Credit Card Fraud Detection - ML

Last Updated : 02 Sep, 2025

The goal of this project is to develop a machine learning model that can accurately detect fraudulent credit card transactions using historical data. By analyzing transaction patterns, the model should be able to distinguish between normal and fraudulent activity, helping financial institutions flag suspicious behavior early and reduce potential risks.

Challenges include:

- Handling imbalanced datasets where fraud cases are a small fraction of total transactions.
- Ensuring high precision to minimize false positives (flagging a valid transaction as fraud).
- Ensuring high recall to detect as many fraud cases as possible.

Step 1: Importing necessary Libraries

We begin by importing the necessary Python libraries: numpy, pandas, matplotlib and seaborn for data handling, visualization and model building.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import gridspec
```

Step 2: Loading the Data

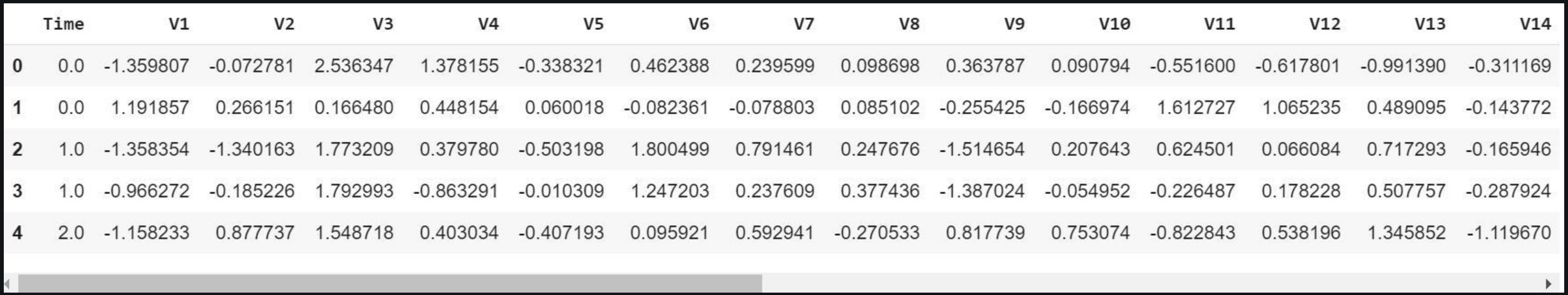
Load the dataset into a pandas DataFrame and examine its structure. The dataset contains **284,807 transactions** with **31 features** including:

- **Time:** This shows how many seconds have passed since the first transaction in the dataset.
- **V1-V28:** These are special features created to hide sensitive information about the original data.
- **Amount:** Transaction amount.
- **Class:** Target variable (0 for normal transactions, 1 for fraudulent transactions).

You can download the dataset from here.

```
data = pd.read_csv("creditcard.csv")
print(data.head())
```

Output:



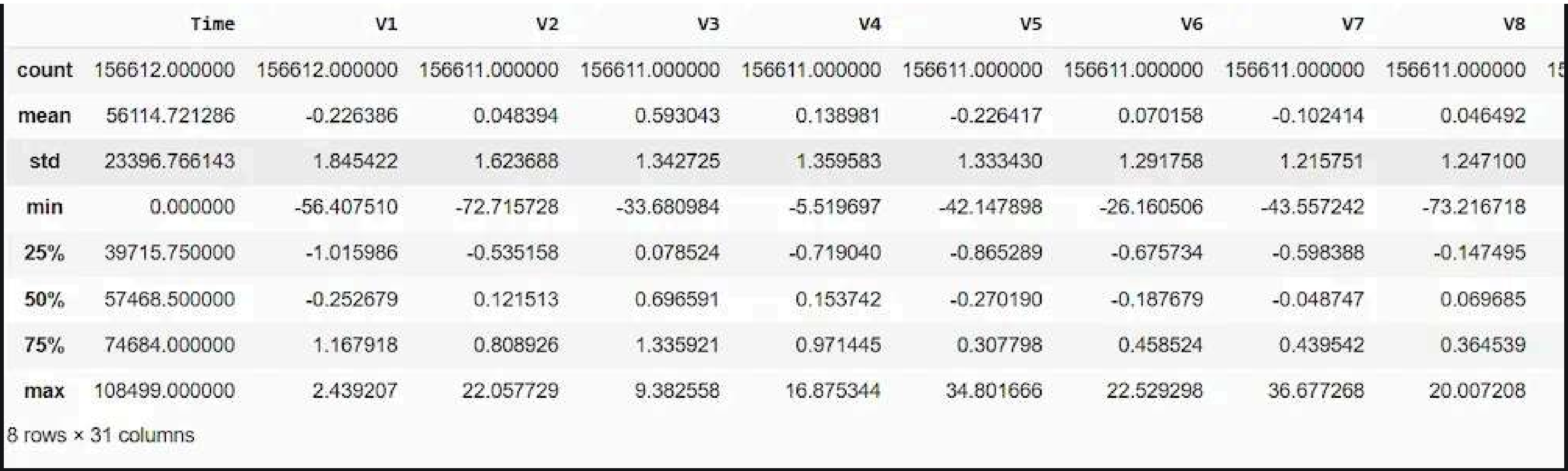
	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670

Credit card data received

Now, let's explore more about the dataset using df.describe() method.

```
print(data.describe())
```

Output :



	Time	V1	V2	V3	V4	V5	V6	V7	V8	
count	156612.000000	156612.000000	156611.000000	156611.000000	156611.000000	156611.000000	156611.000000	156611.000000	156611.000000	15
mean	56114.721286	-0.226386	0.048394	0.593043	0.138981	-0.226417	0.070158	-0.102414	0.046492	
std	23396.766143	1.845422	1.623688	1.342725	1.359583	1.333430	1.291758	1.215751	1.247100	
min	0.000000	-56.407510	-72.715728	-33.680984	-5.519697	-42.147898	-26.160506	-43.557242	-73.216718	
25%	39715.750000	-1.015986	-0.535158	0.078524	-0.719040	-0.865289	-0.675734	-0.598388	-0.147495	
50%	57468.500000	-0.252679	0.121513	0.696591	0.153742	-0.270190	-0.187679	-0.048747	0.069685	
75%	74684.000000	1.167918	0.808926	1.335921	0.971445	0.307798	0.458524	0.439542	0.364539	
max	108499.000000	2.439207	22.057729	9.382558	16.875344	34.801666	22.529298	36.677268	20.007208	

8 rows × 31 columns

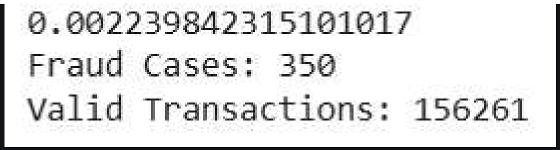
Step 3: Analyzing Class Distribution

The next step is to check the distribution of fraudulent vs. normal transactions.

- We separate the dataset into two groups: fraudulent transactions **(Class == 1)** and valid transactions **(Class == 0)**.
- It calculates the ratio of fraud cases to valid cases to measure how imbalanced the dataset is.
- It then prints the outlier fraction along with the number of fraud and valid transactions.
- This analysis is crucial in fraud detection, as it reveals how rare fraud cases are and whether techniques like resampling or special evaluation metrics are needed.

```
fraud = data[data['Class'] == 1]
valid = data[data['Class'] == 0]
outlierFraction = len(fraud)/float(len(valid))
print(outlierFraction)
print('Fraud Cases: {}'.format(len(data[data['Class'] == 1])))
print('Valid Transactions: {}'.format(len(data[data['Class'] == 0])))
```

Output:



```
0.0022398423151010l7
Fraud Cases: 350
Valid Transactions: 156261
```

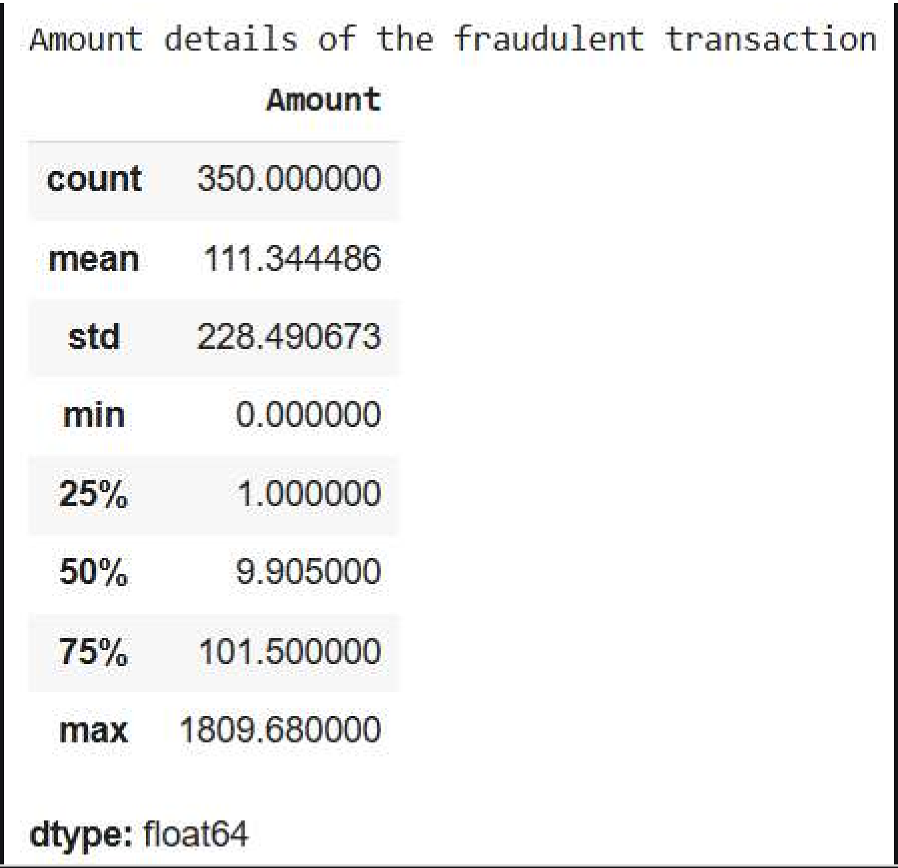
Since the dataset is highly imbalanced with only 0.02% fraudulent transactions. we’ll first try to build a model without balancing the dataset. If we don’t get satisfactory results we will explore ways to handle the imbalance.

Step 4: Exploring Transaction Amounts

Let's compare the transaction amounts for fraudulent and normal transactions. This will help us understand if there are any significant differences in the monetary value of fraudulent transactions.

```python
print("Amount details of the fraudulent transaction")
fraud.Amount.describe()
```

Output :



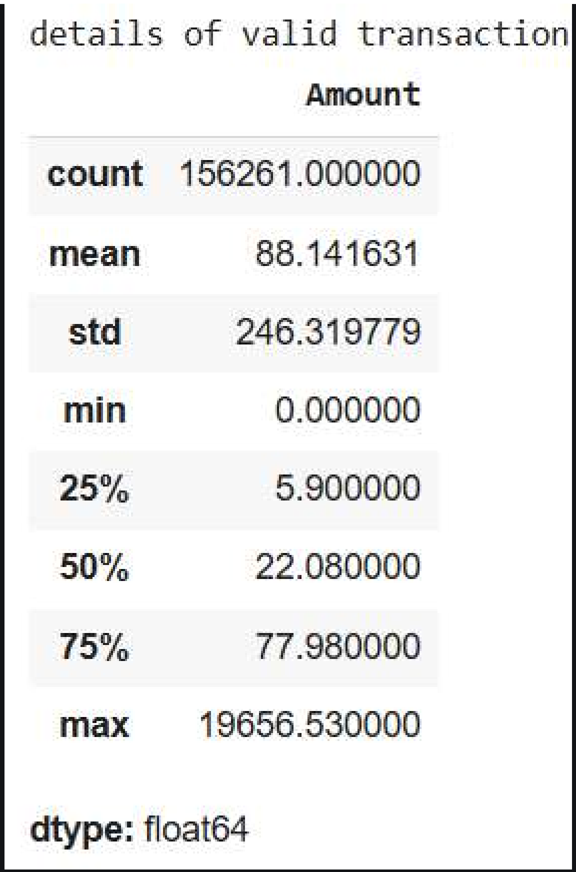
```
Amount details of the fraudulent transaction
```

	Amount
count	350.000000
mean	111.344486
std	228.490673
min	0.000000
25%	1.000000
50%	9.905000
75%	101.500000
max	1809.680000

dtype: float64

```
print("details of valid transaction")
valid.Amount.describe()
```

Output :


details of valid transaction

	Amount
count	156261.000000
mean	88.141631
std	246.319779
min	0.000000
25%	5.900000
50%	22.080000
75%	77.980000
max	19656.530000

dtype: float64

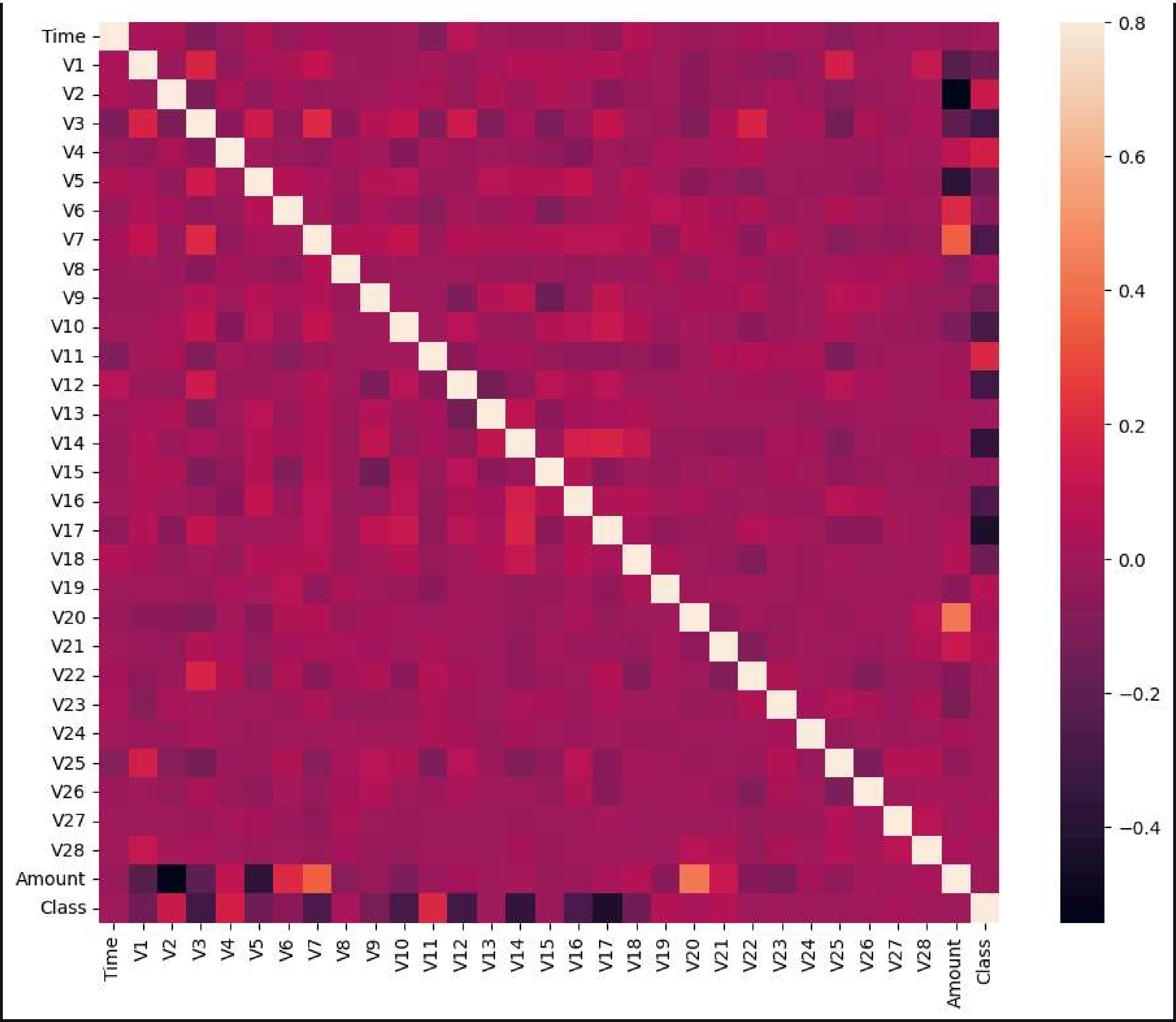
From the output we observe that fraudulent transactions tend to have higher average amounts which is important in fraud detection.

Step 5: Plotting Correlation Matrix

We can visualize the correlation between features using a heatmap using correlation matrix. This will give us an understanding of how the different features are correlated and which ones may be more relevant for prediction.

```
corrmat = data.corr()
fig = plt.figure(figsize = (12, 9))
sns.heatmap(corrmat, vmax = .8, square = True)
plt.show()
```

Output:


Time
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20
V21
V22
V23
V24
V25
V26
V27
V28
Amount
Class
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

Most features do not correlate strongly with others but some features like V2 and V5 have a negative correlation with the Amount feature. This provides valuable insights into how the features are related to the transaction amounts.

Step 6: Preparing Data

Separate the input features (X) and target variable (Y) then split the data into training and testing sets

- **X = data.drop(['Class'], axis = 1)** removes the target column (Class) from the dataset to keep only the input features.
- **Y = data["Class"]** selects the Class column as the target variable (fraud or not).
- **X.shape and Y.shape** print the number of rows and columns in the feature set and the target set.
- **xData = X.values and yData = Y.values** convert the Pandas DataFrame or Series to NumPy arrays for faster processing.
- **train_test_split(...)** splits the data into training and testing sets into 80% for training, 20% for testing.
- **random_state=42** ensures reproducibility (same split every time you run it).

```
X = data.drop(['Class'], axis = 1)
Y = data["Class"]
print(X.shape)
print(Y.shape)

xData = X.values
yData = Y.values

from sklearn.model_selection import train_test_split
xTrain, xTest, yTrain, yTest = train_test_split(
        xData, yData, test_size = 0.2, random_state = 42)
```

Output :

```
(284807, 30)
(284807,)
```

Step 7: Building and Training the Model

Train a Random Forest Classifier to predict fraudulent transactions.

- **from sklearn.ensemble import RandomForestClassifier:** This imports the RandomForestClassifier from sklearn.ensemble, which is used to create a random forest model for classification tasks.
- **rfc = RandomForestClassifier():** Initializes a new instance of the RandomForestClassifier.
- **rfc.fit(xTrain, yTrain):** Trains the RandomForestClassifier model on the training data (xTrain for features and yTrain for the target labels).
- **yPred = rfc.predict(xTest):** Uses the trained model to predict the target labels for the test data (xTest), storing the results in yPred.

```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier()
rfc.fit(xTrain, yTrain)

yPred = rfc.predict(xTest)
```

Step 8: Evaluating the Model

After training the model we need to evaluate its performance using various metrics such as accuracy, precision, recall, F1-score and the Matthews correlation coefficient.

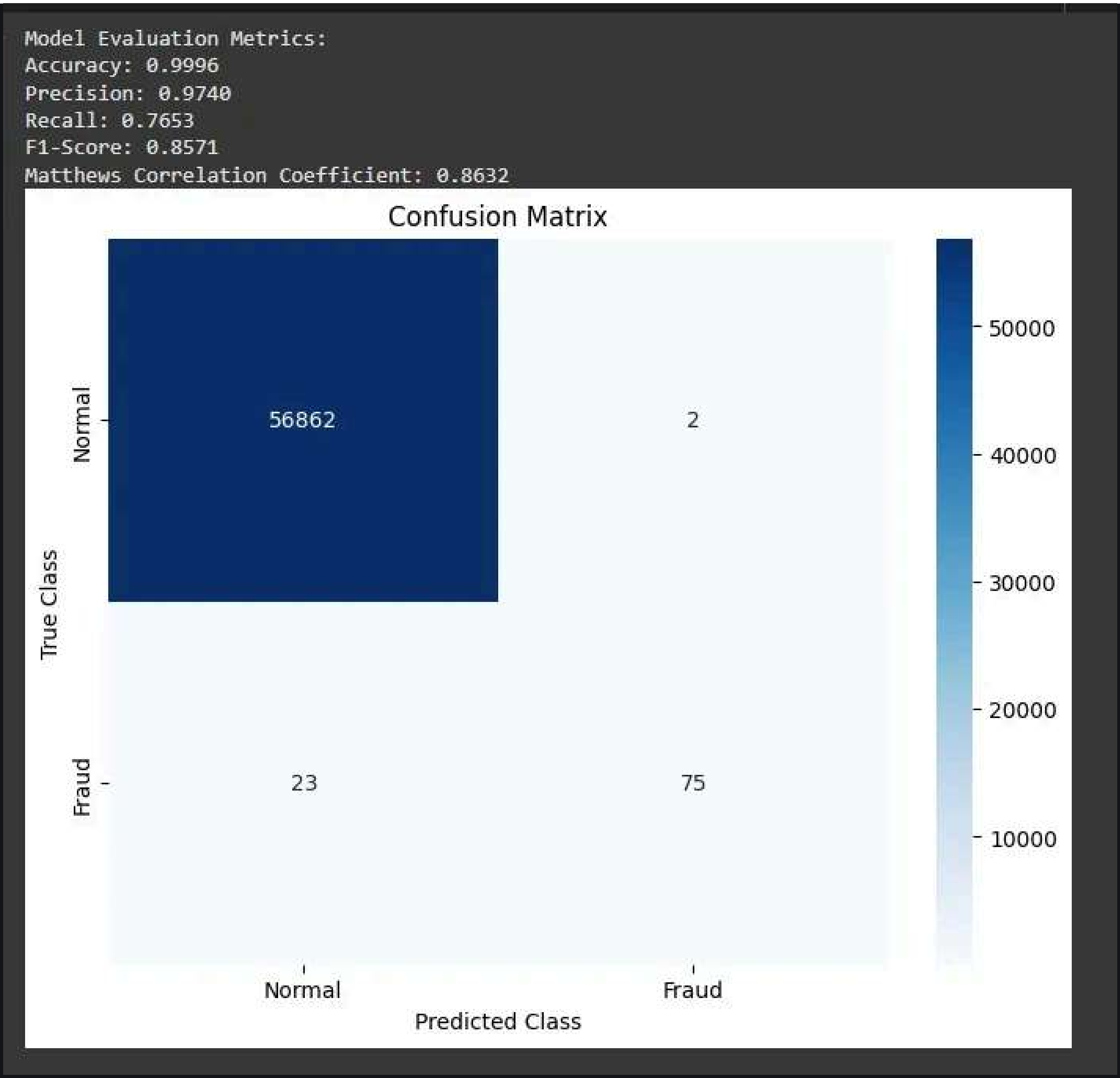
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
matthews_corrcoef, confusion_matrix
accuracy = accuracy_score(yTest, yPred)
```

```
precision = precision_score(yTest, yPred)
recall = recall_score(yTest, yPred)
f1 = f1_score(yTest, yPred)
mcc = matthews_corrcoef(yTest, yPred)

print("Model Evaluation Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
print(f"Matthews Correlation Coefficient: {mcc:.4f}")

conf_matrix = confusion_matrix(yTest, yPred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Normal', 'Fraud'], yticklabels=['Normal', 'Fraud'])
plt.title("Confusion Matrix")
plt.xlabel("Predicted Class")
plt.ylabel("True Class")
plt.show()
```

Output :



```
Model Evaluation Metrics:
Accuracy: 0.9996
Precision: 0.9740
Recall: 0.7653
F1-Score: 0.8571
Matthews Correlation Coefficient: 0.8632
```

Evaluating the Model

Model Evaluation Metrics:

The model accuracy is high due to class imbalance so we will have computed precision, recall and f1 score to get a more meaningful understanding. We observe:

- **Accuracy: 0.9996**: Out of all predictions, 99.96% were correct. However, in imbalanced datasets (like fraud detection), accuracy can be misleading i.e. a model that predicts everything as "not fraud" will still have high accuracy.
- **Precision: 0.9873:** When the model predicted "fraud", it was correct 98.73% of the time. High precision means very few false alarms (false positives).
- **Recall: 0.7959:** Out of all actual fraud cases, the model detected 79.59%. This shows how well it catches real frauds. A lower recall means some frauds were missed (false negatives).
- **F1-Score: 0.8814:** A balance between precision and recall. 88.14% is strong and shows the model handles both catching fraud and avoiding false alarms well.
- **Matthews Correlation Coefficient (MCC): 0.8863:** A more balanced score (from -1 to +1) even when classes are imbalanced. A value of 0.8863 is very good, it means the model is making strong, balanced predictions overall.

We can balance dataset by oversampling the minority class or by undersampling the majority class we can increase accuracy of our model.

Get complete notebook link here: *click here.*

Comment

amank... + Follow

44

Article Tags : Project Machine Learning AI-ML-DS python +2 More

Explore