Credit Card Fraud Detection

Introduction

Overview of Credit Card Fraud Detection:

A **credit card** is one of the most used financial products to make online purchases and payments. Though the Credit cards can be a convenient way to manage your finances, they can also be risky. **Credit card fraud** is the unauthorized use of someone else's credit card or credit card information to make purchases or withdraw cash.

Credit card fraud detection refers to the process of identifying unauthorized or fraudulent transactions conducted using credit cards. In a scenario such as this one, it is extremely important that credit card companies are able to easily recognize when a transaction is a result of a fraud or a genuine purchase, avoiding that customers end up being charged for items they did not acquire.

Importance and Challenges:

Credit card fraud detection is vital for maintaining **the trust and security** of financial systems. The primary challenge in fraud detection is **distinguishing between legitimate and fraudulent transactions** with high accuracy. Fraudulent transactions often mimic genuine behaviour, making it difficult for traditional rule-based systems to detect them effectively. Moreover, the sheer volume of transactions processed every day necessitates the use of automated and scalable detection systems. Various Machine learning models offer promising solutions by learning from historical data to identify patterns indicative of fraud.

Objectives:

The **main objective** is to develop a prediction model that is able to learn and detect when a transaction is deriving from fraud or a genuine purchase. I intend to use some different classification algorithms and try to identify which one of them achieve the best results with our dataset.

Data understanding

Dataset Description:

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions.

In [2]:
```
# loading the dataset to a Pandas DataFrame
credit_card_df = pd.read_csv('creditcard.csv')
```

In [3]:
```
# first 5 rows of the dataset
credit_card_df.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V2
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.12853
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.16717
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.32764
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64737
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20601

5 rows × 31 columns

The dataset has the feature **time**, which shows us the seconds elapsed between each transaction and the first transaction in the dataset.

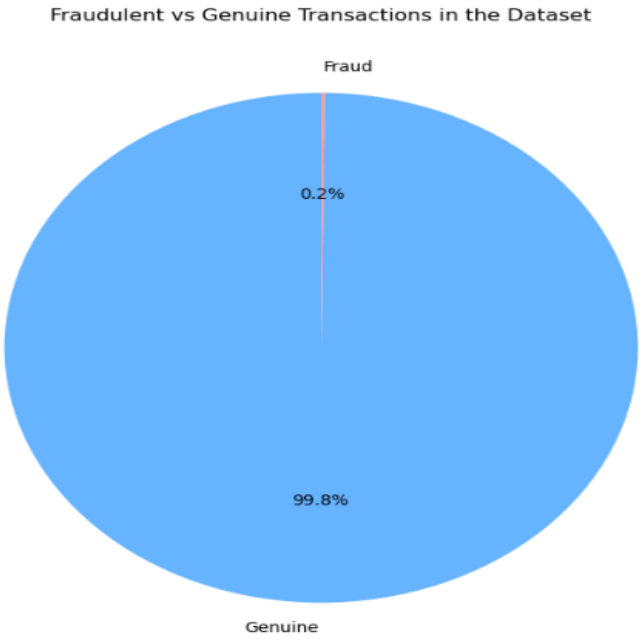
The feature **amount**, containing the transaction amount

The feature **class**, which tells us if that certain transaction is genuine or a fraud, where 1 = fraud and 0 = genuine.

Features V1, V2,... V28 are numerical input variables result of a PCA transformation whose content couldn't be displayed due to their **confidential** nature.

Class Distribution:

Class distribution refers to the proportion of different classes within a dataset. In the context of credit card fraud detection, the class distribution typically involves two classes: fraudulent transactions and non-fraudulent transactions.




```
In [14]: # distribution of legit transactions & fraudulent transactions

         credit_card_df['Class'].value_counts()
```

```
Out[14]: 0    284315
         1       492
         Name: Class, dtype: int64
```

```
0 --> normal transactions
1 --> fraudulent transactions
```

So, it seems **only 492** transactions in the dataset were fraudulent which represents only 0.2% of data, there is a huge class imbalance that we have to work on here!

Exploratory Data Analysis (EDA)

EDA is a vital step in credit card fraud detection as it helps to understand the data, identify patterns, detect anomalies, and prepare the data for modelling. It guides feature engineering, model selection, and improves the overall effectiveness and accuracy of the fraud detection system. By providing insights into the data, EDA ensures that the subsequent steps in the data science process are built on a solid foundation.

Descriptive analysis:

In credit card fraud detection, descriptive analysis is invaluable for understanding the data, identifying patterns, detecting anomalies, and guiding the entire analytical process. It ensures that subsequent modelling and predictive efforts are grounded in a thorough comprehension of the data's characteristics and challenges.

```
In [6]: # Analysis Descriptive Statistics

        credit_card_df.describe()
```

Out[6]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	..
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	..
mean	94813.859575	1.759061e-12	-8.251130e-13	-9.654937e-13	8.321385e-13	1.649999e-13	4.248366e-13	-3.054600e-13	8.777971e-14	-1.179749e-12	..
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	..
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	..
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	..
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	..
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	..
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	..

8 rows × 31 columns

Handling Missing values:

Handling missing values is essential in credit card fraud detection to ensure data quality, model reliability, and accurate predictions. It minimizes data loss, addresses class imbalance, reduces bias, and enhances the overall robustness and interpretability of the fraud detection model. Proper imputation techniques are crucial for leveraging the full potential of the dataset and building effective predictive models.

```
In [7]: # check the number of missing values in each column

        credit_card_df.isnull().sum()
```

```
Out[7]: Time    0
        V1      0
        V2      0
        V3      0
        V4      0
        V5      0
        V6      0
        V7      0
        V8      0
```

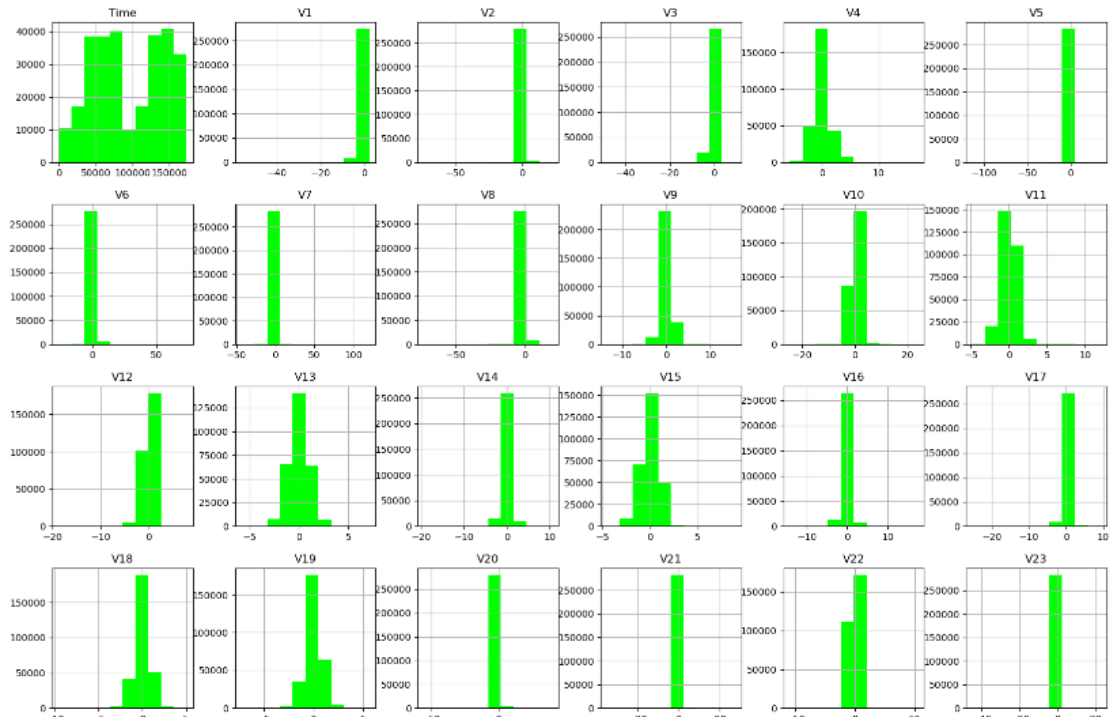
We can see that there are no missing values in any of the columns. Hence, there is no problem with null values in the entire dataset.

Histogram visualization:

Histogram visualization is a crucial tool in data analysis and fraud detection, offering insights into data distribution, outliers, data quality, and more. It aids in feature engineering, model selection, and effective communication of data patterns, ultimately supporting the development of robust fraud detection models.

```
In [9]: credit_card_df.hist(figsize = (20,20) , color = 'lime')
        plt.show

Out[9]: <function matplotlib.pyplot.show(close=None, block=None)>
```



Time: The Time feature shows a bimodal distribution, indicating two peaks which could correspond to different periods of activity during the day or other temporal patterns.

V1 to V28: These features (V1, V2, ..., V28) are the result of a PCA transformation applied to the original dataset. Most of these features show distributions that are centred around zero with varying degrees of spread. This suggests that these features have been normalized and centred as part of the preprocessing step.

Amount: The Amount feature has a right-skewed distribution. Most transactions have a relatively small amount, with a long tail extending towards larger amounts. This is typical for transaction data, where small transactions are more common.

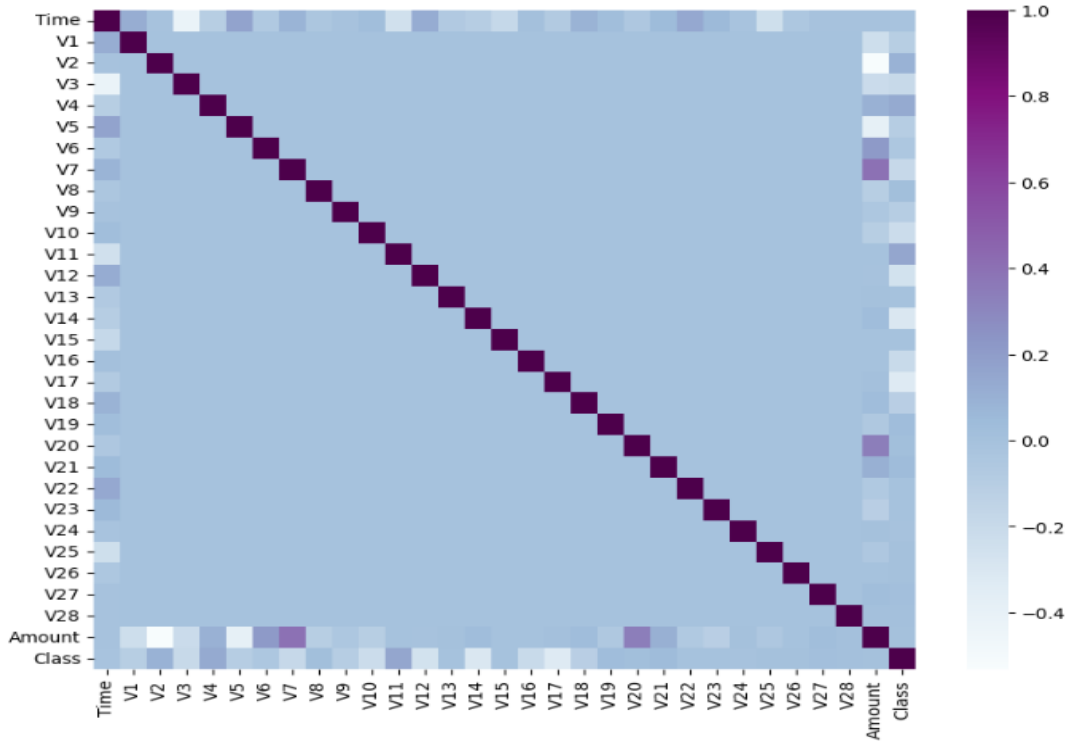
Class: The Class feature, which likely indicates whether a transaction is fraudulent (1) or not (0), shows a very imbalanced distribution. The majority of the transactions are non-fraudulent, with only a small proportion being fraudulent. This class imbalance is a common challenge in fraud detection tasks.

Correlation Heatmap:

Correlation is a statistical measure that expresses the extent to which two variables are linearly related. In machine learning, understanding and using correlation is crucial for various stages of data preprocessing, feature selection, and model evaluation

The **correlation heatmap** is a useful step in understanding the relationships between features and guiding the feature selection and modelling process in credit card fraud detection.

```
In [10]: plt.figure (figsize = (10,8))
         corr = credit_card_df.corr()
         sns.heatmap(corr,cmap = 'BuPu')
Out[10]: <Axes: >
```



Diagonal Line:

- The diagonal line of dark purple squares represents the correlation of each feature with itself, which is always 1.

PCA Features (V1 to V28):

- The PCA-transformed features generally show low correlation with each other. This is expected, as PCA aims to create orthogonal (uncorrelated) components.
- Some off-diagonal squares show light shades, indicating slight correlations between certain PCA features, but these are generally weak.

Time and Amount Features:

- The Time feature shows very low to no correlation with other features.
- The Amount feature shows very low to no correlation with most PCA features, but there might be slight correlations with a few PCA components.

Class Feature:

- The Class feature (indicating fraudulent vs. non-fraudulent transactions) shows low correlations with most other features. However, any non-zero correlation might still be significant for classification tasks.

Prepare Data for Modelling

Removing Time Features:

For this project, we won't be using the **time** attribute, so we will remove it.

```
[19]: credit_card_df = credit_card_df.drop(columns = ['Time'], axis = 1)
      credit_card_df
```

Split the Dataset into Train and Test:

We will also use StandardScaler () to put all the data into the same scale, avoiding bias for a certain attribute when trying to predict our target variable, which is **Class**.

```
: X = credit_card_df.drop(columns=['Class'], axis=1)
  y = credit_card_df.Class
```

Scale the Dataset:

Normalizing 'Amount' feature with StandardScaler, separately on each set, in order to avoid **data leakage**.

```
[26]: # Scaling data on the training set
      scaler = StandardScaler()
      train_x['Amount'] = scaler.fit_transform(train_x.Amount.values.reshape(-1,1))
      train_x
```

```
[27]: # Scaling data on the testing set
      scaler = StandardScaler()
      test_x['Amount'] = scaler.fit_transform(test_x.Amount.values.reshape(-1,1))
      test_x
```

SMOTE Analysis:

Now, considering that we're dealing with **imbalanced** data, we must apply **SMOTE** in order to **oversample** our fraudulent data.

SMOTE will synthetically **generate more** samples of fraudulent data based on the frauds that we already have in the original dataset.

```
n [28]: y.value_counts() # 0 = Genuine Transactions | 1 = Fraud
ut[28]: 0    284315
        1       492
        Name: Class, dtype: int64
```

```
n [29]: from imblearn.over_sampling import SMOTE
        train_x, train_y = SMOTE().fit_resample(train_x,train_y) # Reshaping data
```

```
n [30]: train_y.value_counts()
ut[30]: 0    199032
        1    199032
        Name: Class, dtype: int64
```

Now we have a 50 | 50 data balance between genuine and fraudulent transactions

Model Selection

Model building is a pivotal step in the process of credit card fraud detection. This involves using statistical techniques and machine learning algorithms to create predictive models that can identify potentially fraudulent transactions. Train the model with the algorithms such as:

- Random Forest
- Decision Tree
- Ada Boost
- Gradient Boosting

Model Evaluation

Random forest:

Random Forest is a powerful and versatile machine learning algorithm that is particularly well-suited for classification problems such as credit card fraud detection. It combines the predictions of multiple decision trees to improve accuracy and robustness.

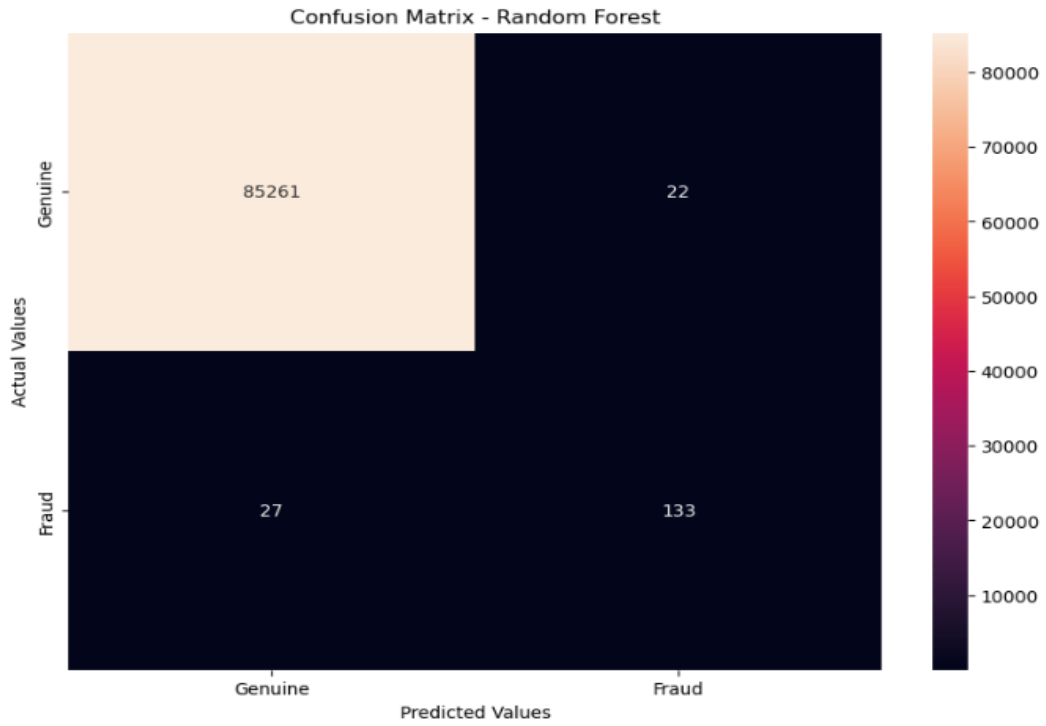
In [30]:
```
# Printing Evaluation Metrics for Random Forest
metrics = [['Accuracy',(accuracy_score(test_y, y_predictions_rf))],
                       ['Precision',precision_score(test_y, y_predictions_rf)],
                       ['Recall', recall_score(test_y, y_predictions_rf)],
                       ['F1_score',f1_score(test_y, y_predictions_rf)]]
metrics_df = pd.DataFrame(metrics, columns = ['Metrics', 'Results'])
metrics_df
```

Out[30]:

	Metrics	Results
0	Accuracy	0.999415
1	Precision	0.852564
2	Recall	0.831250
3	F1_score	0.841772

Confusion Matrix Output:

- The model correctly identified 133 fraudulent transactions.
- The model correctly identified 85,261 genuine transactions.
- The model incorrectly identified 22 genuine transactions as fraudulent.
- The model incorrectly identified 27 fraudulent transactions as genuine.



Decision Tree

A decision tree is a popular machine learning algorithm used for classification and regression tasks. It is particularly useful in credit card fraud detection due to its simplicity, interpretability, and effectiveness in handling both numerical and categorical data.

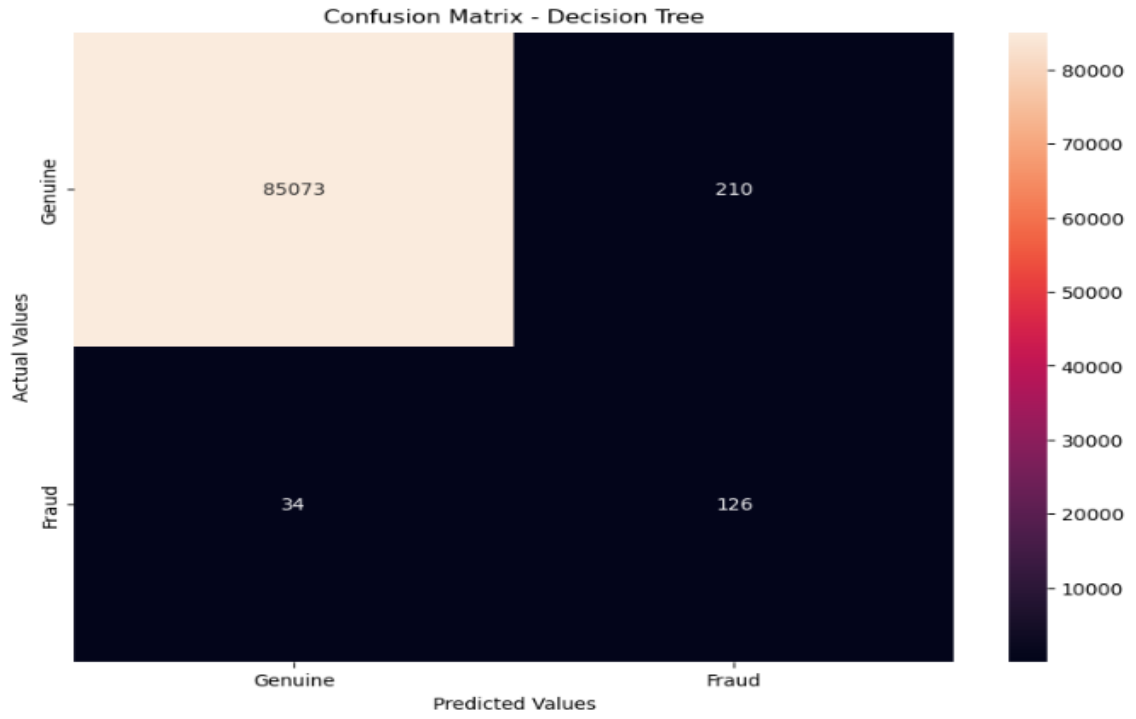
In [34]:
```
# Printing Evaluation Metrics for Decision Tree
metrics_df = [['Accuracy',(accuracy_score(test_y, y_predictions_dt))],
                          ['Precision',precision_score(test_y, y_predictions_dt)],
                          ['Recall', recall_score(test_y, y_predictions_dt)],
                          ['F1_score',f1_score(test_y, y_predictions_dt)]]
metrics_df_dt = pd.DataFrame(metrics_df, columns = ['Metrics', 'Results'])
metrics_df_dt
```

Out[34]:

	Metrics	Results
0	Accuracy	0.997144
1	Precision	0.375000
2	Recall	0.787500
3	F1_score	0.508065

Confusion Matrix Output:

- The model correctly identified 126 fraudulent transactions.
- The model correctly identified 85,073 genuine transactions.
- The model incorrectly identified 210 genuine transactions as fraudulent.
- The model incorrectly identified 34 fraudulent transactions as genuine.



Ada boost

AdaBoost, short for Adaptive Boosting, is an ensemble learning technique that can be particularly effective for binary classification problems, including credit card fraud detection. It combines multiple weak learners to form a strong learner, improving the model's accuracy.

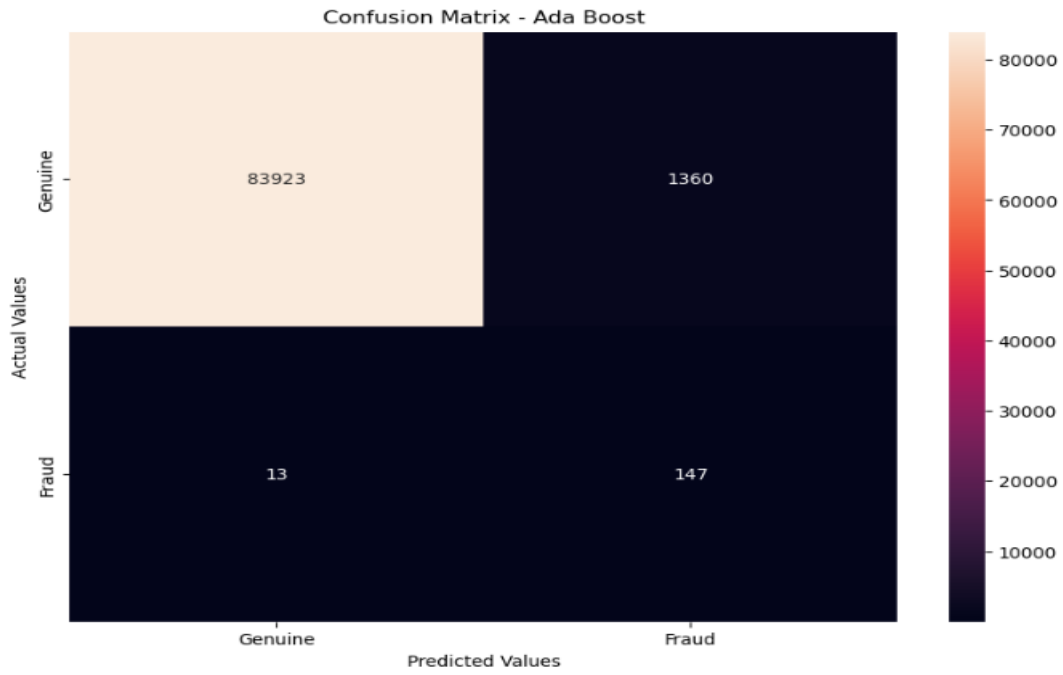
In [36]:
```
# Printing Evaluation Metrics for Ada Boost
metrics_ab = [['Accuracy',(accuracy_score(test_y, y_predictions_ab))],
                          ['Precision',precision_score(test_y, y_predictions_ab)],
                          ['Recall', recall_score(test_y, y_predictions_ab)],
                          ['F1_score',f1_score(test_y, y_predictions_ab)]]
metrics_df_ab = pd.DataFrame(metrics_ab, columns = ['Metrics', 'Results'])
metrics_df_ab
```

Out[36]:

	Metrics	Results
0	Accuracy	0.983931
1	Precision	0.097545
2	Recall	0.918750
3	F1_score	0.176365

Confusion Matrix Output:

- The model correctly identified 147 fraudulent transactions.
- The model correctly identified 83,923 genuine transactions.
- The model incorrectly identified 1360 genuine transactions as fraudulent.
- The model incorrectly identified 13 fraudulent transactions as genuine.



Gradient Boosting

Gradient Boosting is a powerful ensemble machine learning technique that combines the predictions of several base models (usually decision trees) to produce a strong predictive model. It is particularly effective for tasks like credit card fraud detection due to its ability to handle complex data structures and uncover intricate patterns.

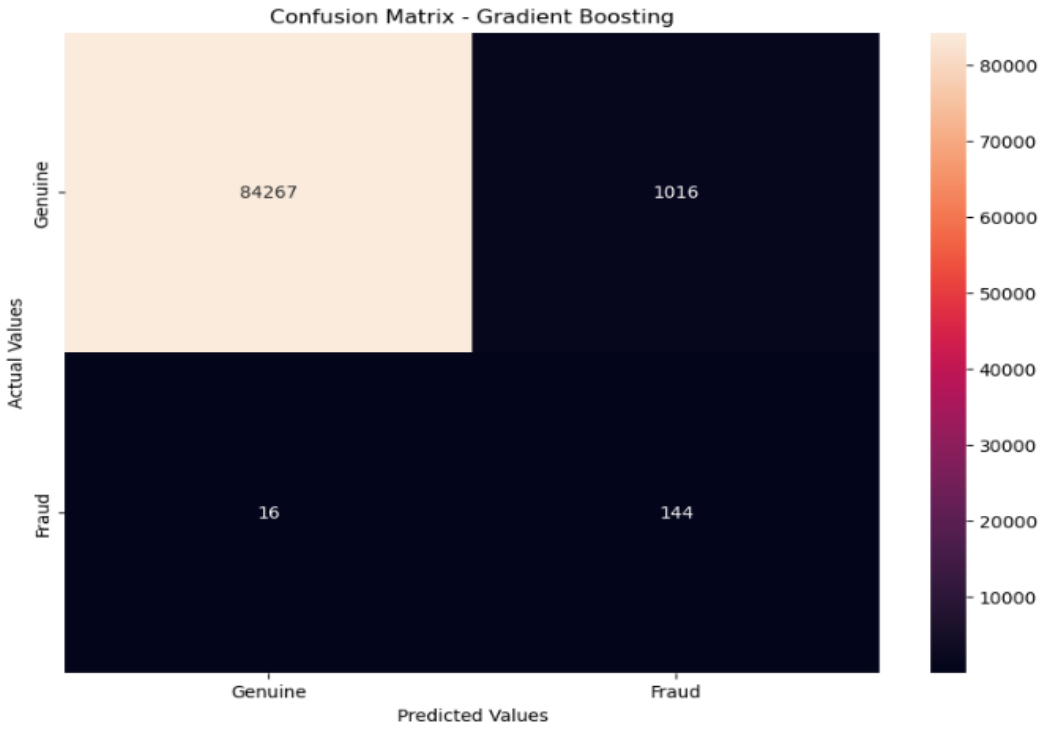
In [38]:
```
# Printing Evaluation Metrics for Gradient Boosting
metrics_gb = [['Accuracy',(accuracy_score(test_y, y_prediction_gb))],
                          ['Precision',precision_score(test_y, y_prediction_gb)],
                          ['Recall', recall_score(test_y, y_prediction_gb)],
                          ['F1_score',f1_score(test_y, y_prediction_gb)]]
metrics_df_gb = pd.DataFrame(metrics_gb, columns = ['Metrics', 'Results'])
metrics_df_gb
```

Out[38]:

	Metrics	Results
0	Accuracy	0.987922
1	Precision	0.124138
2	Recall	0.900000
3	F1_score	0.218182

Confusion Matrix Output:

- The model correctly identified 144 fraudulent transactions.
- The model correctly identified 84,191 genuine transactions.
- The model incorrectly identified 1092 genuine transactions as fraudulent.
- The model incorrectly identified 16 fraudulent transactions as genuine.



ROC AUC Scores:

```
In [41]: # Printing ROC AUC scores
         from sklearn.metrics import roc_auc_score
         print('Random Forest ROC AUC Score: ', (roc_auc_score(test_y, y_predictions_rf) * 100).round(2))
         print('Decision Tree ROC AUC Score: ', (roc_auc_score(test_y, y_predictions_dt) *100).round(2))
         print('Ada Boost ROC AUC Score: ', (roc_auc_score(test_y, y_predictions_ab) * 100).round(2))
         print('Gradient Boost ROC AUC Score: ', (roc_auc_score(test_y, y_prediction_gb) * 100).round(2))

         Random Forest ROC AUC Score:  91.55
         Decision Tree ROC AUC Score:  89.25
         Ada Boost ROC AUC Score:  95.14
         Gradient Boost ROC AUC Score:  94.4
```

Conclusion

```
In [40]: # Counting how many fraudulent and how many genuine transactions we have on the testing set
         test_y.value_counts()

Out[40]: 0    85283
         1      160
         Name: Class, dtype: int64
```

When we work with a **machine learning model,** we must always know for a fact what it is that we're trying to get from that model.

In this project, our goal is to **detect fraudulent transactions** when they occur, and the model who best performed that task was the **Ada Boost Classifier** with a **recall of 90.62%**, correctly detecting 145 fraudulent transactions out of 160. However, it is also important to note that the Ada Boost classifier had the **biggest number of false positives**, that is, 1255 genuine transactions were mistakenly labelled as fraud, that's 1.54% of all genuine transactions.

A genuine purchase being incorrectly identified as a fraud **could be a problem**.

In this scenario it is necessary to understand the business and make a few questions such as:

? how cheap would a false positive be?

? Would we keep the Ada Boost Classifier with the **best performance in detecting frauds**; while also detecting a lot of false positives or should we use the Random Forest Classifier, who also performed pretty well identifying frauds (82.50% recall) and reduced the number of false positives (0.02% of genuine transactions flagged as fraud). But that would also imply in a larger number of fraudsters getting away with it and customers being mistakenly charged...

These questions and a deeper understanding of how the business works and how we want to approach solving a problem using machine learning are fundamental for a decision-making process to choose whether or not if we're willing to deal with a larger number of false positives to detect the largest number of frauds as possible.