

IA TRADE BOT

leodeclercq/CryptoTradingBot: Crypto Trading Bot

investissement (1% de bénéfice/jour = 0,01) :

$$C = C(0) * (1+0,01)^{\text{jours}}$$

Si 100€ = C(0) → C = 3 778,34€ (1 an = 365 jours)

→ C = 136,14€ (1 mois = 31 jours)

Si 1000€ = C(0) → C = 37 783,43€ (1 an = 365 jours)

→ C = 1 361,33€ (1 mois = 31 jours)

Close : dernier prix d'un actif à la fin d'une bougie (1 /s, 1/m).

TEMA : moyenne mobile exponentielle triple sur périodes, utilisée pour lisser les prix et détecter les tendances plus rapidement

1. Architecture du bot de trading #IA

❖ Données en entrée (1/s) :

- Prix '**Close**' en temps réel (Binance Websocket via API).
- Indicateurs techniques : **TEMA20, TEMA50**.

❖ Traitement des données :

- Calcul des indicateurs (**TEMA20, TEMA50**)).
- Normalisation des données (**Close, TEMA20, TEMA50**).
- #Création d'une structure de données utilisable par le modèle IA

❖ Prise de décision IA :

- #Un modèle de **régression logistique** ou un réseau de neurones (**LSTM**) donne une probabilité que le prix monte ou baisse.
- Si **Minima** (11 Derniers points), **Pente** > 0.06 et **TEMA20 Convexe** par rapport à **TEMA50** (=croisement) avec un décalage de -0.8 sur **TEMA50 , ACHAT**.
- Si **Maxima**(11 Derniers points), **Pente** < -0.084 et **TEMA20 Concave** par rapport à **TEMA50** (=croisement) avec un décalage de +0.45 sur **TEMA50 , VENTE**.
- #Si **P(hausse)** > 60%, **ACHAT.(Validation signal)**
- #Si **P(baisse)** > 60%, ouvrir un **VENTE.(Validation signal)**

📌 Exécution des ordres :

- Utilisation d'**ordres du marché**.
- **Backtesting** sur des données historiques pour optimiser la stratégie.

2. Code #avec Intelligence Artificielle 🚀

📌 script optimisé :

- Le script implémente les stratégies définies ci-dessus.

```

• # récupérer temps de notre machine
• import time
• # calcul scientifique, tableau, db etc
• import numpy as np
• # garde en mémoire x éléments de liste
• from collections import deque
• # instance de La bibliothèque binance qui permet d'interagir avec l'API de Binance (échange de cryptomonnaies).
• from binance.client import Client
• # La classe ThreadedWebsocketManager est spécifiquement conçue pour gérer les connexions WebSocket à Binance.
• # Un WebSocket est une technologie qui permet d'établir une connexion permanente entre le client et le serveur
• # pour recevoir des mises à jour en temps réel, sans avoir à faire des appels API répétitifs.
• # Cette classe gère plusieurs WebSockets de manière efficace en utilisant des threads séparés,
• # ce qui permet de recevoir des données en temps réel (comme les prix ou les ordres) de manière fluide
• # tout en gardant l'application réactive.
• from binance import ThreadedWebsocketManager
• # Importation de toutes les énumérations disponibles dans le module `binance.enums` de la bibliothèque Binance.
• # Le module `binance.enums` contient des constantes sous forme d'énumérations (enums) qui sont utilisées
• # pour spécifier des valeurs fixes et prédefinies pour différents paramètres lors des appels API.
• # Ces énumérations sont utilisées dans des méthodes comme `create_order()`, `get_order()`, etc.,
• # pour rendre le code plus lisible et éviter d'utiliser des valeurs littérales comme des chaînes de caractères ou des entiers.
• # Par exemple, les énumérations peuvent inclure :

```

- # - `Side.BUY` ou `Side.SELL` pour indiquer si l'ordre est un achat ou une vente.
- # - `OrderType.MARKET` pour spécifier le type d'ordre.
- # - `TimeInForce.GTC` (Good Till Canceled), `TimeInForce.IOC` (Immediate or Cancel) pour spécifier la durée de validité d'un ordre.
- # L'importation avec * permet d'importer toutes les énumérations à la fois, ce qui simplifie l'utilisation du code
- # sans avoir à les importer spécifiquement une par une. Cela permet d'écrire un code plus propre et plus lisible.
- # Cependant, l'importation avec `*` peut entraîner des conflits de noms si plusieurs modules ou classes
- # contiennent des membres portant le même nom. Dans de tels cas, il est conseillé d'importer uniquement
- # les énumérations nécessaires comme suit :
- # `from binance.enums import OrderType, Side, TimeInForce`
- from binance.enums import *
- # (Technical Analysis Library)
- # Cette bibliothèque permet de calculer des indicateurs techniques comme TEMA20 et TEMA50 (RSI, MACD, etc.)
- # Elle est utilisée pour l'analyse technique des données financières, notamment dans les stratégies de trading automatisées.
- import talib
- # fonction colored, change la couleur du text pour les print()
- from termcolor import colored
- # clés API Binance
- API_KEY = "C37gCN8g7WUfzakDkWlkGCBj63sgaT5tqf8NGj41ZU0z7hyCBFSMtxoVjLAE5ze9"
- API_SECRET = "7wNfvBJx6zYy4h8yUdk5OH15DcwML60mW8lFXJz0Lvwb4cmgNH23tXOPQ9b395ev"
- # Initialisation du client Binance, instance
- client = Client(API_KEY, API_SECRET)
- # Paramètres globaux
- symbol = 'BTCUSDT'
- data_window = deque(maxlen=151) # stockage des 151 dernières valeurs de closes
- time_window = deque(maxlen=151) # stockage des timestamps associés
- temma_window =deque(maxlen=151) # stockage des TEMA20 associés
- # Paramètres pour le calcul du TEMA et du slope
- TEMA_period = 20 # nombre de points pour calculer TEMA20
- slope_window = 3 # nombre de points pour calculer la pente
- # Variable simulation
- usdt_balance = 100.000000 # Balance initiale en USDT
- btc_balance = 0.000000 # Balance initiale en BTC
- last_buy_price = 0.000000
- # Seuils
- bearish_slope_threshold = -0.084 # seuil pour détecter une pente baissière forte (-0.01, -0.087)
- bullish_slope_threshold = 0.06 # seuil pour détecter une pente haussière forte (0.23, 0.13)
- # On considère soit "en position" (buy) soit "hors position"
- current_position = "none"
- # Calcul de la TEMA avec TA-Lib. Renvoie la dernière valeur calculée.
- def compute_TEMA(data, period):
 - if len(data) < period:
 - return None
 - np_data = np.array(data, dtype=float)
 - temma = talib.TEMA(np_data, timeperiod=period)
 - return temma[-1]
- # Calcul de la pente sur les 'window' derniers points par régression linéaire.
- def compute_slope(data, window=slope_window):
 - if len(data) < window:
 - return None

```

y = np.array(data[-window:], dtype=float)
x = np.arange(window)
slope, _ = np.polyfit(x, y, 1)
return slope

# fonction pour définir la couleur de slope selon sa valeur
def colorize_slope(slope):
    if slope < 0:
        return f'\033[91m{slope:.6f}\033[0m' # Rouge pour une valeur négative
    elif slope == 0:
        return f'\033[97m{slope:.6f}\033[0m' # Blanc pour une valeur nulle
    else:
        return f'\033[92m{slope:.6f}\033[0m' # Vert pour une valeur positive

# Définition de la fonction process_message qui traite les messages entrants (des données de marché).
# 'msg' est l'argument, qui représente un message contenant des informations sur les prix.
def process_message(msg):
    # Déclaration des variables globales utilisées dans cette simulation
    global current_position, usdt_balance, btc_balance, last_buy_price
    # 'c' indique qu'il contient des informations sur le prix actuel.
    if 'c' in msg:
        # Conversion du prix en float
        price = float(msg['c'])
        # Récupération du temps de l'événement en millisecondes
        event_time = msg['E']
        # Mise à jour des fenêtres de données
        data_window.append(price)
        time_window.append(event_time)
        # Si la fenêtre de données contient au moins 151 éléments (pour une analyse complète, TEMA50 !)
        if len(data_window) >= 151:
            # Calcul du TEMA20 et TEMA50
            tema20 = compute_TEMA(list(data_window), period=TEMA_period)
            tema_window.append(tema20)
            tema50 = compute_TEMA(list(data_window), period=50)
            # Si la fenêtre de TEMA contient au moins 61 éléments pour slope
            if len(tema_window) >= 61:
                # Calcul slope
                slope = compute_slope(list(tema_window), window=slope_window)
                # valeurs de TEMA20 pour détecter extrêmes
                recent_data = list(tema_window)[-11:]
                recent_max = max(recent_data)
                recent_min = min(recent_data)
                # Vente (si en position et conditions remplies)
                if current_position == "buy" and tema20 <= recent_max * 0.9999999 and
                btc_balance >= 0.0001: #and price >= last_buy_price * 1.001
                    if slope is not None and slope < bearish_slope_threshold and tema20 >
                    (tema50 - 0.4):
                        usdt_balance += btc_balance * price # Conversion de BTC en USDT
                        btc_balance = 0.000000 # Réinitialisation après vente
                        last_buy_price = 0.000000 # Réinitialisation du prix d'achat
                        print(f"[{time.ctime(event_time/1000)}] 🔍 VENTE à ₩{price:.6f}
(pente: {colorize_slope(slope)}), ⚡: {btc_balance:.6f}, $ : {usdt_balance:.6f}, prix
d'achat {last_buy_price:.2f}")
                        current_position = "none"
                # Achat (si hors position et conditions remplies) ⚡
                if current_position == "none" and tema20 >= recent_min * 1.0000001 and
                usdt_balance >= 10:

```

```

if slope is not None and slope > bullish_slope_threshold and tema20 <
(tema50 - 0.8):
    btc_balance = usdt_balance / price # Achat du maximum possible
    usdt_balance = 0.000000 # Tout l'USDT est converti en BTC
    last_buy_price = price # Enregistrer le prix d'achat
    current_position = "buy"
    print(f"[{time.ctime(event_time/1000)}] 🚀 ACHAT à 💰 {price:.6f}
(pente: {colorize_slope(slope)}), 💸: {btc_balance:.6f}, $ : 0.00")
    # Affichage de l'état de la parabole
    if tema20 < tema50:
        print(f"[{time.ctime(event_time/1000)}] ❤️ U (pente:
{colorize_slope(slope)}): (TEMA20: {f'\u033[94m{tema20:.2f}\u033[0m'} < TEMA50:
{f'\u033[93m{tema50:.2f}\u033[0m'}) 💰: {price:.6f}")
    else:
        print(f"[{time.ctime(event_time/1000)}] ❤️ n (pente:
{colorize_slope(slope)}):( (TEMA20: {f'\u033[94m{tema20:.2f}\u033[0m'} >= TEMA50:
{f'\u033[93m{tema50:.2f}\u033[0m'}) 💰: {price:.6f}")
# fonction principale 'main()' qui démarre le processus de gestion du websocket
def main():
    # Boucle infinie pour gérer la connexion WebSocket et les erreurs
    while True:
        try:
            # Création d'une instance du 'ThreadingWebsocketManager' en utilisant la clé API
            et le secret API
            # Cela permet de se connecter à Binance via WebSocket de manière multithreadée
            twm = ThreadingWebsocketManager(api_key=API_KEY, api_secret=API_SECRET)
            # Démarrage du WebSocket Manager
            twm.start()
            # Démarrage du socket pour récupérer les données de ticker en temps réel pour un
            symbole donné
            # Le callback 'process_message' sera appelé chaque fois qu'un message est reçu
            pour traiter les données
            twm.start_symbol_ticker_socket(callback=process_message, symbol=symbol)
            # Boucle interne pour maintenir le WebSocket ouvert
            while True:
                time.sleep(1)
            # Si une exception se produit (erreur de connexion, etc.), cela attrape l'erreur
            except Exception as e:
                print(f"Erreur détectée : {e}. Redémarrage du websocket dans 5s...")
                time.sleep(5) # Pause avant redémarrage
            # Exécution du script
            if __name__ == '__main__':
                main()

```

3. Exemples Résultats bot de tranding#IA 🚀

➡ EXEMPLE ACHAT :

[Sun Feb 16 13:20:22 2025] ❤️ n (pente: 0.209656):((TEMA20: 97233.28 >= TEMA50: 97233.15) 💰 : 97232.870000

[Sun Feb 16 13:20:23 2025] ❤️ n (pente: 0.148419):((TEMA20: 97233.40 >= TEMA50: 97233.40) 💰 : 97232.870000

[Sun Feb 16 13:20:24 2025] ❤️ U (pente: 0.097707):((TEMA20: 97233.48 < TEMA50: 97233.62) 💰 : 97232.870000

[Sun Feb 16 13:20:25 2025] ❤️ U (pente: 0.055870):((TEMA20: 97233.51 < TEMA50: 97233.80) 💰 : 97232.860000

[Sun Feb 16 13:20:26 2025] ❤️ U (pente: 0.024087):((TEMA20: 97233.52 < TEMA50: 97233.97) 💰 : 97232.870000

[Sun Feb 16 13:20:27 2025] 💚 U (pente: 0.000666):) (TEMA20: 97233.51 < TEMA50: 97234.11) 💰 : 97232.870000

[Sun Feb 16 13:20:28 2025] 💚 U (pente: -0.018427):) (TEMA20: 97233.49 < TEMA50: 97234.22) 💰 : 97232.870000

[Sun Feb 16 13:20:29 2025] 🔥 ACHAT à 💰 97236.510000 (pente: 0.439405), 💰 : 0.001032, 💰 : 0.00

[Sun Feb 16 13:20:30 2025] 💚 U (pente: 0.439405):) (TEMA20: 97234.39 < TEMA50: 97234.73) 💰 : 97236.510000

[Sun Feb 16 13:20:31 2025] ❤️ n (pente: 0.649447):) (TEMA20: 97235.69 >= TEMA50: 97235.60) 💰 : 97236.520000

[Sun Feb 16 13:20:32 2025] ❤️ n (pente: 0.509620):) (TEMA20: 97236.14 >= TEMA50: 97235.97) 💰 : 97236.520000



➡ EXEMPLE VENTE :

[Sun Feb 16 13:23:05 2025] 💚 U (pente: 0.000045):) (TEMA20: 97239.91 < TEMA50: 97239.92) 💰 : 97239.920000

[Sun Feb 16 13:23:06 2025] 💚 U (pente: 0.000054):) (TEMA20: 97239.91 < TEMA50: 97239.92) 💰 : 97239.910000

[Sun Feb 16 13:23:07 2025] ❤️ n (pente: 1.260586):) (TEMA20: 97242.44 >= TEMA50: 97241.02) 💰 : 97249.640000

[Sun Feb 16 13:23:08 2025] ❤️ n (pente: 2.291606):) (TEMA20: 97244.50 >= TEMA50: 97242.03) 💰 : 97249.650000

[Sun Feb 16 13:23:09 2025] ❤️ n (pente: 1.863139):) (TEMA20: 97246.16 >= TEMA50: 97242.97) 💰 : 97249.640000

[Sun Feb 16 13:23:10 2025] ❤️ n (pente: 1.613579):) (TEMA20: 97247.72 >= TEMA50: 97243.93) 💰 : 97250.510000

[Sun Feb 16 13:23:11 2025] ❤️ n (pente: 1.462244):) (TEMA20: 97249.09 >= TEMA50: 97244.86) 💰 : 97250.970000

[Sun Feb 16 13:23:12 2025] ❤️ n (pente: 1.250160):) (TEMA20: 97250.22 >= TEMA50: 97245.74) 💰 : 97251.210000

[Sun Feb 16 13:23:13 2025] ❤️ n (pente: 1.102502):) (TEMA20: 97251.29 >= TEMA50: 97246.63) 💰 : 97251.910000

[Sun Feb 16 13:23:14 2025] ❤️ n (pente: 0.959551):) (TEMA20: 97252.14 >= TEMA50: 97247.46) 💰 : 97252.010000

[Sun Feb 16 13:23:15 2025] ❤️ n (pente: 0.750150):) (TEMA20: 97252.79 >= TEMA50: 97248.21) 💰 : 97252.020000

[Sun Feb 16 13:23:16 2025] ❤️ n (pente: 0.564003):) (TEMA20: 97253.27 >= TEMA50: 97248.90) 💰 : 97252.020000

[Sun Feb 16 13:23:17 2025] ❤️ n (pente: 0.409722):) (TEMA20: 97253.61 >= TEMA50: 97249.53) 💰 : 97252.020000

[Sun Feb 16 13:23:18 2025] ❤️ n (pente: 0.282487):) (TEMA20: 97253.84 >= TEMA50: 97250.09) 💰 : 97252.010000

[Sun Feb 16 13:23:19 2025] ❤️ n (pente: 0.180747):) (TEMA20: 97253.97 >= TEMA50: 97250.61) 💰 : 97252.020000

[Sun Feb 16 13:23:20 2025] ❤️ n (pente: 0.100455):) (TEMA20: 97254.04 >= TEMA50: 97251.07) 💰 : 97252.020000

[Sun Feb 16 13:23:21 2025] ❤️ n (pente: 0.034248):) (TEMA20: 97254.04 >= TEMA50: 97251.49) 💰 : 97252.010000

[Sun Feb 16 13:23:22 2025] ❤️ n (pente: -0.016237):) (TEMA20: 97254.00 >= TEMA50: 97251.86) 💰 : 97252.020000

[Sun Feb 16 13:23:23 2025] ❤️ n (pente: -0.053652):(TEMA20: 97253.93 >= TEMA50: 97252.19) 💰 : 97252.020000

[Sun Feb 16 13:23:24 2025] 🔥 VENTE à 💰 97252.010000 (pente: -0.084144), 💳 : 0.000000, 💸 : 100.396936,prix d'achat 0.00

[Sun Feb 16 13:23:24 2025] ❤️ n (pente: -0.084144):(TEMA20: 97253.84 >= TEMA50: 97252.48) 💰 : 97252.010000

[Sun Feb 16 13:23:25 2025] ❤️ n (pente: -0.105029):(TEMA20: 97253.72 >= TEMA50: 97252.74) 💰 : 97252.020000

[Sun Feb 16 13:23:26 2025] ❤️ n (pente: -0.119344):(TEMA20: 97253.60 >= TEMA50: 97252.97) 💰 : 97252.010000

[Sun Feb 16 13:23:27 2025] ❤️ n (pente: -0.129622):(TEMA20: 97253.46 >= TEMA50: 97253.16) 💰 : 97252.010000

[Sun Feb 16 13:23:28 2025] 💚 U (pente: -0.134075):(TEMA20: 97253.33 < TEMA50: 97253.33) 💰 : 97252.010000

[Sun Feb 16 13:23:29 2025] 💚 U (pente: -0.135153):(TEMA20: 97253.19 < TEMA50: 97253.48) 💰 : 97252.010000

s



- Utilisation d'ordre du marcher.
- Backtesting sur des données historiques pour optimiser la stratégie.



4. Optimisations et prochaines étapes

📌 To Do :

- **Optimisation et intégrations d'indicateurs. Ajout d'IA (Régression Logistique, LTSM).**

5. Bot Simple

📌 stratégie (1/m) :

- **Si TEMA20 > TEMA50 , ACHAT.**
- **Si TEMA20 < TEMA50, VENTE.**

📌 script bot.py:

```
• import time
• import sqlite3
• import pandas as pd
• from binance.client import Client
• from database import init_db, save_candle
• from indicators import calculate_indicators
• from strategy import check_signal
• from telegram_bot import send_telegram_message # 🚀 Import du module Telegram
• import hmac
• import time
• import hashlib
• import requests
• from urllib.parse import urlencode
• import requests
• from binance.client import Client
•
• API_KEY = "3qbUh0BQuLBRVH0s0uipwxcJ0TfXe24RwyJ4Y6X8QW3J62nrgAYKRLQt9kzEhZKM"
• API_SECRET = "8Mu0JC0D444145iRQWzsogABddkzC8sK4DvYCrL6K6HgpTWJ6r3tqrO32DxD0d1L"
• # Clés API
• KEY = "C37gCN8g7WUfzakDkWlkGCBj63sgaT5tqf8NGj41ZU0z7hyCBFSMtxoVjLAE5ze9"
• SECRET = "7wNfvBJx6zYy4h8yUdk50H15DcwML60mW8lFXJzOLvwB4cmgNH23tXOPQ9b395ev"
• client = Client(API_KEY, API_SECRET)
• SYMBOL = "BTCUSDT"
• INTERVAL = Client.KLINE_INTERVAL_1MINUTE
• # URL Binance
• #BASE_URL = "https://api.binance.com"
• BASE_URL = "https://testnet.binance.vision"
• # Récupérer le temps du serveur Binance
• def get_binance_server_time():
•     response = requests.get(BASE_URL + "/api/v3/time")
•     server_time = response.json()["serverTime"]
•     local_time = int(time.time() * 1000)
•     print(f"Local time: {local_time}, Binance server time: {server_time}")
•     print(f"Time difference: {server_time - local_time} ms")
•     return server_time
• # Générer une signature HMAC-SHA256
• def hashing(query_string):
```

```
•     return hmac.new(SECRET.encode(), query_string.encode(), hashlib.sha256).hexdigest()
• # Créer une session avec les headers
• session = requests.Session()
• session.headers.update({"Content-Type": "application/json", "X-MBX-APIKEY": KEY})
• # Fonction pour envoyer une requête signée
• def send_signed_request(http_method, url_path, payload={}):
•     payload["timestamp"] = get_binance_server_time() # Synchronisation avec Binance
•     payload["recvWindow"] = 5000 # Augmenter la fenêtre de réception
•     query_string = urlencode(sorted(payload.items())) # Trier les paramètres
•     signature = hashing(query_string) # Générer la signature
•     url = f"{BASE_URL}{url_path}?{query_string}&signature={signature}"
•     #print(f"{http_method} {url}") # Debugging
•     response = session.request(http_method, url)
•     if response.status_code == 200:
•         return response.json()
•     else:
•         #print("Error:", response.json()) # Debug
•         return response.json()
• # Fonction pour envoyer une requête publique (sans signature)
• def send_public_request(url_path, payload={}):
•     query_string = urlencode(payload)
•     url = f"{BASE_URL}{url_path}"
•     if query_string:
•         url += f"?{query_string}"
•     #print("GET", url) # Debugging
•     response = session.get(url)
•     return response.json()
• # Vérifier la synchronisation avec Binance
• get_binance_server_time()
• # Récupérer les informations du compte
• response = send_signed_request("GET", "/api/v3/account")
• print(response)
• def get_btc_balance():
•     account_info = send_signed_request("GET", "/api/v3/account")
•     #print("Réponse de Binance:", account_info) # 🕵️ Debug
•     if "balances" not in account_info:
•         print("⚠️ Erreur: 'balances' n'existe pas dans la réponse !")
•         return 0.0 # Retourne 0 pour éviter le crash
•     for balance in account_info["balances"]:
•         if balance["asset"] == "BTC":
•             return float(balance["free"])
•     return 0.0
• btc_balance = get_btc_balance()
• print(f"Solde BTC disponible : {btc_balance}")
• def get_USDT_balance():
•     account_info = send_signed_request("GET", "/api/v3/account")
•     for balance in account_info["balances"]:
•         if balance["asset"] == "USDT":
•             return float(balance["free"])
•     return 0.0
• USDT_balance = get_USDT_balance()
• print(f"Solde USDT disponible : {USDT_balance}")
• def get_price(symbol):
•     """Récupère le prix actuel du marché pour un symbole donné."""
•     response = requests.get(f"https://api.binance.com/api/v3/ticker/price?symbol={symbol}")
•     return float(response.json()["price"])
• price = get_price("BTCUSDT")
```

```

• print(f"Le prix actuel du BTC en USDT est de {price}")
• def BUYS():
•     USDT_balance = get_USDT_balance()
•     a = round(USDT_balance * 0.9, 4)
•     b = round(a / price, 4)
•     print(b)
•     # Passer un ordre d'achat (market)
•     buy_params = {
•         "symbol": "BTCUSDT",
•         "side": "BUY",
•         "type": "MARKET",
•         "quantity": b,
•     }
•     buy_response = send_signed_request("POST", "/api/v3/order", buy_params)
•     print("Buy order response:", buy_response)
• def SELLS():
•     btc_balance = get_btc_balance()
•     # Passer un ordre de vente (market)
•     sell_params = {
•         "symbol": "BTCUSDT",
•         "side": "SELL",
•         "type": "MARKET",
•         "quantity": btc_balance,
•     }
•     sell_response = send_signed_request("POST", "/api/v3/order", sell_params)
•     print("Sell order response:", sell_response)
• # Afficher uniquement les balances non nulles
• def Get_balance_utile():
•     if "balances" in response:
•         balances = response["balances"]
•         for asset in balances:
•             if float(asset["free"]) > 0 or float(asset["locked"]) > 0:
•                 print(f"{asset['asset']}: {asset['free']} (free), {asset['locked']} (locked)")
•         else:
•             print("Erreur : Impossible de récupérer les balances.")
• import sqlite3
• import pandas as pd
• DB_FILE = "C:\\\\TRADE_BOT\\\\trading_data.db"
• def get_historical_data():
•     """Vérifie si les données historiques existent déjà, sinon les récupère depuis Binance."""
•     print("⌚ Vérification des données historiques...")
•     # Connexion à la base de données
•     with sqlite3.connect(DB_FILE) as conn:
•         cursor = conn.cursor()
•         # Vérifier si la table 'market_data' existe et contient des données
•         cursor.execute("SELECT COUNT(*) FROM market_data")
•         count = cursor.fetchone()[0]
•         if count == 0:
•             print("⚠️ Données historiques trouvées dans la base de données. Chargement...")
•             # Charger les données existantes
•             df = pd.read_sql_query("SELECT * FROM market_data", conn)
•             df['time'] = pd.to_datetime(df['time']).astype(str)
•         else:
•             print("⚠️ Aucune donnée historique trouvée. Récupération des données depuis Binance...")

```

```

# Récupérer les données historiques depuis Binance si elles n'existent pas
klines = client.get_historical_klines(SYMBOL, INTERVAL, "1 Feb, 2025")
df = pd.DataFrame(klines, columns=['time', 'open', 'high', 'low', 'close',
'volume', 'close_time',
                                'quote_asset_volume', 'num_trades',
'taker_buy_base', 'taker_buy_quote', 'ignore'])
df = df[['time', 'open', 'high', 'low', 'close']].astype(float)
df['time'] = pd.to_datetime(df['time'], unit='ms').astype(str)
# Calculer les indicateurs et sauvegarder dans la base de données
df = calculate_indicators(df)
# Sauvegarder les nouvelles bougies dans la table 'market_data'
for _, row in df.iterrows():
    save_candle(tuple(row)) # Sauvegarde dans la base
print("☒ Historique récupéré et sauvegardé dans la base de données.")
return df

# Fonction pour récupérer les données de la base de données SQLite
def backtest():
    df = get_historical_data() # Récupère toutes les bougies historiques
    balance_usdt = 1000
    balance_btc = 0
    history = []
    # Parcourir toutes les bougies de l'historique
    for i in range(1, len(df)): # On commence à 1 pour avoir i-1
        last_row = df.iloc[i - 1]
        current_row = df.iloc[i]
        # Détermination du signal
        signal = None
        # Récupérer le prix actuel (par exemple, le prix de clôture)
        current_price = current_row['close']
        # Signal d'achat basé sur RSI(6), RSI(12), RSI(24) et croisement de TEMA(7),
        TEMA(25), TEMA(99)
        if (last_row['TEMA20'] < last_row['TEMA50']):
            signal = "BUY"
            #print(f"Achat exécuté à {current_price}")
        # Signal de vente basé sur RSI(6), RSI(12), RSI(24) et croisement de TEMA(7),
        TEMA(25), TEMA(99)
        if (last_row['TEMA20'] > last_row['TEMA50']):
            # Vérifier que le prix actuel est supérieur au dernier prix d'achat
            signal = "SELL"
            #print(f"Vente exécutée à {current_price}")
        if signal is None:
            continue
        timestamp = current_row['time']
        # Achat si le signal est "BUY"
        if signal == "BUY" and balance_usdt > 100 :
            amount_to_buy = balance_usdt / current_price
            balance_btc = amount_to_buy
            balance_usdt = 0
            history.append(f"{timestamp} - BUY at {current_price} BTC")
            #print(df.iloc[i - 1 : i + 1]) # Afficher les 2 dernières lignes
        # Vente si le signal est "SELL"
        if signal == "SELL" and balance_btc > 0.0001 :
            amount_to_sell = balance_btc * current_price
            balance_usdt = amount_to_sell
            balance_btc = 0
            history.append(f"{timestamp} - SELL at {current_price} BTC")
            #print(df.iloc[i - 1 : i + 1]) # Afficher les 2 dernières lignes

```

```

•     #print(f"timestamp} | USDT: {balance_usdt:.2f}, BTC: {balance_btc:.6f}, Buy Price:
• {buy_price if buy_price is not None else 'N/A'}, Sell Price: {sell_price if sell_price is
• not None else 'N/A'}, Signal: {signal}")
•     print("\nBacktest terminé")
•     print(f"Solde final USDT: {balance_usdt:.2f}, Solde final BTC: {balance_btc:.6f}")
•     #print("Historique des transactions:")
•     #for transaction in history:
•         #print(transaction)
• def execute_trade(action, data):
•     balance_usdt = get_USDT_balance()
•     balance_btc = get_btc_balance()
•     # Appel de la fonction selon l'action
•     if action == "BUY" and balance_usdt >= 10:
•         BUYs()
•         """Exécute une action d'achat ou de vente et envoie à Telegram."""
•         message = f"⚠️ *SIGNAL DÉTECTÉ* : {action} 📡\n" \
•             f"🕒 *Temps* : {data['time']}\n" \
•             f"฿ *Prix* : {data['close']:.2f}\n" \
•             f"📈 *RSI14* : {data['RSI14']:.2f}\n" \
•             f"📈 *RSI50* : {data['RSI50']:.2f}\n" \
•             f"📊 *TEMA20* : {data['TEMA20']:.2f}\n" \
•             f"📊 *TEMA50* : {data['TEMA50']:.2f}\n" \
•             f"฿ *FAUX Solde USDT* : {get_USDT_balance()}\n" \
•             f"฿ *FAUX Solde BTC* : {get_btc_balance()}"
•         print(message) # Affichage en console
•         send_telegram_message(message) # ⚠️ Envoi sur Telegram
•     # Appel de la fonction selon l'action
•     elif action == "SELL" and balance_btc >= 0.0001:
•         SELLs()
•         """Exécute une action d'achat ou de vente et envoie à Telegram."""
•         message = f"⚠️ *SIGNAL DÉTECTÉ* : {action} 📡\n" \
•             f"🕒 *Temps* : {data['time']}\n" \
•             f"฿ *Prix* : {data['close']:.2f}\n" \
•             f"📈 *RSI14* : {data['RSI14']:.2f}\n" \
•             f"📈 *RSI50* : {data['RSI50']:.2f}\n" \
•             f"📊 *TEMA20* : {data['TEMA20']:.2f}\n" \
•             f"📊 *TEMA50* : {data['TEMA50']:.2f}\n" \
•             f"฿ *FAUX Solde USDT* : {get_USDT_balance()}\n" \
•             f"฿ *FAUX Solde BTC* : {get_btc_balance()}"
•         print(message) # Affichage en console
•         send_telegram_message(message) # ⚠️ Envoi sur Telegram
•     # Appel de la fonction selon l'action
• def run_bot():
•     """Boucle principale du bot, détecte les signaux UNIQUEMENT en live."""
•     init_db()
•     # Charger l'historique sans signal
•     backtest()
•     print("🌐 Mode LIVE activé : Détection des signaux uniquement en temps réel !")
•     send_telegram_message("🚀 *Bot Trading * : en temps réel sur Binance 💰!")
•     live_mode = False # Activer la détection des signaux seulement après le premier live
•     tick
•         while True:
•             try:
•                 candles = client.get_klines(symbol=SYMBOL, interval=INTERVAL, limit=99)
•                 df = pd.DataFrame(candles, columns=['time', 'open', 'high', 'low', 'close',
• 'volume', 'close_time'],

```

```

    'quote_asset_volume', 'num_trades',
    'taker_buy_base', 'taker_buy_quote', 'ignore'])
df = df[['time', 'open', 'high', 'low', 'close']].astype(float)
df['time'] = pd.to_datetime(df['time'], unit='ms').astype(str)
df = calculate_indicators(df)
print(df.head())
save_candle(tuple(df.iloc[-1]))
# 📈 Affichage du dernier DataFrame en live
print("\n📊 Dernières données LIVE :")
print(df.tail(5))
# 📈 Envoi du dernier DataFrame à Telegram (optionnel)
last_data = df.iloc[-1]
print(df.columns) # Ajoute cette ligne juste avant l'envoi de la mise à jour à
Telegram
if 'RSI14' and 'close' and 'RSI50' in last_data:
    telegram_message = f"📊 *Mise à jour LIVE TEST* {last_data['time']}\n" \
        f"💰 *Prix* : {last_data['close']:.2f}\n" \
        f"📈 *RSI14* : {last_data['RSI14']:.2f}\n" \
        f"📈 *RSI50* : {last_data['RSI50']:.2f}\n" \
        f"📊 *TEMA20* : {last_data['TEMA20']:.2f}\n" \
        f"📊 *TEMA50* : {last_data['TEMA50']:.2f}\n" \
        f"💵 *FAUX Solde USDT* : {get_USDT_balance()}\n" \
        f"BTC *FAUX Solde BTC* : {get_btc_balance()}"
    print(telegram_message) # Affichage en console
    send_telegram_message(telegram_message)
else:
    print("⚠️ Colonnes RSI manquantes ou incorrectes.")
if live_mode: # Ne détecter les signaux qu'en live
    check_signal(df, execute_trade)
else:
    live_mode = True # Activer la détection de signaux pour la prochaine
itération
    time.sleep(60) # Attendre 1 minute avant la prochaine itération
except Exception as e:
    print(f"⚠️ Erreur : {e}")
    time.sleep(10)
if __name__ == "__main__":
    run_bot()

```

📌 script database.py:

```

import sqlite3
DB_FILE = "C:\\\\TRADE_BOT\\\\trading_data.db"
def init_db():
    """Initialise la base de données avec la table des prix et indicateurs."""
    with sqlite3.connect(DB_FILE) as conn:
        cursor = conn.cursor()
        cursor.execute('''CREATE TABLE IF NOT EXISTS market_data (
            time TEXT PRIMARY KEY,
            open REAL,
            high REAL,
            low REAL,
            close REAL,
            RSI14 REAL,
            RSI50 REAL,
            TEMA20 REAL,
            TEMA50 REAL
        )'''')
        conn.commit()

```

```

• def save_candle(data):
•     """Sauvegarde une bougie dans la base de données."""
•     with sqlite3.connect(DB_FILE) as conn:
•         cursor = conn.cursor()
•         cursor.execute('''INSERT OR REPLACE INTO market_data
•                         (time, open, high, low, close, RSI14, RSI50, TEMA20, TEMA50)
•                         VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)''', data)
•     conn.commit()

```

📌 script indicators.py:

```

• import pandas as pd
• import ta
• # Calcul du RSI
• def calculate_rsi_14(data, window=14):
•     delta = data.diff()
•     gain = (delta.where(delta > 0, 0)).rolling(window=window).mean()
•     loss = (-delta.where(delta < 0, 0)).rolling(window=window).mean()
•     loss = loss.replace(0, 1e-10) # Empêche la division par zéro
•     rs = gain / loss
•     rsi_14 = 100 - (100 / (1 + rs))
•     return rsi_14
• def calculate_rsi_50(data, window=50):
•     delta = data.diff()
•     gain = (delta.where(delta > 0, 0)).rolling(window=window).mean()
•     loss = (-delta.where(delta < 0, 0)).rolling(window=window).mean()
•     loss = loss.replace(0, 1e-10) # Empêche la division par zéro
•     rs = gain / loss
•     rsi_50 = 100 - (100 / (1 + rs))
•     return rsi_50
• # Fonction pour calculer une moyenne mobile simple (SMA)
• def SMA(series, period):
•     return series.rolling(window=period).mean()
• # Fonction pour calculer MA100
• def calculate_MA100(data):
•     return SMA(data, 100)
• # Fonction pour calculer MA200
• def calculate_MA200(data):
•     return SMA(data, 200)
• # Calcul de l'EMA
• def EMA(series, period):
•     return series.ewm(span=period, adjust=False).mean()
• # Calcul du TEMA
• def TEMA(series, period):
•     ema1 = EMA(series, period)
•     ema2 = EMA(ema1, period)
•     ema3 = EMA(ema2, period)
•     return 3 * (ema1 - ema2) + ema3
• def calculate_indicators(df):
•     """Calcule RSI et TEMA."""
•     df['RSI14'] = calculate_rsi_14(df['close'])
•     df['RSI50'] = calculate_rsi_50(df['close'])
•     df['TEMA20'] = TEMA(df['close'], 20)
•     df['TEMA50'] = TEMA(df['close'], 50)
•     return df

```

📌 strategy.py:

```

• def check_signal(df, execute_trade):
•     """Vérifie s'il y a un signal d'achat ou de vente et exécute une action."""

```

```

•     last_row = df.iloc[-1]
•     # Récupérer le prix actuel (par exemple, le prix de cloture)
•     # Signal d'achat basé sur RSI(6), RSI(12), RSI(24) et croisement de TEMA(7), TEMA(25),
•     TEMA(99)
•         if (last_row['TEMA20'] > last_row['TEMA50']):
•             execute_trade("BUY", last_row)
•         # Signal de vente basé sur RSI(6), RSI(12), RSI(24) et croisement de TEMA(7), TEMA(25),
•     TEMA(99)
•         elif (last_row['TEMA20'] < last_row['TEMA50']):
•             execute_trade("SELL", last_row)

```

📌 script telegram_bot.py:

```

• import requests
• # 🔑 Remplace par ton TOKEN de bot et ID de canal
• TELEGRAM_BOT_TOKEN = "8027599831:AAEhXtUFwv4ZQc48YSwk-Uemm4BesZLBVHI"
• TELEGRAM_CHAT_ID = "-1002264309587"
• def send_telegram_message(message):
•     """Envoie un message à Telegram."""
•     url = f"https://api.telegram.org/bot{TELEGRAM_BOT_TOKEN}/sendMessage"
•     payload = {"chat_id": TELEGRAM_CHAT_ID, "text": message, "parse_mode": "Markdown"}
•     try:
•         requests.post(url, json=payload)
•     except Exception as e:
•         print(f"⚠️ Erreur envoi Telegram : {e}")

```

📌 script résultats:

```

C:\WINDOWS\system32\cmd + v - X
[TIME] [PRICE] [RSI14] [RSI50] [TEMA20] [TEMA50] [FAUX Solde USDT] [FAUX Solde BTC]
Local time: 1739710997905, Binance server time: 1739710997747
Time difference: -158 ms
Local time: 1739710998440, Binance server time: 1739710998282
Time difference: -158 ms
* *Mise à jour LIVE TEST* 2025-02-16 13:03:00
* *Prix* : 97090.30
* *RSI14* : 26.27
* *RSI50* : 30.71
* *TEMA20* : 97089.55
* *TEMA50* : 97090.00
* *FAUX Solde USDT* : 391.8353012
* *FAUX Solde BTC* : 0.3764

```

Local time: 1739711001657, Binance server time: 1739711001500

Time difference: -157 ms

Local time: 1739711002195, Binance server time: 1739711002038

Time difference: -157 ms

🔥 *SIGNAL DÉTECTÉ* : SELL 🔈

📅 *Temps* : 2025-02-16 13:03:00

💰 *Prix* : 97090.30

📈 *RSI14* : 26.27

📈 *RSI50* : 30.71

📊 *TEMA20* : 97089.55

📊 *TEMA50* : 97090.00

💵 *FAUX Solde USDT* : 36936.6189646

🟡 *FAUX Solde BTC* : 0.0

