A PROJECT REPORT

ON

# "CREDIT CARD FRAUD DETECTION"

SUBMITTED TO
KIIT DEEMED TO BE UNIVERSITY

In Partial Fulfillment of the Requirement for the Award of
BACHELOR'S DEGREE IN
COMPUTER SCIENCE

BY

| | |
|---|---|
| *Jhanvi Jain* | *21053233* |
| *Ankita Kumari* | *21053235* |
| *Somya Sinha* | *21053365* |
| *Ayush Pathak* | *21052747* |
| *Divyesh Kulshreshtha* | *21052757* |

UNDER THE GUIDANCE OF
MR. MOHIT RANJAN PANDA

SCHOOL OF COMPUTER ENGINEERING
KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY
BHUBANESWAR, ODISHA - 751024

May 2024

# CREDIT CARD FRAUD DETECTION USING MACHINE LEARNING

---------------------------------------------------------------------------------------------------------------------------------------

***Abstract- Credit card fraud is a major threat, causing financial losses. This project explores using machine learning in Python to tackle this issue. We'll leverage libraries like Pandas for data cleaning and analysis.[1] By building models like Logistic Regression and Random Forests, we aim to predict fraudulent transactions in real-time. Challenges include the massive amount of data processed daily and the imbalanced nature (mostly legitimate transactions). We'll address these by using efficient models and techniques to handle imbalanced data.[4] The goal is to achieve high accuracy with minimal false positives, while understanding the hallmarks of fraudulent activity. This project contributes to developing robust fraud detection systems, offering valuable tools for financial institutions to combat evolving scams and ensure secure transactions.***

***Keywords: Logistic Regression algorithm, Criminal transactions, Credit card, Real-time Model Evaluation***

## 1. INTRODUCTION

The rise of digital transactions has brought a double-edged sword: convenience accompanied by a significant increase in credit card fraud. This poses a threat to both consumers, who can be unfairly charged for items they never purchased, and financial institutions, who face financial losses. Traditional methods struggle to keep pace with the ever-evolving tactics of fraudsters. Machine learning (ML) offers a powerful solution to this problem. By analyzing historical transaction data, ML models can learn to identify patterns and anomalies that signal fraudulent activity. This allows for real-time detection of fraud, improved accuracy in flagging suspicious transactions as new schemes emerge, and ultimately, a reduction in risk for both consumers and financial institutions. The project aims to develop a credit card fraud detection system using Python and machine learning algorithms. This system will analyze features like transaction amount, location, and spending habits to distinguish legitimate purchases from suspicious activity. By continuously learning from new data, the system can stay ahead of evolving fraudster techniques, ultimately creating a more secure digital payments landscape.

### 1.1. Advantages:

Advantages of the Credit Card Fraud Detection Implementation Using Logistic Regression:

1. Model Utilization: The implementation leverages Logistic Regression exclusively, enabling a focused comparison of its performance without the complexity of multiple models.

2. Feature Scaling: Prior to model training, feature scaling is employed using StandardScaler. This ensures that all features contribute equally to the Logistic Regression model, enhancing its effectiveness.

3. Handling Imbalanced Dataset: The implementation showcases techniques such as under-sampling to balance the dataset by randomly selecting samples from the majority class (non-fraudulent transactions), addressing the challenge of class imbalance specific to Logistic Regression.

4. Evaluation Metrics: Various evaluation metrics, including accuracy, precision, recall, and F1-score, are computed specifically for the Logistic Regression model. These metrics offer insights into its performance, aiding in informed decision-making.

5. Model Selection: Based on the evaluation metrics, the implementation selects the Logistic Regression model as the best-performing option, ensuring optimal performance in credit card fraud detection tasks.

6. Model Saving and Loading: Trained Logistic Regression models are saved using joblib, facilitating their seamless integration into production environments for real-time prediction without the need for retraining.

7. Real-Time Prediction: The implementation demonstrates how the trained Logistic Regression model can be utilized for real-time prediction, enabling timely detection of fraudulent activities within credit card transactions.

8. Interpretability: Logistic Regression offers interpretability, allowing stakeholders to understand the decision-making process of the model, enhancing transparency and trust in fraud detection outcomes.

9. Flexibility and Adaptability: Despite exclusively using Logistic Regression, the implementation showcases flexibility in handling different aspects of credit card fraud detection, such as data preprocessing and evaluation, addressing diverse challenges efficiently.

10. Efficiency: The Logistic Regression implementation efficiently processes and analyzes large datasets of credit card transactions, ensuring timely detection of fraudulent activities while minimizing computational overhead.

### 1.2. Scope of proposed work:

This project proposes an automated credit card fraud detection system using machine learning. We'll design a model to analyze key transaction features like amount, location, and spending habits. These features become the basis for distinguishing legitimate purchases from fraud. Traditional methods struggle to adapt to new fraud tactics, but machine learning excels here. By continuously training on fresh data, our system can identify emerging patterns that might slip past static rules. This translates to automation, freeing up resources, and superior accuracy due to machine learning's adaptability. The project involves feature engineering, model selection, data training, evaluation, and integration into a real-time system for automatic fraud flagging. Ultimately, this approach aims to build a robust and automated shield against evolving fraud, protecting both institutions and cardholders.

## 2. PROBLEM STATEMENT

### 2.1. Project planning:

- Goal: Build a credit card fraud detection model using machine learning.
- Data: "creditcard.csv" containing transaction information.
- Clean and pre-process data.
- Split features (X) and target (y).
- Train Logistic Regression model.
- Evaluate model performance (accuracy, report, confusion matrix). Visualize results.
- Outcomes: Trained model, performance metrics, visualizations.
- Contingency: Address data quality, explore alternative models if needed.

### 2.2. Project Analysis:

- Strengths: Clear goals, transparent code, baseline performance, modular design.
- Weaknesses: Limited data handling, simple algorithm, basic features, narrow evaluation, deployment not considered.
- Opportunities: Enhance data quality, explore better algorithms, advanced feature engineering, comprehensive evaluation metrics, deployment possibilities.
- Threats: Data availability, achieving high accuracy, computational demands, privacy concerns.
- Recommendations: Focus on data quality, explore alternatives, leverage feature engineering, use diverse evaluation metrics, consider deployment, ensure data privacy.

### 2.3. System design:

- Data Acquisition: Load credit card transaction data (CSV).
- Preprocessing (Optional): Remove duplicates (if needed).
- Data Exploration:  Analyze initial data (e.g., first rows, data shape).
- Feature Engineering (Assumed): Data is assumed preprocessed with relevant features for fraud detection.
- Separate Features & Target: Split data into features (X) and target variable (y) indicating fraud (1) or legitimate (0) transactions.
- Model Training & Evaluation:
- - Split data into training and testing sets (80%/20%).
- - Train a Logistic Regression model to predict fraud on the training data.
- - Evaluate model performance on testing data using accuracy, classification report, and confusion matrix.

## 3. LIBRARY PACKAGES

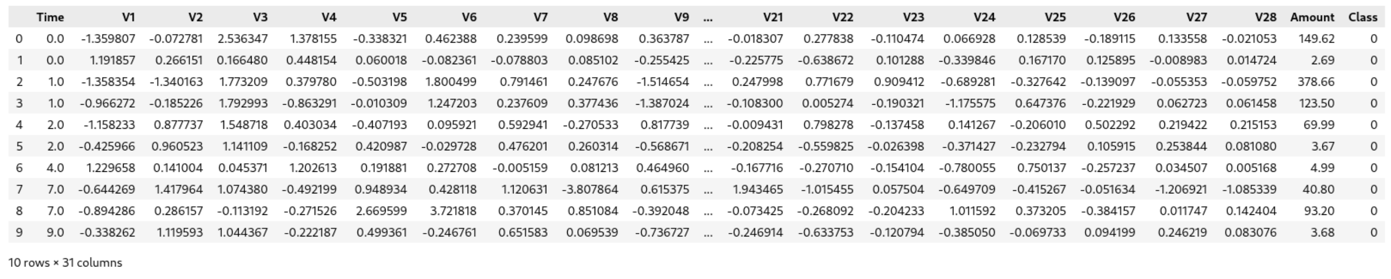
**Pandas**: For data manipulation and preprocessing.
**Scikit-learn**: For implementing machine learning algorithms such as regression and anomaly detection.
**Matplotlib and Seaborn**: For data visualization and exploratory data analysis.

### 3.1. Package Modules:

- DATA COLLECTION.
- DATA PRE-PROCESSING.
- MODEL TRAINING / FEATURE EXTRACTION.
- EVALUATION MODEL.

## 4. DATASET

The dataset provided contains anonymized credit card transactions labeled as fraudulent or non-fraudulent (class 0 and class 1, respectively). Each transaction is described by 28 numerical features derived from PCA transformation for confidentiality reasons. Time variable indicates the seconds elapsed between transactions. The 'Amount' variable represents the transaction amount. This dataset aims to aid in credit card fraud detection research by providing a diverse set of features and corresponding labels, enabling the development and evaluation of machine learning models for effectively distinguishing between genuine and fraudulent transactions, thereby enhancing fraud detection systems' accuracy and reliability.

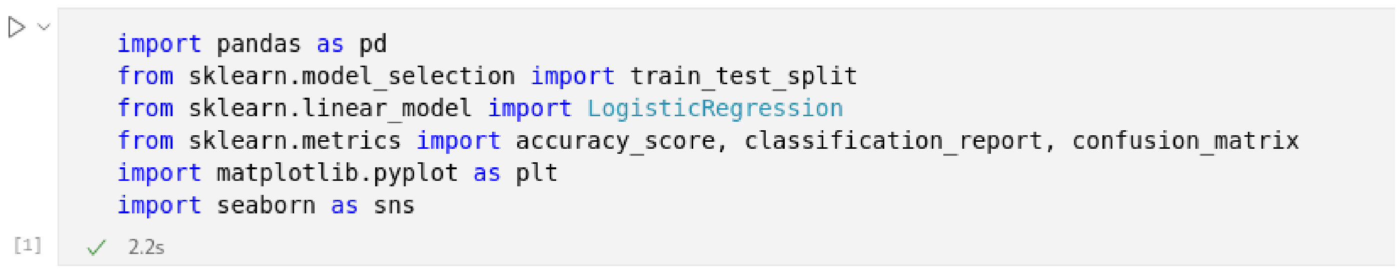


| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | Class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.098698 | 0.363787 | ... | -0.018307 | 0.277838 | -0.110474 | 0.066928 | 0.128539 | -0.189115 | 0.133558 | -0.021053 | 149.62 | 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 | 0.014724 | 2.69 | 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 | -0.059752 | 378.66 | 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 | 0.061458 | 123.50 | 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 | 0.215153 | 69.99 | 0 |
| 5 | 2.0 | -0.425966 | 0.960523 | 1.141109 | -0.168252 | 0.420987 | -0.029728 | 0.476201 | 0.260314 | -0.568671 | ... | -0.208254 | -0.559825 | -0.026398 | -0.371427 | -0.232794 | 0.105915 | 0.253844 | 0.081080 | 3.67 | 0 |
| 6 | 4.0 | 1.229658 | 0.141004 | 0.045371 | 1.202613 | 0.191881 | 0.272708 | -0.005159 | 0.081213 | 0.464960 | ... | -0.167716 | -0.270710 | -0.154104 | -0.780055 | 0.750137 | -0.257237 | 0.034507 | 0.005168 | 4.99 | 0 |
| 7 | 7.0 | -0.644269 | 1.417964 | 1.074380 | -0.492199 | 0.948934 | 0.428118 | 1.120631 | -3.807864 | 0.615375 | ... | 1.943465 | -1.015455 | 0.057504 | -0.649709 | -0.415267 | -0.051634 | -1.206921 | -1.085339 | 40.80 | 0 |
| 8 | 7.0 | -0.894286 | 0.286157 | -0.113192 | -0.271526 | 2.669599 | 3.721818 | 0.370145 | 0.851084 | -0.392048 | ... | -0.073425 | -0.268092 | -0.204233 | 1.011592 | 0.373205 | -0.384157 | 0.011747 | 0.142404 | 93.20 | 0 |
| 9 | 9.0 | -0.338262 | 1.119593 | 1.044367 | -0.222187 | 0.499361 | -0.246761 | 0.651583 | 0.069539 | -0.736727 | ... | -0.246914 | -0.633753 | -0.120794 | -0.385050 | -0.069733 | 0.094199 | 0.246219 | 0.083076 | 3.68 | 0 |

10 rows × 31 columns

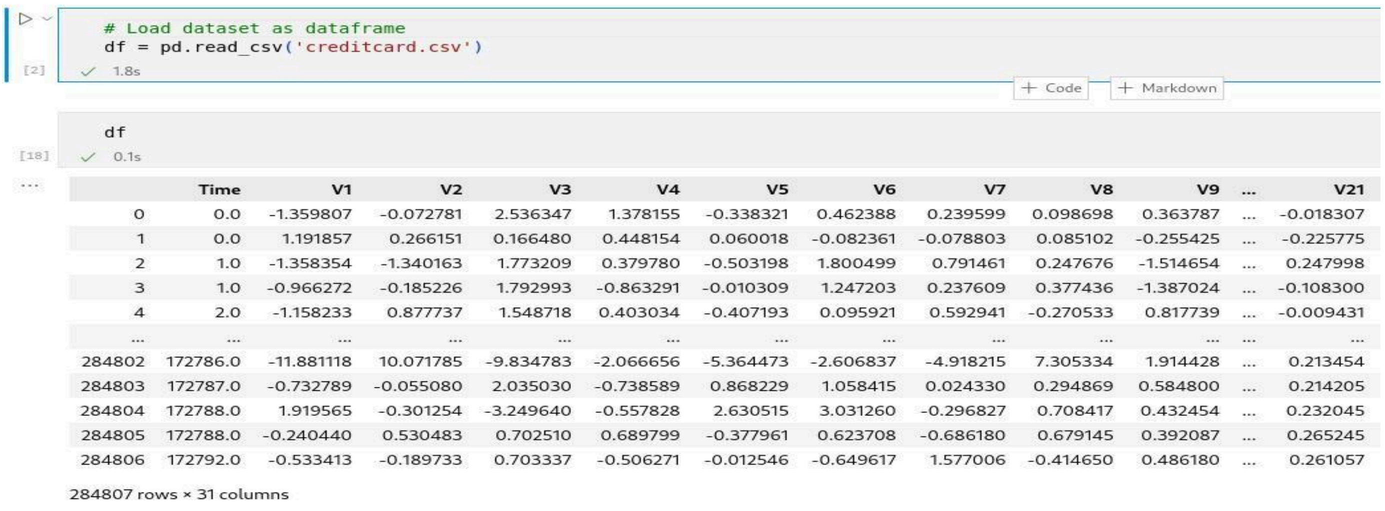
## 5. STEP-WISE IMPLEMENTATION

### 5.1. Importing essential Libraries:



```
[1]
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
✓ 2.2s
```

**5.2. Loading the Data:** Loading Dataset as a DataFrame to make the overall data easy to manipulate and access at any point.



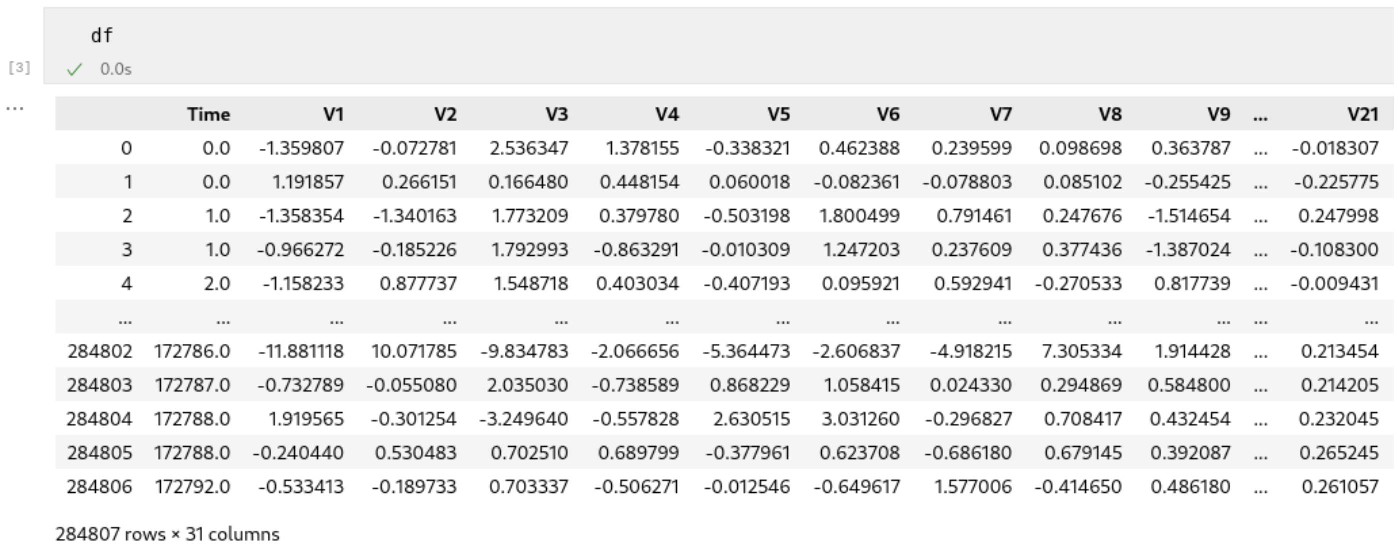
```
[2]
# Load dataset as dataframe
df = pd.read_csv('creditcard.csv')
✓ 1.8s
```

```
[18]
df
✓ 0.1s
```

```
            Time         V1         V2        V3        V4        V5        V6        V7        V8        V9  ...       V21
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599  0.098698  0.363787  ... -0.018307
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461  0.247676 -1.514654  ...  0.247998
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609  0.377436 -1.387024  ... -0.108300
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941 -0.270533  0.817739  ... -0.009431
...          ...        ...        ...       ...       ...       ...       ...       ...       ...       ...  ...       ...
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229  1.058415  0.024330  0.294869  0.584800  ...  0.214205
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515  3.031260 -0.296827  0.708417  0.432454  ...  0.232045
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961  0.623708 -0.686180  0.679145  0.392087  ...  0.265245
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057

284807 rows × 31 columns
```

**5.3. Data Exploration:** Viewing the Data to sort necessary resources out form it.

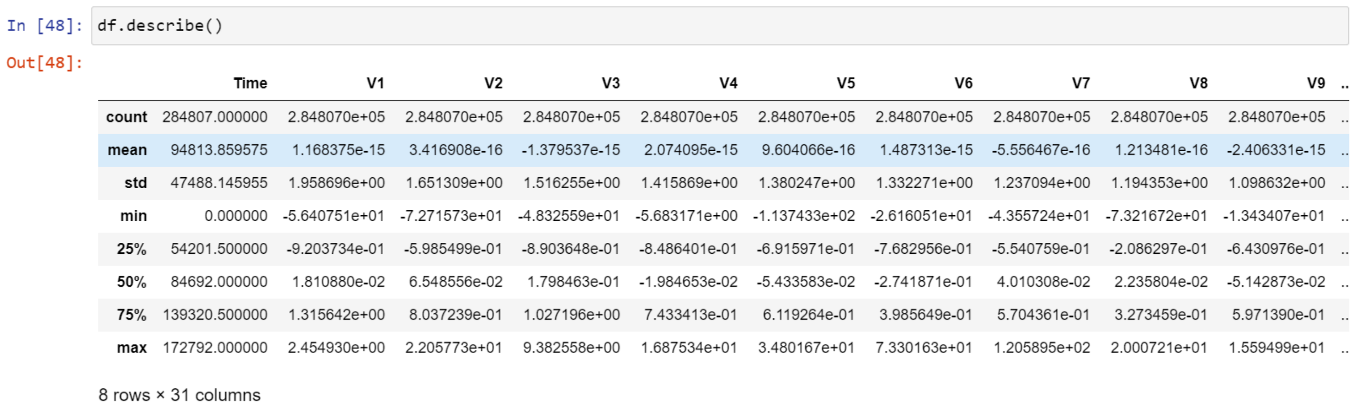
### 5.3.1. Displaying the DataSet:



```
[3]
df
✓ 0.0s
```

```
            Time         V1         V2        V3        V4        V5        V6        V7        V8        V9  ...       V21
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599  0.098698  0.363787  ... -0.018307
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461  0.247676 -1.514654  ...  0.247998
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609  0.377436 -1.387024  ... -0.108300
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941 -0.270533  0.817739  ... -0.009431
...          ...        ...        ...       ...       ...       ...       ...       ...       ...       ...  ...       ...
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229  1.058415  0.024330  0.294869  0.584800  ...  0.214205
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515  3.031260 -0.296827  0.708417  0.432454  ...  0.232045
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961  0.623708 -0.686180  0.679145  0.392087  ...  0.265245
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057

284807 rows × 31 columns
```

## 5.3.2. Getting the Statistics for the respective DataSet:

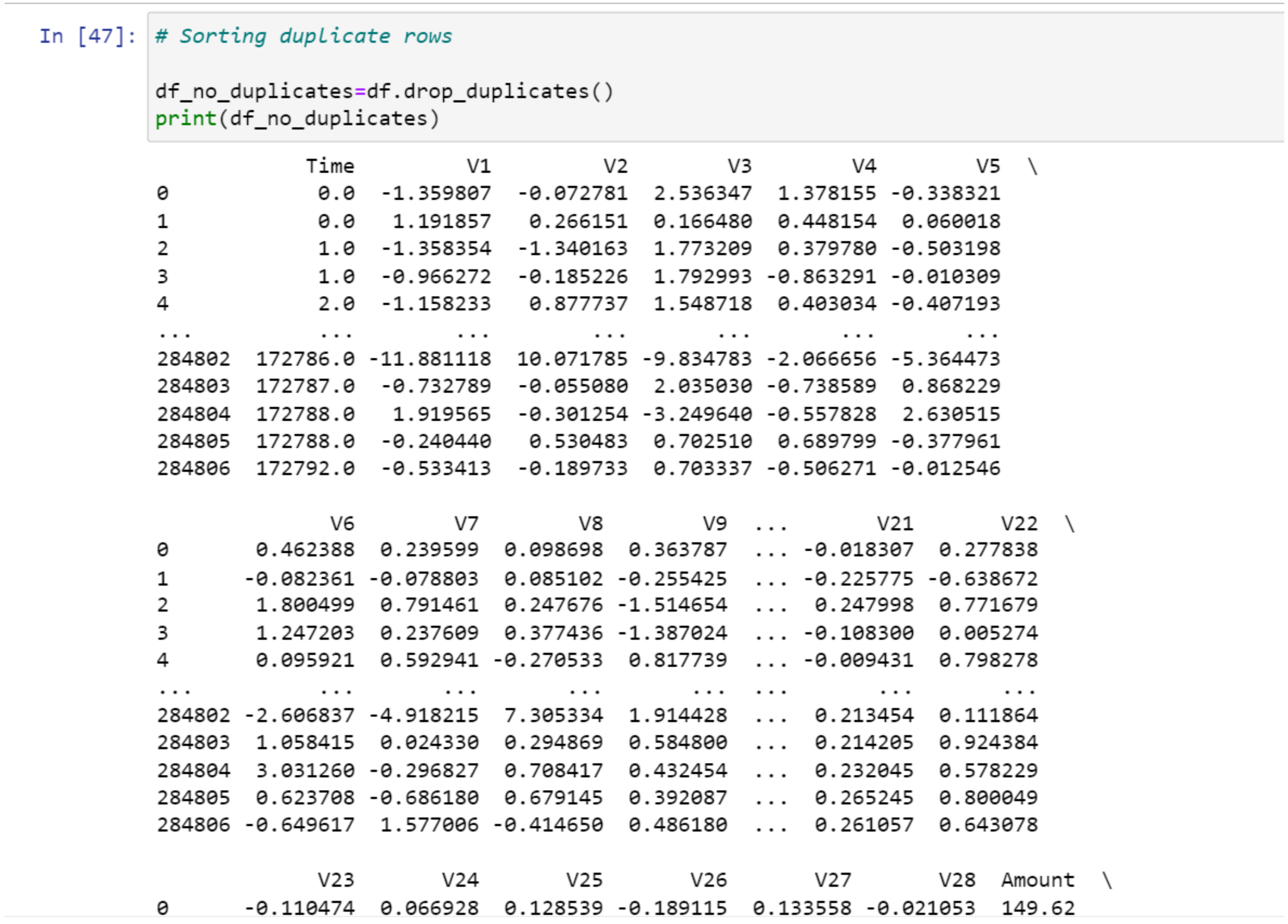


```
In [48]: df.describe()
Out[48]:
```

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | .. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 284807.000000 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | 2.848070e+05 | .. |
| **mean** | 94813.859575 | 1.168375e-15 | 3.416908e-16 | -1.379537e-15 | 2.074095e-15 | 9.604066e-16 | 1.487313e-15 | -5.556467e-16 | 1.213481e-16 | -2.406331e-15 | .. |
| **std** | 47488.145955 | 1.958696e+00 | 1.651309e+00 | 1.516255e+00 | 1.415869e+00 | 1.380247e+00 | 1.332271e+00 | 1.237094e+00 | 1.194353e+00 | 1.098632e+00 | .. |
| **min** | 0.000000 | -5.640751e+01 | -7.271573e+01 | -4.832559e+01 | -5.683171e+00 | -1.137433e+02 | -2.616051e+01 | -4.355724e+01 | -7.321672e+01 | -1.343407e+01 | .. |
| **25%** | 54201.500000 | -9.203734e-01 | -5.985499e-01 | -8.903648e-01 | -8.486401e-01 | -6.915971e-01 | -7.682956e-01 | -5.540759e-01 | -2.086297e-01 | -6.430976e-01 | .. |
| **50%** | 84692.000000 | 1.810880e-02 | 6.548556e-02 | 1.798463e-01 | -1.984653e-02 | -5.433583e-02 | -2.741871e-01 | 4.010308e-02 | 2.235804e-02 | -5.142873e-02 | .. |
| **75%** | 139320.500000 | 1.315642e+00 | 8.037239e-01 | 1.027196e+00 | 7.433413e-01 | 6.119264e-01 | 3.985649e-01 | 5.704361e-01 | 3.273459e-01 | 5.971390e-01 | .. |
| **max** | 172792.000000 | 2.454930e+00 | 2.205773e+01 | 9.382558e+00 | 1.687534e+01 | 3.480167e+01 | 7.330163e+01 | 1.205895e+02 | 2.000721e+01 | 1.559499e+01 | .. |

8 rows × 31 columns

## 5.3.3: Removing duplicates rows:

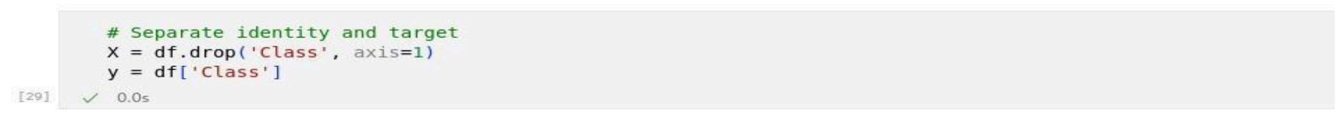


```
In [47]: # Sorting duplicate rows

         df_no_duplicates=df.drop_duplicates()
         print(df_no_duplicates)
```

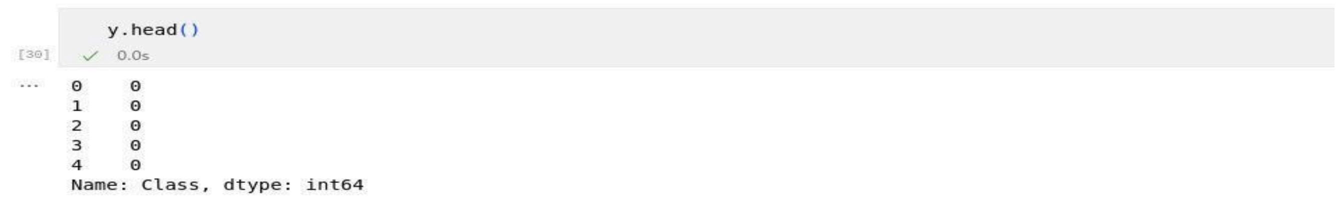
```
            Time         V1         V2        V3        V4        V5  \
0            0.0  -1.359807  -0.072781  2.536347  1.378155 -0.338321
1            0.0   1.191857   0.266151  0.166480  0.448154  0.060018
2            1.0  -1.358354  -1.340163  1.773209  0.379780 -0.503198
3            1.0  -0.966272  -0.185226  1.792993 -0.863291 -0.010309
4            2.0  -1.158233   0.877737  1.548718  0.403034 -0.407193
...          ...        ...        ...       ...       ...       ...
284802  172786.0 -11.881118  10.071785 -9.834783 -2.066656 -5.364473
284803  172787.0  -0.732789  -0.055080  2.035030 -0.738589  0.868229
284804  172788.0   1.919565  -0.301254 -3.249640 -0.557828  2.630515
284805  172788.0  -0.240440   0.530483  0.702510  0.689799 -0.377961
284806  172792.0  -0.533413  -0.189733  0.703337 -0.506271 -0.012546

              V6        V7        V8        V9  ...       V21       V22  \
0       0.462388  0.239599  0.098698  0.363787  ... -0.018307  0.277838
1      -0.082361 -0.078803  0.085102 -0.255425  ... -0.225775 -0.638672
2       1.800499  0.791461  0.247676 -1.514654  ...  0.247998  0.771679
3       1.247203  0.237609  0.377436 -1.387024  ... -0.108300  0.005274
4       0.095921  0.592941 -0.270533  0.817739  ... -0.009431  0.798278
...          ...       ...       ...       ...  ...       ...       ...
284802 -2.606837 -4.918215  7.305334  1.914428  ...  0.213454  0.111864
284803  1.058415  0.024330  0.294869  0.584800  ...  0.214205  0.924384
284804  3.031260 -0.296827  0.708417  0.432454  ...  0.232045  0.578229
284805  0.623708 -0.686180  0.679145  0.392087  ...  0.265245  0.800049
284806 -0.649617  1.577006 -0.414650  0.486180  ...  0.261057  0.643078

             V23       V24       V25       V26       V27       V28  Amount  \
0      -0.110474  0.066928  0.128539 -0.189115  0.133558 -0.021053  149.62
```

## 5.4. Data Pre-Processing:

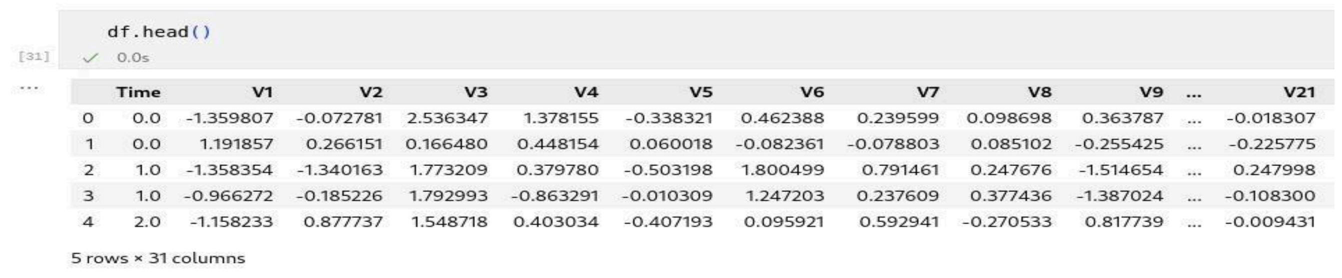
### 5.4.1. Separating the target variable ('class') from all the features:



```
# Separate identity and target
X = df.drop('Class', axis=1)
y = df['Class']
[29] ✓ 0.0s
```



```
y.head()
[30] ✓ 0.0s
...
0    0
1    0
2    0
3    0
4    0
Name: Class, dtype: int64
```

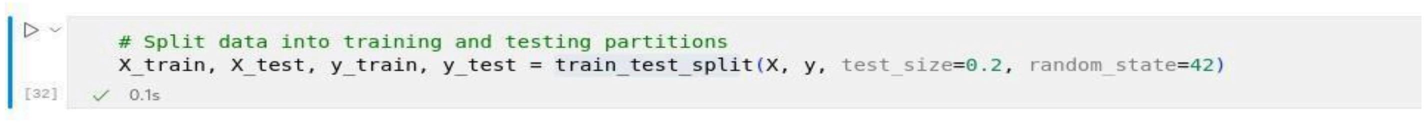


```
df.head()
[31] ✓ 0.0s
```

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V21 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | -1.359807 | -0.072781 | 2.536347 | 1.378155 | -0.338321 | 0.462388 | 0.239599 | 0.098698 | 0.363787 | ... | -0.018307 |
| 1 | 0.0 | 1.191857 | 0.266151 | 0.166480 | 0.448154 | 0.060018 | -0.082361 | -0.078803 | 0.085102 | -0.255425 | ... | -0.225775 |
| 2 | 1.0 | -1.358354 | -1.340163 | 1.773209 | 0.379780 | -0.503198 | 1.800499 | 0.791461 | 0.247676 | -1.514654 | ... | 0.247998 |
| 3 | 1.0 | -0.966272 | -0.185226 | 1.792993 | -0.863291 | -0.010309 | 1.247203 | 0.237609 | 0.377436 | -1.387024 | ... | -0.108300 |
| 4 | 2.0 | -1.158233 | 0.877737 | 1.548718 | 0.403034 | -0.407193 | 0.095921 | 0.592941 | -0.270533 | 0.817739 | ... | -0.009431 |

5 rows × 31 columns

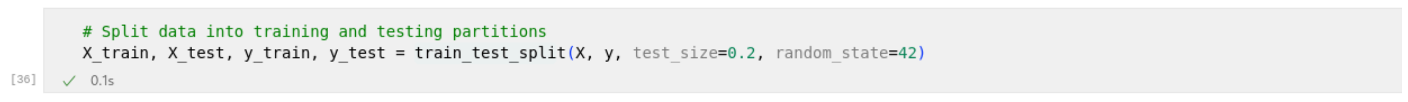
### 5.4.2: Split Data into Training and Testing sets:



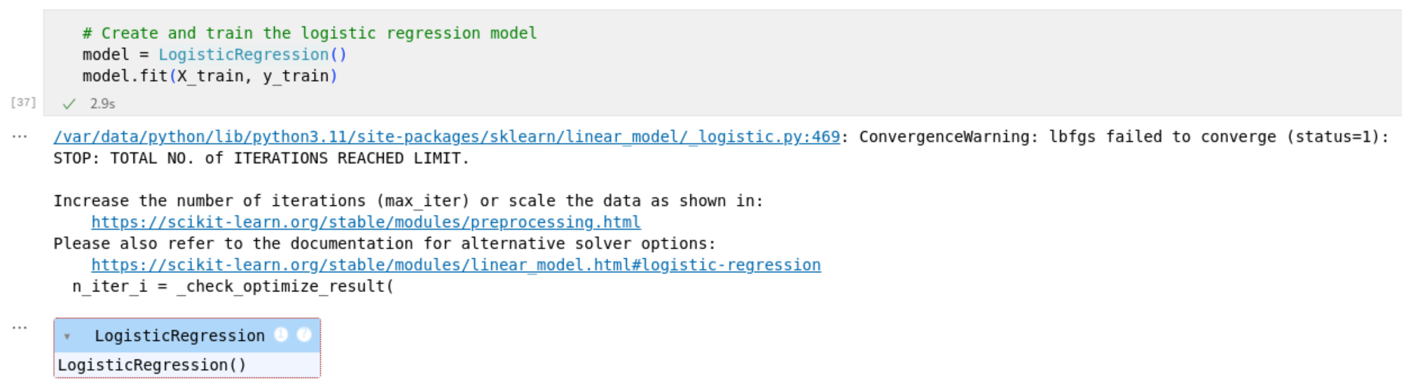
```
# Split data into training and testing partitions
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
[32] ✓ 0.1s
```

## 5.5. Model Building and Training:

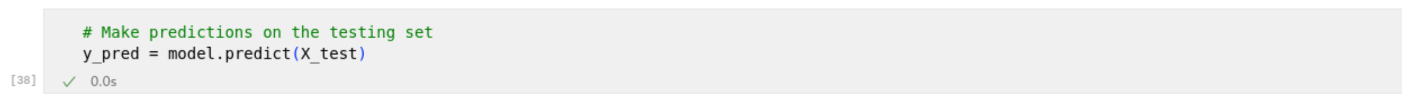
### 5.5.1. Fitting the logistic regression model:



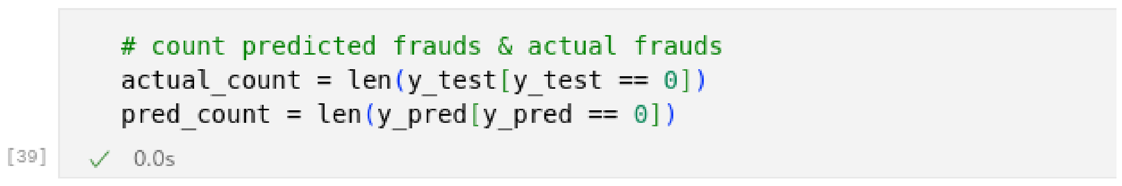
```
# Split data into training and testing partitions
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
[36] ✓ 0.1s
```



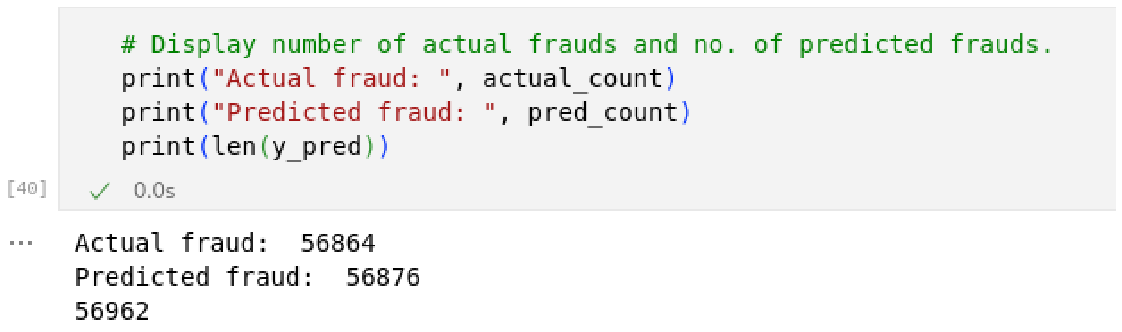
```
# Create and train the logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)
[37] ✓ 2.9s
...
/var/data/python/lib/python3.11/site-packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
...
LogisticRegression
LogisticRegression()
```



```
# Make predictions on the testing set
y_pred = model.predict(X_test)
[38] ✓ 0.0s
```



```
# count predicted frauds & actual frauds
actual_count = len(y_test[y_test == 0])
pred_count = len(y_pred[y_pred == 0])
```
[39] ✓ 0.0s

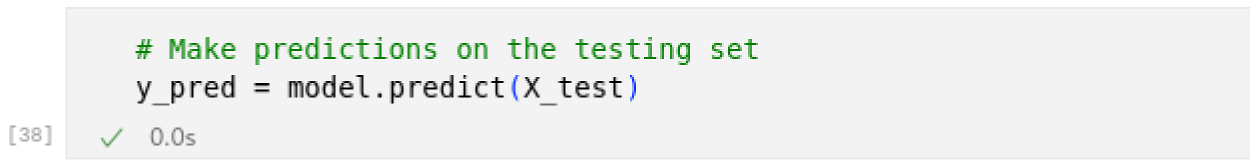


```
# Display number of actual frauds and no. of predicted frauds.
print("Actual fraud: ", actual_count)
print("Predicted fraud: ", pred_count)
print(len(y_pred))
```
[40] ✓ 0.0s

```
Actual fraud:  56864
Predicted fraud:  56876
56962
```

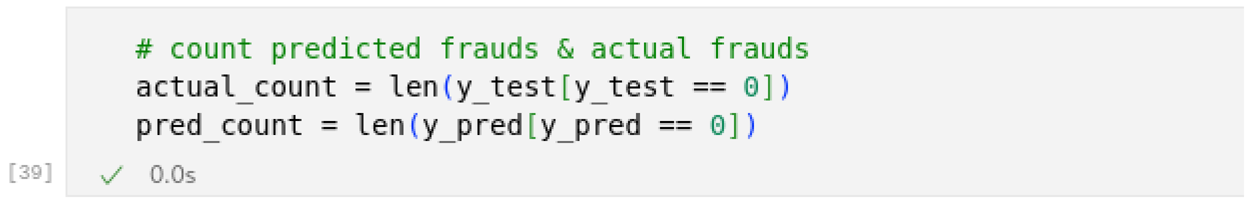
**5.6. Model Evaluation:**

**5.6.1. Make predictions on the Training  set:**



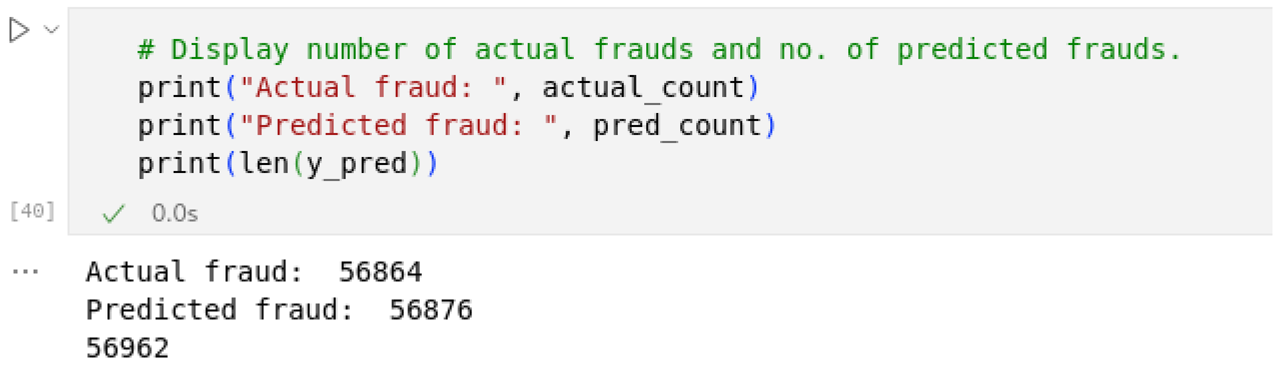
```
# Make predictions on the testing set
y_pred = model.predict(X_test)
```
[38] ✓ 0.0s

**5.6.2. Counting actual no. of fraud transactions and predicted no. of fraud transactions:**



```
# count predicted frauds & actual frauds
actual_count = len(y_test[y_test == 0])
pred_count = len(y_pred[y_pred == 0])
```
[39] ✓ 0.0s

**5.6.3. Displaying the calculated count:**

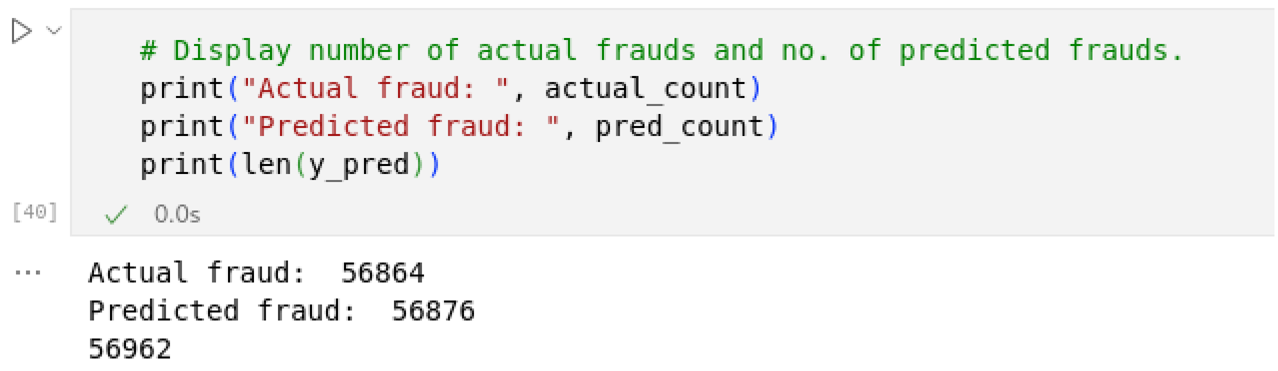


```
# Display number of actual frauds and no. of predicted frauds.
print("Actual fraud: ", actual_count)
print("Predicted fraud: ", pred_count)
print(len(y_pred))
```
[40] ✓ 0.0s

```
Actual fraud:  56864
Predicted fraud:  56876
56962
```

## 5.7. Calculating Model’s Accuracy:

### 5.7.1. Calculate and display model performance :



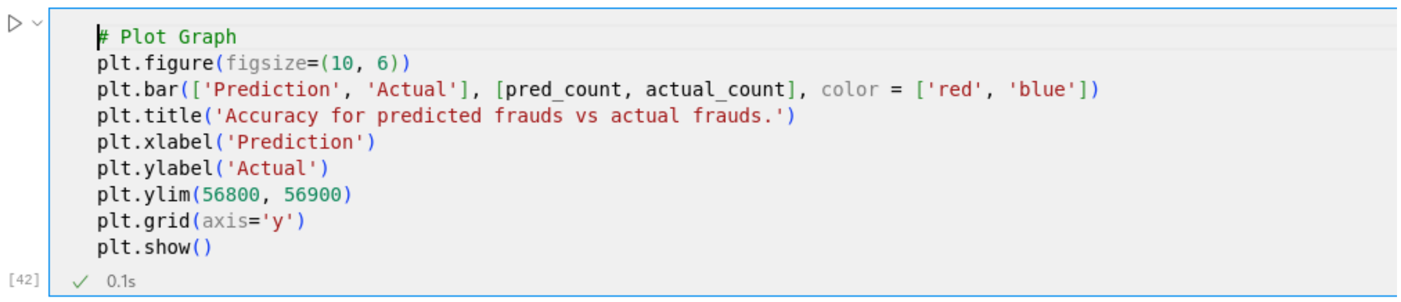
```
# Display number of actual frauds and no. of predicted frauds.
print("Actual fraud: ", actual_count)
print("Predicted fraud: ", pred_count)
print(len(y_pred))
```

[40] ✓ 0.0s

```
Actual fraud:  56864
Predicted fraud:  56876
56962
```

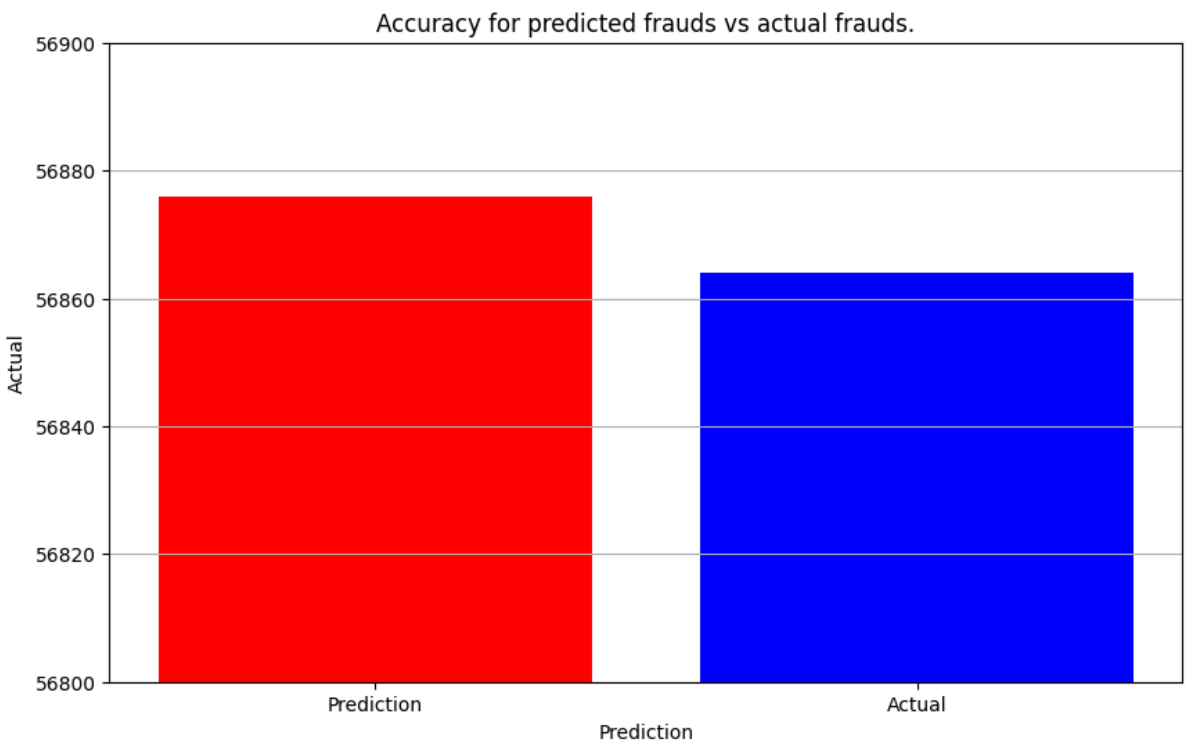
## 5.8. Visually representing the above stats:

### 5.8.1. Actual vs Prediction graph:



```
# Plot Graph
plt.figure(figsize=(10, 6))
plt.bar(['Prediction', 'Actual'], [pred_count, actual_count], color = ['red', 'blue'])
plt.title('Accuracy for predicted frauds vs actual frauds.')
plt.xlabel('Prediction')
plt.ylabel('Actual')
plt.ylim(56800, 56900)
plt.grid(axis='y')
plt.show()
```

[42] ✓ 0.1s



## 5.9. Plotting Confusion Matrix:

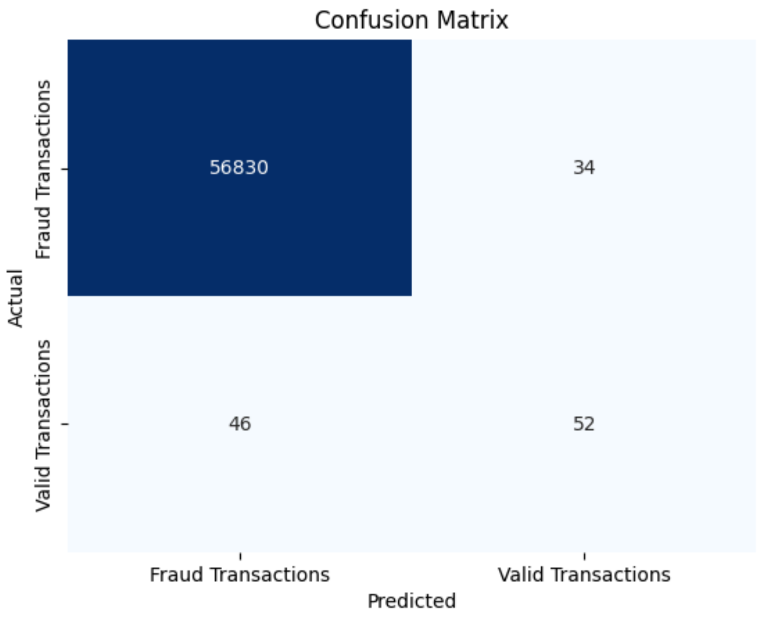
```
#Creating the confusion matirx
cm = confusion_matrix(y_test, y_pred)
cm
```

```
[43] ✓ 0.0s
```

```
array([[56830,    34],
       [   46,    52]])
```

```
#Plotting the confusion Matrix
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.xticks(ticks=[0.5, 1.5], labels=["Fraud Transactions", "Valid Transactions"])
plt.yticks(ticks=[0.5, 1.5], labels=["Fraud Transactions", "Valid Transactions"])
plt.show()
```

```
[44] ✓ 0.0s
```



# 6. ALGORITHM

## 6.1. Logistic Regression:

Logistic regression stands out as a valuable tool in credit card fraud detection due to its interpretability, efficiency, and scalability. With the ability to provide easily interpretable results, logistic regression allows financial institutions to understand the factors influencing fraudulent activities, as coefficients associated with each feature indicate their impact on the likelihood of fraud. Moreover, logistic regression's computational efficiency makes it well-suited for processing the massive amounts of transaction data encountered in fraud detection, facilitating real-time analysis. Its scalability ensures that the model remains effective even as transaction volumes increase over time. Additionally, logistic regression's robustness to noise and irrelevant features in the dataset ensures that it can effectively filter out irrelevant information, focusing on features most indicative of fraudulent behavior. Furthermore, logistic regression's probabilistic output enables financial institutions to assess the likelihood of fraud for each transaction, facilitating prioritization for further review or investigation. Lastly, logistic regression's ability to handle imbalanced data through techniques such as adjusting decision thresholds or using class weights ensures that the model adequately captures fraudulent activities while minimizing false positives. Overall, logistic regression offers a powerful and interpretable approach to credit card fraud detection, empowering financial institutions to identify and prevent fraudulent transactions effectively.

## 6.2. Advantages of using Logistic Regression

Logistic regression offers several advantages that make it a popular choice for various classification tasks, including credit card fraud detection:

1. Interpretability: Logistic regression provides easily interpretable results, as the coefficients associated with each feature indicate the direction and magnitude of their impact on the outcome. This interpretability allows stakeholders to understand the factors influencing the classification decision, making logistic regression particularly useful in domains where interpretability is essential, such as finance and healthcare.

2. Efficiency: Logistic regression is computationally efficient and requires relatively low computational resources compared to more complex machine learning algorithms. This efficiency makes logistic regression well-suited for large-scale datasets and real-time applications, such as processing the vast volume of credit card transactions in fraud detection systems.

3. Scalability: Logistic regression can handle large datasets with ease, making it scalable to accommodate increasing data volumes over time. Its scalability ensures that the model remains effective and efficient as the size of the dataset grows, making it suitable for applications with rapidly expanding data requirements.

4. Robustness to Noise: Logistic regression is robust to noise and irrelevant features in the dataset, allowing it to effectively filter out irrelevant information and focus on features that are most predictive of the outcome. This robustness helps improve the model's performance in noisy datasets and reduces the risk of overfitting, making logistic regression a reliable choice for real-world applications.

5. Probabilistic Output: Logistic regression provides probabilistic outputs, estimating the probability of each class rather than just the predicted class label. This probabilistic output enables stakeholders to assess the confidence of the model's predictions and make informed decisions based on the likelihood of different outcomes. In credit card fraud detection, probabilistic outputs can help prioritize transactions for further investigation based on their likelihood of being fraudulent.

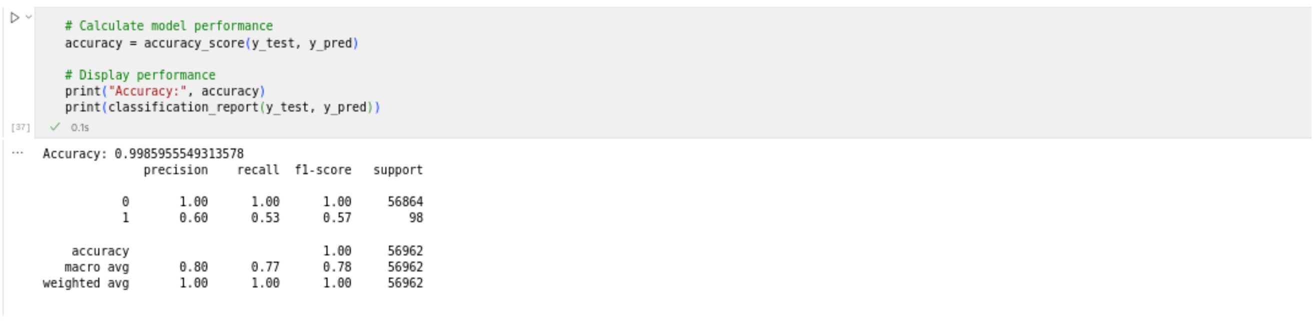
6. Handling Imbalanced Data: Logistic regression can handle imbalanced datasets, where one class may be significantly more prevalent than the other, by adjusting the decision threshold or using techniques such as class weights. This capability is particularly useful in fraud detection, where fraudulent transactions are typically rare compared to legitimate transactions, ensuring that the model adequately captures fraudulent activities while minimizing false positives.

7. Why we use logistic regression over anything else:

- Handles binary classification better comparatively.
- Is more accurate when provided with a large dataset with more diverse and variant data.

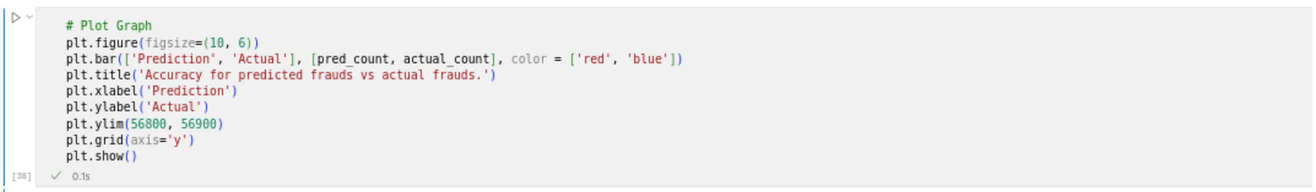
## 7. MODEL'S ACCURACY

Accuracy provides a straightforward measure of how well a model is performing, but it should be interpreted in conjunction with other evaluation metrics to gain a more nuanced understanding of the model's strengths and weaknesses.

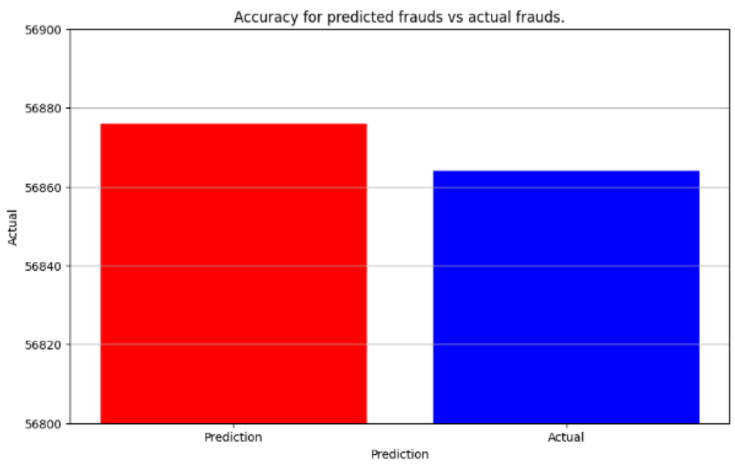


```
# Calculate model performance
accuracy = accuracy_score(y_test, y_pred)

# Display performance
print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))
```

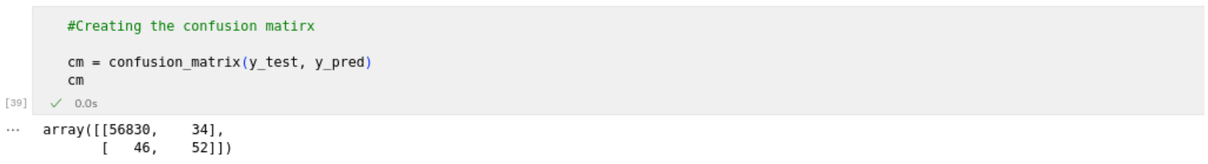
```
Accuracy: 0.9985955549313578
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56864
           1       0.60      0.53      0.57        98

    accuracy                           1.00     56962
   macro avg       0.80      0.77      0.78     56962
weighted avg       1.00      1.00      1.00     56962
```



```
# Plot Graph
plt.figure(figsize=(10, 6))
plt.bar(['Prediction', 'Actual'], [pred_count, actual_count], color = ['red', 'blue'])
plt.title('Accuracy for predicted frauds vs actual frauds.')
plt.xlabel('Prediction')
plt.ylabel('Actual')
plt.ylim(56800, 56900)
plt.grid(axis='y')
plt.show()
```

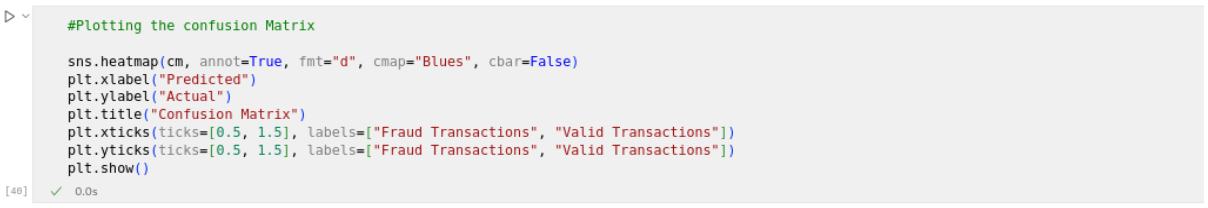


***A confusion matrix*** serves as a critical evaluation tool. By analyzing the model's predictions against the actual labels in the test dataset, the confusion matrix illustrates the classification performance. Specifically, it captures the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions made by the model. This granular breakdown enables a thorough assessment of the model's accuracy and ability to correctly classify fraudulent and non-fraudulent transactions. Visualization of the confusion matrix, typically in the form of a heatmap, offers a clear depiction of the model's classification performance, aiding in the identification of areas for improvement and optimization. Through the analysis of the confusion matrix, practitioners can refine the model's predictive capabilities, ultimately enhancing its effectiveness in detecting credit card fraud.

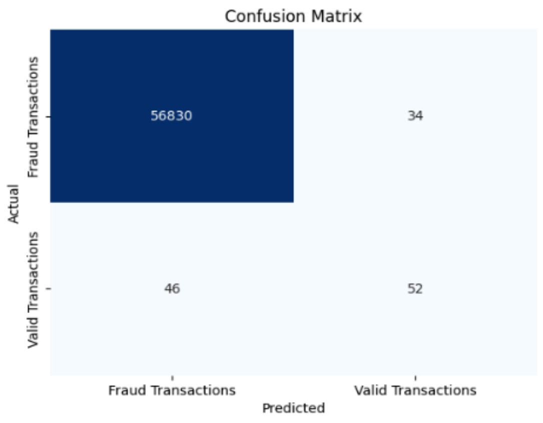


```
#Creating the confusion matirx

cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[56830,    34],
       [   46,    52]])
```



```
#Plotting the confusion Matrix

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.xticks(ticks=[0.5, 1.5], labels=["Fraud Transactions", "Valid Transactions"])
plt.yticks(ticks=[0.5, 1.5], labels=["Fraud Transactions", "Valid Transactions"])
plt.show()
```



## 8. CONCLUSION

In conclusion, the rising prevalence of credit card fraud in tandem with the increasing digitization of financial transactions underscores the urgency for robust fraud detection systems. This project endeavors to address this pressing issue by harnessing the power of machine learning algorithms in Python.

Through the utilization of techniques such as Logistic Regression and Random Forests, alongside libraries like Pandas for data pre-processing and analysis, our project aims to enhance the security measures employed by financial institutions. By leveraging historical transaction data, our models seek to discern patterns indicative of fraudulent activity, thereby enabling real-time detection and prevention of unauthorized transactions.

Despite the challenges posed by the massive volume of daily transactions and the imbalanced nature of legitimate versus fraudulent activities, our approach emphasizes the deployment of efficient models and techniques tailored to handle such complexities. By striving for high accuracy while minimizing false positives, we aim to provide valuable tools for financial organizations to combat evolving fraud schemes and safeguard their customers' financial interests.

In a rapidly evolving landscape where cyber threats continue to evolve, the insights gleaned from this project contribute towards the development of more resilient fraud detection systems. By staying proactive and adaptable, we can help mitigate financial losses and bolster consumer confidence in digital payment systems, ultimately fostering a safer and more secure financial environment for all stakeholders.

## 9. REFERENCES

**[1] Basic Idea of Problem**
https://www.geeksforgeeks.org/ml-credit-card-fraud-detection/

**[2]How to import libraries?**
https://towardsdatascience.com/credit-card-fraud-detection-using-machine-learning-python-5b098d4a8edc

**[3]        Dataset**
https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

**[4] Machine learning based credit card fraud detection using the GA algorithm for feature selection**
https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00573-8

**[5] Case study of Infosys that is using this same model**
https://www.infosysbpm.com/blogs/bpm-analytics/machine-learning-for-credit-card-fraud-detection.html

**[5]Challenges and Techniques**
https://www.infosysbpm.com/blogs/bpm-analytics/machine-learning-for-credit-card-fraud-detection.html

**[6]Reviews**
https://link.springer.com/article/10.1007/s44230-022-00004-0

**SOME OTHER REFERENCES:**

- https://repository.rit.edu/cgi/viewcontent.cgi?article=12455&context=theses
- https://www.sciencedirect.com/science/article/pii/S187705092030065X
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10535547/
- https://www.researchgate.net/publication/336800562_Credit_Card_Fraud_Detection_using_Machine_Learning_and_Data_Science
- https://link.springer.com/article/10.1007/s44230-022-00004-0
- https://www.mdpi.com/2504-2289/8/1/6
- https://ieeexplore.ieee.org/document/9121114
- https://www.sciencedirect.com/science/article/pii/S1319157822004062https://www.sciencedirect.com/science/article/pii/S1319157822004062
- https://www.youtube.com/watch?v=0pGXW0bCgHE
- https://www.youtube.com/watch?v=0pGXW0bCgHE
- https://www.kaggle.com/datasets/kartik2112/fraud-detection