampling raises the number of occurrences of the minority class. This method enhances the decision tree's learning capabilities and boosts performance indicators including F1-score, precision, and recall. Oversampling guarantees the model has sufficient data to correctly distinguish between classes by include more representative examples from the minority class. The findings of this study may be compared to those of Sadgalia et al. (2018), who investigated the effectiveness of machine learning methods in identifying financial fraud. The study's goal is to determine which approaches and strategies produce the best outcomes out of all those that have been refined to date. According to this poll, hybrid fraud detection strategies which combine the benefits of numerous traditional detection techniques—are the most widely used. In the investigation, it was found that the decision tree classifier performed better than all the machine learning techniques individually.

CONCLUSION

The study comes to the conclusion that machine learning approaches can successfully identify fraudulent financial transactions, especially when paired with oversampling techniques. The study's most accurate model turned out to be the Decision Tree Classifier. This outcome emphasizes how crucial it is to resolve data imbalance in order to improve model performance.

Furthermore, the results emphasize how important it is to keep refining machine learning algorithms in order to accommodate changing fraud strategies. The study shows that sophisticated fraud schemes are frequently difficult to detect using conventional fraud detection techniques. Financial organizations can increase their capacity to detect fraud by utilizing sophisticated machine learning algorithms, which will improve security and lower losses. The paper also suggests that in order to preserve the effectiveness of fraud detection models as fraud strategies change, future research should concentrate on real-time detection capabilities and adaptive learning systems.

This research work also highlights how crucial strong preprocessing procedures and high-quality data are to the effectiveness of ML models. To train precise and dependable models, high-quality data must be ensured. Additionally, ML models must be transparent and explainable in order to win over stakeholders and guarantee that the judgments they make are reasonable and comprehensible.

Financial institutions should use sophisticated oversampling methods like SMOTE and ADASYN to enhance the effectiveness of fraud detection models in light of the findings. ML models must be regularly reviewed and updated in order to remain successful against emerging fraud trends. To capitalize on each ML algorithm's advantages and increase detection accuracy, think about creating hybrid models that incorporate several of them. To improve model training and performance, make sure to use high-quality data and thorough preprocessing procedures. Use techniques to make ML models more interpretable so that stakeholders can comprehend and have faith in the model's judgments.

In order to give prompt alerts and solutions, future research could investigate the creation and assessment of machine learning models that are able to detect fraud in real-time. Examine how blockchain technology and machine learning-based fraud detection may be combined to improve security and transparency. Examine adaptive learning systems that enhance their fraud detection skills by continuously learning from fresh data. Expand the study to areas other than financial transactions, such as insurance and healthcare to assess how broadly applicable the suggested techniques are. To guarantee equitable and responsible use of technology, address ethical issues pertaining to data protection and potential biases in ML models. The efficiency and resilience of fraud detection systems can be greatly increased by putting these suggestions into practice and investigating potential avenues for future study which will help to ensure the safety and stability of financial systems.

REFERENCES

Chou, C. Y., Lee, C. Y., & Chiang, C. (2020). A novel hybrid financial fraud detection method integrating clustering and text mining. *Applied Intelligence*, 50(7), 2291–2313.

Ngai, E. W. T., Hu, Y., Wong, Y. H., Chen, Y., & Sun, X. (2011). The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision Support Systems*, 50(3), 559–569.

Kou, Y., Lu, C.-T., & Sirwongwattana, S. (2021). Survey on fraud detection techniques. *IEEE Transactions on Network Science and Engineering*, 8(3), 1643–1658.

Sánchez, D. C., Channouf, L., Vallejo, E. E., & Bolívar, M. C. P. R. (2020). A hybrid method for fraud detection in credit card transactions based on unsupervised and supervised learning. *IEEE Access*, *8*, 70907–70918.

Fu, Y., Cao, J., & Shi, W. (2020). FD-GAN: A deep learning-based financial fraud detection framework. *IEEE Access*, 8, 78017–78025.

Pumsirirat, R., & Nukoolkit, T. (2020). Real-time credit card fraud detection using one-class SVM with semantic embedding. *Journal of Intelligent & Fuzzy Systems*, 39(3), 3147–3159.

Tsai, C.-F., Lin, Y.-C., & Chang, Y.-C. (2018). A hybrid machine learning model for fraud detection in financial transactions. *Journal of Physics: Conference Series*, 1168(5), 052015.

Sánchez, D. C., Channouf, L., Vallejo, E. E., & Bolívar, M. C. P. R. (2021). A review of applications of machine learning techniques for fraud detection. In *2021 16th Iberian Conference on Information Systems and Technologies (CISTI)*, 1–6. IEEE.

Strelcenia, E., & Prakoonwit, S. (2023). Improving classification performance in credit card fraud detection by using new data augmentation. *AI*, 4(1), 8.

Ali, A., Razak, S. A., Othman, S. H., Eisa, T. A. E., Al-Dhaqm, A., Nasser, M., Elhassan, T., Elshafie, H., & Saif, A. (2022). Financial fraud detection based on machine learning: A systematic literature review. *Applied Sciences*, 12(19), 9637.

Eryu, P. (2024). Machine Learning Transaction Fraud Detection and Prevention. *Transactions on Economics, Business and Management Research*. 5(1), 243-249.

Sadgalia, I Saela, N Benabboua, F (2018). Performance of Machine Learning Techniques in the Detection of Financial Frauds, F. Second International Conference on Intelligent Computing in Data Sciences. Procedia Computer Science 148 (2019) 45–54.