

# Identificatore di transazioni fraudolente

Progetto di Machine Learning a.a. 2025-2026

# Qual è il problema?

I pagamenti digitali ha reso le transazioni online comode ma più esposte a frodi finanziarie, che rappresentano un problema significativo per le istituzioni finanziarie e i loro clienti.

L'obiettivo è la realizzazione di un sistema di individuazione automatica di transazioni potenzialmente fraudolente, tramite un modello di Machine Learning.

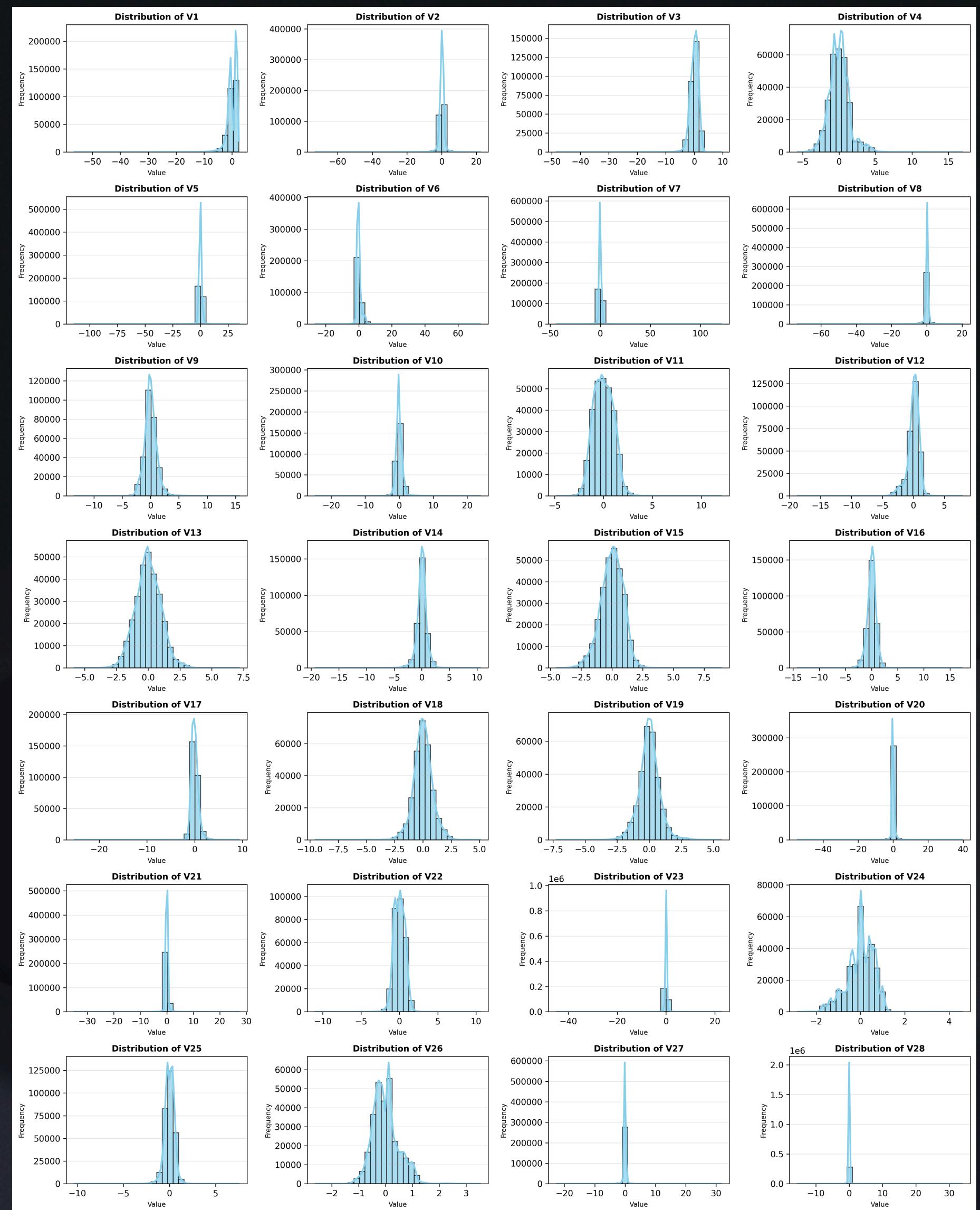


# Dataset utilizzato

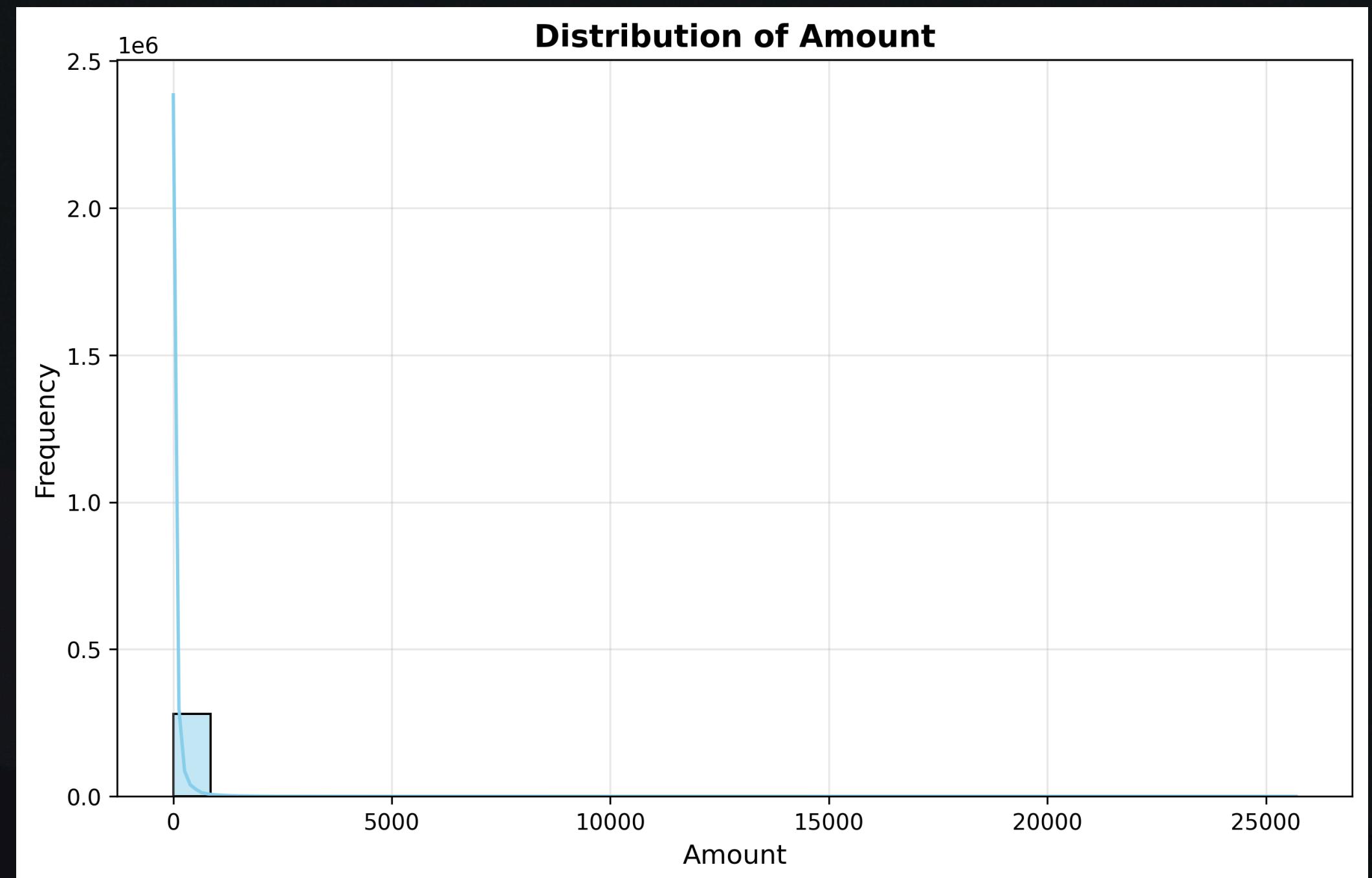
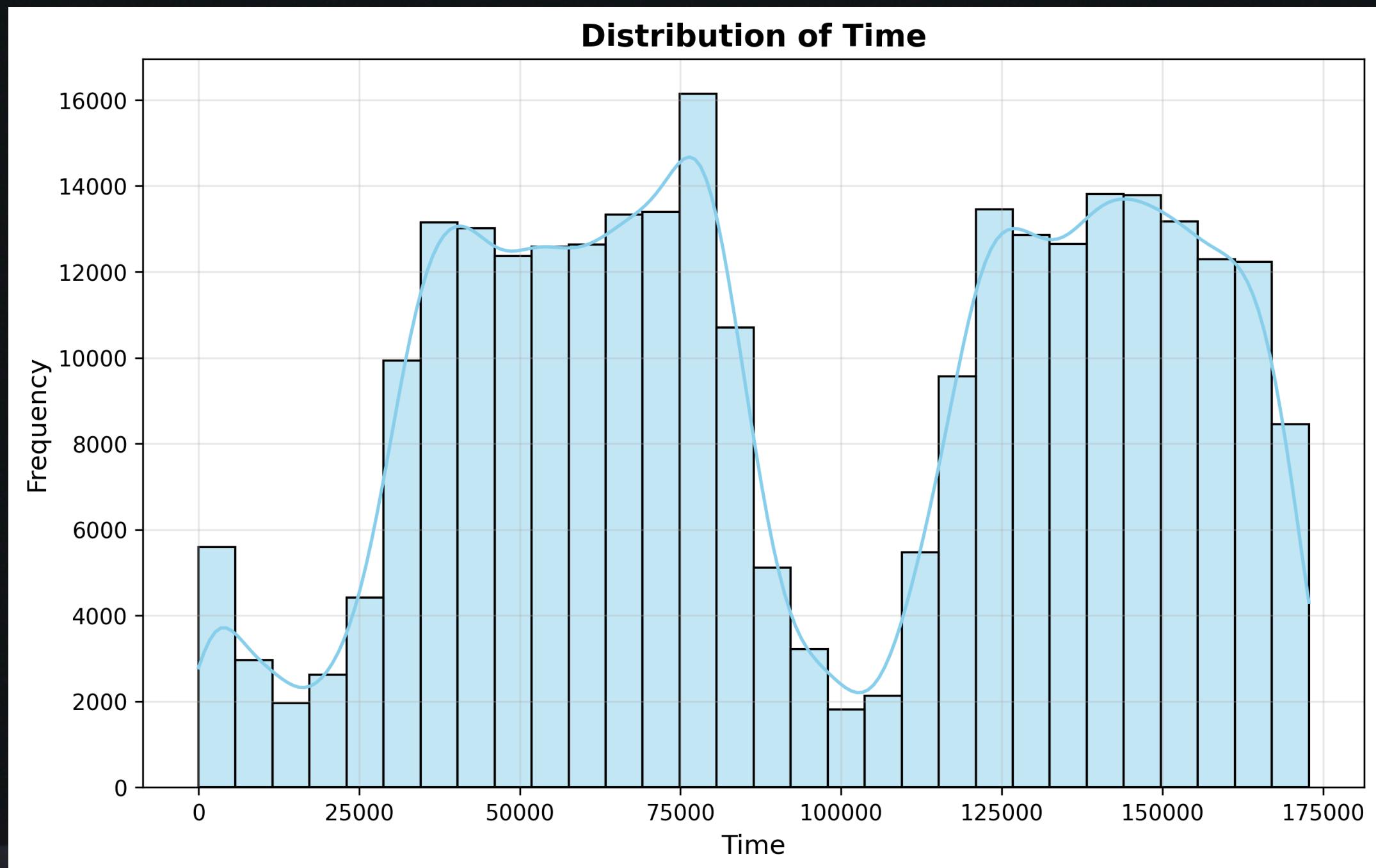
Il dataset è **Credit Card Fraud Detection** di Machine Learning Group of Université Libre de Bruxelles, disponibile su Kaggle (<https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud>), formato in totale da 284,807 campioni e da 31 feature numeriche:

- Time: tempo passato tra la prima transazione del dataset e ogni altra transazione, in numero di secondi;
- V1-V28: 28 diverse feature che rappresentano tutte le informazioni personali censurate tramite trasformazione PCA;
- Amount: quantità di denaro trasferito con la transazione;
- Class: classificazione dell'attività, le attività con Class=0 sono legittime mentre quelle con Class=1 sono fraudolente.

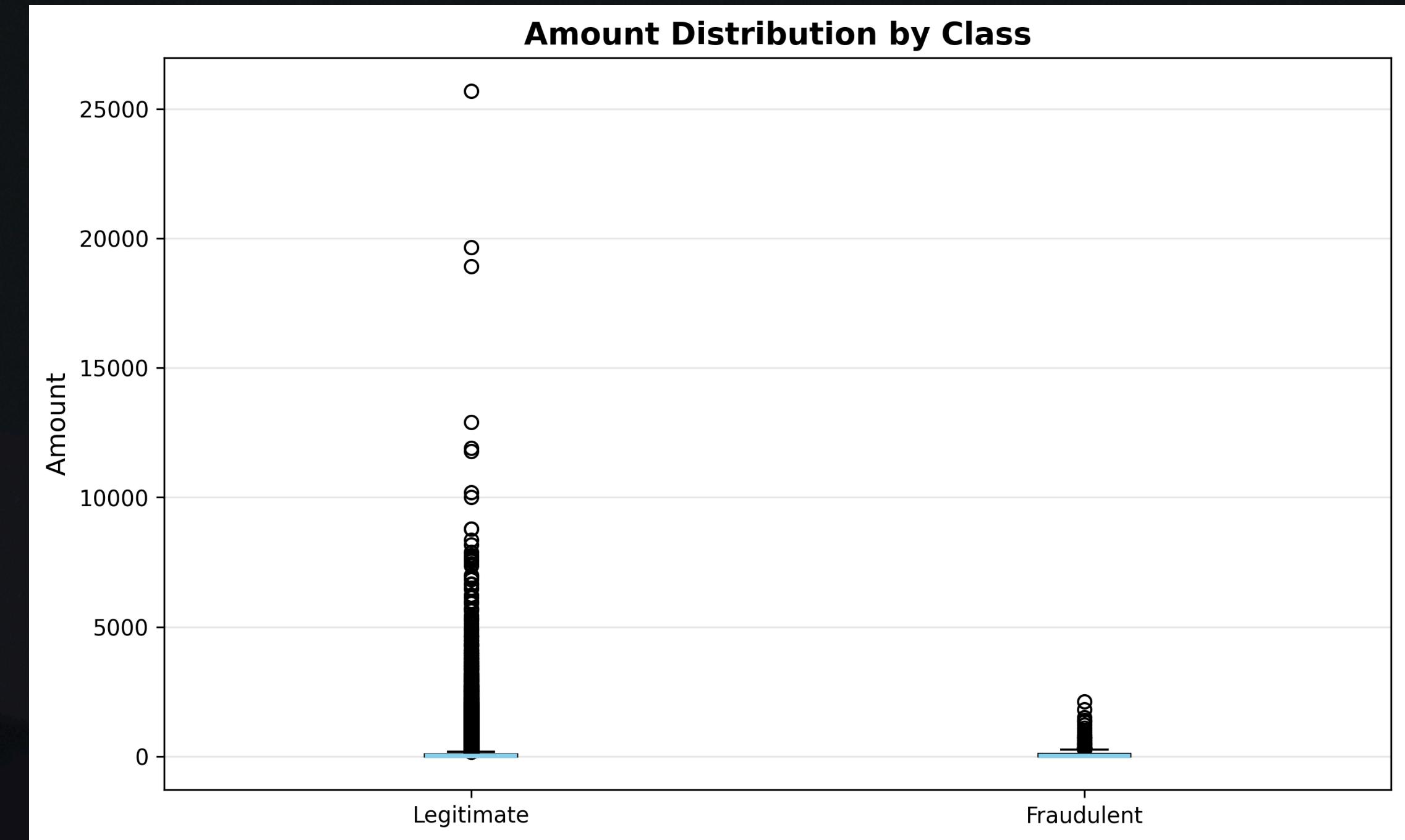
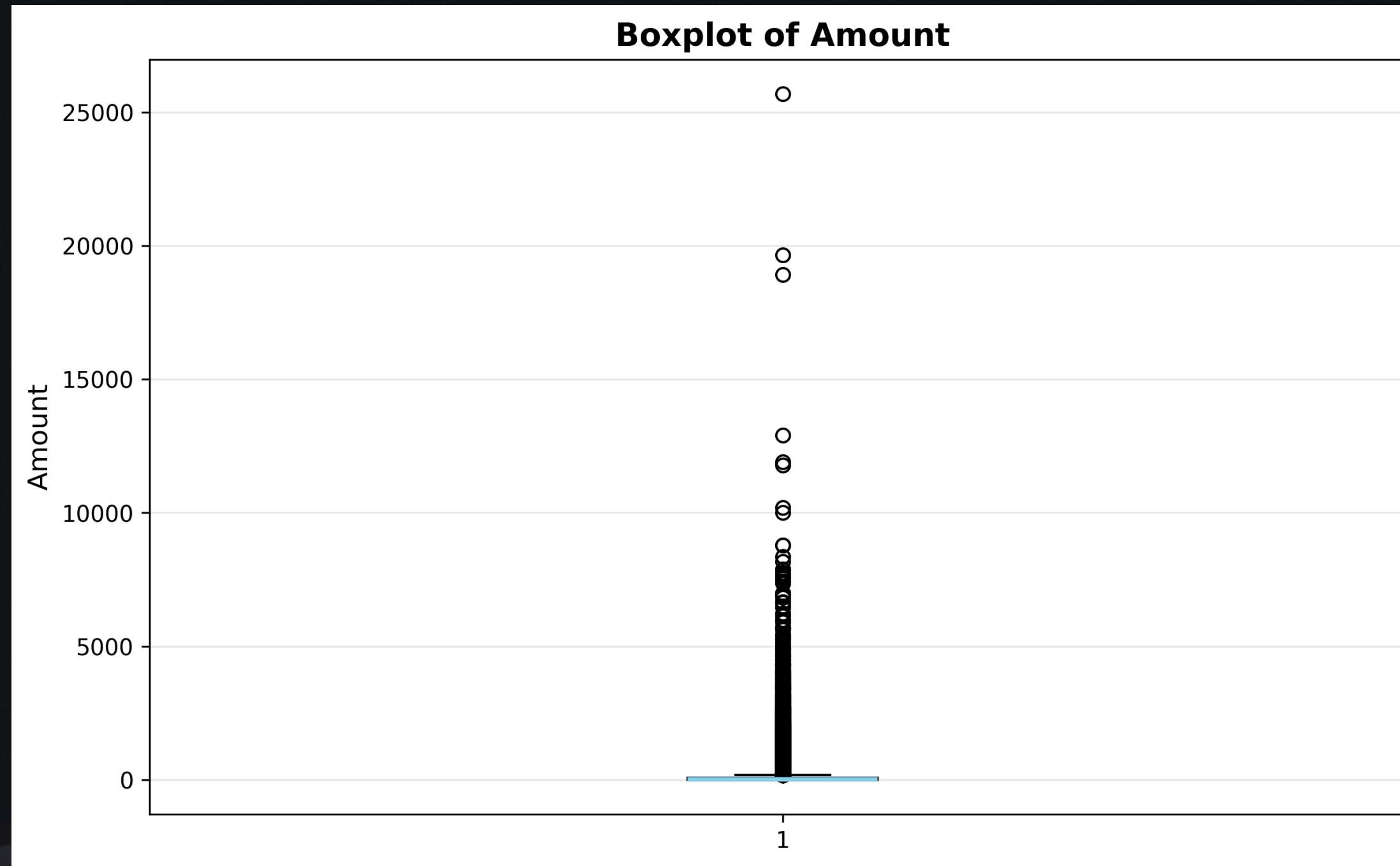
# Distribuzioni feature V1-V28



# Distribuzioni feature Time e Amount



# Boxplot feature Amount



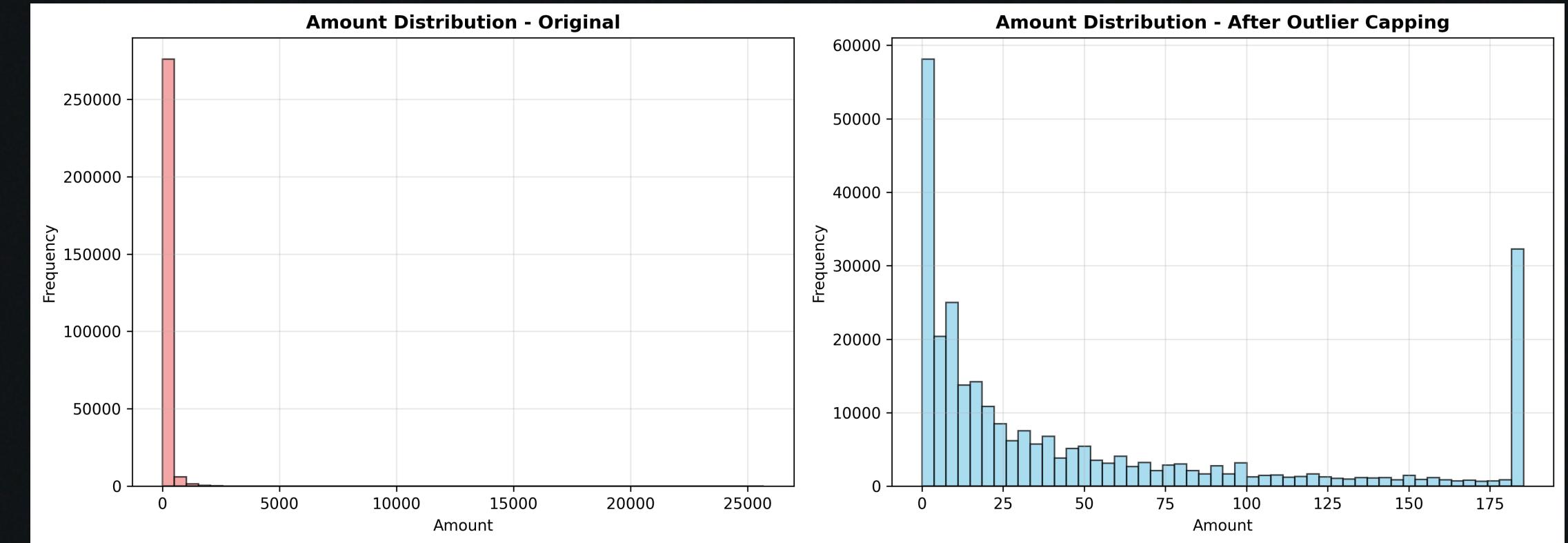
# Sbilanciamento del dataset

Class	Samples	Percentage
Legitimate	284,315	99.83%
Fraudulent	492	0.17%

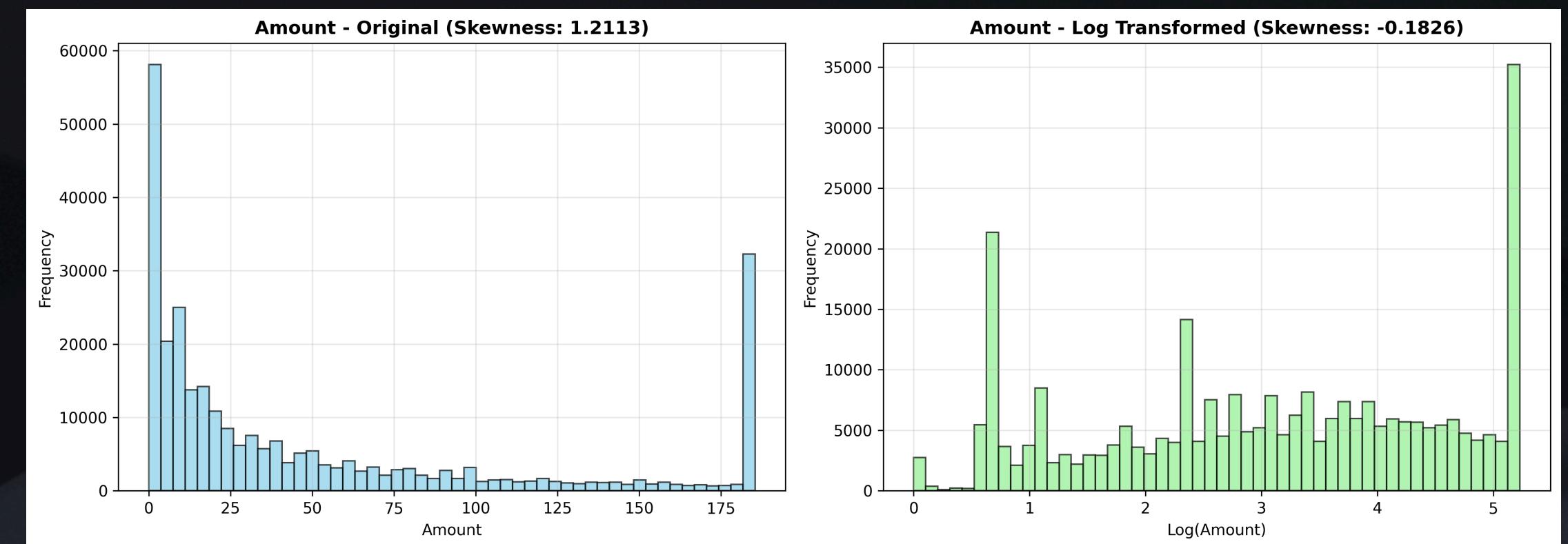
**Total Samples: 284,807**

# Pre-processing e feature engineering

1. Individuazione e rimozione dei duplicati
2. Gestione degli outliers tramite IQR
3. Log transformation su amount
4. Splitting dataset secondo la regola di Pareto
5. Bilanciamento delle classi tramite SMOTE



Distribuzione feature "Amount" prima e dopo la gestione degli outliers



Distribuzione feature "Amount" prima e dopo la log transformation

# Modelli addestrati

1. Gaussian Naïve bayes, usato come baseline per il confronto, iperparametri di default.
2. Random Forest, il modello principale del progetto, hyperparameter tuning tramite K-Fold Cross Validation (K=5) con iperparametri:
  - ▶ n\_estimators: [50, **100**, 150]
  - ▶ max\_depth: [10, 15, **20**]
  - ▶ min\_sample\_split: [5, **10**, 15]
  - ▶ min\_sample\_leaf: [**2**, 5, 10]
  - ▶ class\_weight=['balanced']
  - ▶ random\_state=42;
  - ▶ n\_jobs=-1

# Performance ottenute

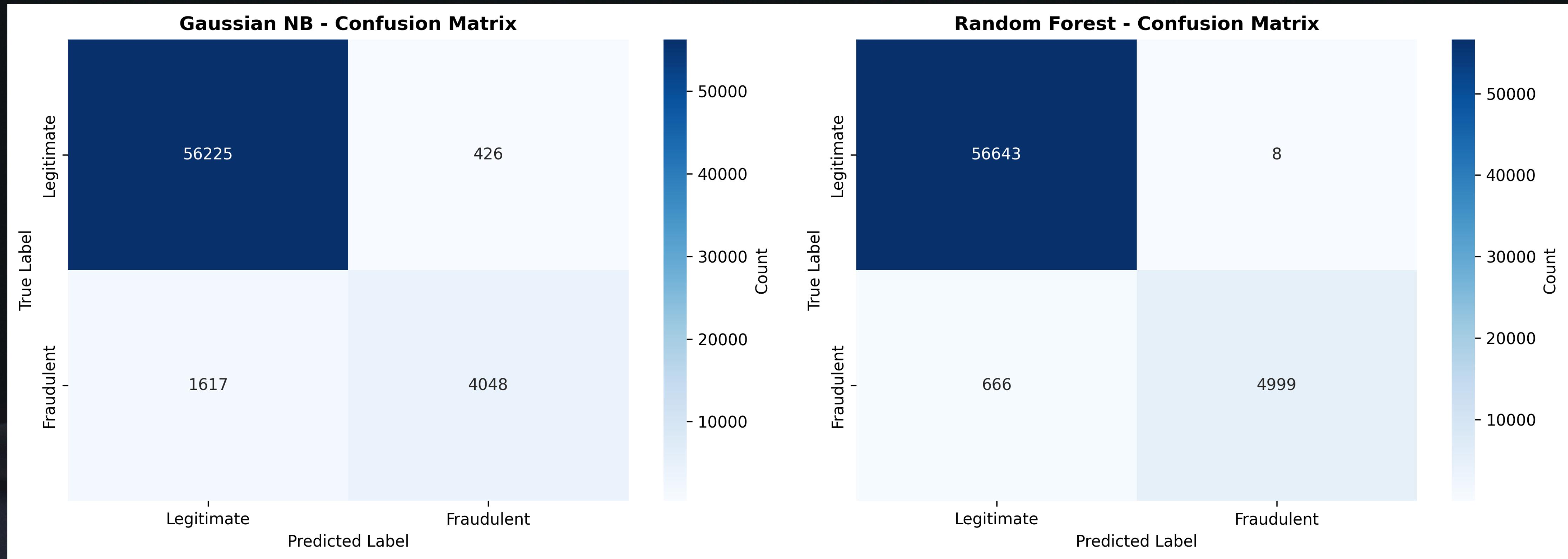
## Gaussian Naïve Bayes

- Accuracy: 96.72%
- Precision: 90.48%
- Recall: 71.46%
- F1-Score: 79.85
- ROC-AUC: 96.96%

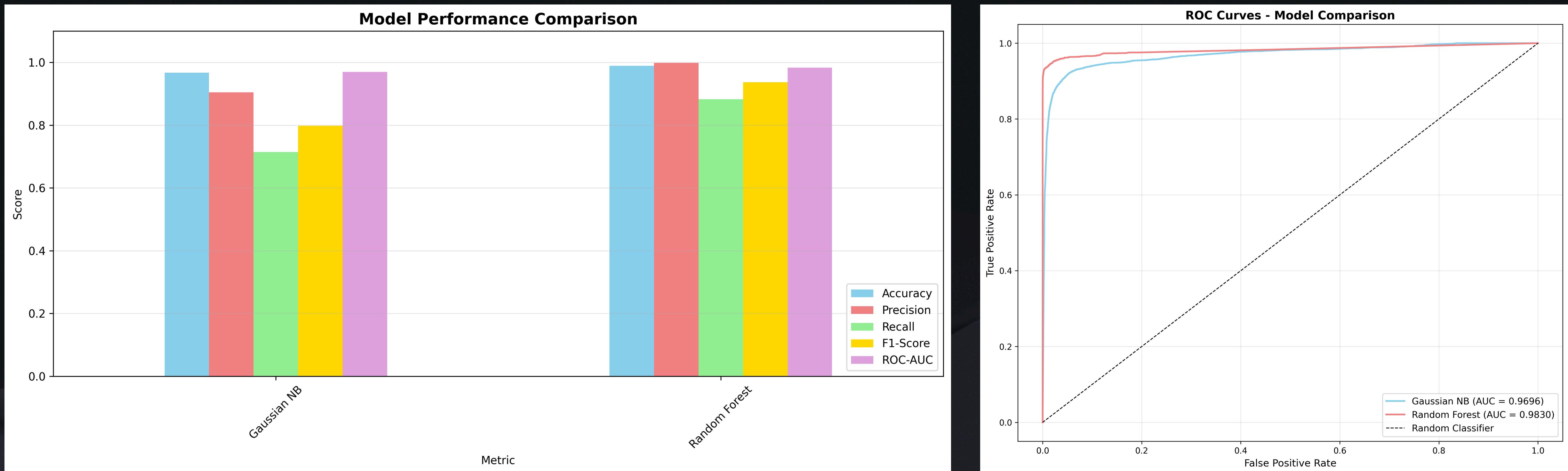
## Random Forest [Best parameters]

- Accuracy: 98.92%
- Precision: 99.84%
- Recall: 88.24%
- F1-Score: 93.68%
- ROC-AUC: 98.30%

# Performance ottenute



# Performance ottenute



# Considerazioni finali

Valutando le prestazioni, si denota come un'ensemble aiuti il sistema ad effettuare predizioni più accurate rispetto ad un singolo classificatore, sarebbe possibile ottenere performance di classificazione migliori con tecniche di boosting per migliorare le performance, come XGBoost.

Un'altra possibilità sono soluzioni algoritmiche ad-hoc che sfruttano la feature “time” per verificare l'orario di determinate transazioni verso lo stesso mittente o una serie di transazioni a cadenza periodica.

```
    def __init__(self):
        self.gpuInfo = GPUInfo()
        self.gpu = self.gpuInfo.get_gpu(0)
        self.load = int(self.gpu.query_load())
        self.gpu_clock = int(round(self.gpu.query_clock()))
        self.gpu_memory_usage = round(self.gpu.query_memory_usage())
        self.gpu_gtt_usage = round(self.gpu.query_gtt_usage())
        self.power = self.gpu.query_power()
        self.voltage = round(self.gpu.query_voltage())
        self.fans = self.sensors_fans()
        self.fan_items = self.fans.items()

    def sensors_fans(self):
        fans = {}
        for name, value in self.fans.items():
            fans[name] = float(value['value'])

        return fans
```

Grazie per l'attenzione!