Data Analytics

Fraud Detection in Credit Card Transactions

Aranzazu Puig Turrion

February 2025

Table of content

1. Introduction

1.1 Objective

Credit card fraud has become an increasing problem due to the growing reliance on electronic payment systems and technological advances.

This challenge affects stakeholders globally, from consumers and businesses to financial institutions. As the volume of digital transactions increases, so does the complexity of fraudulent tactics, necessitating the use of sophisticated analytical tools to detect and prevent such activities.

Data analysis is crucial in this context, as it allows the examination of large datasets to identify patterns and anomalies that may indicate fraud. By using the power of data analytics, it's possible to enhance the effectiveness of fraud detection systems, reduce financial losses, and bolster the security of credit card transactions.

This project explores the nature of credit card fraud and aims to demonstrate how using data-driven methods can effectively address this escalating problem.

1.2 Overview of the payment fraud reported by industry across the European Economic Area (EEA)

The following conclusions and insights are drawn from a report jointly prepared by the European Banking Authority (EBA) and the European Central Bank (ECB) published the 1st August 2024 by EBA[1].

It covers semi-annual data reported for the three reference periods H1 2022, H2 2022 and H1 2023, and focuses on the payment instrument of credit transfers, direct debits, card payments (from an EU/EEA issuing perspective), cash withdrawals and e-money transactions. The data covers all EU/EEA countries.

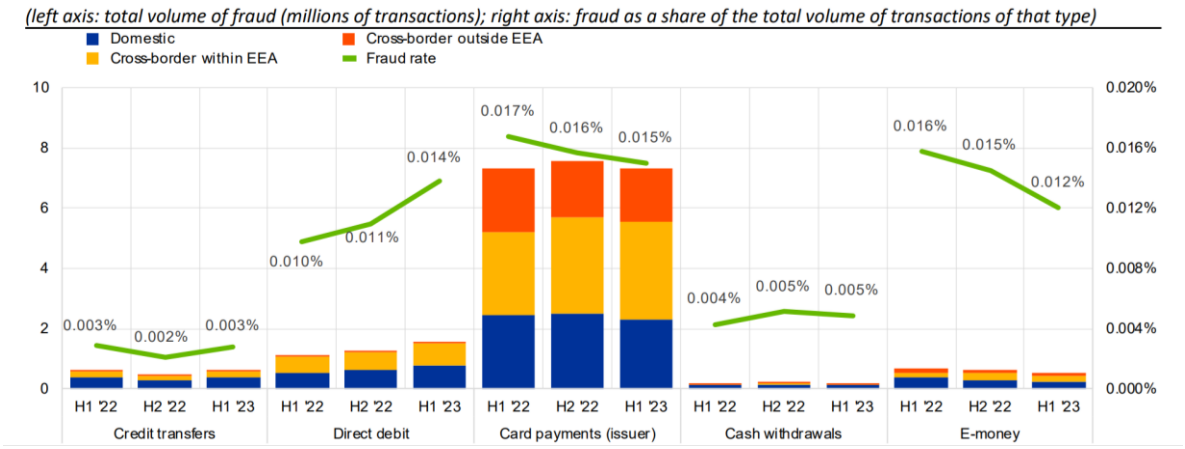
1.2.1 Rates of fraudulent payments

The **total value of fraudulent transactions** reported by the industry across the **European Economic Area (EEA)** amounted to **EUR 4.3 billion in 2022 and EUR 2.0 billion in the first half of 2023**. This considers the sum of all fraudulent transactions reported for credit transfers, direct debits, card payments, cash withdrawals and e-money transactions.

[1] https://www.eba.europa.eu/publications-and-media/press-releases/eba-and-ecb-release-joint-report-payment-fraud

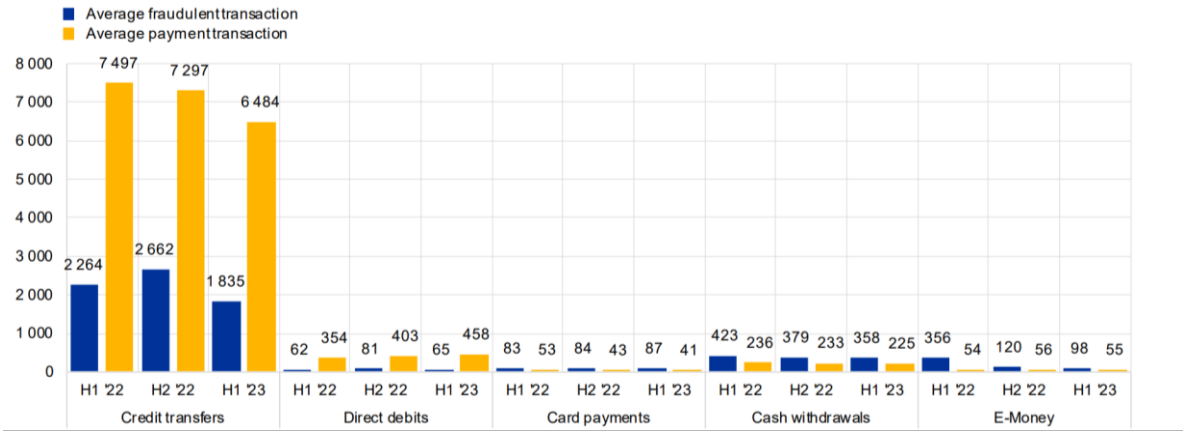
The highest fraud values between H1 2022 and H1 2023 were reported in credit transfers and card payments. If focusing on volumes, card payment fraud accounted for by far the largest number of fraudulent transactions between H1 2022 and H1 2023.

Absolute and relative levels of fraud by type of payment



Focusing on values, the average value of a fraudulent transaction appeared highest for credit transfers compared with other payment instruments.

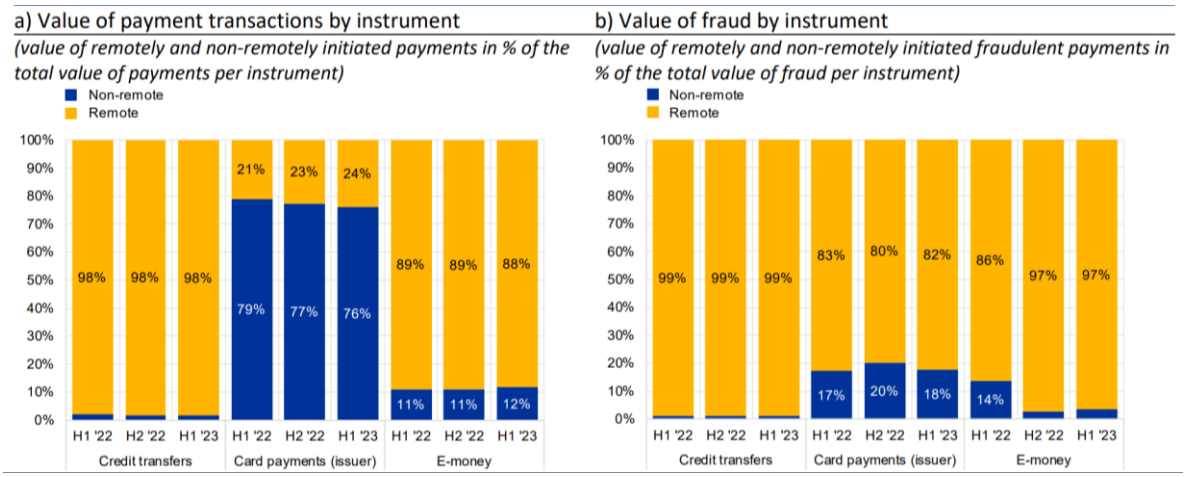
Average value of a transaction and a fraudulent transaction by payment method (average value in EUR)



1.2.2 Fraud types: quick overview

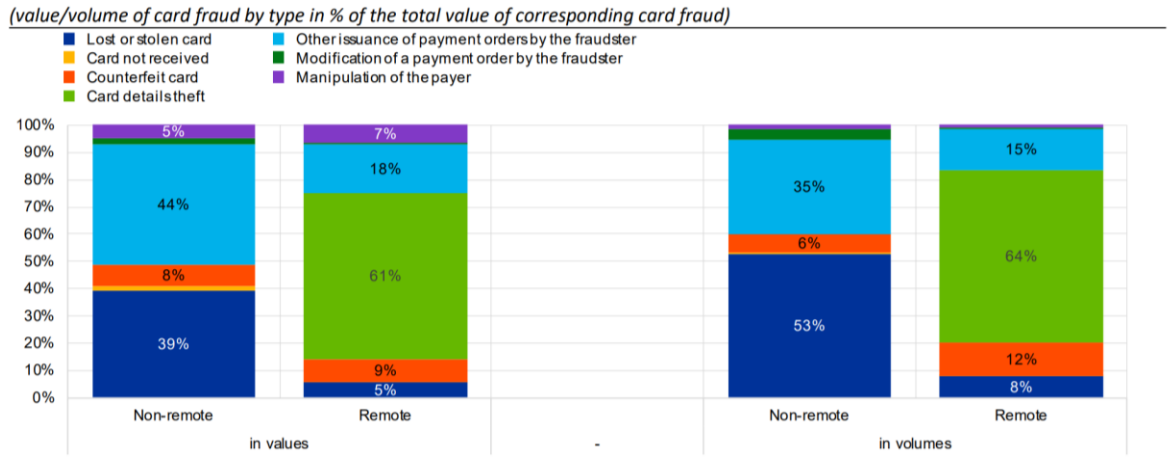
Fraud distinguishing between **non-remotely initiated transactions** (at the point of sale or terminal) and **remote transactions** (via the internet or electronic devices used for distance communication). **Remote versus non-remote transactions and fraud:**

Value shares of non-remotely versus remotely initiated payment transactions and fraud



Additionally, many non-remote card payment frauds are committed using cards that have been lost or stolen. Lost or stolen cards accounted for 39% of the value of non-remote card fraud in H1 2023 and for 53% of the volume of fraudulent, non-remote card payments.

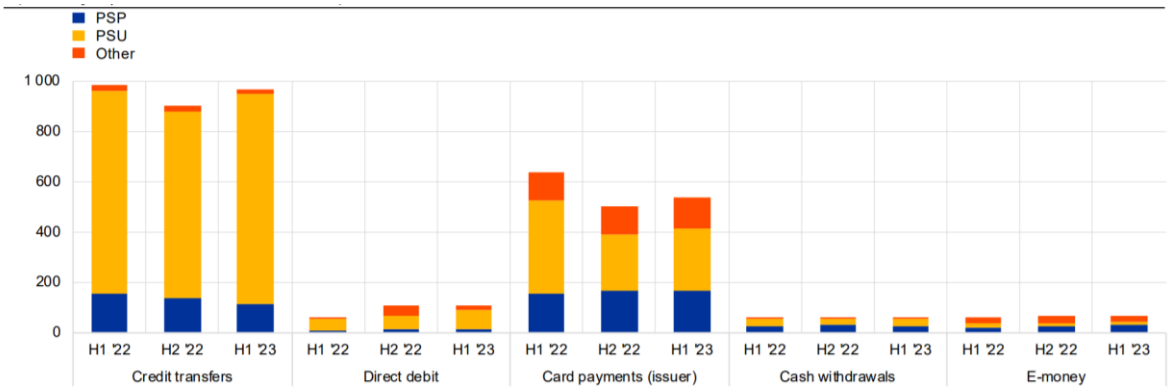
Composition of the value and volume of card fraud by initiation channel and fraud type (value/volume of card fraud by type in % of the total value of corresponding card fraud)



1.2.3 Losses due to credit card fraud

Final fraud losses are reported by PSPs[2] for the period in which they were recorded in the PSP's books, which may be disassociated timewise from the period in which the actual fraudulent transactions took place. Same applies to PSU[3].

Total value of reported losses due to fraud (value of reported losses in million EUR)



Report Highlights[4]:

→ Total value of fraud across main payment instruments at **€4.3 billion for 2022** and **€2.0 billion for first half of 2023**.

→ **Report confirms effectiveness of strong customer authentication requirements**[5], for protecting against card fraud.

→ Card fraud risk **lower for transactions within the European Economic Area**, owing to mandatory application of strong customer authentication.

[2] PSP: Payment Service Provider who provides an online service for accepting electronic payments to businesses, merchants and utility companies, amongst others. These payments can be through a number of methods e.g. credit cards, direct debits, real-time bank transfers, cash payments, wallets and prepaid cards.

[3] PSU: Payment Service User is a person or business that uses a payment service (including AISPs, PISPs, CBPIIs and ASPSPs) to view, send or receive money

[4] These conclusions were extracted through web scraping from an article published by the ECB, with more details provided in a subsequent chapter.

[5] SCA (Strong Customer Authentication), which is a legal requirement for electronic payments and credit cards since 14 September 2019 in the EU (banks will reject all payments that don't fulfill the requirements stipulated in the law)

2. Data Sources and Data Collection

2.1 Flat files

Two souces have been used to get flat files. Kaggle and Github.

2.1.1 Kaggle

For this project, I have searched for a dataset containing the necessary elements to accurately simulate an environment aligned with my project objectives. I chose a dataset that includes emails and IP addresses for analytical purposes. Finding suitable data has proven challenging; I ultimately chose to use a dataset extracted from Kaggle. Data is available at:

https://www.kaggle.com/datasets/aryanrastogi7767/ecommerce-fraud-data/data

There is no documentation about the datasets by the author in Kaggle but column names are quite intuitive. The transactions are identified as fraudulent or not through the 'Fraud' column, which contains a Boolean value.

This dataset appears to have been specifically crafted to support training in Data Analysis, Feature Engineering, and Machine Learning. The dataset consists of 2 csv files both containing information on ecommerce transactions made by customers. The two CSV files were merged using an inner joint on a common column based on email addresses for analytical purposes: the resulting dataset is quite small.

```
df_unido.shape
```

```
(844, 20)
```

2.1.2 Github

Furthermore, I have searched for a dataset of domain emails categorized as spam or fraudulent. I have found this data set in a repository in Github: a list that contains email domains often used to spam or abuse some services. File is available at:

https://github.com/disposable-email-domains/disposable-email-domains/blob/main/disposable_email_blocklist.conf

Shape:

```
file = r'C:\Users\Lenovo\Desktop\Ironhack\FINAL PROJECT\CSVs_and EXCEL_docs\disposable_email_blocklist.conf'
number_of_rows = 0

with open(file, 'r') as f:
    for line in f:
        number_of_rows += 1

print(f"File has {number_of_rows} rows.")
```

```
File has 3993 rows.
```

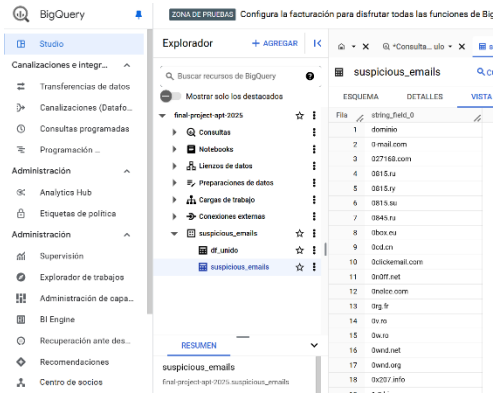
2.2 Big query

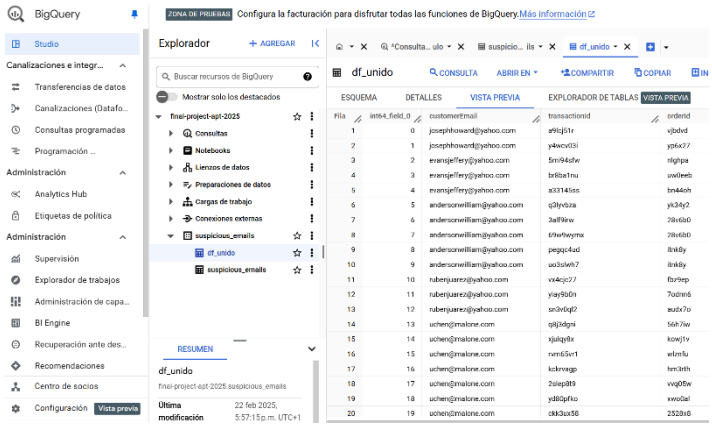
For the Big Data System requirement for this project I have used BigQuery from Google.

Ironhack provided me with access to BigQuery via a shared account. I simply had to log in with my Gmail address to start exploring. Since this account was intended for testing, I relied on public datasets to assess the features of BigQuery. However, I was unable to locate any publicly available datasets that were pertinent to my project's topic.

While preparing my data for BigQuery analysis, I merged two different CSV files from Kaggle and cleaned them up. This was done to denormalize the data, making it ready for an initial analysis in BigQuery.

I have loaded the merged version of the 2 data sets together with the data set of domain emails scoped as spam or fraudulent from Github.





Furthermore, I took advantage of having the data from both datasets in BigQuery to do an initial analysis, cross-referencing the suspicious emails extracted from GitHub with those in my final dataset to see if there were any matches. No matches were found:

2.3

Consulta sin título — EJECUTAR — Completada

```
SELECT
    T.*,
    S.string_field_0 AS suspicious_domain
FROM
    (SELECT *,
            SPLIT(customerEmail, '@')[OFFSET(1)] AS email_domain
    FROM `final-project-apt-2025.suspicious_emails.df_unido`) T
JOIN
    `final-project-apt-2025.suspicious_emails.suspicious_emails` S
ON
    T.email_domain = S.string_field
```

For this project, web scraping was employed to extract conclusions from the fraud report available on the ECB website. These findings are presented in the introduction section of this report[6]:

https://www.ecb.europa.eu/press/pr/date/2024/html/ecb.pr240801~f21cc4a009.en.html

The methodology involved using BeautifulSoup in Python:

```python
import requests
from bs4 import BeautifulSoup
import re
from docx import Document

url = 'https://www.ecb.europa.eu/press/pr/date/2024/html/ecb.pr240801~f21cc4a009.en.html'

response = requests.get(url)

if response.status_code == 200:
    soup = BeautifulSoup(response.text, 'html.parser')

document = Document()
li_tag1 = soup.find_all('li', text=re.compile('Total value of fraud across'))
li_tag2 = soup.find_all('li', text=re.compile('Report confirms effectiveness'))
li_tag3 = soup.find_all('li', text=re.compile('Card fraud risk lower'))

all_li_tags = [li_tag1, li_tag2, li_tag3]
for group in all_li_tags:
    for li in group:
        print(li.text)
        document.add_paragraph(li.text)

new_paragraph = soup.find('p', text=re.compile('The European Central Bank \(ECB\) and the European Banking Authority \(EBA\) today published a'))
if new_paragraph:
    document.add_paragraph(new_paragraph.text)
document.save(r'C:\Users\Lenovo\Desktop\Ironhack\FINAL PROJECT\Web_Scraping\Web_Scraping_text.docx')
```

3. Data Cleaning

The initial dataset obtained in Kaggle was already in a relatively clean state, which streamlined the process of preparing it for database storage. However, a meticulous approach was adopted to ensure the data was optimally formatted for efficient analysis, this involved several key steps:

- **Handling Null Values:** a minimal number of null values were discovered, and upon closer examination, it was determined that these correspond to clients without orders. The null values were not removed because retaining the details of customers without orders was deemed valuable for the analysis of fraud patterns.

[6] Page 6, footnote 4.

```
missing_values = df_unido.isnull().sum()
missing_values
```

```
customerEmail                        0
transactionId                       25
orderId                             25
paymentMethodId                     25
paymentMethodRegistrationFailure    25
paymentMethodType                   25
paymentMethodProvider               25
transactionAmount                   25
transactionFailed                   25
orderState                          25
customerPhone                        0
customerDevice                       0
customerIPAddress                    0
customerBillingAddress               0
No_Transactions                      0
No_Orders                            0
No_Payments                          0
Fraud                                0
```

- **Removal of Unnecessary Columns:** columns that were irrelevant to the project's purpose have been deleted. Columns created by Pandas during merging process "Unnamed: 0_x" and "Unnamed: 0_y" that represent the index from the original CSV files have been dropped.

```
df_unido.drop('Unnamed: 0_x', axis=1, inplace=True)
```

```
df_unido.drop('Unnamed: 0_y', axis=1, inplace=True)
```

After dropping the columns, the shape of the dataframe is as follows:

```
df_unido.shape
```

```
(844, 18)
```

- **Data Type Assignment:** Correct data types were assigned to each column to ensure data integrity and optimize storage efficiency. This step is crucial as it directly impacts the database's performance and the accuracy of data retrieval processes.

```
print(df_unido.dtypes)
```

```
customerEmail                        object
transactionId                        object
orderId                              object
paymentMethodId                      object
paymentMethodRegistrationFailure    float64
paymentMethodType                    object
paymentMethodProvider                object
transactionAmount                   float64
transactionFailed                   float64
orderState                           object
customerPhone                        object
customerDevice                       object
customerIPAddress                    object
customerBillingAddress               object
No_Transactions                       int64
No_Orders                             int64
No_Payments                           int64
Fraud                                  bool
```

For the Github dataset no data cleaning was needed as it is an unique column with no nulls and has the correct data type.

2.4 Exposing Data via API

To improve access to a dataset located in a MySQL database, a Flask-based API was created. Flask, known for its simplicity and ability to be extended, is a micro web framework in Python ideal for constructing RESTful APIs. These APIs operate on a stateless, client-server model where each inquiry by a client includes all necessary information to fulfill the request. This API acts as a conduit between the database and users looking to access data.

```python
from flask import Flask, jsonify, request
import pymysql

#Initialize Flask app
app = Flask(__name__)


def get_data_base_connection():
    try:
        connection = pymysql.connect(user = "root",
                                     password= "[redacted]",
                                     database= "final_project",
                                     cursorclass= pymysql.cursors.DictCursor) #Saying that data will
        return connection                #get shown as a dictionary to the user

    except Exception as e:
        print(f'Error Connecting to database: {e}')
        return None

get_data_base_connection()

@app.route("/final_project", methods = ["GET"])
def get_final_project():
    conn = get_data_base_connection()
    if not conn: # if I don't have connection defined
        return jsonify ({"error":"Failed to connect to database"}), 500

    with conn.cursor() as cursor:
        query= """SELECT customerEmail, transactionAmount FROM final_project;"""
        cursor.execute(query)
        final_project= cursor.fetchall()

    conn.close() #Closing connection

    return jsonify (final_project)



if __name__ == "__main__":
    app.run(debug=True)
```

The endpoint of the API I above is final_project.

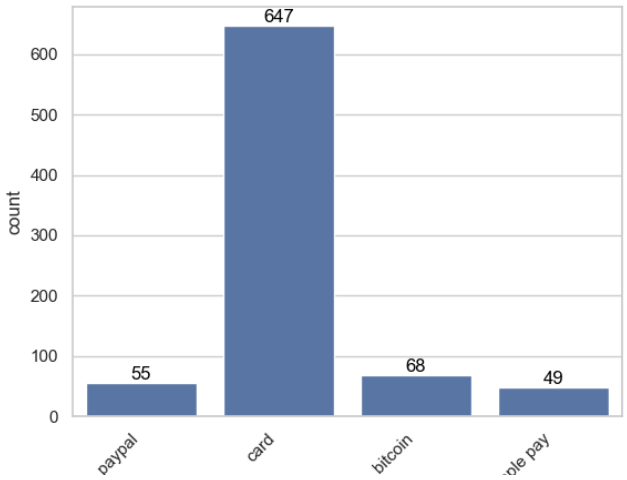
In the example above, we have created an endpoint that fetches each customer's email address along with their transaction amount from the database. Here's how the data is presented as a result:

4. Exploratory data analysis (EDA) and visualization

Exploratory Data Analysis (EDA) has been utilized as an initial method to identify preliminary fraud patterns as well as to understand consumer behavior. This preliminary analysis serves as a foundation for conducting more detailed and in-depth investigations at later stages. By meticulously breaking down and examining the available data, the goal is to gain an initial clear insight that might indicate possible irregularities or trends in consumer actions. This analysis not only highlights specific areas of interest that require more thorough exploration but also sets the stage for developing effective fraud prevention strategies.

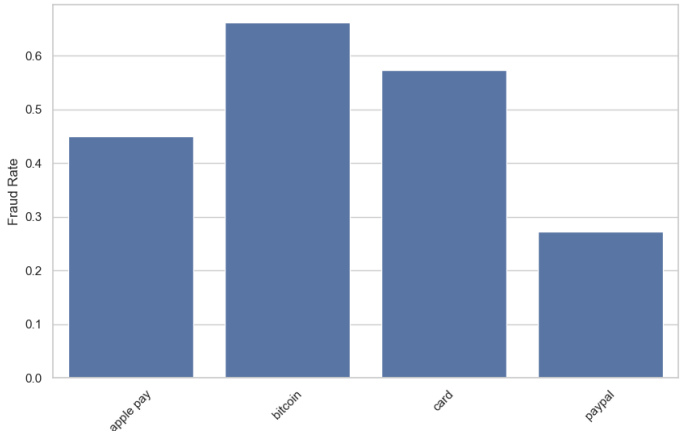
Proportion of payment Method

Credit card transactions, as shown below, are the most frequently used payment method:



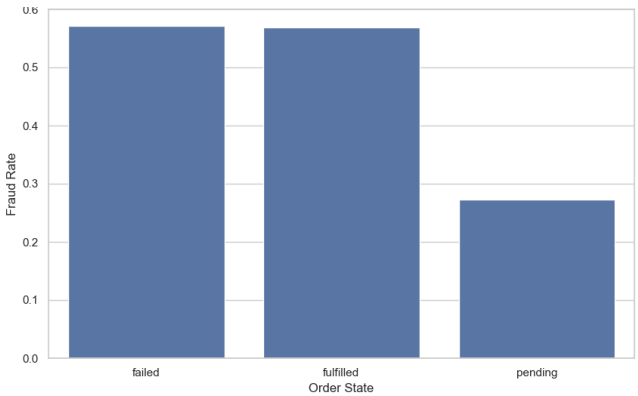
Fraud Rate by Payment method

Bitcoin has the highest level of fraud when measured in absolute terms:



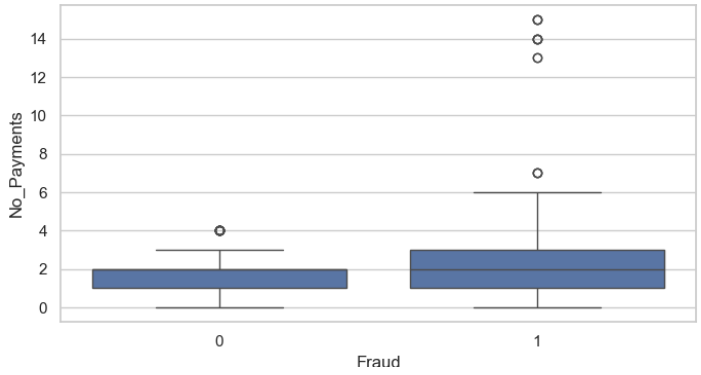
Fraud rate by order state

The state of failed orders has the highest number of transactions identified as fraudulent:



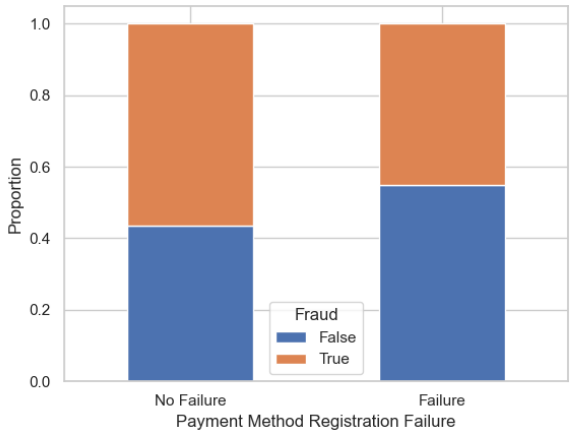
Distribution of fraudulent transactions

Fraud occurs more frequently as the number of payments increases:



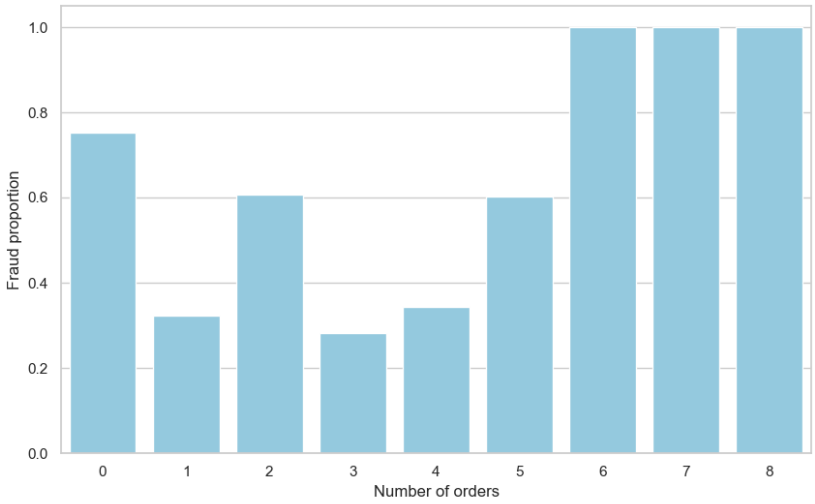
Fraud Proportion by payment Method registration Failure

The proportion of fraudulent transactions increases with successful payments:



Relation between number of orders & fraud

The greater the number of orders under the same order ID, the higher the risk of fraud:



This graph indicates that orders exceeding five are flagged as fraudulent, I checked that this observation is not based in isolated data points:

```
counts_ordered = df_unido['No_Orders'].value_counts().sort_index(ascending=False)
print(counts_ordered)
```

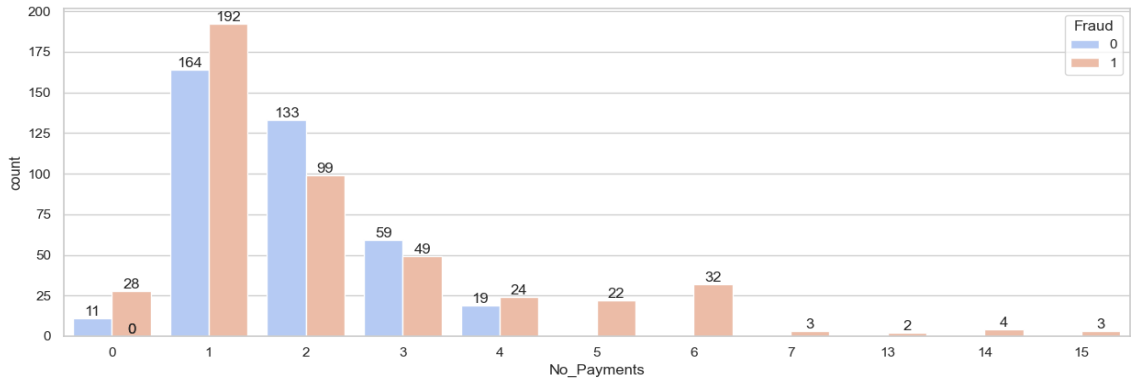
```
No_Orders
8     44
7      7
6     43
5    211
4    201
3    121
2    102
1     34
0     81
```

```
are_all_fraudulent = high_order_frauds['Fraud'].all()
print("¿Orders > than 5 are all flagged as fraudulent ?:", are_all_fraudulent)
```

```
¿Orders > than 5 are all flagged as fraudulent ?: True
```

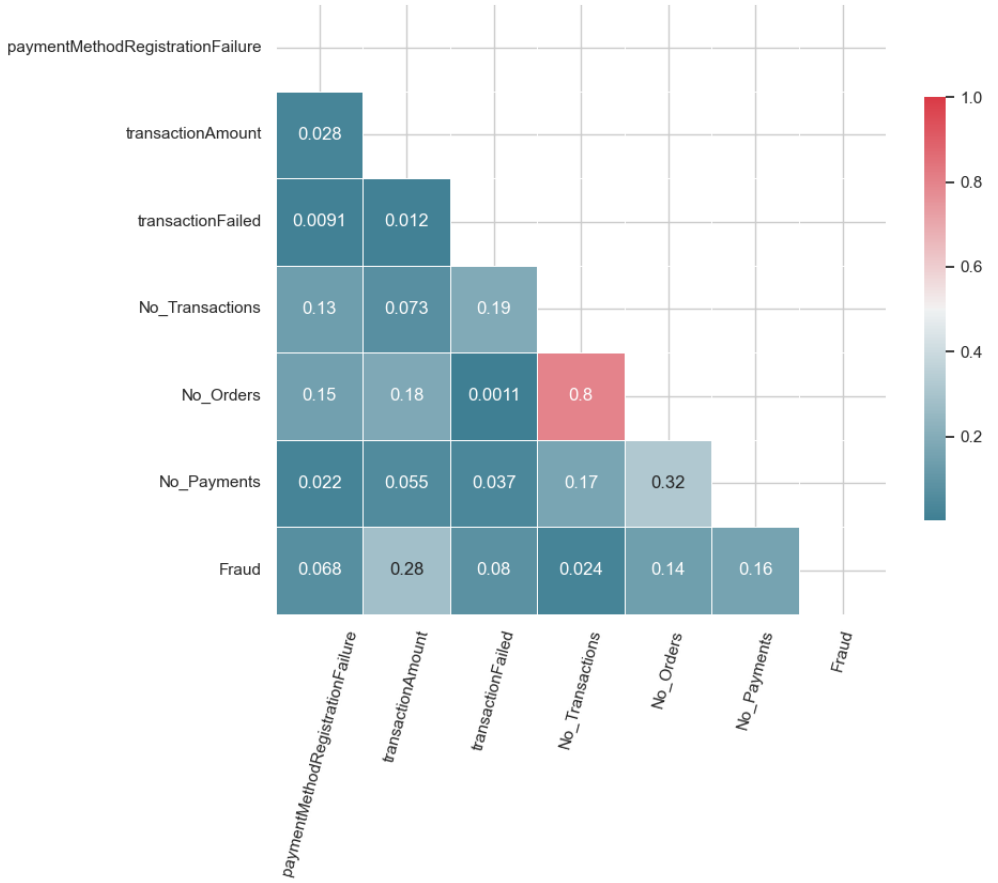
Total number of payments per fraudulent transactions

As the number of payments increases, more fraudulent transactions are detected:



Correlation among features

The most significant correlations related to the target "fraud" are observed with transaction amounts, number of orders, and number of payments. However, these correlations are relatively weak due to the small dataset.



5. Inferential Analysis

In our project, we leveraged the Chi-square test to examine the relationship between various categorical variables and their potential linkage to fraudulent activities. Given the multifaceted nature of fraud detection, understanding these relational dynamics is crucial. The test's utility was unfolded in two primary dimensions:

- Identification of Significant Variables: By applying the Chi-square test across a range of variables such as transaction methods, order states, and payment failures, we could pinpoint which factors carry a statistically significant relation with the occurrence of fraud. This step was instrumental in narrowing down the variables that would prove pivotal in subsequent analytical models focused on fraud prediction.

- Refinement of Feature Selection: The insights garnered from Chi-square testing were invaluable in refining our feature selection for machine learning models. By understanding which features bore a significant relationship with fraud, we were

better positioned to develop a more focused and effective analytical framework, enhancing the predictive accuracy of our models.

I based the Chi-square test on the 3 strongest correlations with the target "fraud" feature.

→ Transaction Amount:

H0: There is no relationship between the 'Fraud' variable and the 'transactionAmount' values (they are independent).

H1: There is a relationship between the 'Fraud' variable and the 'transactionAmount' values (they are not independent).

Results: Chi2 Statistic: **104.085/** p-value: **0.00194**

There is a statistically **significant association** between the transaction amount and whether it is fraudulent or not.

→ No Orders:

H0: There is no relationship between the 'Fraud' variable and the 'No Orders' (they are independent).

H1: There is a relationship between the 'Fraud' variable and the 'No Orders' values (they are not independent).

Chi2 Statistic: 170.556/p-value: 9.858 e-33

There is a statistically **significant association** between the N of orders amount and whether it is fraudulent or not.

→ No Payments:

H0: There is no relationship between the 'Fraud' variable and the 'No of Payments' (they are independent).

H1: There is a relationship between the 'Fraud' variable and the 'No of Payments' values (they are not independent).

Chi2 Statistic: 76.517/ p-value: 2.408 e-12

There is a statistically **significant association** between the N of payments amount and whether it is fraudulent or not.

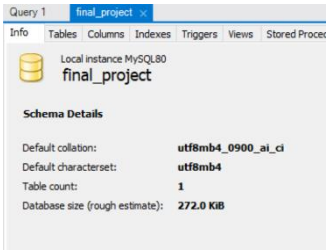
A larger dataset could potentially yield more accurate results.

6. SQL queries for patterns analysis

SQL stands out as a powerful tool for unveiling patterns hidden within datasets. Through the application of SQL, I embarked on a journey to discover these hidden narratives by executing targeted queries that allowed filtration, manipulation, and analysis of the dataset.

By crafting precise SQL queries, I was able to identify significant patterns that may indicate key trends or potential anomalies, such as irregular transaction behaviors hinting at possible fraudulent activities. This process involved analyzing transaction amounts, identifying repeated failures in payment processes, and scrutinizing the use of multiple payment methods associated with single customers.

First I created the database in MySQL Workbench to store the table:



Further, use of SQL has enabled me to compare my analysis with the results identified as fraud in the dataset. Extract from the queries:

- **<u>Identifying different email addresses for same IP</u>**

```
7    -- Identifying different email addresses for same IP
8    SELECT t.customerIPAddress, t.customerEmail
9    FROM final_project t
10   WHERE t.customerIPAddress IN (
11       SELECT customerIPAddress
12       FROM final_project
13       GROUP BY customerIPAddress
14       HAVING COUNT(DISTINCT customerEmail) > 1
15   )
```

Result Grid | Filter Rows: | Export: | Wrap Cell Conte

customerIPAddress	customerEmail
45.203.99.249	christinemills@mcgee.com
45.203.99.249	christinemills@mcgee.com
45.203.99.249	jamescampbell@randall-pacheco.biz
45.203.99.249	jamescampbell@randall-pacheco.biz
45.203.99.249	jamescampbell@randall-pacheco.biz

- **Multiple payment methods same email account**

```
2    -- Multiple paiment methods
3 •  SELECT customerEmail, COUNT(DISTINCT paymentMethodId) AS num_payment_methods
4    FROM final_project
5    GROUP BY customerEmail
6    ORDER BY num_payment_methods DESC;
7
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

customerEmail	num_payment_methods
johnlowery@gmail.com	9
ctaylor@yahoo.com	5
david45@gmail.com	5
uchen@malone.com	5
catherine64@gmail.com	4
ethompson@jackson-sanders.com	4
gwilcox@hotmail.com	4
ksummers@hotmail.com	4

- **Payment method registration failures by client**

```
2    -- Count Payment Method Registration Failures by Customer
3 •  SELECT customerEmail, SUM(paymentMethodRegistrationFailure) AS num_registration_failures
4    FROM final_project
5    GROUP BY customerEmail
6    ORDER BY num_registration_failures DESC;
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

customerEmail	num_registration_failures
tmcpherson@wright.com	13
amywright@wallace-johnson.com	6
michelleherrera@day.info	6
kyle64@stephens-ortiz.com	5
martinezlori@gmail.com	5
ssmith@levine-harmon.biz	5
davismike@hotmail.com	4
gwilcox@hotmail.com	4

- **SUM number failed transactions by client**

```
2    -- Failed transactions
3 •  SELECT customerEmail, SUM(transactionFailed) AS num_failed_transactions
4    FROM final_project
5    GROUP BY customerEmail
6    ORDER BY num_failed_transactions DESC;
```

esult Grid | Filter Rows: | Export: | Wrap Cell Content:

customerEmail	num_failed_transactions
johnlowery@gmail.com	48
mitchellgriffith@yahoo.com	12
tmcpherson@wright.com	6
1yf0@jedyz63t	5
brandon58@conner.com	5
finleybrianna@yahoo.com	4
jamescampbell@randall-pacheco.biz	4
ssmith@levine-harmon.biz	4

- **High value transactions order DESC**

```
2    -- High value transactions
3 •  SELECT customerEmail, transactionId, transactionAmount
4    FROM final_project
5    ORDER BY transactionAmount DESC
6    LIMIT 50;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch

customerEmail	transactionId	transactionAmount
uguzman@yahoo.com	ftiso5mr	353
lleonard@turner-fleming.com	8zufs56k	75
catherine64@gmail.com	8810813t	75
johnlowery@gmail.com	u2xbyl92	74
johnlowery@gmail.com	u2xbyl92	74
johnlowery@gmail.com	u2xbyl92	74
johnlowery@gmail.com	u2xbyl92	74
ugood@mosley.info	gpeuthlg	74

- **Checking small amount purchases same client**

Limit to 50000 rows

```
1 •  USE final_project;
2    -- Checking small amount purchases same client
3 •  CREATE TEMPORARY TABLE SmallPurchases AS
4    SELECT customerEmail, transactionAmount
5    FROM final_project
6    WHERE transactionAmount > 0 AND transactionAmount <= 15;
7
8 •  SELECT customerEmail, transactionAmount, COUNT(*) AS transactionCount
9    FROM SmallPurchases
10   GROUP BY customerEmail, transactionAmount
11   ORDER BY transactionCount DESC;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

customerEmail	transactionAmount	transactionCount
johnlowery@gmail.com	11	16
johnlowery@gmail.com	13	8
johnlowery@gmail.com	12	8
1yf0@jedyz63t	12	5
brandon58@conner.com	14	5
cindydeleon@yahoo.com	14	4
jamescampbell@randall-pacheco.biz	10	3

Using SQL, I succeeded in uncovering patterns indicative of fraudulent activities, enabling me to pinpoint and flag accounts exhibiting behaviors characteristic of fraud.

I have merged the results from the various SQL queries and organized the emails by their frequency of appearance in each query. Additionally, I have verified whether the emails identified as potentially fraudulent were tagged as fraud in the dataset. The coincidences are satisfactory:

Customer Email ★	Sources	Appearance Count	Is Fraudulent
johnlowery@gmail.com	Temp2, Temp3, Temp4, Temp5, Temp6	45	1
lleonard@turner-fleming.com	Temp2, Temp3, Temp4, Temp5, Temp5, Temp5, Temp5, Temp6, Temp6, Temp6, Temp6	11	1
uchen@malone.com	Temp2, Temp3, Temp4, Temp5, Temp5, Temp5, Temp5, Temp6, Temp6, Temp6, Temp6	11	1
ctaylor@yahoo.com	Temp2, Temp3, Temp4, Temp5, Temp5, Temp5, Temp6, Temp6, Temp6	9	1
amywright@wallace-johnson.com	Temp2, Temp3, Temp4, Temp5, Temp5, Temp6, Temp6	7	1
david45@gmail.com	Temp2, Temp3, Temp4, Temp5, Temp5, Temp6, Temp6	7	1
gwilcox@hotmail.com	Temp2, Temp3, Temp4, Temp5, Temp5, Temp6, Temp6	7	1
tmcpherson@wright.com	Temp2, Temp3, Temp4, Temp5, Temp5, Temp6, Temp6	7	1
ugood@mosley.info	Temp2, Temp3, Temp4, Temp5, Temp5, Temp6, Temp6	7	1
wdelacruz@yahoo.com	Temp2, Temp3, Temp4, Temp5, Temp5, Temp6, Temp6	7	1
whitedavid@jones-lloyd.org	Temp2, Temp3, Temp4, Temp5, Temp5, Temp6, Temp6	7	1
catherine64@gmail.com	Temp2, Temp3, Temp4, Temp5, Temp6	5	1
kristina41@gmail.com	Temp2, Temp3, Temp4, Temp5, Temp6	5	1
nancymayo@brown.com	Temp2, Temp3, Temp4, Temp5, Temp6	5	1
uguzman@yahoo.com	Temp2, Temp3, Temp4, Temp5, Temp6	5	1
christinemills@mcgee.com	Temp2, Temp3, Temp4	4	1
jamescampbell@randall-pacheco.biz	Temp2, Temp3, Temp4	4	1
shelby24@hotmail.com	Temp2, Temp3, Temp4	4	1
warrenedward@arnold.com	Temp2, Temp3, Temp4	4	1
1yf0@jedyz63t	Temp2, Temp3, Temp4	3	1
3fooiar@6eph	Temp2, Temp3, Temp4	3	1
9es7t@u6n7x	Temp2, Temp3, Temp4	3	1
abigail08@yahoo.com	Temp2, Temp3, Temp4	3	0
alec27@bell.com	Temp2, Temp3, Temp4	3	0
aliciaanthony@martin.com	Temp2, Temp3, Temp4	3	1
andersonwilliam@yahoo.com	Temp2, Temp3, Temp4	3	0
andre74@patrick-decker.com	Temp2, Temp3, Temp4	3	1
anthony04@gmail.com	Temp2, Temp3, Temp4	3	0
avaldez@gmail.com	Temp2, Temp3, Temp4	3	0

★ Only 30 first rows are presented

Sources

Temp 1 - Identifying different email addresses for same IP

Temp 2 - Multiple payment methods same email account

Temp 3 - Payment method registration failures by client

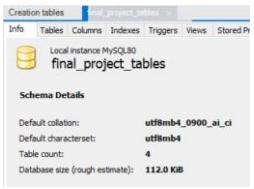
Temp 4 - SUM number failed transactions by client

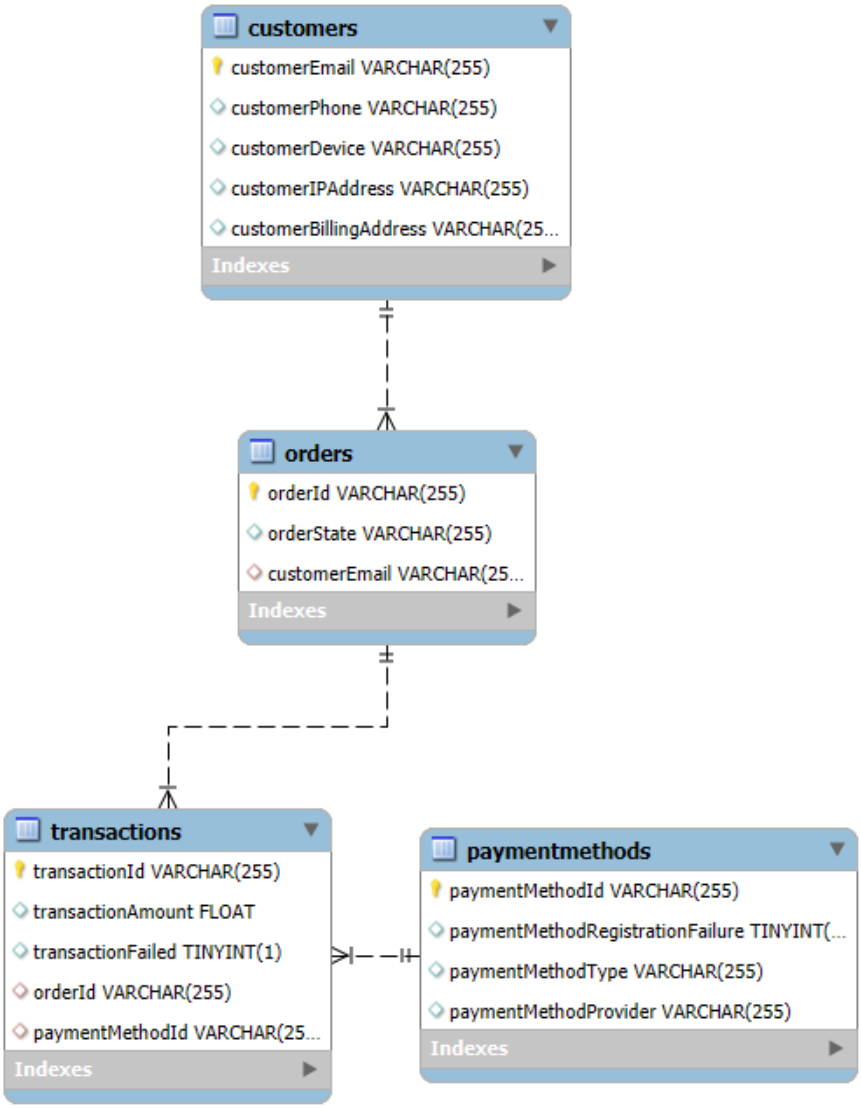
Temp 5 - High value transactions order DESC

Temp 6 - Checking small amount purchases same client

7. Entity Relationship Diagram (ERD)

Since I only had one table in my SQL database, I concurrently created another database in SQL to meet the requirement of creating the ERD.





8. Machine Learning

After assessing different machine learning models Logistic Regression has been selected for this project for different reasons:

- It is very effective at binary classification tasks, where the outcomes are categorical, such as determining whether a transaction is fraudulent or not. This model calculates probabilities using a logistic function, which is ideal for scenarios where you need to make clear decisions based on likelihood thresholds.
- It provides valuable insights into the importance of different predictors. This feature is crucial for understanding which variables, such as transaction value or location, most significantly impact the likelihood of fraud.
- It is a relatively simple model and fast to implement. It is a model highly scalable for handling large volumes of transactions, which is typical in the credit card industry. They also require less computational resources compared to more complex algorithms, ensuring efficiency in real-time fraud detection systems.

Below are the results:

Precision 75% for no fraud, 67% for fraud

Recall: real fraud cases 83%

F1-score (recall/precision)

Accuracy total 70%

	Precision	Recall	f1-score
Not fraud	0.75	0.56	0.64
Fraud	0.67	0.83	0.74

I have experimented with various combinations of features, and this appears to yield the best outcome. Given the context of a relatively small dataset, the model is performing reasonably well.

9. Conclusion

In this project, I have focused on analyzing a dataset related to electronic transactions to identify fraud patterns and investigate different methods that could be effective in understanding the mechanisms and helping to prevent fraud in e-commerce.

Data visualization techniques have provided an intuitive way to understand the complexity and trends within the dataset, allowing for quick identification of key areas of interest. Heatmaps, for example, have been particularly useful in illustrating the strength and direction of correlations between variables and the target 'fraud', facilitating the identification of those factors that have a greater impact on predicting fraudulent activities.

Moreover, through various analyses, different patterns have been identified and areas where fraud is proportionally higher have been pinpointed, especially through SQL queries. This analysis is crucial for establishing protocols for the prevention and detection of fraudulent behaviors *ante eventum*.

Furthermore, through Machine Learning (ML), I have aimed to create a model that enhances predictive analysis, boosting the system's ability to prevent fraud before it occurs.

In conclusion, this project underscores the importance of a multifaceted, data-driven approach for detecting fraud. The implementation of proactive, evidence-based strategies, supported by rigorous data analysis, is essential for addressing financial fraud manifested through payment methods in digitalized and constantly evolving actual environments.

10. GRPD

Under GDPR, data is categorized into two main types: personal data and non-personal data.

● Personal Data: information that can directly or indirectly identify a person. This includes: names, email addresses, location data, identification numbers, online identifiers.

● Non-Personal Data: information that cannot identify a person and relates to objects, places, or anonymized datasets.

Kaggle has its own rules and guidelines that users must follow, which are designed to comply with data protection laws like GDPR:

https://www.kaggle.com/terms