

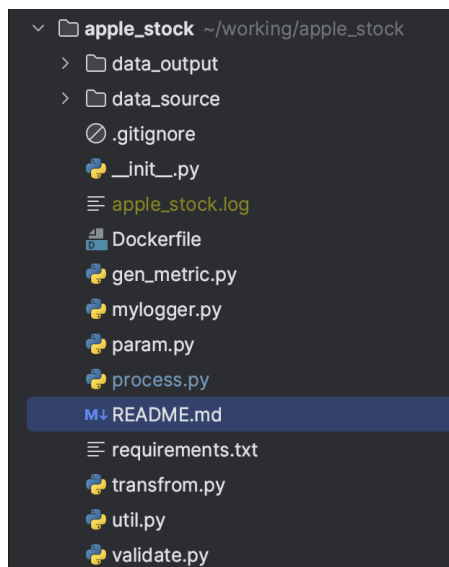
APPLE STOCK DATASET PROCESS

1. Purpose

The purpose of this document is to present how to process an Apple stock dataset. This document has three parts

- Project structure
- Set up and run project
- Technical explanation

2. Project structure



The project includes the following components:

- **data_source:** This is the location that stores the data source.
- **data_output:** This is the location that saves the output results of the data.
- **Dockerfile:** This file is used to create a Docker image.
- **Log:** The log is implemented in the file 'mylogger.py,' and the log file is 'apple_stock.log.' This file keeps track of all the steps in the processing.

- **param**: This is the place to declare all parameters used in the project.
- **requirements**: This lists all the libraries that need to be installed in the project.
- **util**: This is the place that defines all utility functions used for this project.
- **validate**: This is the place that defines functions used for validating data in the project.
- **transform**: This is the place that defines functions used for transforming data in the project.
- **gen_metric**: This is the place that defines functions used for aggregating data and generating metric data.
- **process**: This file defines the main function.

3. Setup and Run the project

The project could be run from Docker image or directly from python.

3.1 Run from Docker image

We need Docker desktop, you can download it from <https://www.docker.com/products/docker-desktop>

Steps to build an image and run it as below:

```
# Build Docker Image
```

```
docker build -t process .
```

```
# Run and create a container "apple_stock"
```

```
docker run -p 8080:8080 --name apple_stock process
```

3.2 Run from source code.

To install the environment and run the project, follow these general steps:

Set up Environment:

- Install Python: Ensure Python is installed on your system. You can download it from python.org.
- Create a Virtual Environment (optional but recommended): Navigate to the project folder in the terminal and run:

```
python -m venv venv
```

- Activate the virtual environment:
 - On Windows: venv\Scripts\activate
 - On macOS/Linux: source venv/bin/activate

Install Dependencies:

- Navigate to your project folder where the requirements.txt file is located.
- Run the following command to install dependencies:

```
pip install -r requirements.txt
```

Run the Project:

- Execute the main file or command to start your project.

```
python process.py
```

3. Technical explanation

The project is compatible with Python version 3.10 and necessitates specific libraries outlined below.

- Pandas 2.1.4
- Pandas_schema 0.3.6
- Dash 2.14.2
- Plotly 5.18.0

he data will undergo the following steps:

- Cleaning and validating data
- Transforming data
- Aggregating data
- Generating metrics
- Visualizing data through a candle chart graph.

Cleaning and validating data : For the Apple stock dataset, certain rules should be applied to the data.

- Date : has format yyyy-mm-dd and should be Monday → Friday and no duplicate
- Price : is integer positive
- Adjust : is integer and can be negative

- Direction : should have data in ["Increasing","Decreasing"]
- All columns are non-null

I use **pandas_schema.validation** and **CustomElementValidation** to validate those rules above by define function to check those rules and use lambda function to apply each column

```
date_validation = [CustomElementValidation(lambda d: U.check_date(d), 'It should be YYYY-mm-dd')]
int_validation = [CustomElementValidation(lambda d: U.check_int(d), 'is not positive integer')]
null_validation = [CustomElementValidation(lambda d: U.check_null(d), 'this field cannot be null')]
trend_validation = [CustomElementValidation(lambda d: U.check_trend(d), 'trend should be increasing or decreasing')]
out_of_business_date = [
    CustomElementValidation(lambda d: U.check_day_of_week(d, P.BUSINESS_DATE_NUMBER), 'out of business date')]
int_adjust = [CustomElementValidation(lambda d: U.check_adjust(d), 'is not integer')]

schema = pandas_schema.Schema([
    Column("Date", date_validation + null_validation + out_of_business_date),
    Column("AAPL.Open", int_validation + null_validation),
    Column("AAPL.High", int_validation + null_validation),
    Column("AAPL.Low", int_validation + null_validation),
    Column("AAPL.Close", int_validation + null_validation),
    Column("AAPL.Volume", int_validation + null_validation),
    Column("AAPL.Adjusted", int_adjust + null_validation),
    Column("dn", int_validation + null_validation),
    Column("avg", int_validation + null_validation),
    Column("up", int_validation + null_validation),
    Column("direction", trend_validation + null_validation)
```

With duplicate Date value : I use function duplicated to check

With data does not match with the rule or duplicate will be extract and save to error file and save at data_output directory

```

v  folder apple_stock ~/working/apple_stock
  v  folder data_output
    ≡  dirty_data
    ≡  duplicate
    ≡  exceeded_avg.csv
```

Transforming data :

I added two more columns

- day_of_week : using `pd.to_datetime(df['Date']).dt.day_name()` to get day of week (Monday , Tuesday)
- Week_of_year : `pd.to_datetime(df['Date']).dt.strftime('%Y%U')` to get week of year . Ex: 201508 it means the eighth week of 2015 . Because I will use this column to aggregate data for the week.

	Date	AAPL.Open	AAPL.High	AAPL.Low	AAPL.Close	AAPL.Volume	AAPL.Adjusted	dn	mavg	up	direction	day_of_week	week_of_year
0	2015-02-17	127.489998	128.880005	126.919998	127.830002	63152400	122.905254	106.741052	117.927667	129.114281	Increasing	Tuesday	201507
1	2015-02-18	127.629997	128.779999	127.449997	128.720001	44891700	123.760965	107.842423	118.940333	130.038244	Increasing	Wednesday	201507
2	2015-02-19	128.479996	129.029999	128.330002	128.449997	37362400	123.501363	108.894245	119.889167	130.884089	Decreasing	Thursday	201507
3	2015-02-20	128.619995	129.500000	128.050003	129.500000	48948400	124.510914	109.785449	120.763500	131.741551	Increasing	Friday	201507
4	2015-02-23	130.020004	133.000000	129.660004	133.000000	70974100	127.876074	110.372516	121.720167	133.867817	Increasing	Monday	201508

2024-01-07 12:57:09.153 - apple_stock - INFO - Calculate avg/min/max of AAPL.Close

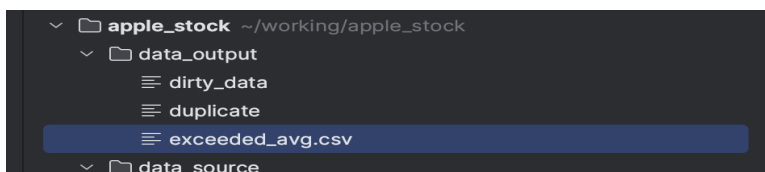
Aggregate data and generate metric :

- I use mean(), min() , max() function to aggregate APPL_Close price .

```
def agg_min_max_avg_close_price(df: DataFrame, derivative_column: str) -> DataFrame:
    try:
        logger.info('Calculate avg/min/max of {}'.format(derivative_column))
        max_value = U.agg_max(df, derivative_column)
        min_value = U.agg_min(df, derivative_column)
        avg_value = U.agg_mean(df, derivative_column)
        logger.info(
            'Calculate avg Volume End with max = {0} , min = {1} , avg = {2} '.format(max_value, min_value, avg_value))
        return pd.DataFrame([{"max_price": [max_value], "min_price": [min_value], "avg_price": [avg_value]}])
    except Exception as e:
        logger.error(f"error in function agg_min_max_avg_close_price {e}:e")
```

- I calculated average of Volume and then find all of records greater than that value then save those rows to file

```
def calculate_avg_volume(df: DataFrame, derivative_column: str):
    try:
        logger.info('Calculate avg Volume Start ')
        avg_volume_value = U.agg_mean(df, derivative_column)
        df = df.query("{}` > @avg_volume_value ".format(derivative_column))
        logger.info('Exceeded file saved at {}'.format(P.DATA_OUT_PUT_PATH,P.OUTPUT_EXCEEDED_FILENAME))
        U.save_csv_data(pd.DataFrame(df), P.DATA_OUT_PUT_PATH, P.OUTPUT_EXCEEDED_FILENAME)
        logger.info("Calculate avg Volume End")
        return df
    except Exception as e:
        logger.error(f"error in function calculate_avg_volume {e}:e")
```



- I generate metric with week level and aggregate with mean value for Columns “APPL.Close” , “APPL.Low”, “APPL.High”, “APPL.Open”

```

Usage: 1. main.py
def generate_metric(df: DataFrame) -> DataFrame:
    try:
        logger.info('Generate Metric Start ')
        df_agg = df.groupby(["week_of_year"])[["AAPL.Close", "AAPL.Low", "AAPL.High", "AAPL.Open"]].mean().reset_index()
        logger.info('Generate Metric End ')
        return df_agg.sort_values("week_of_year", ascending=[True])
    except Exception as e:
        logger.error(f"error in function generate_metric {e}:e")

```

Graph Data

I use dash , plotly package to plot data with candle chart with daily chart and weekly chart

With daily chart :

Axis x is Date

Open is APPL.open

Close is APPL.close

Hight is APPL.high

Low is APPL.low

With daily weekly chart :

Axis x is weekof_year

Open is APPL.open(mean of week)

Close is APPL.close(mean of week)

Hight is APPL.high(mean of week)

Low is APPL.low(mean of week)

The mean weekly stock price of Apple throughout the year



The mean daily stock price of Apple throughout the year

