ETL Pipeline Overview

This pipeline is designed to:

1. Collect stock data from various sources.
2. Cleanse and transform the data to ensure consistency and quality.
3. Engineer additional features (e.g., capital gains, daily returns).
4. Aggregate the data by key features like date and symbol.
5. Merge multiple data sources into one consolidated dataset.
6. Load the final dataset to MongoDB for further analysis or use.

Step 1: Data Collection

The ETL pipeline pulls data from multiple sources:

1.1 MarketStack API

- Fetches historical stock data based on tickers (symbols) from MarketStack API or a mock URL when `IS_MOCK = True`.
- **Function**: `get_stock_data_from_marketstack()`
- **Details**:
  - Uses `get_tickers()` to fetch a list of stock tickers.
  - Constructs the API request URL and fetches data for a specified date range.

1.2 Kaggle Dataset

- Fetches a stock price dataset from Kaggle using KaggleHub.
- **Function**: `get_data_from_kaggle_df()`
- **Details**:

    - Uses KaggleHub to load the "World-Stock-Prices-Dataset" and converts it to a DataFrame.

1.3 Local CSV Files

- Fetches historical stock data from a local CSV file.
- **Function**: `get_data_from_local_csv()`
- **Details**:
    - Reads stock data from a local CSV file containing filtered stock data.

1.4 MongoDB Database

- Fetches stock data from MongoDB for the year 2023.
- **Function**: `get_data_from_mongodb()`
- **Details**:
    - Connects to MongoDB and retrieves the stock data, converting it into a DataFrame.
    - Handles the removal of the `_id` field from the MongoDB documents.

1.5 GitHub Dataset

- Fetches stock data from a GitHub repository.
- **Function**: `get_data_from_github()`
- **Details**:
    - Fetches stock data from a public GitHub repository as a CSV file.

Step 2: Data Transformation

The data transformation process includes several sub-steps to ensure that data is clean, standardized, and feature-engineered:

2.1 Standardizing Timestamps

- Converts date columns in different datasets to a consistent datetime format.
- **Function**: `standardize_timestamps()`
- **Details**:
  - Ensures all datasets have their `date` column in the proper `datetime` format.
  - Filters data to include only the year of interest (e.g., 2025).

2.2 Handling Missing Values

- Detects and handles missing values in the datasets.
- **Function**: `check_for_null_values()` and `handle_missing_values()`
- **Details**:
  - Identifies columns with missing values and can handle them by either dropping or imputing values.
  - In the Kaggle dataset, missing values are handled by creating a new feature `capital_gains` as the difference between the close and open values.

2.3 Normalizing Column Names

- Ensures that column names are consistent across all datasets.
- **Function**: `normalize_column_names()`
- **Details**:
  - Converts column names to lowercase and replaces spaces with underscores.

2.4 Validating Data

- Ensures that the data is valid by removing rows with negative values in key financial columns (e.g., `open`, `close`, `high`, `low`, `volume`).
- **Function**: `validate_data()`
- **Details**:

    - Drops rows with negative values to maintain the integrity of the financial data.

2.5 Feature Engineering

- Creates new features for further analysis, such as daily returns and volatility.
- **Function**: `add_features()`
- **Details**:
    - **Daily Return**: Calculated as the difference between `close` and `open` divided by `open`.
    - **Volatility**: Calculated as the difference between `high` and `low`.

Step 3: Data Aggregation

The data is aggregated by ticker symbol and date, creating summary metrics for further analysis:

3.1 Aggregation by Date and Symbol

- Aggregates the financial metrics (open, close, high, low, volume, daily return, volatility) by symbol and date.
- **Function**: `aggregate_data()`
- **Details**:
    - Groups the data by ticker symbol and date and computes aggregate statistics for each group (e.g., mean, sum, max, min).

Step 4: Merging Datasets

The data from various sources (MarketStack, Kaggle, local CSV, MongoDB, GitHub) are merged into a single DataFrame:

4.1 Merge Datasets

- Combines all the aggregated datasets into a single DataFrame.

- **Function**: `merge_datasets()`
- **Details**:
    - Uses `pd.concat()` to combine all the datasets.
    - Drops duplicates based on the `symbol` and `date_only` columns to ensure no duplicate data.

Step 5: Data Loading

After processing and merging the data, the final step is to load the data into MongoDB:

5.1 Loading Data to MongoDB

- Loads the processed data to MongoDB for storage and further use.
- **Function**: `load_data()`
- **Details**:
    - Converts the `date_only` column to `datetime` to avoid BSON encoding issues in MongoDB.
    - Inserts the data into the `stocksdata_new` collection in MongoDB.

Step 6: Scheduling the ETL Task

The ETL process is scheduled to run daily at a specified time (e.g., 12:00 PM) to fetch and process the latest stock data.

6.1 Scheduling the Task

- Uses the `schedule` library to schedule the ETL task.
- **Function**: `run_daily_etl()`
- **Details**:

- Initializes the ETL pipeline and runs the transformation and loading processes.
- The task is scheduled to run every day at 12:00 PM.