This is a complete study report, including the codes and methodology used. I published a brief version of it, more direct, where I bring only the main results of this research.

1. Introduction:

With the advent of the internet, mobile phones, 4G, and technological banks known as fintech, the volume of digital transactions is rapidly increasing. In 2022, the major credit card companies together conducted over 580 billion transactions worldwide (Statista). In Brazil, there were nearly 40 billion transactions, amounting to over 3 trillion Brazilian reais. That's almost 19,000 transactions per second (Valor Investe). The forecast is for continued growth of around 15% per year, but in the first quarter of 2023, it was already 17% (ABECS) compared to the previous year's quarter**.**

Unfortunately, this presents a sea of opportunities for malicious individuals. **It is estimated that approximately 20 billion dollars are lost each year due to online payment fraud** (Ravelin). Furthermore, from 2021 to 2022, this type of fraud saw a 40% increase (Sumsub), indicating that fraudsters are becoming increasingly creative in their attempts to deceive banks.

These forms of fraud were identified in a study by KPMG, where financial institutions highlighted the 5 major challenges they face today. In the Americas, the following risks were ranked from most significant to least significant:

These forms of fraud were identified in a study by KPMG, where financial institutions highlighted the 5 major challenges they face today. In the Americas, the following risks were ranked from most significant to least significant**:**

- Cyberattacks and data breaches
- Instant and/or online payments
- Open Banking
- Digital channels
- Virtual currencies and/or cryptocurrencies

Therefore, fraud involving instant and/or online payments ranks as the second biggest concern for these institutions. This is due to the significant losses, as we have seen, that this type of crime causes both for financial institutions and for their customers.

Among the various measures to prevent such situations, there is the monitoring of transactions through the development of technologies that can assess risks and make real-time decisions to determine whether a particular payment is fraudulent or not. This is made possible thanks to advancements in artificial intelligence.

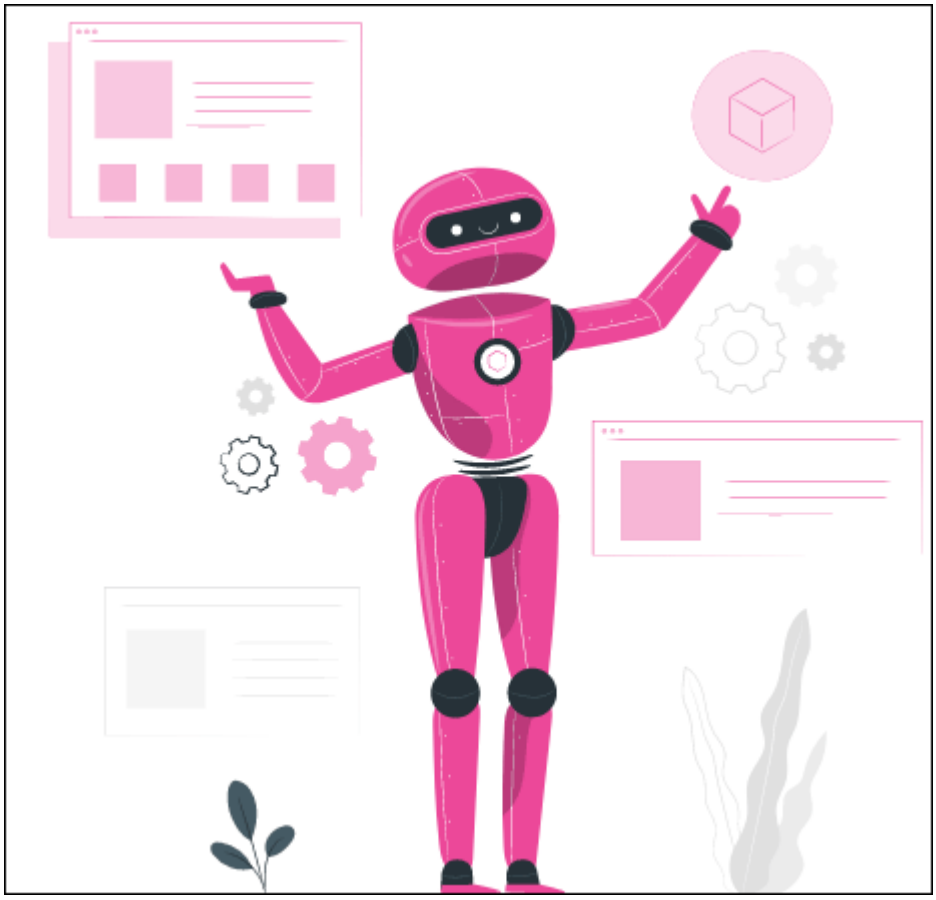
Proof of the effectiveness of using machine learning algorithms is that out of 10 banks, 7 invest in this type of technology (KPMG). However, despite the significant advancements in this area, there is still room for improvement. Half of the banks that perform these analyses reported a significant increase in False Positives, meaning that the model indicates a transaction as fraudulent when it is not. You

may have experienced this situation yourself: when trying to make a purchase, your card was preemptively blocked by the bank. As this situation causes embarrassment and problems for customers, banks aim to reduce these occurrences so that fraud detection becomes more effective.

With the evolution of digital channels, an increasing number of transactions are conducted in this manner. Consequently, the volume of historical data increases, providing access to more customer behavior information. All of this facilitates the identification of potential fraud. Therefore, ongoing studies of this kind are necessary for financial institutions, as even minimal improvement in such algorithms can result in millions of savings.

2. General Objective

To analyze credit card data and aim to improve fraud detection in credit card payments through machine learning methods.



2.1. Specific Objectives:

Perform exploratory data analysis to understand the dataset and extract meaningful insights.

Verify possible correlations between the attributes.

Develop predictive models using machine learning techniques.

Evaluate the models to determine which one performs the best.

3. Obtaining the Data

The data used in this project was provided by European credit card operators. This dataset contains transactions from a period of 2 days, resulting in nearly 290,000 transactions, out of which 492 were classified as fraud. Therefore, it is an imbalanced dataset, as fraud cases represent around 0.17% of the total transactions.

On the original data page, it is also mentioned that all attributes have been converted to numerical values for data protection and client anonymization purposes. Even the variable names have been modified and are identified only by the sequence of the letter ‘V’ followed by a number from 1 to 28. So, what we see are labels such as V1, V2, V3, …, V28.

It is worth noting that these variables have undergone a transformation process using the unsupervised algorithm called Principal Component Analysis (PCA), which aims to reduce dimensionality. In other words, it is a process that aims to decrease the original number of attributes in the dataset with the least loss of information possible. Additionally, PCA can distinguish relevant information from random or even redundant information. Knowing this fact, we can conclude that the data is standardized (all on the same scale), as this is a prerequisite for applying PCA.

Finally, the website mentions that Time and Amount variables did not undergo the transformation process previously mentioned. The Time attribute refers to the elapsed time in seconds between a given transaction and the first transaction in the dataset. The Amount feature represents the monetary value of each transaction. We also have the Class variable, which is our target variable, answering the question of whether a transaction is a fraud or not. For this variable, the response of "Yes" or "No" is converted into a binary value of 0 for non-fraudulent transactions and 1 for fraudulent ones.

3.1. Variable Dictionary

Understanding the dataset involves checking the available variables to conduct a complete analysis. Here is a summary of the attributes found and their respective meanings, listed in alphabetical order:

Alphabetically

Amount: The transaction amount (in euros).

Class: The target class that determines if the transaction is legitimate or fraudulent. It is represented by boolean values, where 0 identifies a legitimate transaction and 1 identifies a fraudulent transaction.

Time: The time elapsed in seconds since the first transaction in the dataset.

[V1, V2, V3, ..., V28]: Numeric attributes resulted from dimensionality reduction using PCA. They are kept this way also to protect sensitive and confidential data.

4. Data and Library Importation

When starting a project, it is necessary to install packages and import libraries that contain specific functions to be used in the subsequent code lines. Additionally, the data set is imported and saved in a specific variable for further use.

```
# install additional packages
!pip install scikit-plot -q
!pip install graphviz -q
!pip install pydotplus -q
```

```
# import libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import scikitplot as skplt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score, roc_curve, accuracy_score,
recall_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
import warnings
warnings.filterwarnings('ignore')

# additional configuration
plt.style.use('ggplot')
sns.set_style('dark')

# configure to display all lines and rows on the results
pd.options.display.max_rows = None
pd.options.display.max_columns = None
```

The data file was saved in the cloud as a precaution in case it is removed. The file is accessed using the provided address and saved in the variable df to be processed later.

```
# import credit card data set and save it in the 'df' variable
df =
pd.read_csv('https://www.dropbox.com/s/b44o3t3ehmnx2b7/creditcard.csv?dl=1')
```

5. Exploratory Data Analysis

This is an essential step in data science projects where the goal is to better understand the data by identifying patterns, outliers, potential relationships between

variables, etc. In this study, I explore the informations that are relevant to address the objectives mentioned earlier (see General Objective and Specific Objectives).

For that, various techniques and tools will be used, such as graphs, frequency tables, statistical data, and other methods that are found necessary. In this phase, the data scientist becomes a detective searching for insights that are not explicitly stated in the data frame. Therefore, the data will be visualized in different ways to gain a better understanding and test initial hypotheses, to obtain insights that can guide the rest of the project.

First, I generate visualizations of the first 5 and last 5 entries to check the composition of the data set and to ensure that there are no irregular records, such as total sums.

```
# print the 5 first entries
df.head()


# print the 5 last entries
df.tail()
```

It doesn't appear to be any apparent abnormalities in the data frame, and the fields seem to be well populated. This suggests that we may have a few missing data, which is great for the analysis. We will check this shortly.

The next step is to determine the size of this data set

```
# check the data set size
print('Data set Size')
print('-' * 30)
print('Total de entries:\t {}'.format(df.shape[0]))
print('Total de attributes:\t {}'.format(df.shape[1]))

"""
Data set Size
------------------------------
Total de entries:  284807
Total de attributes:  31
"""
```

Let's take a closer look at these 31 variables. The objective is to understand the type of variables present in these attributes and check for missing values, data distribution, outliers, etc.

Initially, I checked the names of the 31 attributes and their variable types (object, float, int). In this code output, given that we know we have around 285,000 records, we can get an idea of the number of missing values by observing the number of non-null values for each attribute.

```
# get data set information
df.info()

"""
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
```

```
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
"""
```

We can see that all variables are float type (**float64**), and the Class variable is of integer type (**int64)**, although it is known to be a categorical variable. This is because the values 0 and 1 represent a legitimate and a fraudulent transaction, respectively. However, in order to build machine learning models, the variables need to be in numerical values.

With the generated output above, it is also worth noting that there are no missing variables since the total of non-null values is equal to the number of records in the dataset. Let's confirm this more clearly with the code below:

```
# check the amount of missing data
df.isnull().sum()

"""
Time      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
"""
```

I will check the data imbalance by reviewing the results for legitimate transactions (equal to 0) and transactions flagged as fraud (equal to 1).

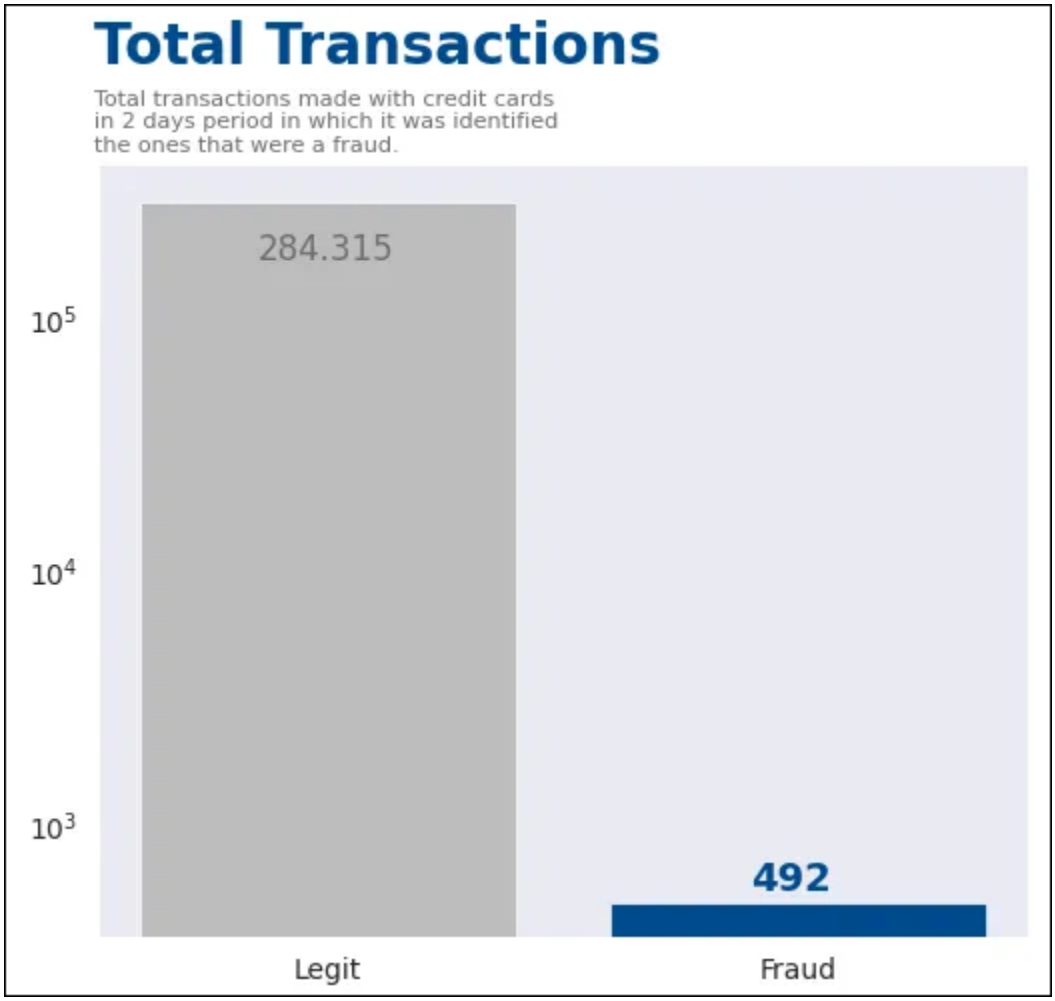
```
# total transactions and percentage of fraude
print('Legitimate transactions: {}'.format(df.Class.value_counts()[0]))
print('Fraud transactions: {}'.format(df.Class.value_counts()[1]))
print('-' * 30)
print('Fraud transactions represent {:.2f}% of the data
set.'.format(((df.Class.value_counts()[1]) * 100) / df.shape[0]))

"""
Legitimate transactions: 284315
Fraud transactions: 492
------------------------------
Fraud transactions represent 0.17% of the data set.
"""
```

Indeed, it is exactly as reported. That is, the data related to fraudulent transactions represent only 0.17% of the data frame.

Let’s visualize this proportion in a bar chart:

```
# plot bar chart for legit vs fraud transactions
## define axes
x = ['Legit', 'Fraud']
y = df.Class.value_counts()

## set color configuration
bar_colors = ['#bdbdbd', '#004a8f']

## plot chat
fig, ax = plt.subplots(figsize=(6, 5))
```

```
ax.bar(x, y, color=bar_colors)

### title
ax.text(-0.5, 1000000, 'Total Transactions', fontsize=20, color='#004a8f',
        fontweight='bold')

### subtitle
ax.text(-0.5, 450000, 'Total transactions made with credit cards\n'
        'in 2 days period in which it was identified\n'
        'the ones that were a fraud.', fontsize=8, color='#6d6e70')

### legit transactions
ax.text(-0.15, 170000, '284.315', fontsize=12, color="#6d6e70")

### fraudulent transactions
ax.text(0.9, 550, '492', fontsize=14, color="#004a8f", fontweight='bold')

### set edges
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
ax.spines['bottom'].set_visible(False)

### set logarithmic scale
ax.set_yscale('log')

plt.show()
```



Note that the visualization of the detected frauds is only possible in a logarithmic scale graph. Otherwise, it would not be possible to see the bar that represents fraudulent transactions.

This confirms the data imbalance and, therefore, specific treatment is necessary. The reason is that if a machine learning model were trained on the raw data, it would negatively impact the prediction results. The model would be very good at predicting legitimate transactions but would perform poorly in predicting fraudulent transactions. As a consequence, many fraudulent transactions would be classified as non-fraudulent, resulting in false negatives. In other words, the model would predict that a transaction is not fraudulent when it is! This would go against the objective of this study and the purpose of building the predictive model we are developing.

Next, I will check a statistical summary of the variables, as it provides important information such as mean, median, maximum and minimum values, standard deviation, as well as quartile values.

```
# generate statistics summarization
# rounded for 3 decimals
df.describe().round(3)
```

The variables treated by PCA, **Time** and **Class** do not show any discrepancies in the statistical data.

As for the **Amount** attribute, we can observe that the average purchase is 88 euros, with the 3rd quartile at 77 euros. This suggests that the majority of transactions are relatively low in value. Additionally, it was observed that the minimum transaction amount is 0, which is a bit strange. However, some transactions for testing the functionality of credit cards are done with this value. Since we do not have more information about it to make a more precise decision, I chose to keep this data.

Since we have these three known variables, let’s study them in more depth in order to identify patterns in fraudulent transactions. Therefore, we will check the frequency distribution of fraudulent and legitimate transactions over the available time data.

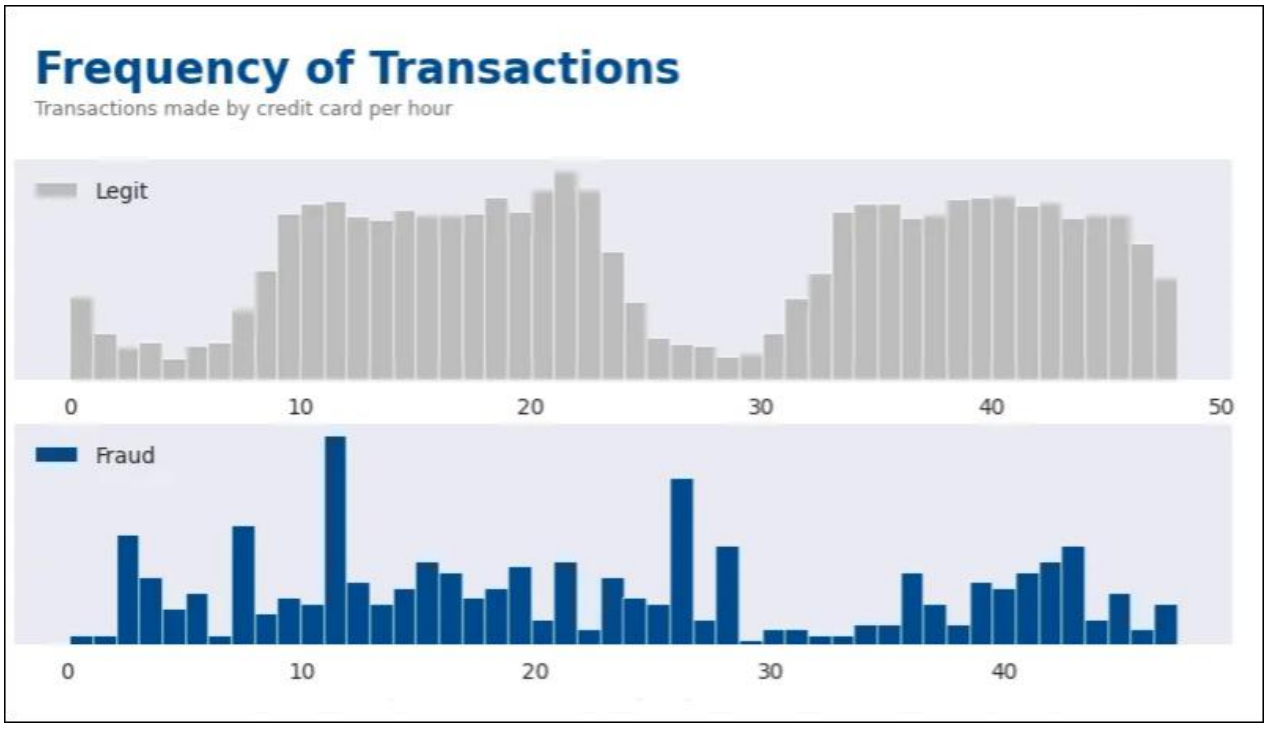
First, I will check how long the data collection lasted.

```
# verify the data collection duration
print('The total duration of data collections: {:.0f}
seconds'.format(df.Time.max()))
print('This corresponds to {:.0f} hours.'.format(df.Time.max() / 3600))

"""
The total duration of data collections: 172792 seconds
This corresponds to 48 hours.
"""
```

Therefore, we have 48 hours of data collection. We can check this on an hourly basis.

```
# plot a histogram of legit and fraudulent transactions in time
```

```
## set variables with legit and fraud transactions
legitimate = df[df.Class == 0].Time.div(3600)
fraud = df[df.Class == 1].Time.div(3600)

## set histogram
fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(10,4))

## title
ax[0].text(-1.5, 14000, 'Frequency of Transactions', fontsize=20,
color='#004a8f',
        fontweight='bold')

## subtitle
ax[0].text(-1.5, 12500, 'Transactions made by credit card per hour',
           fontsize=9, color='#6d6e70')

## data
### chart 1 - legit transactions
ax[0].hist(legitimate, label='Legit', bins=48, color='#bdbdbd')
ax[0].legend(loc='upper left', frameon=False)
ax[0].spines['right'].set_visible(False)
ax[0].spines['top'].set_visible(False)
ax[0].spines['bottom'].set_visible(False)
ax[0].spines['left'].set_visible(False)
ax[0].set_yticklabels(labels=[], visible=False)
ax[0].set_yticks(ticks=[])

### chart 2 - fraudulent transactions
ax[1].hist(fraud, label='Fraud', bins=48, color='#004a8f')
ax[1].legend(loc='upper left', frameon=False)
ax[1].spines['right'].set_visible(False)
ax[1].spines['top'].set_visible(False)
ax[1].spines['bottom'].set_visible(False)
ax[1].spines['left'].set_visible(False)
ax[1].set_yticklabels(labels=[], visible=False)
ax[1].set_yticks(ticks=[])

plt.show()
```



It can be observed that legitimate transactions occur more smoothly and at specific times. Since we do not know the exact start time of data collection, we can only hypothesize that they would be during business hours when the general population is more active.

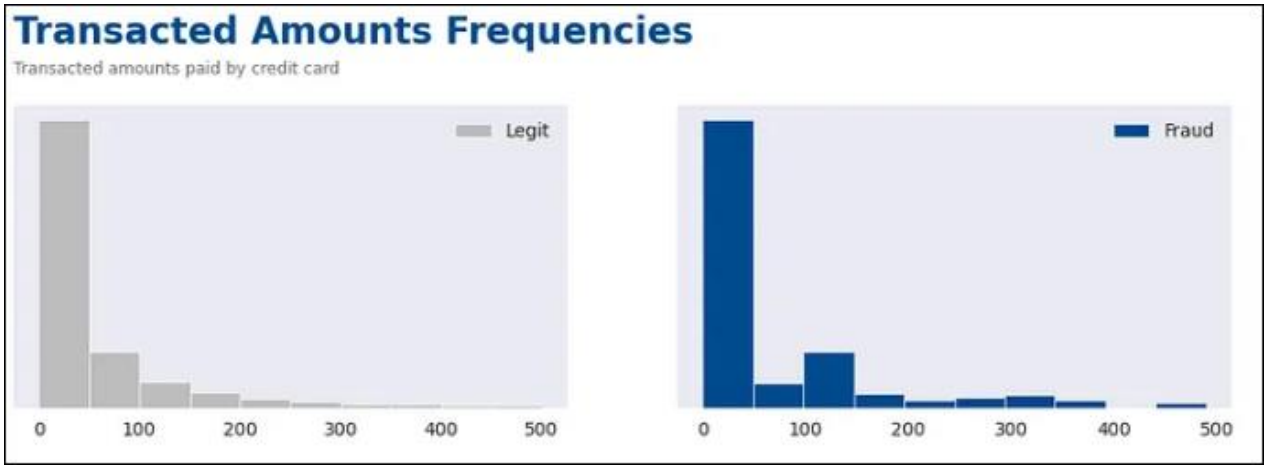
On the other hand, in the histogram showing the frequency of frauds, we have the opposite: there is no smoothness as the graph has many peaks. There are also no clusters of hours with more fraudulent transactions or hours with fewer transactions. Therefore, due to the lack of a pattern, it becomes more difficult to recognize them.

Now, let's check if it is possible to identify a pattern in fraudulent transactions based on the amount paid by credit card. For this, using the **Amount** variable, I will plot a histogram and also a box plot since they provide more statistical information to analyze. In both plots, I will compare the legitimate transactions with the fraudulent ones.

To plot the histogram of the transaction amount, based on the statistical summary, we know that 75% of the transactions were up to 77 euros. Therefore, I will limit the plot to a value of 500 euros, thus excluding a larger portion that could be considered outliers.

```
# plot a histogram of legit and fraudulent transactions by 'Amount'

## set variables with legit and fraud transactions <= 500 euros
legitimate_amount = df[df.Amount <= 500].loc[df.Class == 0]
fraud_amount = df[df.Amount <= 500].loc[df.Class == 1]

## set histogram
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(12,3))

## title
ax[0].text(-25, 240000, 'Transacted Amounts Frequencies', fontsize=20,
           color='#004a8f', fontweight='bold')

## subtitle
ax[0].text(-25, 220500, 'Transacted amounts paid by credit card',
           fontsize=9, color='#6d6e70')

## data
### chart 1 - legit transactions
ax[0].hist(legitimate_amount.Amount, label='Legit', bins=10, color='#bdbdbd')
ax[0].legend(loc='upper right', frameon=False)
ax[0].spines['right'].set_visible(False)
ax[0].spines['top'].set_visible(False)
ax[0].spines['bottom'].set_visible(False)
ax[0].spines['left'].set_visible(False)
ax[0].set_yticklabels(labels=[], visible=False)
ax[0].set_yticks(ticks=[])

### chart 2 - fraudulent transactions
ax[1].hist(fraud_amount.Amount, label='Fraud', bins=10, color='#004a8f')
ax[1].legend(loc='upper right', frameon=False)
ax[1].spines['right'].set_visible(False)
ax[1].spines['top'].set_visible(False)
ax[1].spines['bottom'].set_visible(False)
ax[1].spines['left'].set_visible(False)
```

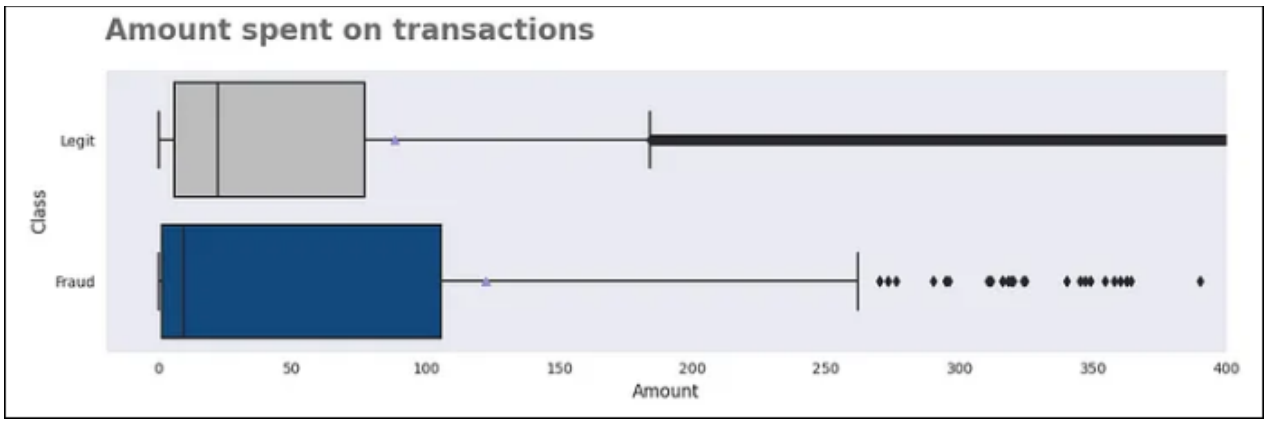
```
    ax[1].set_yticklabels(labels=[], visible=False)
    ax[1].set_yticks(ticks=[])

    plt.show()
```



It is noticeable that the frequency of legit transactions is predominantly low values (up to 50 euros). Above that, the frequency is drastically reduced and gradually decreases further.

In fraudulent transactions, although there is a similar predominance of values up to 50 euros, above that value, the frequency is not as uniform as in legitimate transactions. Among the observed values, the second highest frequency is in the range between 100 and 150 euros.

```
# set boxplots
fig, ax = plt.subplots(figsize=(12,4))

sns.boxplot(x=df.Amount, y=df.Class, orient='h', showmeans=True,
            palette=['#bdbdbd', '#004a8f'])
plt.xlim((-20, 400))
plt.yticks([0, 1], ['Legit', 'Fraud'])
plt.title('Amount spent on transactions', loc='left', fontsize=20,
          color='#6d6e70', fontweight='bold', pad=20)

plt.tight_layout()
```



There are differences between the quartiles and the median of the spent values. However, what catches my attention the most are the maximum values (calculated from the box plot) and the means.

To further analyze this, I will print a statistical summary to look at the spent values in legitimate and fraudulent transactions.

```
# filter legitimate transactions and generate summary statistics of 'Amount'
print('LEGIT'.center(20))
print(' ')
print('{}'.format(df.Amount.loc[df.Class == 0].describe().round(2)))
print(' ')
print('-' * 20)
print(' ')

# filter fraudulent transactions and generate summary statistics of 'Amount'
print('FRAUD'.center(20))
print(' ')
print('{}'.format(df.Amount.loc[df.Class == 1].describe().round(2)))

"""
       LEGIT

count    284315.00
mean         88.29
std         250.11
min           0.00
25%           5.65
50%          22.00
75%          77.05
max       25691.16
Name: Amount, dtype: float64

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

       FRAUD

count     492.00
mean      122.21
std       256.68
min         0.00
25%         1.00
50%         9.25
75%       105.89
```

```
max      2125.87
Name: Amount, dtype: float64
"""
```

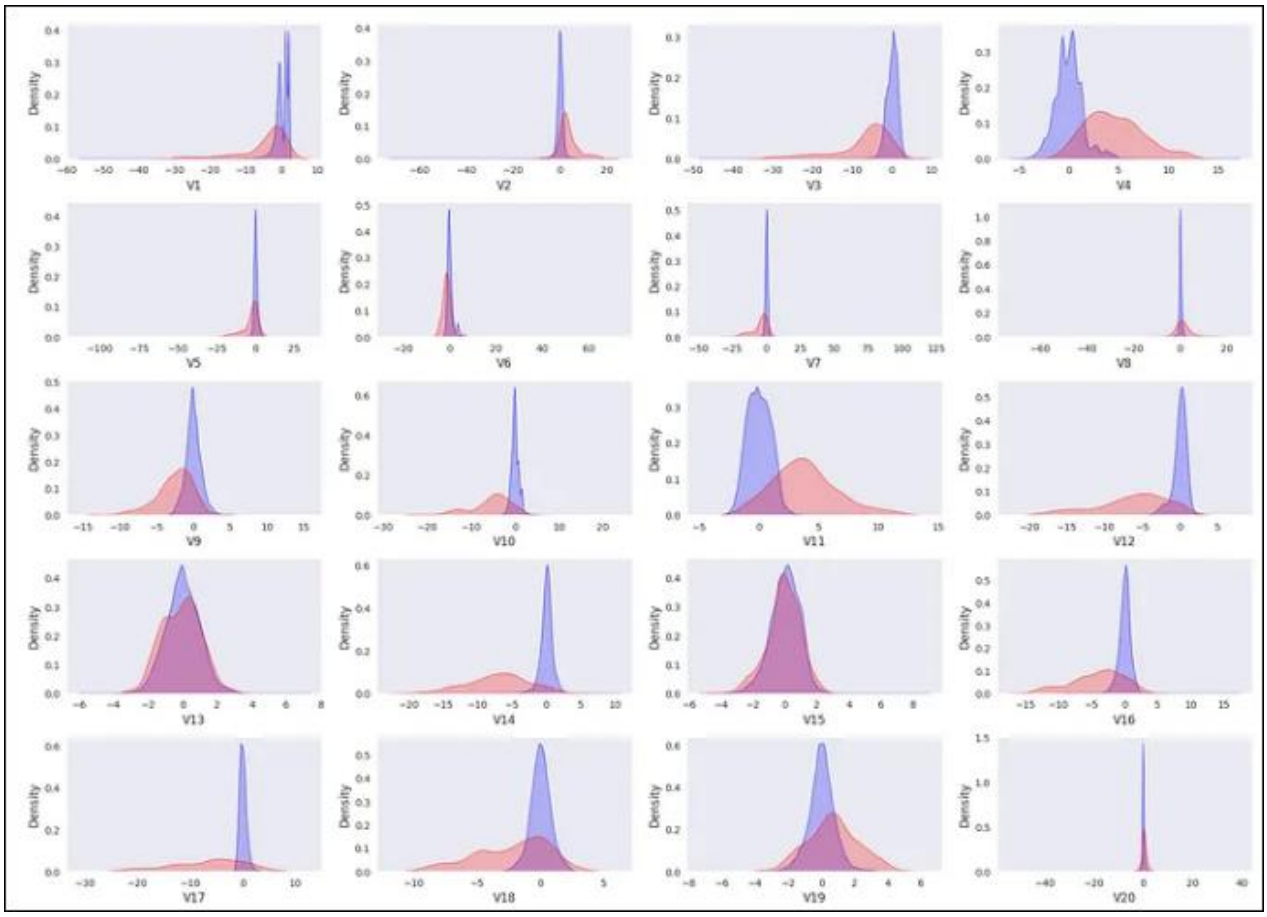
Here, it is found that the average value in a legitimate transaction is 88 euros, while in a fraudulent transaction, it is 122 euros. Looking at the 3rd quartile, 75% of fraudulent transactions were up to 105 euros, while legitimate transactions were up to 77 euros.

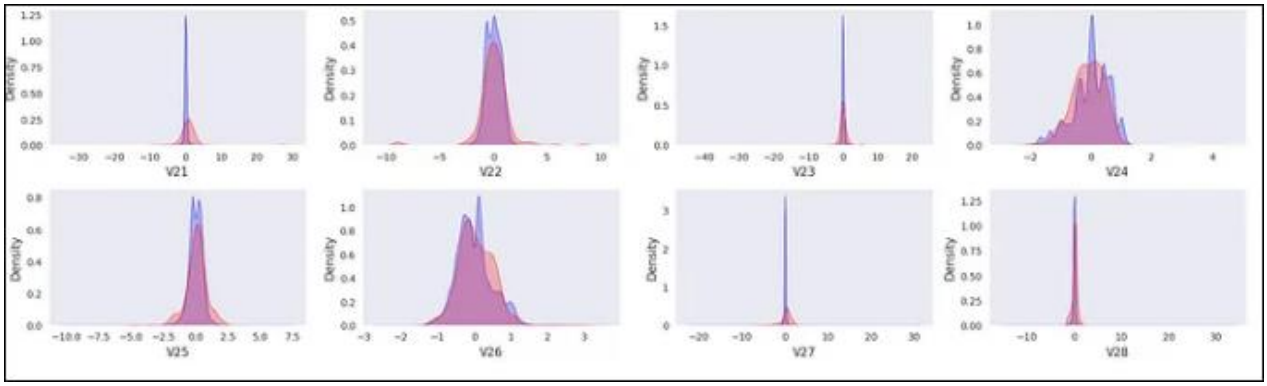
Additionally, legitimate transactions reached values of over 25,000 euros, while frauds remained within a smaller range: the highest fraudulent transaction was around 2,000 euros. This can be due to various reasons, such as the fact that a lower-value transaction is more common and may go unnoticed without raising any alarms, unlike transactions of 5, 10, or 20 thousand euros, where establishments and banks may require additional confirmations for security purposes.

All this information can contribute to training our machine learning models.

For the remaining variables, that means, those that went through PCA (V1, V2, V3, …, V28), density charts will be plotted since this type of plot will be more useful to compare the distributions between them. Thus, any anomalies such as distributions that are different from each other, can be detected and aid in fraud detection.

```
# plot density charts
## create variable with PCA (V1, V2, V3, ..., V28) columns
pca_columns = df.drop(['Class', 'Amount', 'Time'], axis=1).columns

## target legitimate and fraudulent transactions
class_0 = df[df.Class == 0]
class_1 = df[df.Class == 1]

# set chart
fig, ax = plt.subplots(nrows=7, ncols=4, figsize=(18,18))
fig.subplots_adjust(hspace=1, wspace=1)

# loop to plot all 28 variables
idx = 0
for col in pca_columns:
    idx += 1
    plt.subplot(7, 4, idx)
    sns.kdeplot(class_0[col], shade=True, color='b')
    sns.kdeplot(class_1[col], shade=True, color='r')

plt.tight_layout()
```





Note that each graph represents an attribute of the dataset and has two representations: blue for legitimate transactions and red for fraud. As the colors have transparency, when they overlap, they create a purple color.

Compare how different the graphs of attributes V14 and V15 are (as they are side by side, it is easier to analyze). In V14, the areas overlap very little, so we can clearly see the density of legitimate transactions and fraudulent transactions. In

contrast, in the following graph, the densities are so similar that almost everything is a purple area, indicating that both legitimate and fraudulent transactions behave similarly.

Therefore, variables **V3, V4, V10, V11, V12, V14, V16**, and **V17** are the ones that provide the most information for creating a fraud prediction model.

In this section, which aimed to perform an exploratory analysis of the credit card transaction dataset, we observed that:

In the overall check of the dataset, no anomalies in data filling or missing data were found.

The dataset has 284,807 records and 31 attributes.

All values are numerical:

- In the Class variable, 0 represents a legitimate transaction, and 1 represents a fraudulent transaction.

It is an imbalanced dataset, with only 0.17% of the records being frauds. Therefore, some treatment needs to be applied to generate a good predictive model.

In the three known variables (Time, Amount, and Class), a more detailed study concluded:

- Legitimate transactions occur more frequently in a specific time range.

- Frauds do not have a specific time pattern.

- Both occur more frequently with values below 50 euros. However, in legitimate transactions, the higher the value, the lower the frequency. In fraudulent transactions, we do not see this pattern.

This was also confirmed with the boxplot graphs and statistical data: the average transaction value of a fraud transaction is higher than one that is a legitimate transaction. This is also observed with the values found in the 3rd quartiles.

Finally, in the analysis of the PCA variables, some were identified that will be of great value for the construction of the fraud prediction model, as the densities of the variables show significant differences between legitimate transactions from the fraudulent transactions.5.1. Partial Conclusions.

6. Model Selection

To determine which data treatments, need to be applied and which ones are best suited for the task, we need to decide which machine learning models we are going to develop.

We start with the objective we want to achieve: detecting credit card fraud. Since we have a categorical class attribute, where 0 represents a legitimate transaction and

1 represents a fraud in the Class column, we have a target variable, which is why we are focusing on supervised learning models.

Furthermore, the answer to our question is binary, as it is either “Yes, this transaction is a fraud” or “No, this transaction is not a fraud.” So it is a classification problem, and among the algorithms studied so far, Logistic Regression and Decision Tree are the ones that make sense to apply in this case.

7. Data Preprocessing

Before proceeding with the creation of machine learning models, a preliminary data preparation step is necessary to generate the best algorithms. There are three operations that we need to pay attention to, and their order matters.

- Standardize **Time** and **Amount** attributes.
- Split the data into two sets: training and testing.
- Balance the dataset.

Before making any changes to the dataset, I will make a copy of it so that the alterations made from this point onwards are performed on this replica. This way, the original dataset remains intact.

```
# make a dataset copy
df_clean = df.copy()
```

7.1. Standardize Variables

The standardization of data aims to bring all values to the same scale so that different attributes can be compared. When the data is not standardized, some variables may have a greater impact than others, thereby affecting the model’s performance as it may favor one variable over others. Therefore, standardization prevents bias in the model.

As seen, the variables **V1, V2, V3, ..., V28** have already gone through PCA, so it can be inferred that they have been standardized since this process is a requirement for applying PCA to the data. Therefore, the remaining variables to be standardized are **Time** and **Amount** which have not been standardized.

In normal conditions, this step would be performed at two different moments, once the data is split into training and testing sets (before training the model on the training data and before making predictions on the test data). However, since the dataset has already undergone a similar treatment for other variables, performing this step earlier will not cause significant harm.

Thus, I chose to use the StandardScaler, as it follows the normal distribution and transforms the mean into 0 and the standard deviation into 1.

```
# standardize 'Time' and 'Amount' attributes with StandarScaler
## instantiate StandardScaler
std_scaler = StandardScaler()

# create new column with data standardized in 'Time'
df_clean['Time_STD'] =
std_scaler.fit_transform(df_clean['Time'].values.reshape(-1, 1))

# create new column with data standardized in 'Amount'
df_clean['Amount_STD'] =
std_scaler.fit_transform(df_clean['Amount'].values.reshape(-1, 1))

# delete columns 'Time' and 'Amount' without standardization
df_clean.drop(['Time', 'Amount'], axis=1, inplace=True)

# check first entries of modified data set
df_clean.head()
```

We can check the result of standardization on the Time_STD and Amount_STD attributes using the statistical summary:

```
# print summary statistics for 'Time_STD' and 'Amount_STD'
df_clean[['Time_STD', 'Amount_STD']].describe().round(3)
```

	Time_STD	Amount_STD
count	284807.000	284807.000
mean	-0.000	0.000
std	1.000	1.000
min	-1.997	-0.353
25%	-0.855	-0.331
50%	-0.213	-0.265
75%	0.937	-0.045
max	1.642	102.362

Statistical summary for ‘Time_STD’ and ‘Amount_STD’.

In both cases, you can see that the StandardScaler has fulfilled its purpose, and now we have a mean of 0 and a standard deviation of 1.

7.2. Data Split: Training and Testing

To have a generic model that best fits real-world data, it is necessary to divide the dataset into a training set, on which the model will learn, and a testing set, which will be used to evaluate the performance of the created model.

This division should occur randomly to avoid biased samples. Additionally, it should be done before balancing the data to ensure that the balanced dataset is suitable.

Before proceeding with the step above, it is necessary to separate the target class, the **Class** variable, that we want to predict, from the independent variables. We will store the independent variables in a variable called X and the target variable (**Class**) in another variable called y.

- The data split for training and testing involves some additional configurations:
- The size was set to 70:30, meaning that the training set will contain 70% of the total data, while the testing set will have 30% of the total dataset.
- The training and testing sets will contain the same proportion of classes, which means that they are having the same amount of legitimate transactions and fraudulent transactions.
- Data randomization was enabled to shuffle the records and ensure random data in the training and testing sets.
- A seed value was specified to reproduce the code without changes in the result.

```
# separate independent variables from the dependent variable
X = df_clean.drop('Class', axis=1)
y = df['Class']

# split training and test data
## stratify= y (to divide so that the classes have the same proportion)
## random_state so that the result is replicable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    stratify=y, shuffle=True,
                                                    random_state=110)
```

7.3. Data Balancing

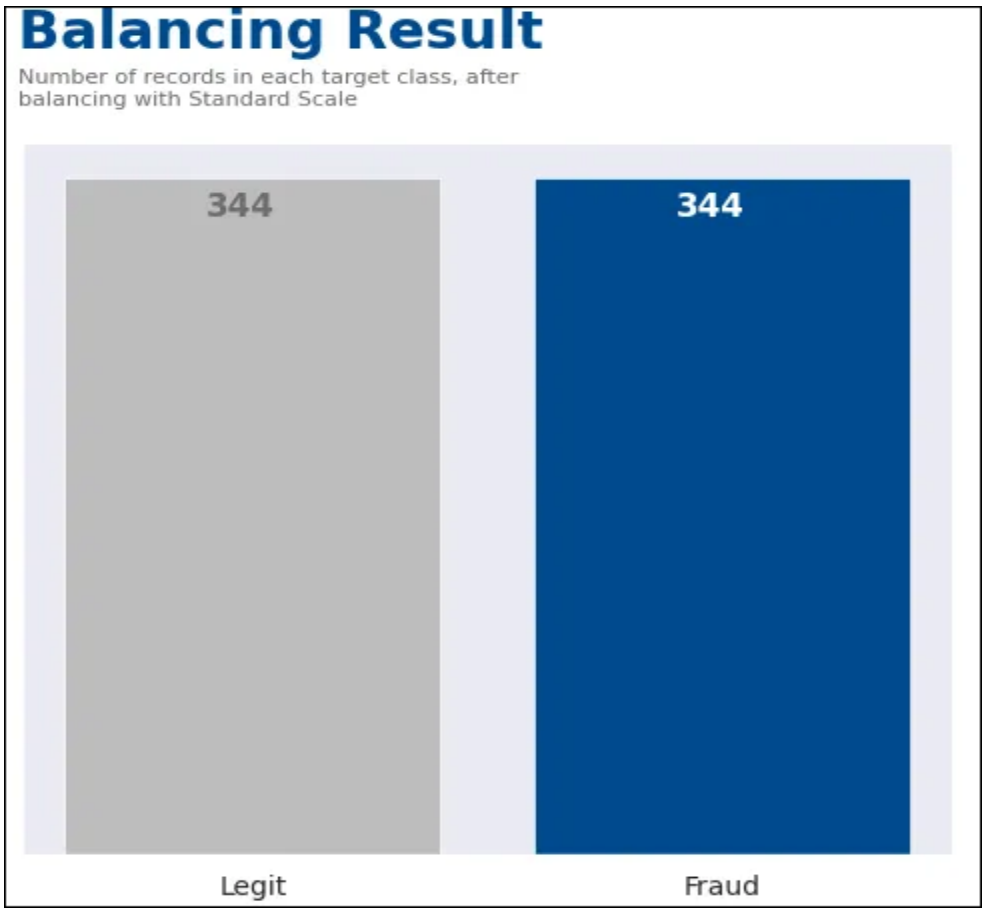
As observed, the dataset is imbalanced, and it is necessary to rebalance the data to avoid creating a model with low performance in identifying fraudulent transactions, as well as overfitting, and to create a good machine learning model.

The method used in this case is Undersampling, which reduces the majority class by randomly excluding some of its data. This way, the characteristics of the minority class, which in this case are the fraudulent transactions, are preserved. These are the most important data for solving our problem.

This technique is applied only to the training set to ensure that the characteristics of the testing set are not distorted.

```
# apply undersampling technique
## random_state so that the result is replicable
rus = RandomUnderSampler(random_state=606)
X_rus, y_rus = rus.fit_resample(X_train, y_train)

# check class balancing
print(pd.Series(y_rus).value_counts())

"""
0    344
1    344
Name: Class, dtype: int64
"""
```

```
# plot bar chart for legit vs fraud transactions
## define axes
x = ['Legit', 'Fraud']
y = y_rus.value_counts()

## set bar colors
bar_colors = ['#bdbdbd', '#004a8f']

## plot chart
fig, ax = plt.subplots(figsize=(6, 5))
ax.bar(x, y, color=bar_colors)
ax.set_yticklabels(labels=[], visible=False)

### title
ax.text(-0.5, 410, 'Balancing Result', fontsize=20, color='#004a8f',
        fontweight='bold')

### subtitle
ax.text(-0.5, 370, 'Number of records in each target class, after\n'
        'balancing with Standard Scale\n', fontsize=8, color='#6d6e70')

### amount of legitimate transactions
ax.text(-0.1, 325, '344', fontsize=12, color="#6d6e70", fontweight='bold')

### number of fraudulent transactions
ax.text(0.9, 325, '344', fontsize=12, color="#FFFFFF", fontweight='bold')

plt.show()
```



After applying the Undersampling method, we can see that both classes now have the same proportion. There are 344 records for legitimate transactions and the same number for fraud cases.

7.4. Result

After this data preprocessing step, let's examine the datasets that will be used in the machine learning models**.**

```
print('Dimensions of sets BEFORE treatment' + '\n' + '-' * 45)
print('TRAINING')
print('X_train: {}'.format(X_train.shape))
print('y_train: {}'.format(y_train.shape))
print('-' * 18)
print('TEST')
print('X_test: {}'.format(X_test.shape))
print('y_test: {}'.format(y_test.shape))
print(' ')
print('\nDimensions of sets AFTER treatment' + '\n' + '-' * 45)
print('TRAINING')
print('X_rus: {}'.format(X_rus.shape))
print('y_rus: {}'.format(y_rus.shape))
print('-' * 18)
print('TEST')
print('X_test: {}'.format(X_test.shape))
print('y_test: {}'.format(y_test.shape))

"""
Dimensions of sets BEFORE treatment
---------------------------------------------
TRAINING
```

```
X_train: (199364, 30)
y_train: (199364,)
------------------
TEST
X_test: (85443, 30)
y_test: (85443,)

Dimensions of sets AFTER treatment
-------------------------------------------
TRAINING
X_rus: (688, 30)
y_rus: (688,)
------------------
TEST
X_test: (85443, 30)
y_test: (85443,)
"""
```

Since we only applied to balance to the training set, it will have a different size after the treatment**.**

8. Correlation Matrix

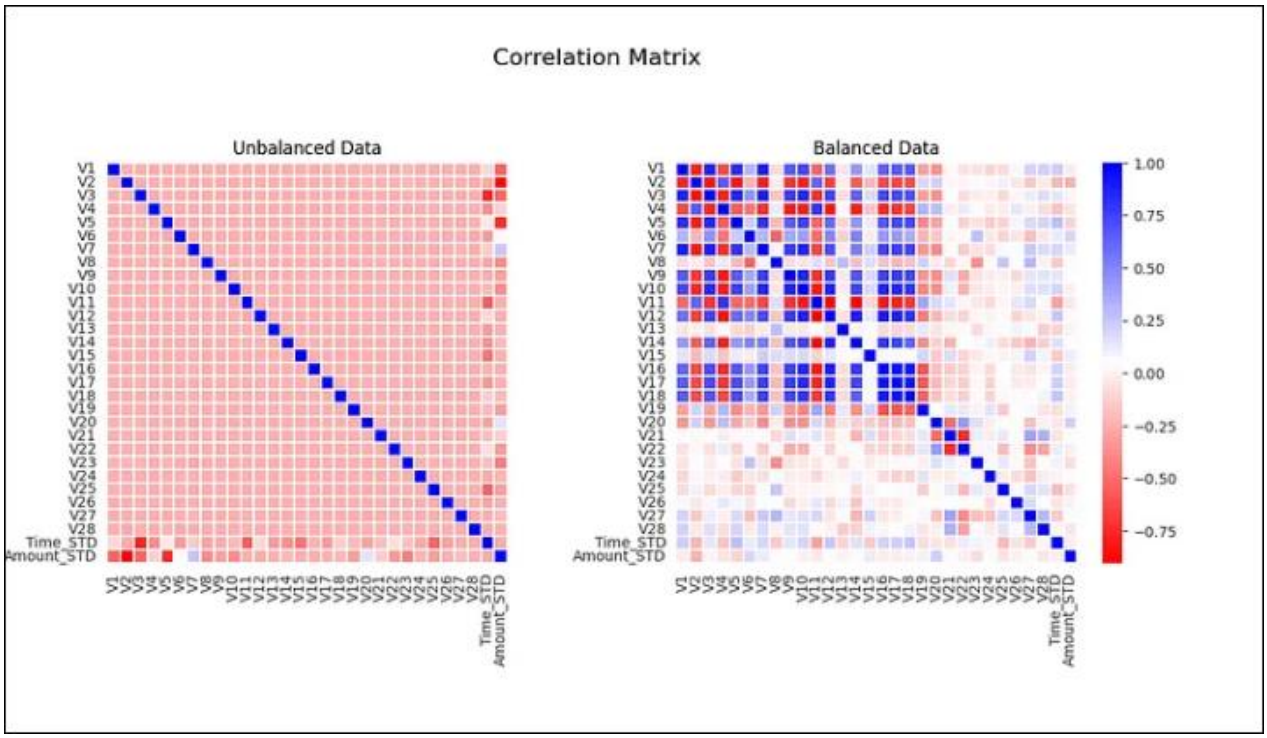
With the data properly balanced, we can proceed to check the relationships between variables to identify possible correlations among them. Without balancing the data, creating a correlation matrix would not be useful for extracting such information (as shown below).

Since it was the training data that was balanced, I will use it to calculate the relationships. To visualize this information more intuitively, I am going to plot it in a heatmap.

The first step is to use the **corr** function, which calculates the Pearson correlation coefficient. The result ranges from -1 to 1. A value closer to 1 indicates a strong positive relationship between variables, while a value closer to -1 indicates a strong negative relationship. Finally, a value close to 0 indicates no relationship between the attributes.

```
# calculate correlations
corr = X_train.corr()
corr_rus = X_rus.corr()
```

```
# plot heatmap
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(14, 5))
fig.suptitle('Correlation Matrix', fontsize=16, y=1.1, color='black')

# unbalanced data
ax[0].set_title('Unbalanced Data', fontsize=12, color='black')
sns.heatmap(corr, cmap='bwr_r', cbar=False, square=True, xticklabels=True,
            yticklabels=True, linewidths=.1, ax=ax[0])

# balanced data
ax[1].set_title('Balanced Data', fontsize=12, color='black')
sns.heatmap(corr_rus, cmap='bwr_r', cbar=True, square=True, xticklabels=True,
            yticklabels=True, linewidths=.1, ax=ax[1])

plt.show()
```



To read the graphs above, keep in mind that the stronger the red or blue color, the stronger the relationship between the variables. From the left heatmap, with imbalanced data, there is little or no relevant information to be inferred.

However, in the right heatmap, with the balanced data, there is a noticeable difference in the relationships between variables. We can observe strong correlations between attributes **V1** to **V18**. This information will be important for building the models, as it helps the algorithm understand certain behaviors between the classes and distinguish fraudulent transactions from legitimate ones.

9. Machine Learning Models

In this section, I will develop two predictive models for credit card fraud detection. The models will be based on Logistic Regression and Decision Tree algorithms. Later, I will evaluate the performance of the constructed models to compare them and decide which one would be better to put into use.

9.1. Evaluation Metrics

To evaluate the performance of classification algorithms, we use metrics such as Precision, Recall, Accuracy, F1-Score, and the confusion matrix. However, when dealing with imbalanced data, as is the case with our dataset (even after balancing it), accuracy is not a good evaluation metric. This is because high accuracy can be achieved even if the detection of fraud is very low, which does not align with our objective.

Among these metrics, Recall provides the best measure for the specific problem at hand. This is because, in the case of fraud detection, False Negatives are more harmful to a company than False Positives. In other words, it is better for the model to incorrectly identify a transaction as fraud when it is not, rather than incorrectly classify a fraudulent transaction as legitimate, which would result in business losses.

Therefore, we aim for a high recall rate.

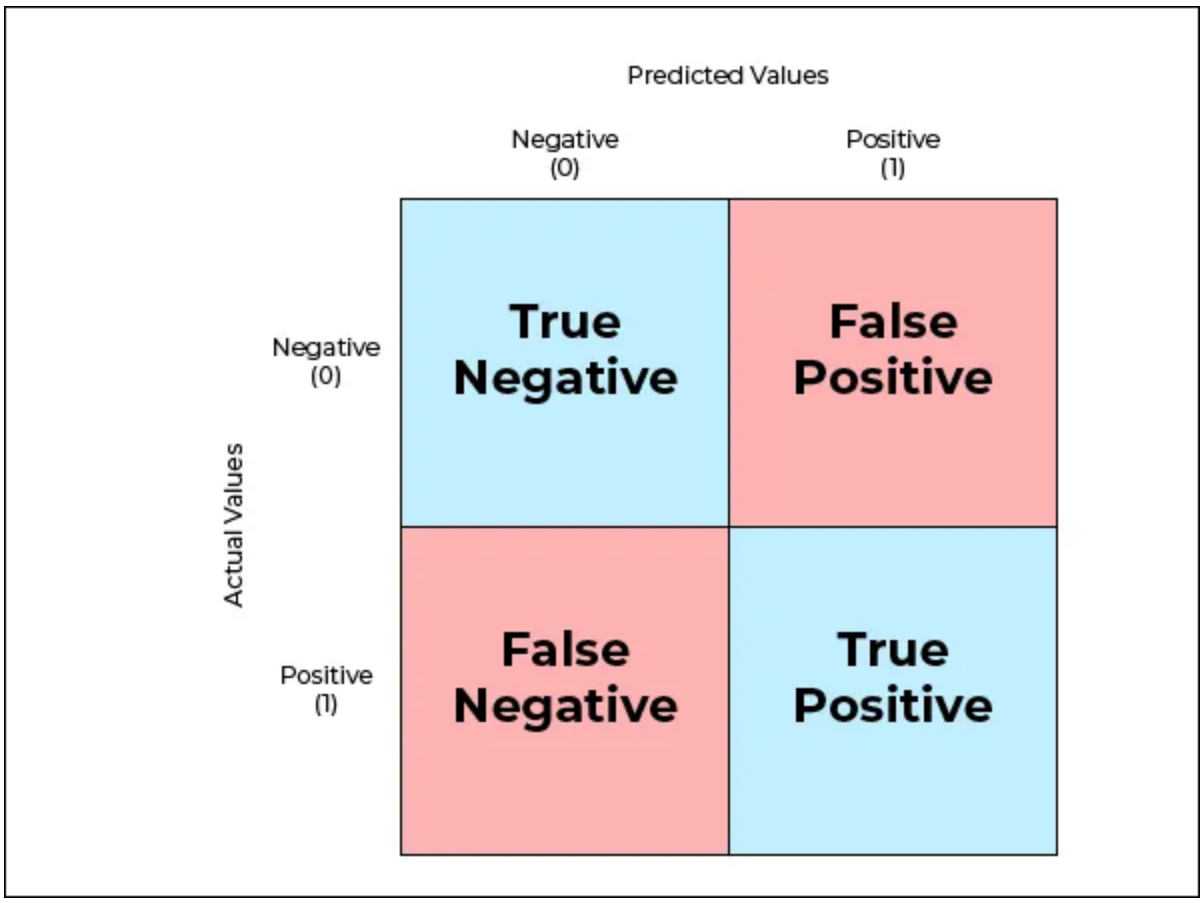
Recall looks at all legitimate transactions and asks: How well does the model perform? The result is a value between 0 and 1, with a value closer to 1 indicating a low False Negative rate.

It can be calculated as follows:

$$\text{Recall:} \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

Considering the purpose of this study, another evaluation metric that can be used is AUC (Area Under the Curve). AUC is derived from ROC (Receiver Operating Characteristic) and indicates how well the model can distinguish between two things. In our example, it represents the ability to distinguish between a legitimate transaction and a fraudulent one. The AUC value ranges between 0 and 1, with a value closer to 1 indicating a better model.

Lastly, the confusion matrix compares the predicted values with the actual values, showing the model's errors and correct predictions. The output consists of four different values arranged as follows:



Each of these values corresponds to:

Model's Correct Predictions

- **True Positive**: A fraud transaction correctly classified as fraud.
- **True Negative**: A legitimate transaction correctly classified as legitimate.

Model's Errors:

- **False Positive**: A legitimate transaction incorrectly classified as fraud.
- **False Negative**: A fraud transaction incorrectly classified as legitimate.

To sum up, to evaluate the performance of the models, we will look at the confusion matrix, recall, and AUC.

9.2. Logistic Regression

Logistic Regression is a classification algorithm that assigns observations to classes based on their probability of belonging to a particular group of classes. Therefore, it is a good model to use when the dependent variable is categorical.

In our study, this means that once the model is trained, it will receive information about a new credit card transaction and, according to the algorithm, determine the probability of it being a fraud. The class with the highest probability is what the algorithm will indicate as the potential class of that transaction.

To do this, logistic regression transforms the output generated by the model using the sigmoid function (which is a logistic function) to return a probability value and then determine the class to which the observation belongs.

9.2.1. Model Training

```
# create model with Logistic Regression

# select seed to replicate the model
np.random.seed(29)

# instantiate the model
model_LR = LogisticRegression()

# train the model on the training data
model_LR.fit(X_rus, y_rus)

# make predictions
y_pred_LR = model_LR.predict(X_test)
```

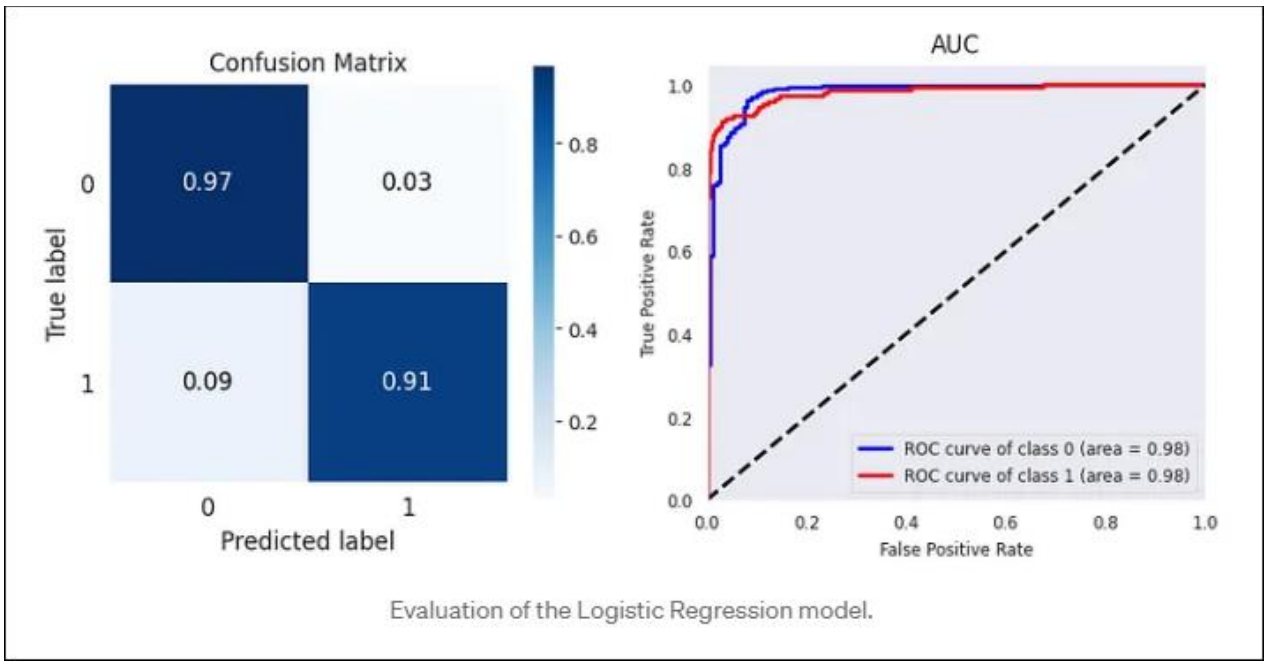
```
# calculate the probability of the classes
y_prob_LR = model_LR.predict_proba(X_test)

# print the probabilities of the first 5 records
print('y_pred_LR \n', y_pred_LR[0:5])
print('\n y_prob_LR \n', y_prob_LR[0:5])

"""
y_pred_LR
 [0 0 0 0 0]

 y_prob_LR
 [[0.98496983 0.01503017]
 [0.88832431 0.11167569]
 [0.9802882  0.0197118 ]
 [0.97829236 0.02170764]
 [0.99845446 0.00154554]]
"""
```

Above, we can see how the model works. The algorithm's result generates a list that contains the probability for each transaction to be legitimate or fraudulent. In the first row, the result is [0.98496983 0.01503017], which means that there is a 98.5% probability of it being a legitimate transaction and a 1.5% chance of it being a fraud**.**

9.2.2. Model Evalutation

Here, we will look at the performance metrics of the model created above. The report will display all the measures mentioned earlier in the Evaluation Metrics section, but we will focus on the ones of interest for this specific study, which are Recall, AUC, and the confusion matrix.

```
# print assessment metrics report
print('Evaluation Metrics Report'.center(65) + ('\n') + ('-' * 65))
print(classification_report(y_test, y_pred_LR, digits=4) + ('\n') + ('-' *
15))

# print  AUC
print('AUC: {:.4f} \n'.format(roc_auc_score(y_test, y_pred_LR)) + ('-' * 65))

# plot charts
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))

# confusion matrix
skplt.metrics.plot_confusion_matrix(y_test, y_pred_LR, normalize=True,
                                    title='Confusion Matrix',
                                    text_fontsize='large', ax=ax[0])

# AUC
skplt.metrics.plot_roc(y_test, y_prob_LR, title='AUC', cmap='brg',
                       text_fontsize='small', plot_macro=False,
                       plot_micro=False, ax=ax[1])

plt.show()

"""
                    Evaluation Metrics Report
-----------------------------------------------------------------
              precision    recall  f1-score   support

           0     0.9998    0.9672    0.9832     85295
           1     0.0460    0.9122    0.0876       148

    accuracy                         0.9671     85443
   macro avg     0.5229    0.9397    0.5354     85443
weighted avg     0.9982    0.9671    0.9817     85443

---------------
AUC: 0.9397
-----------------------------------------------------------------
"""
```

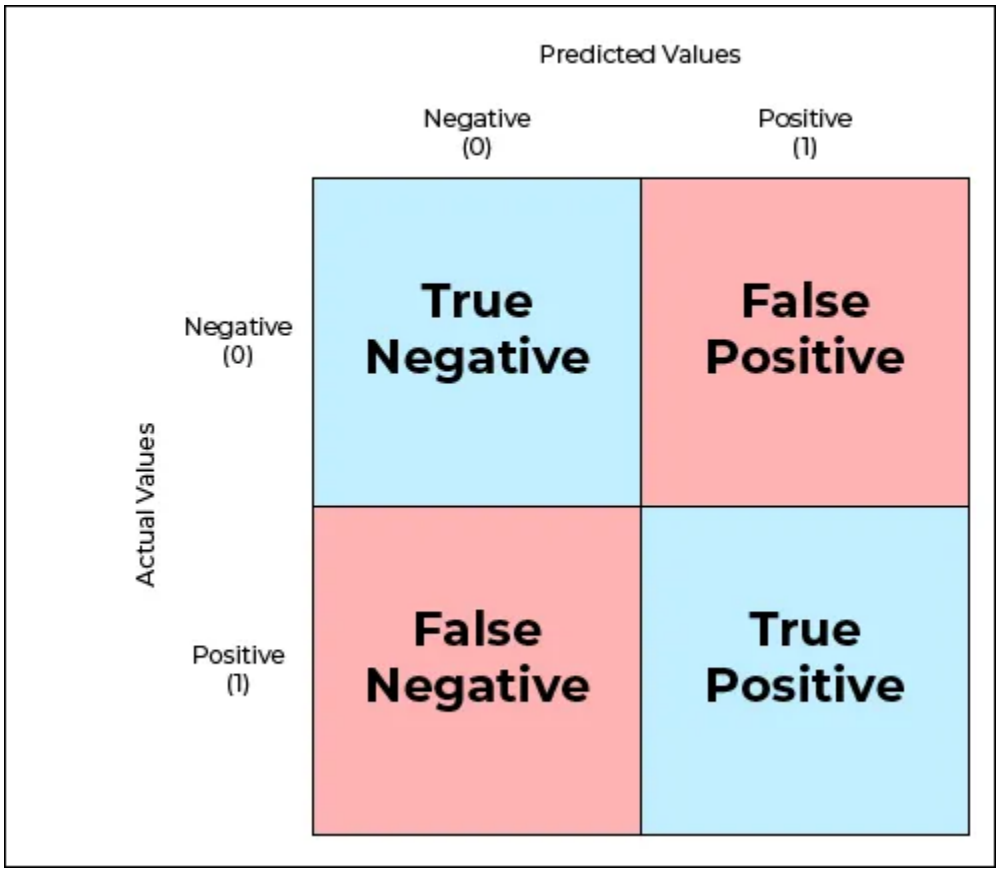


Evaluation of the Logistic Regression model.

9.2.3. Results Interpretation

Confusion Matrix

The initial understanding, we need to have of the model generated from Logistic Regression is its confusion matrix, as shown in the image above and the table below, from which we can derive the following results:



Model's Correct Predictions

- 91% True Positive

A fraudulent transaction is correctly classified as fraud.

- 97% True Negative

A legitimate transaction is correctly classified as legitimate.

Model's Errors

- 3% False Positive

A legitimate transaction is incorrectly classified as fraud.

- 9% False Negative

A fraudulent transaction is incorrectly classified as legitimate

Recall

Recall looks at all legitimate transactions and asks: How well does the model perform? The result is a value between 0 and 1, with a value closer to 1 indicating a low False Negative rate**.**

The generated model has a Recall of 0.9122, which means that in 91.22% of cases involving fraud, the model will be correct.

AUC

The AUC value ranges between 0 and 1, with a value closer to 1 indicating a better model.

The Logistic Regression model created above has an AUC of 93.97%.

9.3. Decision Tree

A Decision Tree is a supervised learning algorithm used for classification and regression problems. Its process involves finding boundaries in the data and then splitting them into subsets.

9.3.1. Model Training

To create a Decision Tree model, you need to specify a hyperparameter that determines the maximum depth the tree should reach. Since there is no specific metric to decide this, only a recommendation to start with a depth of 3 and then test different values, I will do that initially. This means that I am going to create several models with different depths and collect the Recall data for each model. This way, I can determine the depth that produces the model with the best Recall**.**

The decision to choose the model with the best Recall is because it is one of the main metrics to be observed in our study. As seen before, misclassifying a fraud case can be extremely damaging to the business. Therefore, we aim for a low False Negative rate, which translates to a high Recall rate**.**

```
# create models with Decision Tree

## create list to save Recall results
depth_final = []
```

```
## loop to generate models with different depths
for i in range(3, 11):
  # select seed to replicate the model
  np.random.seed(29)

  # instantiate and set hyperparameters
  ## depth = from 3 to 10
  ## criterion for defining leaves = entropy
  model_DT = DecisionTreeClassifier(max_depth=i, criterion='entropy')

  # train the model on the training data
  model_DT.fit(X_rus, y_rus)

  # make predictions
  y_pred_DT = model_DT.predict(X_test)

  # calculate Recall and add it to the list
  depth_final.append(recall_score(y_test, y_pred_DT))

# print the best recall
print('The highest Recall value found was {:.4f}.'.format(max(depth_final)))
print('Position in the list: \t
{}'.format(depth_final.index(max(depth_final))))
print('Depth: \t\t\t {}'.format((depth_final.index(max(depth_final))) + 3))

"""
The highest Recall value found was 0.8919.
Position in the list:   3
Depth:     6
"""
```

Now that we have found the best depth for maximizing Recall, we can run the model:

```
# create Decision Tree model with better depth

# select seed to replicate the template
np.random.seed(29)

# instantiate and set hyperparameters
## depth = 6
## ccriterion for defining leaves = entropy
model_DT_final = DecisionTreeClassifier(max_depth=6, criterion='entropy')

# train the model on the training data
model_DT_final.fit(X_rus, y_rus)

# make predictions
y_pred_DT_final = model_DT_final.predict(X_test)



# calculate the probability of the classes
y_prob_DT_final = model_DT_final.predict_proba(X_test)

# print the probabilities of the first 5 records
print('y_pred_DT_final \n', y_pred_DT_final[0:5])
print('\n y_prob_DT_final \n', y_prob_DT_final[0:5])

"""
```

```
y_pred_DT_final
 [0 1 0 0 0]

 y_prob_DT_final
 [[1. 0.]
 [0. 1.]
 [1. 0.]
 [1. 0.]
 [1. 0.]]
"""
```

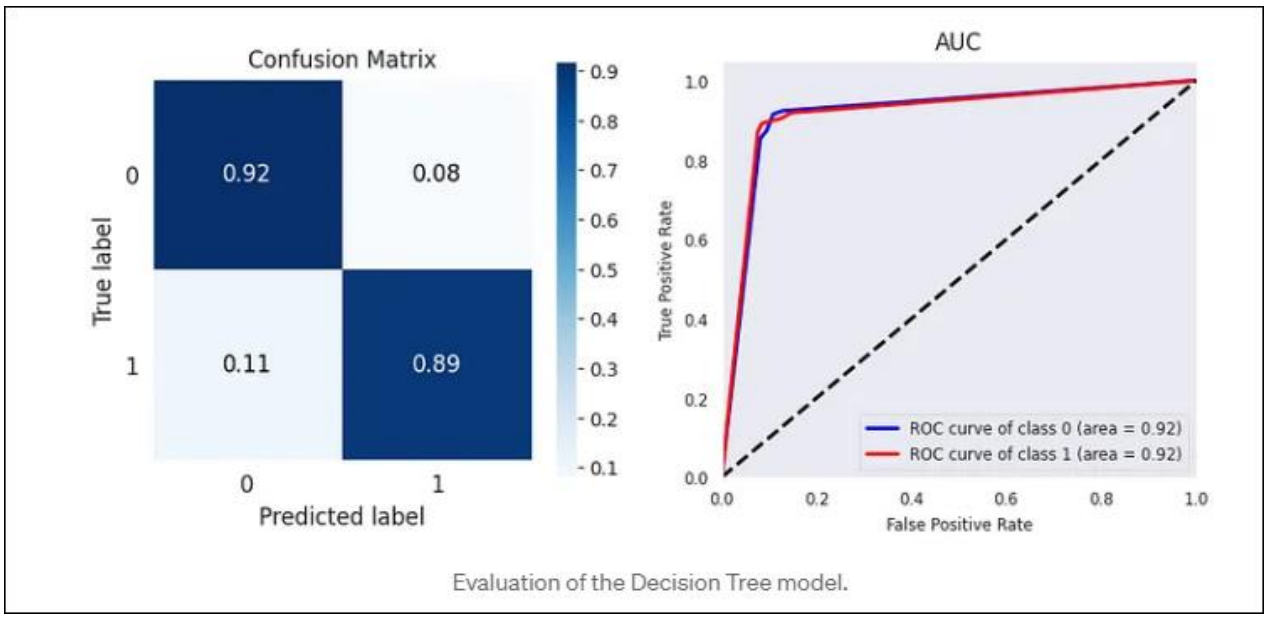
Above, we can see that the result generated by the Decision Tree is similar to that of Logistic Regression. In other words, a list is generated with the results for each transaction, indicating the probability of it being legitimate or fraudulent. In the first row, the result is [1. 0.], which means there is a 100% probability of it being a legitimate transaction and a 0% chance of it being a fraud.

9.3.2. Model Evalutation

Similar to what was done for the Logistic Regression model, let's now examine the performance metrics for the Decision Tree model created above. The report will display all the measures mentioned in the Evaluation Metrics section, but we will focus on the ones of interest for this specific study: Recall, AUC, and the confusion matrix**.**

```
# print assessment metrics report
print('Evaluation Metrics Report'.center(65) + ('\n') + ('-' * 65))
print(classification_report(y_test, y_pred_DT_final, digits=4) + ('\n') + ('-'
* 15))

# print  AUC
print('AUC: {:.4f} \n'.format(roc_auc_score(y_test, y_pred_DT_final)) + ('-' *
65))

# plot charts
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))

# confusion matrix
skplt.metrics.plot_confusion_matrix(y_test, y_pred_DT_final, normalize=True,
                                    title='Confusion Matrix',
                                    text_fontsize='large', ax=ax[0])

# AUC
skplt.metrics.plot_roc(y_test, y_prob_DT_final, title='AUC', cmap='brg',
                       text_fontsize='small', plot_macro=False,
plot_micro=False, ax=ax[1])

plt.show()

"""
                    Evaluation Metrics Report
-----------------------------------------------------------------
              precision    recall  f1-score   support

           0     0.9998    0.9151    0.9556     85295
           1     0.0179    0.8919    0.0351       148

    accuracy                         0.9151     85443
```

```
   macro avg     0.5088    0.9035    0.4953     85443
weighted avg     0.9981    0.9151    0.9540     85443

---------------
AUC: 0.9035
--------------------------------------------------------------------
"""
```

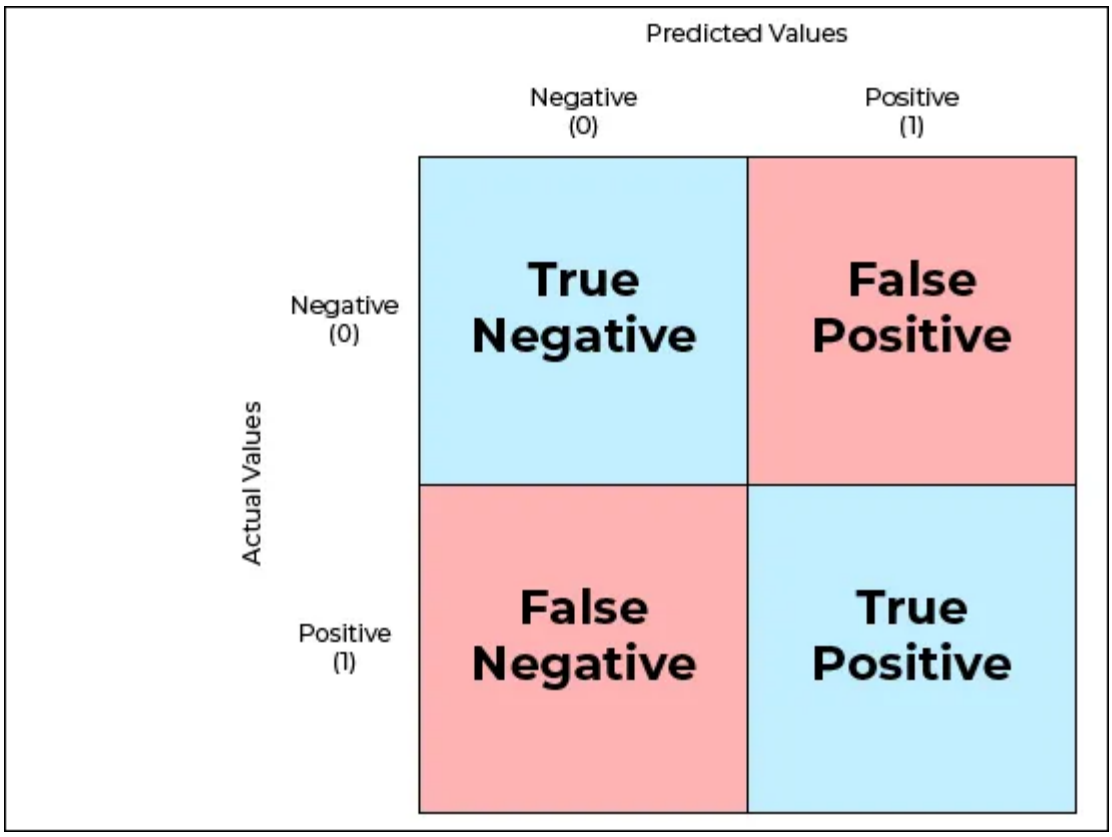


Evaluation of the Decision Tree model.

9.3.3. Results Interpretation

Confusion Matrix

The initial understanding, we need to have of the model generated from the Decision Tree is its confusion matrix, as shown in the image above and the table below, from which we can derive the following results:



Model's Correct Predictions

- 89% True Positive

A fraudulent transaction is correctly classified as fraud.

- 92% True Negative

A legitimate transaction is correctly classified as legitimate.

Model's Errors

- 8% False Positive

A legitimate transaction is incorrectly classified as fraud.

- 11% False Negative

A fraudulent transaction is incorrectly classified as legitimate.

Recall

Recall looks at all legitimate transactions and asks: How well does the model perform? The result is a value between 0 and 1, with a value closer to 1 indicating a low False Negative rate.

The generated model has a Recall of 0.8919, which means that in 89.19% of cases involving fraud, the model will be correct.

AUC

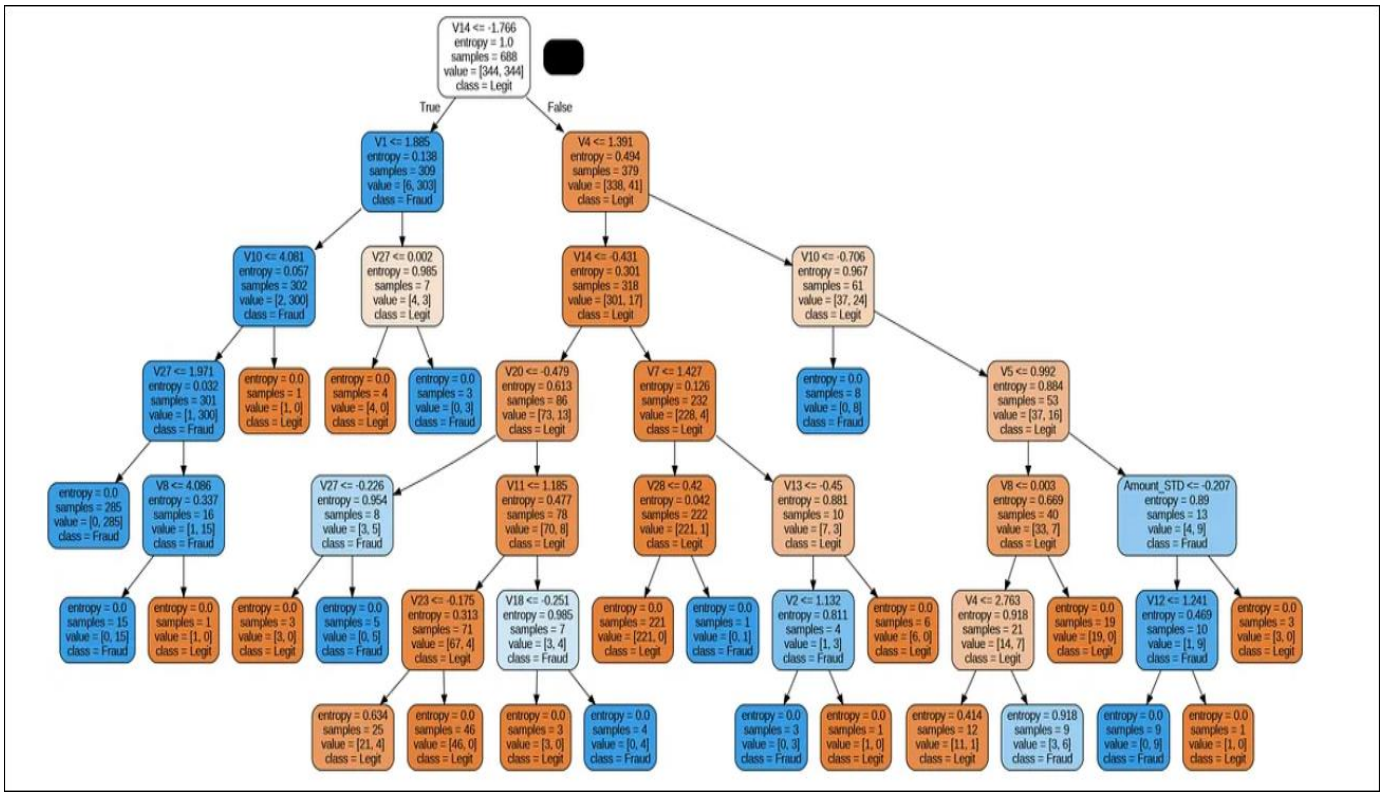
The AUC value ranges between 0 and 1, with a value closer to 1 indicating a better model.

The Decision Tree model created above has an AUC of 90.35%.

9.3.4. Visualizing the Decision Tree

To make the decision tree more understandable, I will plot the created model.

```
# generate graphical visualization of Decision Tree

# create DOT
dot = export_graphviz(model_DT_final, filled=True, rounded=True,
                      feature_names=X.columns, class_names=['Legit', 'Fraud'])

# plot tree
graph = pydotplus.graph_from_dot_data(dot)
Image(graph.create_png())
```



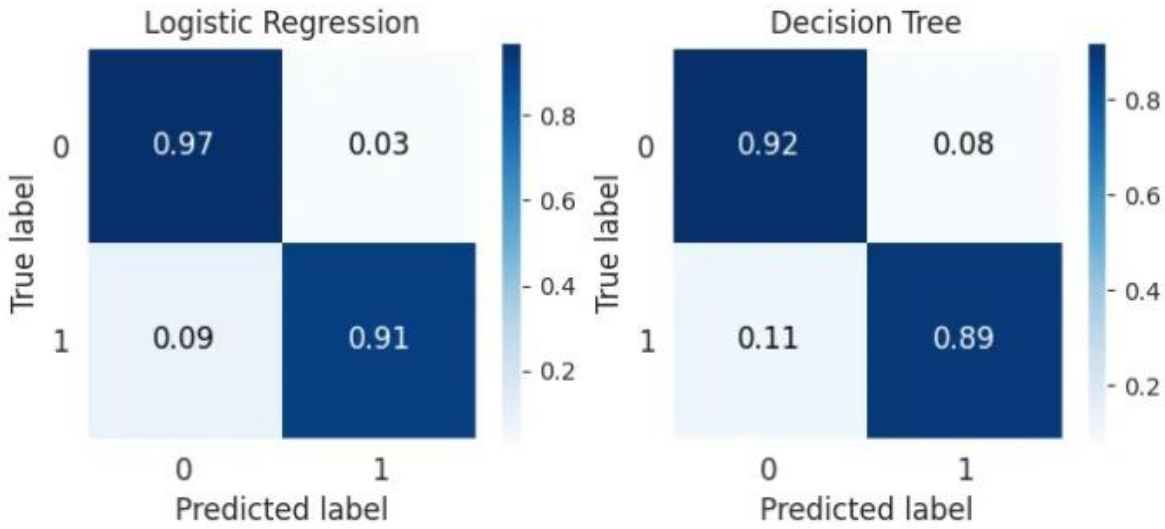
10. Logistic Regression vs Decision Tree

With two classification models created, we can compare the metrics obtained from them and determine which one better suits our problem of identifying credit card fraud.

10.1. Confusion Matrix

I will plot the results generated by the confusion matrix for both model's side by side to facilitate the comparison between them.

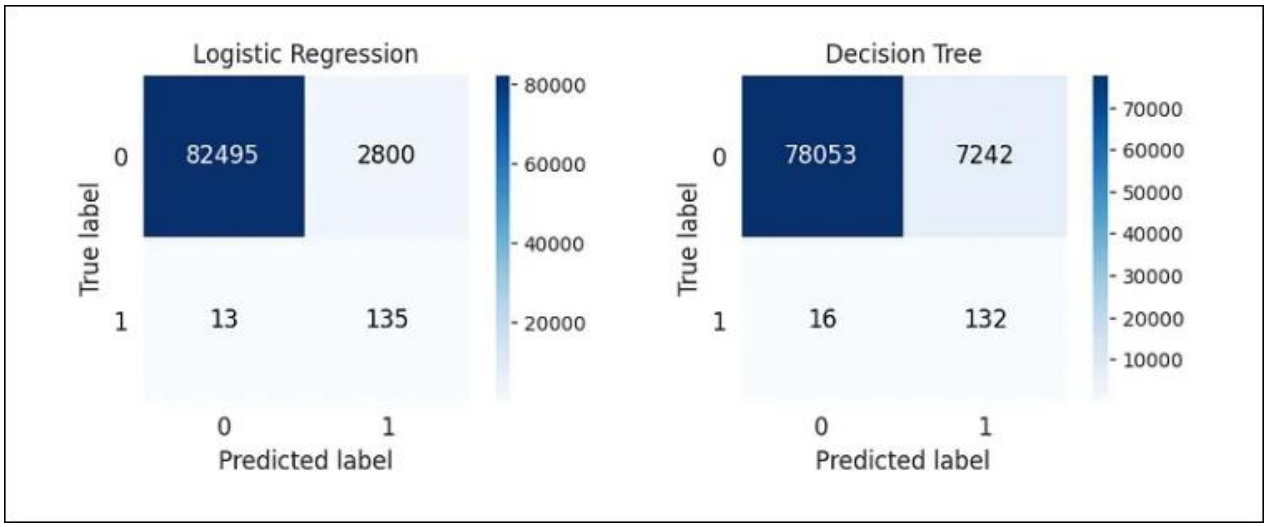
```
# plot confusion matrix
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 3))

# Logistic Regression
skplt.metrics.plot_confusion_matrix(y_test, y_pred_LR, normalize=True,
                                    title='Logistic Regression',
                                    text_fontsize='large', ax=ax[0])

# Decision Tree
skplt.metrics.plot_confusion_matrix(y_test, y_pred_DT_final, normalize=True,
                                    title='Decision Tree',
                                    text_fontsize='large', ax=ax[1])

plt.show()
```



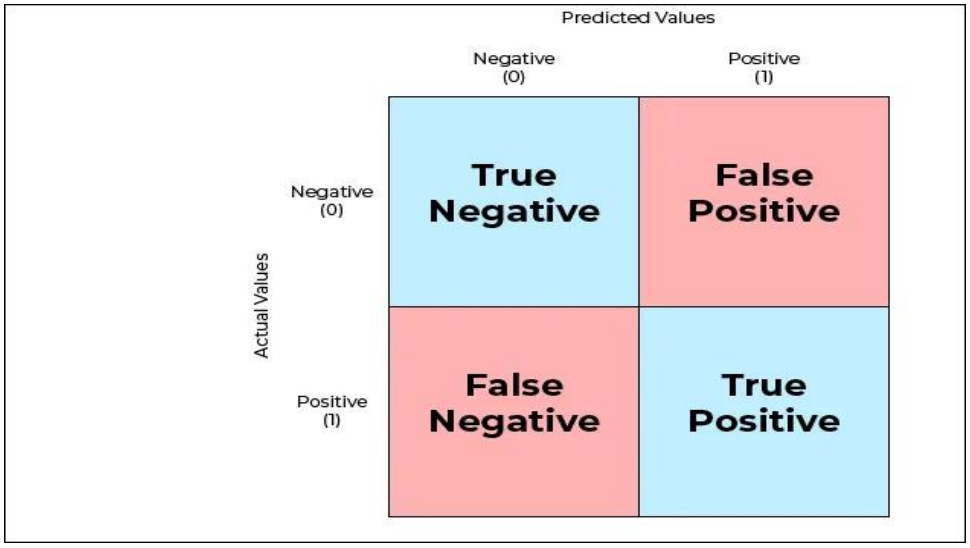
Comparison of the Confusion Matrices of the generated models.

Another interesting way to visualize the confusion matrix is by looking at the data itself. This way, we can determine how many transactions were detected as fraud, how many were legitimate, and how many the model missed.

```
# plot confusion matrix
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10, 3))

# Logistic Regression
skplt.metrics.plot_confusion_matrix(y_test, y_pred_LR,
                                    title='Logistic Regression',
                                    text_fontsize='large', ax=ax[0])

# Decision Tree
skplt.metrics.plot_confusion_matrix(y_test, y_pred_DT_final,
                                    title='Decision Tree',
                                    text_fontsize='large', ax=ax[1])

plt.show()
```



The confusion matrix provides us with four different values, arranged as follows:



Where each of these values corresponds to:

Model's Correct Predictions

- **True Positive:** A fraud transaction correctly classified as fraud**.**
- **True Negative:** A legitimate transaction correctly classified as legitimate**.**

Model's Errors

- **False Positive:** A legitimate transaction incorrectly classified as fraud**.**
- **False Negative:** A fraud transaction incorrectly classified as legitimate.

Out of 85,443 tested transactions, we can observe that regarding False Positives, Logistic Regression only made mistakes in 2,800 transactions, compared to 7,242 by the Decision Tree. This is a considerable difference in dissatisfied customers. As for correctly identifying fraud, Logistic Regression got 3 more cases right and made fewer errors in the case of False Negatives.

Therefore, analyzing the results obtained from both models, the Logistic Regression algorithm is superior in all aspects compared to the Decision Tree model.

10.2. Recall

Recall, as a metric, provides the best measure for our specific problem. The higher the Recall value, the better the model will be at identifying fraud.

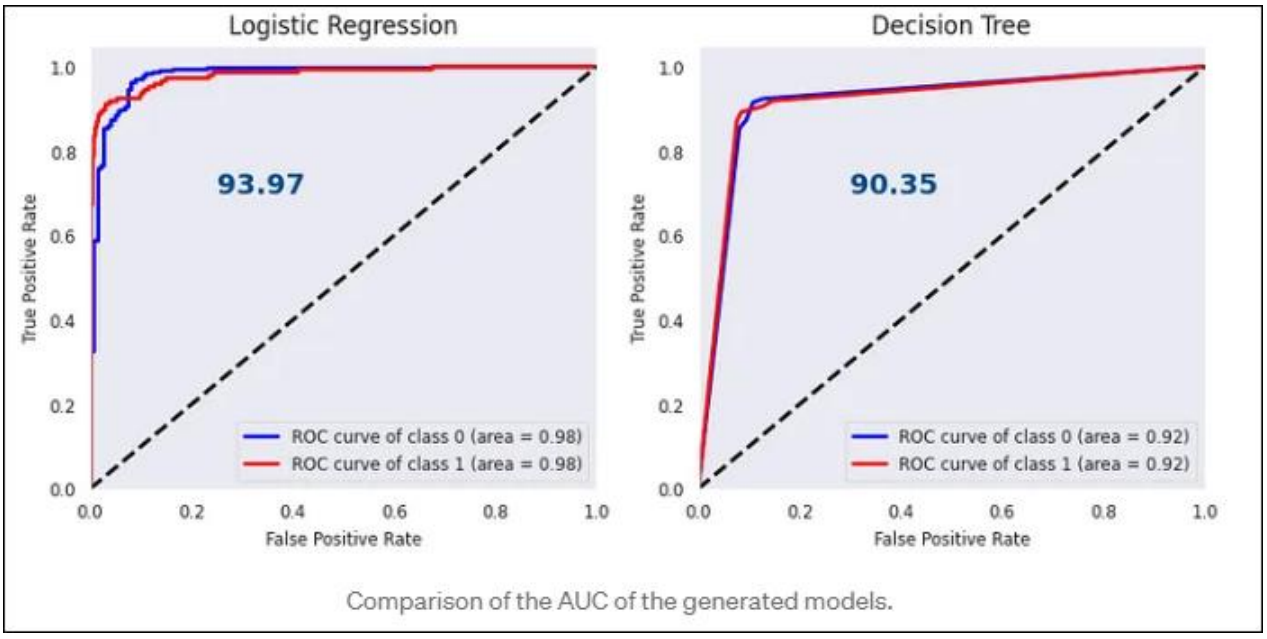
```
# print Recall results
print('RECALL'.center(30) + '\n' + ('-' * 30))

## Logistic Regression
print('Logistic Regression: \t {:.2f}%'.format((recall_score
                                                (y_test, y_pred_LR)) * 100))

# Decision Tree
print('Decision Tree: \t\t {:.2f}%'.format((recall_score
                                            (y_test, y_pred_DT_final)) * 100))

"""
            RECALL
------------------------------
Logistic Regression:   91.22%
Decision Tree:         89.19%
"""
```

The Recall value is higher in the Logistic Regression model.

10.3. AUC

Below, I present the plotted results for the AUC curve, as well as the obtained values, side by side for easy comparison of this metric.

```
# plot AUC
fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10, 4))

## Logistic Regression
skplt.metrics.plot_roc(y_test, y_prob_LR, title='Logistic Regression',
                       cmap='brg', text_fontsize='small', plot_macro=False,
                       plot_micro=False, ax=ax[0])

### print AUC value for Logistic Regression
auc_RL = (roc_auc_score(y_test, y_pred_LR) * 100).round(2)
ax[0].text(0.25, 0.7, auc_RL, fontsize=14, color='#004a8f', fontweight='bold')

## Decision Tree
skplt.metrics.plot_roc(y_test, y_prob_DT_final, title='Decision Tree',
                       cmap='brg', text_fontsize='small', plot_macro=False,
                       plot_micro=False, ax=ax[1])

### print AUC value for Decision Tree
auc_DT = (roc_auc_score(y_test, y_pred_DT_final) * 100).round(2)
ax[1].text(0.3, 0.7, auc_DT, fontsize=14, color='#004a8f', fontweight='bold')

plt.show()
```



Comparison of the AUC of the generated models.

The AUC value for Logistic Regression, 93.97%, is higher than the value of 90.35% given to the Decision Tree model.

Conclusion

This study aimed to analyze the available dataset, containing information about credit card transactions over two days, to improve the detection of fraudulent transactions. For this, an exploratory analysis of the data was made, where the dataset, its composition, and correlations between the attributes were better known.

To achieve the overall goal of this project, two binary classification machine learning models were built to predict whether a new transaction is legitimate or fraudulent. The chosen algorithms were Logistic Regression and Decision Tree, as they best fit the scope of this work.

When evaluating the performance of both models to identify the best one for predicting credit card fraud, considering Recall, AUC, and the confusion matrix, it is concluded that the Logistic Regression algorithm is superior to the Decision Tree.

Finally, it is worth noting that, despite the good results obtained, there is always room for improvement in the models. Other classification models can be used for further performance comparisons, and parameter optimization specific to each algorithm can be applied.