# 

# CREDIT CARD FRAUD DETECTION SYSTEM

# Team – 513

# Kasireddy Sai Deepak Reddy 20BEC1096

# L.Sai Teja Reddy 20BEC1274

# Bharath Suram 20BEC1341

# Chanikya Annabathuni 20BEC1047

# Technology Track :- Applied Data Science

# INTRODUCTION

* 1. Overview

A brief description about your project

* 1. Purpose

The use of this project. What can be achieved using this.

# LITERATURE SURVEY

* 1. Existing problem

Existing approaches or method to solve this problem

* 1. Proposed solution

What is the method or solution suggested by you?

# THEORITICAL ANALYSIS

* 1. Block diagram

Diagrammatic overview of the project.

* 1. Hardware / Software designing

Hardware and software requirements of the project

# EXPERIMENTAL INVESTIGATIONS

Analysis or the investigation made while working on the solution.

# FLOWCHART

Diagram showing the control flow of the solution

# RESULT

Final findings (Output) of the project along with screenshots.

# ADVANTAGES & DISADVANTAGES

List of advantages and disadvantages of the proposed solution

# APPLICATIONS

The areas where this solution can be applied

# CONCLUSION

Conclusion summarizing the entire work and findings.

# FUTURE SCOPE

Enhancements that can be made in the future.

# BIBILOGRAPHY

References of previous works or websites visited/books referred for analysis about the project, solution previous findings etc.

# APPENDIX

A. Source Code

Attach the code for the solution built.

**CHAPTER 1 INTRODUCTION**

* 1. Overview

The increasing popularity of credit cards as a preferred mode of payment has led to a rise in credit card fraud. Fraudulent activities related to credit cards not only cause financial losses but also erode trust in the banking system. To combat this issue, credit card companies and financial institutions are continually exploring innovative techniques for detecting and preventing fraud. One such technique is the application of advanced data analytics and machine learning algorithms for credit card fault detection.

* 1. Purpose

This project aims to develop a robust credit card fault detection system using cutting-edge machine learning techniques. By leveraging historical credit card transaction data, the system will identify patterns and anomalies associated with fraudulent activities. The project's primary goal is to improve the accuracy and efficiency of existing fraud detection methods, ultimately minimizing financial losses and enhancing customer confidence in credit card transaction.

# 

# CHAPTER -2 LITERATURE SURVEY

When it comes to credit card fraud detection, there are several existing approaches and methods that have been widely used. Here are some of the commonly employed techniques:

**Rule-Based Systems:** These systems use predefined rules and heuristics to identify potential fraudulent transactions. Rules can be based on various factors such as transaction amount, location, time, or unusual patterns. While rule-based systems are simple to implement, they may not be effective in detecting sophisticated fraud patterns.

**Machine Learning:** Machine learning techniques have gained significant popularity in credit card fraud detection. Supervised learning algorithms, such as logistic regression, decision trees, random forests, and support vector machines (SVM), can be trained on labeled datasets to identify fraudulent transactions. Unsupervised learning methods like clustering and anomaly detection algorithms like Isolation Forest or Local Outlier Factor can also be employed.

**Neural Networks:** Deep learning models, particularly neural networks, have shown promising results in credit card fraud detection. Techniques such as feedforward neural networks, convolutional neural networks (CNNs), and recurrent neural networks (RNNs) can be utilized to learn complex patterns and detect fraudulent transactions.

**Ensemble Methods:** Ensemble methods combine multiple models to improve fraud detection accuracy. Techniques like bagging (e.g., Random Forests) and boosting (e.g., AdaBoost, XGBoost) can be applied to aggregate predictions from multiple base models and make more accurate predictions.

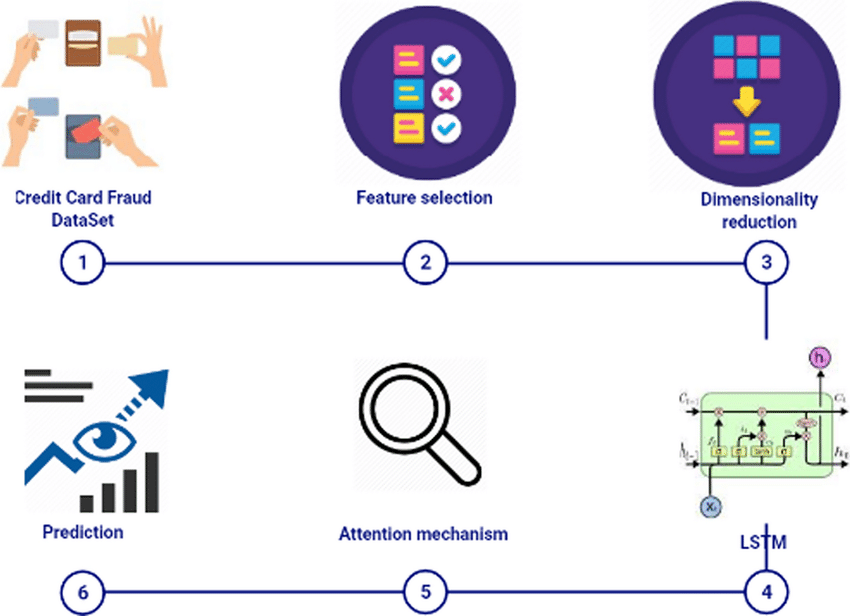
**Anomaly Detection:** Anomaly detection approaches focus on identifying transactions that deviate significantly from normal behavior. These methods rely on statistical techniques, clustering algorithms, or density-based methods to detect unusual patterns in transaction data.

**Hybrid Approaches:** Combining multiple techniques mentioned above can lead to enhanced fraud detection performance. For instance, a hybrid system may integrate rule-based systems with machine learning models to leverage the strengths of both approaches.

It's worth noting that the choice of method depends on various factors, including the available dataset, computational resources, time constraints, and desired level of accuracy. It's common to experiment with multiple approaches to determine the most effective solution for credit card fraud detection.

**CHAPTER 3**

3.1. Block diagram



A block diagram is a graphical representation that illustrates the components and their relationships in a system. In the context of credit card fraud detection, a block diagram can provide an overview of the various stages and components involved in the fraud detection process. Here's a general block diagram for credit card fraud detection:

Data Acquisition:

This stage involves obtaining the credit card transaction data from various sources, such as banking systems or payment processors. The data typically includes features like transaction amount, time, location, and customer information.

Data Preprocessing:

In this stage, the acquired data is preprocessed to ensure its quality and prepare it for further analysis. Steps may include data cleaning, handling missing values, outlier detection, and data normalization.

Feature Extraction:

Feature extraction involves selecting or creating relevant features from the preprocessed data that can be used to distinguish between fraudulent and non-fraudulent transactions. Common features used in credit card fraud detection include transaction amount, time of the day, geographical location, and customer behavior patterns.

Model Development:

This stage involves developing a predictive model using machine learning algorithms or other statistical techniques. The model is trained using labeled data, where fraudulent and non-fraudulent transactions are appropriately identified.

Model Evaluation:

The trained model is evaluated using evaluation metrics such as accuracy, precision, recall, and F1-score to assess its performance in detecting credit card fraud. Cross-validation techniques may be used to ensure the model's generalizability.

Model Deployment:

Once the model is deemed effective, it can be deployed in a production environment where it can analyze incoming credit card transactions in real-time. The model predicts the likelihood of fraud for each transaction.

Fraud Detection:

The deployed model continuously analyzes incoming credit card transactions and flags suspicious transactions based on the predicted fraud likelihood. These flagged transactions can be further reviewed and investigated by fraud analysts

3.2. Hardware / Software designing

* Define the overall system architecture, considering factors like scalability, performance, and integration with existing systems.
* Design the data ingestion component to receive and preprocess credit card transaction data from various sources.
* Design real-time processing components that analyze incoming transactions in real-time using predictive models or rule-based engines.
* Develop and implement predictive models that assess the likelihood of fraud for each transaction, considering machine learning techniques and feature engineering.
* Design a rule engine to enforce predefined rules or thresholds for detecting suspicious transactions, in collaboration with predictive models.
* Design user interface components, including dashboards and visualizations, to present fraud cases, alerts, and statistics to fraud analysts and administrators.
* Ensure seamless integration with external systems and databases, such as customer databases or case management systems.
* Implement security measures to protect sensitive data and comply with data privacy regulations.
* Design robust error handling mechanisms and implement logging and monitoring capabilities for system maintenance and debugging.
* Define a testing strategy and conduct testing to validate the functionality, performance, and accuracy of the system.
* Document the software design, architecture, and implementation details for future maintenance and knowledge transfer.
* Throughout the process, collaborate with stakeholders, including fraud analysts and domain experts, to ensure the system meets their requirements and addresses specific fraud detection challenges.
* It's important to note that this is a simplified summary, and each step involves further details and considerations depending on the specific requirements and context of the credit card fraud detection project.

# 

# CHAPTER-4 EXPERIMENTAL INVESTIGATION

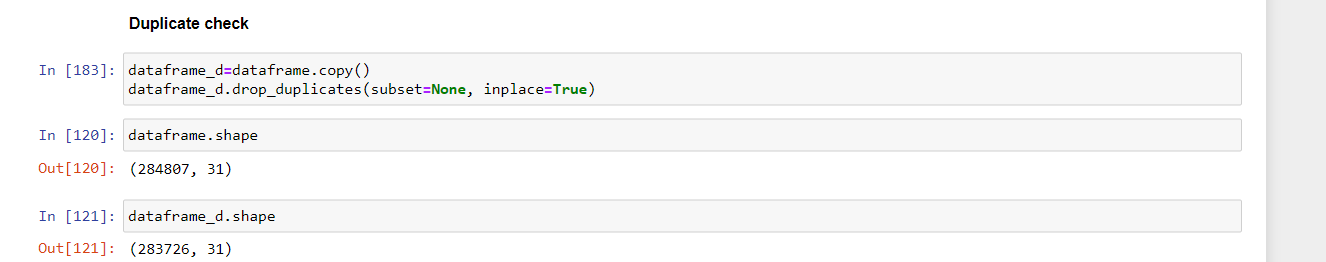
# In an experimental analysis of credit card fraud detection with an imbalanced dataset, several steps are followed to improve the performance of the models. The dataset is first preprocessed by handling missing values, removing irrelevant features, and performing feature scaling or normalization. The dataset is then split into training and testing sets, ensuring a representative distribution of the fraud class in both sets.

# Null Values or Missing Values:The dataset used donot have any null values or missing values

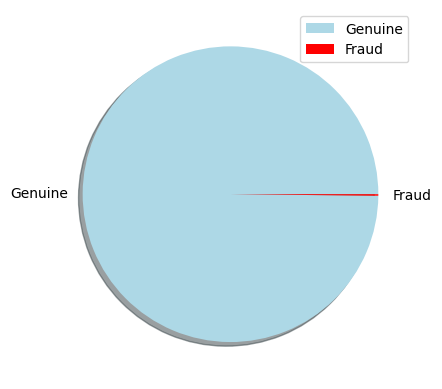
# 

**Duplicate values:**

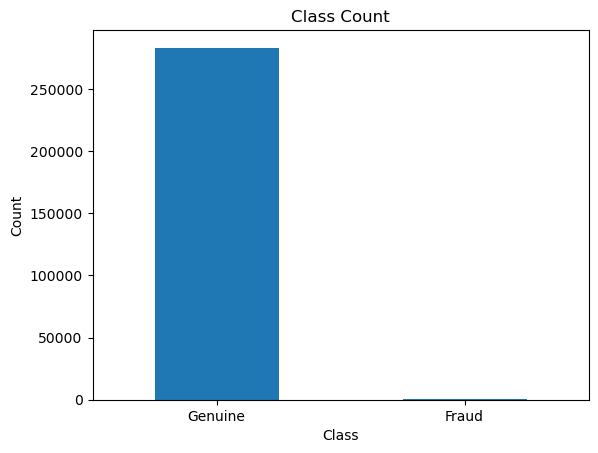
The dataset used contains duplicate values in it. After they are removed from the dataset



**Analysis of the given Data**



The pie chart represents the nonfraud and genuine transactions. Red ones are fraud blue color represent fraud transactions.



The bar-plot gives the count of Genuine and fraud transactions and we can see the that when compared to genuine fraud transactions are very less in number which is almost negligiable.So there is lot of imbalance in the dataset that we are using.

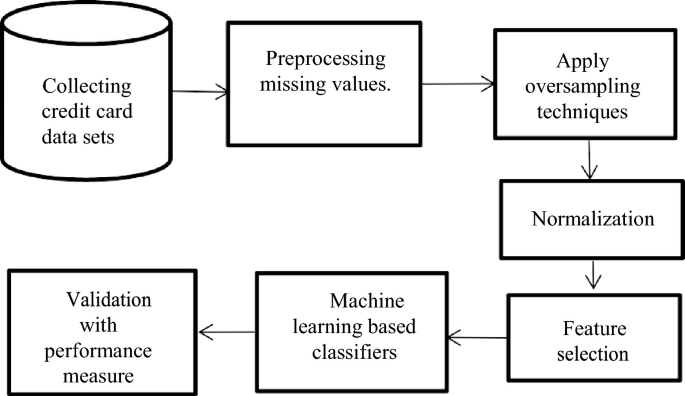
To address the class imbalance, the Synthetic Minority Over-sampling Technique (SMOTE) is applied. SMOTE generates synthetic examples by interpolating between existing instances of the minority class, creating a more balanced dataset. By oversampling the minority class, the models can learn from a wider range of fraud instances, enhancing their ability to detect fraudulent transactions.Once the dataset is balanced, suitable machine learning models such as logistic regression, decision trees, random forests, or gradient boosting algorithms are selected.

These models are trained using the resampled training data. The performance of the models is evaluated using metrics such as accuracy, precision, recall, F1-score, and AUC-ROC.The trained models are then tested on the imbalanced testing dataset to assess their real-world performance. It is important to focus on metrics like recall and precision to ensure accurate detection of fraud while minimizing false positives.

If the initial results are unsatisfactory, further steps can be taken to improve performance. This may involve adjusting model hyperparameters, exploring different resampling techniques, or considering advanced ensemble techniques like stacking or boosting.Cross-validation can be applied to obtain more reliable performance estimates by dividing the dataset into multiple folds. This helps assess the generalizability of the models and ensures their performance is not biased towards a specific subset of the data.

Comparing the results of different models and techniques based on evaluation metrics allows for the selection of the most effective approach for credit card fraud detection. Additionally, analyzing misclassified fraud cases can provide insights into model limitations and potential areas for improvement. Overall, regular monitoring and updating of the models are crucial to adapt to evolving fraud patterns and maintain the effectiveness of the fraud detection system

**CHAPTER-5 FLOWCHART**

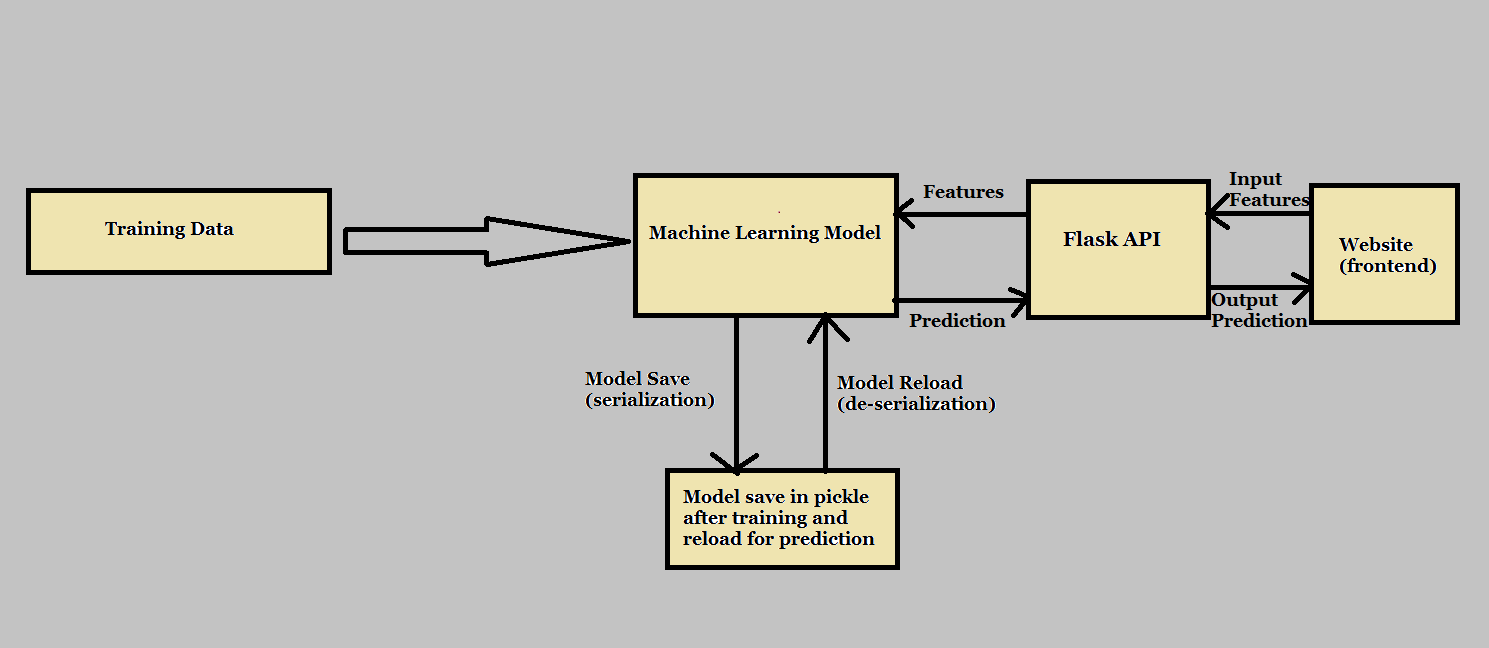


Here Oversampling Technique used is SMOTE ( **Synthetic Minority Oversampling Technique)**

**Machine learning Classifiers: Here we first tested with Decision Trees then Random Forest and finally with Logistic Regression and out of all these Random Forest has more accuracy and good metrics scores(F1,R^2 score etc.)**

**Classifier used for predicting output :** Random Forest

# Machine learning Model Deployment using Flask



Flask is a lightweight [**Web Server Gateway Interface(WSGI)**](https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface) a micro-framework written in python. This means flask provides us with tools, libraries and technologies that allow us to build a web application. This web application can be some web pages, a blog, or our machine learning model prediction web application. Flask is an intermediate medium to connect our model with front end web page for prediction .

**Why Flask?**

* Easy to use.
* Built in development server and debugger.
* Integrated unit testing support.
* RESTful request dispatching.
* Extensively documented.

# Project Structure

This project has four parts :

1. **model.py** — This contains code for the machine learning model to predict sales in the third month based on the sales in the first two months.
2. **app.py** — This contains Flask APIs that receives sales details through GUI or API calls, computes the predicted value based on our model and returns it.
3. **request.py** — This uses requests module to call APIs defined in app.py and displays the returned value.
4. **HTML/CSS** — This contains the HTML template and CSS styling to allow user to enter sales detail and displays the predicted sales in the third month.

# CHAPTER-6 RESULTS

# EXPLORATORY DATA ANALYSIS

# The real transaction have a larger mean value, larger Q1, smaller Q3 and Q4 and larger outliers; fraudulent transactions have a smaller Q1 and mean, larger Q4 and smaller outliers.

# 

## **Features density plot:**

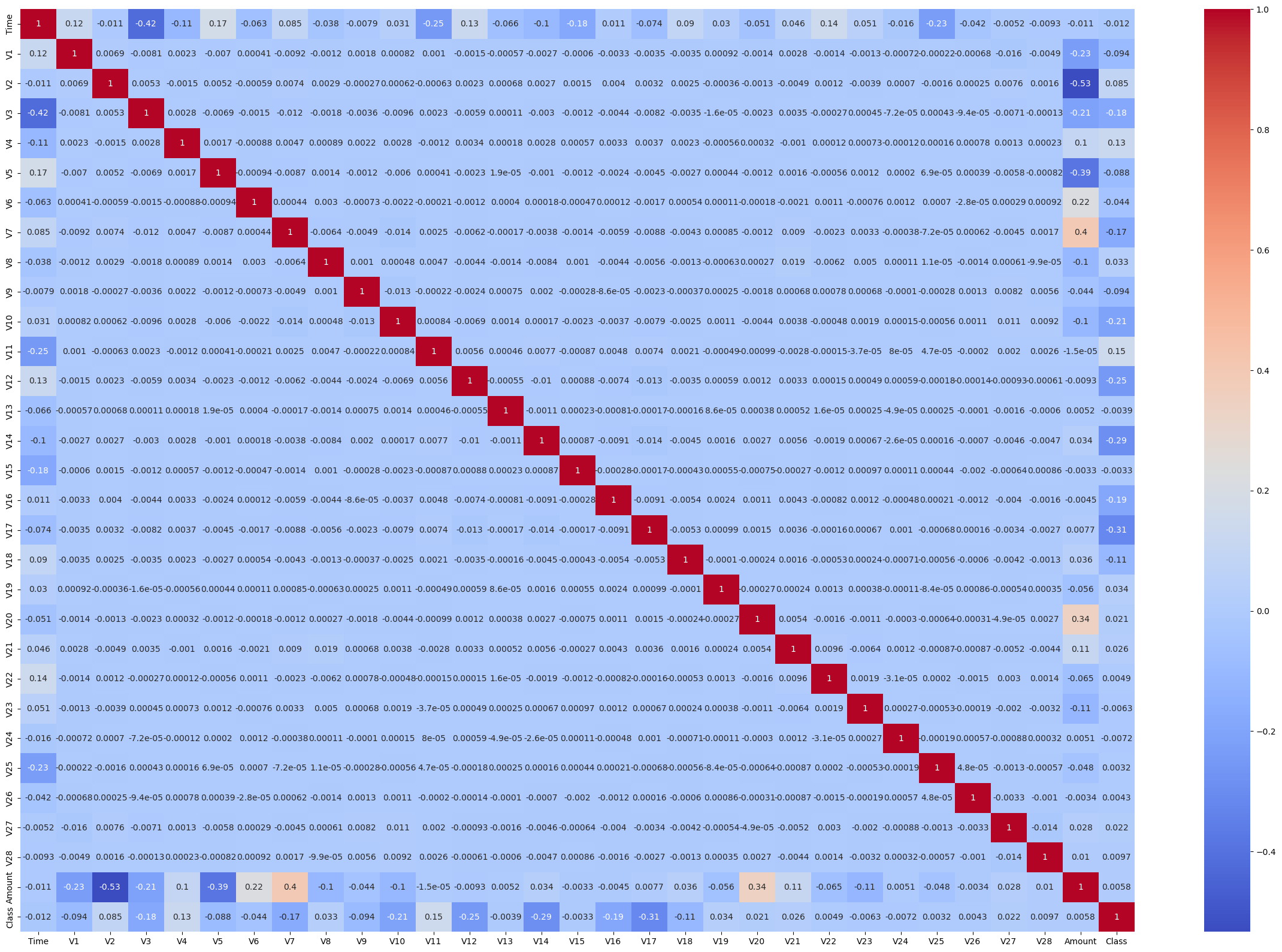
# 

For some of the features we can observe a good selectivity in terms of distribution for the two values of Class: V4, V11 have clearly separated distributions for Class values 0 and 1, V12, V14, V18 are partially separated, V1, V2, V3, V10 have a quite distinct profile, whilst V25, V26, V28 have similar profiles for the two values of Class.

In general, with just few exceptions (Time and Amount), the features distribution for legitimate transactions (values of Class = 0) is centered around 0, sometime with a long queue at one of the extremities. In the same time, the

fraudulent transactions (values of Class = 1) have a skewed (asymmetric) distribution.

**CORRELATION MARIX**

****

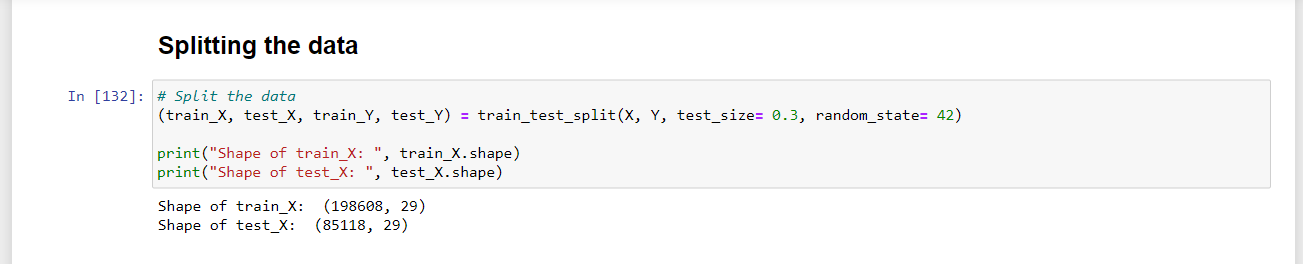
We observe that most of the data features are not correlated. This is because before publishing, most of the features were presented to a Principal Component Analysis (PCA) algorithm. The features V1 to V28 are most probably the Principal Components resulted after propagating the real features through PCA. We do not know if the numbering of the features reflects the importance of the Principal Components**.**

**Top of Form**

**SCALING**



**SPLITING THE DATA**

****

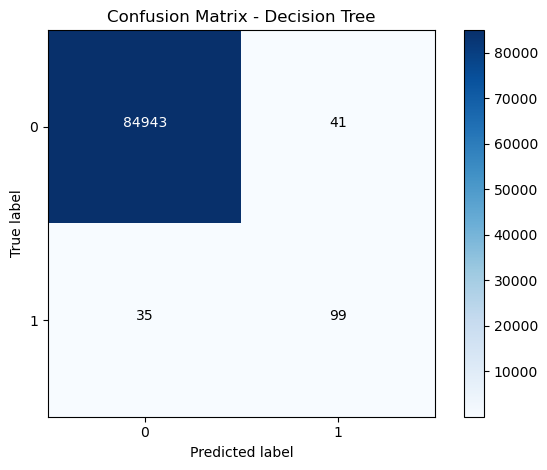
#### Let's train different models on our dataset and observe which algorithm works better for our problem.

Let's apply Random Forests , Decision Trees,Logistic Regression algorithms to our dataset.

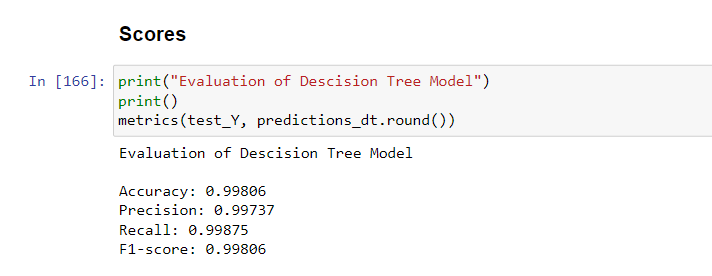
**Decision Tree Classifier**:



**CONFUSION MATRIX**

****

**SCORES:**

****

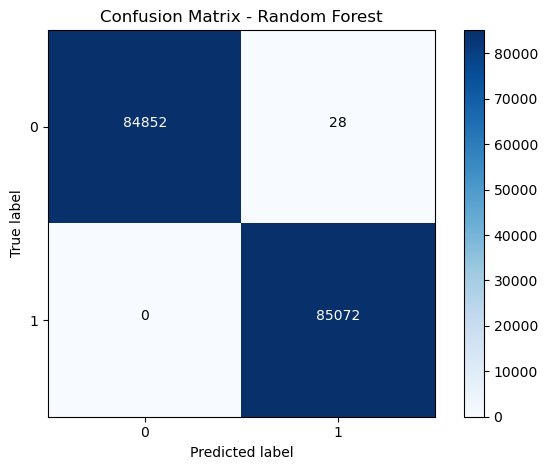
**Random Forest Classifier**

****

**Scores**

****

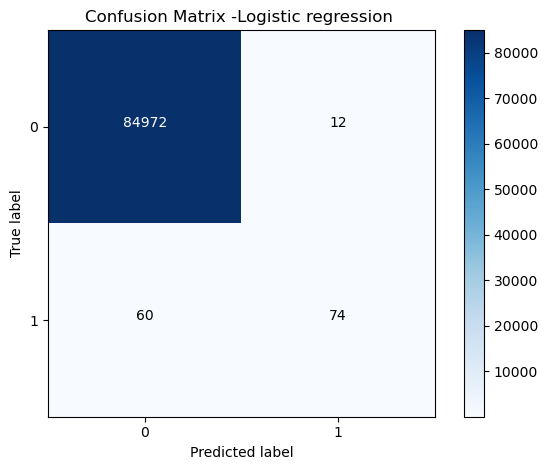
**CONFUSION MATRIX**

****

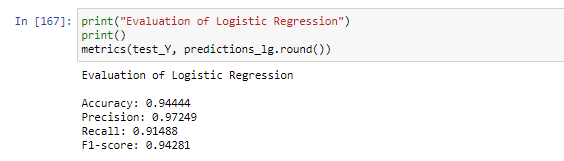
**LOGISTIC REGRESSION CLASSIFIER**

****

**CONFUSION MATRIX**

****

**SCORES**

****

### 

### Clearly, Random Forest model works better than Decision Trees and Logistic regressionHowever, if we look closely, we can see that our dataset has a major **class imbalance** issue. More than 99% of transactions are real (avoid fraud), while 0.17% of transactions are fraudulent.

If we train our model with such a distribution without considering the imbalance problems, it predicts the label with a higher value given to actual transactions (since there is more evidence about them) and so achieves more accuracy.

There are several methods that can be used to address the class imbalance issue. One of these is oversampling.

Oversampling the minority class is one way to deal with unbalanced datasets. Duplicating examples from the minority class is the simplest method, but these examples don't provide any new insight into the model.Instead, fresh examples can be created by synthesising the current ones. The **Synthetic Minority Oversampling Technique**, or **SMOTE** for short, is a technique for data augmentation for the minority class.

We here perform oversampling on Random Forest Classifier

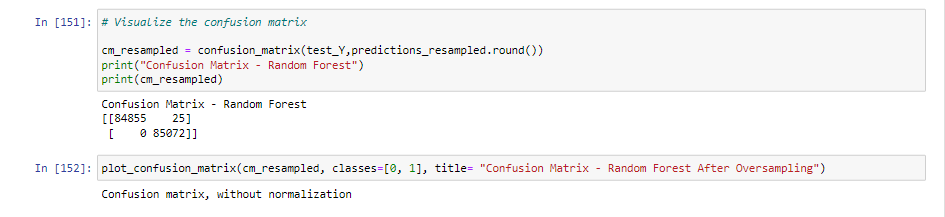


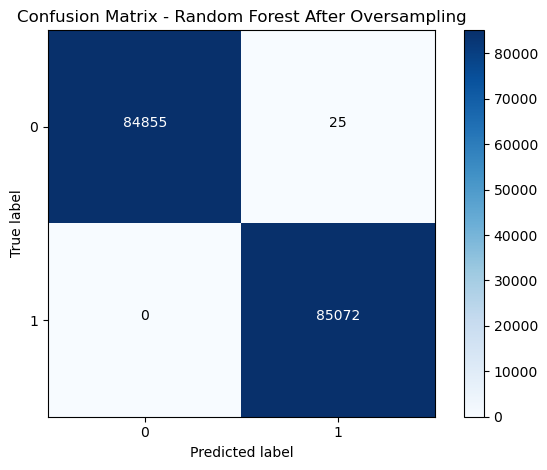
SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line.Specifically, a random example from the minority class is first chosen. Then k of the nearest neighbors for that example are found (typically k=5).

A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space.

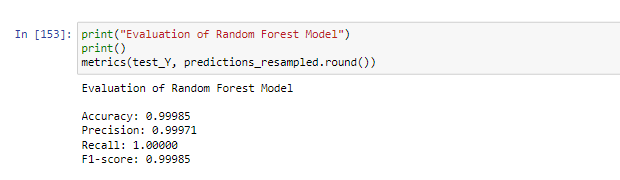
**CONFUSION MATRIX OF RANDOM FOREST CLASSIFFIER**

**AFTER APPLYING SMOTE ON DATASET**

****

****

**SCORES:**



**Now it is evident that after addressing the class imbalance problem, our Random forest classifier with SMOTE performs far better than the Random forest classifier without SMOTE.**

**CHAPTER 7**

**ADVANTAGES AND DISADVANTAGES**

**Advantages of Credit Card Fault Detection:**

Credit card fault detection systems play a crucial role in identifying and preventing fraudulent activities. By analyzing patterns and detecting anomalies in transactions, these systems can flag potential fraud attempts, allowing timely action to be taken to protect customers and financial institutions. Credit card fraud can result in significant financial losses for both consumers and businesses. Implementing an effective fault detection system can help mitigate these losses by reducing the occurrence of fraudulent transactions, thereby saving money for all stakeholders involved. Credit card fraud can lead to a loss of trust between customers and financial institutions. By actively monitoring and detecting fraud, companies can demonstrate their commitment to customer security, fostering trust and loyalty among their clientele.

A credit card fault detection system operates in real-time, allowing for immediate response and intervention. This quick reaction time helps minimize the impact of fraudulent activities, mitigating potential losses and reducing the overall damage caused by fraudsters. Implementing a credit card fault detection system generates a vast amount of transactional data. By analyzing this data, businesses can gain valuable insights into customer spending patterns, identify potential areas of improvement, and optimize their risk management strategies.

**Disadvantages of Credit Card Fault Detection:**

One of the challenges faced in credit card fault detection is the occurrence of false positives. False positives are instances where legitimate transactions are flagged as potentially fraudulent, leading to inconvenience for customers and additional manual verification processes. Striking the right balance between accuracy and false positives is a significant challenge for fault detection systems. Credit card fault detection systems rely on extensive data collection and analysis, which may raise privacy concerns among customers. Striking a balance between data security and the need for effective fraud detection is essential to ensure customer trust and compliance with privacy regulations.

Implementing a credit card fault detection system requires sophisticated algorithms and technologies. Developing, deploying, and maintaining such a system can be complex and resource-intensive, requiring expertise in data analytics, machine learning, and cybersecurity. Fraudsters continually evolve their tactics to bypass detection systems. As a result, credit card fault detection systems need to be regularly updated and enhanced to keep up with emerging fraud techniques. Staying ahead of fraudsters requires ongoing research and development efforts. Implementing an effective credit card fault detection system involves upfront costs for acquiring and integrating the necessary technology, as well as ongoing maintenance and operational expenses. These costs can be a barrier to entry for smaller businesses or institutions with limited resources.

Top of Form

**CHAPTER- 8 APPLICATIONS**

Credit card fault detection has numerous practical applications in the financial industry These applications demonstrate the wide range of benefits and potential uses of credit card fault detection in the financial industry. Some of the applications are

Fraud Detection: One of the primary applications of credit card fault detection is identifying and preventing fraudulent transactions. By analyzing various transaction parameters such as purchase amount, location, time, and customer behavior patterns, machine learning algorithms can identify suspicious activities and trigger fraud alerts. This helps financial institutions take prompt action to prevent financial losses and protect their customers.

Anomaly Detection: Credit card fault detection can be used to detect anomalies in transaction patterns. It can identify unusual spending behaviors or transactions that deviate significantly from a customer's typical spending habits. This can be helpful in identifying potential cases of stolen or compromised credit card information, as well as detecting instances of money laundering or other illicit activities.

Customer Segmentation: By analyzing credit card transaction data, fault detection techniques can help financial institutions segment their customers based on their spending patterns and behavior. This segmentation can be useful for targeted marketing campaigns, personalized offers, and providing better customer experiences.

Risk Assessment: Credit card fault detection can assist in assessing the risk associated with individual credit card accounts. By analyzing historical data and real-time transaction information, machine learning models can calculate risk scores for each account. These scores can be used by financial institutions to determine credit limits, set interest rates, and make decisions regarding credit card applications.

Credit Limit Optimization: Fault detection techniques can help financial institutions optimize credit limits for their customers. By analyzing transaction data, spending habits, and credit utilization patterns, algorithms can recommend appropriate credit limits for individual customers, ensuring they have sufficient credit while minimizing the risk of default.

Early Warning System: Credit card fault detection can serve as an early warning system for potential financial difficulties. By monitoring changes in spending patterns, transaction declines, or missed payments, financial institutions can proactively reach out to customers who may be facing financial challenges and offer assistance or alternative solutions to avoid default.

Pattern Recognition: Analyzing credit card transaction data using fault detection techniques can reveal valuable insights into consumer behavior and market trends. This information can be used for market research, product development, and business strategy planning.

Compliance and Regulatory Requirements: Fault detection in credit card transactions can help financial institutions comply with regulatory requirements, such as anti-money laundering (AML) and know-your-customer (KYC) regulations. By monitoring transactions for suspicious activities, financial institutions can ensure compliance with legal obligations and mitigate the risk of financial crimes.

Top of Form

**CHAPTER- 9 CONCLUSION**

In this credit card fraud detection Python project, we tackled the challenge of dealing with unbalanced data by using three different classifiers: decision tree, random forest, and logistic regression. After evaluating the performance of each classifier, we determined that the random forest algorithm outperformed the others and chose it as the final model.

To address the class imbalance issue, we applied the Synthetic Minority Over-sampling Technique (SMOTE). SMOTE helps to balance the dataset by generating synthetic samples for the minority class (fraudulent transactions) based on interpolation between existing minority class samples. This technique allows the classifier to learn from a more balanced representation of the data and improve its ability to detect fraudulent transactions.

After applying SMOTE and training the random forest classifier on the balanced dataset, we evaluated its performance using appropriate metrics. It is important to note that accuracy alone is not a reliable metric when dealing with imbalanced datasets, as a high accuracy can be achieved by simply predicting the majority class. Therefore, we considered additional metrics such as precision, recall, F1 score, and area under the ROC curve (AUC-ROC) to assess the model's performance.

By analyzing these evaluation metrics, we concluded that the random forest classifier, with the application of SMOTE, achieved excellent results in detecting credit card fraud transactions. The model demonstrated a high accuracy rate, indicating its ability to correctly classify both fraudulent and non-fraudulent transactions. Additionally, the precision and recall scores were also impressive, indicating that the model minimized false positives while effectively identifying actual fraud cases.

In summary, this credit card fraud detection project successfully utilized the random forest classifier along with SMOTE to address class imbalance. By considering multiple evaluation metrics, we determined that the random forest algorithm performed exceptionally well in detecting fraudulent transactions. These findings make the developed model a valuable tool for financial institutions and individuals to identify and prevent credit card fraud effectively.Top of Form

Bottom of Form

**CHAPTER -10 FUTURE SCOPE**

The credit card fraud detection project you have developed using the Random Forest algorithm and SMOTE technique has provided a solid foundation for further improvement and expansion. Here are some potential future scope areas for this project:

**Advanced Ensemble Techniques:** Explore and implement advanced ensemble techniques such as Gradient Boosting or AdaBoost to further enhance the performance of the classifier. These techniques can potentially improve the model's accuracy and robustness in detecting fraud cases.

**Feature Engineering:** Investigate additional features or feature combinations that could provide valuable insights for fraud detection. Feature engineering plays a crucial role in improving the performance of machine learning models, so exploring new ways to represent and analyze credit card transaction data can be beneficial.

**Deep Learning Models:** Consider exploring deep learning models such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs) for credit card fraud detection. Deep learning models have demonstrated significant success in various domains, and they may uncover complex patterns in credit card transaction data that are not easily captured by traditional machine learning algorithms.

**Real-Time Monitoring:** Implement a real-time monitoring system that can detect and flag potential fraud transactions as they occur. This could involve integrating the developed model into a live system that continuously processes incoming credit card transactions and alerts users or financial institutions about suspicious activities.

**Continuous Model Improvement:** Deploy a mechanism to continuously update and retrain the model as new data becomes available. As fraud patterns and techniques evolve over time, it is crucial to keep the model up to date to ensure its effectiveness in detecting emerging fraud trends.

**Collaborations with Financial Institutions**: Collaborate with banks, credit card companies, or other financial institutions to gain access to larger and more diverse datasets. Collaborative efforts can provide valuable insights, improve model generalization, and contribute to the development of more robust fraud detection systems.

**Explaining Model Decisions:** Implement techniques to explain and interpret the model's decisions. This could involve using techniques such as SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-agnostic Explanations) to gain insights into the key features and patterns that contribute to the model's fraud detection predictions.

**Deployment and Integration**: Deploy the developed model as an API or integrate it into existing fraud detection systems used by financial institutions. This would allow real-world applications and enable the model to contribute to the prevention and detection of credit card fraud at scale.

# CHAPTER-11 BIBILOGRAPHY

**References:**

* Dal Pozzolo, A., Caelen, O., Le Borgne, Y. A., Waterschoot, S., Bontempi, G. (2015). "Learned Lessons in Credit Card Fraud Detection from a Practitioner Perspective." Expert Systems with Applications, 42(10), 4716-4727.
* Bhattacharyya, S., Das, S., Kalita, J. K. (2011). "Anomaly detection in credit card transactions." In Proceedings of the International Conference on Recent Trends in Information, Telecommunication, and Computing (pp. 14-18).
* Phua, C., Lee, V., Smith-Miles, K., Gayler, R. (2010). "A Comprehensive Survey of Data Mining-based Fraud Detection Research." In Proceedings of the Australasian Conference on Data Mining and Analytics (AusDM) (pp. 7-12).
* Raza, K., Robertson, W., Apostolopoulos, A. (2017). "Credit Card Fraud Detection Using Machine Learning: A Systematic Review." Big Data Analytics, 2(1), 1-23.
* Sadiq, S., Shabut, A. M., Lasebae, A., & Luo, C. (2018). "Machine Learning-Based Credit Card Fraud Detection: A Systematic Review." Expert Systems with Applications, 114, 57-77.
* <https://www.kaggle.com/code/gpreda/credit-card-fraud-detection-predictive-models>
* <https://towardsdatascience.com/>
* <https://medium.com/@BoluwatifeDebs/credit-card-fraud-detection-analysis-5235f3276542?source=tag_page---------2-84--------------------07b7bf54_9f6c_4df6_a226_603219e16099-------17>
* <https://medium.com/@wenyue2021/credit-card-fraud-detection-f39d295f3e2a?source=tag_page---------5-84--------------------07b7bf54_9f6c_4df6_a226_603219e16099-------17>

**CODE:**

# ### Importing libraries required

# In[112]:

# Import the necessary modules

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from collections import Counter

import seaborn as sns

import itertools

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, precision\_score, confusion\_matrix, recall\_score, f1\_score

# ### Load the dataset

# Load the csv file

dataframe = pd.read\_csv("creditcard.csv")

dataframe.head()

print(dataframe.columns)

dataframe.info()

dataframe.describe()

# ### DATA QUALITY CHECK

#

#

# #### Check for NULL/MISSING values

round(100 \* (dataframe.isnull().sum()/len(dataframe)),2).sort\_values(ascending=False)

# percentage of missing values in each row

round(100 \* (dataframe.isnull().sum(axis=1)/len(dataframe)),2).sort\_values(ascending=False)

# #### Duplicate check

dataframe\_d=dataframe.copy()

dataframe\_d.drop\_duplicates(subset=None, inplace=True)

dataframe.shape

dataframe\_d.shape

# Duplicate are found in the records

#assigning duplicates removed dataset to original dataset

dataframe=dataframe\_d

dataframe.shape

# ### Exploratory Data Analysis

# In[123]:

non\_fraud = len(dataframe[dataframe.Class == 0])

fraud = len(dataframe[dataframe.Class == 1])

fraud\_percent = (fraud / (fraud + non\_fraud)) \* 100

l=[non\_fraud,fraud]

labels = ["Genuine", "Fraud"]

my\_color=['lightblue','red']

print("Number of Genuine transactions: ", non\_fraud)

print("Number of Fraud transactions: ", fraud)

print("Percentage of Genuine transactions: {:.4f}".format(100-fraud\_percent))

print("Percentage of Fraud transactions: {:.4f}".format(fraud\_percent))

plt.pie(l,labels=labels,shadow=True,colors=my\_color)

plt.legend()

plt.show()

# #### Here a pie is plotted to represent the non fraud and genuine transactions.Red ones are fraud blue color represent fraud tranctions

# In[124]:

# Visualize the "Labels" column in our dataset

labels = ["Genuine", "Fraud"]

count\_classes = dataframe.value\_counts(dataframe['Class'], sort= True)

count\_classes.plot(kind = "bar", rot = 0)

plt.title("Class Count")

plt.ylabel("Count")

plt.xticks(range(2), labels)

plt.show()

# #### The above bar graph describes the count of genuine and fraud transactions

# ### Transactions amount

# In[176]:

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(12,6))

s = sns.boxplot(ax = ax1, x="Class", y="Amount", hue="Class",data=dataframe, palette="PRGn",showfliers=True)

s = sns.boxplot(ax = ax2, x="Class", y="Amount", hue="Class",data=dataframe, palette="PRGn",showfliers=False)

plt.show();

tmp= dataframe[['Amount','Class']].copy()

class\_0 = tmp.loc[tmp['Class'] == 0]['Amount']

class\_1 = tmp.loc[tmp['Class'] == 1]['Amount']

class\_0.describe()

class\_1.describe()

# ## Features density plot

# In[168]:

var = dataframe.columns.values

i = 0

t0 = dataframe.loc[dataframe['Class'] == 0]

t1 = dataframe.loc[dataframe['Class'] == 1]

sns.set\_style('whitegrid')

plt.figure()

fig, ax = plt.subplots(8,4,figsize=(16,28))

for feature in var:

i += 1

plt.subplot(8,4,i)

sns.kdeplot(t0[feature], bw=0.5,label="Class = 0")

sns.kdeplot(t1[feature], bw=0.5,label="Class = 1")

plt.xlabel(feature, fontsize=12)

locs, labels = plt.xticks()

plt.tick\_params(axis='both', which='major', labelsize=12)

plt.show();

#Checking correlation in heatmap

plt.figure(figsize=(30,20))

corr=dataframe.corr()

sns.heatmap(corr,cmap="coolwarm",annot=True)

# Perform Scaling

scaler = StandardScaler()

dataframe["NormalizedAmount"] = scaler.fit\_transform(dataframe["Amount"].values.reshape(-1, 1))

dataframe.drop(["Amount", "Time"], inplace= True, axis= 1)

Y = dataframe["Class"]

X = dataframe.drop(["Class"], axis= 1)

X.head()

Y.head()

# Split the data

(train\_X, test\_X, train\_Y, test\_Y) = train\_test\_split(X, Y, test\_size= 0.3, random\_state= 42)

print("Shape of train\_X: ", train\_X.shape)

print("Shape of test\_X: ", test\_X.shape)

# #### Let's train different models on our dataset and observe which algorithm works better for our problem.

#

# Let's apply Random Forests , Decision Trees,Logistic Regression algorithms to our dataset.

# Decision Tree Classifier

decision\_tree = DecisionTreeClassifier()

decision\_tree.fit(train\_X, train\_Y)

predictions\_dt = decision\_tree.predict(test\_X)

decision\_tree\_score = decision\_tree.score(test\_X, test\_Y) \* 100

print("Decision Tree Score: ", decision\_tree\_score)

# ### Confusion Matrix

confusion\_matrix\_dt = confusion\_matrix(test\_Y, predictions\_dt.round())

print("Confusion Matrix - Decision Tree")

print(confusion\_matrix\_dt)

plot\_confusion\_matrix(confusion\_matrix\_dt, classes=[0, 1], title= "Confusion Matrix - Decision Tree")

print("Evaluation of Descision Tree Model")

print()

metrics(test\_Y, predictions\_dt.round())

# ## Random Forest

#

# ### Model Building

random\_forest = RandomForestClassifier(n\_estimators= 100)

random\_forest.fit(train\_X, train\_Y)

predictions\_rf = random\_forest.predict(test\_X)

random\_forest\_score = random\_forest.score(test\_X, test\_Y) \* 100

print("Random Forest Score: ", random\_forest\_score)

# ### Confusion Matrix

# Plot confusion matrix for Random Forests

confusion\_matrix\_rf = confusion\_matrix(test\_Y, predictions\_rf.round())

print("Confusion Matrix - Random Forest")

print(confusion\_matrix\_rf)

plot\_confusion\_matrix(confusion\_matrix\_rf, classes=[0, 1], title= "Confusion Matrix - Random Forest")

# ### Scores

print("Evaluation of Random Forest Model")

print()

metrics(test\_Y, predictions\_rf.round())

# ## Logistic Regression

#

# ### Model Building

from sklearn.linear\_model import LogisticRegression

log = LogisticRegression()

log.fit(train\_X,train\_Y)

predictions\_lg=log.predict(test\_X)

log\_score = log.score(test\_X, test\_Y) \* 100

print("Logistic Regression Score:",log\_score)

# ### Confusion matrix

# Plot confusion matrix for Logistic Regression

confusion\_matrix\_lg = confusion\_matrix(test\_Y, predictions\_lg.round())

print("Confusion Matrix -Logistic regression ")

print(confusion\_matrix\_lg)

plot\_confusion\_matrix(confusion\_matrix\_lg, classes=[0, 1], title= "Confusion Matrix -Logistic regression ")

### Scores

print("Evaluation of Logistic Regression")

print()

metrics(test\_Y, predictions\_lg.round())

def plot\_confusion\_matrix(cm, classes, normalize=False, title='Confusion Matrix', cmap=plt.cm.Blues):

"""

This function prints and plots the confusion matrix.

Normalization can be applied by setting `normalize=True`.

"""

if normalize:

cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

print("Normalized confusion matrix")

else:

print('Confusion matrix, without normalization')

plt.imshow(cm, interpolation='nearest', cmap=cmap)

plt.title(title)

plt.colorbar()

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=0)

plt.yticks(tick\_marks, classes)

fmt = '.2f' if normalize else 'd'

thresh = cm.max() / 2.

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center", color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.tight\_layout()

# The below function prints the following necesary metrics

def metrics(actuals, predictions):

print("Accuracy: {:.5f}".format(accuracy\_score(actuals, predictions)))

print("Precision: {:.5f}".format(precision\_score(actuals, predictions)))

print("Recall: {:.5f}".format(recall\_score(actuals, predictions)))

print("F1-score: {:.5f}".format(f1\_score(actuals, predictions)))

# ### Oversampling on RandomForest

# Performing oversampling on RF

from imblearn.over\_sampling import SMOTE

X\_resampled, Y\_resampled = SMOTE().fit\_resample(X, Y)

print("Resampled shape of X: ", X\_resampled.shape)

print("Resampled shape of Y: ", Y\_resampled.shape)

value\_counts = Counter(Y\_resampled)

print(value\_counts)

(train\_X, test\_X, train\_Y, test\_Y) = train\_test\_split(X\_resampled, Y\_resampled, test\_size= 0.3, random\_state= 42)

# Build the Random Forest classifier on the new dataset

\_rf\_resampled = RandomForestClassifier(n\_estimators = 100)

rf\_resampled.fit(train\_X, train\_Y)

predictions\_resampled = rf\_resampled.predict(test\_X)

random\_forest\_score\_resampled = rf\_resampled.score(test\_X, test\_Y) \* 100

# In[150]:

predictions\_resampled

# Visualize the confusion matrix

cm\_resampled = confusion\_matrix(test\_Y,predictions\_resampled.round())

print("Confusion Matrix - Random Forest")

print(cm\_resampled)

plot\_confusion\_matrix(cm\_resampled, classes=[0, 1], title= "Confusion Matrix - Random Forest After Oversampling")

print("Evaluation of Random Forest Model")

print()

metrics(test\_Y, predictions\_resampled.round())

a=[[1.10321543,-0.040296215,1.267332089,1.28909147,-0.735997164,0.288069163,-0.586056786,0.18937971,0.782332892,-0.267975067,-0.45031128,0.936707715,0.708380406,-0.468647288,0.354574063,-0.246634656,-0.009212378,-0.595912406,-0.575681622,-0.113910177,-0.024612006,0.196001953,0.013801654,0.103758331,0.364297541,-0.382260574,0.092809187,0.037050517,9.99

]]

print(rf\_resampled.predict(a))

import pickle

pickle.dump(rf\_resampled,open("model.pkl","wb"))