**Fraud Detection System**

I decided to create a fraud detection system in order to use data analytics to tackle real world problems such as credit card fraud. As technology evolves so does the way in which fraud is carried out and I was interested in how machine learning can adapt to patterns that are too complex or dynamic for rule-based systems to tackle. This project also acts as a conduit for me to transition into a career in Machine learning, Data analysis and Artificial intelligence.

This is the workflow I will be following:

Model training

Train-Test-Split

Data analysis

Data processing

Credit card data

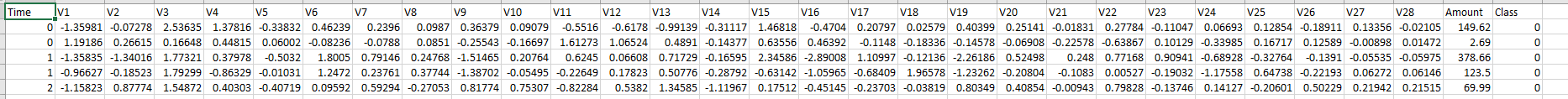
Evaluation

My first steps were to gather data from previous fraud detection system in order to understand what fraud looks like, differentiate what fraud looks like from normal behavior (patterns) and measure performance to test if my system is working as expected. I am sourcing my data from a credit card fraud detection data set from Kaggle. My objective is to find a realistic set of credit card or transaction data that includes both normal and fraudulent record so that I can analyze patterns and build detection logic. I need a dataset that includes these main features:

* Transaction ID (to track individual payments)
* Timestamp (for time-based fraud patterns)
* Amount (transaction value)
* Fraud label (base truth 1 = fraud, 0 = not fraud)

The data set from this site is ideal because it contains:

* A sample size that is large enough to simulate real-world load as it has records of 284,807 transactions
* Labeled fraud cases which allows me to validate detection accuracy
* Class imbalance – of the 284,807 transactions there were 492 frauds. This is ideal because it mimics real business problems as it is hard to detect rare frauds.



This is a snippet of the data set

My next steps were to clean and understand the dataset that I downloaded from Kaggle.

**Technical analysis**

**Importing dataset**

I first imported the dataset into my main program. The IDE I am using, VS code, does not have the pandas library pre-installed so to solve this I simply installed pandas via the terminal and imported it as pd so that I don’t have to type panda every time I use it. Panada is a library that is used for data analysis and manipulating tables/Data frames. Once done I was able to read and load the CSV file (my dataset) and turn it into a Data frame, df, using pandas “.read\_csv()” function. After this I checked the data health and info.

**Visualizing the data**

Similar to when I was importing the dataset, I first installed a matplotlib (mpl), a data visualization library, and seaborn (sb), a python visualization library built on top of matplotlib via the terminal. Through these I was able to create a bar chart that visualizes the class imbalance, a histogram of transaction amounts and a comparison between fraud and non-fraud occurrences.

**Building a fraud detection model**

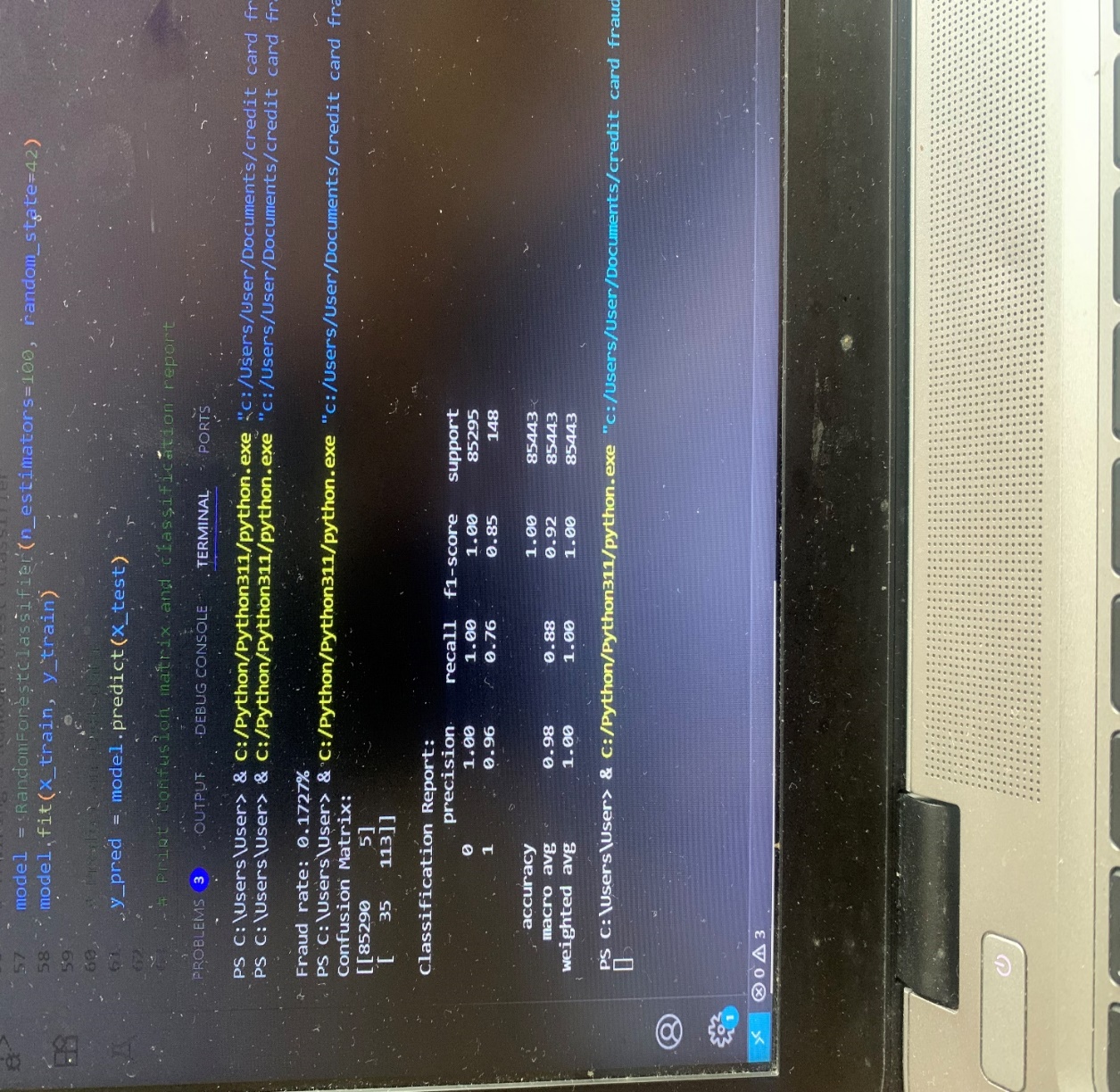
My goal in this section was to train a machine learning model to predict fraud based on transactional data. I first prepared the data to get it into the right shape and format in order for the machine learning model to be able to learn from it. I did this by separating the input and output values. This is required because a machine learning model learns pattern by looking at how inputs (features) relate to the outputs (labels/target). For example, if you want to predict house prices, the features/input you would consider are number of bedrooms, size of the house and age of the house. Using those features, one can make a calculated predication of the value of the home. Similarly to this, I have chosen amount as my input/feature and the target is to predict the class/fraud or not. I also split the data set into testing and train sets, splitting 70% of the data into training set and 30% in testing set. I split the data so that it prevents the machine learning model from memorizing the data. So, the other 30% of the data is unseen to the machine learning model however will also allow me to judge how accurate the model is. I installed scikit-learn via the terminal. This is a module that gives you tools to spilt data, train models and measure accuracy.

**Choosing the best machine learning model and implementing**

After doing research on the different types of machine learning models I decided to use a Random Forest Model. Random Forest Models works well with imbalanced data which is suitable for the fraud detection data set I am using as the data is very imbalanced, having a fraud rate of 0.17%. This is ideal for my project unlike a logistic regression model which assumes linear patterns and needs balances data. Random Forest models also combine multiple trees allowing it to generalize better and produce a more accurate prediction. Unlike models like logistic regression or Support Vector Machines, Random Forest Model are not affected by values on different scales. Factoring these as well as the fact that RFMs are easy to use as it automatically handles complex relationships between variables, I decided to use a Random Forest Model. To implement this, I used scikit-learn’s “RandomForestClassifier” class tool. Lastly, to evaluate the performance of my machine learning model using scikit-learn’s “classification\_report” and “confusion\_matrix” functions which summarizes key metrics for classification and produces a table which shows where the model was right and wrong.

**Running the program and checking accuracy**

When run there were two outputs, a confusion matrix and a classification report in the form of a table. Below is an image of what was outputted:



To read the confusion matrix you can visualize it like this:

|  |  |  |
| --- | --- | --- |
|  | Predicted not fraud | Predicted fraud |
| Actual not fraud | 85290 (True-Negative) | 5 (False-Positive) |
| Actual fraud | 35 (False-Negative) | 113 (True-Positive) |

From this we can see that:

* 85,290 non-fraud transactions were correctly identified as not fraud
* 113 frauds were correctly detected
* 5 people were wrongly flagged for fraud even though they were innocent (false alarms)
* 35 real frauds were missed by the model

The classification report tells me:

* the model was 100% precise as all predicted non fraud cases were correct
* the model correctly identified 100% of all actual non-fraud transactions, meaning it didn’t miss any legitimate transactions when checking for fraud
* the model achieved a perfect F1-score of 1.00 for non-fraud cases, meaning it balanced precision and recall perfectly when identifying legitimate transactions.
* Of the frauds that were predicted, 96% were correct
* It caught 76% of actual fraud cases
* The model achieved an F1-score of 0.85 for fraud detection, indicating strong performance overall, but it still missed some fraudulent transactions.

After successfully producing a basic machine learning model and understanding its performance constraints and limits, I decided to begin improving its flaws iteratively by perhaps using alternative machine learning models.

To further level up my project and the information that is being outputted easy to understand I have decided to add a Streamlit dashboard to help visualise the different metrics and measurements such as fraud rate. I also wanted my product to feel more polished and interactive. This will make it easier to:

* See overall fraud statistics
* Track how many transactions are fraud against how many are not
* Visualise model results
* Explore suspicious transactions

I decided to do some research into what information a dashboard should usually include and also used other dashboards as a framework to help me decide what I will include in mine. To make the information easily accessible readable and understandable to users I decide I need to include a component that shows the total transactions made, the percent of fraud in the data, charts that illustrate trends, model metrics and perhaps even an interactive fraud table. One vital feature I decided to add after probing different dashboards in a filter which helps the users to easier access and traverse the dashboard. A filter helps control the output of information.

**Creating the dashboard**

This is the structure I will be following when creating the dashboard:

Show prediction

Run model

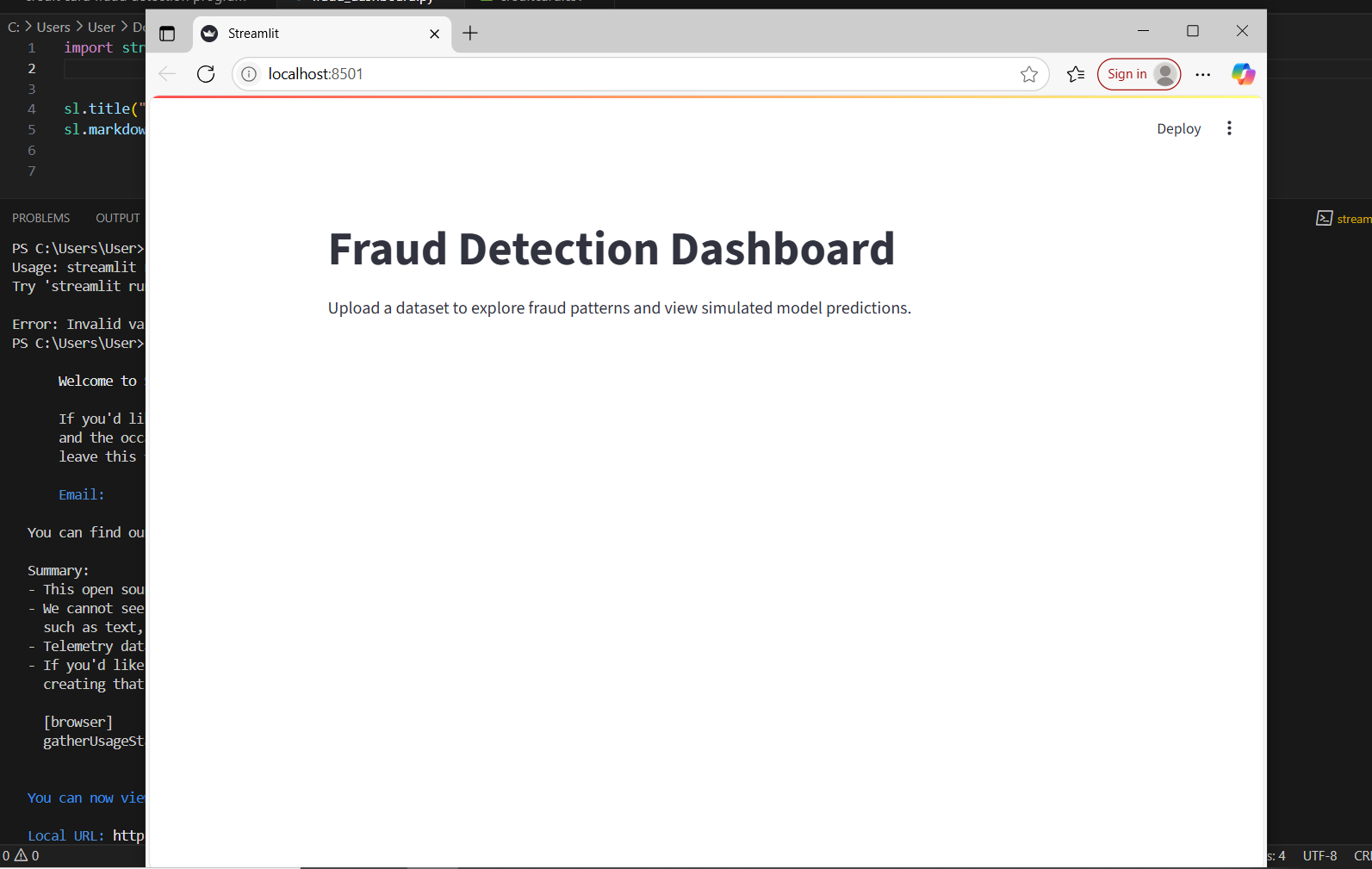
Read CSV

Upload CSV

Layout

To start off I installed streamlit, a python library that lets you build interactive web apps and dashboards without the need for HTML or Javascript, and I imported it as “sl”. Streamlit will allow me to visualise the data in the form of charts and tables. It will also help make my machine learning model interactable to users, including allowing them to upload CSVs and analyse them. I decided to use streamlit for the dashboard instead of HTML, Javascript etc because although I am able to use HTML and Javascript, after making various projects including a platformer game I understandably excel in python and this will help speed up the process of this project. I decided to prioritise time because at the time of working on this I was working on another project. Additionally, from what I have been reading, streamlit seems to be the best option when working with analytic dashboards and machine learning.

To help make my code more organised and readable I created a new file to put all the code regarding the dashboard. I first started designing the layout of the dashboard, which included the title and a summary of what the model does and how user can operate it. After running the program to make sure everything is operating as expected this was the following output:



After confirming everything was working as intended, I started working on adding a section that handles CSV file uploading.

uploaded\_file = sl.file\_uploader("Upload your CSV file", type=["csv"])

this line creates a file upload widget/function using the streamlit “file\_uploader(…)” function and ensures the type is a CSV file

df = pd.read\_csv(uploaded\_file)

Once the CSV file is uploaded this reads the file and puts it into a data frame

My next steps were to load my model to the dashboard and run it. Because of the nature of a trained machine learning model, if the program stops running or the command is terminated the model is removed from memory. When training a model, you are essentially creating an object in memory, therefore this object only exists temporarily in memory and every time you run the program the model must be retrained. Because I want to reuse that trained object in the dashboard, I must serialize it, turning it into a file, using “joblib”. “joblib” is a python library that is used to save and load large python objects efficiently. This removes the need for the model to be retrained every time a user opens the dashboard. This improves performance, efficiency and speed as the it removes the time needed to retrain the model. I was originally going to use “pickle” instead of “joblib” as I have experience with pickle when designing a platformer game but concluded on using “joblib” as it is faster and more efficient for models.

To save the trained machine learning model I did:

import joblib

joblib.dump(model, "fraud\_model.pkl")

#The line below takes the trained model and saves it into a file called fraud\_model.pkl. “pkl” is short for pickle which is pythons’ way of storing objects. “joblib.dump” is a function that takes a python object and saves it to a file on disk. It does this by writing trees, parameters and settings of my model into a “. pkl” file.

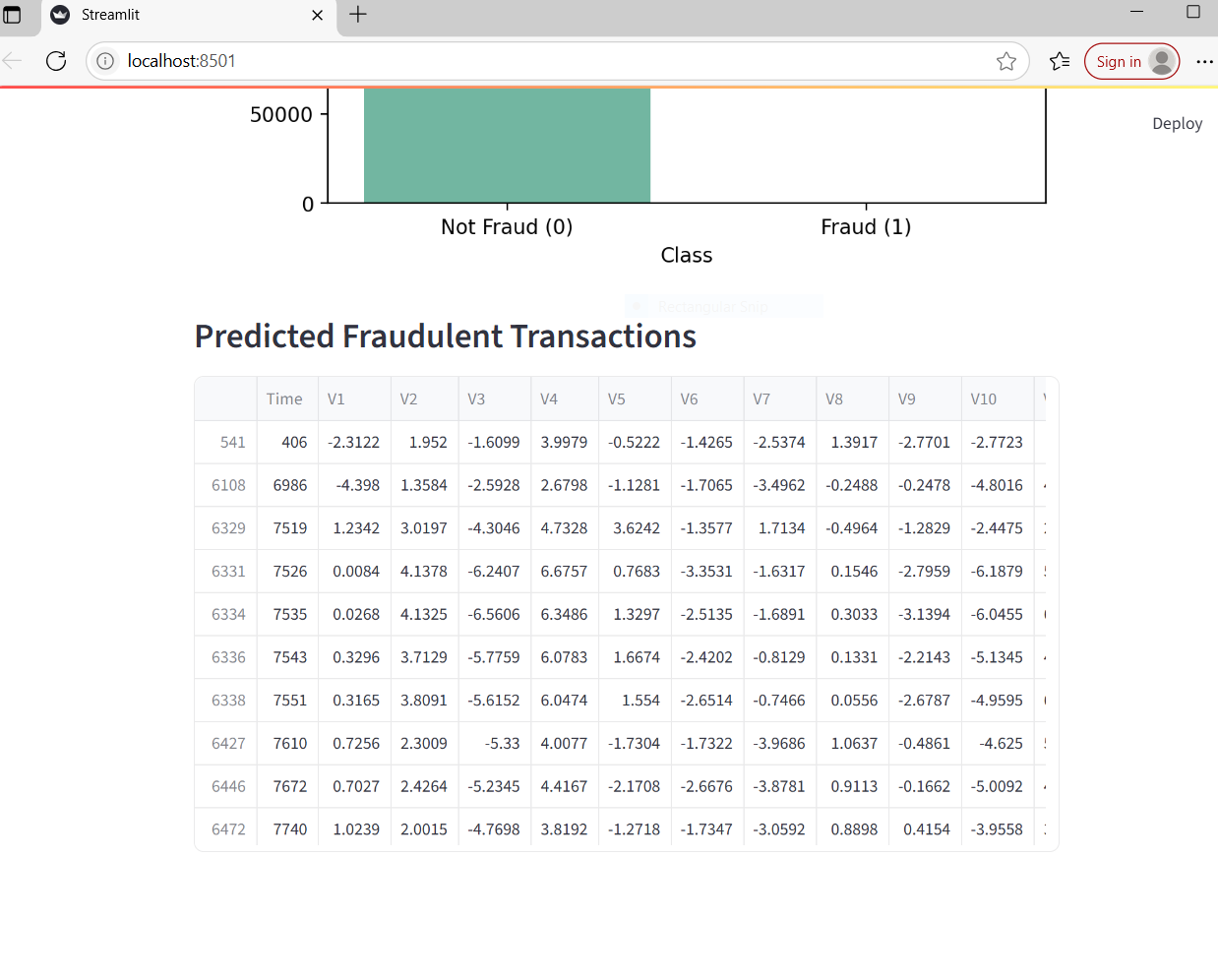
After successfully saving my model, I simply loaded the model into my dashboard:

model = joblib.load("fraud\_model.pkl")

predictions = model.predict(X.fillna(0))

df[“Predicted Fraud”] = predictions

Over here I am loading the file in which I saved my model and calling the predict function. Often datasets have missing values so I use X.fillna(0) which replaces any missing values in X (the input data from when I split the input and output) with 0. This must be done otherwise the “.predict()” function may crash. This also saves all of the prediction values to a variable called “predictions”. The last line of code is used to add a new column called Predicted fraud which contains the predicted values stored in “predictions” to the data frame. Once done I can display the data frame to the user using streamlit’s “st.dataframe(…)” function.



This Is what is displayed after running. This displays a data frame of the predicted fraudulent transactions.

sl.dataframe(df[df[“Predicted Fraud”] == 1].head(10))

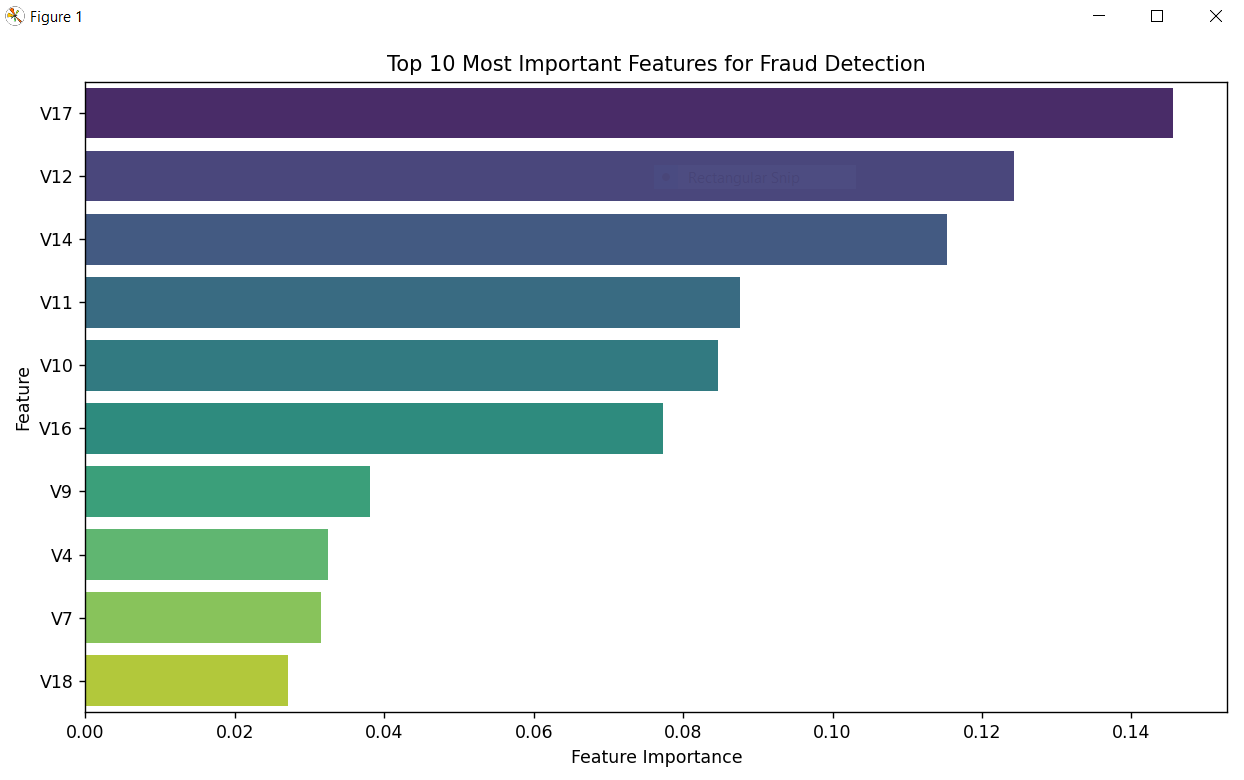
I used this to filter the predicted fraudulent transactions to display only them to the user. df[“Predicted Fraud”] selects the “Predicted Fraud” column that I added to the data frame. This acts as a filter. It is then wrapped in the outside df[…] which applies the filter to the entire data frame. It returns only the rows where the condition is True/1. “.head(10)” displays only the first 10 rows of the filtered data frame. Finally, “sl.dataframe” displays the subset. I have restricted it to 10 rows because it takes a while to load.

To make my page more detailed I added illustrations of the user’s data set which was simple as I had already performed it in the first part of the project. So, I simply had to copy it over to this part of the project.

Since I am tailoring this project towards data analyses, I decided to do a section on feature importance and business insights.

**Feature importance**

Feature importance is a way in which to identify which input variables have the most influence on the machine learning model’s prediction. In my case, since I am tailoring my model to detect fraud, the input variables can be for example transaction amounts or specific behavioural patterns. Doing this is critical in high stakes application such as finance, as it allows data analyst’s to further understand the decision-making process of machine learning models. This in turn allows data analysts to prioritise which features to monitor more closely, potentially improving the efficiency in which fraud is detected. I am using feature importance as a way to translate the model’s complex logic into actionable insights that can help support business decisions.



This figure illustrates the 10 most important features of my model. The model identified V17, V12 and V14 as the 3 most important features that influenced fraud predictions. These features have been anonymized because the data used in training my model was downloaded from Kaggle however in professional environments understanding these behavioural patterns can help focus priorities.

**Business insights**

**Business challenge**

In 2025, the challenge of credit card fraud prevention for businesses is more relevant than ever. According to the Business Wire, the projected global losses from credit card fraud are expected to reach a staggering USD 43 billion by 2026. Additionally, with credit card fraud business costs extend beyond losses. According to LexisNexis every dollar lost to a fraudster cost north Americas financial institutions $4.41. This goes to show how detrimental credit card fraud can be for institutions as not only does it contribute to a direct loss for companies but also indirect losses as well.

Detecting fraud in real time is can be difficult because of the large volume of daily transactions and the fact fraudulent cases represent a very small percentage of the data. This extreme imbalance in data makes it difficult to accurately detect fraud.

Credit card fraud is a multi-billion-dollar problem that affects consumers, cuts company profits, and challenges consumer trust. This project aims to use machine learning to build a detection system that can identify suspicious transactions and highlight influential features during the process of detecting fraud.

**Key data findings**

After carrying out the EDA and modelling section, I gained an insight to things that may not have been as obvious if I hadn’t. For example, after calculating the fraud rate I was able to understand that from the data set that I downloaded from Kaggle, fraud makes up only 0.17% of all transactions showing how highly imbalanced the dataset is and how rarely credit card fraud occurs. This also stresses the importance of reducing false positives otherwise legitimate customers could be wrongly flagged.

From the feature importance section, we can see how V17, V12 and V14 had the greatest influence on the fraud detection process. Although these features are anonymized, these features are what companies must prioritise when detecting fraud. This will help not only speed up the process of detecting fraud and but also reduce false positives.

Transaction amounts range from 0.01 to over 25,000, with fraudulent transactions often clustered around smaller amounts in the dataset. From research, I understand this is a common tactic used by fraudsters to test stolen cards before making transactions.

The time feature did not show a direct relationship with fraud.

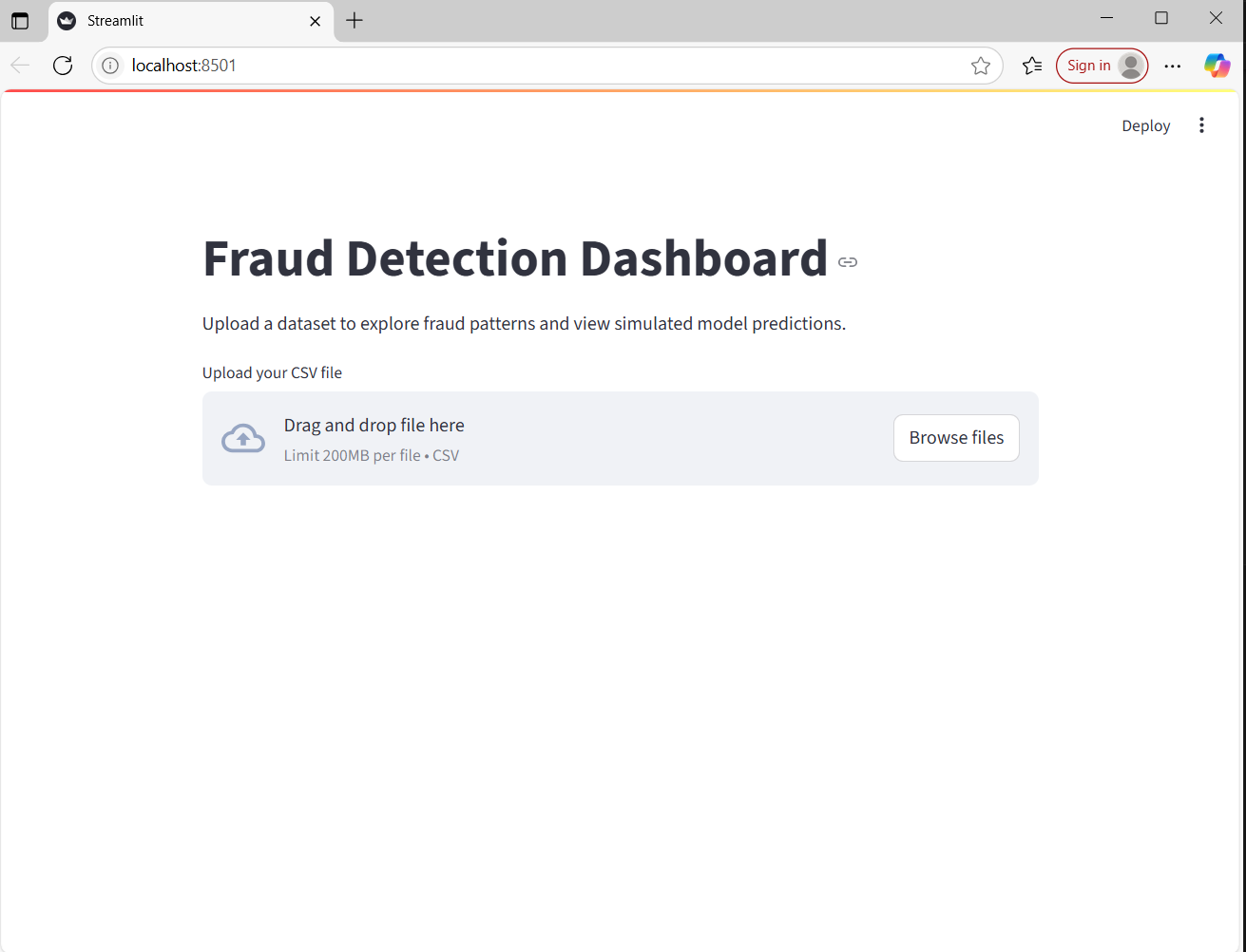
**Limitations**

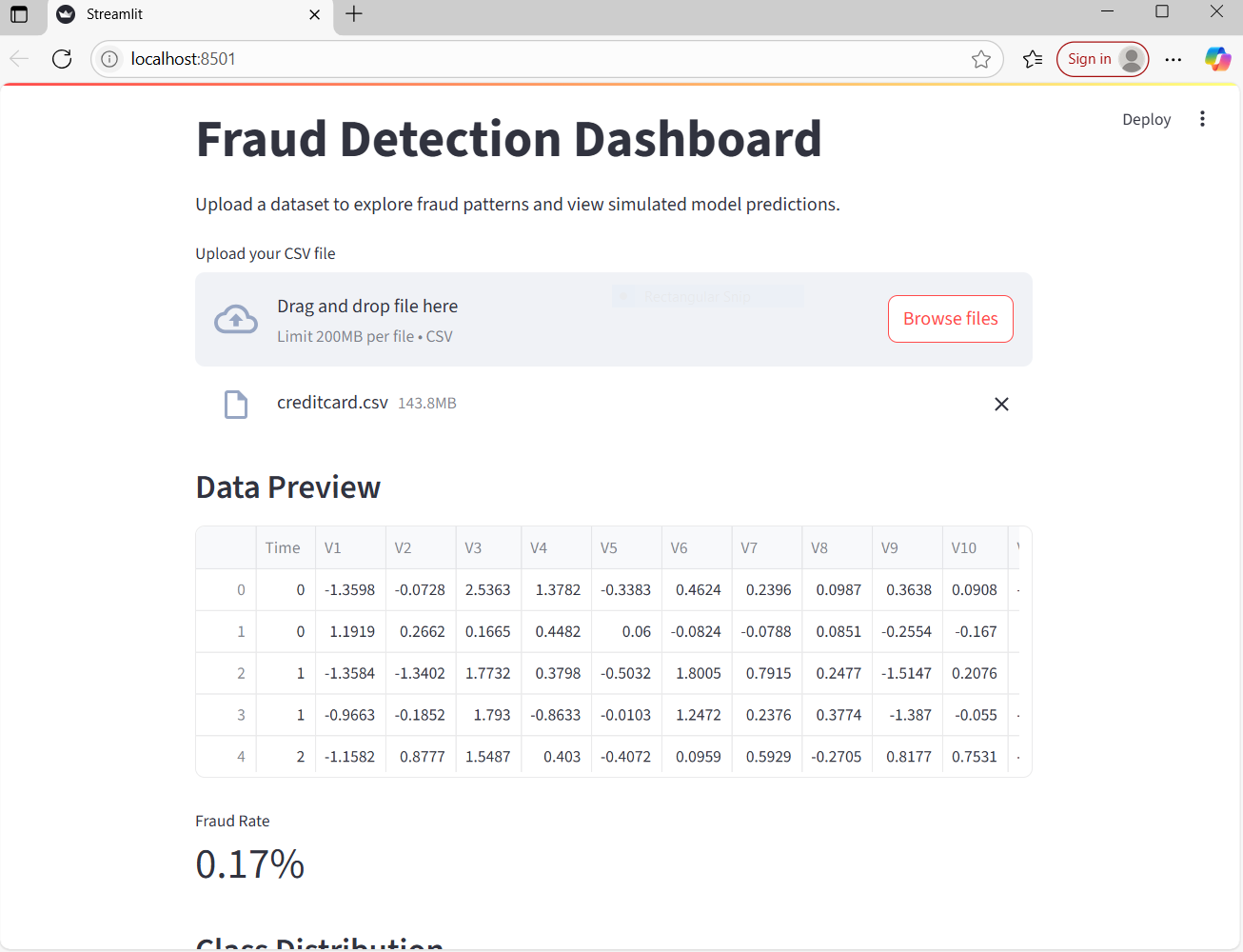
The dataset contains 0.17% fraudulent transactions. While this reflects real world conditions, it can make training a model difficult.

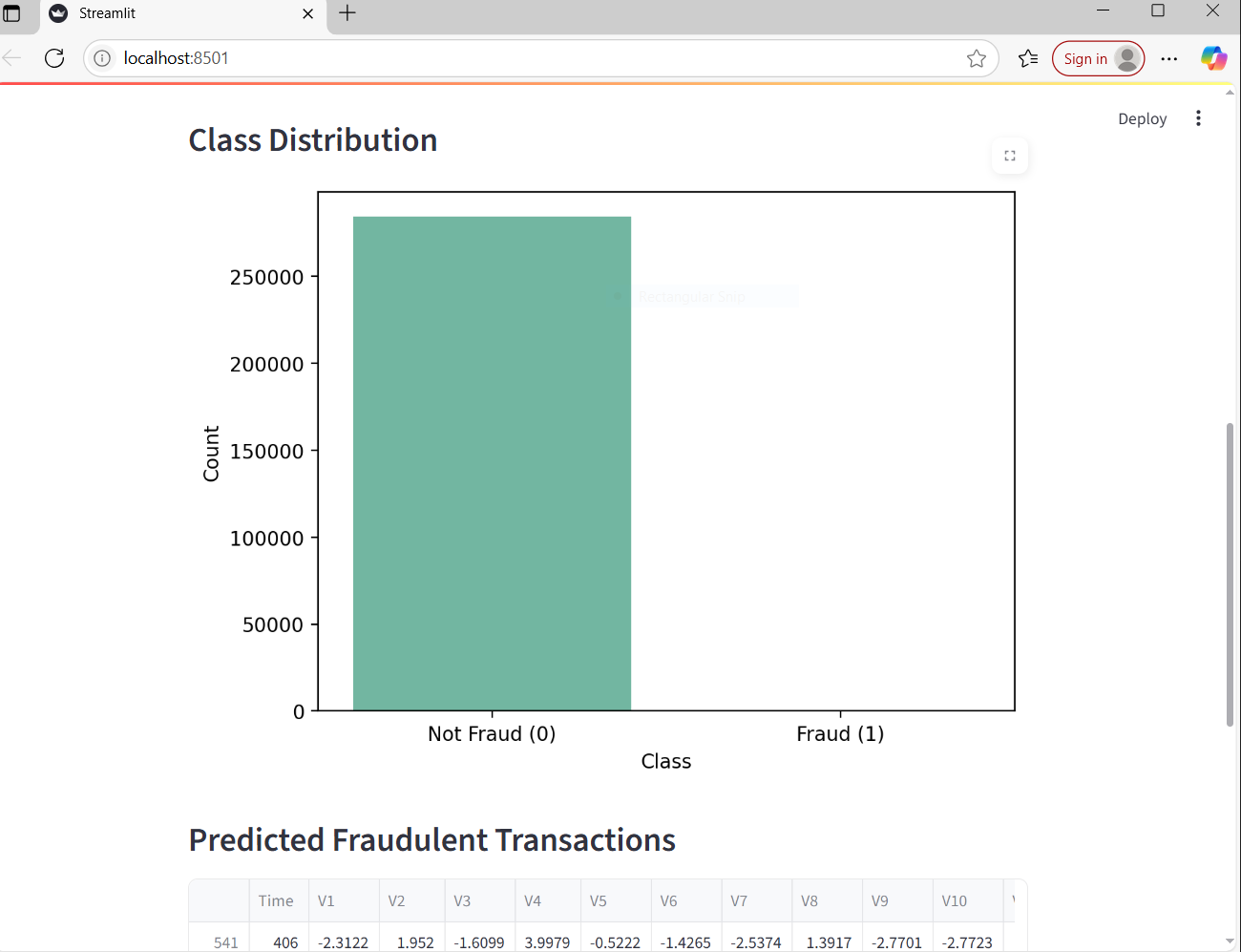
Anonymized features limit interpretability. While the model can detect patterns, it prevents a user from being able to explain features like V17 for example.

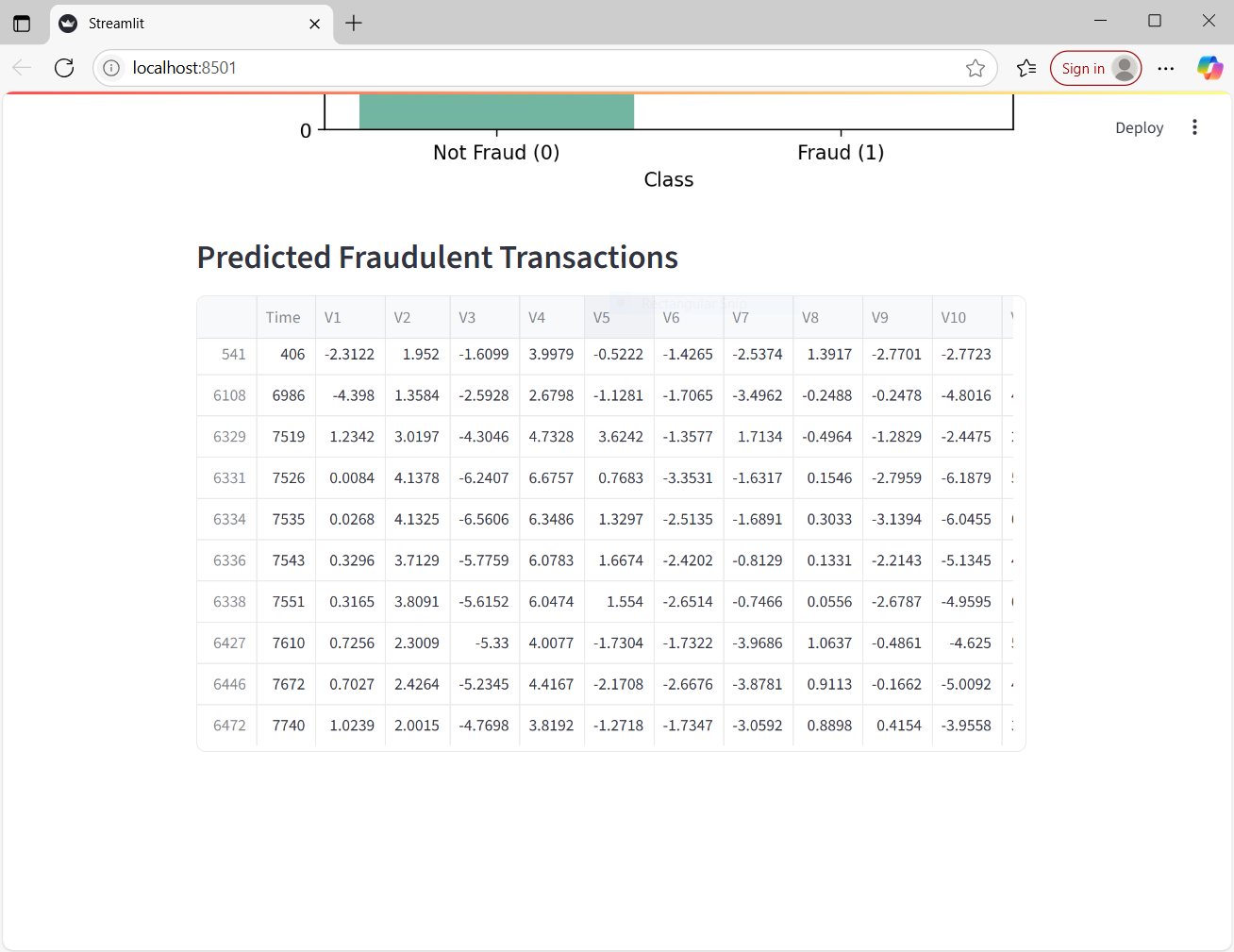
This model is trained on a static dataset and may not perform well over time without regular training.

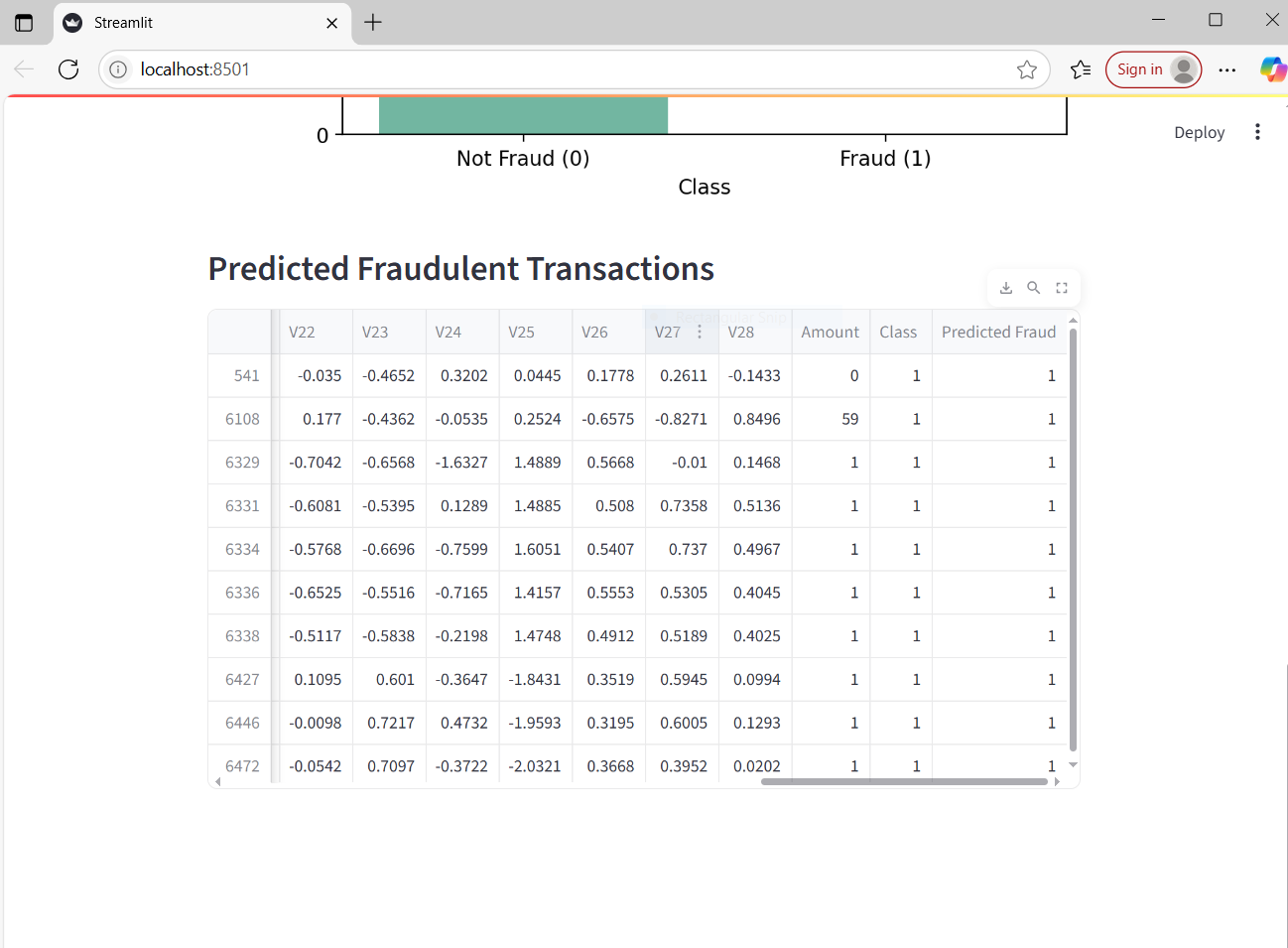
The dataset I used from Kaggle may represent a specific area and may not generalize global fraud patterns.

Images of Dashboard when run:









**Images from visualisation section**

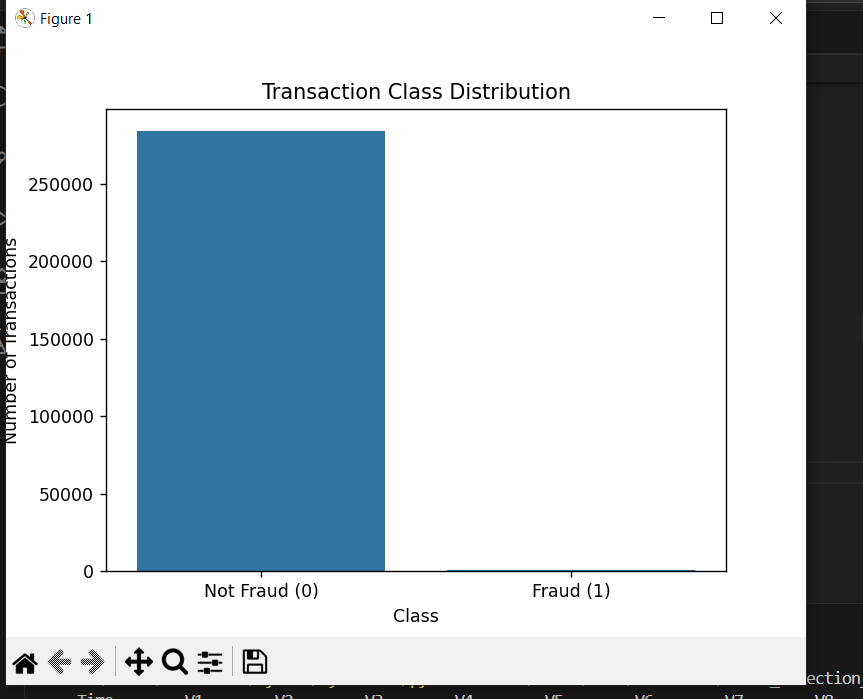
Image of bar chart:

Image of histogram:

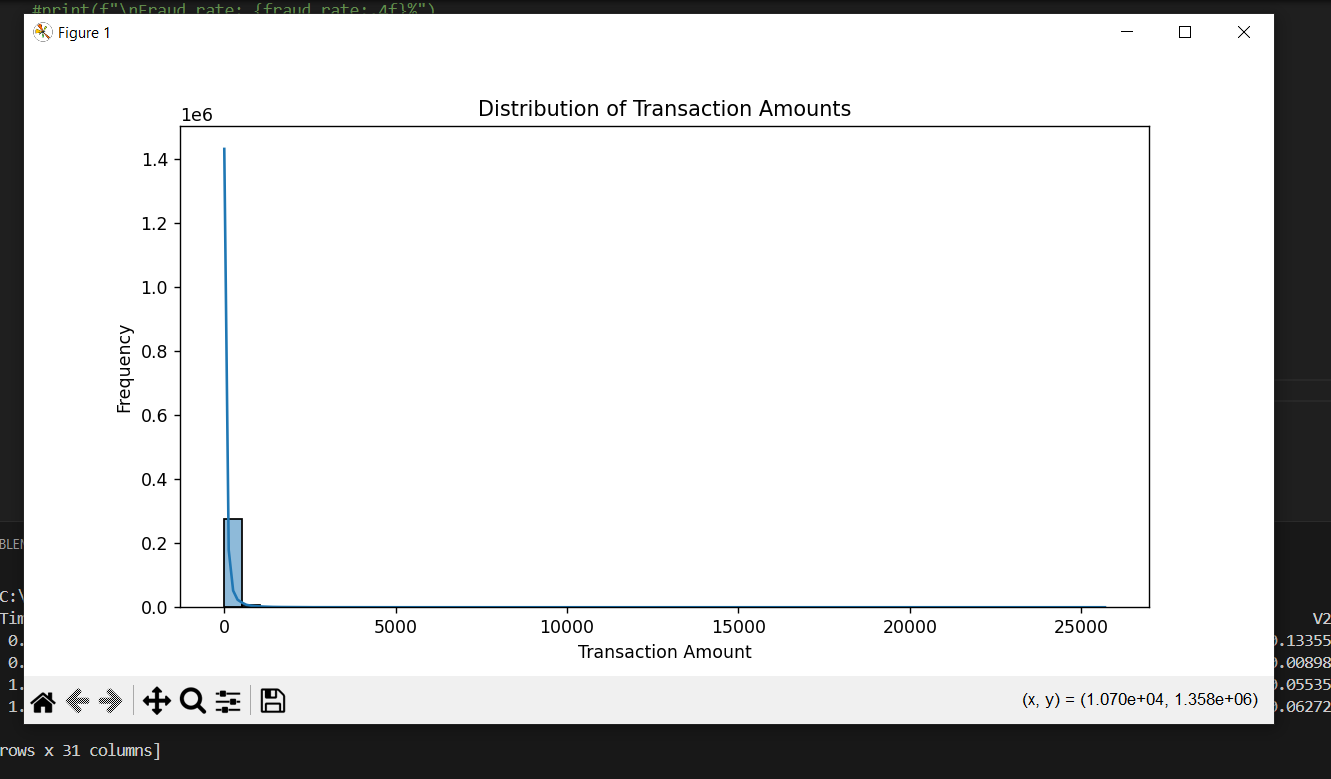
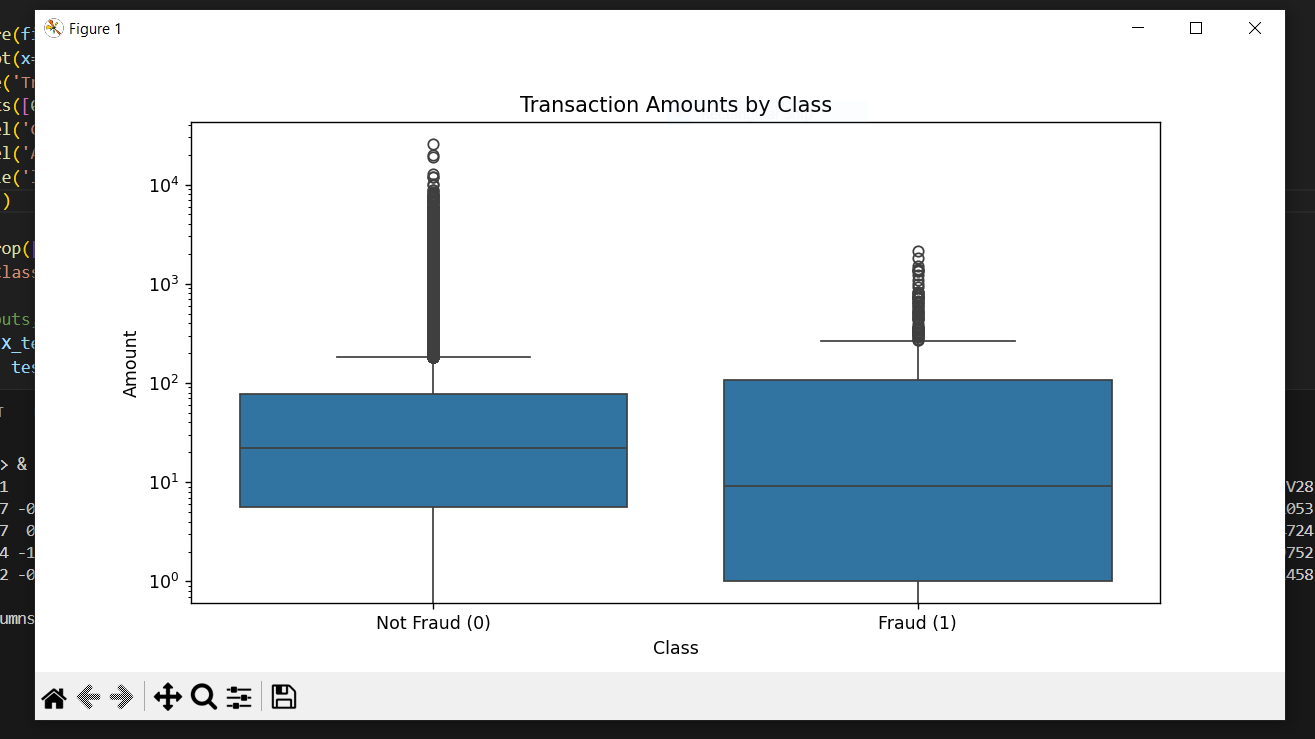


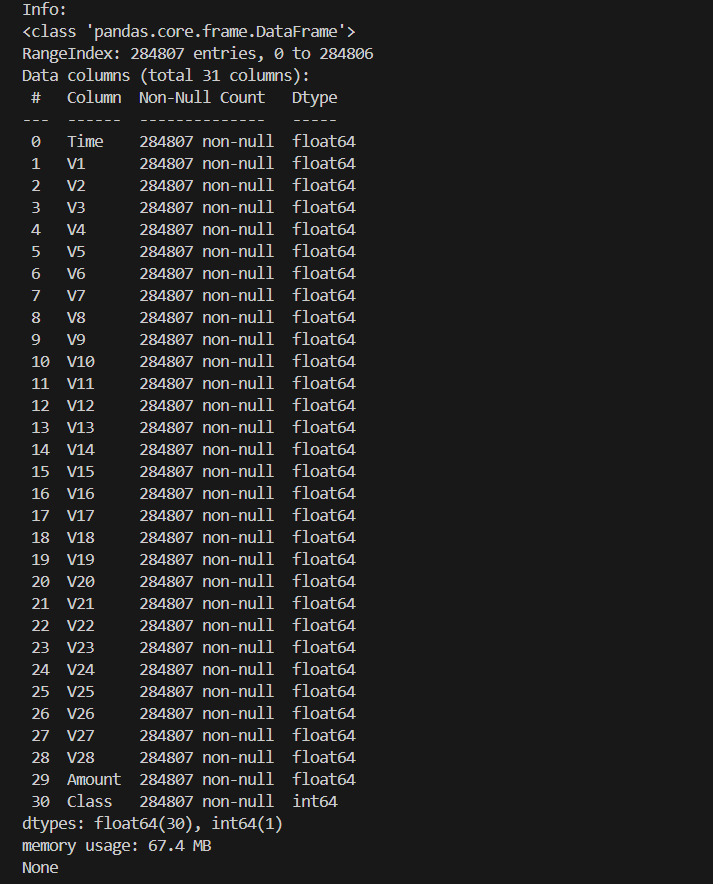
Image of box plot:



**Images from data analyses section**

print("\nInfo:")

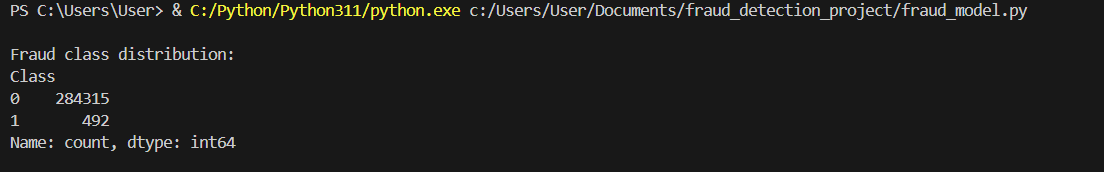
print(df.info())



#Check class distribution

print("\nFraud class distribution:")

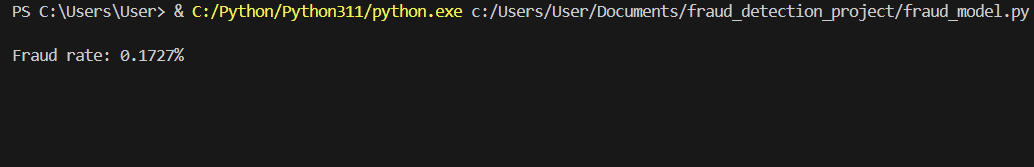
print(df['Class'].value\_counts())

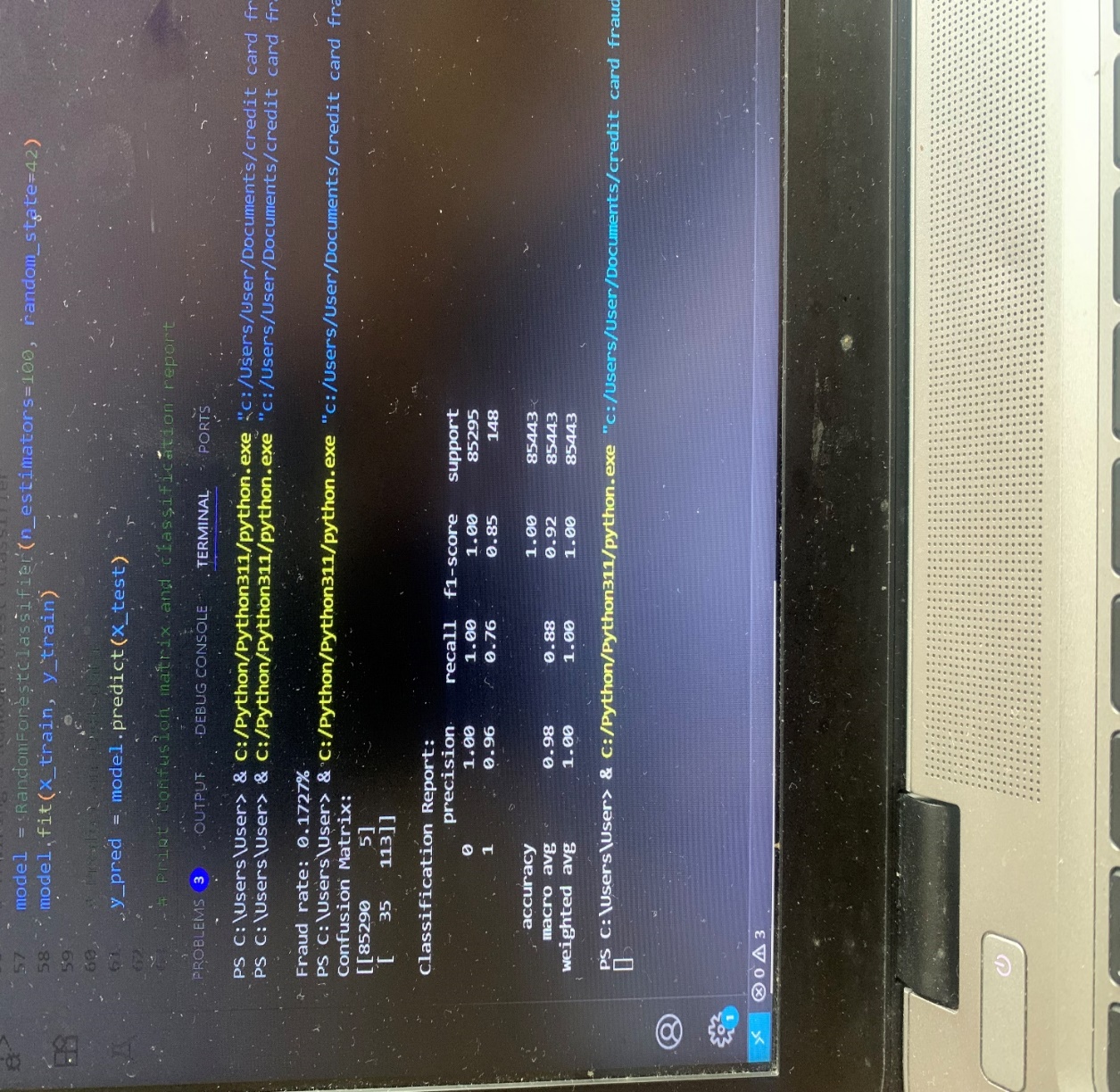


#% of fraud cases

fraud\_rate = df['Class'].value\_counts(normalize=True)[1] \* 100

print(f"\nFraud rate: {fraud\_rate:.4f}%")



Classification report and confusion matrix: