

**A Project/Dissertation ETE VIVA Report  
On**

**CREDIT CARD FRAUD DETECTION USING MACHINE LEARNING**

**Submitted in partial fulfillment of the  
requirement for the award of the degree of**

**Bachelor's of Technology  
(Computer Science Engineering)**



**Under The Supervision of  
Name of Supervisor Designation: Mrs. INDRAKUMARI**

S.NO	Enrollment No	Admission No.	Student Name	Degree/Branch	Sem
01	19021011869	19SCSE1010724	VIKRAM SINGH	B .TECH (CSE)	05
02	19021011564	19SCSE1010383	AYUSHI GANGWAR	B .TECH (CSE)	05

**SCHOOL OF COMPUTING SCIENCE AND ENGINEERING  
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING  
GALGOTIAS UNIVERSITY, GREATER NOIDA**

**INDIA**

**DECEMBER,2021**

*Indrakumari*  
**INDRAKUMARI**

## Table of Contents

Title	Page No.
<b>Abstract</b>	<b>III</b>
<b>List of Figures</b>	<b>III</b>
<b>Chapter 1 Introduction</b>	<b>VI</b>
1.1 Introduction	<b>VI</b>
1.2 Formulation of Problem	
1.2.1 Tools and Technology used	
<b>Chapter 2 Literature Survey/Project Design</b>	<b>IX</b>

## List of Figures

Figure No.	Figure Name	Page No.
1.	Count of fraudulent vs. Non-fraudulent transactions	VI
2.	Distribution of time feature	VI
3.	Distribution of Monetary Value	VII
4.	Database & Algorithm	VII
5.	Heatmap of Correlation	VIII

# Abstract

Financial fraud is an ever-growing menace with far consequences in the financial industry. Data mining had played an imperative role in the detection of credit card fraud in online transactions. Credit card fraud detection, which is a data mining problem, becomes challenging due to two major reasons - first, the profiles of normal and fraudulent behaviour change constantly and secondly, credit card fraud data sets are highly skewed. The performance of fraud detection in credit card transactions is greatly affected by the sampling approach on dataset, selection of variables and detection technique(s) used. This investigates the performance of naïve bayes, k-nearest neighbor and logistic regression on highly skewed credit card fraud data. Dataset of credit card transactions is sourced from European cardholders containing 284,807 transactions. A hybrid technique of under-sampling and oversampling is carried out on the skewed data. The three techniques are applied on the raw and preprocessed data. The work is implemented in Python. The performance of the techniques is evaluated based on accuracy, sensitivity, specificity, precision, Matthew's correlation coefficient and balanced classification rate. The results show of optimal accuracy for naïve bayes, k-nearest neighbor and logistic regression classifiers are 97.92%, 97.69% and 54.86% respectively. The comparative results show that k-nearest neighbour performs better than naïve bayes and logistic regression techniques.

Keywords : Credit Card Fraud Detection Using Machine Learning

# 1. Introduction

'Fraud' in credit card transactions is unauthorized and unwanted usage of an account by someone other than the owner of that account. Necessary prevention measures can be taken to stop this abuse and the behaviour of such fraudulent practices can be studied to minimize it and protect against similar occurrences in the future. In the other words, Credit Card Fraud can be defined as a case where a person uses someone else's credit card for personal reasons while the owner and the card issuing authorities are unaware of the fact that the card is being used.

Fraud detection involves monitoring the activities populations of users in order to estimate, perceive or avoid objectionable behaviour, which consist of fraud, intrusion, and defaulting.

This is a very relevant problem that demands the attention of communities such as machine learning and data science where the solution to this problem can be automated.

This problem is particularly challenging from the perspective of learning, as it is characterized by various factors such as class imbalance. The number of valid transactions far outnumber fraudulent ones. Also, the transaction patterns often change their statistical properties over the course of time. These are not only challenges in the implementation of a real-world fraud detection system, however. In real world examples, the massive stream of payment requests is quickly scanned by automatic tools that determine which transactions to authorize. Credit card frauds are easy targets. Without any risks, a significant amount can be withdrawn without the owner's knowledge, in a short period. Fraudsters always try to make every fraudulent transaction legitimate, which makes fraud detection very challenging and difficult task to detect.

## 2. Formulation of Problem

Machine learning algorithms are employed to analyse all the authorized transactions and report the suspicious ones. These reports are investigated by professionals who contact the cardholders to confirm if the transaction was genuine or fraudulent.

The investigators provide a feedback to the automated system which is used to train and update the algorithm to eventually improve the fraud-detection performance over time.

Fraud detection methods are continuously developed to defend criminals in adapting to their fraudulent strategies. These frauds are classified as:

- Credit Card Frauds: Online and Offline
- Card Theft
- Account Bankruptcy
- Device Intrusion
- Application Fraud
- Counterfeit Card
- Telecommunication Fraud

With different frauds mostly credit card frauds, often in the news for the past few years, frauds are in the top of mind for most of the world's population. Credit card dataset is highly imbalanced because there will be more legitimate transactions when compared with a fraudulent one. As advancement, banks are moving to EMV cards, which are smart cards that store their data on integrated circuits

rather than on magnetic stripes, have made some on-card payments safer, but still leaving card-not-present frauds on higher rates.

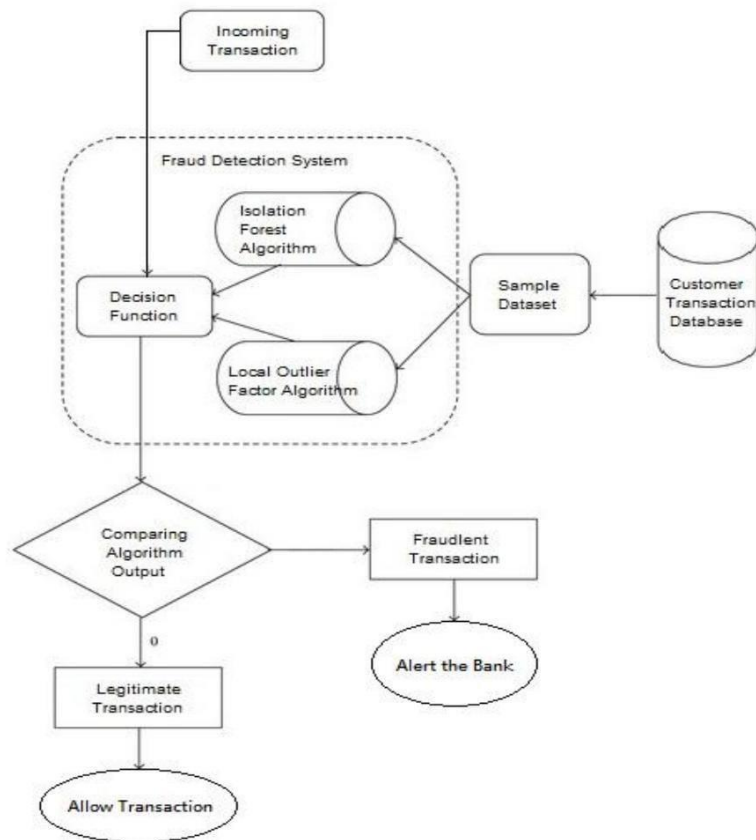


Figure 1: Working Process

Fraud detection methods are continuously developed to defend criminals in adapting to their fraudulent strategies. These frauds are classified as:

- Credit Card Frauds: Online and Offline
- Card Theft
- Account Bankruptcy
- Device Intrusion
- Application Fraud
- Counterfeit Card
- Telecommunication Fraud

Some of the currently used approaches to detection of such fraud are:

- Artificial Neural Network
- Fuzzy Logic
- Genetic Algorithm
- Logistic Regression
- Decision tree
- Support Vector Machines
- Bayesian Networks
- Hidden Markov Model
- K-Nearest Neighbour

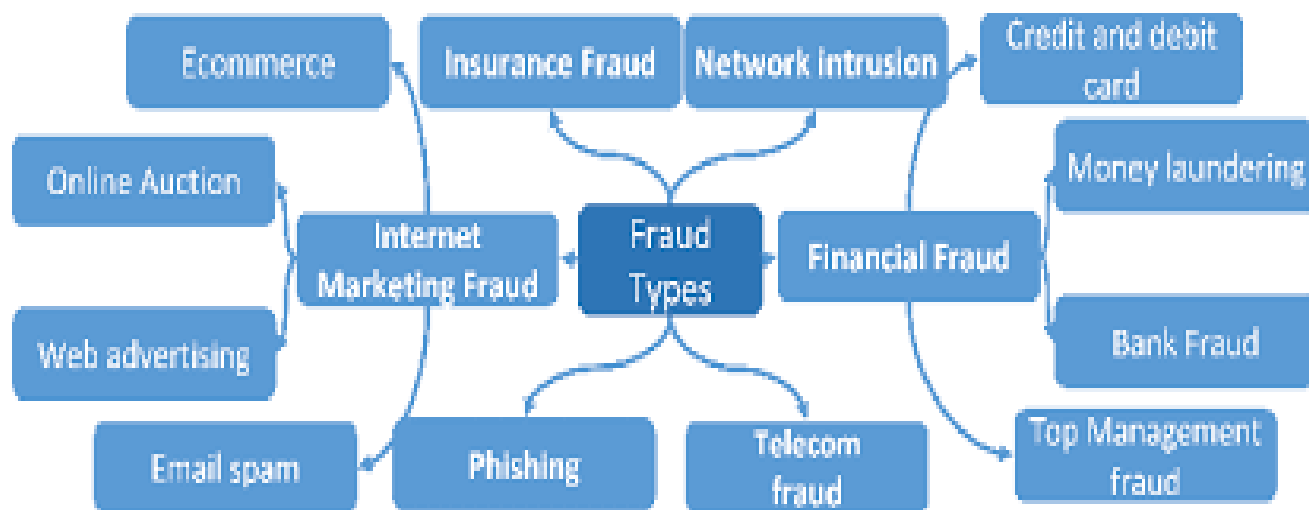


Fig. . Taxonomy of Fraud

### 3. Literature Survey

Literature Survey Multiple Supervised and Semi-Supervised machine learning techniques are used for fraud detection, but we aim is to overcome three main challenges with card frauds related dataset i.e., strong class imbalance, the inclusion of labelled and un-labelled samples, and to increase the ability to process a large number of transactions. Different Supervised machine learning algorithms like Decision Trees, Naive Bayes Classification, Least Squares Regression, Logistic Regression and SVM are used to detect fraudulent transactions in real-time datasets. Two methods under random forests are used to train the behavioural features of normal and abnormal transactions. They are Random-tree-based random forest and CART-based. Even though random forest obtains good results on small set data, there are still some problems in case of imbalanced data. The future work will focus on solving the above-mentioned problem. The algorithm of the random forest itself should be improved. Performance of Logistic Regression, K-Nearest Neighbour, and Naïve Bayes are analysed on highly skewed credit card fraud data where Research is carried out on examining meta-classifiers and meta-learning approaches in handling highly imbalanced credit card fraud data. Through supervised learning methods can be used there may fail at certain cases of detecting the fraud cases. A model of deep Auto-encoder and restricted Boltzmann machine (RBM) that can construct normal transactions to find anomalies from normal patterns. Not only that a hybrid method is developed with a combination of Adaboost and Majority Voting methods.



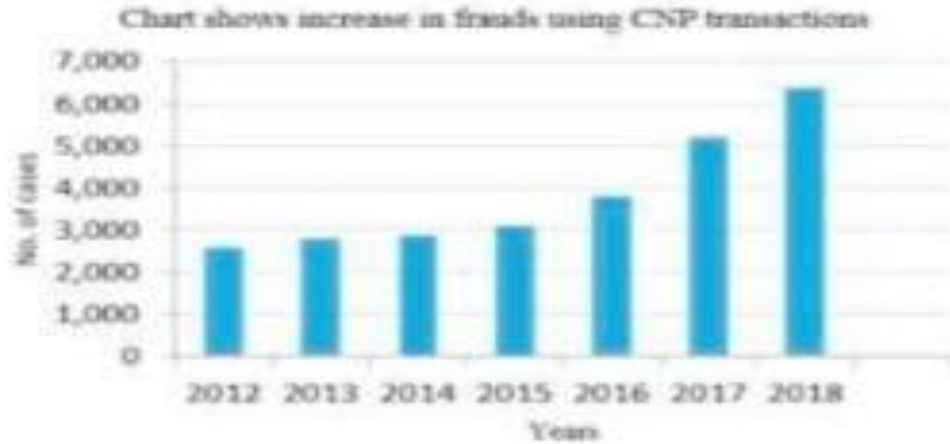


Fig.2. Frauds Using Card Not Present Transactions

With different frauds mostly credit card frauds, often in the news for the past few years, frauds are in the top of mind for most the world's population. Credit card dataset is highly imbalanced because there will be more legitimate transaction when compared with a fraudulent one. As advancement, banks are moving to EMV cards, which are smart cards that store their data on integrated circuits rather than on magnetic stripes, have made some on-card payments safer, but still leaving card-not-present frauds on higher rates

## 4. Proposed Model

Card transactions are always unfamiliar when compared to previous transactions made the customer. This unfamiliarity is a very difficult problem in real-world when are called concept drift problems . Concept drift can be said as a variable which changes over time and in unforeseen

ways. These variables cause a high imbalance in data. The main aim of our research is to overcome the problem of Concept drift to implement on real-world scenario.

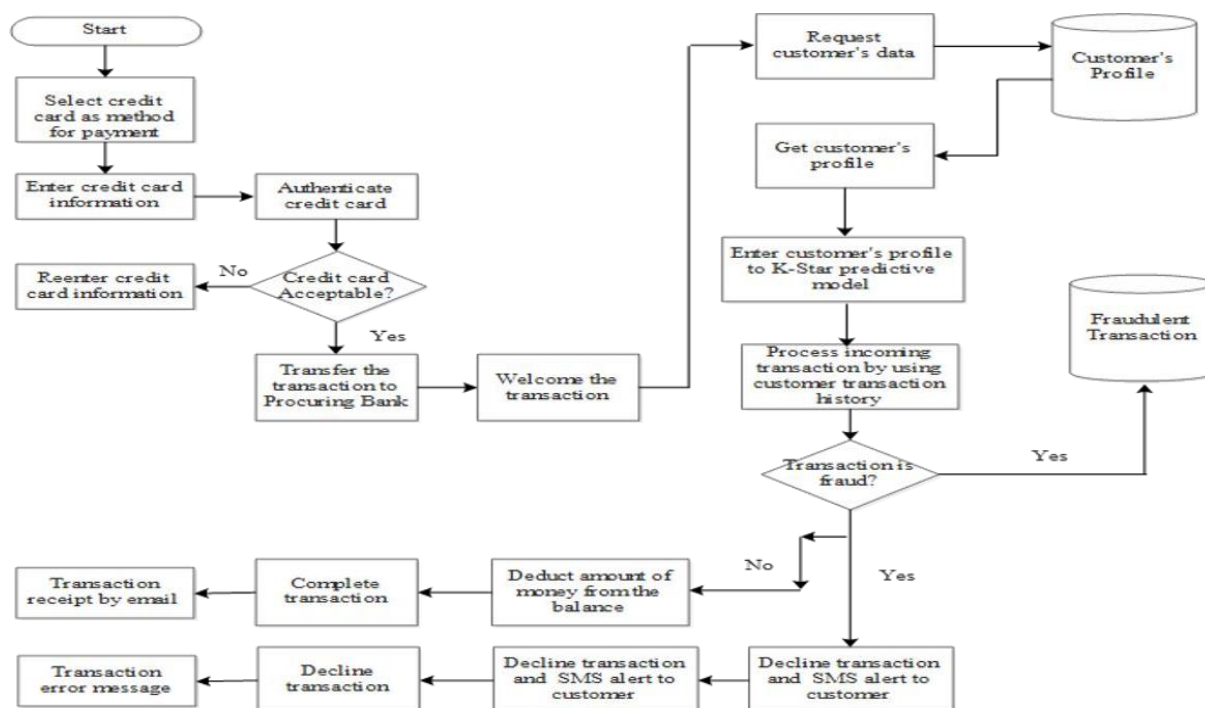


Fig.3. Model of Work

Fraud act as the unlawful or criminal deception intended to result in financial or personal benefit. It is a deliberate act that is against the law, rule or policy with an aim to attain unauthorized financial benefit.

Numerous literatures pertaining to anomaly or fraud detection in this domain have been published already and are available for public usage. A comprehensive survey conducted by Clifton Phua

and his associates have revealed that techniques employed in this domain include data mining applications, automated fraud detection, adversarial detection. In another paper, Suman, Research Scholar, GJUS&T at Hisar HCE presented techniques like Supervised and Unsupervised Learning for credit card fraud detection. Even though these methods and algorithms fetched an unexpected success in some areas, they failed to provide a permanent and consistent solution to fraud detection.

A similar research domain was presented by Wen-Fang YU and Na Wang where they used Outlier mining, Outlier detection mining and Distance sum algorithms to accurately predict fraudulent-transaction in an emulation experiment of credit card transaction data set of one certain commercial bank. Outlier mining is a field of data mining which is basically used in monetary and internet fields. It deals with detecting objects that are detached from the main system I.e., the transactions that aren't genuine. They have taken attributes of customer's behaviour and based on the value of those attributes they've calculated that distance between the observed value of that attribute and its predetermined value.

Unconventional techniques such as hybrid data mining/complex network classification algorithm is able to perceive illegal instances in an actual card transaction data set, based on network reconstruction algorithm that allows creating representation of the deviation of one instance from a reference group have proved efficient typically on medium sized online transaction.

There have also been efforts to progress from a completely new aspect. Attempts have been made to improve the alert-feedback interaction in case of fraudulent transaction.

In case of fraudulent transaction, the authorized system would be alerted and feedback would be sent to deny the ongoing transaction.

Artificial Genetic Algorithm, one of the approaches that shed new light in this domain, countered fraud a different direction.

It proved accurate in finding out the fraudulent transactions and minimizing the number of false alerts. Even though, it was accompanied by classification problem with variable misclassification costs.

## **5. Design and Implementation of Algorithm**

The procedure which we followed to predict the result are understanding problem statement and data by performing statistical analysis and visualization then checking whether the data is balance or not, In this data set the data is imbalanced, balanced by using oversampling, then scaling the data using standardization and normalization and testing data with different ML algorithms For any data science project some package are very important such as Numpy that is numeric python And pandas and for visualization of the data, matplotlib and seaborn is used which build n matplotlib with some extra features.

Anaconda navigator is used as it is having several IDEs installed in it python programming language is used to implement machine learning algorithms as it is easy to learn and implement. In this project Jupyter notebook is used to process the complete code where the code can be viewed as block of codes and running each section and identifying the errors is easier. User interface to train and test the algorithms is implemented using python Tkinter module. Test and train buttons are given to train or test the data.

Fraud can be committed in different ways and in many industries. The majority of detection methods combine a variety of fraud detection datasets to form a connected overview of both valid and non-valid payment data to make a decision. This decision must consider IP address,

geolocation, device identification, “BIN” data, global latitude/longitude, historic transaction patterns, and the actual transaction information. In practice, this means that merchants and issuers deploy analytically based responses that use internal and external data to apply a set of business rules or analytical algorithms to detect fraud.

## A. Machine learning algorithms

### **1. Logistic Regression:**

Logistic regression works with sigmoid function because the sigmoid function can be used to classify the output that is dependent feature and it uses the probability for classification of the dependent feature.

This algorithm works well with less amount of data set because of the use of sigmoid function if value the of sigmoid function is greater than 0.5 the output will 1 if the output the sigmoid function is less than 0.5 then the output is considered as the 0. But this sigmoid function is not suitable for deep learning because the if deep learning when we back tracking from the output to input we have to update the weights to minimize the error in weight update. we have to do differentiation of sigmoid activation function in middle layer neuron then results in the value of 0.25 this will affect the accuracy of the module in deep learning.

### **2. Decision Tree:**

Decision tree can be used for the classification and regression problems working for both is same but some formulas will change. Classification problem uses the entropy and information gain for

the building of the decision tree model. entropy tell about how the data is random and information gain tells about how much information we can get from this feature.

Regression problem uses the gini and gini index for the building of the decision tree model. In classification problems the root node is selected by using information gain that the root node t id selected by using is having the high information again and low entropy. In Regression problems the root node is selected by using gini , the feature which is having the less gini is selected as the root here Depth of the tree can be determined

by using hyper parameter optimization, this can be achieved by Using grid search cv algorithm.

### **3. Random Forest:**

The random forest randomly selects the features that is independent variables and also randomly selects the rows by row sampling and the number of decision tree can be determined by using hyper parameter optimization. For classification problem statement the output is the maximum occurrence outputs from each decision tree models inside the random forest. This is one the widely used machine learning algorithm in real word scenarios and in deployed models. And in most of the Kaggle computation challenges this algorithm is used to solve the problem statement.

## **6. Results and Discussions**

Fig.6.1 shows the user interface for test and train the data. Train and Test buttons are given to the user where using train the algorithms are trained and then o predict the fraud by clicking predict

button it will take to another window where the input is given and output is seen as fraud or nonfraud.

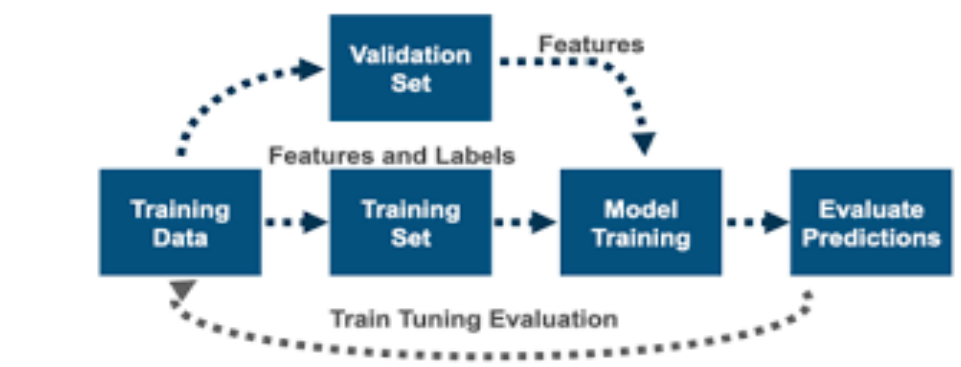


Fig. 6.1: User interface for train and test dat

The figure 6.2 shows detection of fraud or nonfraud transaction. when predict button is clicked it will take to another window where it asks for data which is input to the machine learning algorithms and in the predict it will give output as fraud or nonfraud. comma separated 30 values are given including amount and time. Predicted result is displayed as fraud after providing the data. These results along with the classication report for each algorithm is given in the output as follows, where class 0 means the transaction was determined to be valid and 1 means it was determined as a fraud transaction.

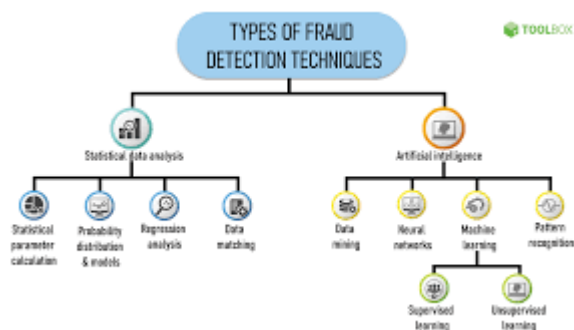


Fig. 6.2: Detection of fraud or normal transaction

1) Confusion matrix for Logistic regression Algorithm:

Fig 6.3 represents confusion matrix for Logistic regression algorithm. Which contains True Positive, True Negative, False Positive, False Negative. False positive value is lesser which shows fraud not detected cases are low. For logistic regression algorithm accuracy, recall, precision achieved are 94.84, 92.00, 97.58 respectively.

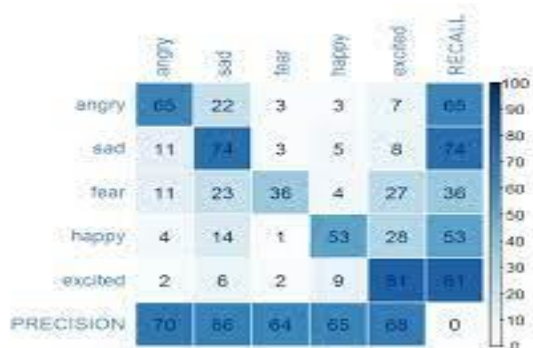


Fig. 6.3: Confusion matrix for Logistic regression

## 7. Methodology

- Firstly, we use clustering method to divide the cardholders into different clusters/groups based on their transaction amount, i.e., high, medium and low using range partitioning.
- Using Sliding-Window method, we aggregate the transactions into respective groups, i.e., extract some features from window to find cardholder's behavioural patterns. Features like maximum amount, minimum amount of transaction, followed by the average amount in the window and even the time elapsed.
- Every time a new transaction is fed to the window the old ones are removed and step-2 is processed for each group of transactions.
- After pre-processing, we train different classifiers on each group using the cardholders behavioural patterns in that group and extract fraud features. Even when we apply classifiers on the dataset, due to imbalance in the dataset, the classifiers do not work well on the dataset.



- Thus, we perform SMOTE (Synthetic Minority Over-Sampling Technique) operation on the dataset.
- Oversampling does not provide any good results.
- Thus, there are two different ways of dealing with imbalance dataset i.e., consider Matthew Coefficient Correlation of the classifier on the original dataset or we make use of one-class classifiers.
- Finally, the classifier that is used for training the group is applied to each cardholder in that group. The classifier with highest rating score is considered as cardholder's recent behavioural pattern.

## PROGRAM

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score

credit_card_data = pd.read('/content/creditcard.csv')

# first 5 rows of the data

credit_card_data.head()
```

Credit Card Fraud Detection Using ML.ipynb

```

NameError: name 'pd' is not defined

# first 5 rows of the data
credit_card_data.head()

```

	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	-0.617801	-0.991390	-0.311169	1.466177	-0.470401	0.207971	0.025791	0.403993	0.251412	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	1.065235	0.489095	-0.143772	0.635558	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	2.69	0
2	0.060084	0.717293	-0.165946	2.345865	-2.890083	1.109969	-0.121359	-2.261857	0.524980	0.247988	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055303	-0.059752	378.66	0
3	0.178228	0.507757	-0.287924	-0.631418	-1.059647	-0.684093	1.965775	-1.232622	-0.208038	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	0.538196	1.345852	-1.119670	0.175121	-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	69.99	0

credit\_card\_data.tail()

Credit Card Fraud Detection Using ML.ipynb

```

NameError: name 'pd' is not defined

# first 5 rows of the data
credit_card_data.head()

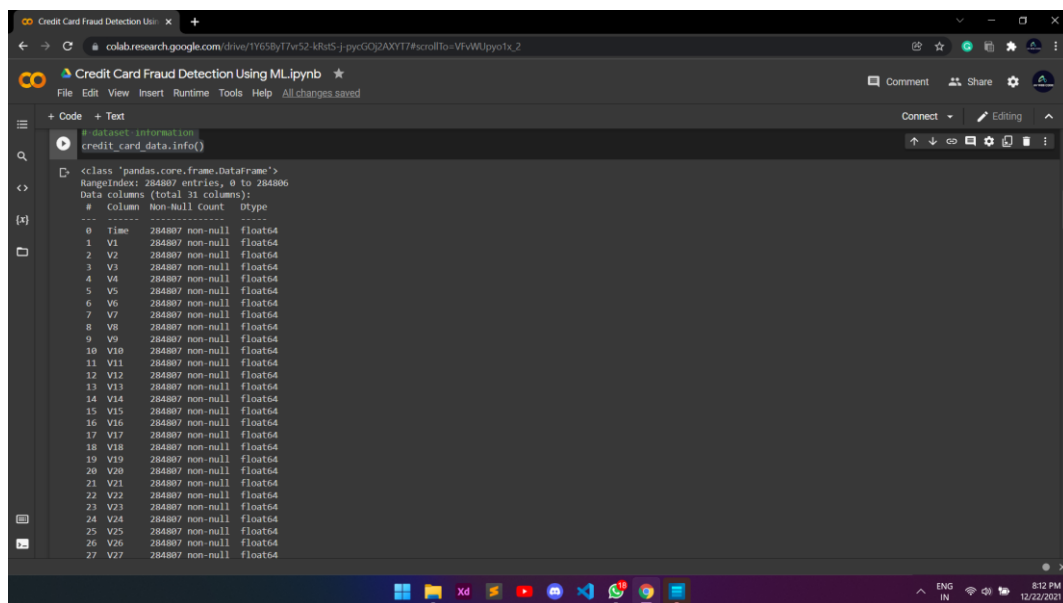
credit_card_data.tail()

```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	-1.593105	2.711941	-0.689256	4.626942	-0.924459	1.107641	1.991
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	-0.150189	0.915802	1.214756	-0.675143	1.164931	-0.711757	-0.026
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	-0.484782	0.411614	0.063119	-0.183699	-0.510602	1.329284	0.140716	0.318
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	-0.399126	-1.933849	-0.962886	-1.042082	0.449624	1.962563	-0.608577	0.505
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	-1.040458	-0.031513	-0.188093	-0.084316	0.041333	-0.302820	-0.660

# dataset information

credit\_card\_data.info()



```
# dataset information
credit_card_data.info()

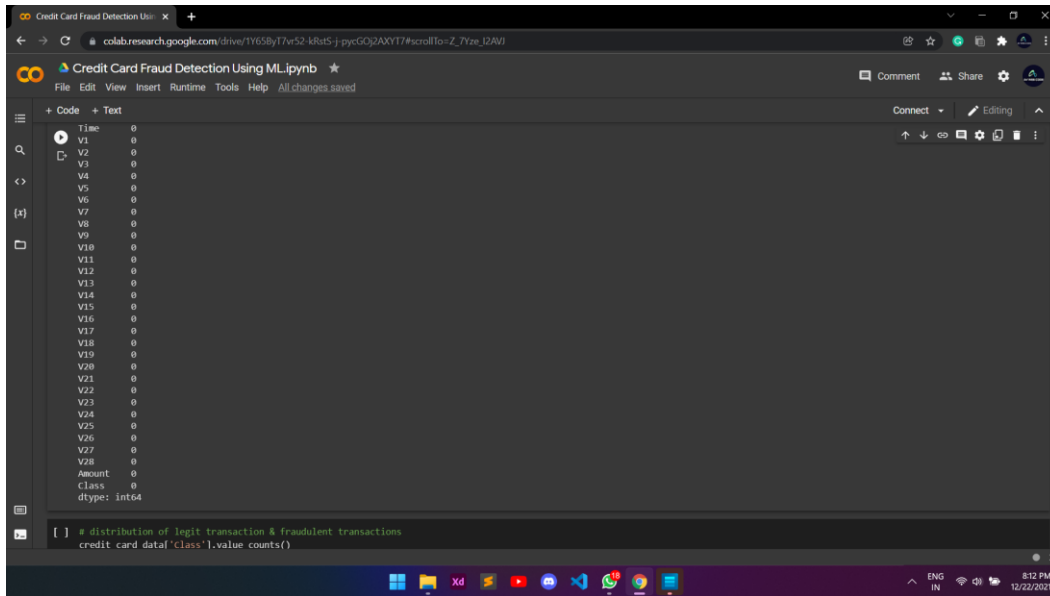
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284887 entries, 0 to 284886
Data columns (total 28 columns):
 #   Column      Non-Null Count  Dtype
---  -
 0   Time       284887 non-null float64
 1   V1         284887 non-null float64
 2   V2         284887 non-null float64
 3   V3         284887 non-null float64
 4   V4         284887 non-null float64
 5   V5         284887 non-null float64
 6   V6         284887 non-null float64
 7   V7         284887 non-null float64
 8   V8         284887 non-null float64
 9   V9         284887 non-null float64
10  V10        284887 non-null float64
11  V11        284887 non-null float64
12  V12        284887 non-null float64
13  V13        284887 non-null float64
14  V14        284887 non-null float64
15  V15        284887 non-null float64
16  V16        284887 non-null float64
17  V17        284887 non-null float64
18  V18        284887 non-null float64
19  V19        284887 non-null float64
20  V20        284887 non-null float64
21  V21        284887 non-null float64
22  V22        284887 non-null float64
23  V23        284887 non-null float64
24  V24        284887 non-null float64
25  V25        284887 non-null float64
26  V26        284887 non-null float64
27  V27        284887 non-null float64
```

# checking the number of missing value in each column

```
credit_card_data.isnull().sum()
```

# distribution of legit transaction & fraudulent transactions

```
credit_card_data['Class'].value_counts()
```



The screenshot shows a Jupyter Notebook titled "Credit Card Fraud Detection Using ML.ipynb". The code cell contains the following text:

```
Time 0
V1 0
V2 0
V3 0
V4 0
V5 0
V6 0
V7 0
V8 0
V9 0
V10 0
V11 0
V12 0
V13 0
V14 0
V15 0
V16 0
V17 0
V18 0
V19 0
V20 0
V21 0
V22 0
V23 0
V24 0
V25 0
V26 0
V27 0
V28 0
Amount 0
Class 0
dtype: int64
```

Below the list, there is a comment and a code line:

```
[ ] # distribution of legit transaction & fraudulent transactions
credit_card_data['Class'].value counts()
```

The notebook interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a toolbar (Connect, Editing), and a Windows taskbar at the bottom showing the time as 8:18 PM on 12/22/2021.

#separating the data for analysis

```
legit = credit_card_data[credit_card_data. Class == 0]
```

```

V23      0
V24      0
V25      0
V26      0
V27      0
V28      0
Amount   0
Class    0
dtype: int64

#distribution of legit transaction & fraudulent transactions
credit_card_data['class'].value_counts()

0      284315
1         492
Name: class, dtype: int64

This dataset is highly unbalanced.
0 -> Normal Transactions
1 -> Fraudulent Transactions

#separating the data for analysis
legit = credit_card_data[credit_card_data. Class == 0]
fraud = credit_card_data[credit_card_data. Class == 1]

print(legit.shape)
print(fraud.shape)

(284315, 31)
(492, 31)

```

```
fraud = credit_card_data[credit_card_data. Class == 1]
```

```
print(legit.shape)
```

```
print(fraud.shape)
```

```
(284315, 31) (492, 31)
```

```
#statistical measures of the data
```

```
legit.Amount.describe()
```

The screenshot shows a Jupyter Notebook with two code cells. The first cell runs `legit.Amount.describe()` and displays the following statistics:

```

count    284315.000000
mean      88.291022
std       250.105992
min        0.000000
25%        5.650000
50%       22.000000
75%       77.050000
max      25691.160000
Name: Amount, dtype: float64

```

The second cell runs `fraud.Amount.describe()` and displays the following statistics:

```

count     492.000000
mean     122.211321
std      256.683288
min       0.000000
25%       1.000000
50%       9.250000
75%      105.890000
max     2125.870000
Name: Amount, dtype: float64

```

Below the code cells, a table shows the mean values for each variable across different classes:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
Class																	
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	0.009824	-0.006576	0.010832	0.000189	0.012064	0.000161	0.007164

`fraud.Amount.describe()`

`# compare the values for the both transactions`

`credit_card_data.groupby('Class').mean()`

The screenshot shows a Jupyter Notebook with two code cells. The first cell runs `credit_card_data.groupby('Class').mean()` and displays the following statistics:

```

count     492.000000
mean     122.211321
std      256.683288
min       0.000000
25%       1.000000
50%       9.250000
75%      105.890000
max     2125.870000
Name: Amount, dtype: float64

```

The second cell runs `credit_card_data.groupby('Class').mean()` and displays a table with two rows of mean values for each variable across different classes:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
Class																	
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	0.009824	-0.006576	0.010832	0.000189	0.012064	0.000161	0.007164
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	-5.676883	3.800173	-6.259393	-0.109334	-6.971723	-0.092929	-4.139946

Below the table, there is a section titled "Under Sampling" with the following text:

Build a sample dataset containing similar distribution of legit & fraudulent transaction

Number of Fraudulent Transaction -> 56

The final code cell runs `legit_sample = legit.sample(n=492)`.

## Under Sampling

Build a sample dataset containing similar distribution of legit & fraudulent transaction

Number of Fraudulent Transaction --> 56

```
legit_sample = legit.sample(n=492)
```

-----

NameErrorTraceback (most recent call last)

```
<ipython-input-5-11c6ac7e9cdd>in <module>()---->
```

```
1legit_sample=legit.sample(n=492)
```

```
NameError: name 'legit' is not defined
```

## Concatening two Data Frames

```
new_dataset = pd.concat([legit_sample,fraud],axis=0)
```

The screenshot shows a Jupyter Notebook titled "Credit Card Fraud Detection Using ML.ipynb". The code cell contains the following code:

```
#statistical measures of the data
legit.Amount.describe()

fraud.Amount.describe()

#compare the values for the both transactions
credit_card_data.groupby('class').mean()
```

The output for `legit.Amount.describe()` is:

```
count    284315.000000
mean      88.291022
std       259.105992
min         0.000000
25%        5.650000
50%       22.000000
75%       77.050000
max      25691.160000
Name: Amount, dtype: float64
```

The output for `fraud.Amount.describe()` is:

```
count      492.000000
mean     122.211321
std     256.683288
min       0.000000
25%       1.000000
50%       9.250000
75%     105.890000
max     2125.870000
Name: Amount, dtype: float64
```

The output for `credit_card_data.groupby('class').mean()` is a table with columns: Class, Time, V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16, and a final column with values 0.0.

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	0.009824	-0.006576	0.010832	0.000189	0.012064	0.000161	0.007164	0.0

new\_dataset.head()

The screenshot shows a Google Colab notebook titled "Credit Card Fraud Detection Using ML.ipynb". The code cell contains the following Python code:

```

[ ] <ipython-input-5-11c6ac7e9cdd> in <module>()
----> 1 legit_sample = legit.sample(n=492)
NameError: name 'legit' is not defined
SEARCH STACK OVERFLOW

Concatenating two Data Frames

[ ] new_dataset = pd.concat([legit_sample, fraud], axis=0)

[ ] new_dataset.head()

```

The output of `new_dataset.head()` is a DataFrame with 5 rows and 17 columns. The columns are labeled Time, V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16, and V17. The rows contain numerical values for each feature.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17
196088	131356.0	2.091936	0.148705	-3.738184	0.159139	3.386352	2.869366	0.462437	0.467049	-0.437896	0.415274	-0.132049	0.132293	-0.441097	1.110540	0.076008	-0.935165	-0.3006
254893	156965.0	1.978648	-1.555866	-1.887961	-1.881698	1.130933	3.709389	-1.619579	0.953556	-0.045954	0.744668	0.037306	-0.481764	0.230474	-0.272885	0.996942	1.312164	-0.1086
49762	44199.0	1.351546	-1.390759	0.713643	-1.244118	-2.008408	-0.801667	-1.222513	-0.101894	-1.449164	1.422011	-0.688937	-1.450402	-0.426721	-0.276468	0.962478	-0.091587	0.4346
159281	112390.0	-0.467520	-4.475601	-2.316311	0.869426	-0.645675	2.233655	0.848250	0.294705	0.787210	-0.591245	0.724197	1.021004	-0.221404	0.457611	0.190564	-0.196274	0.1223
19327	30190.0	-1.416985	-1.932028	1.321694	-2.173934	0.324299	-1.484920	-1.988174	-0.723678	-1.443325	1.167898	-1.552772	-1.898673	-0.880261	-0.323279	0.265376	0.335234	-0.0377

The code cell also shows `new_dataset.tail()` at the bottom, which displays the last few rows of the dataset.

new\_dataset.tail()

The screenshot shows a Google Colab notebook titled "Credit Card Fraud Detection Using ML.ipynb". The code cell contains the following Python code:

```

[ ] 280149 169351.0 -0.676143 1.126366 -2.213700 0.468308 -1.120541 -0.003346 -2.234739 1.210158 -0.652250 -3.463891 1.794969 -2.775022 -0.418950 -4.057162 -0.712616 -1.603015 -5.03532
281144 169966.0 -3.113832 0.585884 -5.399730 1.817092 -0.840618 -2.943548 -2.208002 1.058733 -1.632333 -5.245984 1.933520 -5.030465 -1.127455 -6.416628 0.141237 -2.549498 -4.61471
281674 170348.0 1.991976 0.158476 -2.583441 0.408670 1.151147 -0.096695 0.223050 -0.068384 0.577829 -0.888722 0.491140 0.728903 0.380428 -1.948883 -0.832498 0.519436 0.90356

[ ] new_dataset['Class'].value_counts()
1    492
0     56
Name: Class, dtype: int64

[ ] new_dataset.groupby('Class').mean()

```

The output of `new_dataset['Class'].value_counts()` shows the distribution of the 'Class' variable, with 492 instances of class 1 and 56 instances of class 0.

The output of `new_dataset.groupby('Class').mean()` is a DataFrame showing the mean values for each feature, grouped by class.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17
Class																		
0	80835.375000	-0.002041	-0.203528	0.187796	-0.026107	-0.024774	0.038651	-0.181050	-0.334998	0.141425	0.073726	0.002476	0.031928	-0.014840	-0.046562	0.133623	-0.168151	0.0
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	-5.676883	3.800173	-6.259393	-0.109334	-6.971723	-0.092929	-4.139946	-6.6

The code cell also shows the splitting of the data into features and targets:

```

[ ] X = new_dataset.drop(columns='Class', axis=1)
[ ] Y = new_dataset['Class']

[ ] print(X)

```

new\_dataset['Class'].value\_counts()



```
new_dataset.groupby('Class').mean()
```

```
X = new_dataset.drop(columns='Class', axis=1)
```

```
Y = new_dataset['Class']
```

The image displays two screenshots of a Google Colab notebook titled "Credit Card Fraud Detection Using ML.ipynb".

**Top Screenshot:** Shows the execution of statistical measures for legit and fraud transactions. The code includes:

```
#statistical measures of the data
legit.Amount.describe()

fraud.Amount.describe()

#compare the values for the both transactions
credit_card_data.groupby('class').mean()
```

The output for `legit.Amount.describe()` is:

```
count    284315.000000
mean      88.291022
std       250.105092
min        0.000000
25%        5.650000
50%       22.000000
75%       77.050000
max      25691.160000
Name: Amount, dtype: float64
```

The output for `fraud.Amount.describe()` is:

```
count     492.000000
mean     122.211321
std     256.683288
min      0.000000
25%      1.000000
50%      9.250000
75%     105.890000
max     2125.870000
Name: Amount, dtype: float64
```

The output for `credit_card_data.groupby('class').mean()` is a table with columns V1 through V16 and a 'class' column.

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	
class																		
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	0.009824	-0.006576	0.010832	0.000189	0.012064	0.000161	0.007164	0.0

**Bottom Screenshot:** Shows the distribution of transactions and the separation of data into legit and fraud datasets. The code includes:

```
#distribution of legit transaction & fraudulent transactions
credit_card_data['class'].value_counts()

#separating the data for analysis
legit = credit_card_data[credit_card_data.class == 0]
fraud = credit_card_data[credit_card_data.class == 1]

print(legit.shape)
print(fraud.shape)
```

The output for `credit_card_data['class'].value_counts()` is:

```
0    284315
1     492
Name: class, dtype: int64
```

The output for `print(legit.shape)` and `print(fraud.shape)` is:

```
(284315, 31)
(492, 31)
```

The notebook also displays a warning: "This dataset is highly unbalanced. 0 -> Normal Transactions, 1 -> Fraudulent Transactions".

```
X = new_dataset.drop(columns='Class', axis=1)
```

```
Y = new_dataset['Class']
```

```
print(X)
```

```
print(Y)
```

### Split the data into Training & Testing Data

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
```

The screenshot shows a Jupyter Notebook window titled "Credit Card Fraud Detection Using ML.ipynb". The notebook is open to a code cell containing the following code:

```
new_dataset.groupby('Class').mean()
```

The output of this code is a table with columns labeled 'Class' and 'V1' through 'V16'. The 'Class' column has two categories: 0 and 1. The 'V1' through 'V16' columns contain numerical values for each class.

Below the table, the notebook shows a code cell with the following code:

```
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']

print(X)
```

The output of the `print(X)` command is a table with columns labeled 'Time', 'V1', 'V2', ..., 'V27', 'V28', and 'Amount'. The 'Time' column contains numerical values, and the 'Amount' column contains numerical values. The 'V1' through 'V27' columns contain numerical values. The 'Amount' column contains numerical values. The output is truncated with ellipses in the middle.

The notebook interface also shows a toolbar with various icons and a status bar at the bottom indicating the language is English (IN) and the time is 8:34 PM on 12/22/2021.

```
print(X.shape, X_train.shape, X_test.shape)
```

```

#statistical measures of the data
[ ] legit.Amount.describe()

count    284315.000000
mean      88.291022
std       250.105092
min        0.000000
25%        5.650000
50%       22.000000
75%       77.050000
max      25691.160000
Name: Amount, dtype: float64

[ ] fraud.Amount.describe()

count     492.000000
mean     122.211321
std      256.683288
min       0.000000
25%       1.000000
50%       9.250000
75%      185.890000
max     2125.870000
Name: Amount, dtype: float64

[ ] #compare the values for the both transactions
credit_card_data.groupby('class').mean()


```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16		
class	0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	0.009824	-0.006576	0.010832	0.000189	0.012064	0.000161	0.007164	0.0

(548, 30) (438, 30) (110, 30)

## Model Training

### Logistics Regression

```
model = LogisticRegression()
```

(548, 30) (438, 30) (110, 30)

#training the Logistics Regression with training data

The screenshot shows a Google Colab notebook titled "Credit Card Fraud Detection Using ML.ipynb". The code cell contains the following Python code:

```
new_dataset.groupby('Class').mean()
```

The output is a table showing the mean values for each variable (V1-V16) across two classes (0 and 1).

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	
0	80835.375000	-0.002041	-0.203528	0.187796	-0.026107	-0.024774	0.038651	-0.181050	-0.334998	0.141425	0.073726	0.002476	0.031928	-0.014840	-0.046562	0.133623	-0.168151	0.0
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	-5.676883	3.800173	-6.259393	-0.109334	-6.971723	-0.092929	-4.139946	-6.6

Below this, the code splits the data into features (X) and targets (Y):

```
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']
```

The output shows the first few rows of the feature matrix X:

Time	V1	V2	...	V27	V28	Amount	
196088	131356.0	2.091936	0.148705	...	-0.034289	-0.085717	1.00
254893	156965.0	1.978648	-1.555866	...	0.019441	-0.035722	99.00
49762	44199.0	1.351546	-1.390759	...	0.030841	0.038353	88.00
159281	112390.0	-0.467520	-4.475601	...	-0.236393	0.144232	1228.82
19327	30190.0	-1.416985	-1.932028	...	0.470380	0.103516	13.00

The final output is a 548 rows x 30 columns matrix.

`model.fit(X_train, Y_train)`

The screenshot shows the same Google Colab notebook, but now the code cell contains the training step:

```
print(X)
```

The output shows the first few rows of the feature matrix X (repeated from the previous screenshot).

```
print(Y)
```

The output shows the target variable Y, which is a 1D array of 0s and 1s:

196088	0
254893	0
49762	0
159281	0

The final output is a 548 rows x 30 columns matrix for X and a 1D array for Y.

`LogisticRegression()`

Credit Card Fraud Detection Using ML.ipynb

```
#statistical measures of the data
legit.Amount.describe()
```

```
count    284315.000000
mean      88.291022
std       250.105092
min        0.000000
25%        5.650000
50%       22.000000
75%       77.050000
max      25691.160000
Name: Amount, dtype: float64
```

```
fraud.Amount.describe()
```

```
count     492.000000
mean     122.211321
std      256.683288
min       0.000000
25%       1.000000
50%       9.250000
75%      185.890000
max     2125.870000
Name: Amount, dtype: float64
```

```
#compare the values for the both transactions
credit_card_data.groupby('Class').mean()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
Class																	
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	0.009824	-0.006576	0.010832	0.000189	0.012064	0.000161	0.007164

Credit Card Fraud Detection Using ML.ipynb

```
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']
```

```
print(X)
```

	Time	V1	V2	...	V27	V28	Amount
196088	131356.0	2.091936	0.148705	...	-0.034289	-0.085717	1.00
254893	156965.0	1.978648	-1.558066	...	0.019441	-0.025722	99.00
49762	44199.0	1.351546	-1.390759	...	0.030841	0.038353	88.00
159281	112390.0	-0.467520	-4.475601	...	-0.236393	0.144232	1228.82
19327	30190.0	-1.416985	-1.932028	...	0.470380	0.103516	13.00
...	...	...	...	...	...	...	...
279863	169142.0	-1.927883	1.125653	...	0.292680	0.147968	390.00
280143	169347.0	1.378559	1.289381	...	0.389152	0.186637	0.76
280149	169351.0	-0.676143	1.126366	...	0.385107	0.194361	77.89
281144	169966.0	-3.113832	0.585864	...	0.884876	-0.253700	245.00
281674	170348.0	1.991976	0.158476	...	0.002988	-0.015309	42.53

```
[548 rows x 30 columns]
```

```
print(Y)
```

196088	0
254893	0
49762	0
159281	0

Model Evaluation

## Accuracy Score

#accuracy on training data

```

new_dataset.groupby('Class').mean()

```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	
0	80835.375000	-0.002041	-0.203528	0.187796	-0.026107	-0.024774	0.038651	-0.181050	-0.334998	0.141425	0.073726	0.002476	0.031928	-0.014840	-0.046562	0.133623	-0.168151	0.0
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	-5.676883	3.800173	-6.259393	-0.109334	-6.971723	-0.092929	-4.138946	-6.6

```

X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']

print(X)

```

	Time	V1	V2	...	V27	V28	Amount
196888	121356.0	2.891936	0.148705	...	-0.034289	-0.085717	1.00
254893	156965.0	1.978648	-1.555866	...	0.019441	-0.035722	99.00
49762	44199.0	1.351546	-1.390759	...	0.038841	0.038353	88.00
159281	112390.0	-0.467520	-4.475601	...	-0.236393	0.144232	1228.82
19327	30190.0	-1.416989	-1.932028	...	0.470380	0.103516	13.00
...	...	...	...	...	...	...	...
279863	169142.0	-1.927883	1.125653	...	0.292680	0.147968	390.00
280143	169347.0	1.378559	1.289381	...	0.389152	0.186637	0.76
280149	169351.0	-0.676143	1.126366	...	0.385107	0.194361	77.89
281144	169966.0	-3.113832	0.585864	...	0.884876	-0.253700	245.00
281674	170348.0	1.991976	0.158476	...	0.002988	-0.015309	42.53

[548 rows x 30 columns]

X\_train\_prediction = model.predict(X\_train)

```

LogisticRegression()

Model Evaluation

Accuracy Score

#accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

print('Accuracy on Training Data: ', training_data_accuracy)
Accuracy on Training Data: 0.9611872146118722

#accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

print('Accuracy on Training Data: ', test_data_accuracy)
Accuracy on Training Data: 0.9272727272727272

```

Credit Card Fraud Detection Using ML.ipynb

```
new_dataset.groupby('Class').mean()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	
0	80835.375000	-0.002041	-0.203528	0.187796	-0.026107	-0.024774	0.038651	-0.181050	-0.334998	0.141425	0.073726	0.002476	0.031928	-0.014840	-0.046562	0.133623	-0.168151	0.0
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	-5.676883	3.800173	-6.259393	-0.109334	-6.971723	-0.092929	-4.139946	-6.6

Splitting the data into Features & Targets

```
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']
```

```
print(X)
```

	Time	V1	V2	...	V27	V28	Amount
196088	131356.0	2.091936	0.148705	...	-0.034289	-0.085717	1.00
254893	156965.0	1.978648	-1.555866	...	0.019441	-0.035722	99.00
49762	44199.0	1.351546	-1.390759	...	0.030841	0.038353	88.00
159281	112390.0	-0.467520	-4.475601	...	-0.236393	0.144232	1228.82
19327	30190.0	-1.416985	-1.932028	...	0.470380	0.103516	13.00
...	...	...	...	...	...	...	...
279863	169142.0	-1.927883	1.125653	...	0.292680	0.147968	390.00
280143	169347.0	1.378559	1.289381	...	0.389152	0.186637	0.76
280149	169351.0	-0.676143	1.126366	...	0.385107	0.194361	77.89
281144	169966.0	-3.113832	0.585864	...	0.884876	-0.253700	245.00
281674	170348.0	1.991976	0.158476	...	0.002988	-0.015309	42.53

[548 rows x 30 columns]

Credit Card Fraud Detection Using ML.ipynb

```
new_dataset.groupby('Class').mean()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	
0	80835.375000	-0.002041	-0.203528	0.187796	-0.026107	-0.024774	0.038651	-0.181050	-0.334998	0.141425	0.073726	0.002476	0.031928	-0.014840	-0.046562	0.133623	-0.168151	0.0
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	-5.676883	3.800173	-6.259393	-0.109334	-6.971723	-0.092929	-4.139946	-6.6

Splitting the data into Features & Targets

```
X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']
```

```
print(X)
```

	Time	V1	V2	...	V27	V28	Amount
196088	131356.0	2.091936	0.148705	...	-0.034289	-0.085717	1.00
254893	156965.0	1.978648	-1.555866	...	0.019441	-0.035722	99.00
49762	44199.0	1.351546	-1.390759	...	0.030841	0.038353	88.00
159281	112390.0	-0.467520	-4.475601	...	-0.236393	0.144232	1228.82
19327	30190.0	-1.416985	-1.932028	...	0.470380	0.103516	13.00
...	...	...	...	...	...	...	...
279863	169142.0	-1.927883	1.125653	...	0.292680	0.147968	390.00
280143	169347.0	1.378559	1.289381	...	0.389152	0.186637	0.76
280149	169351.0	-0.676143	1.126366	...	0.385107	0.194361	77.89
281144	169966.0	-3.113832	0.585864	...	0.884876	-0.253700	245.00
281674	170348.0	1.991976	0.158476	...	0.002988	-0.015309	42.53

[548 rows x 30 columns]

The screenshot shows a Google Colab notebook titled "Credit Card Fraud Detection Using ML.ipynb". The code cell contains the following Python code:

```
#statistical measures of the data
legit.Amount.describe()

fraud.Amount.describe()

#compare the values for the both transactions
credit_card_data.groupby('Class').mean()
```

The output for `legit.Amount.describe()` is:

```
count    284315.000000
mean      88.291022
std       250.105092
min        0.000000
25%       5.650000
50%       22.000000
75%       77.050000
max      25691.160000
Name: Amount, dtype: float64
```

The output for `fraud.Amount.describe()` is:

```
count      492.000000
mean     122.211321
std       256.683288
min        0.000000
25%       1.000000
50%       9.250000
75%      105.890000
max     2125.870000
Name: Amount, dtype: float64
```

The output for `credit_card_data.groupby('Class').mean()` is a table with columns: Class, Time, V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16.

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	0.009824	-0.006576	0.010832	0.000189	0.012064	0.000161	0.007164

```
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)
```

The screenshot shows a Google Colab notebook titled "Credit Card Fraud Detection Using ML.ipynb". The code cell contains the following Python code:

```
NameError: name 'pd' is not defined

SEARCH STACK OVERFLOW

# first 5 rows of the data
credit_card_data.head()

credit_card_data.tail()
```

The output for `credit_card_data.head()` is a table with columns: V12, V13, V14, V15, V16, V17, V18, V19, V20, V21, V22, V23, V24, V25, V26, V27, V28, Amount, Class.

V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
-0.617801	-0.991390	-0.311169	1.468177	-0.470401	0.207971	0.025791	0.403993	0.251412	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1.065235	0.489095	-0.143772	0.635558	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	2.69	0
0.066084	0.717293	-0.165946	2.345865	-2.890083	1.109969	-0.121359	-2.261857	0.524980	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
0.178228	0.507757	-0.287924	-0.631418	-1.059647	-0.684093	1.965775	-1.232622	-0.208038	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
0.538196	1.345852	-1.119670	0.175121	-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	69.99	0

The output for `credit_card_data.tail()` is a table with columns: Time, V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, V16.

Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16		
284802	172788.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	4.356170	-1.593105	2.711941	-0.689256	4.628942	-0.924459	1.107641	1.991
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	-0.975926	-0.150189	0.915802	1.214756	-0.675143	1.164931	-0.711757	-0.025
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.298827	0.708417	0.432454	-0.484782	0.411614	0.063119	-0.183699	-0.510602	1.329284	0.140716	0.313
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.688180	0.679145	0.392087	-0.399126	-1.933849	-0.962886	-1.042082	0.449624	1.962563	-0.608577	0.508
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	-0.915427	-1.040458	-0.031513	-0.188093	-0.084316	0.041333	-0.302620	-0.660

```
print('Accuracy on Training Data: ', training_data_accuracy)
```



The screenshot shows a Google Colab notebook titled "Credit Card Fraud Detection Using ML.ipynb". The code in the notebook is as follows:

```

LogisticRegression()

Model Evaluation

Accuracy Score

[] #accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

print('Accuracy on Training Data: ', training_data_accuracy)

Accuracy on Training Data: 0.9611872146118722

[] #accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

print('Accuracy on Training Data: ', test_data_accuracy)

Accuracy on Training Data: 0.9272727272727272

```

The output of the notebook shows the accuracy on training data as 0.9611872146118722 and the accuracy on test data as 0.9272727272727272.

Accuracy on Training Data: 0.9611872146118722

The screenshot shows a Google Colab notebook titled "Credit Card Fraud Detection Using ML.ipynb". The code in the notebook is as follows:

```

#statistical measures of the data
legit.Amount.describe()

count    284315.000000
mean     88.291022
std      259.105992
min       0.000000
25%       5.650000
50%      22.000000
75%      77.050000
max     25691.160000
Name: Amount, dtype: float64

fraud.Amount.describe()

count    492.000000
mean     122.211321
std      256.683288
min       0.000000
25%       1.000000
50%       9.250000
75%     105.890000
max     2125.870000
Name: Amount, dtype: float64

[] #compare the values for the both transactions
credit_card_data.groupby('class').mean()

```

The output of the notebook shows the statistical measures of the data for both legitimate and fraudulent transactions. The output for the legitimate transactions is as follows:

count	mean	std	min	25%	50%	75%	max
284315.000000	88.291022	259.105992	0.000000	5.650000	22.000000	77.050000	25691.160000

The output for the fraudulent transactions is as follows:

count	mean	std	min	25%	50%	75%	max
492.000000	122.211321	256.683288	0.000000	1.000000	9.250000	105.890000	2125.870000

The output for the comparison of values for both transactions is as follows:

Class	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	0.009824	-0.006576	0.010832	0.000189	0.012064	0.000161	0.007164

# accuracy on test data

X\_test\_prediction = model.predict(X\_test)

The screenshot shows a Google Colab notebook titled "Credit Card Fraud Detection Using ML.ipynb". The code cell contains the following Python code:

```

LogisticRegression()

Model Evaluation

Accuracy Score

[] #accuracy on training data
X_train_prediction = model.predict(X_train)
training_data_accuracy = accuracy_score(X_train_prediction, Y_train)

print('Accuracy on Training Data: ', training_data_accuracy)

Accuracy on Training Data: 0.9611872146118722

[] #accuracy on test data
X_test_prediction = model.predict(X_test)
test_data_accuracy = accuracy_score(X_test_prediction, Y_test)

print('Accuracy on Training Data: ', test_data_accuracy)

Accuracy on Training Data: 0.9272727272727272

```

The output shows the training accuracy as 0.9611872146118722 and the test accuracy as 0.9272727272727272.

test\_data\_accuracy = accuracy\_score(X\_test\_prediction, Y\_test)

The screenshot shows a Google Colab notebook titled "Credit Card Fraud Detection Using ML.ipynb". The code cell contains the following Python code:

```

!python-input-5-11c6ac7e9cdd> in cmodule()
----> 1 legit_sample = legit.sample(n=492)
NameError: name 'legit' is not defined

Concatenating two Data Frames

[] new_dataset = pd.concat([legit_sample, fraud], axis=0)

new_dataset.head()

   Time    V1    V2    V3    V4    V5    V6    V7    V8    V9    V10   V11   V12   V13   V14   V15   V16   V17
196088  131356.0  2.091936  0.148705 -3.738184  0.159139  3.386352  2.869366  0.462437  0.467049 -0.437896  0.415274 -0.132049  0.132293 -0.441097  1.110540  0.076008 -0.935165 -0.3006
254893  156965.0  1.978648 -1.555866 -1.887961 -1.881698  1.130933  3.709389 -1.619579  0.953556 -0.045954  0.744668  0.037306 -0.481764  0.230474 -0.272885  0.996942  1.312164 -0.1085
49762  44199.0  1.351546 -1.390759  0.713643 -1.244118 -2.008408 -0.801667 -1.222513 -0.101894 -1.449164  1.422011 -0.688937 -1.450402 -0.426721 -0.276468  0.962478 -0.091587  0.4346
159281  112390.0 -0.467520 -4.475601 -2.316311  0.869426 -0.645675  2.233655  0.848250  0.294705  0.787210 -0.591245  0.724197  1.021004 -0.221404  0.457611  0.190564 -0.196274  0.1223
19327  30190.0 -1.416985 -1.932028  1.321694 -2.173934  0.324299 -1.484920 -1.988174 -0.723678 -1.443325  1.167898 -1.552772 -1.898673 -0.880261 -0.323279  0.265376  0.335234 -0.0377

new_dataset.tail()

   Time    V1    V2    V3    V4    V5    V6    V7    V8    V9    V10   V11   V12   V13   V14   V15   V16   V17
279883  169142.0 -1.927883  1.125663 -4.518331  1.748293 -1.566487 -0.010484 -0.882850  0.697211 -0.064945 -5.687794  2.116795 -5.417424 -1.236123 -6.666177  0.401701 -0.897826 -4.57062

```

print('Accuracy on Training Data: ', test\_data\_accuracy)

```

[ ] 1 80746.806911 -4.771948 3.623778 -7.033281 4.542029 -3.151225 -1.397737 -5.568731 0.570636 -2.581123 -5.676883 3.800173 -0.259393 -0.109334 -6.971723 -0.092929 -4.139946 -6.6

Splitting the data into Features & Targets

[ ] X = new_dataset.drop(columns='Class', axis=1)
    Y = new_dataset['Class']

print(X)

   Time      V1      V2      ...      V27      V28      Amount
196088 131356.0  2.091936  0.148705  ...  -0.034289 -0.085717    1.00
254893 156965.0  1.978648 -1.555866  ...   0.019441 -0.035722    99.00
49762  44199.0  1.351546 -1.390759  ...   0.030841  0.038353    88.00
159281 112390.0 -0.467520 -4.475601  ...  -0.236393  0.144232   1228.82
19327  30190.0  -1.416985 -1.932028  ...   0.470380  0.103516    13.00
...
279863 169142.0 -1.927883  1.125653  ...   0.292680  0.147968   390.00
280143 169347.0  1.378559  1.289381  ...   0.389152  0.186637    0.76
280149 169351.0 -0.676143  1.126366  ...   0.385107  0.194361    77.89
281144 169966.0 -3.113832  0.585864  ...   0.884876 -0.253700   245.00
281674 170348.0  1.991976  0.158476  ...   0.002988 -0.015309    42.53

[548 rows x 30 columns]

print(Y)

196088 0
254893 0
49762  0
159281 0

```

Accuracy on Training Data: 0.9272727272727272

```

logisticRegression()

Model Evaluation

Accuracy Score

[ ] #accuracy on training data
    x_train_prediction = model.predict(x_train)
    training_data_accuracy = accuracy_score(x_train_prediction, y_train)

print('Accuracy on Training Data: ', training_data_accuracy)

Accuracy on Training Data: 0.9611872146118722

[ ] #accuracy on test data
    x_test_prediction = model.predict(x_test)
    test_data_accuracy = accuracy_score(x_test_prediction, y_test)

print('Accuracy on Training Data: ', test_data_accuracy)

Accuracy on Training Data: 0.9272727272727272

```

```

new_dataset.groupby('Class').mean()

```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	
Class																		
0	80835.375000	-0.002041	-0.203528	0.187796	-0.026107	-0.024774	0.038651	-0.181050	-0.334998	0.141425	0.073726	0.002476	0.031928	-0.014840	-0.046562	0.133623	-0.168151	0.0
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	-5.676883	3.800173	-6.259393	-0.108334	-6.971723	-0.092929	-4.139946	-6.6

```

Splitting the data into Features & Targets

X = new_dataset.drop(columns='Class', axis=1)
Y = new_dataset['Class']

print(X)

```

	Time	V1	V2	...	V27	V28	Amount
196088	131356.0	2.091936	0.148705	...	-0.034289	-0.085717	1.00
254893	156965.0	1.978648	-1.555866	...	0.019441	-0.035722	99.00
49762	44199.0	1.351546	-1.390759	...	0.030841	0.038353	88.00
159281	112390.0	-0.467520	-4.475601	...	-0.236393	0.144232	1228.82
19327	30190.0	-1.416985	-1.932028	...	0.470380	0.103516	13.00
...	...	...	...	...	...	...	...
279863	169142.0	-1.927883	1.125653	...	0.292680	0.147968	390.00
280143	169347.0	1.378559	1.289381	...	0.389152	0.186637	0.76
280149	169351.0	-0.676143	1.126366	...	0.385107	0.194361	77.89
281144	169966.0	-3.113832	0.585864	...	0.884876	-0.253700	245.00
281674	170348.0	1.991976	0.158476	...	0.002988	-0.015309	42.53

[548 rows x 30 columns]

Currently, businesses work on fraud detection systems that incorporate machine learning and artificial intelligence. Using modern fraud protection systems powered by ML, many industries can keep their finances safe. There are already some fraud detection solutions for FinTech, e-commerce, banking, healthcare, online gaming, and other industries. No matter your industry, there's always a way to benefit from AI and ML. Machine learning algorithms can process huge amounts of data and draw patterns for every business to protect it from fraud. For instance, machine learning helps online gaming businesses detect account takeovers and other scams by tracing patterns in a player's in-game behavior.

Capgemini claims their ML fraud detection system can reduce fraud investigation time by 70% while increasing accuracy by 90%. Another ML fraud prevention solution provider, Feedzai, claims

that a well-trained machine learning solution can identify and prevent 95% of all fraud while minimizing the amount of human labor required during the investigation stage.

Large corporations like Airbnb, Yelp, and Jet.com are already using AI solutions to get insights from big data and prevent issues such as fake accounts, account takeover, payment fraud, and promotion abuse. Machine learning takes care of all the dirty work of data analysis and predictive analytics and allows companies to grow and develop safe from fraud.

## Final thoughts

Businesses all over the world have already started using data science to prevent financial fraud. Machine learning is currently the most promising innovative tool that can help companies prevent fraudulent operations that lead to greater losses each year. Yet apart from implementing modern fraud detection solutions, companies also need modern and secure FinTech services and custom software development services that are harder for fraudsters to manipulate. An outdated financial system is always full of loopholes tricksters can use. Luckily, machine learning has the potential to improve bank fraud detection with data analytics and help nearly every industry.

From the moment the e-commerce payment systems came to existence, there have always been people who will find new ways to access someone's finances illegally. This has become a major problem in the modern era, as all transactions can easily be completed online by only entering your credit card information. Even in the 2010s, many American retail website users were the victims of online transaction fraud right before two-step verification was used for shopping online.

Organizations, consumers, banks, and merchants are put at risk when a data breach leads to monetary theft and ultimately the loss of customers' loyalty along with the company's reputation.

Unauthorized card operations hit an astonishing amount of 16.7 million victims in 2017.

Additionally, as reported by the Federal Trade Commission (FTC), the number of credit card fraud claims in 2017 was 40% higher than the previous year's number. There were around 13,000 reported cases in California and 8,000 in Florida, which are the largest states per capita for such type of crime. The amount of money at stake will exceed approximately \$30 billion by 2020. Here are some credit card fraud statistics.

## **8. CONCLUSION**

In this we developed a novel method for fraud detection, where customers are grouped based on their transactions and extract behavioural patterns to develop a profile for every cardholder. Then different classifiers are applied on three different groups later rating scores are generated for every type of classifier. This dynamic changes in parameters lead the system to adapt to new cardholder's transaction behaviours timely. Followed by a feedback mechanism to solve the problem of concept drift. We observed that the Matthews Correlation Coefficient was the better parameter to deal with imbalance dataset. MCC was not the only solution. By applying the SMOTE, we tried balancing the dataset, where we found that the classifiers were performing better than before. The other way of handling imbalance dataset is to use one-class classifiers like one-class SVM. We finally

observed that Logistic regression, decision tree and random forest are the algorithms that gave better results. Credit card fraud is without a doubt an act of criminal dishonesty. This article has listed out the most common methods of fraud along with their detection methods and reviewed recent findings in this field. This paper presents a study on credit card fraud detection using K-Star machine learning algorithm.

## REFERENCES

- [1] “Credit Card Fraud Detection Based on Transaction Behaviour -by John Richard D. Kho, Larry A. Veal” published by Proc. of the 2017 IEEE Region 10 Conference (TENCON), Malaysia, November 5-8, 2017
- [2] CLIFTON PHUA<sup>1</sup>, VINCENT LEE<sup>1</sup>, KATE SMITH<sup>1</sup> & ROSS GAYLER<sup>2</sup> “ A Comprehensive Survey of Data Mining-based Fraud Detection Research” published by School of Business Systems, Faculty of Information Technology, Monash University, Wellington Road, Clayton, Victoria 3800, Australia
- [3] “Survey Paper on Credit Card Fraud Detection by Suman” , Research Scholar, GJUS Hisar HCE, Sonapat published by International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 3 Issue 3, March 2014
- [4] “Research on Credit Card Fraud Detection Model Based on Distance Sum – by Wen-Fang YU and Na Wang” published by 2009 International Joint Conference on Artificial Intelligence
- [5] “Credit Card Fraud Detection through Parenclitic Network Analysis-By Massimiliano Zanin, Miguel Romance, Regino Criado, and SantiagoMoral” published by Hindawi Complexity Volume 2018, Article ID 5764370,
- [6] “Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy” published by IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, VOL. 29, NO. 8, AUGUST 2018
- [7] “Credit Card Fraud Detection-by Ishu Trivedi, Monika, Mrigya, Mridushi” published by International Journal of Advanced Research in Computer and Communication Engineering Vol. 5, Issue 1, January 2016
- [8] David J.Watson,David J.Hand,M Adams,Whitrow and Piotr Juszczak “Plastic Card Fraud Detection using Peer Group Analysis” Springer, Issue 2008.







