**main.py**

import os, time, requests, statistics, math

from datetime import datetime

# --- Config (better to set as env vars in Railway) ---

BOT\_TOKEN = os.getenv("BOT\_TOKEN") # telegram bot token

CHAT\_ID = os.getenv("CHAT\_ID") # telegram chat id (numeric)

SYMBOL = os.getenv("SYMBOL", "BTCUSDT")

SLEEP\_SECONDS = int(os.getenv("SLEEP\_SECONDS", "300")) # 5 minutes

# thresholds (strict mode)

VOL\_RATIO\_MIN = float(os.getenv("VOL\_RATIO\_MIN", "1.15")) # current\_vol / avg\_vol

OI\_PC\_MIN = float(os.getenv("OI\_PC\_MIN", "5")) # percent change vs avg

# SL/TP in points (change as you like)

SL\_PTS = int(os.getenv("SL\_PTS", "100"))

TP1\_PTS = int(os.getenv("TP1\_PTS", "170"))

TP2\_PTS = int(os.getenv("TP2\_PTS", "250"))

TP3\_PTS = int(os.getenv("TP3\_PTS", "300"))

# anti-spam: don't send same type of signal within X minutes

MIN\_REPEAT\_MINUTES = int(os.getenv("MIN\_REPEAT\_MINUTES", "30"))

# Binance endpoints (public)

OI\_API = "https://fapi.binance.com/futures/data/openInterestHist"

KLINES\_API = "https://api.binance.com/api/v3/klines"

# helper: send telegram

def send\_telegram(text):

if not BOT\_TOKEN or not CHAT\_ID:

print("Telegram credentials not set. Skipping send.")

return

url = f"https://api.telegram.org/bot{BOT\_TOKEN}/sendMessage"

payload = {"chat\_id": CHAT\_ID, "text": text, "parse\_mode": "Markdown"}

try:

r = requests.post(url, json=payload, timeout=10)

print("Telegram:", r.status\_code, r.text[:200])

except Exception as e:

print("Telegram send error:", e)

def fetch\_oi(symbol, period="5m", limit=12):

params = {"symbol": symbol, "period": period, "limit": limit}

r = requests.get(OI\_API, params=params, timeout=10)

r.raise\_for\_status()

data = r.json()

oi\_list = []

for item in data:

# API returns keys like 'sumOpenInterest' or 'openInterest'; be flexible

oi = item.get("sumOpenInterest") or item.get("openInterest") or item.get("sumOpenInterestValue")

try:

oi\_list.append(float(oi))

except:

oi\_list.append(0.0)

return oi\_list

def fetch\_klines\_vol\_price(symbol, interval="5m", limit=50):

params = {"symbol": symbol, "interval": interval, "limit": limit}

r = requests.get(KLINES\_API, params=params, timeout=10)

r.raise\_for\_status()

klines = r.json()

volumes = [float(k[5]) for k in klines] # index 5 = volume

closes = [float(k[4]) for k in klines] # index 4 = close price

return volumes, closes

def compute\_confidence(oi\_pct, vol\_ratio):

# basic heuristic -> scale to 0-100

score = 30 + min(40, abs(oi\_pct)\*2) + min(30, max(0, (vol\_ratio-1)\*100))

return int(max(0, min(100, score)))

def main\_loop():

last\_signal = {"type": None, "time": 0}

send\_telegram("🚀 Bot started. Only STRONG BUY/SELL will be sent ✅")

while True:

try:

volumes, closes = fetch\_klines\_vol\_price(SYMBOL, interval="5m", limit=50)

oi\_list = fetch\_oi(SYMBOL, period="5m", limit=12)

if not oi\_list or not volumes or not closes:

print("No data, sleeping...")

time.sleep(10)

continue

current\_vol = volumes[-1]

avg\_vol = statistics.mean(volumes[-21:-1]) if len(volumes) >= 22 else statistics.mean(volumes[:-1] or volumes)

vol\_ratio = current\_vol / (avg\_vol if avg\_vol>0 else 1)

current\_oi = oi\_list[-1]

avg\_oi = statistics.mean(oi\_list[:-1]) if len(oi\_list) >= 2 else statistics.mean(oi\_list)

oi\_pct = (current\_oi - avg\_oi) / (avg\_oi if avg\_oi>0 else 1) \* 100

last\_price = closes[-1]

print(f"[{datetime.utcnow().isoformat()}] Close={last\_price:.2f} | Vol={current\_vol} (Avg={avg\_vol:.0f}) | OI={current\_oi:.0f}")

print(f"VolRatio={vol\_ratio:.2f} | OI%={oi\_pct:.2f}")

signal = None

# STRONG BUY condition

if (current\_oi > avg\_oi\*(1 + OI\_PC\_MIN/100)) and (vol\_ratio >= VOL\_RATIO\_MIN):

signal = "BUY"

# STRONG SELL condition

elif (current\_oi < avg\_oi\*(1 - OI\_PC\_MIN/100)) and (vol\_ratio >= VOL\_RATIO\_MIN):

signal = "SELL"

now\_ts = time.time()

minutes\_since = (now\_ts - last\_signal["time"]) / 60

if signal:

# avoid duplicates

if last\_signal["type"] == signal and minutes\_since < MIN\_REPEAT\_MINUTES:

print("Duplicate signal within cool-down, skipping.")

else:

# build SL/TP

if signal == "BUY":

sl = round(last\_price - SL\_PTS, 2)

tp1 = round(last\_price + TP1\_PTS, 2)

tp2 = round(last\_price + TP2\_PTS, 2)

tp3 = round(last\_price + TP3\_PTS, 2)

direction = "📈 \*STRONG BUY\*"

else:

sl = round(last\_price + SL\_PTS, 2)

tp1 = round(last\_price - TP1\_PTS, 2)

tp2 = round(last\_price - TP2\_PTS, 2)

tp3 = round(last\_price - TP3\_PTS, 2)

direction = "📉 \*STRONG SELL\*"

conf = compute\_confidence(oi\_pct, vol\_ratio)

volpct = (vol\_ratio - 1) \* 100

message = (

f"{direction}\n\n"

f"\*Pair:\* {SYMBOL}\n"

f"\*Price:\* {last\_price:.2f}\n"

f"\*OI Δ:\* {oi\_pct:.2f}% | \*Vol %:\* {volpct:.1f}%\n"

f"\*Confidence:\* {conf}%\n\n"

f"\*Entry:\* {last\_price:.2f}\n"

f"\*SL:\* {sl}\n"

f"\*TP1:\* {tp1} | \*TP2:\* {tp2} | \*TP3:\* {tp3}\n\n"

f"\_Signal generated at {datetime.utcnow().strftime('%Y-%m-%d %H:%M UTC')}\_"

)

send\_telegram(message)

last\_signal = {"type": signal, "time": now\_ts}

else:

print("No strong signal.")

except Exception as e:

print("Main loop error:", e)

# sleep for configured time

time.sleep(SLEEP\_SECONDS)

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

main\_loop()