Price Fetching

import pandas as pd

import requests

import numpy as np

from datetime import datetime, timedelta

import time

import re

from sqlalchemy import create\_engine, text

from sqlalchemy.types import Integer, Float

import os

from dotenv import load\_dotenv

from sqlalchemy import create\_engine

# -------------------- CONFIGURATION --------------------

# Load environment variables from Credentials.env

dotenv\_path = os.path.join(os.path.dirname(\_\_file\_\_), "Credentials.env")

load\_dotenv(dotenv\_path)

# Database credentials from environment variables

DB\_USERNAME = os.getenv("DB\_USERNAME")

DB\_PASSWORD = os.getenv("DB\_PASSWORD")

DB\_HOST = os.getenv("DB\_HOST")

DB\_PORT = os.getenv("DB\_PORT")

DB\_NAME = os.getenv("DB\_NAME")

# Create database connection with SSL forced

engine = create\_engine(

    f"mysql+pymysql://{DB\_USERNAME}:{DB\_PASSWORD}@{DB\_HOST}:{DB\_PORT}/{DB\_NAME}",

    connect\_args={

        "ssl": {

            "fake\_flag\_to\_enable": True

        }

    }

)

# API Key for Financial Modeling Prep (FMP)

FMP\_API\_KEY = os.getenv("FMP\_API\_KEY")

# File paths (loaded from environment variables)

reit\_list\_path = os.getenv("REIT\_LIST\_PATH")

ffo\_payout\_path = os.getenv("FFO\_PAYOUT\_PATH")

financial\_display\_path = os.getenv("FINANCIAL\_DISPLAY\_PATH")

# -------------------- LOAD REIT UNIVERSE --------------------

try:

    reit\_data = pd.read\_csv(reit\_list\_path)

    def clean\_column\_name(col\_name):

        col\_name = col\_name.strip()

        col\_name = re.sub(r'[^\w]', '\_', col\_name)

        col\_name = re.sub(r'\_+', '\_', col\_name)

        return col\_name

    reit\_data.columns = [clean\_column\_name(col) for col in reit\_data.columns]

    tickers = reit\_data['Ticker'].dropna().astype(str).tolist()  # Ensure tickers are strings

    print("✅ REIT data loaded and sanitized successfully.")

except Exception as e:

    print(f"❌ Error loading REIT list: {e}")

    exit()

# -------------------- FETCH PRICE & VOLUME FROM FMP --------------------

end\_date = datetime.now().strftime('%Y-%m-%d')

start\_date = (datetime.now() - timedelta(days=5 \* 365)).strftime('%Y-%m-%d')

def fetch\_ticker\_data\_fmp(ticker):

    url = f"https://financialmodelingprep.com/api/v3/historical-price-full/{ticker}?apikey={FMP\_API\_KEY}"

    try:

        print(f"🔄 Fetching data for {ticker} from FMP...")

        response = requests.get(url)

        try:

            data = response.json()

        except ValueError:

            print(f"❌ Non-JSON response for {ticker}. Skipping.")

            return pd.DataFrame()

        if not isinstance(data, dict) or "historical" not in data:

            print(f"❌ No data found for {ticker}. Skipping.")

            return pd.DataFrame()

        df = pd.DataFrame(data["historical"])

        df = df[['date', 'close', 'volume']]

        df['ticker'] = ticker  # Add ticker column

        return df

    except Exception as e:

        print(f"❌ Error fetching data for {ticker}: {e}")

        return pd.DataFrame()

# -------------------- FETCH & STORE PRICE DATA --------------------

cleaned\_price\_data = pd.DataFrame()

for ticker in tickers:

    ticker\_data = fetch\_ticker\_data\_fmp(ticker)

    if not ticker\_data.empty:

        cleaned\_price\_data = pd.concat([cleaned\_price\_data, ticker\_data], ignore\_index=True)

    time.sleep(0.02)  # Small delay to avoid rate limits

if cleaned\_price\_data.empty:

    print("❌ No data fetched. Exiting.")

    exit()

# -------------------- SQL DATABASE SETUP --------------------

drop\_price\_table\_query = "DROP TABLE IF EXISTS reit\_price\_data;"

create\_price\_table\_query = (

    "CREATE TABLE reit\_price\_data ("

    "date DATE NOT NULL, "

    "ticker VARCHAR(10) NOT NULL, "

    "close\_price FLOAT NOT NULL, "

    "volume FLOAT, "

    "PRIMARY KEY (date, ticker)"

    ");"

)

# Build business table creation query explicitly

business\_columns = ""

for col in reit\_data.columns:

    if col:

        business\_columns += f"{col} VARCHAR(255), "

business\_columns = business\_columns.rstrip(", ")

create\_business\_table\_query = f"CREATE TABLE IF NOT EXISTS reit\_business\_data ({business\_columns});"

try:

    with engine.connect() as conn:

        print("Dropping existing reit\_price\_data table (if any)...")

        conn.execute(text(drop\_price\_table\_query))

        print("Creating new reit\_price\_data table...")

        conn.execute(text(create\_price\_table\_query))

        print("Verifying/creating reit\_business\_data table...")

        conn.execute(text(create\_business\_table\_query))

        print("✅ SQL Tables created successfully.")

except Exception as e:

    print(f"❌ Error setting up tables: {e}")

    exit()

# -------------------- INSERT PRICE DATA --------------------

cleaned\_price\_data.rename(columns={'date': 'date', 'ticker': 'ticker', 'close': 'close\_price', 'volume': 'volume'}, inplace=True)

cleaned\_price\_data.drop\_duplicates(subset=['date', 'ticker'], keep='last', inplace=True)

try:

    cleaned\_price\_data.to\_sql('reit\_price\_data', con=engine, if\_exists='append', index=False, chunksize=2000, method='multi')

    print("✅ New REIT price data inserted successfully into MySQL.")

except Exception as e:

    print(f"❌ Error inserting price data into MySQL: {e}")

# -------------------- INSERT BUSINESS DATA --------------------

try:

    reit\_data.to\_sql('reit\_business\_data', con=engine, if\_exists='replace', index=False)

    print("✅ New REIT business data inserted successfully into MySQL.")

except Exception as e:

    print(f"❌ Error inserting business data into MySQL: {e}")

# -------------------- LOAD & INSERT FFO PAYOUT DATA --------------------

try:

    ffo\_payout\_data = pd.read\_csv(ffo\_payout\_path, dtype=str, keep\_default\_na=False)

    ffo\_payout\_data["Years"] = pd.to\_numeric(ffo\_payout\_data["Years"], errors="coerce").dropna().astype(int)

    ffo\_payout\_data.replace(["NA", "Inf", "-Inf", ""], np.nan, inplace=True)

    for col in ffo\_payout\_data.columns[1:]:

        ffo\_payout\_data[col] = pd.to\_numeric(ffo\_payout\_data[col], errors="coerce")

    # Sanitize column names for FFO payout data

    def clean\_column\_name(col\_name):

        col\_name = col\_name.strip()

        col\_name = re.sub(r'[^\w]', '\_', col\_name)

        col\_name = re.sub(r'\_+', '\_', col\_name)

        return col\_name

    ffo\_payout\_data.columns = [clean\_column\_name(col) for col in ffo\_payout\_data.columns]

    print("✅ REIT FFO Payout data loaded and sanitized successfully.")

except Exception as e:

    print(f"❌ Error loading REIT FFO Payout list: {e}")

    exit()

# Build FFO payout table creation query explicitly

ffo\_columns = ""

for col in ffo\_payout\_data.columns:

    if col != "Years" and col:

        ffo\_columns += f"{col} FLOAT, "

ffo\_columns = ffo\_columns.rstrip(", ")

if ffo\_columns:

    create\_ffo\_payout\_table\_query = f"CREATE TABLE IF NOT EXISTS reit\_ffo\_payout (Years INT NOT NULL, {ffo\_columns});"

else:

    create\_ffo\_payout\_table\_query = "CREATE TABLE IF NOT EXISTS reit\_ffo\_payout (Years INT NOT NULL);"

try:

    with engine.connect() as conn:

        conn.execute(text(create\_ffo\_payout\_table\_query))

        print("✅ REIT FFO Payout table created successfully.")

except Exception as e:

    print(f"❌ Error creating FFO Payout table: {e}")

    exit()

try:

    with engine.begin() as conn:

        conn.execute(text("DELETE FROM reit\_ffo\_payout;"))

        print("✅ Existing REIT FFO Payout data cleared.")

except Exception as e:

    print(f"❌ Error clearing FFO Payout data: {e}")

    exit()

try:

    ffo\_payout\_data.to\_sql(

        'reit\_ffo\_payout',

        con=engine,

        if\_exists='replace',

        index=False,

        dtype={'Years': Integer()}

    )

    print("✅ New REIT FFO Payout data inserted successfully into MySQL.")

except Exception as e:

    print(f"❌ Error inserting FFO Payout data into MySQL: {e}")

# -------------------- LOAD & INSERT FINANCIAL DISPLAY DATA --------------------

try:

    # Read CSV and skip any columns that start with 'Unnamed' if desired,

    # or keep them (they may appear due to extra delimiters)

    df\_fd = pd.read\_csv(financial\_display\_path)

    # Sanitize all column names

    def clean\_column\_name(col\_name):

        col\_name = col\_name.strip()

        # Replace any non-alphanumeric characters (including colon) with an underscore

        col\_name = re.sub(r'[^\w]+', '\_', col\_name)

        return col\_name

    df\_fd.columns = [clean\_column\_name(c) for c in df\_fd.columns]

    # Ensure first column is "Dates"

    if df\_fd.columns[0] != "Dates":

        df\_fd.rename(columns={df\_fd.columns[0]: "Dates"}, inplace=True)

    # Replace placeholder values

    df\_fd.replace(["NA", "Inf", "-Inf", ""], np.nan, inplace=True)

    # Convert "Dates" to datetime

    df\_fd["Dates"] = pd.to\_datetime(df\_fd["Dates"], errors="coerce")

    # Convert all other columns to numeric

    for col in df\_fd.columns.drop("Dates"):

        df\_fd[col] = pd.to\_numeric(df\_fd[col], errors="coerce")

    print("✅ REIT Financial Display data loaded and sanitized successfully.")

except Exception as e:

    print(f"❌ Error loading REIT Financial Display: {e}")

    exit()

# Build Financial Display table creation query explicitly

fd\_columns = ""

for col in df\_fd.columns:

    if col != "Dates" and col:

        fd\_columns += f"{col} FLOAT, "

fd\_columns = fd\_columns.rstrip(", ")

if fd\_columns:

    create\_financial\_display\_table\_query = f"CREATE TABLE IF NOT EXISTS reit\_financial\_display (Dates DATE, {fd\_columns});"

else:

    create\_financial\_display\_table\_query = "CREATE TABLE IF NOT EXISTS reit\_financial\_display (Dates DATE);"

print("Financial Display Table Query:")

print(create\_financial\_display\_table\_query)

try:

    with engine.connect() as conn:

        conn.execute(text(create\_financial\_display\_table\_query))

        print("✅ REIT Financial Display table created successfully.")

except Exception as e:

    print(f"❌ Error creating Financial Display table: {e}")

    exit()

try:

    with engine.begin() as conn:

        conn.execute(text("DELETE FROM reit\_financial\_display;"))

        print("✅ Existing REIT Financial Display data cleared.")

except Exception as e:

    print(f"❌ Error clearing Financial Display data: {e}")

    exit()

# Use if\_exists='replace' to ensure the table is recreated with the current DataFrame schema

try:

    df\_fd.to\_sql(

        'reit\_financial\_display',

        con=engine,

        if\_exists='replace',  # Recreate the table so its schema matches the DataFrame

        index=False

    )

    print("✅ New REIT Financial Display data inserted successfully into MySQL.")

except Exception as e:

    print(f"❌ Error inserting Financial Display data into MySQL: {e}")