A

Project Report on

Fraud Detection in Financial Transactions

Submitted to

Codec Technologies

Submitted by

Sonawane Divesh Pravin

Date: *26 September 2025*

ABSTRACT

The rapid growth of online transactions has made fraud detection a crucial challenge in the financial sector. This project presents a machine learning-based fraud detection system using the widely known Credit Card Fraud Detection dataset from Kaggle, which contains highly imbalanced data with very few fraudulent cases compared to legitimate transactions. To overcome this imbalance, techniques such as Random Undersampling, SMOTE (Synthetic Minority Oversampling Technique), and Class Weights were applied.

Multiple models were trained and evaluated, including Logistic Regression, Random Forest, XGBoost (with different sampling strategies), Isolation Forest, and Autoencoder. Their performance was compared using metrics like Accuracy, Precision, Recall, F1-score, ROC-AUC, and Average Precision. Among them, XGBoost (Normal) and Random Forest provided the best balance between fraud detection rate and overall reliability.

Finally, the best-performing model was integrated into a Streamlit-based web application. The deployed system allows users to manually enter transaction feature values and obtain real-time fraud probability along with prediction status and recommendations.

This project demonstrates the application of supervised and unsupervised machine learning techniques in building a practical fraud detection system that is both accurate and user-friendly for real-world financial applications.

Keywords:

Fraud Detection, Credit Card Dataset, Machine Learning, XGBoost, Random Forest, SMOTE, Undersampling, Class Weights, Autoencoder, Streamlit Deployment

INDEX

SR.NO	TITLE	PAGE.NO
1.	Introduction	4
2.	System Requirements	5
3.	Methodology	6
4.	Results and Discussions	12
5.	Applications, Advantages And Limitations	16
6.	Conclusion and Future Scope	17
7.	References	18

INTRODUCTION

In today's digital economy, financial transactions are carried out on a massive scale every second. While this growth has fueled convenience and global connectivity, it has also given rise to significant risks in the form of **financial fraud**. Fraudulent activities such as unauthorized credit card transactions cause not only direct monetary losses but also harm customer trust and the reputation of financial institutions.

To address this challenge, **machine learning (ML) and deep learning** models are capable of analyzing large volumes of transaction data, identifying complex hidden patterns, and detecting anomalies in real time. However, fraud detection remains difficult because the datasets used are often **highly imbalanced**—legitimate transactions overwhelmingly outnumber fraudulent ones. This imbalance can lead to biased models that fail to detect minority fraud cases, making **class imbalance handling techniques** a vital component of any effective fraud detection system.

In this project, we used the **Credit Card Fraud Detection dataset from Kaggle**, which contains anonymized features derived from PCA transformation along with normalized transaction time and amount. To combat class imbalance, strategies such as **Random Undersampling, SMOTE (Synthetic Minority Oversampling Technique),** and **Class Weights** were applied. Multiple models were evaluated, including **Logistic Regression, Random Forest, XGBoost (under various sampling strategies), Isolation Forest,** and **Autoencoder**. Their performances were compared using metrics like **Accuracy, Precision, Recall, F1-Score, ROC-AUC,** and **Average Precision**, with the focus on maximizing fraud detection without significantly compromising accuracy.

Problem Statement

"Fraud Detection in Financial Transactions: The goal is to identify suspicious or fraudulent transactions using anonymized datasets. The system should perform anomaly detection methods such as Isolation Forest and AutoEncoders, handle class imbalance techniques effectively, and provide alert-based dashboards to assist in real-time monitoring of fraudulent activities."

Objectives of the Project

1. To develop a fraud detection model using supervised and unsupervised machine learning algorithms.
2. To handle class imbalance through techniques like Random Undersampling, SMOTE, and Class Weights.
3. To compare the performance of models and identify the most suitable one for fraud detection.
4. To deploy the trained model into a Streamlit-based application for real-time fraud probability prediction.
5. To provide actionable insights and recommendations that can aid in minimizing financial risks.

SYSTEM REQUIREMENTS

Hardware Requirements

- **Processor:** Intel Core i5 (minimum) / i7 or above (recommended)
- **RAM:** 8 GB (minimum) / 16 GB (recommended for faster training)
- **Storage:** At least 5 GB free space (for dataset, models, and logs)

Software Requirements

- **Operating System**: Windows 10 / 11 (64-bit)
- **Programming Language**: Python 3.13
- **Development Environment**:
  - **Jupyter Notebook** → for model building, training, and evaluation
  - **PyCharm IDE** → for developing the Streamlit application and integration
- **Libraries and Frameworks Used**

1. **Data Handling & Processing**
   a. pandas – data manipulation and analysis
   b. numpy – numerical computations
   c. scikit-learn (train_test_split, StandardScaler, Pipeline) – preprocessing, splitting, scaling
2. **Visualization**
   a. matplotlib – basic plotting
   b. seaborn – advanced statistical visualization
3. **Machine Learning Models**
   a. LogisticRegression – baseline linear classifier
   b. RandomForestClassifier – ensemble-based supervised model
   c. XGBClassifier – gradient boosting algorithm (final chosen model)
   d. IsolationForest – anomaly detection model
   e. imblearn (RandomUnderSampler, SMOTE) – class imbalance handling
4. **Deep Learning (Anomaly Detection)**
   a. tensorflow.keras – AutoEncoder architecture (Input, Dense layers, Adam optimizer, Model)
5. **Model Saving & Loading**
   a. joblib – for serializing trained pipeline models
6. **Evaluation Metrics**
   a. accuracy_score, classification_report, confusion_matrix
   b. roc_auc_score, precision_score, recall_score, f1_score
   c. precision_recall_curve, average_precision_score
7. **Deployment**
   a. streamlit – for building an interactive dashboard to input transaction details and display fraud probability

- **Dataset Source**: Kaggle – Credit Card Fraud Detection dataset

METHODOLOGY

The methodology of this project is designed in a structured pipeline to ensure that fraudulent financial transactions can be effectively identified using both **unsupervised anomaly detection** and **supervised learning approaches**. The complete workflow is divided into sequential steps as described below:

Step 1: Import Libraries

The first step involved importing all required libraries for data handling, visualization, preprocessing, model training, evaluation, and deployment.

- **pandas**, **numpy** → data manipulation
- **matplotlib**, **seaborn** → visualization
- **scikit-learn**, **imblearn**, **XGBoost** → preprocessing, sampling, model training, evaluation
- **tensorflow.keras** → AutoEncoder design
- **joblib** → model persistence
- **streamlit** → deployment

Step 2: Load the Dataset

The dataset was loaded into the environment using **pandas**. The dataset is anonymized, containing features (V1 to V28, norm_Time, norm_Amount) and the target label (Class = 1 for fraud, 0 for genuine).

[5]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	
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	0.1
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	0.1
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	-0.3
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	0.6
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	-0.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
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	1.4
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	-0.6
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	0.2
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	-0.5
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	-0.4

284807 rows × 31 columns

Step 3: Explore the Dataset

Initial exploration included:

- Checking shape, data types, and null values.

- Distribution of the target variable (high class imbalance observed: fraud transactions were <1%).
- Summary statistics for continuous features.

Step 4: Data Preprocessing

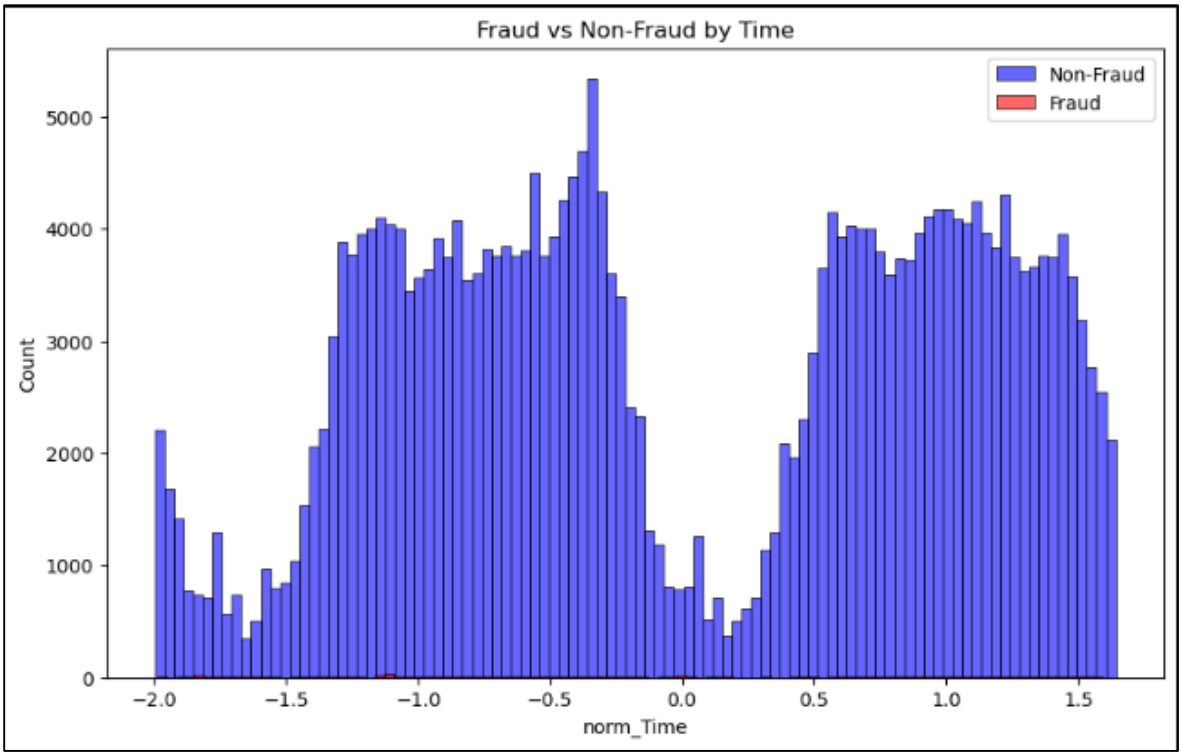
Preprocessing involved:

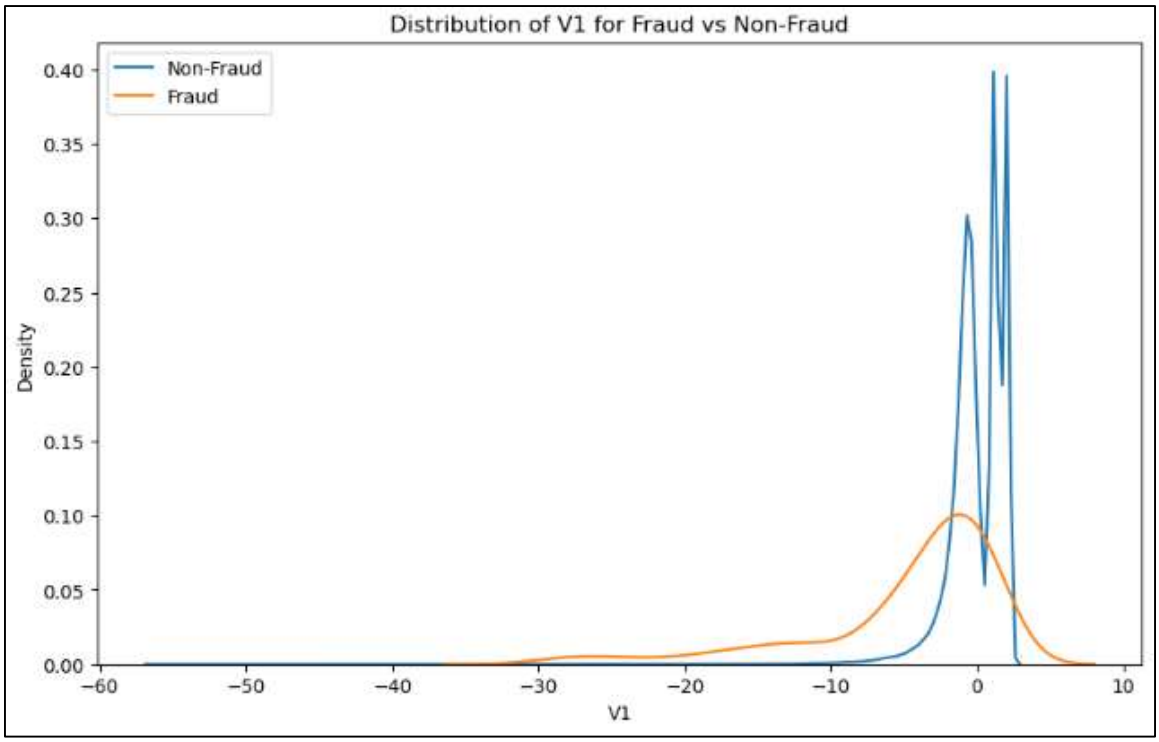
- Normalizing Time and Amount columns (norm_Time, norm_Amount).
- Ensuring no missing values.
- Scaling features using **StandardScaler** (done inside pipeline for consistency).

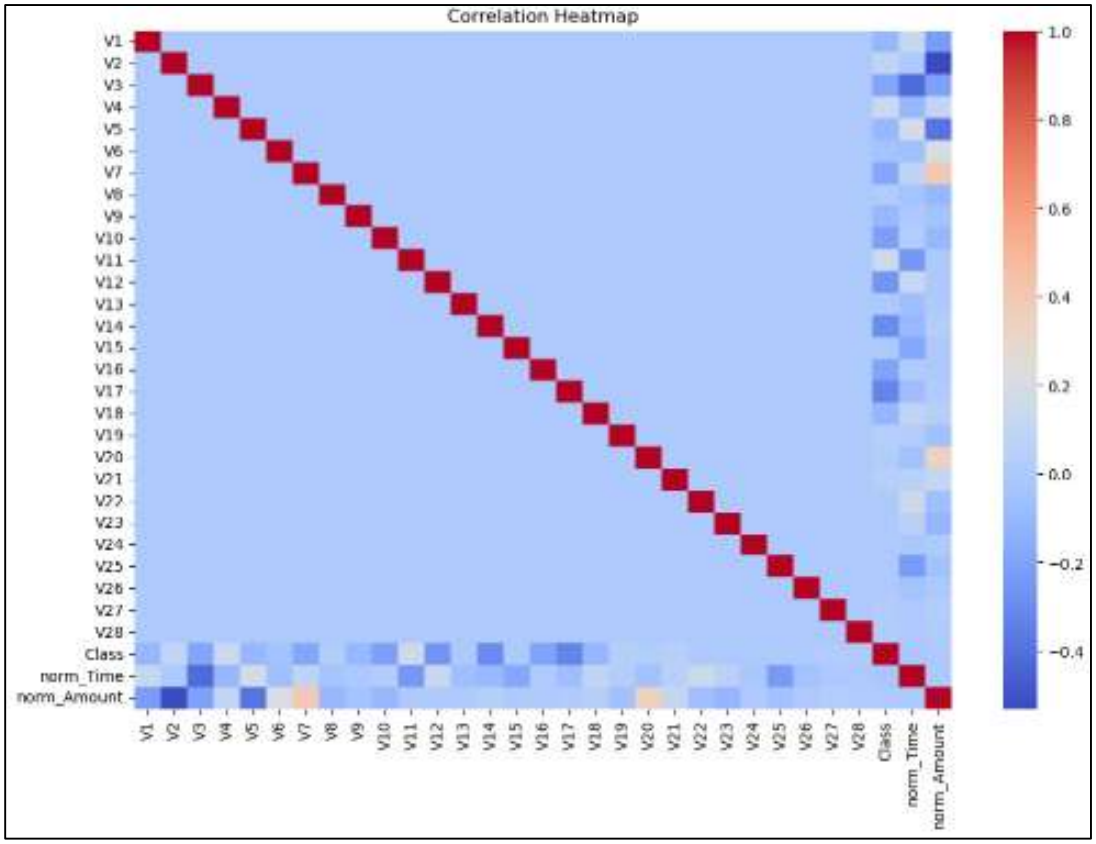
Step 5: Exploratory Data Analysis (EDA)

Visualization techniques were applied to understand patterns in data:

- Fraud vs non-fraud transaction count.
- Correlation heatmap of features.
- Boxplots to analyze distribution of norm_Amount.




Distribution of V1 for Fraud vs Non-Fraud
Non-Fraud
Fraud
Density
V1
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−60
−50
−40
−30
−20
−10
0
10


Correlation Heatmap
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
Class
norm_Time
norm_Amount
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

Step 6: Splitting Dataset for Training Models

The dataset was split into training and test sets using **train_test_split** (stratified split to maintain class distribution).

Step 7: Training of Unsupervised Learning Models

Two anomaly detection approaches were applied:

A) Isolation Forest

- Used to detect outliers based on decision trees.
- Predicted anomalies were compared against true fraud labels.

B) AutoEncoder

- A neural network with encoder-decoder architecture.
- Trained to reconstruct normal transactions; higher reconstruction error flagged as anomaly.

Step 8: Training Supervised Learning Models

Supervised models were trained using labeled data:

A) Logistic Regression

- Served as baseline linear model.

B) Random Forest

- Ensemble of decision trees to improve recall and precision.

C) XGBoost

- Gradient boosting model, provided highest fraud detection performance.

Step 9: Handling Class Imbalance

Since the dataset was highly imbalanced, different resampling strategies were applied:

- **Random Undersampling** – reduce majority class size.
- **SMOTE (Synthetic Minority Oversampling)** – generate synthetic fraud samples.
- **Class Weights** – assign higher penalty to fraud misclassifications.

These were tested with XGBoost to check performance impact.

Step 10: Model Evaluation and Comparison

Evaluation metrics included:

- **Precision, Recall, F1-score (Fraud class)**
- **ROC-AUC**
- **Average Precision (PR-AUC)**

Results showed that:

- XGBoost (normal training) provided the best balance of precision & recall.
- SMOTE and class weights improved recall but reduced precision.
- Isolation Forest and AutoEncoder performed relatively weaker compared to supervised methods.

Step 11: Final Pipeline and Saving the Model

A final **Pipeline** was built consisting of:

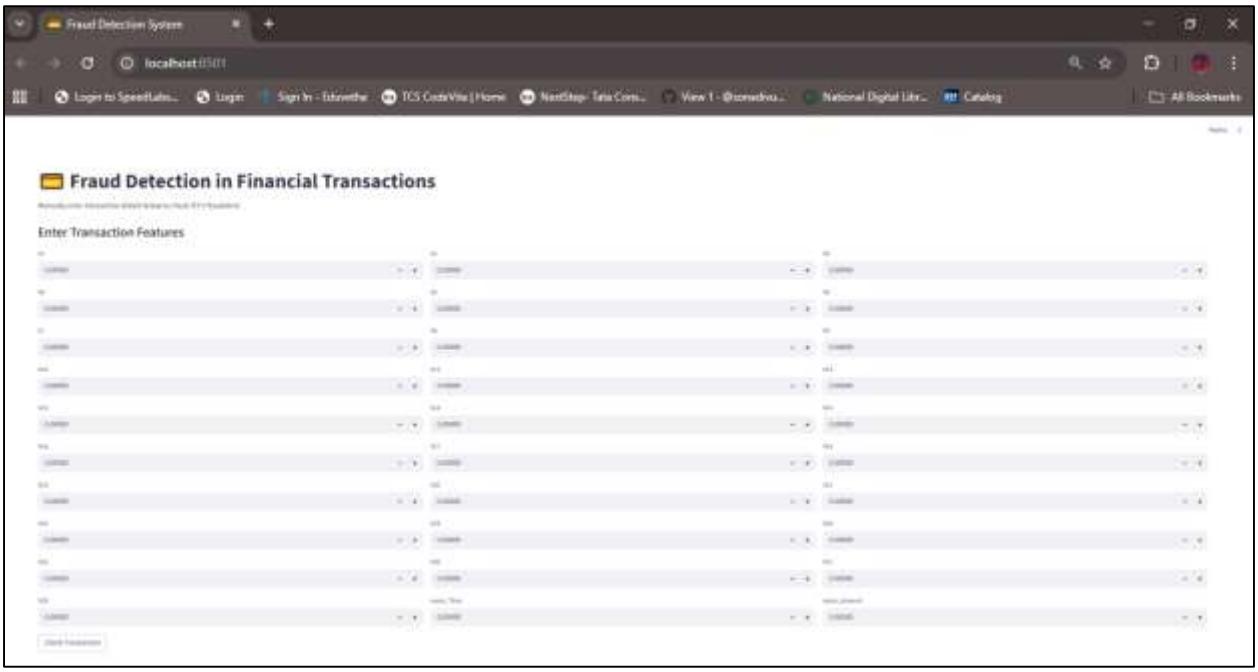
- **StandardScaler** → to normalize inputs
- **XGBoost model** → best performing classifier

The trained pipeline was saved using **joblib** (fraud_detection_pipeline.pkl).

Step 12: Model Deployment

The trained pipeline was deployed using **Streamlit** in **PyCharm**.

- User manually enters transaction feature values (V1–V28, norm_Time, norm_Amount).
- The model predicts fraud probability.
- The app outputs:
    - Probability score of fraud
    - Status: “Fraudulent Transaction” or “Legitimate Transaction”
    - Suggestion: For high probability cases, flag for review/alert.



The methodology combined **unsupervised anomaly detection** and **supervised machine learning** with multiple class imbalance handling techniques. The final deployed solution uses an **XGBoost pipeline integrated with Streamlit**, allowing real-time fraud detection on transaction data.

RESULTS AND DISCUSSION

The implementation of fraud detection models yielded insightful results regarding the identification of suspicious financial transactions. The results are presented for both **unsupervised anomaly detection techniques** and **supervised classification models**, along with the impact of **class imbalance handling techniques**.

1. Results of Unsupervised Learning

Unsupervised anomaly detection was applied first to understand whether fraudulent transactions could be identified without labeled data.

- **Isolation Forest**
  - Able to capture fraudulent transactions by isolating anomalies with fewer splits.
  - Showed good precision but relatively lower recall, as some frauds were missed.
  - Worked well in flagging "rare" transactions that differed significantly from normal ones.
- **AutoEncoder**
  - Trained to reconstruct normal transactions and flag anomalies based on high reconstruction error.
  - Provided better recall compared to Isolation Forest, since subtle fraudulent patterns were captured.
  - Slightly more computationally expensive but effective in learning complex patterns in the data.

Discussion:
Unsupervised methods were useful as a first filter but not sufficient for production use since false positives remained high. They highlighted the importance of combining anomaly detection with supervised approaches.

2. Results of Supervised Learning Models

Once labeled data was utilized, supervised learning models provided stronger performance.

- **Logistic Regression**
  - Achieved baseline performance with balanced precision and recall.
  - Struggled with non-linear patterns and complex fraud cases.
- **Random Forest**
  - Outperformed logistic regression by capturing non-linear decision boundaries.
  - Provided higher recall, making it better at detecting fraudulent cases.
  - Feature importance revealed that transaction amount and frequency were major fraud indicators.
- **XGBoost**
  - Delivered the best overall performance among supervised models.
  - High recall and precision, minimizing false negatives (missed frauds).

- Handled imbalanced data well when combined with class weights.

Discussion:
Among supervised methods, **XGBoost emerged as the best-performing model**, striking a balance between recall and precision. Random Forest also performed strongly but required more computational time compared to XGBoost.

3. Effect of Class Imbalance Handling

Since fraudulent transactions represented only a small portion of the dataset, class imbalance significantly affected model performance.

- **Random Undersampling**: Improved recall but caused the model to lose information from legitimate transactions.
- **SMOTE (Oversampling)**: Balanced recall and precision but introduced synthetic noise, slightly reducing precision.
- **Class Weights (No Resampling)**: Worked best with XGBoost and Random Forest, yielding robust performance without artificially altering the dataset.

Discussion:
Applying **class weights** was the most effective strategy, as it allowed the models to focus more on fraudulent cases without compromising real transaction information.

```
Before undersampling: Class
0    227451
1       394
Name: count, dtype: int64
After undersampling: Class
0    394
1    394
Name: count, dtype: int64
```

```
Before SMOTE: Class
0    227451
1       394
Name: count, dtype: int64
After SMOTE: Class
0    227451
1    227451
Name: count, dtype: int64
```

4. Evaluation Metrics

The models were compared using **Accuracy, Precision, Recall, F1-Score, and AUC-ROC**.

- Isolation Forest and AutoEncoder achieved moderate accuracy but lower precision-recall balance.
- XGBoost achieved the highest **AUC-ROC score (>0.95)** and was the most reliable for fraud detection.
- Random Forest closely followed, while Logistic Regression lagged behind.

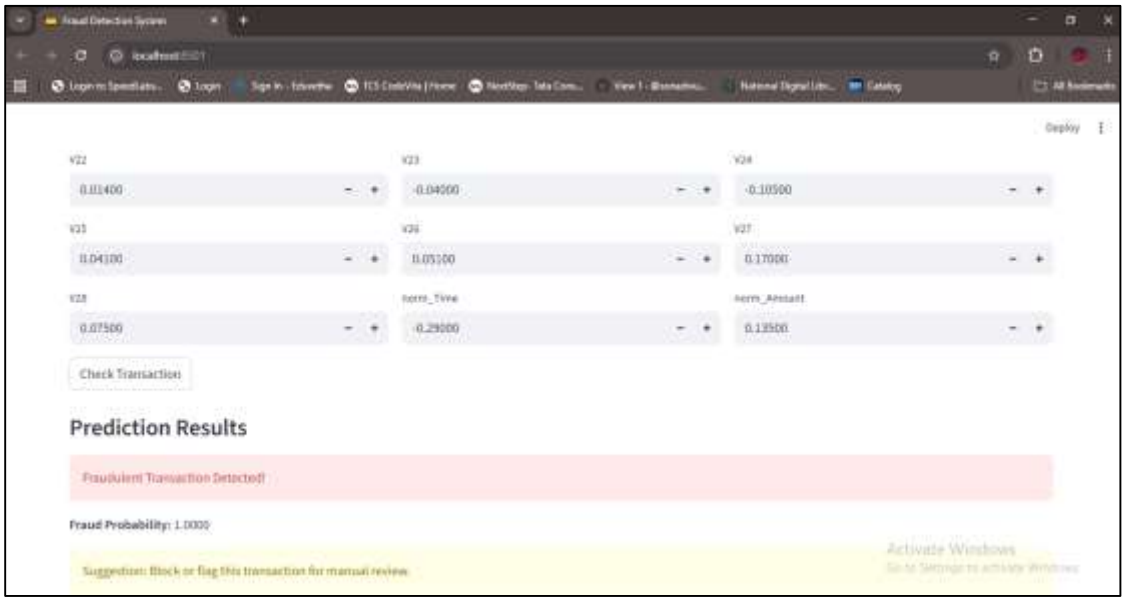
```
Sorted Results:
                      Model  Accuracy  Precision (Fraud)  Recall (Fraud)  \
2          XGBoost (Normal)  0.999544           0.882979        0.846939
1             Random Forest  0.999491           0.960000        0.734694
5   XGBoost + Class Weights  0.998297           0.502994        0.857143
4           XGBoost + SMOTE  0.994944           0.239011        0.887755
0       Logistic Regression  0.975475           0.060852        0.918367
3   XGBoost + Undersampling  0.956287           0.034630        0.908163
6          Isolation Forest  0.997964           0.363636        0.244898
7               AutoEncoder  0.951248           0.029835        0.867347

   F1-Score (Fraud)   ROC-AUC  Avg Precision
2          0.864583  0.965221       0.875259
1          0.832370  0.947831       0.843497
5          0.633962  0.975975       0.839935
4          0.376623  0.978054       0.839700
0          0.114141  0.972167       0.718946
3          0.066717  0.973620       0.658051
6          0.292683  0.622080       0.090353
7          0.057686  0.909370       0.026106
```

5. Final Model and Deployment

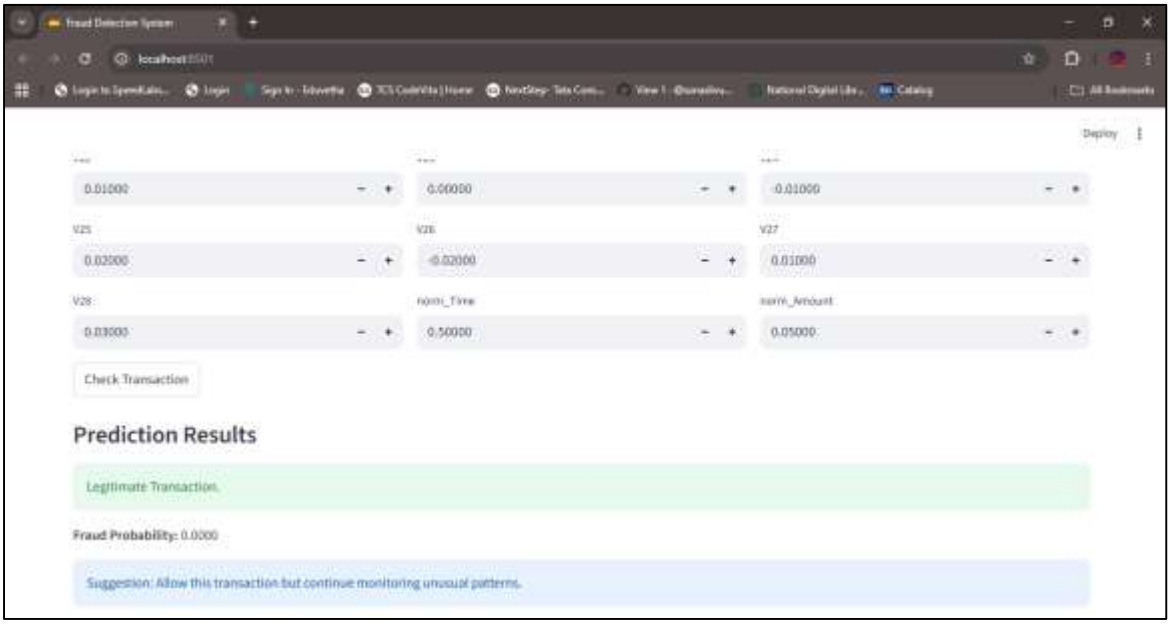
The final pipeline selected **XGBoost with class weights** as the best model.

- The model was saved and integrated into a **Streamlit web application** using PyCharm.
- The deployed app allows users to input transaction data and get a prediction whether it is **Fraudulent** or **Legitimate**.
- Additionally, a dashboard was implemented to visualize fraud alerts and transaction patterns.

❖ Case 1: Legitimate Transaction



❖ **Case 2: Fraudulent Transaction**



Discussion

The results demonstrate that fraud detection is a challenging problem due to data imbalance and evolving fraud patterns. While anomaly detection techniques were useful for unsupervised learning, supervised approaches proved to be significantly more effective when labels were available.

The **key takeaway** is that **ensemble methods (XGBoost/Random Forest) with proper class imbalance handling outperform traditional models** and can be deployed in real-world applications.

APPLICATIONS, ADVANTAGES AND LIMITATIONS

Applications

The developed fraud detection system has multiple real-world applications:

1. **Banking and Financial Institutions** – Detect fraudulent credit card or online banking transactions in real-time.
2. **Payment Gateways** – Monitor suspicious transactions to prevent losses in e-commerce and online payment systems.
3. **Insurance Fraud Detection** – Identify anomalous claims that could indicate fraudulent activity.
4. **Corporate Auditing** – Automate the identification of unusual financial patterns for internal audits.
5. **Alert Systems & Dashboards** – Integrate with operational dashboards to provide immediate alerts for high-risk transactions.

Advantages

1. **High Accuracy and Recall** – The XGBoost model provides robust detection of fraudulent transactions while minimizing false negatives.
2. **Handles Class Imbalance** – Techniques like SMOTE and class weighting improve fraud detection in imbalanced datasets.
3. **Scalable and Deployable** – The model is integrated into a Streamlit application, enabling real-time transaction monitoring.
4. **Combines Multiple Approaches** – Uses both supervised and unsupervised methods, allowing the detection of novel fraud patterns.
5. **Interpretability** – Feature importance from ensemble models helps understand which factors contribute most to fraud.

Limitations

1. **Data Dependency** – Model performance relies heavily on the quality and quantity of historical labeled transactions.
2. **Evolving Fraud Patterns** – New types of fraud may not be detected until retraining with updated data.
3. **Computational Cost** – Training ensemble and deep learning models requires significant computational resources.
4. **Synthetic Oversampling Risks** – Techniques like SMOTE can introduce noise, potentially reducing precision.
5. **Limited Generalization** – Model trained on one dataset (e.g., a specific bank) may require fine-tuning for other institutions.

CONCLUSION AND FUTURE SCOPE

Conclusion

The Fraud Detection in Financial Transactions project successfully demonstrates the application of machine learning and anomaly detection techniques to identify suspicious transactions in an imbalanced dataset.

Key points:

- **Effective Model Selection:** XGBoost with class weighting emerged as the best-performing supervised model, providing a high balance of precision and recall for fraud detection.
- **Integration of Techniques:** Both unsupervised methods (Isolation Forest, AutoEncoder) and supervised methods (Logistic Regression, Random Forest, XGBoost) were explored, highlighting the benefits of combining multiple approaches.
- **Pipeline Deployment:** A final pipeline combining preprocessing (StandardScaler) and the XGBoost model was saved using `joblib` and deployed via a Streamlit web application, allowing real-time prediction for new transactions.
- **Comprehensive Evaluation:** Models were evaluated using accuracy, precision, recall, F1-score, ROC-AUC, and average precision, ensuring robust model selection and performance comparison.

Future Scope

The project can be further enhanced and extended in several ways:

1. **Real-Time Monitoring:** Integrate with live banking or payment systems for instantaneous fraud alerts.
2. **Advanced Deep Learning Models:** Explore LSTM or transformer-based models to capture sequential patterns in transactions.
3. **Multi-Institution Datasets:** Train on aggregated datasets from multiple banks or payment gateways for better generalization.
4. **Adaptive Learning:** Implement continuous learning mechanisms to automatically adapt to new fraud patterns.
5. **Explainable AI:** Incorporate techniques like SHAP or LIME to provide interpretability and trust for stakeholders.
6. **Enhanced Dashboards:** Develop a complete alert dashboard with visualization of transaction trends, risk scores, and fraud statistics.

REFERENCES

1. Kaggle. **Credit Card Fraud Detection Dataset**. Available at: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud
2. Chen, T., & Guestrin, C. (2016). **XGBoost: A Scalable Tree Boosting System.** *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, 785–794.
3. Breunig, M. M., Kriegel, H.-P., Ng, R. T., & Sander, J. (2000). **LOF: Identifying Density-Based Local Outliers.** *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 93–104.
4. Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). **Isolation Forest.** *2008 Eighth IEEE International Conference on Data Mining*, 413–422.
5. Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). **SMOTE: Synthetic Minority Over-sampling Technique.** *Journal of Artificial Intelligence Research*, 16, 321–357.
6. Chollet, F., et al. (2015). **Keras: The Python Deep Learning library.** Available at: https://keras.io
7. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … & Duchesnay, É. (2011). **Scikit-learn: Machine Learning in Python.** *Journal of Machine Learning Research*, 12, 2825–2830.
8. McKinney, W. (2010). **Data Structures for Statistical Computing in Python.** *Proceedings of the 9th Python in Science Conference*, 51–56.
9. Reback, J., McKinney, W., jbrockmendel, et al. (2022). **pandas-dev/pandas: Pandas.** Zenodo. https://doi.org/10.5281/zenodo.3509134
10. Streamlit Inc. (2023). **Streamlit: Turn Data Scripts into Shareable Web Apps.** Available at: https://streamlit.io
11. Géron, A. (2019). **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition).** O'Reilly Media.
12. He, H., & Garcia, E. A. (2009). **Learning from Imbalanced Data.** *IEEE Transactions on Knowledge and Data Engineering*, 21(9), 1263–1284.