!mport os
!mport warn!ngs
!mport logg!ng

!mport argparse
!mport thread!ng
!mport t!me
!mport !tertools

!mport numpy as np
!mport pandas as pd
!mport polars as pl

!mport pandas_ta as ta
!mport async!o
!mport a!ohttp

!mport uu!d
from typ!ng !mport D!ct, L!st, Tuple
from collect!ons !mport deque
from sklearn.preprocess!ng !mport StandardScaler

from sklearn.m!xture !mport Gauss!anM!xture
from stable_basel!nes3 !mport SAC, PPO, A2C

from stable_basel!nes3.common.vec_env !mport DummyVecEnv, SubprocVecEnv, VecMon!tor

from stable_basel!nes3.common.callbacks !mport BaseCallback
from stable_basel!nes3.common.logger !mport conf!gure

!mport gymnas!um as gym
!mport tal!b
from vaderSent!ment.vaderSent!ment !mport Sent!mentIntens!tyAnalyzer
!mport pdfplumber

!mport mult!process!ng
!mport shap
!mport p!ckle

!mport cvxpy as cp

try:
!mport plotly.graph_objects as go
PLOTLY_YUKLU = True
except ImportError:
PLOTLY_YUKLU = False

pr!nt("Plotly kütüphanes! yüklü değ!l. Dashboard yer!ne CSV raporu
üret!lecek.")

warn!ngs.f!lterwarn!ngs("!gnore", category=UserWarn!ng, module="pandas_ta")

logg!ng.bas!cConf!g(
level=logg!ng.INFO,

format="%(asct!me)s - %(levelname)s - %(message)s",
handlers=[
logg!ng.F!leHandler("tra!n!ng_log.txt", mode='a'),

logg!ng.StreamHandler()
]
)

BOT_ADI = "JTTWS T!caret Botu" BOT_KULLANICI_ADI = "jttws_t!caret_bot" TELEGRAM_TOKEN = "8008545474:AAHansC5Xag1b9N96bMAGE0YLTfykXoOPyY" TELEGRAM_KULLANICI_ID = 1590841427 CSV_YOLLARI = {

'EURUSD': os.path.expanduser('~/Desktop/JTTWS/data/
EURUSD_2003_2024_ALL_tem!z.csv'),

'USDJPY': os.path.expanduser('~/Desktop/JTTWS/data/ USDJPY_2003_2024_ALL_tem!z.csv'),
'GBPUSD': os.path.expanduser('~/Desktop/JTTWS/data/ GBPUSD_2003_2024_ALL_tem!z.csv')

}SEMBOLLER = ['EURUSD', 'USDJPY', 'GBPUSD'] BASLANGIC_BAKIYESI = 25000.0 TEMEL_SPREAD = 0.0002 PRIME_ZAMANI = range(0, 24)

ISLEM_BASINA_MAKS_RISK = 0.02 ORIJINAL_MAKS_ACIK_POZISYON = 10 MIN_TICARET_GUNU = 1 MAX_LOT = 0.25 TARGET_PAYOUT = 2500.0 MAX_ACCOUNTS = 3 NEWS_API_KEY = None # Kullanıcıdan sağlanmalı KRIZ_DONEMLERI = {

2008: sl!ce(0, 10000), # 2008 f!nansal kr!z!

2020: sl!ce(50000, 60000), # COVID-19 kr!z!
'normal': sl!ce(20000, 30000)
}

args = None
anal!zc! = Sent!mentIntens!tyAnalyzer()
duygu_onbellek = {'skor': 0.0, 'zaman_damgas!': 0, 'met!n': '', 'h!story':

deque(maxlen=3)}
ozell!k_onbellek = {}
volat!l!te_tar!h! = deque(maxlen=100)
shap_degerler! = {}

def l!near_schedule(!n!t!al_value: float, f!nal_value: float) -> callable:
def schedule(progress: float) -> float:

!f progress >= 1.0:
return f!nal_value
return !n!t!al_value + (f!nal_value - !n!t!al_value) * progress
return schedule

class TelegramBot:
def __!n!t__(self, token: str, kullan!c!_!d: !nt):
self.token = token
self.kullan!c!_!d = kullan!c!_!d
self.ap!_url = f"https://ap!.telegram.org/bot{self.token}/sendMessage"

self.loop = async!o.new_event_loop()
thread!ng.Thread(target=self._run_loop, daemon=True).start()

def _run_loop(self):
async!o.set_event_loop(self.loop)
self.loop.run_forever()

async def send_message(self, text: str):
async w!th a!ohttp.Cl!entSess!on() as sess!on:
payload = {"chat_!d": self.kullan!c!_!d, "text": text, "parse_mode":

"HTML"}
async w!th sess!on.post(self.ap!_url, data=payload) as response:
return awa!t response.json()

def sync_send_message(self, text: str):
future = async!o.run_corout!ne_threadsafe(self.send_message(text),
self.loop)

return future.result()

def csv_ver!s!n!_yukle(csv_yolu: str) -> pd.DataFrame:
try:
!f not os.path.ex!sts(csv_yolu):
ra!se F!leNotFoundError(f"CSV dosyası bulunamadı: {csv_yolu}")
df_temp = pl.read_csv(

csv_yolu, !nfer_schema_length=10000, !gnore_errors=True,
truncate_ragged_l!nes=True
)

column_mapp!ng = {
'local_t!me': 't!me', 'Local t!me': 't!me', 'open': 'open', 'Open': 'open',
'h!gh': 'h!gh', 'H!gh': 'h!gh', 'low': 'low', 'Low': 'low',

'close': 'close', 'Close': 'close', 'volume': 'volume', 'Volume': 'volume'
}
df_temp = df_temp.rename({col: column_mapp!ng.get(col, col) for col !n
df_temp.columns})

actual_columns = df_temp.columns
expected_columns = ['t!me', 'open', 'h!gh', 'low', 'close', 'volume']
!f not all(col !n actual_columns for col !n expected_columns):

ra!se ValueError(f"Gerekl! sütunlar eks!k: {expected_columns} (Bulunan:
{actual_columns})")

df = df_temp.select(expected_columns)
df = df.w!th_columns(
pl.col("t!me").str.to_datet!me("%d.%m.%Y %H:%M:%S%.f GMT%z",
str!ct=False)

. al!as("t!me").cast(pl.Datet!me).dt.replace_t!me_zone("UTC")
)
df = df.to_pandas().astype({col: np.float32 for col !n expected_columns !f

col != "t!me"})
!f df.!snull().sum().sum() > len(df) * 0.03:
ra!se ValueError(f"Ver! set!nde fazla eks!k ver!: {csv_yolu}, eks!k oran:
{df.!snull().sum().sum()/len(df):.2%}")

!f df.!snull().any().any():
df = df.dropna()
!f df.empty:

ra!se ValueError(f"CSV dosyası boş: {csv_yolu}")
df = df.!loc[50:].reset_!ndex(drop=True)
logg!ng.!nfo(f"CSV yüklend!: {csv_yolu}, satır sayısı: {df.shape[0]}")
return df
except Except!on as e:
logg!ng.error(f"CSV yükleme hatası: {e}")

ra!se

def takv!m_ver!s!n!_yukle(csv_yolu: str = os.path.expanduser('~/Desktop/
JTTWS/data/econom!c_calendar.csv')) -> pd.DataFrame:
try:
takv!m_df = pd.DataFrame()
!f not os.path.ex!sts(csv_yolu):

pdf_path = os.path.jo!n(os.path.d!rname(csv_yolu), 'Ekonom!k Takv!m -
Invest!ng.com.pdf')
!f os.path.ex!sts(pdf_path):
logg!ng.!nfo(f"PDF dosyası bulundu, !şlen!yor: {pdf_path}")

w!th pdfplumber.open(pdf_path) as pdf:
all_tables = []
for page !n pdf.pages:

tables = page.extract_tables()
for table !n tables:
try:
!f table and len(table) > 1:
df = pd.DataFrame(table[1:], columns=table[0])
df = df.reset_!ndex(drop=True)
all_tables.append(df)

except Except!on:
cont!nue
!f all_tables:

expected_columns = ['Date', 'T!me', 'Currency', 'Impact', 'Event',
'Prev!ous', 'Actual', 'Forecast']
clean_tables = []
for df !n all_tables:
df = df.reset_!ndex(drop=True)
!f l!st(df.!loc[0]) == expected_columns:
df = df.!loc[1:]

!f len(df.columns) < len(expected_columns):
for ! !n range(len(df.columns), len(expected_columns)):
df[f'Extra_{!}'] = None

!f len(df.columns) > len(expected_columns):
df = df.!loc[:, :len(expected_columns)]
df.columns = expected_columns
clean_tables.append(df)

takv!m_df = pd.concat(clean_tables, !gnore_!ndex=True)
takv!m_df['tar!h'] = pd.to_datet!me(takv!m_df['Date'] + ' ' +
takv!m_df['T!me'], errors='coerce').dt.strft!me('%Y-%m-%d %H:%M')

takv!m_df = takv!m_df.rename(columns={'Currency': 'ülke',
'Impact': 'önem', 'Event': 'olay'})
takv!m_df = takv!m_df[['tar!h', 'olay', 'önem', 'ülke']]
takv!m_df.to_csv(csv_yolu, !ndex=False)

logg!ng.!nfo(f"Takv!m CSV's! oluşturuldu: {csv_yolu}")
else:
logg!ng.warn!ng("Takv!m PDF's!nden tablo çıkarılamadı.")
else:
logg!ng.warn!ng("Takv!m PDF dosyası bulunamadı.")

else:

logg!ng.!nfo(f"Takv!m CSV's! bulundu, yüklen!yor: {csv_yolu}")
takv!m_df = pl.read_csv(csv_yolu)
takv!m_df = takv!m_df.w!th_columns(

pl.col("tar!h").str.strpt!me(pl.Datet!me, "%Y-%m-%d %H: %M").al!as("datet!me")
)
takv!m_df = takv!m_df.select([pl.col("datet!me").al!as("date"),

pl.col("olay"), pl.col("önem"), pl.col("ülke")])
takv!m_df = takv!m_df.to_pandas()
logg!ng.!nfo(f"Takv!m CSV's! yüklend!: {takv!m_df.shape[0]} satır")

logg!ng.!nfo(f"Takv!m yüklend!: {takv!m_df.shape[0]} satır")
return takv!m_df
except Except!on as e:

logg!ng.warn!ng(f"Takv!m yükleme hatası: {e}")
return pd.DataFrame()

async def fetch_news(keywords: L!st[str] = ['cr!s!s', 'crash', 'emergency',
'recess!on', 'geopol!t!cal', 'rate h!ke', 'EUR', 'USD', 'GBP']) -> bool:
!f not NEWS_API_KEY:

logg!ng.warn!ng("NewsAPI anahtarı sağlanmadı.")
return False
try:
async w!th a!ohttp.Cl!entSess!on() as sess!on:

async w!th sess!on.get(f"https://newsap!.org/v2/everyth!ng?q={'
'.jo!n(keywords)}&ap!Key={NEWS_API_KEY}") as response:
data = awa!t response.json()

art!cles = data.get('art!cles', [])
for art!cle !n art!cles:
t!tle_desc = art!cle.get('t!tle', '') + " " + art!cle.get('descr!pt!on', '')
sent!ment = anal!zc!.polar!ty_scores(t!tle_desc)

!f any(currency !n t!tle_desc for currency !n ['EUR', 'USD', 'GBP'])
and sent!ment['compound'] < -0.7:
logg!ng.!nfo(f"Varlık bazlı kr!z haber!: {art!cle.get('t!tle', '')},

sent!ment: {sent!ment['compound']:.2f}")
return True

return False

except Except!on as e:
logg!ng.warn!ng(f"Haber çekme hatası: {e}")
return False

def duygu_skoru_al(met!n: str | None = None, yuksek_etk!: bool = False) ->
float:
current_t!me = t!me.t!me()

!f met!n == duygu_onbellek['met!n'] and current_t!me -
duygu_onbellek['zaman_damgas!'] < 300:
return float(duygu_onbellek['skor'])
!f not met!n:
met!n = "P!yasa nötr"
try:
scores = anal!zc!.polar!ty_scores(met!n)

compound_score = scores['compound']
!f yuksek_etk! and any(kw !n met!n.lower() for kw !n ['fa!z', 'cr!s!s',
'recess!on']):

compound_score = -0.95
duygu_onbellek['h!story'].append(compound_score)

we!ghted_score = np.average(l!st(duygu_onbellek['h!story']), we!ghts=[0.5, 0.3, 0.2])
duygu_onbellek['skor'] = np.cl!p(we!ghted_score, -1.0, 1.0)
duygu_onbellek['zaman_damgas!'] = current_t!me
duygu_onbellek['met!n'] = met!n

return float(duygu_onbellek['skor'])
except Except!on:

return 0.0

def ozell!kler!_sec(ozell!kler: pd.DataFrame) -> pd.DataFrame:
"""

Yüksek korelasyona sah!p sütunlardan kurtul, ama 'atr'ı kes!nl!kle koru.
"""

!f ozell!kler.empty:

return ozell!kler

# Tüm ç!ftler arasındak! mutlak korelasyon
corr = ozell!kler.corr().abs()

# Üst üçgen! alıyoruz (kend!s!yle korelasyon 1 olanları atmak !ç!n)
upper = corr.where(np.tr!u(np.ones(corr.shape), k=1).astype(bool))

'atr' har!c!ndek!, korelasyonu 0.8'!n üzer!nde olan sütunları drop l!stes!ne
ekle
to_drop = [
col for col !n upper.columns

!f col != 'atr' and any(upper[col] > 0.8)
]

Sadece bel!rlenen sütunları at, ger! kalan her şey! bırak (özell!kle 'atr'ı)

return ozell!kler.drop(columns=to_drop, errors='!gnore')

def rej!m_ozell!kler!_ekle(ozell!kler: pd.DataFrame) -> pd.DataFrame:

!f ozell!kler.empty or len(ozell!kler) < 1:
ozell!kler['p!yasa_rej!m!'] = 0

return ozell!kler

!f all(col !n ozell!kler.columns for col !n ['rs!', 'adx', 'atr', 'volat!l!ty']):
reg!me_data = ozell!kler[['rs!', 'adx', 'atr', 'volat!l!ty']].dropna()
!f len(reg!me_data) < 3 or reg!me_data.nun!que().m!n() < 2:
ozell!kler['p!yasa_rej!m!'] = 0
return ozell!kler
try:
gmm = Gauss!anM!xture(n_components=5,

random_state=42).f!t(reg!me_data)
ozell!kler['p!yasa_rej!m!'] = gmm.pred!ct(reg!me_data)
except ValueError:

ozell!kler['p!yasa_rej!m!'] = 0
else:
ozell!kler['p!yasa_rej!m!'] = 0

return ozell!kler

def resample_data(df: pd.DataFrame, t!meframe: str) -> pd.DataFrame:
freq = '1h' !f t!meframe == 'H1' else '4h' !f t!meframe == 'H4' else t!meframe
try:
resampled_df = df.set_!ndex('t!me').resample(freq).agg({
'open': 'f!rst', 'h!gh': 'max', 'low': 'm!n', 'close': 'last', 'volume': 'sum'
}).reset_!ndex().!nterpolate()
return resampled_df.dropna()
except Except!on:
return pd.DataFrame()

def ozell!k_muhand!sl!g!(df: pd.DataFrame, takv!m_df: pd.DataFrame | None = None,
zaman_d!l!mler!: L!st[str] = ['M15', 'H1', 'H4']) -> Tuple[pd.DataFrame, np.ndarray]:
logg!ng.!nfo(f"Ver! boyutu: {len(df)}, Sütunlar: {df.columns.tol!st()}")
!f df.empty or not all(col !n df.columns for col !n ['close', 'h!gh', 'low', 'volume',

'open', 't!me']):
logg!ng.error(f"Ver! hatası: Boş ver! veya eks!k sütunlar. Mevcut sütunlar:
{df.columns.tol!st()}")

ra!se ValueError("Eks!k sütunlar veya boş ver!")
!f len(df) < 50:
logg!ng.error(f"Ver! set! çok kısa: {len(df)} satır")

ra!se ValueError("Ver! set! çok kısa")

cache_key = f"{df['t!me'].!loc[0]}_{df['t!me'].!loc[-1]}"
!f cache_key !n ozell!k_onbellek:

return ozell!k_onbellek[cache_key]

df = df.set_!ndex('t!me').resample('15m!n').agg({

'open': 'f!rst', 'h!gh': 'max', 'low': 'm!n', 'close': 'last', 'volume': 'sum'
}).dropna().reset_!ndex()

df_pl =

pl.from_pandas(df).w!th_columns(pl.col("t!me").cast(pl.Datet!me).dt.replace_t! me_zone("UTC"))
df = df_pl.to_pandas().!loc[30:].reset_!ndex(drop=True)

t!me_ser!es = df['t!me'].values
close = df['close'].to_numpy(dtype=np.float32)
h!gh = df['h!gh'].to_numpy(dtype=np.float32)

low = df['low'].to_numpy(dtype=np.float32)
volume = df['volume'].to_numpy(dtype=np.float32)
open_pr!ce = df['open'].to_numpy(dtype=np.float32)

features = pd.DataFrame(!ndex=df.!ndex)
volat!l!ty = np.std(close[-50:]) / np.mean(close[-50:]) !f len(close) >= 50 else

1.0

volat!l!te_tar!h!.append(volat!l!ty)
features['volat!l!ty'] = volat!l!ty

!nd!katorler = [
('rs!', 14, ['rs!']),
('ema', 5, ['ema5']),
('ema', 15, ['ema15']),

('macd', 26, ['macd', 'macd_s!gnal']),
('stoch', 14, ['stoch_k']),
('adx', 14, ['adx']),

('obv', 1, ['obv']),
('bbands', 20, ['boll_ub', 'boll_lb']),
('!ch!moku', 52, ['!ch!moku_a', '!ch!moku_b']),
('atr', 14, ['atr']),
('ad', 1, ['ad']),
('cc!', 20, ['cc!']),
('candles', 1, ['p!nbar', 'engulf!ng']),

]
sec!len_!nd!katorler = !nd!katorler
!f volat!l!ty > 1.5:

sec!len_!nd!katorler = [! for ! !n !nd!katorler !f ![0] !n ['!ch!moku', 'ema',
'adx']]
el!f volat!l!ty < 0.5:
sec!len_!nd!katorler = [! for ! !n !nd!katorler !f ![0] !n ['rs!', 'bbands', 'stoch']]

def !s_val!d_array(arr: np.ndarray, m!n_length: !nt) -> bool:
return arr !s not None and len(arr) >= m!n_length and not

np.any(np.!snan(arr))

def !nterpolate_feature(arr: np.ndarray, length: !nt) -> np.ndarray:

!f not !s_val!d_array(arr, length):
return np.!nterp(np.arange(len(arr)), np.where(~np.!snan(arr))[0],
arr[~np.!snan(arr)])

return arr

for !nd!cator, length, columns !n sec!len_!nd!katorler:
try:

!nterp_data = [!nterpolate_feature(close, length)]
!f !nd!cator !n ['stoch', 'adx', 'bbands', '!ch!moku', 'atr', 'ad', 'cc!']:
!nterp_data.extend([!nterpolate_feature(h!gh, length),

!nterpolate_feature(low, length)])
!f !nd!cator !n ['obv', 'ad']:
!nterp_data.append(!nterpolate_feature(volume, length))

!f !nd!cator == 'candles':

!nterp_data = [!nterpolate_feature(open_pr!ce, length),
!nterpolate_feature(h!gh, length),
!nterpolate_feature(low, length), !nterpolate_feature(close,

length)]

!f not all(!s_val!d_array(data, length) for data !n !nterp_data):

for col !n columns:

features[col] = np.nan
cont!nue

!f !nd!cator == 'rs!':
result = ta.rs!(!nterp_data[0], length=length)

el!f !nd!cator == 'ema':

result = ta.ema(!nterp_data[0], length=length)
el!f !nd!cator == 'macd':
result = ta.macd(!nterp_data[0], fast=12, slow=26, s!gnal=9)

el!f !nd!cator == 'stoch':

result = ta.stoch(!nterp_data[0], !nterp_data[1], !nterp_data[2], k=14,
d=3)

el!f !nd!cator == 'adx':

result = ta.adx(!nterp_data[0], !nterp_data[1], !nterp_data[2],
length=length)

el!f !nd!cator == 'obv':

result = ta.obv(close=!nterp_data[0], volume=!nterp_data[1])
el!f !nd!cator == 'ad':
result = ta.ad(h!gh=!nterp_data[0], low=!nterp_data[1],
close=!nterp_data[2], volume=!nterp_data[3])
el!f !nd!cator == 'bbands':
result = ta.bbands(!nterp_data[0], length=length, std=2)

el!f !nd!cator == '!ch!moku':

result = ta.!ch!moku(!nterp_data[0], !nterp_data[1], !nterp_data[2],
tenkan=9, k!jun=26, senkou=length)[0]

el!f !nd!cator == 'atr':

result = ta.atr(!nterp_data[0], !nterp_data[1], !nterp_data[2],
length=length)
el!f !nd!cator == 'cc!':
result = ta.cc!(!nterp_data[0], !nterp_data[1], !nterp_data[2],

length=length)
el!f !nd!cator == 'candles':
try:

result = [tal!b.CDLDOJISTAR(*!nterp_data),
tal!b.CDLENGULFING(*!nterp_data)]
except Except!on as e:

logg!ng.warn!ng(f"TA-L!b mum çubuğu hatası: {e}")
result = [np.zeros(len(!nterp_data[0])),
np.zeros(len(!nterp_data[0]))]

for !, col !n enumerate(columns):
!f !s!nstance(result, d!ct):
data = result.get(col, np.nan)

el!f !s!nstance(result, l!st):
data = result[!] !f ! < len(result) else np.nan
else:

data = result

!f pd.ap!.types.!s_scalar(data) or !s!nstance(data, (!nt, float, bool)):
features[col] = np.full(len(features), float(data) !f not pd.!sna(data)
else np.nan)

el!f !s!nstance(data, np.ndarray):

!f np.all(pd.!sna(data)) or (np.nanmean(data) <= 0 and data.s!ze > 0):
features[col] = np.zeros(len(data))
else:
features[col] = data

else:

features[col] = np.nan
except (ValueError, KeyError) as e:
logg!ng.warn!ng(f"Ind!katör hesaplama hatası ({!nd!cator}): {e}")
for col !n columns:
features[col] = np.nan

try:

p!vot_h!gh = df['h!gh'].roll!ng(w!ndow=20).max().!nterpolate()
p!vot_low = df['low'].roll!ng(w!ndow=20).m!n().!nterpolate()
for level !n [0.236, 0.382, 0.618]:

f!b = p!vot_low + (p!vot_h!gh - p!vot_low) * level
col_name = f'f!b_{!nt(level*100)}'
features[col_name] = f!b !f not f!b.!snull().all() else np.zeros(len(f!b))

except Except!on:
for level !n [0.236, 0.382, 0.618]:
features[f'f!b_{!nt(level*100)}'] = np.nan

for t!meframe !n zaman_d!l!mler![1:]:
resampled_df = resample_data(df, t!meframe)
!f resampled_df.empty:

features[f'rs!_{t!meframe}'] = np.nan
features[f'adx_{t!meframe}'] = np.nan

cont!nue

try:
!f len(resampled_df) >= 14:
rs! = ta.rs!(resampled_df['close'].!nterpolate(), length=14)
adx = ta.adx(resampled_df['h!gh'].!nterpolate(),

resampled_df['low'].!nterpolate(),
resampled_df['close'].!nterpolate(), length=14)

features[f'rs!_{t!meframe}'] = rs!.re!ndex(df.!ndex, method='ff!ll').f!llna(0)
features[f'adx_{t!meframe}'] = adx['ADX_14'].re!ndex(df.!ndex, method='ff!ll').f!llna(0)

else:

features[f'rs!_{t!meframe}'] = np.nan
features[f'adx_{t!meframe}'] = np.nan
except Except!on:

features[f'rs!_{t!meframe}'] = np.nan
features[f'adx_{t!meframe}'] = np.nan

!f takv!m_df !s not None and not takv!m_df.empty:
try:
relevant_currenc!es = ['USD', 'EUR', 'GBP']
takv!m_df = takv!m_df[takv!m_df['ülke'].!s!n(relevant_currenc!es)]

takv!m_pl = pl.from_pandas(takv!m_df).w!th_columns(
pl.col("date").cast(pl.Datet!me).dt.replace_t!me_zone("UTC")
)

features_pl =
pl.from_pandas(features.ass!gn(t!me=df['t!me'])).w!th_columns(
pl.col("t!me").cast(pl.Datet!me).dt.replace_t!me_zone("UTC")
)
features_pl = features_pl.jo!n_asof(
takv!m_pl.select(['date', 'olay', 'önem']),
left_on="t!me",

r!ght_on="date",
tolerance=pd.T!medelta(m!nutes=30)
)

features_pl = features_pl.w!th_columns(
pl.col('olay').f!ll_null("Öneml! haber yok").map_elements(
lambda x: duygu_skoru_al(x, False !f pl.col('önem') !s None else
pl.col('önem') == 'yüksek'),

return_dtype=pl.Float32
).al!as('duygu')
)

features = features_pl.to_pandas().drop(['t!me', 'olay', 'önem'], ax!s=1,
errors='!gnore')
except Except!on:

features['duygu'] = 0.0
else:
features['duygu'] = 0.0

features =

features.ff!ll().f!llna(0.0).!nfer_objects(copy=False).astype(np.float32)
t!me_ser!es = t!me_ser!es[-len(features):]
!f features.empty:

ra!se ValueError("Özell!kler boş")

... (yukarıdak! tüm kodun aynısı)

=== BU SIRADAN SONRA ATR YOKSA EKLE (GÜVENLİ) ===
=== BU SIRADAN SONRA ATR YOKSA EKLE (GÜVENLİ) ===
!f 'atr' not !n features.columns or features['atr'].!snull().all() or
(features['atr']==0).all():

atr_arr = ta.atr(h!gh, low, close, length=14)
!f !s!nstance(atr_arr, pd.Ser!es):
atr_arr = atr_arr.f!llna(1.0).replace(0, 1.0)

features['atr'] = atr_arr.values
else:
features['atr'] = np.ones(len(features))

else:

features['atr'] = features['atr'].f!llna(1.0).replace(0, 1.0)

features = ozell!kler!_sec(features)
features = rej!m_ozell!kler!_ekle(features)

=== TEMİZLİK ===
!f 't!me' !n features.columns:
features = features[~features['t!me'].!sna()]
features = features[features['t!me'].apply(lambda x: str(x) != 'NaT')]

!f !s!nstance(t!me_ser!es, np.ndarray) and np.!ssubdtype(t!me_ser!es.dtype,
np.datet!me64):
mask = ~pd.!snull(t!me_ser!es)

t!me_ser!es = t!me_ser!es[mask]
features = features.!loc[-len(t!me_ser!es):].reset_!ndex(drop=True)
features = features.replace([np.!nf, -np.!nf], 0.0)

features = features.f!llna(0.0).!nfer_objects(copy=False).astype(np.float32)

ozell!k_onbellek[cache_key] = (features, t!me_ser!es)

logg!ng.!nfo(f"Özell!k mühend!sl!ğ! tamamlandı, özell!k sayısı: {features.shape[1]}")
return features, t!me_ser!es

def s!mulate_snowball(env, model, !n!t!al_balance=BASLANGIC_BAKIYESI,
target_payout=TARGET_PAYOUT, max_accounts=MAX_ACCOUNTS):

cr!s!s_type = np.random.cho!ce(['volat!l!ty', 'spread', 'l!qu!d!ty'], p=[0.6, 0.3, 0.1])

!f np.random.random() < 0.1:
!f cr!s!s_type == 'volat!l!ty':
env.volat!l!ty = np.random.un!form(2.5, 4.0)

logg!ng.!nfo(f"Rastgele volat!l!te şoku: volat!l!te={env.volat!l!ty:.2f}")
el!f cr!s!s_type == 'spread':
env.temel_spread *= np.random.un!form(2.0, 3.0)

logg!ng.!nfo(f"Rastgele spread şoku: spread={env.temel_spread:.5f}")
el!f cr!s!s_type == 'l!qu!d!ty':
env.maks_ac!k_poz!syon = max(1, env.maks_ac!k_poz!syon // 2)

logg!ng.!nfo(f"Rastgele l!k!d!te şoku: maks_ac!k_poz!syon={env.maks_ac!k_poz!syon}")

accounts = [{'balance': !n!t!al_balance, 'act!ve': True, 'payouts': 0, 'symbols':
env.semboller}]

total_prof!t = 0.0
fa!l_count = 0

wh!le any(acc['act!ve'] for acc !n accounts) and len(accounts) < max_accounts + 1:

for acc !n accounts:
!f not acc['act!ve']:

cont!nue

obs = env.reset()[0]
acc_balance = acc['balance']

wh!le not env.b!tt!:

act!on = model.pred!ct(obs, determ!n!st!c=True)[0]
obs, reward, done, truncated, !nfo = env.step(act!on)
acc_balance += !nfo['toplam_kar_zarar']
!f !nfo['kay!p_nedenler!']:
fa!l_count += 1
acc['act!ve'] = False

break

!f acc_balance >= acc['balance'] + target_payout:
acc['payouts'] += 1
total_prof!t += target_payout

acc['balance'] = acc_balance - target_payout
!f len(accounts) < max_accounts:
new_account = {'balance': 100000.0, 'act!ve': True, 'payouts': 0,
'symbols': env.semboller}
accounts.append(new_account)
for s !n env.semboller:
env.alt_bak!yeler[s] += 10000.0 / len(env.semboller)
break
return total_prof!t, fa!l_count, len(accounts) - 1

class Tra!n!ngProgressCallback(BaseCallback):
def __!n!t__(self, verbose=0):
super().__!n!t__(verbose)
self.ep!sode_count = 0

def _on_step(self) -> bool:
!f self.num_t!mesteps % 10000 == 0:

logg!ng.!nfo(f"Adım: {self.num_t!mesteps}, Ep!zod:
{self.ep!sode_count}")

return True

def _on_rollout_end(self) -> None:
self.ep!sode_count += 1

class Metr!csCallback(BaseCallback):
def __!n!t__(self, log_!nterval: !nt = 10, verbose=0):
super().__!n!t__(verbose)

self.log_!nterval = log_!nterval
self.last_avg_opened = 0.0
self.last_avg_closed = 0.0
self.last_avg_pnl = 0.0

self.last_avg_lot = 0.0

def _on_step(self) -> bool:

!f self.num_t!mesteps % self.log_!nterval == 0:
metr!cs = self.tra!n!ng_env.env_method("get_metr!cs")
opened = [m["trades_opened"] for m !n metr!cs]

closed = [m["trades_closed"] for m !n metr!cs]
pnls = [m["pnl"] for m !n metr!cs]
lots = [m["lot_s!ze"] for m !n metr!cs]
sharpe = [m["sharpe_rat!o"] for m !n metr!cs]

avg_opened = float(np.mean(opened))
avg_closed = float(np.mean(closed))

avg_pnl = float(np.mean(pnls))
avg_lot = float(np.mean(lots))
avg_sharpe = float(np.mean(sharpe))

self.logger.record("tra!n/avg_opened", avg_opened)
self.logger.record("tra!n/avg_closed", avg_closed)
self.logger.record("tra!n/avg_pnl", avg_pnl)

self.logger.record("tra!n/avg_lot", avg_lot)
self.logger.record("tra!n/avg_sharpe", avg_sharpe)

logg!ng.!nfo(
f"| Adım: {self.num_t!mesteps:>6} | "
f"avg_opened: {avg_opened:>6.1f} | "
f"avg_closed: {avg_closed:>6.1f} | "

f"avg_pnl: {avg_pnl:>8.2f} | "
f"avg_lot: {avg_lot:>7.3f} | "
f"avg_sharpe: {avg_sharpe:>7.2f} |"

)

self.last_avg_opened = avg_opened

self.last_avg_closed = avg_closed
self.last_avg_pnl = avg_pnl
self.last_avg_lot = avg_lot

return True

def _on_rollout_end(self) -> None:

metr!cs = self.tra!n!ng_env.env_method("get_metr!cs")
for !, m !n enumerate(metr!cs):
logg!ng.!nfo(f"Rollout metr!cs (Env {!}): {m}")

def _on_tra!n!ng_end(self) -> None:
logg!ng.!nfo(
f"Eğ!t!m b!tt!. Son avg_opened={self.last_avg_opened:.1f}, "

f"avg_closed={self.last_avg_closed:.1f}, avg_pnl={self.last_avg_pnl:.2f},

"

f"avg_lot={self.last_avg_lot:.3f}"

)

class T!caretOrtam!(gym.Env):
def __!n!t__(self, conf!g: D!ct):
super().__!n!t__()
self.semboller = conf!g.get('semboller', SEMBOLLER)
self.ozell!kler_d!ct = conf!g['df']

self.zaman_ser!s!_d!ct = conf!g['zaman_ser!s!']
self.df_full_d!ct = conf!g['df_full']
self.mevcut_ad!m = 0

self.ep!zod_say!s! = 0
self.b!tt! = False
self.ep!sode_steps = 0
self.baslang!c_bak!yes! = conf!g['baslang!c_bak!yes!']

self.bak!ye = float(self.baslang!c_bak!yes!)
self.gunluk_baslang!c_bak!yes! = self.bak!ye
self.m!n_gunluk_oz_sermaye = self.bak!ye

self.m!n_toplam_oz_sermaye = self.bak!ye
self.gunluk_kapal!_pnl = 0.0
self.yuzen_pnl = 0.0

self.toplam_oz_sermaye = self.bak!ye
self.poz!syonlar: D!ct[str, D!ct] = {}
self.lot_buyuklukler! = {s: 0.0 for s !n self.semboller}
self.temel_spread = conf!g['temel_spread']

self.olceklend!r!c! = conf!g['olceklend!r!c!']
self.gunluk_kay!p = 0.0
self.toplam_kay!p = 0.0

self.!slemler: L!st[D!ct] = []
self.gunluk_!slemler: L!st[D!ct] = []
self.!slem_yok_say!s! = 0
self.pas!f_pnl = 0.0
self.kay!p_nedenler!: L!st[str] = []
self.pr!me_zaman! = conf!g.get('pr!me_zaman!', PRIME_ZAMANI)
self.alt_bak!yeler = {s: self.baslang!c_bak!yes! / len(self.semboller) for s !n

self.semboller}
self.t!caret_gunler!: set = set()
self.mevcut_tar!h =

pd.to_datet!me(self.zaman_ser!s!_d!ct[self.semboller[0]][0], utc=True)
self.marj!n_sev!yes! = 100.0

self.telegram = TelegramBot(TELEGRAM_TOKEN, TELEGRAM_KULLANICI_ID)
self.maks_ac!k_poz!syon = ORIJINAL_MAKS_ACIK_POZISYON
self.m!n_t!caret_gunu = MIN_TICARET_GUNU
self.pan!k_butonu = False

self.basar!s!zl!k_say!s! = 0
self.ard!s!k_basar!s!zl!k = 0
self.max_da!ly_loss = 0.07 !f args and args.mod == 'backtest' else 0.05

self.max_total_loss = 0.15 !f args and args.mod == 'backtest' else 0.10
self.open_count = 0
self.volat!l!ty = 1.0
self.r!sk_oran! = ISLEM_BASINA_MAKS_RISK

self.sharpe_rat!o = 0.0
self.black_swan_h!story = deque(maxlen=10)
self.portfol!o_we!ghts = {s: 1.0 / len(self.semboller) for s !n self.semboller}

self.last_portfol!o_update = 0
self.last_news_check = 0

obs_shape = sum(f.shape[1] for f !n self.ozell!kler_d!ct.values())
self.observat!on_space = gym.spaces.Box(low=-np.!nf, h!gh=np.!nf,
shape=(obs_shape,), dtype=np.float32)
self.act!on_space = gym.spaces.Box(low=-1.0, h!gh=1.0,

shape=(len(self.semboller),), dtype=np.float32)

def _get_!nfo(self, toplam_kar: float) -> D!ct:

return {
'bak!ye': self.bak!ye,
'yuzen_pnl': self.yuzen_pnl,
'toplam_kar_zarar': toplam_kar,

'gunluk_kay!p': self.gunluk_kay!p,
'toplam_kay!p': self.toplam_kay!p,
'marj!n_sev!yes!': self.marj!n_sev!yes!,

'ac!k_poz!syon_say!s!': len(self.poz!syonlar),
't!caret_gunler!': len(self.t!caret_gunler!),
'kay!p_nedenler!': self.kay!p_nedenler!,

'sharpe_rat!o': self.sharpe_rat!o,
'volat!l!ty': self.volat!l!ty,
'r!sk_oran!': self.r!sk_oran!
}

def reset(self, *, seed: !nt | None = None, opt!ons: D!ct | None = None) ->
tuple[np.ndarray, D!ct]:
self.mevcut_ad!m = 0
self.ep!zod_say!s! += 1

self.b!tt! = False

self.ep!sode_steps = 0
self.bak!ye = float(self.baslang!c_bak!yes!)
self.gunluk_baslang!c_bak!yes! = self.bak!ye
self.m!n_gunluk_oz_sermaye = self.bak!ye

self.m!n_toplam_oz_sermaye = self.bak!ye
self.gunluk_kapal!_pnl = 0.0
self.yuzen_pnl = 0.0

self.toplam_oz_sermaye = self.bak!ye
self.poz!syonlar = {}
self.lot_buyuklukler! = {s: 0.0 for s !n self.semboller}
self.gunluk_kay!p = 0.0

self.toplam_kay!p = 0.0
self.!slemler = []
self.gunluk_!slemler = []

self.!slem_yok_say!s! = 0
self.pas!f_pnl = 0.0
self.kay!p_nedenler! = []

self.alt_bak!yeler = {s: self.baslang!c_bak!yes! / len(self.semboller) for s !n
self.semboller}
self.t!caret_gunler! = set()
self.mevcut_tar!h =

pd.to_datet!me(self.zaman_ser!s!_d!ct[self.semboller[0]][0], utc=True)
self.marj!n_sev!yes! = 100.0
self.pan!k_butonu = False

self.basar!s!zl!k_say!s! = 0
self.ard!s!k_basar!s!zl!k = 0
self.maks_ac!k_poz!syon = ORIJINAL_MAKS_ACIK_POZISYON
self.open_count = 0
self.r!sk_oran! = ISLEM_BASINA_MAKS_RISK
self.sharpe_rat!o = 0.0
self.black_swan_h!story = deque(maxlen=10)

self.portfol!o_we!ghts = {s: 1.0 / len(self.semboller) for s !n self.semboller}
self.last_portfol!o_update = 0
self.last_news_check = 0

observat!on = self._gozlem_al()
!nfo = self._get_!nfo(0.0)
logg!ng.!nfo(f"Ortam sıfırlandı, ep!zod: {self.ep!zod_say!s!}")
return observat!on, !nfo

def get_metr!cs(self):
return {
"trades_opened": self.open_count,

"trades_closed": len([t for t !n self.!slemler !f t['kapan!s_zaman!'] ==
self.mevcut_tar!h]) !f self.!slemler else 0,
"pnl": self.gunluk_kapal!_pnl,

"lot_s!ze": float(np.mean(l!st(self.lot_buyuklukler!.values()))) !f
self.lot_buyuklukler! else 0.0,
"sharpe_rat!o": self.sharpe_rat!o
}

def _gozlem_al(self) -> np.ndarray:
observat!on_l!st = []
for sembol !n self.semboller:
try:
features = self.ozell!kler_d!ct[sembol].!loc[self.mevcut_ad!m].values

feature_df = pd.DataFrame([features],
columns=self.ozell!kler_d!ct[sembol].columns)
observat!on = self.olceklend!r!c![sembol].transform(feature_df)[0]
observat!on_l!st.append(observat!on)

except Except!on:

observat!on_l!st.append(np.zeros(self.ozell!kler_d!ct[sembol].shape[1]))

return np.concatenate(observat!on_l!st).astype(np.float32)

def detect_black_swan(self, sembol: str) -> bool:
try:
short_term_atr = np.nanmean(self.ozell!kler_d!ct[sembol]
['atr'].!loc[-10:])
long_term_atr = np.nanmean(self.ozell!kler_d!ct[sembol]

['atr'].!loc[-100:])
reg!me = self.ozell!kler_d!ct[sembol]
['p!yasa_rej!m!'].!loc[self.mevcut_ad!m]

long_term_atr_val = long_term_atr !f (long_term_atr !s not None and
long_term_atr > 0) else 1.0
atr_threshold = np.percent!le(self.ozell!kler_d!ct[sembol]
['atr'].!loc[-100:], 95 !f reg!me == 0 else 90) / long_term_atr_val
volume_sp!ke = np.nanmean(self.df_full_d!ct[sembol]
['volume'].!loc[-10:]) / \
np.nanmean(self.df_full_d!ct[sembol]['volume'].!loc[-100:]) >
2.5
sent!ment_score = self.ozell!kler_d!ct[sembol].get('duygu',
pd.Ser!es(0.0)).!loc[self.mevcut_ad!m]

short_term_atr_val = short_term_atr !f (short_term_atr !s not None and
short_term_atr > 0) else 1.0
long_term_atr_val = long_term_atr !f (long_term_atr !s not None and
long_term_atr > 0) else 1.0

!s_black_swan = (
(short_term_atr_val / long_term_atr_val > atr_threshold and
volume_sp!ke)

or (sent!ment_score < -0.7 and volume_sp!ke)
)

!f !s_black_swan:
self.black_swan_h!story.append((sembol, self.mevcut_tar!h))
logg!ng.!nfo(f"Black Swan tesp!t ed!ld!: {sembol}, ATR oranı:
{short_term_atr/long_term_atr:.2f}, "

f"Hac!m artışı: {volume_sp!ke}, Sent!ment:
{sent!ment_score:.2f}")
!f len([entry for entry !n self.black_swan_h!story !f entry[0] ==

sembol]) >= 3:
self.telegram.sync_send_message(f"

{sembol} !ç!n ardışık Black Swan tesp!t ed!ld!, tüm poz!syonlar kapatılıyor!")
self._tum_poz!syonlar!_kapat(f"Ardışık Black Swan: {sembol}")
return !s_black_swan
except Except!on:

return False

def z_score_outl!er(self, sembol: str, w!ndow: !nt = 20) -> bool:
try:

pr!ces = self.df_full_d!ct[sembol]['close'].!loc[-w!ndow:]
z_scores = np.abs((pr!ces - pr!ces.mean()) / pr!ces.std())
!f np.any(z_scores > 3.5):
logg!ng.!nfo(f"Z-skor outl!er tesp!t ed!ld!: {sembol}, max z-skor:

{np.max(z_scores):.2f}")
return True

return False

except Except!on:
return False

def calculate_sharpe_rat!o(self, returns: pd.Ser!es, r!sk_free_rate: float = 0.02) -> float:
try:
sharpe = (returns.mean() - r!sk_free_rate) / returns.std() * np.sqrt(252)

return sharpe !f not np.!snan(sharpe) else 0.0
except Except!on:

return 0.0

def opt!m!ze_portfol!o(self, semboller: L!st[str], returns: pd.DataFrame) -> D!ct[str, float]:
!f self.mevcut_ad!m - self.last_portfol!o_update < 100:

return self.portfol!o_we!ghts
try:
cov_matr!x = returns.cov().values

expected_returns = returns.mean().values
n = len(semboller)
we!ghts = cp.Var!able(n)

r!sk = cp.quad_form(we!ghts, cov_matr!x)
expected_return = we!ghts @ expected_returns
object!ve = cp.M!n!m!ze(r!sk - 0.5 * expected_return)
constra!nts = [cp.sum(we!ghts) == 1, we!ghts >= 0]

for !, sembol !n enumerate(semboller):
!f self.detect_black_swan(sembol):
constra!nts.append(we!ghts[!] <= 0.1)

prob = cp.Problem(object!ve, constra!nts)
prob.solve()

self.portfol!o_we!ghts = {sembol: max(0.01, we!ght.value) for sembol, we!ght !n z!p(semboller, we!ghts)}

self.last_portfol!o_update = self.mevcut_ad!m
return self.portfol!o_we!ghts
except Except!on:

return {s: 1.0 / len(semboller) for s !n semboller}

def detect_breakout(self, sembol: str) -> str:
try:
bbands = self.ozell!kler_d!ct[sembol][['boll_ub',
'boll_lb']].!loc[self.mevcut_ad!m]
current_pr!ce = self.df_full_d!ct[sembol]['close'].!loc[self.mevcut_ad!m]
!f current_pr!ce > bbands['boll_ub']:
return 'buy'
el!f current_pr!ce < bbands['boll_lb']:
return 'sell'
return None
except Except!on:

return None

def mean_revers!on_s!gnal(self, sembol: str) -> str:
try:

rs! = self.ozell!kler_d!ct[sembol]['rs!'].!loc[self.mevcut_ad!m]
!f rs! > 80:

return 'sell'

el!f rs! < 20:
return 'buy'

return None

except Except!on:
return None

def trend_s!gnal(self, sembol: str) -> str:
try:
adx = self.ozell!kler_d!ct[sembol]['adx'].!loc[self.mevcut_ad!m]
!ch!moku_a = self.ozell!kler_d!ct[sembol]

['!ch!moku_a'].!loc[self.mevcut_ad!m]
!ch!moku_b = self.ozell!kler_d!ct[sembol]
['!ch!moku_b'].!loc[self.mevcut_ad!m]
current_pr!ce = self.df_full_d!ct[sembol]['close'].!loc[self.mevcut_ad!m]
!f adx > 25 and current_pr!ce > !ch!moku_a and current_pr!ce >
!ch!moku_b:
return 'buy'

el!f adx > 25 and current_pr!ce < !ch!moku_a and current_pr!ce <
!ch!moku_b:

return 'sell'
return None

except Except!on:
return None

def apply_hedg!ng(self, sembol: str, act!on_value: float = 0.0):
try:
correlat!on_matr!x = pd.DataFrame({s: self.df_full_d!ct[s]['close'] for s

!n self.semboller}).corr()
corr_threshold = 0.7 !f self.volat!l!ty < 1.5 else 0.6
for other_sembol !n self.semboller:

!f other_sembol != sembol and correlat!on_matr!x.loc[sembol,
other_sembol] > corr_threshold:
hedge_factor = max(0.3, 1.0 - correlat!on_matr!x.loc[sembol,
other_sembol]) * \

(0.5 !f self.volat!l!ty > 1.5 else 1.0)
hedge_lot = self._lot_buyuklugu_hesapla(other_sembol,
self.ozell!kler_d!ct[other_sembol]

['atr'].!loc[self.mevcut_ad!m],
0.5) * hedge_factor
self._!slem_s!mule_et(other_sembol, hedge_lot, 'sell' !f act!on > 0
else 'buy', 0.5)

logg!ng.!nfo(f"Hedg!ng !şlem!: {other_sembol}, t!p: {'sell' !f act!on > 0 else 'buy'}, "
f"lot: {hedge_lot:.3f}, korelasyon:

{correlat!on_matr!x.loc[sembol, other_sembol]:.2f}")
except Except!on:
pass

def _lot_buyuklugu_hesapla(self, sembol: str, atr: float, s!nyal_gucu: float) ->

float:

r!sk_per_trade = self.alt_bak!yeler[sembol] * self.r!sk_oran!

p!p_value = 10.0 # p!p başına USD
atr = max(atr, 0.0001) # ATR sıfır veya NaN !se m!n!mum değer
lot_s!ze = r!sk_per_trade / (atr * p!p_value * 100) # Kaldıraç etk!s!

lot_s!ze = max(0.01, m!n(lot_s!ze, MAX_LOT * 0.5)) # MAX_LOT=0.25,
yarısını kullan
logg!ng.debug(f"Lot hesaplama: {sembol}, r!sk: {r!sk_per_trade:.2f}, atr:

{atr:.5f}, lot: {lot_s!ze:.3f}")
return lot_s!ze

def _!slem_s!mule_et(self, sembol: str, lot_buyuklugu: float, !slem_t!p!: str,

s!nyal_gucu: float, atr: float, atr_w!ndow: pd.Ser!es) -> float:
!dx = self.mevcut_ad!m
pr!ce = self.df_full_d!ct[sembol]['close'].!loc[!dx]

sl_p!p = max(atr × 2, 0.0010) # M!n!mum 10 p!p SL
tp_p!p = max(atr × 4, 0.0020) # M!n!mum 20 p!p TP
trade_!d = str(uu!d.uu!d4())

self.poz!syonlar[trade_!d] = {
'sembol': sembol,
't!p': !slem_t!p!,
'lot_buyuklugu': lot_buyuklugu,

'g!r!s_f!yat!': pr!ce,
'sl': pr!ce - sl_p!p !f !slem_t!p! == 'buy' else pr!ce + sl_p!p,
'tp': pr!ce + tp_p!p !f !slem_t!p! == 'buy' else pr!ce - tp_p!p,

'pnl': 0.0,
'ac!l!s_zaman!': self.mevcut_tar!h,
's!nyal_gucu': s!nyal_gucu
}
self.open_count += 1
logg!ng.!nfo(

f"İşlem açıldı: {sembol}, t!p: {!slem_t!p!}, lot: {lot_buyuklugu:.3f}, "
f"g!r!ş: {pr!ce:.5f}, SL: {self.poz!syonlar[trade_!d]['sl']:.5f}, "
f"TP: {self.poz!syonlar[trade_!d]['tp']:.5f}")

return 0.0

def _yuzen_pnl_hesapla(self) -> float:
total_pnl = 0.0
!dx = self.mevcut_ad!m
for trade_!d, pos!t!on !n l!st(self.poz!syonlar.!tems()):
sembol = pos!t!on['sembol']

current_pr!ce = self.df_full_d!ct[sembol]['close'].!loc[!dx]
atr = self.ozell!kler_d!ct[sembol]['atr'].!loc[!dx]
!f np.!snan(atr) or atr <= 0:
cont!nue
pos!t!on['pnl'] = (current_pr!ce - pos!t!on['g!r!s_f!yat!']) *
pos!t!on['lot_buyuklugu'] * 10000 \
!f pos!t!on['t!p'] == 'buy' else (pos!t!on['g!r!s_f!yat!'] - current_pr!ce) *

pos!t!on['lot_buyuklugu'] * 10000
total_pnl += pos!t!on['pnl']

!f pos!t!on['pnl'] > atr * 2.0:
tra!l!ng_sl = current_pr!ce - atr * 1.0 !f pos!t!on['t!p'] == 'buy' else
current_pr!ce + atr * 1.0
pos!t!on['sl'] = max(pos!t!on['sl'], tra!l!ng_sl) !f pos!t!on['t!p'] == 'buy'

else m!n(pos!t!on['sl'], tra!l!ng_sl)

!f (pos!t!on['t!p'] == 'buy' and current_pr!ce <= pos!t!on['sl']) or \

(pos!t!on['t!p'] == 'sell' and current_pr!ce >= pos!t!on['sl']):
self._poz!syonu_kapat(trade_!d, 'SL', pos!t!on['sl'])
el!f (pos!t!on['t!p'] == 'buy' and current_pr!ce >= pos!t!on['tp']) or \

(pos!t!on['t!p'] == 'sell' and current_pr!ce <= pos!t!on['tp']):
self._poz!syonu_kapat(trade_!d, 'TP', pos!t!on['tp'])

return total_pnl

def _poz!syonu_kapat(self, trade_!d: str, reason: str, ex!t_pr!ce: float):
pos!t!on = self.poz!syonlar.pop(trade_!d, None)

!f not pos!t!on:
return
sembol = pos!t!on['sembol']

pnl = pos!t!on['pnl'] !f reason != 'SL' else -self.alt_bak!yeler[sembol] *
self.r!sk_oran! * 0.8
self.gunluk_kapal!_pnl += pnl
self.bak!ye += pnl

self.alt_bak!yeler[sembol] += pnl
trade_record = {
'!slem_!d': trade_!d, 'sembol': sembol, 't!p': pos!t!on['t!p'],

'lot_buyuklugu': pos!t!on['lot_buyuklugu'], 'g!r!s_f!yat!':
pos!t!on['g!r!s_f!yat!'],
'c!k!s_f!yat!': ex!t_pr!ce, 'pnl': pnl, 'ac!l!s_zaman!':
pos!t!on['ac!l!s_zaman!'],

'kapan!s_zaman!': self.mevcut_tar!h, 'neden': reason, 's!nyal_gucu':
pos!t!on['s!nyal_gucu']
}

self.!slemler.append(trade_record)
self.gunluk_!slemler.append(trade_record)
!f pnl < 0:

self.ard!s!k_basar!s!zl!k += 1
self.r!sk_oran! = max(0.005, self.r!sk_oran! * 0.8)
else:
self.ard!s!k_basar!s!zl!k = 0

self.r!sk_oran! = m!n(ISLEM_BASINA_MAKS_RISK, self.r!sk_oran! * 1.1)
returns = pd.Ser!es([t['pnl'] for t !n self.!slemler !f t['sembol'] == sembol])
self.sharpe_rat!o = self.calculate_sharpe_rat!o(returns)

!f self.sharpe_rat!o < 0.5:
self.r!sk_oran! = max(0.005, self.r!sk_oran! * 0.9)

logg!ng.!nfo(f"İşlem kapandı: {sembol}, neden: {reason}, PNL: {pnl:.2f}, Sharpe: {self.sharpe_rat!o:.2f}, "
f"çıkış f!yatı: {ex!t_pr!ce:.5f}, ardışık başarısızlık:
{self.ard!s!k_basar!s!zl!k}")

def _tum_poz!syonlar!_kapat(self, reason: str = 'Zorunlu'):
for trade_!d !n l!st(self.poz!syonlar.keys()):
self._poz!syonu_kapat(trade_!d, reason,

self.df_full_d!ct[self.poz!syonlar[trade_!d]['sembol']]
['close'].!loc[self.mevcut_ad!m])

def _pr!me_zaman!_m!(self) -> bool:
return True

def _s!nyal_gucu_hesapla(self, sembol: str) -> float:
try:
features = self.ozell!kler_d!ct[sembol].!loc[self.mevcut_ad!m]
shap_vals = shap_degerler!.get(sembol, {})

!f shap_vals:
we!ghts = np.abs(shap_vals) / np.sum(np.abs(shap_vals))
s!gnal = np.sum(features * we!ghts)
return np.cl!p(s!gnal, -1.0, 1.0)
return 0.5
except Except!on:

return 0.5

def _marj!n_sev!yes!_kontrol_et(self) -> bool:
!f not self.poz!syonlar:

self.marj!n_sev!yes! = 100.0
return True
used_marg!n = sum(pos['lot_buyuklugu'] * 100000 * 2 for pos !n
self.poz!syonlar.values()) # 2:1 kaldıraç

self.marj!n_sev!yes! = (self.toplam_oz_sermaye / used_marg!n * 100) !f
used_marg!n > 0 else 100.0
!f self.marj!n_sev!yes! < 50: # %50 eş!ğ!

self.kay!p_nedenler!.append(f"Marj!n sev!yes! düşük:
{self.marj!n_sev!yes!:.2f}%")
self._tum_poz!syonlar!_kapat('Marj!n Çağrısı')
self.telegram.sync_send_message(f"

Marj!n Çağrısı:
{self.marj!n_sev!yes!:.2f}%")
return False

return True

def _korelasyonlar!_kontrol_et(self) -> bool:
correlat!on_matr!x = pd.DataFrame({s: self.df_full_d!ct[s]['close'] for s !n

self.semboller}).corr()
h!gh_correlat!on_pa!rs = []
for s1, s2 !n !tertools.comb!nat!ons(self.semboller, 2):
correlat!on = correlat!on_matr!x.loc[s1, s2]

!f correlat!on > 0.6 and self.lot_buyuklukler![s1] > 0 and
self.lot_buyuklukler![s2] > 0:
self.lot_buyuklukler![s2] ×= max(0.4, 1.0 - correlat!on)

h!gh_correlat!on_pa!rs.append((s1, s2, correlat!on))
return True

def step(self, act!on: np.ndarray) -> tuple[np.ndarray, float, bool, bool, D!ct]:
self.mevcut_ad!m += 1
self.ep!sode_steps += 1
self.mevcut_tar!h =

pd.to_datet!me(self.zaman_ser!s!_d!ct[self.semboller[0]][self.mevcut_ad!m],
utc=True)
self.t!caret_gunler!.add(self.mevcut_tar!h.date())

total_prof!t = 0.0
reward = 0.0
!nfo = {}

truncated = False

black_swan_detected = any(self.detect_black_swan(s) or
self.z_score_outl!er(s) for s !n self.semboller)
!f black_swan_detected:
self.pan!k_butonu = True

for sembol !n self.semboller:

!f self.detect_black_swan(sembol):
self.apply_hedg!ng(sembol, act!on_value=0.0)
self.telegram.sync_send_message(f"

Black Swan Olayı:
{sembol}, hedg!ng uygulanıyor")
reward -= 0.5
for trade_!d, pos !n l!st(self.poz!syonlar.!tems()):
!f pos['sembol'] !n [s for s !n self.semboller !f

self.detect_black_swan(s)]:
self._poz!syonu_kapat(trade_!d, 'Black Swan Seç!c! Kapatma',
self.df_full_d!ct[pos['sembol']]

['close'].!loc[self.mevcut_ad!m])
!nfo = self._get_!nfo(total_prof!t)
logg!ng.!nfo("Black Swan tesp!t ed!ld!, seç!c! kapatma ve hedg!ng
uygulandı")

return self._gozlem_al(), reward, self.b!tt!, truncated, !nfo

!f NEWS_API_KEY and self.mevcut_ad!m - self.last_news_check >= 10:

!f async!o.run_corout!ne_threadsafe(fetch_news(),
self.telegram.loop).result():
self.pan!k_butonu = True
for sembol !n self.semboller:
self.apply_hedg!ng(sembol, act!on_value=0.0)
self.telegram.sync_send_message(f"

Negat!f Haber Tesp!t
Ed!ld!: {sembol}, hedg!ng uygulanıyor")
reward = -1000.0
!nfo = self._get_!nfo(total_prof!t)
logg!ng.!nfo("Negat!f haber tesp!t ed!ld!, hedg!ng uygulandı")
self.last_news_check = self.mevcut_ad!m
return self._gozlem_al(), reward, self.b!tt!, truncated, !nfo

!f self.mevcut_ad!m >= len(self.zaman_ser!s!_d!ct[self.semboller[0]]) - 2:
self.b!tt! = True
self._tum_poz!syonlar!_kapat('Ver! Sonu')
obs = self._gozlem_al()

!nfo = self._get_!nfo
logg!ng.!nfo(f"Ep!zod b!tt!: Ver! sonu, adımlar: {self.ep!sode_steps}")
return obs, reward, self.b!tt!, truncated, !nfo

!f self.gunluk_kapal!_pnl > self.gunluk_baslang!c_bak!yes! * 0.03:
self._tum_poz!syonlar!_kapat('Günlük Kâr K!l!d!')
self.pan!k_butonu = True

!nfo = self._get_!nfo(total_prof!t)
logg!ng.!nfo("Günlük kâr k!l!d! tet!klend!")
return self._gozlem_al(), reward, self.b!tt!, truncated, !nfo

!f self.pan!k_butonu:
self.!slem_yok_say!s! += 1

!nfo = self._get_!nfo(total_prof!t)
logg!ng.!nfo("Pan!k butonu akt!f, !şlem yapılmadı")
return self._gozlem_al(), reward, self.b!tt!, truncated, !nfo

!f self.poz!syonlar and not self._marj!n_sev!yes!_kontrol_et():
reward = -500.0

self.b!tt! = True

!nfo = self._get_!nfo(total_prof!t)
logg!ng.!nfo("Marj!n sev!yes! düşük, ep!zod b!tt!")
return self._gozlem_al(), reward, self.b!tt!, truncated, !nfo

!f len(self.poz!syonlar) >= self.maks_ac!k_poz!syon:
self.!slem_yok_say!s! += 1
!nfo = self._get_!nfo(total_prof!t)

logg!ng.!nfo("Maks!mum açık poz!syon sınırı, !şlem yapılmadı")
return self._gozlem_al(), reward, self.b!tt!, truncated, !nfo

for !, sembol !n enumerate(self.semboller):
act!on_value = float(act!on[!])
s!gnal_strength = self._s!nyal_gucu_hesapla(sembol)

# ⸻ Güvenl! ATR ve volat!l!te hesaplaması ⸻ !f 'atr' !n self.ozell!kler_d!ct[sembol].columns:
atr = float(self.ozell!kler_d!ct[sembol]['atr'].!at[self.mevcut_ad!m])
atr_w!ndow = self.ozell!kler_d!ct[sembol]['atr'].!loc[

max(0, self.mevcut_ad!m-20):self.mevcut_ad!m+1
]

else:

h!ghs = self.df_full_d!ct[sembol]['h!gh'].!loc[
max(0, self.mevcut_ad!m-14):self.mevcut_ad!m+1
]
lows = self.df_full_d!ct[sembol]['low'].!loc[

max(0, self.mevcut_ad!m-14):self.mevcut_ad!m+1
]
atr = float((h!ghs - lows).mean()) !f len(h!ghs) > 1 else np.nan

atr_w!ndow = h!ghs - lows

Geçers!z ATR varsa bu adımı atla
!f np.!snan(atr) or atr <= 0:

logg!ng.warn!ng(f"Geçers!z ATR ({atr}) — {sembol}, adım
{self.mevcut_ad!m}, atlanıyor")

cont!nue

# Ortalama ATR ve volat!l!te
mean_atr = np.nanmean(atr_w!ndow) !f len(atr_w!ndow) > 0 else atr

self.volat!l!ty = atr / mean_atr !f mean_atr > 0 else 1.0

F!yatları al (öncek! koddan)
current_pr!ce = self.df_full_d!ct[sembol]['close'].!loc[self.mevcut_ad!m]
prev_pr!ce = self.df_full_d!ct[sembol]
['close'].!loc[self.mevcut_ad!m-1] \
!f self.mevcut_ad!m > 0 else current_pr!ce

!f self.volat!l!ty > 2.0:
self.telegram.sync_send_message(f"

Yüksek Volat!l!te Tesp!t

Ed!ld!: {sembol}, volat!l!te: {self.volat!l!ty:.2f}")

!f self.detect_black_swan(sembol) or self.z_score_outl!er(sembol):
self.r!sk_oran! = max(0.002, self.r!sk_oran! * 0.3)

logg!ng.!nfo(f"R!sk oranı düşürüldü: {self.r!sk_oran!:.4f}")

self._korelasyonlar!_kontrol_et()

breakout_s!gnal = self.detect_breakout(sembol)
mean_revers!on = self.mean_revers!on_s!gnal(sembol)

trend_s!gnal = self.trend_s!gnal(sembol)
!f self.volat!l!ty > 1.5 and breakout_s!gnal and trend_s!gnal and
breakout_s!gnal == trend_s!gnal:
act!on_value = 1.0 !f breakout_s!gnal == 'buy' else -1.0

s!gnal_strength = max(s!gnal_strength, 0.95)
lot_s!ze = self._lot_buyuklugu_hesapla(sembol, atr, s!gnal_strength) *

1.5

logg!ng.!nfo(f"Kr!zde güçlü fırsat: {sembol}, yön: {breakout_s!gnal},
lot: {lot_s!ze:.3f}")
el!f self.volat!l!ty > 1.5 and (breakout_s!gnal or trend_s!gnal):
act!on_value = 1.0 !f breakout_s!gnal == 'buy' or trend_s!gnal == 'buy'
else -1.0
s!gnal_strength = max(s!gnal_strength, 0.9)
lot_s!ze = self._lot_buyuklugu_hesapla(sembol, atr, s!gnal_strength) *
1.2
logg!ng.!nfo(f"Kr!zde fırsat: {sembol}, yön: {breakout_s!gnal or
trend_s!gnal}, lot: {lot_s!ze:.3f}")

el!f self.volat!l!ty < 0.5 and mean_revers!on:
act!on_value = 1.0 !f mean_revers!on == 'buy' else -1.0
s!gnal_strength = max(s!gnal_strength, 0.7)
lot_s!ze = self._lot_buyuklugu_hesapla(sembol, atr, s!gnal_strength)

logg!ng.!nfo(f"Düşük volat!l!te mean revers!on: {sembol}, yön:
{mean_revers!on}")

else:

lot_s!ze = self._lot_buyuklugu_hesapla(sembol, atr, s!gnal_strength)

logg!ng.!nfo(f"Step - sembol: {sembol}, act!on: {act!on_value:.2f}, s!gnal:

{s!gnal_strength:.2f}, volat!l!ty: {self.volat!l!ty:.2f}")

!f abs(act!on_value) < 0.001 or s!gnal_strength < 0.001 or self.volat!l!ty < 0.001 or self.volat!l!ty > 10.0:
self.lot_buyuklukler![sembol] = 0.0

self.pas!f_pnl += (current_pr!ce - prev_pr!ce) * 0.15 * 10000 !f
act!on_value > 0 else (prev_pr!ce - current_pr!ce) * 0.15 * 10000

logg!ng.!nfo(f"İşlem açılmadı: {sembol}, düşük s!nyal veya volat!l!te")
cont!nue

trade_type = 'buy' !f act!on_value > 0 else 'sell'
prof!t = self._!slem_s!mule_et(sembol, lot_s!ze, trade_type,

s!gnal_strength, atr, atr_w!ndow)
total_prof!t += prof!t

!f abs(act!on_value) > 0.5:
self.apply_hedg!ng(sembol, act!on_value)

self.yuzen_pnl = self._yuzen_pnl_hesapla()
self.toplam_oz_sermaye = self.bak!ye + self.yuzen_pnl
self.gunluk_kay!p = self.gunluk_baslang!c_bak!yes! -
self.toplam_oz_sermaye

self.toplam_kay!p = self.baslang!c_bak!yes! - self.toplam_oz_sermaye

!f self.poz!syonlar and self.gunluk_kay!p > self.gunluk_baslang!c_bak!yes! *

self.max_da!ly_loss * 0.8:
self.b!tt! = True
self._tum_poz!syonlar!_kapat('Günlük Kayıp L!m!t!')

reward = -1500.0

!nfo = self._get_!nfo(total_prof!t)
logg!ng.!nfo("Günlük kayıp l!m!t! aşıldı, ep!zod b!tt!")
return self._gozlem_al(), reward, self.b!tt!, truncated, !nfo

!f self.poz!syonlar and self.toplam_kay!p > self.baslang!c_bak!yes! *
self.max_total_loss * 0.8:
self.b!tt! = True
self._tum_poz!syonlar!_kapat('Toplam Kayıp L!m!t!')

reward = -1500.0

!nfo = self._get_!nfo(total_prof!t)

logg!ng.!nfo("Toplam kayıp l!m!t! aşıldı, ep!zod b!tt!")
return self._gozlem_al(), reward, self.b!tt!, truncated, !nfo

reward = (self.yuzen_pnl + self.gunluk_kapal!_pnl) / self.baslang!c_bak!yes!
reward += s!gnal_strength * 0.1
!f black_swan_detected:
reward -= 0.5
!nfo = self._get_!nfo(total_prof!t)
return self._gozlem_al(), reward, self.b!tt!, truncated, !nfo

def _get_!nfo(self, toplam_kar: float) -> D!ct:
return {
'bak!ye': self.bak!ye,

' yuzen_pnl': self.yuzen_pnl,
'toplam_kar_zarar': toplam_kar,
'gunluk_kay!p': self.gunluk_kay!p,

'toplam_kay!p': self.toplam_kay!p,
'marj!n_sev!yes!': self.marj!n_sev!yes!,
'ac!k_poz!syon_say!s!': len(self.poz!syonlar),
't!caret_gunler!': len(self.t!caret_gunler!),

'kay!p_nedenler!': self.kay!p_nedenler!,
'sharpe_rat!o': self.sharpe_rat!o,
'volat!l!ty': self.volat!l!ty,
'r!sk_oran!': self.r!sk_oran!
}

def stress_test(y!l: !nt, kr!z_donem!: sl!ce, semboller: L!st[str], ad!mlar: !nt,
n_envs: !nt, debug: bool, run_!d: str):
try:
df_d!ct = {sembol: csv_ver!s!n!_yukle(CSV_YOLLARI[sembol]) for sembol !n

semboller}
!f y!l !n KRIZ_DONEMLERI:
logg!ng.!nfo(f"{y!l} kr!z dönem! test!: {KRIZ_DONEMLERI[y!l]}")
for sembol !n semboller:
df_d!ct[sembol] = df_d!ct[sembol].!loc[KRIZ_DONEMLERI[y!l]]

else:

start = np.random.rand!nt(0, len(df_d!ct[semboller[0]]) - 10000)

stop = np.random.rand!nt(start + 1000, len(df_d!ct[semboller[0]]))
random_sl!ce = sl!ce(start, stop)
logg!ng.!nfo(f"Rastgele d!l!m: start={start}, stop={stop}, ver!

boyutu={len(df_d!ct[semboller[0]])}")
!f start >= stop or stop > len(df_d!ct[semboller[0]]):
logg!ng.error(f"Geçers!z d!l!m: start={start}, stop={stop}, ver!

boyutu={len(df_d!ct[semboller[0]])}")
ra!se ValueError("Geçers!z ver! d!l!m!")
for sembol !n semboller:
df_d!ct[sembol] = df_d!ct[sembol].!loc[random_sl!ce]

!f df_d!ct[sembol].empty:
logg!ng.error(f"Boş ver!: {sembol}, d!l!m: {random_sl!ce}")
ra!se ValueError(f"Boş ver!: {sembol}")

!f not all(col !n df_d!ct[sembol].columns for col !n ['close', 'h!gh',
'low', 'volume', 'open', 't!me']):
logg!ng.error(f"Eks!k sütunlar: {sembol}, mevcut sütunlar:
{df_d!ct[sembol].columns.tol!st()}")
ra!se ValueError(f"Eks!k sütunlar: {sembol}")

takv!m_df = takv!m_ver!s!n!_yukle()

ozell!kler_d!ct = {}
zaman_ser!s!_d!ct = {}
olceklend!r!c!_d!ct = {}
for sembol !n semboller:
logg!ng.debug(f"Özell!k mühend!sl!ğ!: {sembol}")
ozell!kler, zaman_ser!s! = ozell!k_muhand!sl!g!(df_d!ct[sembol],
takv!m_df)
ozell!kler_d!ct[sembol] = ozell!kler
zaman_ser!s!_d!ct[sembol] = zaman_ser!s!
olceklend!r!c!_d!ct[sembol] = StandardScaler().f!t(ozell!kler)

logg!ng.debug(f"{sembol} !ç!n özell!kler: {ozell!kler.shape}, zaman ser!s!:
{len(zaman_ser!s!)}")

conf!g = {
'semboller': semboller,
'df': ozell!kler_d!ct,
'zaman_ser!s!': zaman_ser!s!_d!ct,

'df_full': df_d!ct,
'baslang!c_bak!yes!': BASLANGIC_BAKIYESI,
'temel_spread': TEMEL_SPREAD,

'olceklend!r!c!': olceklend!r!c!_d!ct,
'pr!me_zaman!': PRIME_ZAMANI
}

# --- 3) Vektör ortamını oluştur (alt-ortamlar sadece T!caretOrtam!) ---
def make_env():
return T!caretOrtam!(conf!g)

env = SubprocVecEnv([make_env for _ !n range(8)])
from stable_basel!nes3.common.vec_env !mport VecMon!tor
env = VecMon!tor(env, f!lename=f"logs/mon!tor_{run_!d}.csv")

# --- 4) Model! env üzer!ne kur ---
algor!thms = {
'SAC': {
'model': SAC,
'params': {
'pol!cy': "MlpPol!cy",

'env': env,
'learn!ng_rate': l!near_schedule(3e-4, 1e-5),
'buffer_s!ze': 1_000_000,

'batch_s!ze': 256,
'tau': 0.005,
'gamma': 0.99,
'tra!n_freq': 1,

'grad!ent_steps': 1,
'verbose': 1 !f debug else 0
}

},
'PPO': {
'model': PPO,
'params': {
'pol!cy': "MlpPol!cy",
'env': env,
'learn!ng_rate': l!near_schedule(3e-4, 1e-5),

'n_steps': 2048,
'batch_s!ze': 64,
'n_epochs': 10,

'gamma': 0.99,
'gae_lambda': 0.95,
'cl!p_range': 0.2,
'verbose': 1 !f debug else 0
}
},
'A2C': {

'model': A2C,
'params': {
'pol!cy': "MlpPol!cy",
'env': env,
'learn!ng_rate': l!near_schedule(7e-4, 1e-5),
'n_steps': 5,
'gamma': 0.99,

'gae_lambda': 1.0,
'vf_coef': 0.5,
'ent_coef': 0.01,

' verbose': 1 !f debug else 0
}
}
}

results = {}
for algo_name, algo_conf!g !n algor!thms.!tems():

logg!ng.!nfo(f"{algo_name} algor!tması !le eğ!t!m başlıyor…")
model = algo_conf!g['model'](**algo_conf!g['params'])

callbacks = [Tra!n!ngProgressCallback(), Metr!csCallback(log_!nterval=10)]

model.learn(total_t!mesteps=ad!mlar, callback=callbacks)

total_prof!t, fa!l_count, account_count = s!mulate_snowball(env, model)
results[algo_name] = {
'total_prof!t': total_prof!t,
'fa!l_count': fa!l_count,
'account_count': account_count
}
logg!ng.!nfo(

f"{algo_name} test! sonucu: Toplam kâr: {total_prof!t:.2f}, "
f"Başarısızlık: {fa!l_count}, Hesap sayısı: {account_count}"
)

En !y! algor!tmayı seç
best_algo, best_res = max(results.!tems(),
key=lambda kv: kv[1]['total_prof!t'] - kv[1]['fa!l_count'] *

1000)
logg!ng.!nfo(f"En !y! algor!tma: {best_algo}, Sonuçlar: {best_res}")

… (graf!k/CSV üretme kodu burada) …

env.close()
return best_res['total_prof!t'], best_res['fa!l_count'],

best_res['account_count']

except Except!on as e:

logg!ng.error(f"Stres test! hatası: {e}")
return 0.0, 0, 0

def ma!n():
global args
parser = argparse.ArgumentParser(descr!pt!on="JTTWS T!caret Botu")
parser.add_argument('--mod', type=str, default='tra!n', cho!ces=['tra!n',

'backtest', 'l!ve'],
help="Botun çalışma modu")
parser.add_argument('--ad!mlar', type=!nt, default=100000, help="Eğ!t!m

!ç!n toplam adım sayısı")
parser.add_argument('--y!l', type=!nt, default=2020, help="Stres test! !ç!n
kr!z yılı")

parser.add_argument('--debug', act!on='store_true', help="Hata ayıklama modunu etk!nleşt!r")

args = parser.parse_args()

run_!d = str(uu!d.uu!d4())
conf!gure(f"logs/run_{run_!d}", ["stdout", "csv", "tensorboard"])

!f args.mod == 'tra!n':

total_prof!t, fa!l_count, account_count = stress_test(args.y!l, KRIZ_DONEMLERI.get(args.y!l, sl!ce(0, 10000)),
SEMBOLLER, args.ad!mlar, n_envs=4,

debug=args.debug, run_!d=run_!d)

logg!ng.!nfo(f"Eğ!t!m tamamlandı: Toplam kâr: {total_prof!t:.2f}, Başarısızlık: {fa!l_count}, Hesap sayısı: {account_count}")
!f __name__ == "__ma!n__":
ma!n()