# ✅ STEP 0: Install dependencies

!pip install yfinance ta tqdm requests chardet --quiet

# ✅ STEP 1: Mount Google Drive

from google.colab import drive

drive.mount('/content/drive')

# ✅ STEP 2: Imports

import yfinance as yf

import pandas as pd

import numpy as np

from datetime import datetime, timedelta

from ta.momentum import RSIIndicator

from ta.trend import MACD, EMAIndicator, SMAIndicator, ADXIndicator

from ta.volatility import AverageTrueRange, BollingerBands

import requests

from io import StringIO

from tqdm import tqdm

import os, csv, re, time

from urllib.parse import quote

import chardet

pd.set\_option("display.max\_columns", None)

# ✅ STEP 3: Index fetcher config

index\_folder = '/content/drive/My Drive/mathew\_index'

os.makedirs(index\_folder, exist\_ok=True)

all\_indices = [

"NIFTY 50", "NIFTY NEXT 50", "NIFTY 100", "NIFTY 200", "NIFTY 500",

"NIFTY BANK", "NIFTY MIDCAP 50", "NIFTY MIDCAP 100", "NIFTY SMALLCAP 100",

"NIFTY MIDCAP 150", "NIFTY SMALLCAP 250", "NIFTY MICROCAP 250", "NIFTY TOTAL MARKET",

"NIFTY LARGEMIDCAP 250", "NIFTY500 LARGEMIDSMALL EQUAL-CAP WEIGHTED",

"NIFTY MIDSMALLCAP 400", "NIFTY MIDSMALL INDIA CONSUMPTION",

"NIFTY SMALLCAP 50", "NIFTY MIDSMLCAP", "NIFTY IT", "NIFTY FMCG", "NIFTY PHARMA",

"NIFTY AUTO", "NIFTY ENERGY", "NIFTY MEDIA", "NIFTY REALTY", "NIFTY METAL",

"NIFTY PSU BANK", "NIFTY PRIVATE BANK", "NIFTY FINANCIAL SERVICES",

"NIFTY CONSUMER DURABLES", "NIFTY OIL & GAS", "NIFTY HEALTHCARE INDEX",

"NIFTY CAPITAL MARKETS", "NIFTY TRANSPORTATION & LOGISTICS",

"NIFTY CPSE", "NIFTY PSE", "NIFTY SERVICES SECTOR",

"NIFTY INDIA CONSUMPTION", "NIFTY INFRASTRUCTURE", "NIFTY MNC",

"NIFTY GROWTH SECTORS 15", "NIFTY INDIA MANUFACTURING",

"NIFTY INDIA DEFENCE", "NIFTY INDIA TOURISM", "NIFTY INDIA DIGITAL",

"NIFTY EV & NEW AGE AUTOMOTIVE", "NIFTY MOBILITY", "NIFTY HOUSING",

"NIFTY CORE HOUSING", "NIFTY IPO", "NIFTY SHARIAH 25", "NIFTY50 SHARIAH",

"NIFTY500 SHARIAH", "NIFTY100 ESG", "NIFTY100 ENHANCED ESG",

"NIFTY100 ALPHA 30", "NIFTY200 VALUE 30", "NIFTY200 ALPHA 30",

"NIFTY200 MOMENTUM 30", "NIFTY200 QUALITY 30", "NIFTY100 QUALITY 30",

"NIFTY100 LOW VOLATILITY 30", "NIFTY100 EQUAL WEIGHT", "NIFTY50 VALUE 20",

"NIFTY50 EQUAL WEIGHT", "NIFTY50 LOW VOLATILITY 50",

"NIFTY500 VALUE 50", "NIFTY500 QUALITY 50", "NIFTY500 LOW VOLATILITY 50",

"NIFTY500 EQUAL WEIGHT", "NIFTY500 MOMENTUM 50",

"NIFTY500 MULTICAP MOMENTUM QUALITY 50", "NIFTY500 MULTICAP INFRASTRUCTURE 50:30:20",

"NIFTY500 MULTICAP INDIA MANUFACTURING 50:30:20",

"NIFTY500 MULTICAP 50:25:25", "NIFTY TOP 10 EQUAL WEIGHT",

"NIFTY TOP 15 EQUAL WEIGHT", "NIFTY TOP 20 EQUAL WEIGHT",

"NIFTY ALPHA 50", "NIFTY ALPHA LOW-VOLATILITY 30",

"NIFTY ALPHA QUALITY LOW-VOLATILITY 30",

"NIFTY ALPHA QUALITY VALUE LOW-VOLATILITY 30",

"NIFTY MIDCAP150 QUALITY 50", "NIFTY MIDCAP150 MOMENTUM 50",

"NIFTY SMALLCAP250 QUALITY 50", "NIFTY SMALLCAP250 MOMENTUM QUALITY 100",

"NIFTY MIDSMALLCAP400 MOMENTUM QUALITY 100",

"NIFTY QUALITY LOW-VOLATILITY 30", "NIFTY DIVIDEND OPPORTUNITIES 50",

"NIFTY FINANCIAL SERVICES 25/50", "NIFTY MIDCAP SELECT",

"NIFTY LIQUID 15", "NIFTY MIDCAP LIQUID 15", "NIFTY SHARIAH 25",

"NIFTY FINANCIAL SERVICES EX-BANK", "NIFTY MIDSMALL HEALTHCARE",

"NIFTY MIDSMALL IT & TELECOM", "NIFTY MIDSMALL FINANCIAL SERVICES",

"NIFTY RURAL", "NIFTY NON-CYCLICAL CONSUMER"

]

def get\_output\_filename(index\_name):

formatted\_date = datetime.now().strftime("%d-%b-%Y")

clean\_name = index\_name.lower().replace("&", "and")

clean\_name = re.sub(r"[^a-z0-9]+", "-", clean\_name).strip("-")

return f"MW-{clean\_name.upper()}-{formatted\_date}.csv"

def fetch\_live\_constituents(index\_name):

headers = {

'accept': '\*/\*',

'accept-language': 'en-US,en;q=0.9',

'connection': 'keep-alive',

# --- PASTE YOUR NSE COOKIE BELOW ---

'cookie': 'PUT\_YOUR\_NSE\_COOKIE\_HERE',

'host': 'www.nseindia.com',

'referer': 'https://www.nseindia.com/market-data/live-equity-market',

'sec-ch-ua': '"Not)A;Brand";v="8", "Chromium";v="138", "Microsoft Edge";v="138"',

'sec-ch-ua-mobile': '?0',

'sec-ch-ua-platform': '"Windows"',

'sec-fetch-dest': 'empty',

'sec-fetch-mode': 'cors',

'sec-fetch-site': 'same-origin',

'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36 Edg/138.0.0.0',

'x-requested-with': 'XMLHttpRequest',

}

api\_url = f"https://www.nseindia.com/api/equity-stockIndices?csv=true&index={quote(index\_name)}&selectValFormat=crores"

print(f"📥 Fetching {index\_name} ...")

try:

resp = requests.get(api\_url, headers=headers, timeout=20)

resp.raise\_for\_status()

df = pd.read\_csv(StringIO(resp.text), skiprows=10)

df = df.map(lambda x: x.strip() if isinstance(x, str) else x)

if "SYMBOL" not in df.columns or df.shape[0] < 3:

print(f"❌ Invalid data received for {index\_name}. The response was not a valid CSV.")

return None

return df

except Exception as e:

print(f"❌ Error fetching {index\_name}: {e}")

return None

for index in all\_indices:

df = fetch\_live\_constituents(index)

if df is not None and not df.empty:

filename = get\_output\_filename(index)

filepath = os.path.join(index\_folder, filename)

df.to\_csv(filepath, index=False)

print(f"✅ Saved: {filepath}")

else:

print(f"❌ Failed to process: {index}")

time.sleep(2)

def get\_nse\_symbols(max\_stocks=100):

url = "https://nsearchives.nseindia.com/content/equities/EQUITY\_L.csv"

headers = {"User-Agent": "Mozilla/5.0", "Referer": "https://nsearchives.nseindia.com"}

try:

resp = requests.get(url, headers=headers, timeout=20)

resp.raise\_for\_status()

df = pd.read\_csv(StringIO(resp.text))

df.columns = df.columns.str.strip()

syms = df["SYMBOL"].dropna().astype(str).str.strip().unique()

return [s + ".NS" for s in syms][:max\_stocks]

except Exception as e:

print(f"❌ Failed to fetch NSE symbols: {e}")

return []

def fetch\_bse\_corporate\_actions(from\_date="20150101"):

to\_date = datetime.now().strftime("%Y%m%d")

base\_url = "https://api.bseindia.com/BseIndiaAPI/api/CorpactCSVDownload/w"

params = {"scripcode": "", "Fdate": from\_date, "TDate": to\_date, "Purposecode": "", "strSearch": "S",

"ddlindustrys": "", "ddlcategorys": "E", "segment": "0"}

query\_string = "&".join([f"{k}={v}" for k, v in params.items()])

full\_url = f"{base\_url}?{query\_string}"

headers = {"User-Agent": "Mozilla/5.0", "Accept": "text/csv",

"Referer": "https://www.bseindia.com/corporates/corporate\_act.aspx"}

try:

response = requests.get(full\_url, headers=headers, timeout=30)

response.raise\_for\_status()

csv\_data = response.text.lstrip('\ufeff')

df = pd.read\_csv(StringIO(csv\_data))

df.columns = df.columns.str.strip()

df["Ex Date"] = pd.to\_datetime(df["Ex Date"], errors="coerce")

df["Security Code"] = df["Security Code"].astype(str)

df["CorporateAction"] = df["Purpose"].str.strip().str.slice(0, 40)

df.dropna(subset=['Ex Date'], inplace=True)

return df

except Exception as e:

print(f"❌ Failed to fetch BSE corporate actions: {e}")

return pd.DataFrame()

def compute\_supertrend(df, atr, period=10, multiplier=3):

high = df['High'].to\_numpy()

low = df['Low'].to\_numpy()

close = df['Close'].to\_numpy()

atr\_np = atr.to\_numpy()

hl2 = (high + low) / 2

upperband = hl2 + (multiplier \* atr\_np)

lowerband = hl2 - (multiplier \* atr\_np)

direction = np.ones(len(df), dtype=int)

supertrend = np.full(len(df), np.nan)

for i in range(1, len(df)):

if close[i] > upperband[i-1]: direction[i] = 1

elif close[i] < lowerband[i-1]: direction[i] = -1

else: direction[i] = direction[i-1]

if direction[i] == 1: lowerband[i] = max(lowerband[i], lowerband[i-1])

else: upperband[i] = min(upperband[i], upperband[i-1])

supertrend[i] = lowerband[i] if direction[i] == 1 else upperband[i]

df["Supertrend"] = supertrend

df["Supertrend\_Direction"] = direction

return df

def add\_advanced\_candle\_patterns(df):

body = abs(df['Close'] - df['Open'])

candle\_range = df['High'] - df['Low']

upper\_wick = df['High'] - df[['Close', 'Open']].max(axis=1)

lower\_wick = df[['Close', 'Open']].min(axis=1) - df['Low']

with np.errstate(divide='ignore', invalid='ignore'):

df["Doji"] = ((body / candle\_range) < 0.1).astype(int)

df["Hammer"] = ((lower\_wick > 2 \* body) & (upper\_wick < body)).astype(int)

df["EngulfingBull"] = ((df['Close'] > df['Open']) & (df['Close'].shift(1) < df['Open'].shift(1)) &

(df['Close'] > df['Open'].shift(1)) & (df['Open'] < df['Close'].shift(1))).astype(int)

df["EngulfingBear"] = ((df['Close'] < df['Open']) & (df['Close'].shift(1) > df['Open'].shift(1)) &

(df['Close'] < df['Open'].shift(1)) & (df['Open'] > df['Close'].shift(1))).astype(int)

return df

def compute\_streaks(df):

change = np.sign(df["Close"].diff())

up\_mask = (change == 1)

down\_mask = (change == -1)

df['UpStreak'] = up\_mask.cumsum() - up\_mask.cumsum().where(~up\_mask).ffill().fillna(0)

df['DownStreak'] = down\_mask.cumsum() - down\_mask.cumsum().where(~down\_mask).ffill().fillna(0)

return df

# ==== BUILD DATASET ====

START\_DATE = (datetime.now() - timedelta(days=365\*10)).strftime("%Y-%m-%d")

END\_DATE = datetime.now().strftime("%Y-%m-%d")

TARGET\_DAYS = 5

TARGET\_PCT = 0.02

VOLUME\_SPIKE\_THRESHOLD = 2.0

symbols = get\_nse\_symbols(100)

bse\_df = fetch\_bse\_corporate\_actions()

raw = yf.download(

tickers=symbols,

start=START\_DATE,

end=END\_DATE,

progress=True,

threads=True,

group\_by="ticker",

auto\_adjust=True

)

all\_dfs = []

for sym in tqdm(symbols, desc="Processing symbols"):

try:

if sym not in raw or raw[sym].dropna().empty: continue

df = raw[sym].copy().dropna(subset=['Open', 'High', 'Low', 'Close', 'Volume'])

if df.shape[0] < 250: continue

df.index = pd.to\_datetime(df.index)

df = df.sort\_index()

close = df["Close"]

# --- INDICATORS ---

df["RSI"] = RSIIndicator(close).rsi()

df["MACD"] = MACD(close).macd\_diff()

df["EMA20"] = EMAIndicator(close, 20).ema\_indicator()

df["EMA50"] = EMAIndicator(close, 50).ema\_indicator()

df["SMA50"] = SMAIndicator(close, 50).sma\_indicator()

df["SMA200"] = SMAIndicator(close, 200).sma\_indicator()

df["ADX"] = ADXIndicator(df["High"], df["Low"], close).adx()

bb = BollingerBands(close, 20)

df["BB\_Width"] = (bb.bollinger\_hband() - bb.bollinger\_lband()) / close

df["ATR"] = AverageTrueRange(df["High"], df["Low"], close).average\_true\_range()

# --- CUSTOM FEATURES ---

df = compute\_supertrend(df, atr=df["ATR"])

df = add\_advanced\_candle\_patterns(df)

df = compute\_streaks(df)

# --- EVENT-BASED FEATURES ---

df["MACD\_BullishCross"] = ((df["MACD"] > 0) & (df["MACD"].shift(1) <= 0)).astype(int)

df["GoldenCross"] = ((df["SMA50"] > df["SMA200"]) & (df["SMA50"].shift(1) <= df["SMA200"].shift(1))).astype(int)

df["VolatilityCompression"] = (df["BB\_Width"] < df["BB\_Width"].rolling(20).quantile(0.1)).astype(int)

df["VolumeSpike"] = (df["Volume"] > df["Volume"].rolling(20).mean() \* VOLUME\_SPIKE\_THRESHOLD).astype(int)

# --- TIME-BASED FEATURES ---

df["DayOfWeek"] = df.index.dayofweek

df["Month"] = df.index.month

# --- TARGET VARIABLE ---

df["FutureClose"] = close.shift(-TARGET\_DAYS)

df["Target"] = ((df["FutureClose"] - close) / close > TARGET\_PCT).astype(int)

df["Symbol"] = sym

# --- CORPORATE ACTIONS ---

nse\_base\_symbol = sym.replace(".NS", "")

bse\_relevant = bse\_df[bse\_df['Security Name'].str.contains(nse\_base\_symbol, case=False, na=False)].copy()

if not bse\_relevant.empty:

bse\_agg = bse\_relevant.groupby('Ex Date')['CorporateAction'].apply(lambda x: '; '.join(x.astype(str).unique())).reset\_index()

action\_dates\_df = bse\_agg.set\_index('Ex Date').sort\_index()

action\_dates\_df['action\_date'] = action\_dates\_df.index

merged\_last = pd.merge\_asof(df, action\_dates\_df, left\_index=True, right\_index=True, direction='backward')

df['DaysSinceAction'] = (df.index - merged\_last['action\_date']).dt.days

df['CorporateAction'] = merged\_last['CorporateAction']

merged\_next = pd.merge\_asof(df, action\_dates\_df[['action\_date']], left\_index=True, right\_index=True, direction='forward')

df['DaysUntilAction'] = (merged\_next['action\_date'] - df.index).dt.days

#for col in ['CorporateAction', 'DaysSinceAction', 'DaysUntilAction']:

for col in ['CorporateAction', 'DaysSinceAction', 'DaysUntilAction']:

if col not in df.columns:

df[col] = np.nan if 'Days' in col else '-'

df['CorporateAction'] = df['CorporateAction'].fillna('-')

df["HasCorporateAction"] = (df["CorporateAction"] != "-").astype(int)

df["IsDividend"] = df["CorporateAction"].str.contains("dividend", case=False, na=False).astype(int)

df["IsBonus"] = df["CorporateAction"].str.contains("bonus", case=False, na=False).astype(int)

df["IsSplit"] = df["CorporateAction"].str.contains("split", case=False, na=False).astype(int)

df["DividendAmount"] = pd.to\_numeric(df["CorporateAction"].str.extract(r'Rs.?\s?([\d.]+)', expand=False), errors='coerce')

all\_dfs.append(df)

except Exception as e:

print(f"❌ Error processing {sym}: {e}")

if all\_dfs:

dataset = pd.concat(all\_dfs)

dataset.dropna(subset=['SMA200', 'FutureClose'], inplace=True)

dataset.replace([np.inf, -np.inf], np.nan, inplace=True)

numeric\_cols = dataset.select\_dtypes(include=np.number).columns

dataset[numeric\_cols] = dataset[numeric\_cols].fillna(0)

object\_cols = dataset.select\_dtypes(include='object').columns

dataset[object\_cols] = dataset[object\_cols].fillna('-')

# ----------- INDEX TAGGING (Optimized for Performance) ------------

print("\n🔖 Adding index membership columns...")

index\_files = [f for f in os.listdir(index\_folder) if f.endswith('.csv')]

new\_index\_columns = []

for file in index\_files:

index\_path = os.path.join(index\_folder, file)

try:

with open(index\_path, 'rb') as f:

rawdata = f.read(4096)

encoding = chardet.detect(rawdata)['encoding'] or 'utf-8'

df\_index = pd.read\_csv(index\_path, encoding=encoding)

symbol\_col\_candidates = [c for c in df\_index.columns if "symbol" in c.lower()]

if not symbol\_col\_candidates: continue

symbols\_in\_index = df\_index[symbol\_col\_candidates[0]].astype(str).str.replace('.NS','',regex=False).str.strip().str.upper().unique()

match = re.search(r'MW-(.\*?)-\d{2}-\w{3}-\d{4}\.csv', file, re.IGNORECASE)

if match:

tag = match.group(1).upper().replace('-', '\_').replace("&", "AND")

new\_col = dataset['Symbol'].str.replace('.NS', '', regex=False).str.upper().isin(symbols\_in\_index).astype(int)

new\_col.name = tag

new\_index\_columns.append(new\_col)

print(f"✅ {file} -> {tag} ({len(symbols\_in\_index)} symbols)")

except Exception as e:

print(f"❌ Could not tag {file}: {e}")

if new\_index\_columns:

dataset = pd.concat([dataset] + new\_index\_columns, axis=1)

dataset = dataset.copy()

# ----------- SECTOR TAGGING -----------

print("\n🏷️ Adding PrimarySector column using index columns...")

pattern\_sector = [

("IT", "IT"),

("FMCG", "FMCG"),

("PHARMA", "Pharma"),

("BANK", "Banking"),

("AUTO", "Auto"),

("ENERGY", "Energy"),

("MEDIA", "Media"),

("REALTY", "Realty"),

("METAL", "Metal"),

("HEALTH", "Healthcare"),

("FINANCIAL", "Finance"),

("CAPITAL MARKETS", "Finance"),

("CONSUMER DURABLES", "Consumer"),

("CONSUMPTION", "Consumer"),

("SERVICES", "Services"),

("CPSE", "PSU"),

("PSE", "PSU"),

("INFRA", "Infrastructure"),

("MNC", "MNC"),

("DIGITAL", "Tech"),

("DEFENCE", "Defence"),

("RURAL", "Rural"),

("MANUFACTURING", "Manufacturing"),

("HOUSING", "Real Estate"),

("SHARIAH", "Shariah"),

("ESG", "ESG"),

("QUALITY", "Quality"),

("MOMENTUM", "Momentum"),

("LOW VOLATILITY", "Low Volatility"),

("DIVIDEND", "Dividend"),

("LIQUID", "High Liquidity"),

("MULTICAP", "Multicap"),

("EQUAL WEIGHT", "Equal Weight"),

("VALUE", "Value"),

("ALPHA", "Alpha"),

("GROWTH", "Growth"),

("TOTAL MARKET", "Total Market"),

("LARGEMIDCAP", "Large-Midcap"),

("MIDCAP", "Midcap"),

("SMALLCAP", "Smallcap"),

("MICROCAP", "Microcap"),

("IPO", "IPO"),

]

dataset["PrimarySector"] = "Unclassified"

for pattern, sector in pattern\_sector:

pattern\_key = pattern.replace(" ", "\_").upper()

matching\_cols = [col for col in dataset.columns if pattern\_key in col.upper()]

for col in matching\_cols:

mask = (dataset[col] == 1) & (dataset["PrimarySector"] == "Unclassified")

dataset.loc[mask, "PrimarySector"] = sector

# ✅ Save final dataset with sector

output\_path = f"/content/drive/My Drive/stock\_dataset\_with\_sector\_{datetime.now().strftime('%Y-%m-%d')}.csv"

dataset.to\_csv(output\_path, index=False)

print(f"✅ Dataset updated with PrimarySector and saved as:\n{output\_path}")

# ✅ Preview

#print(dataset[["Symbol", "PrimarySector"]].drop\_duplicates().head(30))

# Option 1: Show the first 30 rows with all columns

display(dataset.head(30)) # This shows a nice table in Colab

else:

print("❌ No stock data was processed.")