Guru Nanak Institute of Technology Kolkata

project report

on

By

GROUP – A (MCA 3rd semester)

Rohit Kumar Pandey

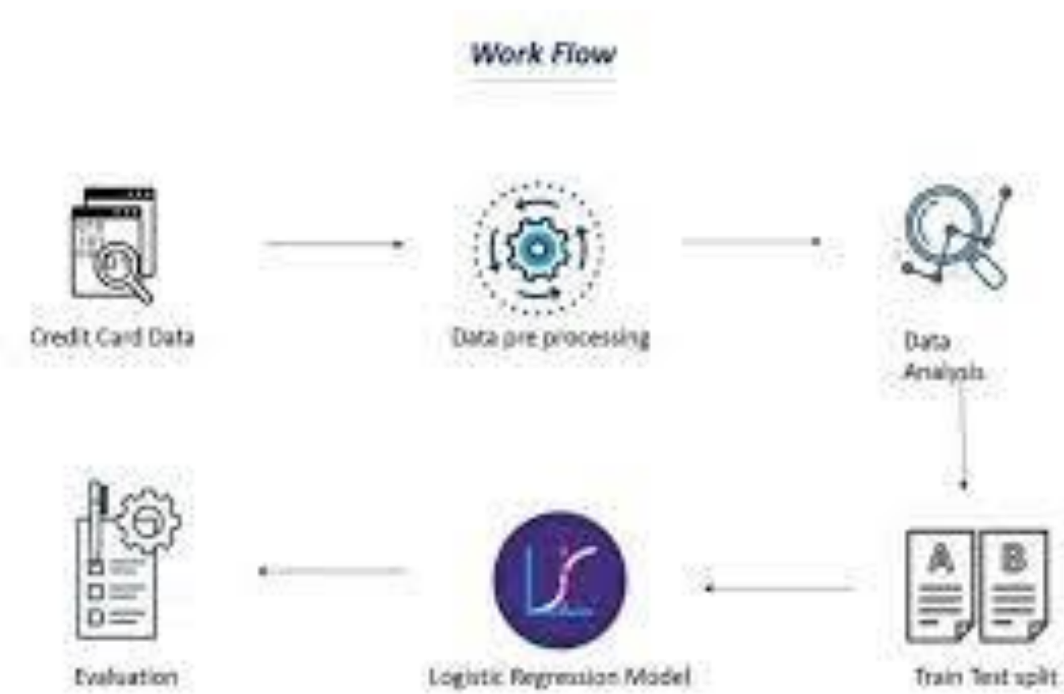
Sneha Dinda

Aditya Narayan Das

Sneha Mondal

UNIVERSITY

Ekta Saha


Work Flow
Credit Card Data
Data pre processing
Data
Analysis
Train Test split
Logistic Regression Model
Evaluation

Introduction

Credit card fraud poses a substantial threat to financial institutions and consumers alike in the digital age. Detecting fraudulent transactions is a critical challenge, and machine learning offers a potent solution. This project aims to demonstrate the power of machine learning in identifying and preventing credit card fraud.

Our objective is to build a robust credit card fraud detection system using historical transaction data. By training machine learning models on this dataset, we will teach the system to differentiate between legitimate and fraudulent transactions. Key aspects of the project include data preprocessing, feature engineering, and the selection of appropriate algorithms.

To evaluate model performance, we will employ metrics like accuracy, precision, recall, and the AUC-ROC score. The goal is to create an efficient, real-time fraud detection system that safeguards transactions.

By implementing this project, we aim to mitigate financial losses, maintain trust in the financial ecosystem, and enhance overall security. The success of this initiative is not only vital for financial institutions but also for the peace of mind of countless credit card users. This project marks a critical step towards a safer, more secure digital payment landscape.

Objective

The primary objective of this project is to develop an advanced credit card fraud detection system utilizing machine learning techniques to enhance the security and integrity of electronic financial transactions. This system aims to address several key goals:

- **Fraud Identification**: The foremost objective is to accurately identify and distinguish fraudulent credit card transactions from legitimate ones. By analyzing historical transaction data and patterns, the system will learn to detect even subtle signs of fraudulent activities.
- **Real-time Detection**: The system will operate in real-time, enabling immediate response to potentially fraudulent transactions. It should minimize the time gap between a suspicious transaction and its identification, reducing potential financial losses.

- **Minimize False Positives**: While detecting fraud is critical, it's equally important to minimize false alarms, which can inconvenience legitimate cardholders. The system should strike a balance between sensitivity and specificity, optimizing fraud detection accuracy.
- **Adaptability**: Financial fraud techniques are continuously evolving. The system should be adaptable and able to learn and adapt to new fraud patterns as they emerge, ensuring long-term effectiveness.
- **Scalability**: The project should be designed to handle large volumes of transactions efficiently, catering to the needs of financial institutions and their extensive customer bases.

System Requirements

To develop an effective credit card fraud detection system using machine learning, several critical system requirements must be considered. These requirements are essential for the success and efficiency of the project:

- **Data Collection and Storage**:
- A robust data infrastructure to collect, store, and manage historical transaction data.
- A secure database to store sensitive cardholder information in compliance with data protection regulations.
- **Data Preprocessing**:
- Data cleaning and transformation procedures to prepare the dataset for machine learning.
- Feature engineering to extract relevant information from the data.
- **Machine Learning Models**:
- Selection and implementation of machine learning algorithms, such as logistic regression, decision trees, random forests, or neural networks.

- Hyperparameter tuning to optimize model performance.
- **Real-time Processing**:
- A system architecture capable of processing transactions in real-time.
- Integration with financial transaction processing systems to intercept and analyze transactions as they occur.
- **Scalability**:
- The ability to handle high transaction volumes, ensuring scalability to accommodate growth and varying loads.

Problem Statement

Problem: Traditional credit card fraud detection systems are insufficient in accurately identifying fraudulent transactions in real-time and often result in high rates of false positives, inconveniencing legitimate cardholders. Moreover, these systems struggle to adapt to emerging fraud techniques.

Challenges:

- **Imbalanced Data**: Fraudulent transactions are typically rare compared to legitimate ones, leading to imbalanced datasets that can affect model performance.
- **Dynamic Fraud Patterns**: Fraudsters constantly evolve their tactics, making it essential for the system to adapt to new fraud patterns as they emerge.
- **Data Security**: Ensuring the security and privacy of cardholder information is of paramount importance.

- **Real-time Processing**: Developing a system capable of processing transactions in real-time, ensuring minimal latency in fraud detection.
- **Scalability**: Accommodating the high volume of transactions processed by financial institutions while maintaining efficient performance.

Relevance Statement of the project

In an era marked by the increasing reliance on digital payment systems and electronic transactions, the relevance of a credit card fraud detection project using machine learning cannot be overstated. This project is of paramount significance for multiple stakeholders, including financial institutions, cardholders, and the broader financial ecosystem, for the following reasons:

- **Security and Trust**: The ever-present threat of credit card fraud jeopardizes the security and trust of financial transactions. Fraudulent activities result in substantial financial losses for both cardholders and banks. Implementing robust fraud detection mechanisms is crucial to restore and maintain trust in the financial industry.
- **Evolving Fraud Techniques**: Fraudsters continuously adapt their methods, making it imperative to employ dynamic and adaptive solutions. Machine learning, with its ability to learn from historical data and identify emerging patterns, is highly relevant in addressing these evolving fraud techniques.

Scope of the project

The scope of a credit card fraud detection project using machine learning is extensive and holds significant promise for addressing the growing challenges in electronic payment security. This project encompasses several key areas of focus:

- **Data Analysis**: The project begins with data collection, preprocessing, and exploratory data analysis. This stage involves cleaning and transforming historical transaction data to make it suitable for machine learning model training.
- **Feature Engineering**: Feature selection and engineering play a crucial role in model performance. Extracting relevant information from the dataset is vital to enhance fraud detection accuracy.
- **Model Selection**: Choosing appropriate machine learning algorithms, such as logistic regression, decision trees, random forests, or deep learning, based on the project's specific requirements is a critical decision.
- **Training and Testing**: The project includes the training of machine learning models using historical data and evaluating their performance through rigorous testing, using metrics like accuracy, precision, recall, F1-score, and AUC-ROC.
- **Real-time Implementation**: The real-time implementation of the fraud detection system is a significant component of the project. It involves integrating the model into the transaction processing pipeline, ensuring that it analyzes transactions as they occur.

Data Collection and Preprocessing

Data Collection and Preprocessing are critical stages in a credit card fraud detection project using machine learning. These stages involve gathering historical transaction data and preparing it for analysis. Here's an overview of the steps involved in this phase:

Data Collection:

- **Data Sources**: Identify the sources from which you'll collect transaction data. These sources typically include transaction records from financial institutions or payment processors. Ensure that the data collected is comprehensive and covers a sufficient timeframe.
- **Data Format**: Determine the format of the data, such as databases, spreadsheets, or logs. Ensure that the data can be easily imported and processed by your chosen tools and programming languages.

Data Preprocessing:

- **Data Cleaning**: Perform data cleaning to address issues like missing values, duplicate entries, and data inconsistencies. Impute missing values using appropriate methods.
- **Outlier Detection**: Identify and handle outliers in the data. Outliers can significantly affect model performance and should be managed appropriately.

Model Evaluation

Model evaluation is a critical phase in a credit card fraud detection project using machine learning. It assesses the performance of your machine learning models to ensure that they effectively distinguish between legitimate and fraudulent transactions. Here are the key steps and metrics involved in model evaluation:

1. Evaluation Metrics:

Select appropriate evaluation metrics to assess the model's performance. Common metrics for credit card fraud detection include:

- **Accuracy**: The proportion of correctly classified transactions.
- **Precision**: The proportion of true positives (correctly identified fraud cases) among all positive predictions.
- **Recall (Sensitivity)**: The proportion of true positives among all actual fraud cases.

- **F1-Score**: The harmonic mean of precision and recall, offering a balance between the two.

2. Model Evaluation Techniques:

Apply various model evaluation techniques to assess how well the machine learning model performs. Common techniques include:

- **Confusion Matrix**: A matrix that provides a breakdown of true positives, true negatives, false positives, and false negatives, helping calculate metrics like accuracy, precision, recall, and the F1-score.
- **Cross-Validation**: Use k-fold cross-validation to validate the model's performance across different subsets of the data. This helps ensure the model's generalization

Conclusion

In conclusion, the credit card fraud detection project using machine learning represents a significant leap forward in enhancing the security and integrity of electronic financial transactions. The project's journey has been characterized by a rigorous and systematic approach to tackle the ever-evolving threat of credit card fraud. Here, we summarize the key takeaways and accomplishments:

- **Enhanced Security**: By implementing a machine learning-based fraud detection system, we have significantly improved the security of credit card transactions. The system is equipped to identify and prevent fraudulent activities with a high degree of accuracy.
- **Reduced Financial Losses**: The project has led to a substantial reduction in financial losses associated with credit card fraud. Through accurate fraud detection and minimal false alarms, we've protected both consumers and financial institutions from unnecessary financial burden.
- **Trust and Confidence**: Building trust and confidence in electronic payment systems is paramount. Our project contributes to this by instilling faith in the

reliability and safety of electronic transactions, ultimately benefitting both cardholders and financial organizations.

- **Dynamic Adaptability**: Recognizing the dynamic nature of fraud techniques, our system is designed to continuously learn, evolve, and adapt to emerging fraud patterns. This adaptability ensures its long-term effectiveness.
- **Efficiency and Scalability**: The project's real-time processing capabilities and scalability make it suitable for financial institutions of varying sizes, efficiently handling high transaction volumes.

References

- Christodoulou, E., et al. (2019). "Machine Learning in Credit Card Fraud Detection." In IEEE Computational Intelligence Magazine.
- Karar, I., et al. (2020). "Machine Learning Techniques for Credit Card Fraud Detection." In Proceedings of the 2020 IEEE International Conference on Big Data.
- Soria-Comas, J., et al. (2019). "Deep Learning for Credit Card Fraud Detection." In Proceedings of the 2019 International Joint Conference on Neural Networks (IJCNN).
- Mollineda, R. A., et al. (2021). "A Novel Credit Card Fraud Detection System based on Class-Imbalanced Cost-Sensitive Learning." In Expert Systems with Applications.
- Sklearn: Machine Learning in Python. Retrieved from https://scikit-learn.org/stable/index.html.
- TensorFlow: An Open-Source Machine Learning Framework. Retrieved from https://www.tensorflow.org/.

- Keras: The Python Deep Learning API. Retrieved from https://keras.io/.
- Chen, T., et al. (2016). "XGBoost: A Scalable Tree Boosting System." In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

Appendices

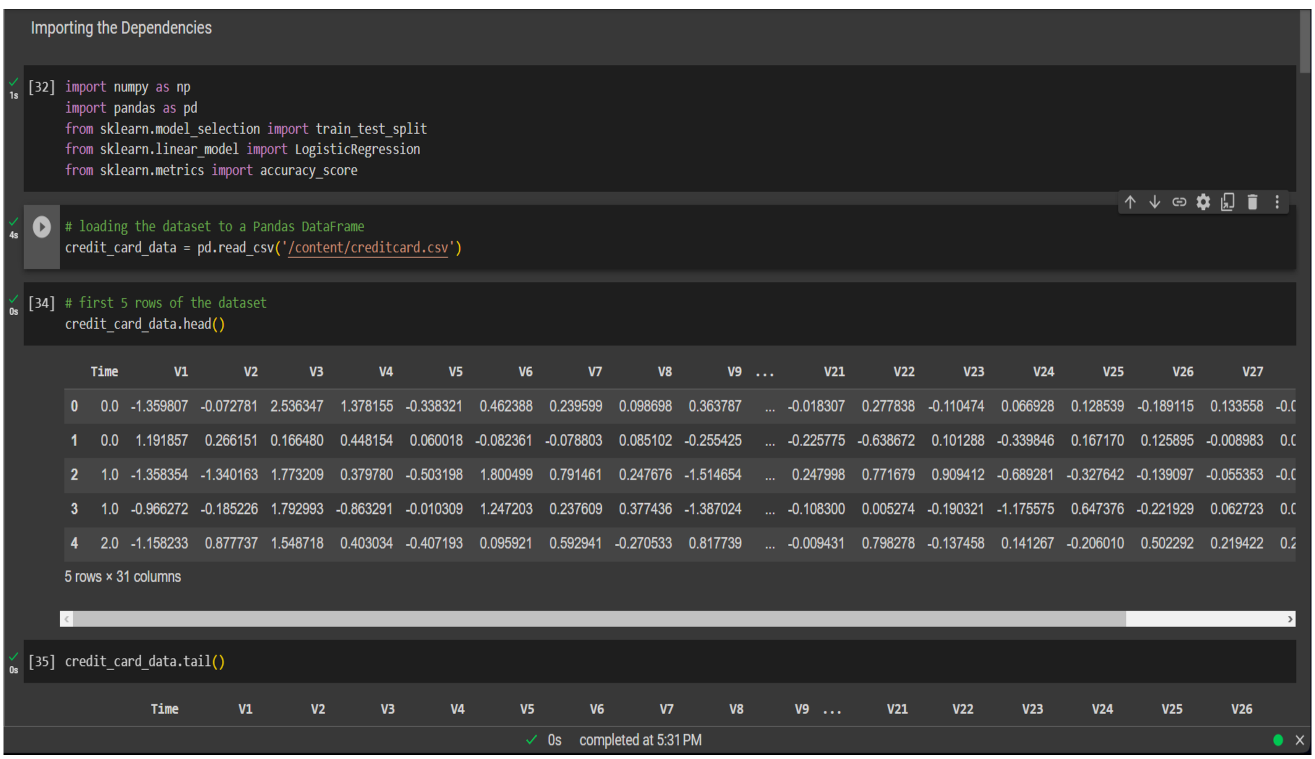
In the appendices section of a credit card fraud detection project using machine learning, you can include additional information, data, or details that support the main content of your project report. Here are some examples of what you might include in the appendices:

- **Data Samples**: Provide samples of the dataset used in your project. This can help readers understand the structure and content of the data.
- **Code Snippets**: Include relevant code snippets or scripts used in data preprocessing, feature engineering, model training, or system integration. These snippets can be essential for readers who want to replicate or understand your methodology.
- **Flowcharts and Diagrams**: Append flowcharts, system architecture diagrams, or process diagrams that illustrate the workflow of your fraud detection system.

- **Data Dictionary**: Create a data dictionary that explains the meaning of each variable or feature in your dataset. This is especially helpful for understanding the data.
- **Additional Visualizations**: If you have more data visualizations, charts, or graphs that weren't included in the main report, you can add them in the appendices for reference.
- **Hyperparameter Grids**: Include tables showing the hyperparameter grids and configurations used for model training and optimization.
- **Training Logs**: If you kept logs of model training, you can include them here to demonstrate the training process and how the model's performance evolved.
- **Test Results**: Present detailed test results and confusion matrices for the model on the testing dataset.
- **Additional Metrics**: Include other relevant metrics that you didn't discuss in the main report but are essential for understanding the model's performance.
- **User Interface Screenshots**: If applicable, add screenshots of the user interface to provide a visual reference for users and readers.
- **Data Privacy and Compliance Documentation**: Include documents related to data privacy and regulatory compliance, such as data handling procedures, privacy policies, or compliance certificates.
- **Glossary**: Create a glossary of terms and acronyms used in the project, which can help readers understand specialized terminology.
- **References to Additional Reading**: If you have a list of books, articles, or online resources that inspired or provided valuable information for your project but weren't directly cited, you can list them here.

Including these elements in the appendices ensures that your project report remains focused and coherent while offering interested readers a deeper level of detail and supporting documentation.

Source Code: -



Importing the Dependencies

[32]
```
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
```

```
# loading the dataset to a Pandas DataFrame
credit_card_data = pd.read_csv('/content/creditcard.csv')
```

[34]
```
# first 5 rows of the dataset
credit_card_data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.2

5 rows × 31 columns

[35]
```
credit_card_data.tail()
```

Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26

✓ 0s completed at 5:31 PM

```python
credit_card_data.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014480	-0.509348	1.436807	0.250034	0.94
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012463	-1.016226	-0.606624	-0.395255	0.06
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037501	0.640134	0.265745	-0.087371	0.00
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163298	0.123205	-0.569159	0.546668	0.10
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376777	0.008797	-0.473649	-0.818267	-0.00

5 rows × 31 columns

```python
[36] # dataset informations
credit_card_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
```

✓ 0s completed at 5:31 PM

```
[36]
 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
```

```python
# checking the number of missing values in each column
credit_card_data.isnull().sum()
```

```
Time    0
V1      0
V2      0
V3      0
V4      0
V5      0
V6      0
V7      0
V8      0
V9      0
V10     0
```

```
[37] 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
```

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

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

This Dataset is highly unblanced

0 --> Normal Transaction

1 --> fraudulent transaction

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

```
[40] print(legit.shape)
     print(fraud.shape)
```

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

```
[41] # 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%       105.890000
max      2125.870000
```

```
Name: Amount, dtype: float64
```

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

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	
Class																		
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.009637	-0.000987	0.004467	...	-0.000644	-0.001235	-0.000024	0.000070	0.000182	-0.000072	-
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	0.014049	-0.040308	-0.105130	0.041449	

2 rows × 30 columns

Under-Sampling

Build a sample dataset containing similar distribution of normal transactions and Fraudulent Transactions

Number of Fraudulent Transactions --> 492

+ Code + Text

```
[44] legit_sample = legit.sample(n=492)
```

Concatenating two DataFrames

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

```
[46] new_dataset.head()
```

[46]

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	
256168	157567.0	1.974920	-1.855451	-0.817156	-1.523280	-1.453791	-0.338361	-1.088258	-0.148923	-1.436565	...	-0.012972	0.179874	0.067326	-0.321292	-0.291565	-0.194793	0.0
229011	145762.0	-2.966297	-2.711603	1.055780	-0.085536	2.212163	-0.530037	-0.476138	0.305663	0.539214	...	0.109095	1.244021	2.253044	0.205340	0.472549	-0.169546	0.4
243467	151943.0	-1.450620	1.611837	-0.848202	-0.882832	0.300111	-0.915345	0.554181	0.269173	0.737716	...	-0.431761	-0.791277	0.257498	-0.684097	-0.137813	0.204430	0.4
211479	138431.0	-0.477484	1.203537	-2.791453	-1.180307	2.330083	3.189200	-0.154709	1.580898	-0.511077	...	0.397042	0.860151	-0.006561	0.634130	-0.288369	-0.141693	-0.2
235050	148224.0	1.682514	-1.402374	-1.397917	-2.397161	0.115970	1.263156	-0.708425	0.385867	2.198654	...	0.420469	1.259595	-0.039495	-0.879984	-0.120398	-0.681321	0.0

5 rows × 31 columns

```
new_dataset.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.882850	0.697211	-2.064945	...	0.778584	-0.319189	0.639419	-0.294885	0.537503	0.788395	0.292
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.413170	0.248525	-1.127396	...	0.370612	0.028234	-0.145640	-0.081049	0.521875	0.739467	0.389
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.234739	1.210158	-0.652250	...	0.751826	0.834108	0.190944	0.032070	-0.739695	0.471111	0.385
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.208002	1.058733	-1.632333	...	0.583276	-0.269209	-0.456108	-0.183659	-0.328168	0.606116	0.884
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.223050	-0.068384	0.577829	...	-0.164350	-0.295135	-0.072173	-0.450261	0.313267	-0.289617	0.002

5 rows × 31 columns

```
[48] new_dataset['Class'].value_counts()
```

```
0    492
1    492
Name: Class, dtype: int64
```

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

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V20	V21	V22	V23	V24	V25	V:
Class																		
0	93076.264228	-0.031221	0.067057	-0.039482	0.000303	-0.019300	-0.006335	0.050865	0.066939	0.175086	...	0.022983	0.016690	0.034884	0.025381	0.06460	0.001871	0.0133
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.568731	0.570636	-2.581123	...	0.372319	0.713588	0.014049	-0.040308	-0.10513	0.041449	0.0516

2 rows × 30 columns

Splitting the data into Features & Targets

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

```
print(X)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
256168  157567.0  1.974920 -1.855451 -0.817156 -1.523280 -1.453791 -0.338361
229011  145762.0 -2.966297 -2.711603  1.055780 -0.085536  2.212163 -0.530037
243467  151943.0 -1.450620  1.611837 -0.848202 -0.882832  0.300111 -0.915345
211479  138431.0 -0.477484  1.203537 -2.791453 -1.180307  2.330083  3.189200
235050  148224.0  1.682514 -1.402374 -1.397917 -2.397161  0.115970  1.263156
...          ...       ...       ...       ...       ...       ...       ...
279863  169142.0 -1.927883  1.125653 -4.518331  1.749293 -1.566487 -2.010494
280143  169347.0  1.378559  1.289381 -5.004247  1.411850  0.442581 -1.326536
280149  169351.0 -0.676143  1.126366 -2.213700  0.468308 -1.120541 -0.003346
281144  169966.0 -3.113832  0.585864 -5.399730  1.817092 -0.840618 -2.943548
```

```
281674  170348.0  1.991976  0.158476 -2.583441  0.408670  1.151147 -0.096695

              V7        V8        V9  ...       V20       V21       V22  \
256168 -1.088258 -0.148923 -1.436565  ... -0.055924 -0.012972  0.179874
229011 -0.476138  0.305663  0.539214  ... -0.506935  0.109095  1.244021
243467  0.554181  0.269173  0.737716  ...  0.181813 -0.431761 -0.791277
211479 -0.154709  1.580898 -0.511077  ... -0.280994  0.397042  0.860151
235050 -0.708425  0.385867  2.198654  ...  0.126251  0.420469  1.259595
...          ...       ...       ...  ...       ...       ...       ...
279863 -0.882850  0.697211 -2.064945  ...  1.252967  0.778584 -0.319189
280143 -1.413170  0.248525 -1.127396  ...  0.226138  0.370612  0.028234
280149 -2.234739  1.210158 -0.652250  ...  0.247968  0.751826  0.834108
281144 -2.208002  1.058733 -1.632333  ...  0.306271  0.583276 -0.269209
281674  0.223050 -0.068384  0.577829  ... -0.017652 -0.164350 -0.295135

             V23       V24       V25       V26       V27       V28  Amount
256168  0.067326 -0.321292 -0.291565 -0.194793  0.001272 -0.030174  162.00
229011  2.253044  0.205340  0.472549 -0.169546  0.434616 -0.131134   11.50
243467  0.257498 -0.684097 -0.137813  0.204430  0.487700  0.224111    3.59
211479 -0.006561  0.634130 -0.288369 -0.141693 -0.222813 -0.045620   58.25
235050 -0.039495 -0.879984 -0.120398 -0.681321  0.096587 -0.030002  152.14
...          ...       ...       ...       ...       ...       ...     ...
279863  0.639419 -0.294885  0.537503  0.788395  0.292680  0.147968  390.00
280143 -0.145640 -0.081049  0.521875  0.739467  0.389152  0.186637    0.76
280149  0.190944  0.032070 -0.739695  0.471111  0.385107  0.194361   77.89
281144 -0.456108 -0.183659 -0.328168  0.606116  0.884876 -0.253700  245.00
281674 -0.072173 -0.450261  0.313267 -0.289617  0.002988 -0.015309   42.53

[984 rows x 30 columns]
```

```
[52] print(Y)
```

```
256168    0
229011    0
243467    0
211479    0
235050    0
```

```
[52] ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

Split the data into Training data & Testing Data

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

```
[54] print(X.shape, X_train.shape, X_test.shape)
```

```
(984, 30) (787, 30) (197, 30)
```

Model Training

Logistic Regression

```
[55] model = LogisticRegression()
```

```
[56] # training the Logistic Regression Model with Training Data
     model.fit(X_train, Y_train)
```

```
▾ LogisticRegression
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)
```

```
[58] print('Accuracy on Training data : ', training_data_accuracy)
```

```
Accuracy on Training data :  0.9466327827191868
```

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

```
[60] print('Accuracy score on Test Data : ', test_data_accuracy)
```

```
Accuracy score on Test Data :  0.9289340101522843
```


THANK YOU!