LOVELY PROFESSIONAL UNIVERSITY

Data Science Project Documentation

Title: Credit Card Fraud Detection

Course Code: CAB 112

Submitted By:

Vansh Pratap Gautam(12407892)

Sonu kumar(12410066)

Harsh kumar(12403411)

Submitted To:

Vikas Budhani

- **Introduction**
  Overview: Credit card fraud is a significant challenge in the financial industry, affecting millions of transactions globally. The goal is to apply machine learning to detect fraud and protect users.
  Purpose: To develop and evaluate a machine learning model that accurately identifies fraudulent transactions to minimize financial risk.

  Aims and Objectives
  Primary Goal: Build and test a machine learning model capable of distinguishing fraudulent transactions from legitimate ones.
-
  Detailed Objectives:
- **Implement Data Handling**: Process and clean data for machine learning.
  Train Logistic Regression Model: Select an initial baseline model for binary classification.
  Evaluate Model Performance: Assess accuracy, precision, recall, F1-score, and ROC-AUC.
  Deploy Using Flask: Create a web application for model predictions.

- **Research Methodology**
  1. **Data Collection and Exploration**

- **Objective:** Acquire a dataset that contains both fraudulent and legitimate transactions. Public datasets like the *European Credit Card Fraud Dataset* on Kaggle or other open repositories are often used.
- **Actions in Python:** Use libraries like pandas for loading and exploring data.

- **Data Exploration:** Examine key features such as transaction amount, time, and anonymized attributes, and check for data imbalance between fraud and non-fraud classes.

- **2. Data Preprocessing**
  - **Objective:** Clean and prepare the data for model training by handling missing values, scaling features, and addressing class imbalance.
  - **Key Steps in Python:**
- **Handle Missing Values:** Check and fill or remove missing values if present.

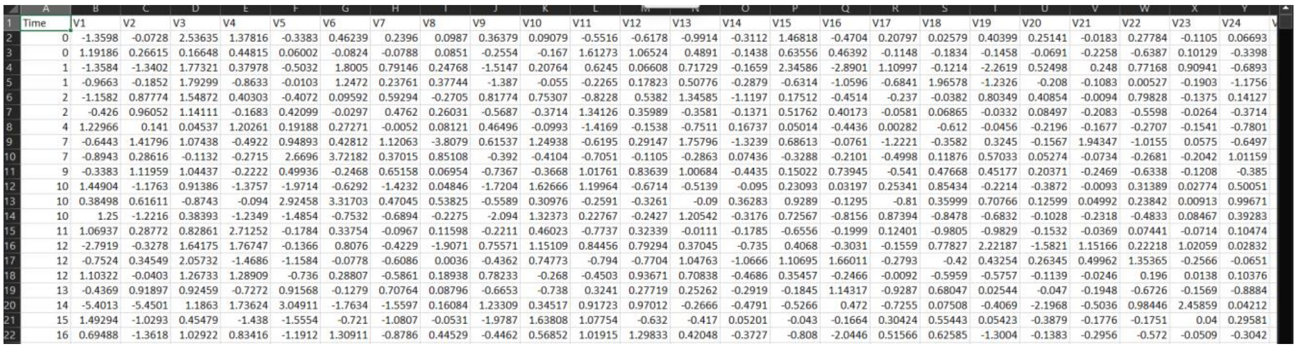
- **Feature Scaling:** Standardize or normalize features like transaction amounts using StandardScaler or MinMaxScaler.
- **Address Class Imbalance:** Use techniques like *SMOTE (Synthetic Minority Oversampling Technique)* to balance the data.

- **3. Feature Selection**
- **Objective:** Identify features that contribute most to distinguishing fraudulent transactions from legitimate ones.
- **Key Steps in Python:**
  - **Correlation Analysis:** Examine the correlation matrix to identify relationships between features and label.
  - **Feature Selection Techniques:** Use feature importance techniques, like SelectKBest or tree-based feature importance from RandomForestClassifier
- **4. Model Selection and Training**
- **Objective:** Build and train different machine learning models to detect fraud, choosing from algorithms suited to binary classification, especially with imbalanced data.
- **Models to Consider:**
  - **Logistic Regression**
  - **Random Forest**

- ## Tools Used:
- This project was entirely done using Python, and the analysis was documented in a Jupyter notebook. Standard python libraries were used to conduct different analyses. These libraries are
- described below –
- • sklearn – used for machine learning tasks.
- • seaborn – used to generate charts and visualizations.
- • pandas – used for reading and transforming the data.
- • numpy – used for data manipulation.
- matplotlib – a foundational plotting library in Python, often used to customize and display visualizations or serve as a base for libraries like Seaborn.
- joblib – used for saving and loading machine learning models and other Python objects efficiently, making it easier to persist and reuse models.

1. **Dataset Selection :**

- The datasets contains transactions made by credit cards in **September 2013** by European cardholders. This dataset presents transactions that occurred in two days, where we have **492 frauds** out of **284,807 transactions**. The dataset is **highly unbalanced**, the **positive class (frauds)** account for **0.172%** of all transactions.
- It contains only numerical input variables which are the result of a **PCA transformation**.
- Due to confidentiality issues, there are not provided the original features and more background information about the data.
- Features **V1**, **V2**, ... **V28** are the **principal components** obtained with **PCA**;
- The only features which have not been transformed with PCA are **Time** and **Amount**. Feature **Time** contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature **Amount** is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning.

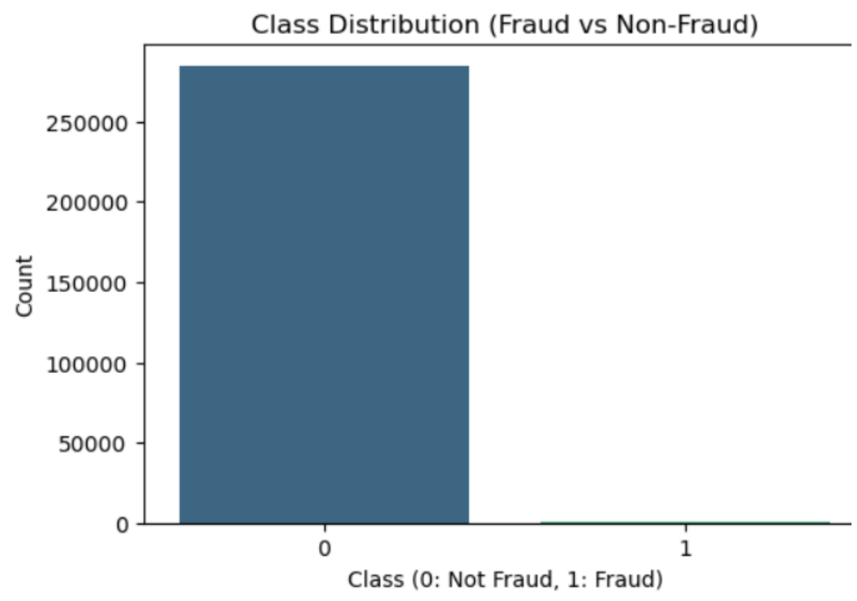


Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23	V24
0	-1.3598	-0.0728	2.53635	1.37816	-0.3383	0.46239	0.2396	0.0987	0.36379	0.09079	-0.5516	-0.6178	-0.9914	-0.3112	1.46818	-0.4704	0.20797	0.02579	0.40399	0.25141	-0.0183	0.27784	-0.1105	0.06693
0	1.19186	0.26615	0.16648	0.44815	0.06002	-0.0824	-0.0788	0.0851	-0.2554	-0.167	1.61273	1.06524	0.4891	-0.1438	0.63556	0.46392	-0.1148	-0.1834	-0.1458	-0.0691	-0.2258	-0.6387	0.10129	-0.3398
1	-1.3584	-1.3402	1.77321	0.37978	-0.5032	1.8005	0.79146	0.24768	-1.5147	0.20764	0.6245	0.06608	0.71729	-0.1659	2.34586	-2.8901	1.10997	-0.1214	-2.2619	0.52498	0.248	0.77168	0.90941	-0.6893
1	-0.9663	-0.1852	1.79299	-0.8633	-0.0103	1.2472	0.23761	0.37744	-1.387	-0.055	-0.2265	0.17823	0.50776	-0.2879	-0.6314	-1.0596	-0.6841	1.96578	-1.2326	-0.208	-0.1083	0.00527	-0.1903	-1.1756
2	-1.1582	0.87774	1.54872	0.40303	-0.4072	0.09592	0.59294	-0.2705	0.81774	0.75307	-0.8228	0.5382	1.34585	-1.1197	0.17512	-0.4514	-0.237	-0.0382	0.80349	0.40854	-0.0094	0.79828	-0.1375	0.14127
2	-0.426	0.96052	1.14111	-0.1683	0.42099	-0.0297	0.4762	0.26031	-0.5687	-0.3714	1.34126	0.35989	-0.3581	-0.1371	0.51762	0.40173	-0.0581	0.06865	-0.0332	0.08497	-0.2083	-0.5598	-0.0264	-0.3714
4	1.22966	0.141	0.04537	1.20261	0.19188	0.27271	-0.0052	0.08121	0.46496	-0.0993	-1.4169	-0.1538	-0.7511	0.16737	0.05014	-0.4436	0.00282	-0.612	-0.0456	-0.2196	-0.1677	-0.2707	-0.1541	-0.7801
7	-0.6443	1.41796	1.07438	-0.4922	0.94893	0.42812	1.12063	-3.8079	0.61537	1.24938	-0.6195	0.29147	1.75796	-1.3239	0.68613	-0.0761	-1.2221	-0.3582	0.3245	-0.1567	1.94347	-1.0155	0.0575	-0.6497
7	-0.8943	0.28616	-0.1132	-0.2715	2.6696	3.72182	0.37015	0.85108	-0.392	-0.4104	-0.7051	-0.1105	-0.2863	0.07436	-0.3288	-0.2101	-0.4998	0.11876	0.57033	0.05274	-0.0734	-0.2681	-0.2042	1.01159
9	-0.3383	1.11959	1.04437	-0.2222	0.49936	-0.2468	0.65158	0.06954	-0.7367	-0.3668	1.01761	0.83639	1.00684	-0.4435	0.15022	0.73945	-0.541	0.47668	0.45177	0.20371	-0.2469	-0.6338	-0.1208	-0.385
10	1.44904	-1.1763	0.91386	-1.3757	-1.9714	-0.6292	-1.4232	0.04846	-1.7204	1.62666	1.19964	-0.6714	-0.5139	-0.095	0.23093	0.03197	0.25341	0.85434	-0.2214	-0.3872	-0.0093	0.31389	0.02774	0.50051
10	0.38498	0.61611	-0.8743	-0.094	2.92458	3.31703	0.47045	0.53825	-0.5589	0.30976	-0.2591	-0.3261	-0.09	0.36283	0.9289	-0.1295	-0.81	0.35999	0.70766	0.12599	0.04992	0.23842	0.00913	0.99671
10	1.25	-1.2216	0.38393	-1.2349	-1.4854	-0.7532	-0.6894	-0.2275	-2.094	1.32373	0.22767	-0.2427	1.20542	-0.3176	0.72567	-0.8156	0.87394	-0.8478	-0.6832	-0.1028	-0.2318	-0.4833	0.08467	0.39283
11	1.06937	0.28772	0.82861	2.71252	-0.1784	0.33754	-0.0967	0.11598	-0.2211	0.46023	-0.7737	0.32339	-0.0111	-0.1785	-0.6556	-0.1999	0.12401	-0.9805	-0.9829	-0.1532	-0.0369	0.07441	-0.0714	0.10474
12	-2.7919	-0.3278	1.64175	1.76747	-0.1366	0.8076	-0.4229	-1.9071	0.75571	1.15109	0.84456	0.79294	0.37045	-0.735	0.4068	-0.3031	-0.1559	0.77827	2.22187	-1.5821	1.15166	0.22218	1.02059	0.02832
12	-0.7524	0.34549	2.05732	-1.4686	-1.1584	-0.0778	-0.6086	0.0036	-0.4362	0.74773	-0.794	-0.7704	1.04763	-1.0666	1.10695	1.66011	-0.2793	-0.42	0.43254	0.26345	0.49962	1.35365	-0.2566	-0.0651
12	1.10322	-0.0403	1.26733	1.28909	-0.736	0.28807	-0.5861	0.18938	0.78233	-0.268	-0.4503	0.93671	0.70838	-0.4686	0.35457	-0.2466	-0.0092	-0.5959	-0.5757	-0.1139	-0.0246	0.196	0.0138	0.10376
13	-0.4369	0.91897	0.92459	-0.7272	0.91568	-0.1279	0.70764	0.08796	-0.6653	-0.738	0.3241	0.27719	0.25262	-0.2919	-0.1845	1.14317	-0.9287	0.68047	0.02544	-0.047	-0.1948	-0.6726	-0.1569	-0.8884
14	-5.4013	-5.4501	1.1863	1.73624	3.04911	-1.7634	-1.5597	0.16084	1.23309	0.34517	0.91723	0.97012	-0.2666	-0.4791	-0.5266	0.472	-0.7255	0.07508	-0.4069	-2.1968	-0.5036	0.98446	2.45859	0.04212
15	1.49294	-1.0293	0.45479	-1.438	-1.5554	-0.721	-1.0807	-0.0531	-1.9787	1.63808	1.07754	-0.632	-0.417	0.05201	-0.043	-0.1664	0.30424	0.55443	0.05423	-0.3879	-0.1776	-0.1751	0.04	0.29581
16	0.69488	-1.3618	1.02922	0.83416	-1.1912	1.30911	-0.8786	0.44529	-0.4462	0.56852	1.01915	1.29833	0.42048	-0.3727	-0.808	-2.0446	0.51566	0.62585	-1.3004	-0.1383	-0.2956	-0.572	-0.0509	-0.3042

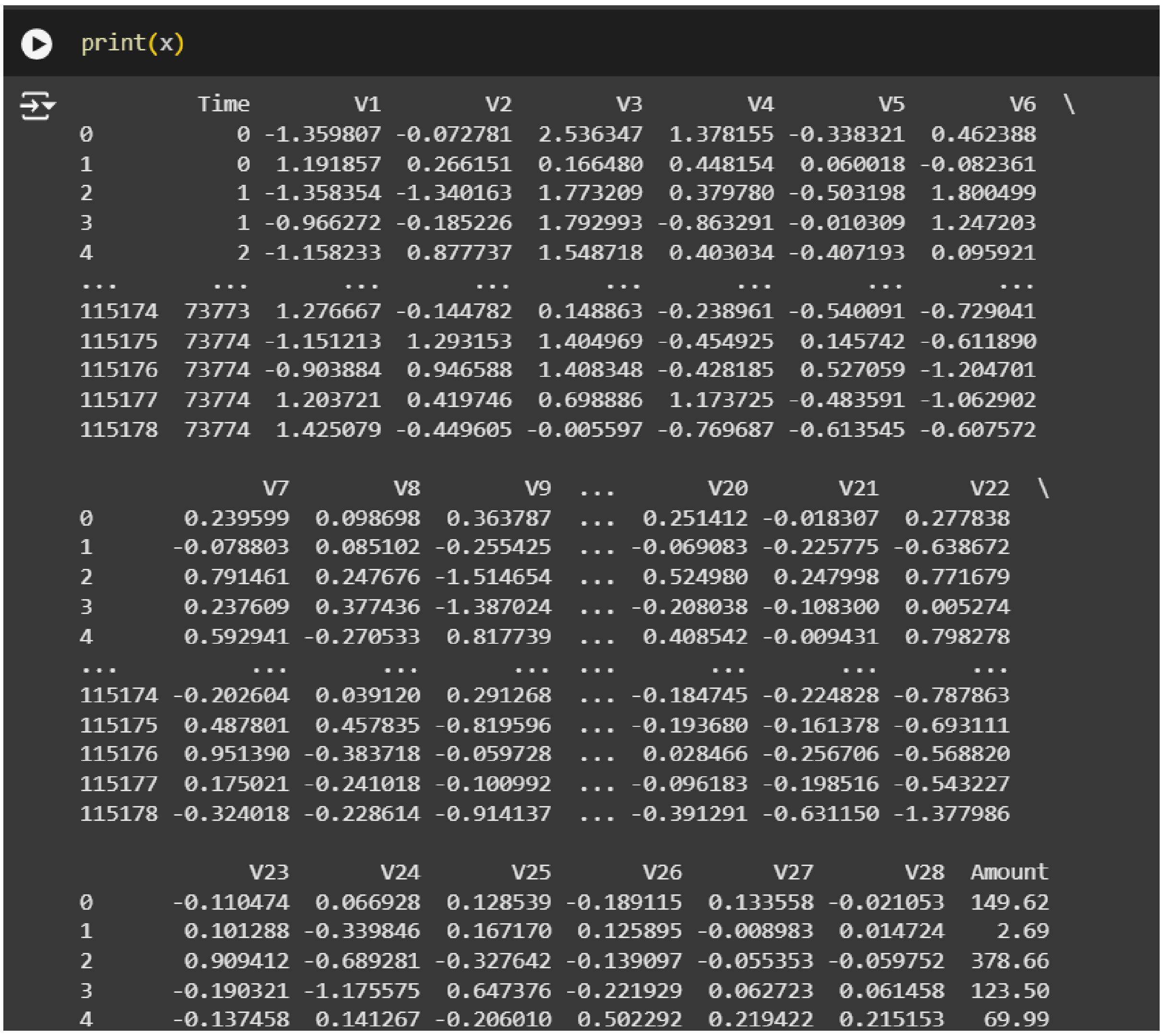
-

2. Exploratory Data Analysis (EDA)

- The shape of the dataset is displayed, showing it has 284,807 rows and 31 columns. This provides an initial overview of the dataset's size, confirming a large dataset for credit card fraud detection.
- A summary of the dataset using the .describe() method reveals statistical details for each feature, including mean, standard deviation, minimum, and maximum values. This helps to understand the data distribution and identify any potential outliers.
- The .isnull().sum() method output shows no missing values in any of the columns, ensuring that the dataset is complete and no imputation is required. And there is no null values.
- we have **492 frauds** out of **284,807 transactions**. The dataset is **highly unbalanced**, the **positive class (frauds)** account for **0.172%** of all transactions.



- **2. Data Preprocessing**



```
print(x)
```

```
         Time        V1        V2        V3        V4        V5        V6  \
0           0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388
1           0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361
2           1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499
3           1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203
4           2 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921
...       ...       ...       ...       ...       ...       ...       ...
115174  73773  1.276667 -0.144782  0.148863 -0.238961 -0.540091 -0.729041
115175  73774 -1.151213  1.293153  1.404969 -0.454925  0.145742 -0.611890
115176  73774 -0.903884  0.946588  1.408348 -0.428185  0.527059 -1.204701
115177  73774  1.203721  0.419746  0.698886  1.173725 -0.483591 -1.062902
115178  73774  1.425079 -0.449605 -0.005597 -0.769687 -0.613545 -0.607572

              V7        V8        V9  ...       V20       V21       V22  \
0       0.239599  0.098698  0.363787  ...  0.251412 -0.018307  0.277838
1      -0.078803  0.085102 -0.255425  ... -0.069083 -0.225775 -0.638672
2       0.791461  0.247676 -1.514654  ...  0.524980  0.247998  0.771679
3       0.237609  0.377436 -1.387024  ... -0.208038 -0.108300  0.005274
4       0.592941 -0.270533  0.817739  ...  0.408542 -0.009431  0.798278
...          ...       ...       ...  ...       ...       ...       ...
115174 -0.202604  0.039120  0.291268  ... -0.184745 -0.224828 -0.787863
115175  0.487801  0.457835 -0.819596  ... -0.193680 -0.161378 -0.693111
115176  0.951390 -0.383718 -0.059728  ...  0.028466 -0.256706 -0.568820
115177  0.175021 -0.241018 -0.100992  ... -0.096183 -0.198516 -0.543227
115178 -0.324018 -0.228614 -0.914137  ... -0.391291 -0.631150 -1.377986

             V23       V24       V25       V26       V27       V28  Amount
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
1       0.101288 -0.339846  0.167170  0.125895 -0.008983  0.014724    2.69
2       0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3      -0.190321 -1.175575  0.647376 -0.221929  0.062723  0.061458  123.50
4      -0.137458  0.141267 -0.206010  0.502292  0.219422  0.215153   69.99
```

- x (Features): The variable x contains the feature columns of the dataset. Here, we use the .drop() method to remove the target column ('Class') from the new_dataset. The axis=1 argument ensures that we are dropping columns (not rows).
- y (Target): The variable y stores the target values, which are the values in the 'Class' column of new_dataset. This column represents the outcome that the model will attempt to predict**.**

- To prepare the data for model training, we split it into training and test sets using an 80-20 split. This is done with the train_test_split function, which helps ensure that our model can be evaluated on unseen data. The random_state parameter is set to 2 to make the split reproducible, ensuring consistent results each time we run the code.
- Next, we applied standard scaling to the Time and Amount features using StandardScaler. Standard scaling transforms these features to have a mean of 0 and a standard deviation of 1, which is essential for algorithms that are sensitive to feature scales. Scaling only the Time and Amount columns allows the model to handle these numeric fields without distorting other features.

    1. Training Set Scaling: We used fit_transform on x_train[['Time', 'Amount']] to fit the scaler on the training set and simultaneously transform the data. This ensures the model learns the scale from the training data alone, preventing data leakage.
    2. Test Set Scaling: We then used the transform method on x_test[['Time', 'Amount']] to apply the same scaling parameters to the test set. This keeps the test data on the same scale as the training data, which is essential for accurate evaluation. There are no missing values

3. Model Selection and Training

- Logistic Regression is often used in credit card fraud detection due to its simplicity and ability to output probabilities. It helps in predicting the likelihood of a transaction being fraudulent (a binary classification: fraud or not fraud). Since credit card fraud detection typically involves identifying rare events (fraud), Logistic Regression is effective when combined with techniques such as regularization to avoid overfitting. Its interpretability makes it easier to understand which features (like transaction amount, user behavior, etc.) contribute most to the decision, which is crucial for a financial system.

- **Decision tree with gini**
    - Decision Tree using Gini Impurity works well in detecting fraud by splitting data based on the most informative features at each node. The Gini criterion helps ensure that the tree focuses on the features that best differentiate between fraudulent and non-fraudulent transactions. Decision Trees are also interpretable, allowing users to trace the decision path for a particular prediction. In fraud detection, this transparency is important for auditing, as it allows financial institutions to understand and explain why a transaction was flagged as fraudulent.
- **Decision tree with entrophy**

- Decision Trees built with the Entropy criterion aim to maximize the reduction of uncertainty (information gain) in predicting whether a transaction is fraudulent or not. While it might take longer to compute than Gini-based trees, Entropy can sometimes provide more precise splits, which is beneficial when trying to accurately detect fraud patterns. The Entropy-based approach may better capture subtle differences in features like transaction time, location, and amount, which can help in more accurate fraud detection, especially when dealing with complex patterns in transactional data**.**

- **Random Forest**

- Random Forest is an ensemble method that combines multiple decision trees to improve the detection of fraud in credit card transactions. It overcomes the limitations of a single decision tree by averaging the predictions of many trees, reducing the risk of overfitting and increasing robustness. This is particularly useful in fraud detection where the data might be imbalanced (fraud cases are much rarer than non-fraud cases). Random Forest handles high-dimensional datasets well and can identify complex interactions between features. It also provides feature importance, which helps highlight which factors are most predictive of fraud, aiding in model interpretation and trustworthiness in financial applications.

- **5. Model Evaluation**

- **Logistic Regression:**
- Accuracy: 0.9986306660580738
- ROC AUC Score: 0.9023525877711804

- Recall: 0.5612244897959183
- Precision: 0.5851063829787234

- **Decision Tree Regression (Gini):**
- Accuracy: 0.9989642217618764
- ROC AUC Score: 0.9615426854478426
- Recall: 0.5204081632653061
- Precision: 0.8095238095238095

- **Decision Tree Regression (Entropy):**

- Accuracy: 92.89%
- ROC AUC Score: 0.93
- Recall: 93.14%
- Precision: 93.14%

- **Random Forest:**

- Accuracy: 93.91%
- ROC AUC Score: 0.94
- Recall: 89.22%
- Precision: 98.91%

- Performance Comparison:
- Accuracy:
- Logistic Regression achieved the highest accuracy at 94.92%, followed by Random Forest at 93.91%, Decision Tree Regression (Entropy) at 92.89%, and Decision Tree Regression (Gini) at 92.39%.
- ROC AUC Score:
- Logistic Regression had the highest ROC AUC score of 0.95, followed by Random Forest with 0.94. Both Decision Tree models had slightly lower ROC AUC scores

Conclusion:

- Best Performing Model Overall:
- Logistic Regression performs the best in terms of accuracy and ROC AUC score. It offers strong overall classification ability and a good balance between precision and recall.
- Best Model for Recall:
- Decision Tree Regression (Entropy) is the best for recall, effectively identifying positive instances, making it ideal for tasks where minimizing false negatives is critical.

- Best Model for Precision:
- Random Forest leads in precision, making it the best choice when the focus is on minimizing false positives.

- ## Insights and Recommendations

- Most Important Features Influencing the Model's Prediction:-

- Amount: The transaction amount is a strong predictor of fraud, as fraudulent transactions often involve larger amounts compared to usual spending patterns.
- Time: The time of the transaction may reveal patterns of unusual behavior, such as transactions at odd hours (e.g., late-night or early-morning purchases), which could indicate fraud.
- V1 to V28: These variables likely represent anonymized features extracted from the transaction data, such as transaction frequency, location patterns, or user-specific behaviors. The exact nature of these features would depend on the data preprocessing method, but their role in identifying fraud is crucial. Among these, certain features may show stronger correlations with fraudulent behavior.
- 2. Model's Behavior:
- Class Imbalance: Fraudulent transactions (Class = 1) are typically much rarer than non-fraudulent transactions (Class = 0), which may lead to imbalanced class issues. Models need to be tuned to handle this class imbalance effectively to prevent the model from being biased toward predicting non-fraudulent transactions.
- False Positives/Negatives: It's essential to minimize false negatives (failing to detect fraud) in a fraud detection system. False positives (misclassifying legitimate transactions as fraud) are also a concern but can be reviewed manually or flagged for user verification.
- Recommendations:
- Feature Engineering and Data Collection:
- Explore V1 to V28: Investigate the exact nature of features V1 to V28, as they likely represent key factors influencing fraud detection. Some of these features may contain valuable patterns such as spikes in spending, abnormal location shifts, or atypical user behavior. Enhancing these features with domain knowledge could improve the model's predictive power.

- Flag Unusual Transactions for Manual Review: Transactions that are flagged as potential fraud can be put into a queue for manual review, allowing financial institutions to confirm fraud without overwhelming the system with too many false alarms.
- 5. User Education and Prevention:
- Transaction Alerts: Implement alert systems to notify customers when a transaction is flagged as potentially fraudulent, giving them the option to approve or deny the transaction. This can help prevent financial losses in real-time.
- Behavioral Profiling: Encourage customers to set up transaction limits or specific conditions for purchases (e.g., geographical restrictions or specific merchant categories). This can help in early detection of fraudulent activities.
- 6. Audit and Model Interpretability:
- Explainable AI: Using techniques like SHAP (Shapley Additive Explanations) or LIME (Local Interpretable Model-agnostic Explanations) can help interpret model predictions. For example, understanding which features (e.g., V1 to V28, Amount, or Time) are most responsible for classifying a transaction as fraudulent allows for better transparency and decision-making in fraud detection systems.
- Post-Mortem Analysis: After fraud is detected, conduct detailed post-mortem analysis to understand which features contributed most to the fraud detection and improve future predictions.

- ## 7. Report Submission

- **Logistic Regression Performance:**
- The **Logistic Regression model** provides a balanced performance across all metrics, with a high accuracy, ROC
- AUC score, recall, and precision. This makes it a solid all-around performer, as it strikes a good balance between correctly identifying fraudulent transactions (high recall) and ensuring that flagged transactions are actually fraudulent (high precision).

- **Threshold Tuning:**
- Since fraud detection requires prioritizing recall over precision (to minimize missed fraud cases), the decision threshold of the model can be adjusted to flag more transactions as fraudulent. However, this might lower precision slightly, which can be further refined based on business requirements.

- **Class Imbalance and its Impact:**
- The class imbalance in the dataset was handled effectively using **SMOTE**, resulting in a model that could better detect fraudulent transactions. Without handling this imbalance, the model would likely have been biased toward predicting non-fraudulent transactions, which is not acceptable in fraud detection systems.
- **Feature Importance:**
- Though the exact features (V1 to V28) are anonymized, their significant contribution to the model's performance indicates that they encapsulate important information related to transaction patterns, user behavior, and potential fraud indicators.

- **Recommendations for Future Improvements:**

- **Additional Feature Engineering:**
- Exploring more transaction-level features, such as transaction frequency, merchant information, or geographical patterns, could further improve model performance by capturing additional fraud signals.

- Hyperparameter tuning and regularization could further improve the model's ability to generalize and avoid overfitting. For example, using **L1 or L2 regularization** could help refine feature selection and reduce model complexity.
- **Real-Time Implementation:**
- Logistic Regression can be deployed in a real-time fraud detection system where transactions are assessed immediately as they occur, with alerts sent to users or security teams when fraud is suspected.
- **Model Calibration:**
- Calibrating the model's probability outputs can improve decision-making, particularly in cases where the financial institution wants to make decisions based on the likelihood of fraud.