**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**

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

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.

from binance.client import Client

import json

import websockets

import time

# Fonction pour traiter les données reçues par WebSocket

def on\_message(ws, message):

    data = json.loads(message)

    # Vérifier si c'est un message de prix ticker

    if 'e' in data and data['e'] == 'ticker':

        symbol = data['s']  # Récupérer le symbole

        price = float(data['c'])  # Récupérer le prix en float

        # Ajouter les données à un dictionnaire

        ticker\_data[symbol] = price

        # Enregistrer les données dans un fichier JSON toutes les 5 secondes

        if time.time() - last\_save\_time > 5:

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

                json.dump(ticker\_data, f)

            last\_save\_time = time.time()

# Fonction pour gérer les erreurs WebSocket

def on\_error(ws, error):

    print(error)

# Fonction pour gérer la fermeture de la connexion WebSocket

def on\_close(ws):

    print("WebSocket closed")

# Fonction pour ouvrir la connexion WebSocket

def on\_open(ws):

    # Envoyer une demande d'abonnement pour les prix de tous les tickers

    ws.send(json.dumps({

        "method": "SUBSCRIBE",

        "params": ["!ticker@arr"],

        "id": 1

    }))

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

    # Initialiser le dictionnaire pour stocker les données ticker

    ticker\_data = {}

    # Initialiser le temps de dernière sauvegarde

    last\_save\_time = time.time()

    # Ouvrir une connexion WebSocket

    ws = websockets.WebSocketApp("wss://stream.binance.com:9443/ws", on\_message=on\_message, on\_error=on\_error, on\_close=on\_close)

    ws.on\_open = on\_open

    ws.run\_forever()  # démarrer la boucle infinie pour recevoir des données en continu

En faisant ça, nous recoltons les 10 dernières valeurs des ‘tickers’ « BTCUSDT » avec un intervalle d’un jour.

1. Établissez une architecture de streaming pour recevoir les données de manière continue.
2. Créez une fonction générique de récupération de données pour pouvoir obtenir les données de n'importe quel marché.
3. Enregistrez les données dans un format approprié, tel que JSON.

*Vrac :*

*Faire un fichier pour tester la stratégie sur le passé et comparer avec d’autre*