Challenge – FinCrime

Congratulations for making it to the data challenge! Its goal is to assess your critical thinking and give us a taste of your experience and technical skills. The problem and synthetic data we provide represent a simplified version of what we get to work on.

Time frame – 7 Days.

Please note that we do not expect the challenge to take more than 4–6 hours depending on how familiar you are with the tasks involved and different tools.

Deliverable

1. You should submit at least one Jupyter **notebook** – letting us see how and why you went about our challenge in a given way is integral to completing this task.
2. The code should be efficient, clean and sufficiently commented.

Your submission should be in a **.zip** file – do **not** host it on GitHub or other platforms. Group the contents into the following sub-directories, omitting the empty ones.

- notebooks – please also include **.html** versions of the notebooks
- data – only include new data files, if any, excluding the files we provide
- artifacts
- code
- misc

Name the submission <your_first_name>_<your_last_name>_ht.zip.
For example, aubrey_graham_ht.zip. Please do **not** protect it with a password.

Before proceeding to the task, go through **README.md** to learn about the data.

Good luck!

ANALYSIS, MODELLING & GOING INTO PRODUCTION

Your goal is to design an algorithm to identify fraudsters and take appropriate action. In this case, a fraudster is someone who signs up and uses Revolut to steal other people's money from the outside into an account with us via top-ups. You are provided with the first 2 weeks of users' **transactions** and info on **users** that is available from the start, at sign-up. Here, to streamline some of the decision-making, we make a few assumptions.

1. Each transaction of a fraudster is a means to commit and disguise crime.
2. We treat the task as a supervised binary classification problem, where we classify users based on their transactions. **fraudsters.csv** gives users of class positive.
3. We make a prediction at **each transaction** of a user, rather than only once at the end of their transaction history.

a) **EDA and Feature Engineering**

Explore the data to get an understanding of the problem and how to proceed in the later steps. Consider looking at user country, transaction created_date, amount_gbp, type and state. Come up with features that could help you tell fraudsters from other users, as raw data may be of little use to your model.

b) **Modelling and Validation**

Make a model to identify fraudsters. We ask you to start with a logistic regression. Assess its raw quality using your chosen metrics.

We suggest you focus on fundamentals and robust modelling – split strategy, avoiding data leakage, examining your metrics – rather than getting the best performance, yet expect your model to make some sense of the data. We leave trying extra steps or other algorithms, if any, to you.

c) **Raw Predictions to Actions**

Building on your model's predictions, decide in which cases a user is to be locked.

LOCK_USER – Current transaction is stopped and user's account is locked. To be unlocked and make transactions, the user is to contact an agent. If they do not, the agent will still investigate the case some time later.

You may use any trade-off between checking more people and catching more fraud you find reasonable having considered that we both deal with people's money and have limited human resources. Evaluate the performance of your proposed strategy. Here it is how you do the evaluation that we do care about.

Given the assumptions in the task overview, we will check a given user only once, the first time they are locked, so the performance should be at **user-level** even though raw model's predictions are per transaction. Show what drives your choice of specific thresholds/cut-off points **graphically** with a plot.

d) **Putting It into Prod**

Make **Patrol** class implementing your proposed locking logic. To keep it simple, we suggest it omits for now some of the pieces we would need in it in prod.

- Get the model and other objects needed, such as features at the moment of each transaction, in advance and simply load them on instantiation.
- **.check_transaction()** method is to take a transaction ID as a string, compute the prediction and return an action ("PASS" if none), here regardless of if the previous transactions have already resulted in a lock.
- In this part we expect production-grade code, exception handling, etc.