**PROJET OPA – DE NOV 2022 DATASCIENTEST**

Equipe projet : Benjamin AYELA / Bertrand WULFF / Loïc CHEVALIER

Mentor Datascientest : Fall Lewis YOMBA

Chef de cohorte Datascientest : Dimitri CONDORIS

SUJET :

**CryptoBot avec Binance**

**Cursus concerné :** Data Engineer

**Difficulté :** 8/10

**Description détaillée :**

*De nos jours, le monde des crypto commence à prendre une place importante et grossi. Il s’agit tout simplement de marchés financiers assez volatiles et instables se basant sur la technologie de la Blockchain.  
Le but principal de ce projet est de créer un bot de trading basé sur un modèle de Machine Learning et qui investira sur des marchés crypto.*

*Vous trouverez ci-dessous les étapes du projet tel qu’elles sont décrites dans le sujet du projet.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Etape** | **Description** | **Objectif** | **Modules / Masterclass / Templates** | **Conditions de validation du projet** |
| **1** | Récolte des données | Récolter deux types de données en passant par l’API Binance en passant par une architecture de streaming.   1. Grâce à cette API, on peut aller récupérer des informations sur les cours des différents marchés (BTC-USDT, BTC-ETH, …). Le but sera de créer une fonction de récupération de données générique afin de pouvoir avoir les données de n’importe quel marché.   Il faudra aussi créer un script de pré-processing pour réorganiser les données sortant du streaming afin qu’elles soient propres.   1. Récupérer les données historiques, pré-processé pour pouvoir entraîner notre futur modèle |  | Fichier explicatif du traitement (doc / pdf)  Un fichier json d’exemple de récupération. |
| **2** | Architecture de la donnée | Il s’agit de choisir la solution de stockage la plus adaptée.   * 2 tables SQL, une pour les données historiques et une autre pour les données streaming * Une DB Mongo comportant 2 collections: une pour les données stream et une autre pour les données | 142 - SQL  (Architecture des données)  Elasticsearch  MongoDB | Une base de données relationnelle  Un fichier de requête SQL pour montrer que c’est bien fonctionnel  Même rendu mais exemples de requêtes Elastic/Mongo |
| **3** | Consommation de la donnée | Utiliser un algo de Machine Learning appliqué à la finance qui permettra de retourner une décision d’achat ou non. (Quelle stratégie pour la vente ?) Aller plus loin: prédiction de gains  Si pas le temps de faire ce travail, possible de faire un dashboard de suivi de la crypto | DE120  Dash | Notebook de ML |
| **4** | Mise en production | Faire une API pour tester le modèle de ML et pourquoi pas requêter les données historique  Dockeriser tout le projet pour qu’il soit reproduisible sur n’importe quel machine | Docker  FastAPI | Fichier Yaml du docker-compose  API FastAPI |
| **5** | Automatisation des flux | **Étape Facultative**  Il faudra requêter l’API quotidiennement via Airflow. Nous ne pouvons pas utiliser Airflow pour les données en streaming, mais vous pouvez tester Nifi. | Airflow | Fichier python du DAG |
| **6** | Soutenance | Démonstration de leur appli et explication du raisonnement effectué lors de leur projet. | X | Soutenance Rapport |

**Etape 1 : Recolte des données**

Afin de recolter les données des marchés cryptomonaie, l’enoncé nous impose de passer par la plateforme d’échange BINANCE (<https://www.binance.com/fr>). Pour commencer, nous avons du créer un compte afin d’obtenir des clefs (une publique et une privée) pour utiliser l’API.

1. **Créez un compte sur la plateforme Binance pour obtenir une clé API.**

L’API est aujourd’hui paramètrée uniquement en lecture pour éviter de mauvaises manipulations. En effet, pour obtenir la clef, il fallait faire un premier dépôt en € ou acheter une crytommonaie. Nous avons acheté pour 15€ de BTC pour commencer.

Une image contenant texte

Description générée automatiquement

1. **Créer notre environnement de travail.**

Nous avons créer un Github pour pouvoir nous organiser dans notre travail en équipe.

Il est organisé de la façon suivante :

* Dossier ***Gestion de projet*** pour tout ce qui est relatif aux rendus du projet (dont ce document)
* Dossier ***playground*** pour tester du code en autonomie
* Dossier ***src*** (source) pour le code validé ensemble.

Une image contenant texte, moniteur, capture d’écran, intérieur

Description générée automatiquement

1. **Sécurisé les clefs API (car le Github est public)**

Les clefs fournis par Binance ne doivent pas être visibles sur Github car il est public. Pour cela, un fichier .gitignore permet d’ignorer le fichier config/config.ini dans lequel nous stockons la clef localement sur nos machines.

Une image contenant texte

Description générée automatiquement

1. **Librairies / packages**

Afin de simplifier notre organisation, nous avons créé un fichier ***requirements.txt*** dans lequel nous mettons les différents paquets Python dont l'installation est requise dans un environnement virtuel pour que l'application s'exécute correctement.

Une image contenant texte

Description générée automatiquement

1. **Utilisez une bibliothèque Python, telle que "python-binance" pour interagir avec l'API Binance et récupérer les données de marché (hist).**

Ci-dessous est décrit le code nous permettant d’extraire les données historique de 10 ‘ticker’ (pour commencer). Fichier : ***data\_api\_hist.py***

Le code permet de se connecter à l’API avec les clefs, de récupérer tout les ‘tickers’, puis pour les 10 premiers de récupérer les données, de les convertir en Dataframe, puis de le stocker dans un dictionnaire et enfin d’exporter en format JSON.

from binance.client import Client

import json

from tqdm import tqdm

import pandas as pd

import config

# Initialise le client Binance

client = Client(config.BINANCE\_API\_KEY, config.BINANCE\_API\_SECRET)

client.ping()

# Récupère tout les 'tickers' de l'API

tickers = client.get\_all\_tickers()

# Initialise un dictionnaire pour stocker les données de chaque ticker

ticker\_data = {}

loading\_tickers = tqdm(tickers[0:10])

# Récupère les données de l'API sur chacun des tickers

for ticker in loading\_tickers:

    loading\_tickers.set\_description(f"loading symbol {ticker['symbol']}")

    klines = client.get\_historical\_klines(ticker['symbol'], Client.KLINE\_INTERVAL\_8HOUR, "1000 days ago")

    # Transforme les données en DataFrame

    data = pd.DataFrame(klines, columns=["timestamp", "open", "high", "low", "close", "volume", "close\_time", "quote\_asset\_volume", "number\_of\_trades", "taker\_buy\_base\_asset\_volume", "taker\_buy\_quote\_asset\_volume", "ignore"])

    data.drop(columns="ignore", inplace=True)

    # Change le type de la colonne timestamp en datetime

    data["timestamp"] = pd.to\_datetime(data["timestamp"], unit="ms").dt.strftime('%Y-%m-%d %H:%M:%S')

    # Stocke la DataFrame correspondante dans le dictionnaire

    ticker\_data[ticker['symbol']] = data.to\_dict(orient='records')

# Exporte les données au format JSON

with open('ticker\_data\_hist.json', 'w') as f:

    json.dump(ticker\_data, f)

En faisant ça, nous recoltons toutes les valeurs des 10 ‘tickers’ depuis 1000 jours (donc + de 2,5 ans) avec un intervalle de 8 heures.

1. **Établissez une architecture de streaming pour recevoir les données de manière continue.**

Comme demandé dans l’énoncé, nous avons chercher à collecter les données du marché de manière continue en utilisant une architecture de streaming.

import asyncio

import websockets

import json

import time

async def subscribe(websocket):

    # Abonnement au ticker de BTCUSDT

    await websocket.send('{"method": "SUBSCRIBE", "params": ["btcusdt@ticker"], "id": 1}')

async def receive(websocket, data\_list):

    async for message in websocket:

        data = json.loads(message)

        print(data)

        data\_list.append(data)

        break

async def main():

    async with websockets.connect("wss://stream.binance.com:9443/ws") as websocket:

        await subscribe(websocket)

        # Mettre en place une boucle d'attente de 10 secondes

        # et d'arrêt après 1 minute (6 boucles)

        data\_list = []

        for i in range(6):

            await receive(websocket, data\_list)

            time.sleep(10)

        # Écrire les données dans un fichier JSON

        with open("BTCUSDT\_data\_stream.json", "w") as json\_file:

            json.dump(data\_list, json\_file)

if \_\_name\_\_ == "\_\_main\_\_":

    asyncio.run(main())

1. **Enregistrez les données dans un format approprié, tel que JSON.**

Vu dans les deux fichiers de récupération de données précédents, les données sont exportés

A rajouter :

* fichier pour afficher sous forme graphique les données.
* Average price into CSV
* Lastest price into CSV

*Questions :*

*1/ l'intérêt d'avoir du stream car on peut demander c'est chandelles toutes les 1min et remonter des années en arrière.*

*2/ avoir une petite aide sur le fonctionnement du marché ie comment acheter et vendre. Faut-il placer un ordre d'achat avec un prix ? Pour achter faut-il regarder les différentes offres avec un prix associé ? ..*

*A faire :*

*Rajouter Fall dans le Github*

*Archiver les livrables et les mettre sur le canal Slack*

*Fall lewis fait un retour sur les livrables*

*Planifier les points toutes les 3 semaines, les mercredi 10h.*

*Suite : stockage des données, organiser les données (05/06). Base de données relationnelle + NOSQL + requetes des 2 bases fonctionnels.*

*Ranger le github en fonction des étapes projets.*

*Possible de faire un call sur un fichier concert de données pour aider à faire le choix SQL /NOSQL.*

*Faire sur une monnaie est suffisant pour valider le principe.*

*Prochains points :*

* *29/03/23 à 10h*
* *19/04/23 à 10h*
* *10/05/23 à 10h*