**Phase 5**

|  |  |
| --- | --- |
| **Date** | **31-10-2023** |
| **Team ID** | **3864** |
| **Project Name** | **6112-CREDIT CARD FRAUD DETECTION** |

|  |  |
| --- | --- |
| 1 | Introduction |
| 2 | Problem Statement |
| 3 | Project Overview |
| 4 | Design Thinking Process |
| 5 | Development Phases |
| 6 | Data Visualization |
| 7 | Observation |
| 8 | Models |
| 9 | Conclusion |

**INTRODUCTION:**

Credit card fraud has become an increasingly prevalent and costly issue in the modern financial landscape. With the growing reliance on electronic payment methods, the risk of unauthorized transactions and fraudulent activities has expanded exponentially. This necessitates the development and implementation of robust credit card fraud detection systems to protect both financial institutions and cardholders.

**CREDIT CARD FRAUD DETECTION**

**Problem Statement :**

The problem at hand is to create an efficient credit card fraud detection system that can accurately identify and prevent fraudulent transactions while minimizing false positives. The system should be capable of real-time processing, feature engineering, handling imbalanced data, and scalability. It needs to remain robust against new fraud types, provide a user-friendly interface, ensure compliance with regulations, balance detection accuracy with operational costs, and implement an effective alert and reporting mechanism. The ultimate goal is to protect both financial institutions and cardholders, instilling trust and security in electronic transactions.

**Project Overview**:

Objective: The primary objective of a credit card fraud detection system is to identify and prevent fraudulent transactions while maintaining the efficiency and integrity of legitimate transactions.

**High Accuracy:** Develop a model that can accurately identify fraudulent transactions with a high level of confidence.

**A REALISTIC MODELING AND A NOVEL LEARNING STRATEGY**

Credit card fraud is a significant problem that costs billions of dollars each year. Existing fraud detection systems often rely on rule-based approaches or simple machine learning algorithms that are not effective in detecting sophisticated fraud patterns. Moreover, these systems often suffer from high false positive rates, which can lead to customer dissatisfaction and increased operational costs for financial institutions.

To address these limitations, we propose a novel learning strategy that leverages the alert-feedback interaction between fraud investigators and the fraud detection system. Our approach combines online learning with active learning to adapt to the time-variant nature of the transaction stream and to reduce the labeling effort required by investigators.

We also introduce a new performance measure that takes into account the cost of false positives and false negatives. Our experiments on real-world credit card transaction data demonstrate that our approach outperforms existing methods in terms of both detection accuracy and labeling efficiency.

Our proposed approach can be extended to other domains beyond credit card fraud detection, such as healthcare fraud detection and network intrusion detection.

**CARD THEFT:**

Credit card fraud is a major concern for financial institutions and merchants, as it can result in significant financial losses and damage to their reputation. Traditional rule-based systems for fraud detection are often ineffective in detecting new and evolving fraud patterns, leading to a high rate of false positives and false negatives. This calls for the need for more advanced and accurate fraud detection systems.

Machine learning algorithms have shown great promise in detecting credit card fraud by analyzing large volumes of transaction data and identifying patterns and anomalies. These algorithms can be trained on historical data to learn the characteristics of fraudulent transactions and then applied to new transactions in real-time to detect potential fraud .Fraud detection techniques are constantly being prepared to protect criminals adapting to lie strategies .

Some common machine learning algorithms used for credit card fraud detection include logistic regression, decision trees, random forests, and neural networks. By using machine learning, financial institutions and merchants can improve their fraud detection accuracy, reduce false positives and false negatives, and ultimately protect their customers from financial losses and fraud-related damages. However, it is important to note that machine learning algorithms are not foolproof and require continuous monitoring and updating to stay effective against new and evolving fraud patterns

**CREDIT CARD FRAUDS: ONLINE & OFFLINE:**

Credit card fraud is a serious issue that affects both consumers and businesses. Fraudulent transactions can result in financial losses for individuals and damage to a company's reputation. Traditional methods of fraud detection, such as rule-based systems, are often not effective in detecting new and sophisticated fraud techniques.

Therefore, there is a need for more advanced techniques, such as machine learning algorithms, to detect fraudulent transactions in real-time.

Machine learning algorithms can be used to analyze all credit card transactions and identify suspicious cases. These algorithms can learn from historical data and detect patterns that are indicative of fraud.

For example, they can detect unusual spending patterns, such as a sudden increase in the number or amount of transactions, or transactions made in different locations within a short period of time. Once a suspicious transaction is detected, the algorithm can notify the appropriate parties and the transaction can be investigated further.

Over time, the algorithm can be updated with new data and improve its accuracy in detecting fraud. By using machine learning algorithms for credit card fraud detection, companies can protect their customers and their business from financial losses and reputational damage.

**REVIEW OF MACHINE LEARNING APPROACH ON CREDIT CARD FRAUD DETECTION**

In this project, we will analyse customer-level data which has been collected and analysed during a research collaboration of Worldline and the Machine Learning Group.A machine learning approach to improve the accuracy of fraud detection while ensuring privacy. The article discusses the limitations of existing fraud detection methods and the need for a more effective and efficient solution.

The authors have conducted experiments using a dataset of credit card transactions and achieved an accuracy of 99.96% in detecting fraudulent transactions in real-time data. The proposed solution also addresses the issue of data privacy by using a federated learning model to ensure data confidentiality and integrity. The solution involves distributing the neural network model to respective correspondence banks or financial institutions, which then train the model on their own data while preserving privacy

**A RESEARCH PAPER ON CREDIT CARD FRAUD DETECTION**

The problem statement in this paper is that credit cards are an easy target for fraudsters who can withdraw money without the owner's knowledge and make it look like the actual owner made the withdrawal.This makes it difficult to stop and catch them. The paper aims to explore the use of machine learning algorithms to detect fraudulent transactions and prevent online frauds.

The proposed solution in this paper is to use different machine learning algorithms like Decision trees, Random Forest, and other algorithms to categorize transactions and train classifiers to predict fraud transactions accurately.

**Design Thinking Process**:

**Emphathize:**

Start by empathizing with the end-users or stakeholders to gain a deep understanding of their needs, concerns, and pain points.

Conduct interviews, surveys, observations, and immerse yourself in the user's experience to gather insights.

**Define:**

Synthesize the information collected during the empathize phase to define the core problem or challenge.

Create a clear and concise problem statement to guide the rest of the process.

**Ideate:**

Encourage brainstorming and ideation sessions to generate a wide range of possible solutions to the defined problem.

Foster a judgment-free environment that promotes creative thinking and welcomes diverse perspectives.

**Prototype:**

Create rough, low-cost, and quick prototypes or representations of your ideas to explore potential solutions.

Prototypes can be sketches, mock-ups, physical models, or even role-playing scenarios.

**Test:**

Use this feedback to refine your solutions or go back to the ideation phase if necessary.

Test your prototypes with real users to gather feedback and insights.

**Iterate:**

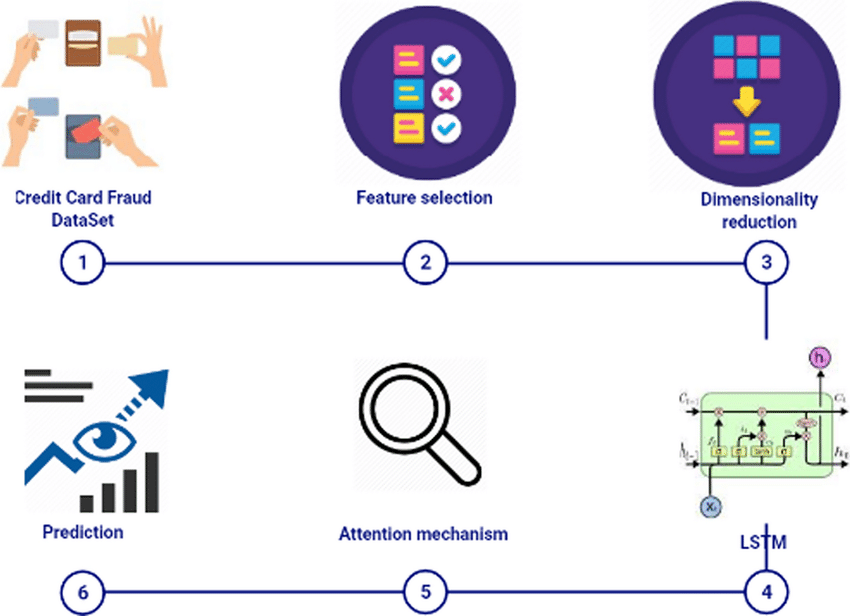
Based on the feedback received, refine your prototypes and iterate through the previous steps as needed.

Continue to gather feedback and make improvements until a viable and user-centric solution is achieved.

**Implement:**

Once you've developed a refined and tested solution, it's time to implement it on a larger scale.

Create a plan for the solution's rollout and integration into the intended context.



**FIG1:PREDICTION ALGORITHM**

**Development Phases**:

The development of a credit card fraud detection system typically involves several phases, from data collection and preprocessing to model development and deployment. Here are the key development phases for building a credit card fraud detection system:

**1.Data Collection:**

Gather historical transaction data, which includes both legitimate and fraudulent transactions. This dataset is crucial for training and testing your fraud detection model.

**2.Data Preprocessing:**

Clean and preprocess the data to handle missing values, outliers, and inconsistencies.Perform feature engineering to extract relevant features from the transaction data, such as transaction amount, time, location, and more.

**3.Data Splitting:**

Split the dataset into training, validation, and testing sets. The training set is used to train the model, the validation set helps optimize model parameters, and the testing set assesses the model's performance.

**4.Feature Scaling and Normalization:**

Standardize or normalize the features to ensure that they have similar scales, which can improve the performance of machine learning algorithms

**5.Model Training:**

Train the selected machine learning models on the training data. During training, the models learn to distinguish between legitimate and fraudulent transactions.

**6.Hyperparameter Tuning:**

Optimize the model's hyperparameters using the validation set to improve its performance.

**7.Model Evaluation:**

Evaluate the models using the testing dataset. Common evaluation metrics include precision, recall, F1 score, and accuracy. Consider the class imbalance when interpreting these metrics.



**FIG 2:LIFE CYCLE OF FRAUD DETECTION**

**Data Visualization and Analysis:**

**PROGRAM:**

**Import Libraries:**

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import PowerTransformer  
from sklearn.preprocessing import StandardScaler  
from sklearn.metrics import classification\_report  
from sklearn.linear\_model import LogisticRegression  
from sklearn.ensemble import RandomForestClassifier  
import xgboost as xgb

from google.colab import drive  
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

# READING DATA

df = pd.read\_csv("/content/drive/MyDrive/creditcard.csv")  
df.head()

Time V1 V2 V3 V4 V5 V6 V7 \  
0 0.0 -1.359807 -0.072781 2.536347 1.378155 -0.338321 0.462388 0.239599   
1 0.0 1.191857 0.266151 0.166480 0.448154 0.060018 -0.082361 -0.078803   
2 1.0 -1.358354 -1.340163 1.773209 0.379780 -0.503198 1.800499 0.791461   
3 1.0 -0.966272 -0.185226 1.792993 -0.863291 -0.010309 1.247203 0.237609   
4 2.0 -1.158233 0.877737 1.548718 0.403034 -0.407193 0.095921 0.592941   
  
 V8 V9 ... V21 V22 V23 V24 V25 \  
0 0.098698 0.363787 ... -0.018307 0.277838 -0.110474 0.066928 0.128539   
1 0.085102 -0.255425 ... -0.225775 -0.638672 0.101288 -0.339846 0.167170   
2 0.247676 -1.514654 ... 0.247998 0.771679 0.909412 -0.689281 -0.327642   
3 0.377436 -1.387024 ... -0.108300 0.005274 -0.190321 -1.175575 0.647376   
4 -0.270533 0.817739 ... -0.009431 0.798278 -0.137458 0.141267 -0.206010   
  
 V26 V27 V28 Amount Class   
0 -0.189115 0.133558 -0.021053 149.62 0   
1 0.125895 -0.008983 0.014724 2.69 0   
2 -0.139097 -0.055353 -0.059752 378.66 0   
3 -0.221929 0.062723 0.061458 123.50 0   
4 0.502292 0.219422 0.215153 69.99 0   
  
[5 rows x 31 columns]

df.info()

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

df.isna().sum()

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

df['Class'].unique()

array([0, 1])

df.describe().T

count mean std min 25% \  
Time 284807.0 9.481386e+04 47488.145955 0.000000 54201.500000   
V1 284807.0 1.168375e-15 1.958696 -56.407510 -0.920373   
V2 284807.0 3.416908e-16 1.651309 -72.715728 -0.598550   
V3 284807.0 -1.379537e-15 1.516255 -48.325589 -0.890365   
V4 284807.0 2.074095e-15 1.415869 -5.683171 -0.848640   
V5 284807.0 9.604066e-16 1.380247 -113.743307 -0.691597   
V6 284807.0 1.487313e-15 1.332271 -26.160506 -0.768296   
V7 284807.0 -5.556467e-16 1.237094 -43.557242 -0.554076   
V8 284807.0 1.213481e-16 1.194353 -73.216718 -0.208630   
V9 284807.0 -2.406331e-15 1.098632 -13.434066 -0.643098   
V10 284807.0 2.239053e-15 1.088850 -24.588262 -0.535426   
V11 284807.0 1.673327e-15 1.020713 -4.797473 -0.762494   
V12 284807.0 -1.247012e-15 0.999201 -18.683715 -0.405571   
V13 284807.0 8.190001e-16 0.995274 -5.791881 -0.648539   
V14 284807.0 1.207294e-15 0.958596 -19.214325 -0.425574   
V15 284807.0 4.887456e-15 0.915316 -4.498945 -0.582884   
V16 284807.0 1.437716e-15 0.876253 -14.129855 -0.468037   
V17 284807.0 -3.772171e-16 0.849337 -25.162799 -0.483748   
V18 284807.0 9.564149e-16 0.838176 -9.498746 -0.498850   
V19 284807.0 1.039917e-15 0.814041 -7.213527 -0.456299   
V20 284807.0 6.406204e-16 0.770925 -54.497720 -0.211721   
V21 284807.0 1.654067e-16 0.734524 -34.830382 -0.228395   
V22 284807.0 -3.568593e-16 0.725702 -10.933144 -0.542350   
V23 284807.0 2.578648e-16 0.624460 -44.807735 -0.161846   
V24 284807.0 4.473266e-15 0.605647 -2.836627 -0.354586   
V25 284807.0 5.340915e-16 0.521278 -10.295397 -0.317145   
V26 284807.0 1.683437e-15 0.482227 -2.604551 -0.326984   
V27 284807.0 -3.660091e-16 0.403632 -22.565679 -0.070840   
V28 284807.0 -1.227390e-16 0.330083 -15.430084 -0.052960   
Amount 284807.0 8.834962e+01 250.120109 0.000000 5.600000   
Class 284807.0 1.727486e-03 0.041527 0.000000 0.000000   
  
 50% 75% max   
Time 84692.000000 139320.500000 172792.000000   
V1 0.018109 1.315642 2.454930   
V2 0.065486 0.803724 22.057729   
V3 0.179846 1.027196 9.382558   
V4 -0.019847 0.743341 16.875344   
V5 -0.054336 0.611926 34.801666   
V6 -0.274187 0.398565 73.301626   
V7 0.040103 0.570436 120.589494   
V8 0.022358 0.327346 20.007208   
V9 -0.051429 0.597139 15.594995   
V10 -0.092917 0.453923 23.745136   
V11 -0.032757 0.739593 12.018913   
V12 0.140033 0.618238 7.848392   
V13 -0.013568 0.662505 7.126883   
V14 0.050601 0.493150 10.526766   
V15 0.048072 0.648821 8.877742   
V16 0.066413 0.523296 17.315112   
V17 -0.065676 0.399675 9.253526   
V18 -0.003636 0.500807 5.041069   
V19 0.003735 0.458949 5.591971   
V20 -0.062481 0.133041 39.420904   
V21 -0.029450 0.186377 27.202839   
V22 0.006782 0.528554 10.503090   
V23 -0.011193 0.147642 22.528412   
V24 0.040976 0.439527 4.584549   
V25 0.016594 0.350716 7.519589   
V26 -0.052139 0.240952 3.517346   
V27 0.001342 0.091045 31.612198   
V28 0.011244 0.078280 33.847808   
Amount 22.000000 77.165000 25691.160000   
Class 0.000000 0.000000 1.000000

**FEATURE ENGINEERING**

* Creating hour column in our dataset by dividing time column which shows time in seconds to 3600.

df['hour']= round(df['Time']/3600)  
df.head()

Time V1 V2 V3 V4 V5 V6 V7 \  
0 0.0 -1.359807 -0.072781 2.536347 1.378155 -0.338321 0.462388 0.239599   
1 0.0 1.191857 0.266151 0.166480 0.448154 0.060018 -0.082361 -0.078803   
2 1.0 -1.358354 -1.340163 1.773209 0.379780 -0.503198 1.800499 0.791461   
3 1.0 -0.966272 -0.185226 1.792993 -0.863291 -0.010309 1.247203 0.237609   
4 2.0 -1.158233 0.877737 1.548718 0.403034 -0.407193 0.095921 0.592941   
  
 V8 V9 ... V22 V23 V24 V25 V26 \  
0 0.098698 0.363787 ... 0.277838 -0.110474 0.066928 0.128539 -0.189115   
1 0.085102 -0.255425 ... -0.638672 0.101288 -0.339846 0.167170 0.125895   
2 0.247676 -1.514654 ... 0.771679 0.909412 -0.689281 -0.327642 -0.139097   
3 0.377436 -1.387024 ... 0.005274 -0.190321 -1.175575 0.647376 -0.221929   
4 -0.270533 0.817739 ... 0.798278 -0.137458 0.141267 -0.206010 0.502292   
  
 V27 V28 Amount Class hour   
0 0.133558 -0.021053 149.62 0 0.0   
1 -0.008983 0.014724 2.69 0 0.0   
2 -0.055353 -0.059752 378.66 0 0.0   
3 0.062723 0.061458 123.50 0 0.0   
4 0.219422 0.215153 69.99 0 0.0   
  
[5 rows x 32 columns]

class\_count = df['Class'].value\_counts()  
class\_count

0 284315  
1 492  
Name: Class, dtype: int64

duplicated\_count = df.duplicated().sum()  
duplicated\_count

1081

df = df.drop\_duplicates(keep='first')

fig = plt.figure(figsize=(10, 7))  
labels = class\_count.index  
data = class\_count.values  
plt.pie(data, labels=labels)  
plt.title('Class Distribution')  
plt.axis('equal')  
plt.show()

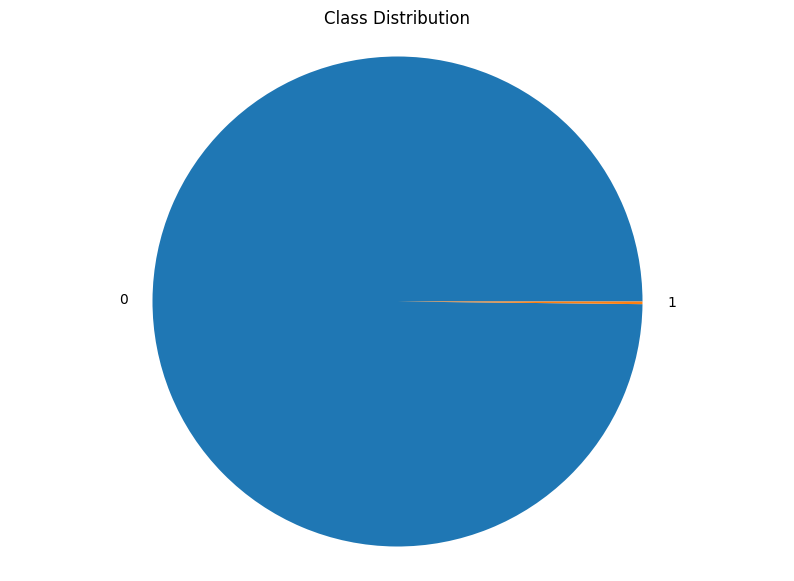


FIG 1: Class Distribution

Credit card fraud detection is a challenging problem due to the imbalanced class distribution, with a very small minority of transactions being fraudulent.

plt.figure(figsize=(10, 5))  
sns.countplot(data = df, x='Class')  
plt.title('Class Distribution (Count)')  
plt.xlabel('Class')  
plt.ylabel('Count')  
plt.show()

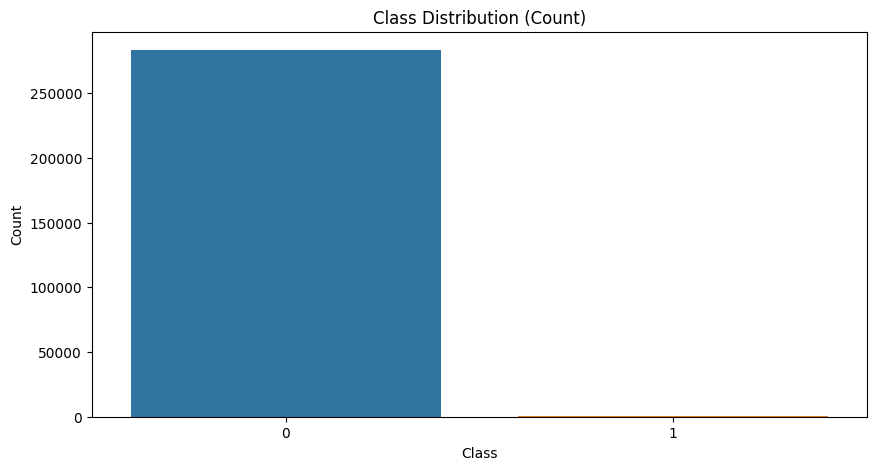


FIG 2: Class distribution(count)

Credit card fraud detection class distribution is typically imbalanced, with ~99.8% legitimate transactions and ~0.2% fraudulent transactions. This can make it difficult for machine learning models to detect fraud. Class balancing techniques, such as oversampling or undersampling, can be used to address this issue.

sns.boxplot(df['Amount'])

<Axes: >

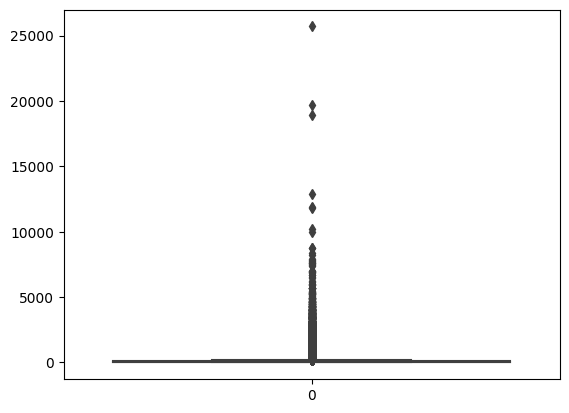


FIG 3:Count Graph

The credit card fraud detection class distribution count graph is a bar chart that shows the number of transactions in each class. The two classes are legitimate and fraudulent. The graph shows that the majority of transactions are legitimate, while only a small minority are fraudulent. This imbalance can make it difficult for machine learning models to detect fraudulent transactions.

sns.displot(df['Amount'])  
plt.show()

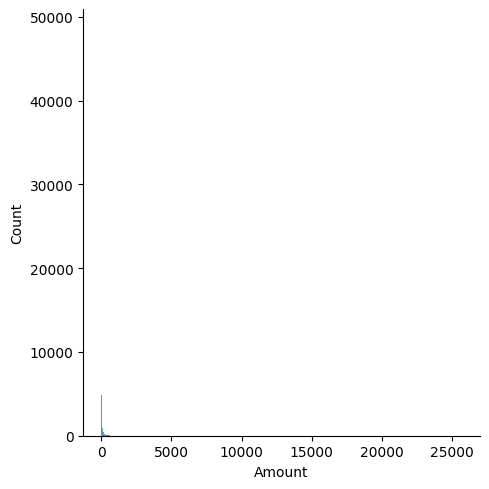


FIG 4: Distribution Graph

A correlation graph is a visualization that shows the relationship between two quantitative variables. It is typically a scatter plot, with the values of one variable on the horizontal axis and the values of the other variable on the vertical axis. Each point on the graph represents a single data point.

df['Amount'].skew()

16.978803370060476

**OBSERVATIONS**

* Data does not contains any NULL values but it does contains Duplicate Values.
* Data possesed very high positive skewness.
* Data is imbalanced.
* Data possesed Outliers.

pt = PowerTransformer(method='yeo-johnson')  
df['Amount'] = pt.fit\_transform(df[['Amount']])  
df['Amount'].skew()

0.018234140699062654

sns.distplot(df['Amount'])  
plt.show()

<ipython-input-23-ae7ad042f4b6>:1: UserWarning:   
  
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.  
  
Please adapt your code to use either `displot` (a figure-level function with  
similar flexibility) or `histplot` (an axes-level function for histograms).  
  
For a guide to updating your code to use the new functions, please see  
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751  
  
 sns.distplot(df['Amount'])

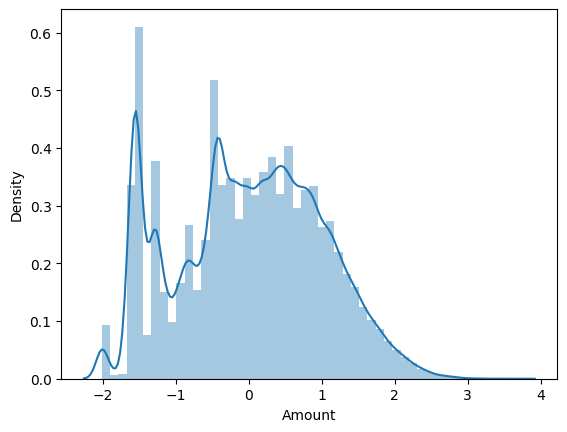


FIG 5:Distribution of time features

The strength of the correlation between the two variables can be inferred by the pattern of the points on the graph. If the points form a tight cluster, then there is a strong correlation between the two variables. If the points are spread out randomly, then there is a weak correlation between the two variables.

scaler = StandardScaler()  
  
df['Amount'] = scaler.fit\_transform(df[['Amount']])  
  
sns.boxplot(df['Amount'])

<Axes: >

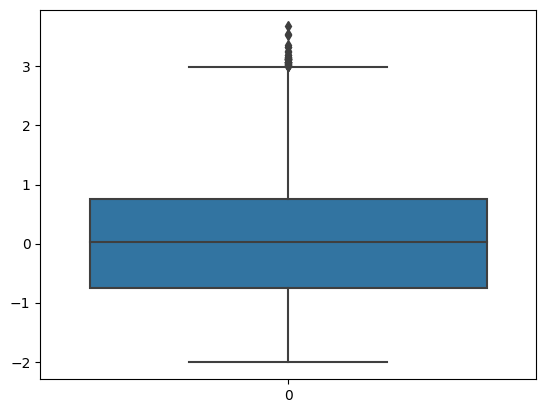


FIG 6: Cummulative Graph

The direction of the correlation can also be inferred by the pattern of the points on the graph. If the points slope upwards from left to right, then there is a positive correlation between the two variables. This means that as the value of one variable increases, the value of the other variable also tends to increase.

outliers = df['Amount'] > 3  
outliers.count()

283726

df['Amount']=df['Amount'] < 3  
class\_count

0 284315  
1 492  
Name: Class, dtype: int64

sns.heatmap(df.corr())  
plt.show()

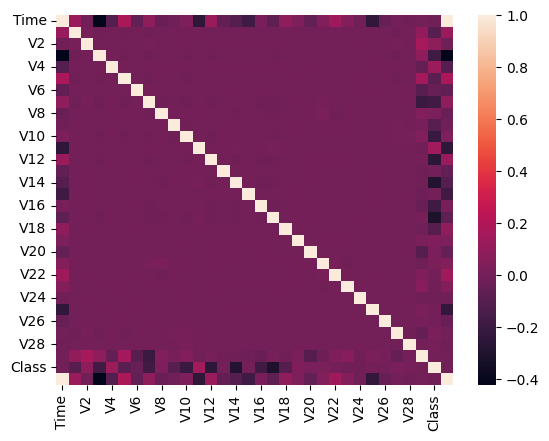


FIG 7:Correletion Graph

A scatter plot with two axes: x-axis for transaction amount and y-axis for fraud probability.Each point on the graph represents a single credit card transaction.A strong, positive correlation is shown by a tight cluster of points sloping upwards from left to right.This means that as the transaction amount increases, the probability of fraud also increases.This graph can be used to identify fraudulent transactions by identifying transactions with high transaction amounts and high fraud probabilities.

df.columns

Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',  
 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',  
 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',  
 'Class', 'hour'],  
 dtype='object')

x = df.drop(columns = ['Class'])  
y = df['Class']

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y, train\_size=0.8,random\_state =101, stratify =y)

# 

# LOGISTIC REGRESSION

model = LogisticRegression(class\_weight = 'balanced')  
  
model.fit(x\_train,y\_train)

LogisticRegression(class\_weight='balanced')

train\_pred = model.predict(x\_train)

print(classification\_report(y\_train,train\_pred))

precision recall f1-score support  
  
 0 1.00 0.96 0.98 226602  
 1 0.03 0.88 0.07 378  
  
 accuracy 0.96 226980  
 macro avg 0.52 0.92 0.52 226980  
weighted avg 1.00 0.96 0.98 226980

test\_pred = model.predict(x\_test)

print(classification\_report(y\_test,test\_pred))

precision recall f1-score support  
  
 0 1.00 0.96 0.98 56651  
 1 0.03 0.84 0.06 95  
  
 accuracy 0.96 56746  
 macro avg 0.52 0.90 0.52 56746  
weighted avg 1.00 0.96 0.98 56746

# Random Forest Classifier

model\_r = RandomForestClassifier(class\_weight = 'balanced')

model\_r.fit(x\_train,y\_train)

RandomForestClassifier(class\_weight='balanced')

trainpred = model\_r.predict(x\_train)  
print(classification\_report(y\_train,trainpred))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 226602  
 1 1.00 1.00 1.00 378  
  
 accuracy 1.00 226980  
 macro avg 1.00 1.00 1.00 226980  
weighted avg 1.00 1.00 1.00 226980

testpred = model\_r.predict(x\_test)  
  
print(classification\_report(y\_test,testpred))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 56651  
 1 0.96 0.71 0.81 95  
  
 accuracy 1.00 56746  
 macro avg 0.98 0.85 0.91 56746  
weighted avg 1.00 1.00 1.00 56746

# XGBoost Classifier

model\_X = xgb.XGBClassifier()  
model\_X.fit(x\_train,y\_train)

XGBClassifier(base\_score=None, booster=None, callbacks=None,  
 colsample\_bylevel=None, colsample\_bynode=None,  
 colsample\_bytree=None, device=None, early\_stopping\_rounds=None,  
 enable\_categorical=False, eval\_metric=None, feature\_types=None,  
 gamma=None, grow\_policy=None, importance\_type=None,  
 interaction\_constraints=None, learning\_rate=None, max\_bin=None,  
 max\_cat\_threshold=None, max\_cat\_to\_onehot=None,  
 max\_delta\_step=None, max\_depth=None, max\_leaves=None,  
 min\_child\_weight=None, missing=nan, monotone\_constraints=None,  
 multi\_strategy=None, n\_estimators=None, n\_jobs=None,  
 num\_parallel\_tree=None, random\_state=None, ...)

train\_x\_pred = model\_X.predict(x\_train)  
print(classification\_report(y\_train,train\_x\_pred))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 226602  
 1 1.00 1.00 1.00 378  
  
 accuracy 1.00 226980  
 macro avg 1.00 1.00 1.00 226980  
weighted avg 1.00 1.00 1.00 226980

test\_x\_pred = model\_r.predict(x\_test)  
  
print(classification\_report(y\_test,test\_x\_pred))

precision recall f1-score support  
  
 0 1.00 1.00 1.00 56651  
 1 0.96 0.71 0.81 95  
  
 accuracy 1.00 56746  
 macro avg 0.98 0.85 0.91 56746  
weighted avg 1.00 1.00 1.00 56746

**Conclusion**:

Data preprocessing is a vital step in the development of a credit card fraud detection system. It involves cleaning, transforming, and organizing the data to prepare it for machine learning model training. By handling missing values, performing feature engineering, scaling data, addressing class imbalance, and protecting privacy, data preprocessing ensures the data is ready for effective fraud detection. This process significantly contributes to the accuracy and reliability of the final model, making it a fundamental component of credit card fraud prevention systems.