# Creating working environment to run the code

- You can start by creating a folder named mansa
  > mkdir mansa
- I have been working with Python version 3.8.10

```
PS C:\Users\          \Documents\Privé\Mansa\Assignement_submission\mansa> python --version
Python 3.8.10
```

- Create a virtual environment in the created folder

```
> python -m venv env
```

- Activate the venv

```
> . .\env\Scripts\activate
```

- Next, install the project requirements from file requirements.txt

```
> pip install -r requirements.txt
```

  At the end, you should see the required packages installed as follow

```
Installing collected packages: sniffio, numpy, idna, typing-extensions, threadpoolctl, six, scipy, joblib, colorama, anyio, wheel, ur
llib3, tomli, starlette, scikit-learn, regex, pytz, python-dateutil, pydantic, platformdirs, pathspec, mypy-extensions, h11, click, c
harset-normalizer, certifi, asgiref, xgboost, uvicorn, requests, pandas, lightgbm, fastapi, black
Successfully installed anyio-3.3.4 asgiref-3.4.1 black-21.11b1 certifi-2021.10.8 charset-normalizer-2.0.7 click-8.0.3 colorama-0.4.4
fastapi-0.70.0 h11-0.12.0 idna-3.3 joblib-1.1.0 lightgbm-3.3.1 mypy-extensions-0.4.3 numpy-1.21.4 pandas-1.3.4 pathspec-0.9.0 platfor
mdirs-2.4.0 pydantic-1.8.2 python-dateutil-2.8.2 pytz-2021.3 regex-2021.11.10 requests-2.26.0 scikit-learn-1.0.1 scipy-1.7.2 six-1.16
.0 sniffio-1.2.0 starlette-0.16.0 threadpoolctl-3.0.0 tomli-1.2.2 typing-extensions-4.0.0 urllib3-1.26.7 uvicorn-0.15.0 wheel-0.37.0
xgboost-1.5.0
```

- In order to run notebooks within the environment, install the packages from requirements-eda.txt

```
> pip install -r requirements-eda.txt
```

  You should have something similar to this

```
Installing collected packages: wcwidth, traitlets, pywin32, parso, tornado, pyzmq, pygments, prompt-toolkit, pickleshare, nest-asynci
o, matplotlib-inline, jupyter-core, jedi, entrypoints, decorator, backcall, jupyter-client, ipython, debugpy, ipykernel
Successfully installed backcall-0.2.0 debugpy-1.5.1 decorator-5.1.0 entrypoints-0.3 ipykernel-6.5.1 ipython-7.29.0 jedi-0.18.1 jupyte
r-client-7.1.0 jupyter-core-4.9.1 matplotlib-inline-0.1.3 nest-asyncio-1.5.1 parso-0.8.2 pickleshare-0.7.5 prompt-toolkit-3.0.22 pygm
ents-2.10.0 pywin32-302 pyzmq-22.3.0 tornado-6.1 traitlets-5.1.1 wcwidth-0.2.5
```

- To open the project notebook, type:

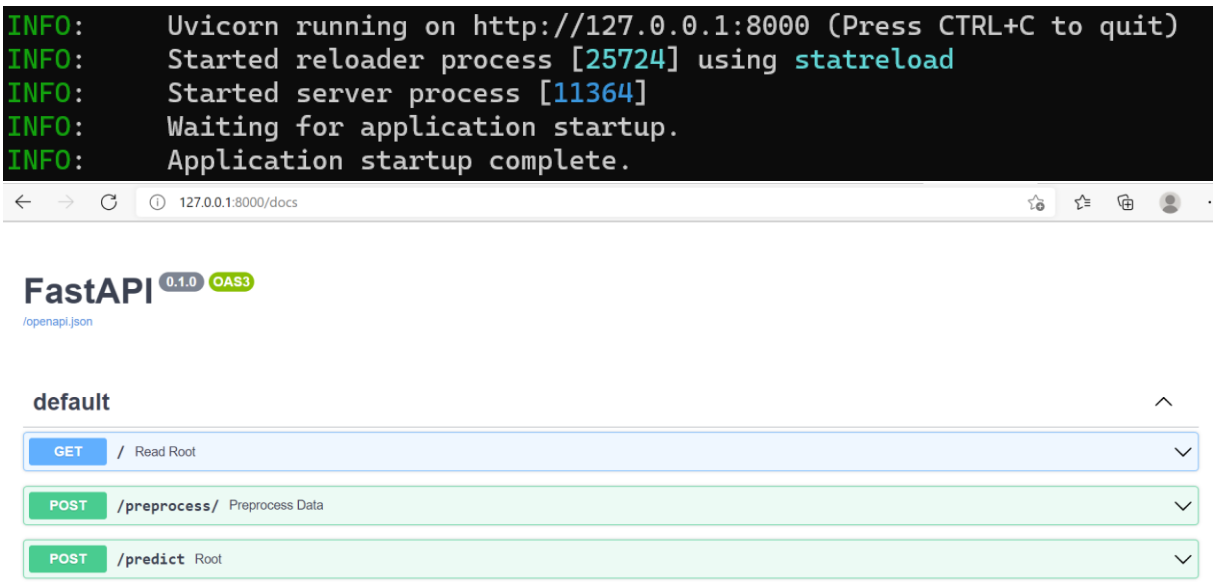
```
> jupyter notebook
```

  A new window window of the local project will be opened to you and you can open the jupyter notebook named model_training.ipynb where the prediction model was created.

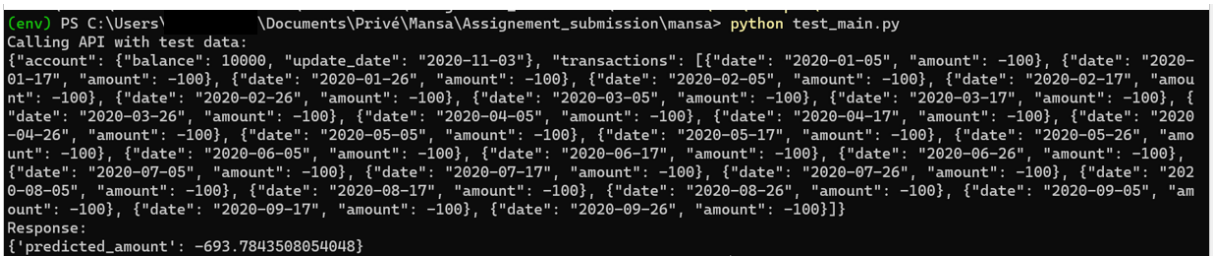

- Open a new cmd window to run the FastAPI, to load the API type the following:

```
> uvicorn main:app --reload
```

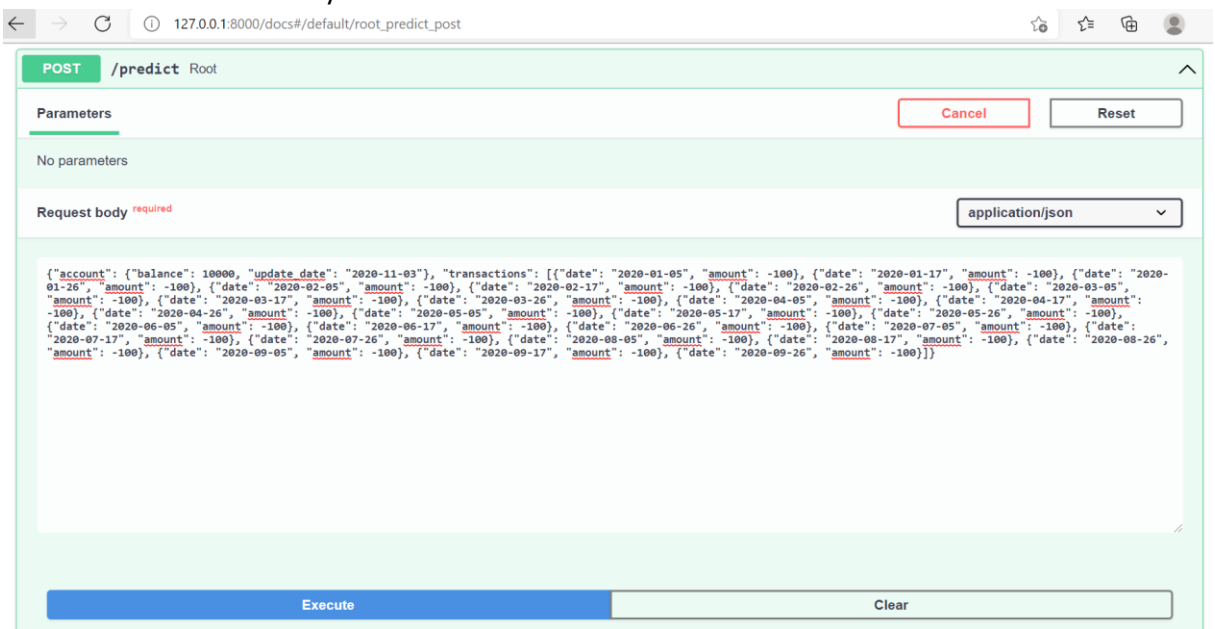
The server is now running and we can access to our APIs at the address http:127.0.0.1:8000

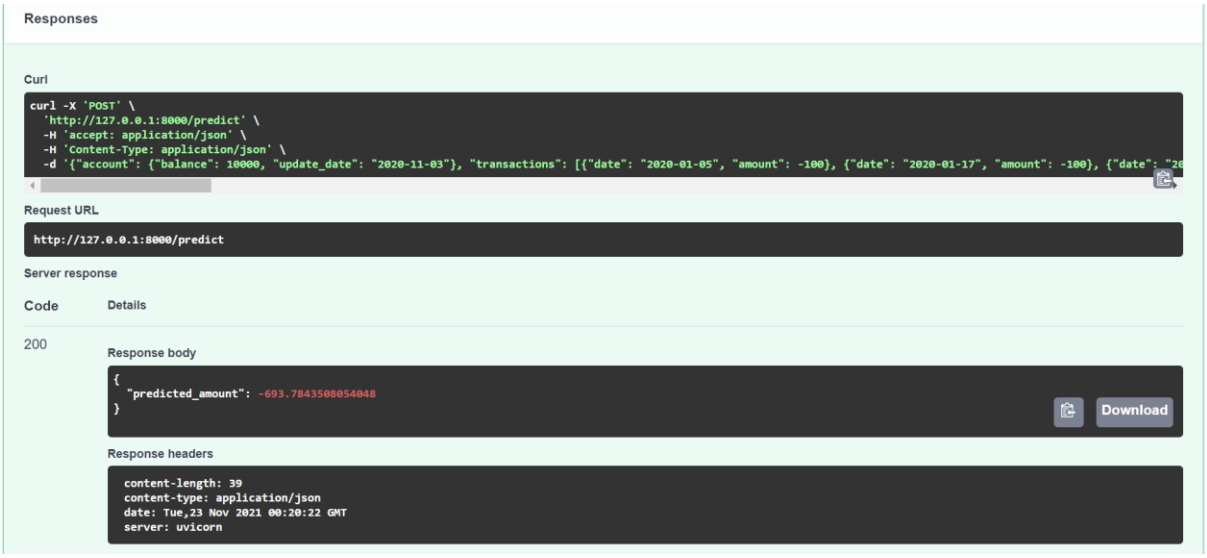



- You can test the API, by opening a third window, activating the virtual environment and typing python test_main.py




- You can test the API directly from the interface as well:






The response code is 200 and the predicted amount for the next month of the given account is displayed.

# Forecast Model

1. Dataset

In order to create an accurate model based on the available data, we had to create our own datasets that we perceive represent the problem in a better way in order to solve it.

In the next section, we will provide insights on how we filtered the provided data, the new features that we had created, and all the transformations made on this data.

2. Data exploration & Preprocessing

Before building any model, we assume that accounts with less than 180 days of history are not relevant, hence we discarded them by computing the number of days between the update date of the account and the oldest transaction date. We also discarded duplicate data and constructed the following features for the forecast model:

| Feature | Description |
|---|---|
| month | The current month of the obeservation |
| average_transactions_per_month | The number of transaction per month for the account of the observation |
| age | The age of the account up to the update date |
| initial_balance | The initial balance of the account |
| outgoing_1 | The total outgoing of the previous month |
| ingoing_1 | The total ingoing of the previous month |
| outgoing_2 | The total outgoing of the 2$^{nd}$ previous month |
| ingoing_2 | The total ingoing of the 2$^{nd}$ previous month |
| outgoing_3 | … |
| ingoing_3 | … |
| outgoing_4 | … |

| | |
|---|---|
| ingoing_4 | … |
| outgoing_5 | … |
| ingoing_5 | … |
| outgoing_6 | … |
| ingoing_6 | … |

Further details can be found in the notebook of the project (Plots, statistics, VIF test...)

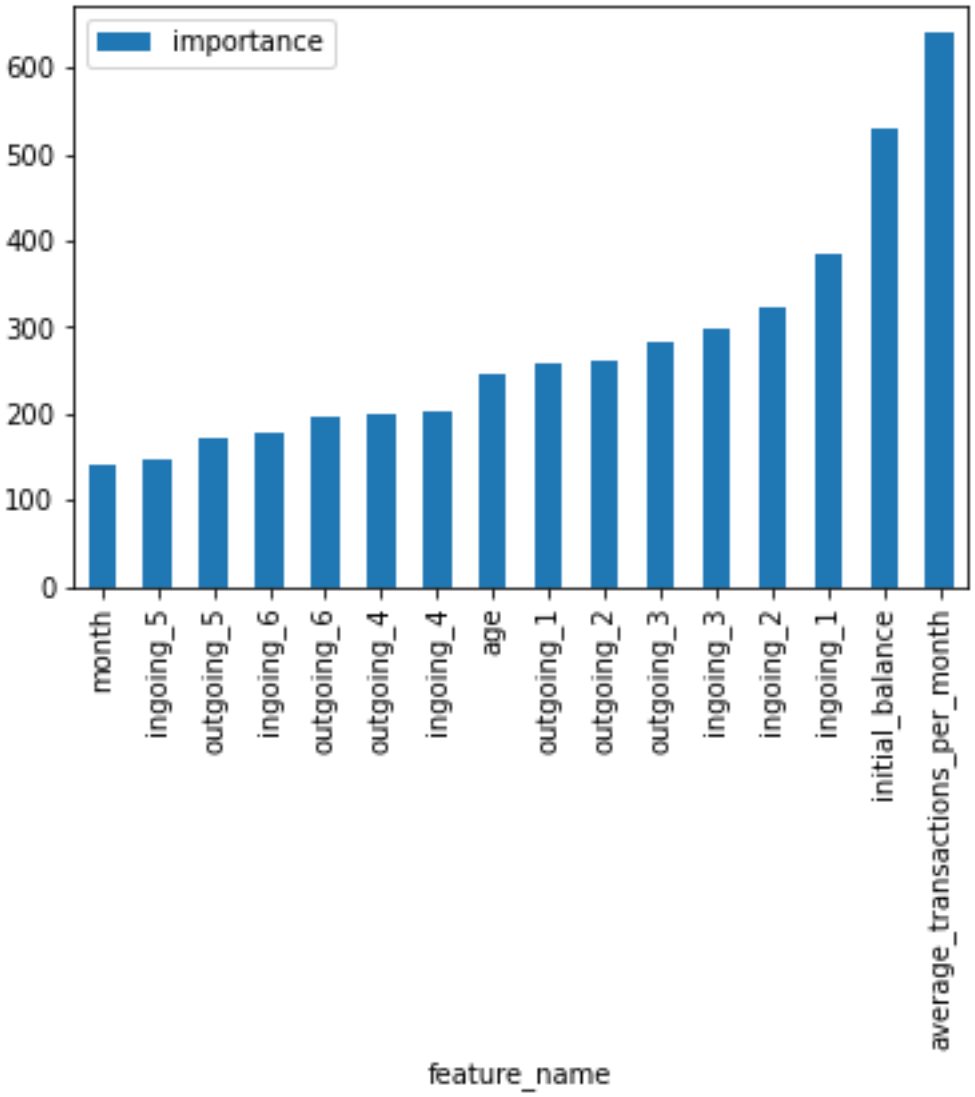
3. Machine Learning Model

This problem can be seen as a regression problem. Given the data of an account in the previous months, we would like to compute an estimate of the account outgoing for the next month.

This model is known to be a 1-step forecast. The train/test split is explained at the level of the notebook as well.

I tried these two Machine Learning algorithms:

- Random Forest : based on multiple decision trees.
- Lightgbm : gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages: ( Faster training speed and higher efficiency, has better accuracy, can handle non-assigned data, doesn't require scaling data)
  The plot of the features importance of this algorithm is the following



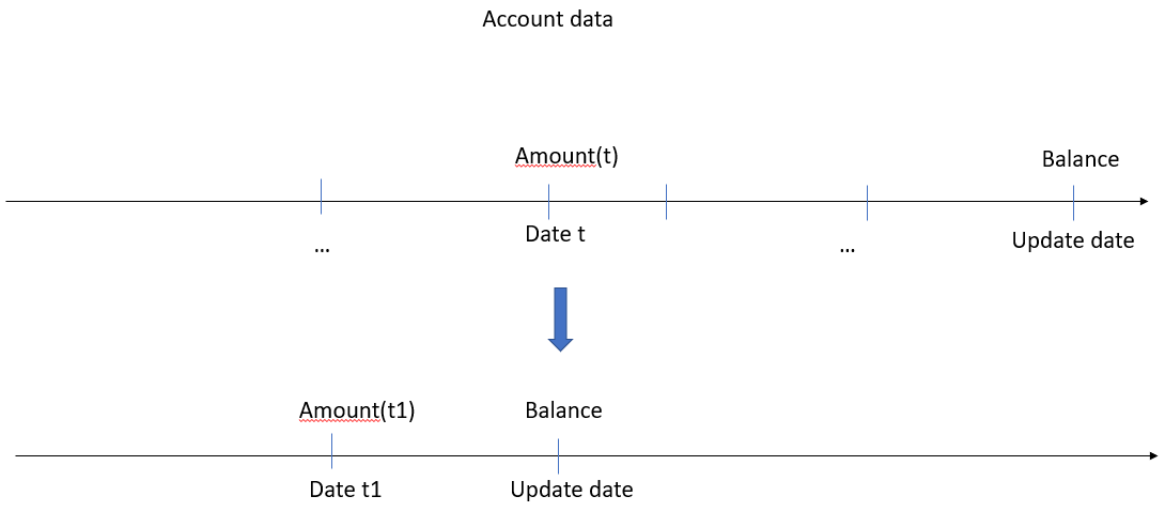

4. Error metrics
   The error metrics used in our case are:
   - L1
   - L2
   - Mean absolute percentage error
   - Root Mean Square Error

The lightgbm seems to be promising. This metric can be improved by fine tuning the model.

5. Strengths and weaknesses

This model is straightforward and quite intuitive. It is easy to develop and its computational time is good. However by aggregating data on monthly level, I ended with very few data after discarding irrelevant accounts and outliers ( ~ 8000 rows for the training which is very low).

To overcome this, I thought about generating new data from the available ones: in fact knowing the balance at the update date and all the past transactions, I will be able to consider every snapshot date as un update date for this account and I would be able to compute the balance at this snapshot date. The following figure summarizes this idea:




In my case, I would also enhance the train test split since we are dealing with time series data, it is better to build this fold validation while respecting the chronological coherence.