Documentation of the European Factor Stockpicking and Screener project

Jonathan Willnow

08 March 2022

CONTENTS

CHAPTER

ONE

INTRODUCTION

You can find the documentation on the rationale, pytask, and more background at https://econ-project-templates.readthedocs.io/en/stable/.

1.1 Getting started

This assumes you have completed the steps in the `Getting Started section of the documentation <https://econ-project-templates.readthedocs.io/en/stable/getting_started.html>`_ and **everything worked.

The logic of the project template works by step of the analysis:

1. Original data / Data scraping
2. Data management
3. Final - The actual product
4. Visualisation and Presentation using a Dash-App
5. Research paper and online documentation

1.2 Requirements

There are three files that list the requirements for this project: - scraper_env.yml contains all requirements that are needed to use the scrapers. - environment.yaml contains all reqquirements to build the output/product from the scraped data - requirements.txt contains all requirements to run the DashApp alone

Example: Updating the environment from environment.yml to try out the scraper can be done by:

conda activate stockpicking_screener

conda env update –file sraper_env.yml –prune

Since the requirements for the Dash-App are also contained in the environment.yml, I recommend to use the environment.yml file if you just want to have a look at the tasks and processes involved in the building of the final product. If you want to try the scraping, go ahead but keep in mind that it will take several hours/ days to scrape all the stocks and at least hours to scrape the corresponding metrics and numbers.

CHAPTER

TWO

ORIGINAL DATA

Documentation of the different datasets in *original_data*.

2.1 Data Scraping

The following functions are used to obtain the datasets for the different stockindicies / stockexchnages contained in *original_data* and can be found in *data_management*.

2.2 Stockinfo Scraping

Several functions that obtain the info about stocks of a specific exchange / index. Currently this has to be done manually since it uses Selenium and BeautifoulSoup, which tend to be a bit unstable. From time to time it can happen that the scraper runs into a problem and the user then has to fix it by hand (for isntance closing an advertisement window which is randomly triggered and is not detected by Selenium).

`get_stock_data`(*url*, *index_exchange*)

Function to fetch stock information such as name and wkn from a specified URL of the traderfox.de website with an specified index / exchange. The function relies on Selenium and BeautifulSoup.

Inputs:

- url (str): one of the available urls from traderfox.de.
- index_exchange (str): name of the index / exchnage from which the stocks should be selected (e.g.: Nasdaq, Amex, NYSE, ...).

Returns:

- A pd.DataFrame containing the name, wkn, and index / exchange of stocks.

`get_ticker`(*initial_frame*)

This function provides usefull information about a stock by its wkn. The function provides the stocks ticker, de_ticker, ISIN and the industry in which the comany is operating within. The function relies on Selenium and BeautifulSoup. For security reasons, I am using a VPN / Privacy Badger to avid getting blocked while scraping, which might caus issues when you try to run it. To run it, you only have to tailor the browser and the extensions used to your specific requirements.

To get all the information, I am using finanznachrichten.de and finance.yahoo.com as sources.

Inputs:

- initial_frame(pd.DataFrame): A pd.DataFrame contating the wkn and name of the stocks

Returns:

- A pd.DataFrame with all the collected information.

`validate_ticker`(*path*)

Function to update the stock information for the different exchanges / indicies The function closely follows get_ticker() and is also based on Selenium / BS4. This function is usefull to:

- check if info is up to date.
- after adding new stocks to the {exchange / index}Stocks.csv files.
- to check that scraping was not corruped by bad internet connection or other issues.
- to validate the scraping.

Inputs: path (str): path of the file for which to perform the validation.

Returns: None, but saves the validated run of the file.

Task file to execute the data scraping that is done as a first step

The following is done with pytask:

- collect wkn and names of stocks on all different available indicies and exchnages listed on traderfox.de

The following is planned to implement with pytask, but is quite complicated to achieve because it is 1) really slow,

- collect info on the stocks, such as industry, ISIN, ticker_de and ticker (This is sofar done manually by calling the script and let it run overnight)

`task_get_EuroStoxx600Stocks`(*produces*)

Pytask function to get all the stocks by name and wkn of the EuroStoxx600. If this works on your machine, a firefox browser is opened and used to scape the data.

CHAPTER

THREE

DATA MANAGEMENT

Documentation of the code in *src/data_management*.

3.1 Process of obtaining metrics

This code is the core of this project. It is used for collecting and calculating all the metrics wich will be used for https://stockpickingapp.herokuapp.com. It is planned to collect the metrics of the stocks every week, or every 2 weeks. In a future version and once enough weekly or two-weekly period data is collected, the Dash-App shall also incorporate this historic metrics.

calc_precentiles(*final_frame*)

Function to calculate percentiles of several selected metrics.

Inputs:

- final_frame(pd.DataFrame): pd.DataFrame with all the information and the metrics for which percentiles should be calculated.

Returns:

- The same pd.DataFrame, but now also with the calculated percentiles.

calculate_FF_CA(*stock*)

Function to calculate the Fama French Conservative Asset Factor. The Conservative Asset Factor is defined as the ratio of total assets of a stock at the fiscal year end of t-1 and the total assets at fiscal year end of t-2. For more informations, refer to Fama and French (2015).

Inputs:

- stock(str): The ticker smybol used in get_data()

Returns:

- A pd.DataFrame containing information about the metrics building up the factor and information about the factor itself. Additional I report growth rates for all emtrics and the factor.

calculate_FF_Quality(*stock*)

Function to calculate the Fama French Quality Factor. This factor is calculated using all accounting numbers from the end of the previous fiscal year. It is defined by the annual revenues minus the cost of goods sold, interest expenses, selling, general, and administrative expenses divided by the book equity. For more informations, refer to Fama and French (2015).

Inputs:

- stock(str): The ticker smybol used in get_data()

Returns:

- A pd.DataFrame containing information about the metrics building up the factor and information about the factor itself. Additional I report growth rates for all emtrics and the factor. This helps indentify outliers and steady trends.

`clean_stock_selection`(*uncleaned_stocks*)

Function to perform some cleaning before using urllib.requests and finance.yahoo.com to obtain all the stockinfo.

Inputs:

- uncleaned_stocks(pd.DataFrame): pd.DataFrame containing the stocks (name, wkn, exchange, ticker, ISIN, ...)

Returns:

- pd.DataFrame cleaned s.t. every stock has an industry.

`fun_process_stocks`(*stockinfo_pkl*, *datalist*)

Function that combines the whole process of obtaining the information. This function in itself calls clean_stock_selection(), get_data() and calc_percentiles(). To increase the performance, this function uses ThreadPool from multiprocessing.pool. This allows multithreading. The number of threads is automatically determined by the default option of ThreadPool.

Inputs:

- stockinfo_pkl(pd.DataFrame): A pd.DataFrame that contains the information about stocks. Must contain ticker and industry such that the function works.
- datalist(str): provides a namethat is incorporated into the naming of the produced file.

Return:

- None, but saves a file.

`get_data`(*stockticker*)

Function to obtain a large amounts of mertics from finance.yahoo.com for a specific ticker. This is done by using urllib.requests and exploiting the react frontend of finance.yahoo.com.

Inputs:

- stockticker(str): The homeexchange ticker of a specified stock, e.g.: BC8.F for the german Bechtle AG.

Returns:

- A pd.DataFrame with metrics. For a list of the metrics that a re currently collected ahve a look at the implementation of the function.

`task_process_eu_stocks`(*depends_on*, *produces*)

Pytask function to collect the metrics for all the atocks of the EuroStoxx600 index.

Inputs:

- depends_on(): A pickle file containing all the information that is needed to obtain the metrics.

Returns:

- produces(): A pickle file containing all the collected indormations and metrics.

CHAPTER

FOUR

BUILDING UP THE FINAL DATASET FOR THE STOCKSCREENER

Documentation of the code in *src.final*.

4.1 Process of building final product

calculate_precentiles(*rv_dataframe*, *metric_dict*)

Function to calculate the percentiles of the calculated and obtained metrics. This gives the user and intuition about an particular stock compared to other stocks.

Inputs:

- rv_dataframe(pd.DataFrame): A pd.DataFrame with the stockinformation.
- metric_dict(dict): A python dicionary with the pairs ondicating for which metrics the percentiles should be calculated.

Returns:

- A pd.DataFrame with the calculated percentiles and the initial info.

get_european_weights_ken_french()

Function to obtain the historic European 5 Fama French Factors from the official webiste. These will then be used for the weighting od the factors to build the final score. Inputs:

- None

Returns:

- pd.DataFrame with the historic average monthly performance of the factors over the market return.

This code is cleaning up the collected data and builds the final dataset which is then used for the stockscreening on https://stockpickingapp.herokuapp.com. I tried to limit the cleaning in this step as much as possible since there is more filtering possible on https://stockpickingapp.herokuapp.com.

save_data(*sample*, *path*)

Function to save the data / sample as pickle in a specified location.

Inputs: sample(pd.DataFrame): DataFrame with the data that has to be saved as pickle. path (str): path of the file for which to perform the validation.

task_create_product(*depends_on*, *produces*)

Pytask function to create the european product. The output can be found in src.product_data.

CHAPTER

FIVE

VISUALISATION AND PRESENTATION USING A DASH-APP

Documentation of the code in *src.dashoard.pages*.

5.1 Dashboard

CHAPTER

SIX

RESEARCH PAPER / PRESENTATIONS

Purpose of the different files (rename them to your liking):

- `research_paper.tex` contains the actual paper.

CHAPTER

SEVEN

REFERENCES

PYTHON MODULE INDEX

d

f

INDEX