# End to End Data Science Project

December 7, 2023

Credit Card Fraud Detection 2023

## 1 1. Abstract

In this research, we aim to address the burgeoning issue of credit card fraud in the digital age. Using an extensive set of more than 550,000 credit card transactions made by European cardholders in 2023, we are concentrating on applying cutting-edge AI and machine learning methods to create an effective fraud detection system. The scope of our study includes a thorough examination of the literature, an in-depth investigation of key features in the dataset, the development of relevant research questions, an effective methodology, and an objective evaluation of many machine learning models.

## 2 2. Introduction

Credit card fraud, an increasingly formidable challenge in today's dynamic digital landscape, poses significant threats to both financial institutions and consumers alike. The growing number of electronic transactions has made it necessary to develop and apply creative solutions that can effectively block the constantly evolving tactics used by fraudsters. The significant financial consequences and imminent risk to customer trust are sufficient to highlight how serious this issue is. Deploying and establishing improved fraud detection methods that can effectively identify and reduce the dangers associated with illegal financial transactions is imperative considering these challenges.

## 3 3. Literature Review

The detection of credit card fraud has been a recurring issue in the field of financial security, and researchers have thoroughly investigated several approaches to address this dynamic problem. A thorough analysis of this field of literature indicates the intricacy of credit card theft and the demand for advanced detection technologies.

**Machine Learning Approaches:** Numerous studies have delved into the application of machine learning algorithms for fraud detection. Vaishnavi Nath Dornadula et al. (2020) emphasized the efficacy of machine learning in their research on credit card fraud detection. They explored diverse algorithms, highlighting the importance of algorithm selection in achieving high detection accuracy.

**Anomaly Detection Techniques:** Anomaly detection has emerged as a pivotal approach in identifying fraudulent transactions. Meenu et al. (2020) conducted research specifically on anomaly

detection in credit card transactions using machine learning. Their work shed light on the effectiveness of anomaly detection techniques in capturing irregular patterns indicative of fraud.

**Integration of Multiple Techniques:** The landscape of credit card fraud is dynamic, requiring a holistic approach. Research by Btoush et al. (2023) emphasizes the importance of integrating multiple techniques, such as machine learning algorithms, anomaly detection, and deep learning, to create robust and adaptive fraud detection systems.

**Gap Analysis:** While existing studies provide valuable insights, there is a recognized gap in the literature. The need for improved precision and adaptability in fraud detection systems is highlighted, aligning with the objectives of the current research.

In conclusion, the assessment of the literature highlights the variety of methods used in credit card fraud detection, from traditional machine learning algorithms to more advanced deep learning approaches. Developing efficient and flexible fraud detection systems that can handle the changing strategies used by fraudsters in the digital era requires integrating different methods.

# 4 4. Dataset Features

1 - The dataset includes credit card transactions in 2023 carried out by cardholders across Europe.

2 - It contains more than 550,000 records with anonymised transaction characteristics, including the time and location of the transaction as well as several features (V1 to V28).

3 - A binary label ("Class") indicating whether the transaction is fraudulent (1) or not (0) also has recorded, along with the transaction value.

# 5 5. Research Questions

Our exploration is guided by a set of pertinent research questions, steering the investigation towards practical solutions:

5.1 - What are the key features or indicators crucial for identifying fraudulent transactions?

5.2 - What are the potential implications for customer service and communication when a fraudulent transaction is detected?

5.3 - How can innovative approaches like deep learning and anomaly detection enhance the precision of fraud detection?

5.4 - What are the most common fraud-related transaction categories, and how can business tactics be modified to counteract these risks?

5.5 - How can organizations adapt to emerging fraud tactics over time to maintain the effectiveness of fraud detection models?

# 6 6. Methodology

## 6.1 6.1 Data Collection

Over 550,000 records have been collected, all of which conceal the details of credit card transactions made by European cardholders in 2023. The source of this invaluable dataset is attributed to Kaggle.

## 6.2 6.2 Data Exploration

We thoroughly reviewed the dataset, gathering important information and calculating thorough statistics with the help of the info() and describe() methods. We thoroughly investigated the column data types to ensure that they were compatible with machine learning models. The class distribution was presented visually using a countplot, which let us distinguish between authentic and fraudulent transactions. The dataset's integrity was thoroughly checked for missing values, providing an accurate starting point for further analysis and model building.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
df = pd.read_csv("creditcard_2023.csv")
df.head()
```

[2]:

| | id | V1 | V2 | V3 | V4 | V5 | V6 | V7 | \ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | -0.260648 | -0.469648 | 2.496266 | -0.083724 | 0.129681 | 0.732898 | 0.519014 | |
| 1 | 1 | 0.985100 | -0.356045 | 0.558056 | -0.429654 | 0.277140 | 0.428605 | 0.406466 | |
| 2 | 2 | -0.260272 | -0.949385 | 1.728538 | -0.457986 | 0.074062 | 1.419481 | 0.743511 | |
| 3 | 3 | -0.152152 | -0.508959 | 1.746840 | -1.090178 | 0.249486 | 1.143312 | 0.518269 | |
| 4 | 4 | -0.206820 | -0.165280 | 1.527053 | -0.448293 | 0.106125 | 0.530549 | 0.658849 | |

| | V8 | V9 | … | V21 | V22 | V23 | V24 | V25 | \ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | -0.130006 | 0.727159 | … | -0.110552 | 0.217606 | -0.134794 | 0.165959 | 0.126280 | |
| 1 | -0.133118 | 0.347452 | … | -0.194936 | -0.605761 | 0.079469 | -0.577395 | 0.190090 | |
| 2 | -0.095576 | -0.261297 | … | -0.005020 | 0.702906 | 0.945045 | -1.154666 | -0.605564 | |
| 3 | -0.065130 | -0.205698 | … | -0.146927 | -0.038212 | -0.214048 | -1.893131 | 1.003963 | |
| 4 | -0.212660 | 1.049921 | … | -0.106984 | 0.729727 | -0.161666 | 0.312561 | -0.414116 | |

| | V26 | V27 | V28 | Amount | Class |
|---|---|---|---|---|---|
| 0 | -0.434824 | -0.081230 | -0.151045 | 17982.10 | 0 |
| 1 | 0.296503 | -0.248052 | -0.064512 | 6531.37 | 0 |
| 2 | -0.312895 | -0.300258 | -0.244718 | 2513.54 | 0 |
| 3 | -0.515950 | -0.165316 | 0.048424 | 5384.44 | 0 |
| 4 | 1.071126 | 0.023712 | 0.419117 | 14278.97 | 0 |

[5 rows x 31 columns]

Checking the shape of the dataset.

[3]:
```
df.shape
```

[3]:
```
(568630, 31)
```

Display basic information about the dataset using .info()

**Output:** The database possesses 31 columns and 568,630 assets that reflect different credit card transaction information. These characteristics include the target variable ("Class"), which determines whether a transaction is fraudulent (Class=1) or not (Class=0), anonymized features (V1 to V28), and transaction amounts ("Amount"). There are no missing values in any of the columns, suggesting that the data is completely non-null. The dataset takes up around 134.5 MB of RAM. Two columns are of type int64, which is used for discrete integer values, while the remaining columns are of type float64, which represents numerical data. comprehending and becoming ready for additional analysis of the data requires having an in-depth knowledge of the dataset's structure and data types.

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 568630 entries, 0 to 568629
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   id      568630 non-null  int64
 1   V1      568630 non-null  float64
 2   V2      568630 non-null  float64
 3   V3      568630 non-null  float64
 4   V4      568630 non-null  float64
 5   V5      568630 non-null  float64
 6   V6      568630 non-null  float64
 7   V7      568630 non-null  float64
 8   V8      568630 non-null  float64
 9   V9      568630 non-null  float64
 10  V10     568630 non-null  float64
 11  V11     568630 non-null  float64
 12  V12     568630 non-null  float64
 13  V13     568630 non-null  float64
 14  V14     568630 non-null  float64
 15  V15     568630 non-null  float64
 16  V16     568630 non-null  float64
 17  V17     568630 non-null  float64
 18  V18     568630 non-null  float64
 19  V19     568630 non-null  float64
 20  V20     568630 non-null  float64
 21  V21     568630 non-null  float64
 22  V22     568630 non-null  float64
 23  V23     568630 non-null  float64
 24  V24     568630 non-null  float64
```

```
 25  V25     568630 non-null  float64
 26  V26     568630 non-null  float64
 27  V27     568630 non-null  float64
 28  V28     568630 non-null  float64
 29  Amount  568630 non-null  float64
 30  Class   568630 non-null  int64
dtypes: float64(29), int64(2)
memory usage: 134.5 MB
```

Using .describe() for Summary statistics

**Output:** The dataset's distribution and attributes are shown by the summary statistics, which show that features V1 to V28 have been standardized and concealed with a mean close to 0 and a standard deviation close to 1. The transaction amounts represented by the "Amount" feature have an estimated mean of 12,041.96 and a standard deviation of 6,919.64. With a mean of 0.5, the "Class" column, which differentiates fraudulent (Class=1) from non-fraudulent (Class=0) transactions, indicates a rather balanced distribution. An in-depth understanding of feature qualities is provided by additional statistics, such as minimum, maximum, and percentile values, which aid in gathering the data for further study.

[5]:
```
df.describe()
```

[5]:

| | id | V1 | V2 | V3 | V4 | \ |
|---|---|---|---|---|---|---|
| count | 568630.000000 | 5.686300e+05 | 5.686300e+05 | 5.686300e+05 | 5.686300e+05 | |
| mean | 284314.500000 | -5.638058e-17 | -1.323544e-16 | -3.518788e-17 | -2.879008e-17 | |
| std | 164149.486121 | 1.000001e+00 | 1.000001e+00 | 1.000001e+00 | 1.000001e+00 | |
| min | 0.000000 | -3.495584e+00 | -4.996657e+01 | -3.183760e+00 | -4.951222e+00 | |
| 25% | 142157.250000 | -5.652859e-01 | -4.866777e-01 | -6.492987e-01 | -6.560203e-01 | |
| 50% | 284314.500000 | -9.363846e-02 | -1.358939e-01 | 3.528579e-04 | -7.376152e-02 | |
| 75% | 426471.750000 | 8.326582e-01 | 3.435552e-01 | 6.285380e-01 | 7.070047e-01 | |
| max | 568629.000000 | 2.229046e+00 | 4.361865e+00 | 1.412583e+01 | 3.201536e+00 | |

| | V5 | V6 | V7 | V8 | V9 | \ |
|---|---|---|---|---|---|---|
| count | 5.686300e+05 | 5.686300e+05 | 5.686300e+05 | 5.686300e+05 | 5.686300e+05 | |
| mean | 7.197521e-18 | -3.838678e-17 | -3.198898e-17 | 2.069287e-17 | 9.116859e-17 | |
| std | 1.000001e+00 | 1.000001e+00 | 1.000001e+00 | 1.000001e+00 | 1.000001e+00 | |
| min | -9.952786e+00 | -2.111111e+01 | -4.351839e+00 | -1.075634e+01 | -3.751919e+00 | |
| 25% | -2.934955e-01 | -4.458712e-01 | -2.835329e-01 | -1.922572e-01 | -5.687446e-01 | |
| 50% | 8.108788e-02 | 7.871758e-02 | 2.333659e-01 | -1.145242e-01 | 9.252647e-02 | |
| 75% | 4.397368e-01 | 4.977881e-01 | 5.259548e-01 | 4.729905e-02 | 5.592621e-01 | |
| max | 4.271689e+01 | 2.616840e+01 | 2.178730e+02 | 5.958040e+00 | 2.027006e+01 | |

| | … | V21 | V22 | V23 | V24 | \ |
|---|---|---|---|---|---|---|
| count | … | 5.686300e+05 | 5.686300e+05 | 5.686300e+05 | 5.686300e+05 | |
| mean | … | 4.758361e-17 | 5.398140e-18 | 5.395017e-18 | -1.999311e-18 | |
| std | … | 1.000001e+00 | 1.000001e+00 | 1.000001e+00 | 1.000001e+00 | |
| min | … | -1.938252e+01 | -7.734798e+00 | -3.029545e+01 | -4.067968e+00 | |
| 25% | … | -1.664408e-01 | -4.904892e-01 | -2.376289e-01 | -6.515801e-01 | |

```
50%    … -3.743065e-02 -2.732881e-02 -5.968903e-02  1.590123e-02
75%    …  1.479787e-01  4.638817e-01  1.557153e-01  7.007374e-01
max    …  8.087080e+00  1.263251e+01  3.170763e+01  1.296564e+01

                V25           V26           V27           V28         Amount  \
count  5.686300e+05  5.686300e+05  5.686300e+05  5.686300e+05  568630.000000
mean  -3.028957e-17 -7.547400e-18 -3.598760e-17  2.499139e-17   12041.957635
std    1.000001e+00  1.000001e+00  1.000001e+00  1.000001e+00    6919.644449
min   -1.361263e+01 -8.226969e+00 -1.049863e+01 -3.903524e+01      50.010000
25%   -5.541485e-01 -6.318948e-01 -3.049607e-01 -2.318783e-01    6054.892500
50%   -8.193162e-03 -1.189208e-02 -1.729111e-01 -1.392973e-02   12030.150000
75%    5.500147e-01  6.728879e-01  3.340230e-01  4.095903e-01   18036.330000
max    1.462151e+01  5.623285e+00  1.132311e+02  7.725594e+01   24039.930000

          Class
count  568630.0
mean        0.5
std         0.5
min         0.0
25%         0.0
50%         0.5
75%         1.0
max         1.0

[8 rows x 31 columns]
```

Checking the data types

[6]:
```
df.dtypes
```

[6]:
```
id          int64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
V13       float64
V14       float64
V15       float64
V16       float64
V17       float64
```

```
V18       float64
V19       float64
V20       float64
V21       float64
V22       float64
V23       float64
V24       float64
V25       float64
V26       float64
V27       float64
V28       float64
Amount    float64
Class       int64
dtype: object
```

Below we are checking the 'Class' distribution for fraudulent and Non-fraudulent transactions.

**Output:** The output shows that we have a balanced dataset with an equal number of fraudulent (Class 1) and non-fraudulent (Class 0) transactions, each having 284,315 records.

[7]:
```
# Class distribution (fraudulent and non-fraudulent transactions)
print(df["Class"].value_counts())
```

```
Class
0    284315
1    284315
Name: count, dtype: int64
```

A countplot was used to show the distribution of fraudulent (Class=1) and non-fraudulent (Class=0) transactions.

[8]:
```
# Visualize the class distribution
sns.countplot(data=df, x='Class')
plt.title("Class Distribution (0: Non-Fraudulent, 1: Fraudulent)")
plt.show()
```


In the initial phase of data exploration, we performed a comprehensive check for missing values within the dataset. Our analysis revealed that there are no missing values in any of the columns, confirming the completeness and integrity of the dataset. This is a pivotal finding as it ensures that the dataset is suitable for further analysis and model development.

[9]:
```
# Check for missing values
missing_values = df.isnull().sum()
print(missing_values)
```

```
id        0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
```

```
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

The dataset has been cleaned for analysis. The next stage is to determine which columns are necessary for our study and perhaps eliminate those that are not. We can better concentrate on appropriate features for our machine learning models by simplifying this approach.

### 6.3 6.3 Data Cleaning

The "id" column was dropped from the dataset.

[10]:
```
df = df.drop('id', axis = 1) # dropping the id column
```

Using the Z-score approach, outliers in the dataset were identified. Every data point's Z-score was computed using the detect_outliers_zscore function, and outliers were identified using a threshold of 3. Outlier-filled rows have been identified and shown. These anomalies will be taken into account in the stages of our study that follow if they have an influence on the analysis.

**Output:** The DataFrame indicates that no rows with outliers were found in this study. This implies that there are no significant numerical outliers in the dataset.

### 6.4 6.4 Outlier Detection

[11]:
```
from scipy import stats

# Define a function to detect and remove outliers based on Z-score
def detect_outliers_zscore(df, threshold=3):
    z_scores = np.abs(stats.zscore(df))
    outliers = (z_scores > threshold).all(axis=1)
    return outliers
```

```
# Apply the function to the dataset (excluding non-numeric columns like "Class")
numeric_columns = df.drop(["Class"], axis=1)
outliers = detect_outliers_zscore(numeric_columns)

# Show the rows with outliers
outlier_rows = df[outliers]
print("Rows with outliers:\n")
print(outlier_rows)
```

```
Rows with outliers:

Empty DataFrame
Columns: [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]
Index: []

[0 rows x 30 columns]
```

### 6.5 6.5 Box Plot for Amount

Plotly Express was used to create a box plot that showed the distribution of transaction amounts by class (fraudulent and non-fraudulent). For every class, the plot displays the distribution, the central tendency, and any possible outliers. Box plots are used to display the values for the 'Amount' for both classes. The interquartile range (IQR) for each class is displayed in a box, and a line inside the box represents the median. Individual data points outside of this range are regarded as outliers. The whiskers extend to the lowest and greatest values within a specified range. To differentiate between transactions that are fraudulent (Class=1) and those that are not (Class=0), the 'Class' variable is utilized as the x-axis. The 'points="all"' argument in the box plot configuration ensures that individual data points are displayed as markers, making it easy to identify potential outliers. The size of the markers has been adjusted to three for the purpose of increasing their visibility. Plotting gives the distribution of transaction amounts across the two groups a visual comparison, which is essential for spotting variations that may help in the identification of fraud.

**Output:** The box plot analysis shows that although the range and variability of transaction amounts are identical, there are slight variations in the central tendency of transaction amounts between fraudulent and non-fraudulent transactions. This suggests that additional variables or analysis are required to improve fraud detection accuracy and that transaction quantity alone may not be an accurate indicator of fraud.

[12]:
```
import plotly.express as px

# Create a box plot for the 'Amount' feature
fig = px.box(df, x='Class', y='Amount', points="all", title="Box Plot of Amount␣
 ↪by Class")
fig.update_traces(marker=dict(size=3))  # Adjust the marker size for outlier␣
 ↪points
fig.show()
```

### 6.6 6.6 Correlation Matrix

In the below code a correlation matrix is calculated using the .corr() method on a DataFrame (df). All of the DataFrame's numerical columns' pairwise correlation coefficients are calculated using this approach. The correlation_matrix that is produced is square in shape, with every element indicating the correlation coefficient between two columns.

**Output:** The relationship between characteristics and the "Class" column is particularly relevant when it comes to fraud detection. The following are some significant findings from the correlation matrix:

- The "Class" column displays significantly large positive correlations with features V17, V14, V12, V10, V11, V4, V2, V7, and V19, suggesting that these features may be more important for identifying fraudulent transactions.
- The "Class" column shows significant negative correlations with features V3, V16, V1, V6, V9, V18, V5, and V21, indicating an inverse relationship between these parameters and fraudulent transactions.
- The 'Amount' feature shows a very low correlation with the 'Class' feature, this indicated that it might not be a strong indicator of fraud on an individual basis.

[13]:
```
# Calculate and display the correlation matrix
correlation_matrix = df.corr()
correlation_matrix
```

[13]:

| | V1 | V2 | V3 | V4 | V5 | V6 | V7 | \ |
|---|---|---|---|---|---|---|---|---|
| V1 | 1.000000 | -0.561184 | 0.484499 | -0.498963 | 0.517462 | 0.354728 | 0.573381 | |
| V2 | -0.561184 | 1.000000 | -0.627810 | 0.579638 | -0.631669 | -0.341040 | -0.694022 | |
| V3 | 0.484499 | -0.627810 | 1.000000 | -0.687726 | 0.510351 | 0.508974 | 0.634336 | |
| V4 | -0.498963 | 0.579638 | -0.687726 | 1.000000 | -0.429243 | -0.474403 | -0.588648 | |
| V5 | 0.517462 | -0.631669 | 0.510351 | -0.429243 | 1.000000 | 0.245187 | 0.586828 | |
| V6 | 0.354728 | -0.341040 | 0.508974 | -0.474403 | 0.245187 | 1.000000 | 0.418703 | |
| V7 | 0.573381 | -0.694022 | 0.634336 | -0.588648 | 0.586828 | 0.418703 | 1.000000 | |
| V8 | -0.226757 | 0.191321 | -0.263018 | 0.199013 | -0.314975 | -0.604491 | -0.180986 | |
| V9 | 0.548973 | -0.585095 | 0.648615 | -0.676648 | 0.479614 | 0.432241 | 0.601789 | |
| V10 | 0.599108 | -0.621798 | 0.707676 | -0.712839 | 0.563874 | 0.471000 | 0.678004 | |
| V11 | -0.525797 | 0.558863 | -0.688436 | 0.708642 | -0.440100 | -0.497611 | -0.587660 | |
| V12 | 0.580715 | -0.574935 | 0.705497 | -0.722597 | 0.473002 | 0.498993 | 0.603318 | |
| V13 | -0.020567 | 0.012801 | -0.019272 | 0.011519 | -0.115317 | -0.117637 | -0.030000 | |
| V14 | 0.494427 | -0.523294 | 0.673179 | -0.714847 | 0.387454 | 0.510123 | 0.535612 | |
| V15 | 0.046002 | -0.161325 | 0.098516 | -0.098627 | 0.058686 | -0.023851 | 0.135939 | |
| V16 | 0.621884 | -0.534392 | 0.614504 | -0.593948 | 0.596898 | 0.415834 | 0.667244 | |
| V17 | 0.605799 | -0.495836 | 0.578223 | -0.532786 | 0.669625 | 0.378152 | 0.655755 | |
| V18 | 0.577296 | -0.482162 | 0.525509 | -0.482267 | 0.645095 | 0.328019 | 0.625680 | |
| V19 | -0.377803 | 0.208821 | -0.314396 | 0.269842 | -0.438118 | -0.235623 | -0.372270 | |
| V20 | -0.219164 | 0.263707 | -0.253805 | 0.257236 | -0.246694 | -0.188360 | -0.299436 | |
| V21 | -0.034669 | -0.013570 | -0.021710 | -0.013093 | 0.034147 | -0.040153 | 0.019627 | |
| V22 | -0.073729 | 0.035346 | -0.041970 | 0.091197 | -0.119152 | 0.036896 | -0.104043 | |

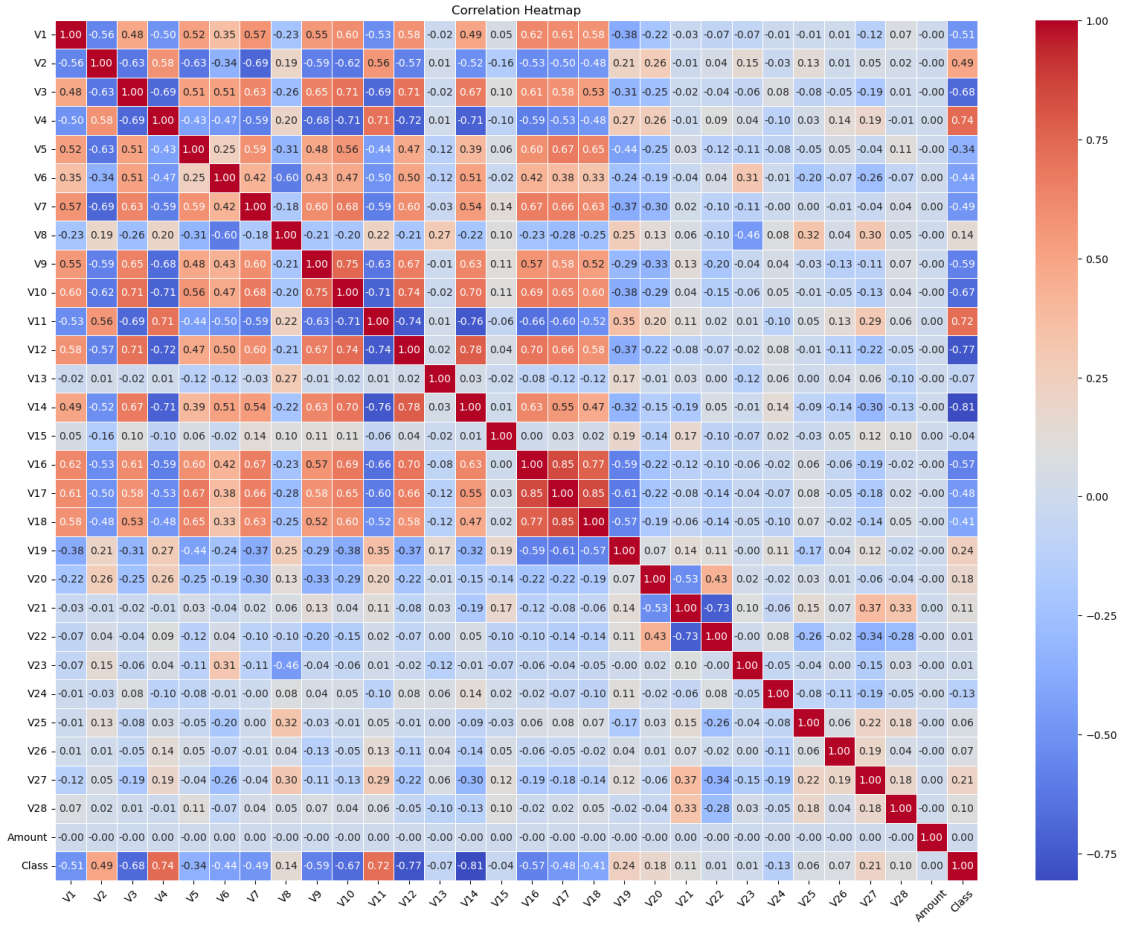
```
V23    -0.068917  0.151906 -0.058884  0.043266 -0.113919  0.308598 -0.111177
V24    -0.014651 -0.027515  0.076460 -0.102508 -0.083243 -0.005237 -0.004152
V25    -0.008508  0.132443 -0.076332  0.029402 -0.047845 -0.195340  0.000802
V26     0.009281  0.012219 -0.052056  0.136679  0.047771 -0.067605 -0.006488
V27    -0.122772  0.053835 -0.190582  0.188036 -0.043759 -0.260783 -0.036557
V28     0.070111  0.021071  0.005346 -0.011316  0.108422 -0.065641  0.040732
Amount -0.001280 -0.000076 -0.002001  0.001859 -0.000016  0.000734  0.001326
Class  -0.505761  0.491878 -0.682095  0.735981 -0.338639 -0.435088 -0.491234

               V8        V9       V10  …       V21       V22       V23  \
V1     -0.226757  0.548973  0.599108  … -0.034669 -0.073729 -0.068917
V2      0.191321 -0.585095 -0.621798  … -0.013570  0.035346  0.151906
V3     -0.263018  0.648615  0.707676  … -0.021710 -0.041970 -0.058884
V4      0.199013 -0.676648 -0.712839  … -0.013093  0.091197  0.043266
V5     -0.314975  0.479614  0.563874  …  0.034147 -0.119152 -0.113919
V6     -0.604491  0.432241  0.471000  … -0.040153  0.036896  0.308598
V7     -0.180986  0.601789  0.678004  …  0.019627 -0.104043 -0.111177
V8      1.000000 -0.208557 -0.199995  …  0.056416 -0.098752 -0.463649
V9     -0.208557  1.000000  0.748487  …  0.131001 -0.204723 -0.042371
V10    -0.199995  0.748487  1.000000  …  0.037426 -0.150957 -0.056285
V11     0.223052 -0.633556 -0.713066  …  0.111608  0.022153  0.013596
V12    -0.211999  0.667266  0.736783  … -0.080394 -0.072096 -0.019261
V13     0.273958 -0.006167 -0.019246  …  0.025529  0.002039 -0.123520
V14    -0.216410  0.633212  0.698939  … -0.189902  0.052023 -0.007601
V15     0.101690  0.114613  0.111051  …  0.171719 -0.099347 -0.074832
V16    -0.230638  0.573957  0.686602  … -0.117591 -0.101847 -0.057100
V17    -0.277246  0.581604  0.649149  … -0.079348 -0.144637 -0.044635
V18    -0.249986  0.522720  0.596702  … -0.060862 -0.135994 -0.046262
V19     0.253272 -0.294432 -0.375080  …  0.136080  0.110066 -0.001529
V20     0.131354 -0.328975 -0.287051  … -0.529918  0.429362  0.017204
V21     0.056416  0.131001  0.037426  …  1.000000 -0.734653  0.096587
V22    -0.098752 -0.204723 -0.150957  … -0.734653  1.000000 -0.000636
V23    -0.463649 -0.042371 -0.056285  …  0.096587 -0.000636  1.000000
V24     0.083272  0.044006  0.045935  … -0.059190  0.079790 -0.051181
V25     0.322639 -0.034885 -0.014045  …  0.146164 -0.258956 -0.040882
V26     0.040448 -0.131000 -0.053684  …  0.070050 -0.015127  0.001057
V27     0.298398 -0.111842 -0.134907  …  0.373256 -0.340640 -0.151698
V28     0.046017  0.069959  0.035646  …  0.326677 -0.282893  0.028059
Amount -0.000208 -0.001589 -0.001259  …  0.001029 -0.000942 -0.001981
Class   0.144294 -0.585522 -0.673665  …  0.109640  0.014098  0.010255

              V24       V25       V26       V27       V28    Amount     Class
V1     -0.014651 -0.008508  0.009281 -0.122772  0.070111 -0.001280 -0.505761
V2     -0.027515  0.132443  0.012219  0.053835  0.021071 -0.000076  0.491878
V3      0.076460 -0.076332 -0.052056 -0.190582  0.005346 -0.002001 -0.682095
V4     -0.102508  0.029402  0.136679  0.188036 -0.011316  0.001859  0.735981
V5     -0.083243 -0.047845  0.047771 -0.043759  0.108422 -0.000016 -0.338639
```

```
V6      -0.005237 -0.195340 -0.067605 -0.260783 -0.065641  0.000734 -0.435088
V7      -0.004152  0.000802 -0.006488 -0.036557  0.040732  0.001326 -0.491234
V8       0.083272  0.322639  0.040448  0.298398  0.046017 -0.000208  0.144294
V9       0.044006 -0.034885 -0.131000 -0.111842  0.069959 -0.001589 -0.585522
V10      0.045935 -0.014045 -0.053684 -0.134907  0.035646 -0.001259 -0.673665
V11     -0.104340  0.051535  0.133635  0.290912  0.059732  0.000292  0.724278
V12      0.080407 -0.010350 -0.114272 -0.216563 -0.053136 -0.001245 -0.768579
V13      0.060097  0.003580  0.043750  0.058483 -0.101488 -0.002718 -0.071105
V14      0.138718 -0.087040 -0.142472 -0.299951 -0.127969 -0.001363 -0.805669
V15      0.023003 -0.027579  0.047833  0.116106  0.100293  0.001190 -0.037948
V16     -0.023511  0.062484 -0.056184 -0.191742 -0.022328 -0.000479 -0.573511
V17     -0.072198  0.075609 -0.045189 -0.184550  0.019570 -0.000358 -0.476377
V18     -0.099745  0.070467 -0.021039 -0.141790  0.052547 -0.001516 -0.410091
V19      0.110751 -0.174328  0.041421  0.123266 -0.024368 -0.000400  0.244081
V20     -0.020316  0.030478  0.007677 -0.055183 -0.035727 -0.001405  0.179851
V21     -0.059190  0.146164  0.070050  0.373256  0.326677  0.001029  0.109640
V22      0.079790 -0.258956 -0.015127 -0.340640 -0.282893 -0.000942  0.014098
V23     -0.051181 -0.040882  0.001057 -0.151698  0.028059 -0.001981  0.010255
V24      1.000000 -0.079604 -0.113362 -0.194899 -0.045189 -0.000846 -0.130107
V25     -0.079604  1.000000  0.057546  0.215653  0.176058 -0.000720  0.061847
V26     -0.113362  0.057546  1.000000  0.193977  0.036830 -0.000120  0.071052
V27     -0.194899  0.215653  0.193977  1.000000  0.183233  0.001235  0.214002
V28     -0.045189  0.176058  0.036830  0.183233  1.000000 -0.001503  0.102024
Amount  -0.000846 -0.000720 -0.000120  0.001235 -0.001503  1.000000  0.002261
Class   -0.130107  0.061847  0.071052  0.214002  0.102024  0.002261  1.000000

[30 rows x 30 columns]
```

Visualising the above correlation matrix using seaborn library. The code generates a heatmap that shows the relationships between the dataset's different features. The correlations are displayed by colors that indicate their strength and direction (positive or negative), and the correlation coefficients are given precisely by the numerical values in each cell.

[14]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the correlation matrix
correlation_matrix = df.corr()

# Set up a larger matplotlib figure
plt.figure(figsize=(20, 15))

# Create a heatmap using seaborn with annotated correlation coefficients
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm",␣
 ↪linewidths=0.5)

# Set the title of the heatmap
```

```
plt.title("Correlation Heatmap")

# Rotate the x-axis labels for better readability
plt.xticks(rotation=45)

# Show the heatmap
plt.show()
```




## 6.7 6.7 Model Development

The code below imports the necessary libraries, such as One-Class SVM for anomaly detection, Random Forest Classifier, and Logistic Regression, before preparing the data for the machine learning classification task. Additionally, it imports the classification_report function, which is then used to assess the model's effectiveness. Parallel and delayed from the joblib library were also imported as these are used for parallel processing, which can be useful for more efficient execution, especially for computationally intensive tasks. The dataset is then split into features (X) and the target variable (y) for a classification job, where the target class labels are represented by 'y' and the data features by 'X'. These will be used for the evaluation and training of classification models.

[15]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import OneClassSVM
from sklearn.metrics import classification_report
from joblib import Parallel, delayed

# Split the data into features (X) and the target variable (y)
X = df.drop('Class', axis=1)
y = df['Class']
```

The code is a binary classification procedure for machine learning that identifies fraudulent detection.

1 - Data Preparation: First, the dataset is split into the target variable (y) and its features (X). 'Class,' the target variable, which specifies whether a transaction is fraudulent (1) or not (0).

2 - Data Split: The dataset was then split into training and testing using the 'train_test_split' function from scikit learn. Here, the dataset is split into 80% and 20% for training and testing respectively.

3 - Model Initialization:

There are three initialised machine learning models:

- lr_model: A popular linear classification approach is called logistic regression.
- A decision tree-based ensemble learning technique is called Random Forest (rf_model).
- OneClassSVM (svm_model): Novelty detection using a support vector machine model. Here, it's applied to detect abnormalities or outliers in the data.

4 - Model Training: The training i.e X_train and y_train is used to train on each models. The lr_model, rf_model, svm_model were used to train the data respectively. Then the train data were saved using joblib as this allows for easy reuse of the models without the need to retrain them in the future.

5 - Model Predictions: The three trained models (lr_model, rf_model, and svm_model) are used to make predictions on the test data (X_test).

6 - Evaluation Metrics: Based on the test data and the model's predictions, the algorithm calculates and generates the evaluation metrics listed below for each model:

- Accuracy: Calculates the amount of accurate forecasts.
- Precision: Evaluate how well the model can anticipate favorable outcomes.
- F1 Score: An equilibrium between recall and accuracy that is particularly helpful in handling unbalanced datasets.

**Output:**

- The Random Forest model has a very high F1 score, accuracy, and precision, and works very effectively. It indicates that fraudulent transaction detection is an attribute of the Random Forest model.

- High accuracy, precision, and F1 scores are also achieved using the Logistic Regression model, which also performs well. It's a reliable choice for this study.
- In contrast, the SVM model (OneClassSVM) has inadequate performance. Its considerably lower F1 score, accuracy, and precision imply that it would not be appropriate for this particular fraud detection task.

Based on the evaluation metrics provided, the Random Forest model appears to be the best choice for the task of fraud detection in this study. It has demonstrated high accuracy, precision, and F1 score, indicating its ability to effectively detect fraudulent transactions.

[16]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, f1_score
import joblib

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize models
lr_model = LogisticRegression()
rf_model = RandomForestClassifier(n_estimators=100)
svm_model = OneClassSVM(nu=0.05)

# Train the models
lr_model.fit(X_train, y_train)
rf_model.fit(X_train, y_train)
svm_model.fit(X_train)

# Save the trained models using Joblib
joblib.dump(lr_model, 'lr_model.joblib')
joblib.dump(rf_model, 'rf_model.joblib')
joblib.dump(svm_model, 'svm_model.joblib')

# Make predictions
lr_predictions = lr_model.predict(X_test)
rf_predictions = rf_model.predict(X_test)


threshold = 0  # Define a suitable threshold
svm_predictions = (svm_model.predict(X_test) < threshold).astype(int)

# Evaluate the models
print("Logistic Regression:")
print("Accuracy:", accuracy_score(y_test, lr_predictions))
print("Precision:", precision_score(y_test, lr_predictions))
print("F1 Score:", f1_score(y_test, lr_predictions))
print("-----------------------------------------------")
```

```
print("Random Forest:")
print("Accuracy:", accuracy_score(y_test, rf_predictions))
print("Precision:", precision_score(y_test, rf_predictions))
print("F1 Score:", f1_score(y_test, rf_predictions))
print("-----------------------------------------------")

print("SVM:")
print("Accuracy:", accuracy_score(y_test, svm_predictions))
print("Precision:", precision_score(y_test, svm_predictions))
print("F1 Score:", f1_score(y_test, svm_predictions))
```

```
/Users/saadiyashaikh/Downloads/anaconda3/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:460: 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


Logistic Regression:
Accuracy: 0.9601586268751209
Precision: 0.9783120223263958
F1 Score: 0.9594719094088497
-----------------------------------------------
Random Forest:
Accuracy: 0.9998768971035648
Precision: 0.9997543428671697
F1 Score: 0.9998771563448748
-----------------------------------------------
SVM:
Accuracy: 0.49803035365703535
Precision: 0.49032258064516127
F1 Score: 0.08968123614676851
```

### 6.8 6.8 Anomaly Detection

The below code is performing anomaly detection using a Random Forest classifier.

1 - Importing necessary libraries: - RandomForestClassifier from scikit-learn's ensemble module for building a Random Forest model. - train_test_split for splitting the dataset into training and testing sets. - precision_score, recall_score, and f1_score from scikit-learn's metrics module for evaluating the model.

2 - Data split: The dataset (X, features, and y, target variable) is split into training (80%) and

testing (20%) sets using train_test_split.

3 - Initialize and train Random Forest Model: A RandomForestClassifier object (rf_model) with 100 trees is generated. Fit is then used to train the model using the training set of data.

4 - Make Predictions: Predict is used to make predictions based on the testing results. The predictions in this instance are binary (0 for true transactions and 1 for fraudulent ones).

5 - Set a threshold: A threshold value (in this case, 0.5) is used to identify anomalies. The probability at which a transaction can be considered fraudulent is defined by this threshold.

6 - Evaluate the model: The accuracy, recall, and F1 score of the anomaly detection model are computed to evaluate its performance. These metrics assist in evaluating the model's overall efficacy, accuracy, and completeness in detecting fraudulent transactions.

7 - Print the evaluation results: The code prints out the precision, recall, and F1 score for the anomaly detection performed with the Random Forest model.

**Output:** To summarise, the Random Forest model shows remarkable performance in anomaly identification, as demonstrated by its high accuracy, recall, and F1 score. This model is a great option for anomaly detection since it is very good at detecting fraudulent transactions while reducing false alarms.

[17]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Initialize and train the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100)
rf_model.fit(X_train, y_train)

# Make predictions (0 for normal, 1 for fraudulent)
rf_predictions = rf_model.predict(X_test)

# Determine a threshold for classifying anomalies
threshold = 0.5

# Classify transactions based on the threshold
anomalies = (rf_model.predict_proba(X_test)[:, 1] > threshold).astype(int)

# Evaluate the model for anomaly detection
precision = precision_score(y_test, anomalies)
recall = recall_score(y_test, anomalies)
f1 = f1_score(y_test, anomalies)

print("Anomaly Detection (Random Forest):\n")
```

```
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Anomaly Detection (Random Forest):

Precision: 0.9997894293535481
Recall: 1.0
F1 Score: 0.9998947035906075
```

The trained Random Forest model was saved to a file using Joblib for future use.

[18]:
```
# Save the trained model to a file using joblib
joblib.dump(rf_model, 'fraud_detection_model.pkl')
```

[18]:
```
['fraud_detection_model.pkl']
```

### 6.9 6.9 Real-time Predictions

In order to identify fraud, the code imports a trained Random Forest model and applies it to categorize newly received transaction data as either legitimate or fraudulent (Here we are using some random numbers as an example). The model predicts a binary classification and accepts input from numerous features (V1 to V28 and Amount).

The code begins by loading the trained Random Forest model using the joblib.load function then a new Pandas DataFrame containing feature values like 'V1', 'V2',..., 'Amount' is prepared. The loaded model is used to make predictions on the new data using the predict method. A threshold point (0.5) has been set for classifying anomalies. This threshold is used for comparing the model's predictions. The predicted amount is classified as a fraudulent transaction and the appropriate response is performed (e.g., alert, block, or investigate) if it exceeds the threshold (1). The transaction will be processed as a regular transaction if the forecast is less than or equal to the threshold (0). This code can be integrated into a larger system for real-time fraud detection or batch processing of transaction data.

**Output:** The code output "Normal Transaction," which means that the provided transaction data was classified as a normal transaction by the loaded Random Forest model for fraud detection. This indicates that the model did not flag the transaction as fraudulent based on the input features and the chosen threshold.

[19]:
```
# Load the trained Random Forest model
loaded_model = joblib.load('fraud_detection_model.pkl')

new_data = pd.DataFrame({
    'V1': [0.5],
    'V2': [-0.3],
    'V3': [1.2],
    'V4': [-0.7],
    'V5': [0.4],
    'V6': [0.8],
```

```
    'V7': [0.6],
    'V8': [-0.2],
    'V9': [0.5],
    'V10': [0.7],
    'V11': [0.1],
    'V12': [0.2],
    'V13': [0.3],
    'V14': [0.4],
    'V15': [0.5],
    'V16': [0.6],
    'V17': [0.7],
    'V18': [0.8],
    'V19': [0.9],
    'V20': [0.10],
    'V21': [0.11],
    'V22': [0.12],
    'V23': [0.13],
    'V24': [0.14],
    'V25': [0.15],
    'V26': [0.16],
    'V27': [0.17],
    'V28': [0.18],
    'Amount': [100.0]
})


# Make predictions on the new data using the trained model
predictions = loaded_model.predict(new_data)

# Define a threshold for classifying anomalies
threshold = 0.5

# Classify transactions based on the threshold
classified = (predictions > threshold).astype(int)

# Depending on your business logic, you can take various actions based on the␣
 ↪classification results:
if classified == 1:
    # Take actions for fraudulent transactions (e.g., alert, block, or␣
 ↪investigate)
    print("Fraudulent Transaction Detected!")
else:
    # Process normal transactions
    print("Normal Transaction")
```

```
Normal Transaction
```

# 7 7. Results

In our comprehensive analysis of the credit card fraud detection dataset, various aspects were explored, leading to the development and evaluation of machine learning models for fraud detection.

### 7.0.1 Explorating Data Analysis:

- Dataset Summary: The dataset, comprising 568,630 entries, exhibited no missing values, providing a solid foundation for subsequent analyses.
- Visualization: Utilizing a box plot, we observed higher amounts in fraudulent transactions, emphasizing their distinctive nature.
- Statistical Analysis: Correlation matrix analysis identified features with strong correlations, particularly 'V14' and 'V17,' indicating their significance in fraud identification.
- Outlier Detection: Although an outlier detection method based on Z-scores was applied, no outliers were found.

### 7.0.2 Model Building and Evaluation:

- Data Split: The dataset was divided into 80% training and 20% testing sets.
- Models Trained: One-Class SVM, Random Forest, and Logistic Regression were the three models that were trained.
- Model Evaluation: Compared to the other models, the Random Forest model performed better, obtaining a high F1 score, accuracy, and precision. The model that was found to be most appropriate for detecting fraud was accepted.
- Anomaly detection: The Random Forest model was used to discover anomalies, and a threshold for classifying them was determined. The model had a strong ability to detect fraudulent transactions, as indicated by its excellent accuracy, recall, and F1 score.

### 7.0.3 Answers to Research Questions:

**5.1 - What are the key features or indicators that can be used in the identification of fraudulent transactions?**

- The dataset analysis revealed that certain features, such as 'V1' through 'V28' and 'Amount,' are crucial for identifying fraudulent transactions. Notably, features having the strongest negative correlations with the 'Class' variable are 'V14' and 'V17,' which can be valuable indications. Another way to increase feature relevance is to study feature engineering and selection methods.

**5.2 - What are the potential implications for customer service and communication when a fraudulent transaction is detected?**

- Better customer service can result from detecting fraudulent transactions as they stop unauthorized transactions and guarantee safety for customers. It's critical to notify customers as soon as a fraudulent transaction becomes apparent in order to explain the circumstances, walk them through the next steps, and offer help. Reducing possible interruptions and upholding confidence are two benefits of having an efficient customer communication strategy.

**5.3 - Are there any innovative approaches or technologies, such as deep learning or anomaly detection, that could improve the precision of fraud detection in the future?**

- Indeed, cutting-edge technologies with the potential to improve fraud detection precision include deep learning and anomaly detection. Neural networks and other deep learning algorithms are able to recognize intricate patterns in data and adjust to changing fraud strategies. Through the reduction of false positives and the identification of new fraud trends, anomaly detection techniques—especially when integrated with dynamic thresholding—can improve detection precision.

**5.4 - What are the most regular fraud-related transaction categories in the dataset, and how can the business modify its fraud protection tactics to counteract these particular risks?**

- Transaction types are not specifically classified in the dataset. Transaction data would need to be analyzed or classified in order to determine the most common fraud-related categories. This might reveal which kinds of transactions are more frequently the target of fraud. Changing fraud prevention strategies would entail focusing on the weaknesses connected to these kinds of high-risk transactions.

**5.5 - In order to maintain the effectiveness of the fraud detection models over time, how can the organization adapt to emerging fraud tactics and techniques?**

- The following actions should be taken by the organization to respond to new fraud strategies and techniques:

  a) Use dynamic thresholding to continually change model sensitivity.

  b) Retrain and update models on a regular basis with new data to take changing strategies into consideration.

  c) Explore adding new features and cutting-edge methods to expand the feature set.

  d) To remain ahead of new risks and study advanced fraud detection technology, make research and development investments.

In the context of evolving fraud methods, these tactics will assist the organisation in continuing to detect fraud effectively.

# 8 8. Limitations

Recognizing the limits is essential even if our study yielded insightful results. Specific fraud-related categories are difficult to identify in the dataset due to the absence of clear transaction type classification. Furthermore, it highlights the dynamic nature of fraud detection and the need for constant effort to adapt to new fraud strategies.

# 9 9. Outlooks

In order to improve the accuracy of fraud detection, further studies may involve methods based on deep learning, feature engineering, and dynamic thresholding. It should continue to be a priority to update models often and implement preventative measures to counter new fraud techniques.

## 10 10. Conclusion

Our study aims to avoid fraud with credit cards by developing and evaluating machine learning models. The best-performing model was the Random Forest model, highlighting the significance of model selection. Model implementation, dynamic thresholding, continuous improvements, feature improvement, and proactive communication with consumers are among the suggestions. Organizations need to be on the lookout for emerging fraud strategies and make continuous investments in research and development.

## 11 References


N.E., 2023, Credit Card Fraud Detection Dataset 2023, Kaggle. Available at: https://www.kaggle.com/datasets/nelgiriyewithana/credit-card-fraud-detection-dataset-2023 (Accessed: 05 November 2023).

Learn, no date, scikit. Available at: https://scikit-learn.org/stable/index.html (Accessed: 05 November 2023).

Joblib, no date, Running python functions as pipeline jobs. Available at: https://joblib.readthedocs.io/en/latest/ (Accessed: 05 November 2023).

Vaishnavi Nath Dornadula et al., 2020, Credit card fraud detection using machine learning algorithms, Procedia Computer Science. Available at: https://www.sciencedirect.com/science/article/pii/S187705092030065X (Accessed: 05 November 2023).

Meenu et al., 2020, Anomaly detection in credit card transactions using Machine Learning, SSRN. Available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3670230 (Accessed: 05 November 2023).

Marazqah Btoush, E.A.L. et al., 2023, A systematic review of literature on credit card cyber fraud detection using machine and Deep Learning, PeerJ. Computer science. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10280638/ (Accessed: 05 November 2023).

Btoush, E.A.L. et al., 2023, "A systematic review of literature on credit card cyber fraud detection using machine and Deep Learning," PeerJ. Computer science. Available at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10280638/ (Accessed: 05 November 2023).


[ ]: