Quantitative Research Modeling Library

Design Document v2

Team
sdMay19-06

Team Members
Josiah Anderson -- Meeting Facilitator
Doh Yun Kim -- Scribe
Gabriel Klein -- Report Manager
Drake Mossman -- Communication Manager
Jacob Richards -- Quality Assurance Manager
Nathan Schaffer -- Overseer

Client
Joseph Byrum
(Principal Financial Group)

Advisor
Srikanta Tirthapura

Contact
sdmay19-06@iastate.edu
https://sdmay19-06.sd.ece.iastate.edu

Last Updated:
2 December 2018

Table of Contents

List of Figures

List of Tables

None

1 Introduction

1.1 Acknowledgement

Team 06's Faculty advisor: Srikanta Tirthapura
Team 06's client: Principal Financial, primarily Benjamin Harlander and Vishnu Vemuru
Interviewed Data Scientists: Josh Zimmerman, Krisoye Smith, Ryan Lam, Markus Sauter, Q Mabasa, Yaoliang He.
We would like to thank Srikanta Tirthapura for being our advisor and helping us out.

1.2 Problem and Project Statement

Principal Financial has a relatively new team of data scientists and interns who work within their Global Investments Department. The goal of this team is to analyze equity data to develop new quantitative strategies the company can use to make informed investment decisions. This team does not have much consistency or standardization in many of their common tasks, which slows down productivity and leaves excessive room for error by requiring duplication of efforts by individuals working on different yet overlapping tasks. Through interviewing members of the data science team at Principal Financial, we found three main areas where the process (or processes) being used were not optimal.

- The first area was in the aggregation of stock level data to the portfolio level. This process can be very time consuming, especially for less experienced team members. Nearly every project involves data aggregation and it is often done many many times within the life of one project.

- The second area was in the handling of predictive models. A lot of similar models are used on multiple projects, and currently they are being rewritten for each new project. This causes some duplicity that could be reduced.

- The third area is in the visualization of the data. Currently there is not a lot of visibility into the individual steps of the process used to create prediction models for stock data.

The inefficiencies in these three areas existed across many of the projects handled by the equities team at Principal, but for the scope of this project we decided to focus our efforts into solving these problems for one process specifically. Multiple student teams across the United States are working on the Dynamic Risk Premium 2.0, which from now on will be referred to as DRP 2.0. The goal of the DRP 2.0 is to create predictive models for stock data using machine learning models. Our task is to create a uniform software pipeline that can be used to connect each of the various components in this multi-step process. This solution needs to be easy to integrate into the current workflow for minimal disruption and save the company time by

introducing standards and automation. Our hope is to create a flexible, user-friendly pipeline that can be used for projects across Principal's Global Investments team.

1.3 Operational Environment

Since our project is entirely software-defined, the operating environment is defined by the systems it may run on and the data which it will be manipulating.

During our current development, our working environment is an AWS server we have been given access to, and a Postgres database containing decades of weekly and monthly stock data up to August 2018. The data is static, since it hasn't been updated for months. While it is mostly complete data, there are a few anomalies with NaN values that must be taken into consideration.

We anticipate the final environment of our library to be work computers within Principal's data science teams and servers running automatically scheduled tasks. When the operating environment is individual computers, the data may come from a local csv containing some view into a more expansive database, or it could come from a more direct database connection within the same script. Once DRP 2.0 moves past the prototype stage within Principal, parts of our library will likely be utilized in automatic tasks to predict the performance of current stocks given weekly-updated data. We expect the list of factors, companies involved, and completeness of the data for both of these cases to be fundamentally similar to those of the database we are working with now.

1.4 Intended Users and Uses

The intended users of this project are the 11 data scientists already employed at Principal Financial and any future data scientists that will work alongside them. This means that this tool must be easy to grasp and must be intuitive for those experienced in data science. The uses of this product should be similar for all users: to take stock equity data and use it to model the most likely trajectory of those stocks.

1.5 Assumptions and Limitations

Assumptions:

1. All data will be reachable through a SQL database consisting of stock level data taken from Factset and Bloomberg.
2. Users will have basic knowledge of either Python or R.
3. Users will have access to a personal computer to run scripts on.

Limitations:

1. Some data scientists using the pipeline may not be expert programmers.

2. Some data scientists may know either Python or R well but not both.
3. There exist already completed components with standardized inputs and outputs.

1.6 Expected End Product and Deliverables

Our final expected end product will be a unified pipeline for the DRP 2.0, standardizing the process of how data is passed between each stage.

The deliverables will be:

1. The DRP 2.0 pipeline framework as Python/R packages
2. Documentation on using the pipeline's functionality for each version of the library

DRP 2.0 Pipeline

The pipeline will allow a user to perform data science functions by taking raw stock level data from a database and transforming it into portfolio simulations and predictions using user input. The user will be able to specify the stock universe, portfolio strategies, prediction models, and factor policies among other variables to run different simulations and see different predictions. It will be implemented in Python and be accessible as both a Python package as written and as an R package through a wrapper. There will also be a number of places where diagnostic or similar interfaces can examine the data as it's being processed for visualization and examination. This deliverable will be delivered on May 10th, 2019 as the Spring semester ends.

Documentation

The documentation for the pipeline will consist of guides for using either the Python or R packages. Each will contain essentially the same material, with potentially different function names or argument types as the languages differ in requirements. They will cover all relevant functions and data types necessary for a user to operate the pipeline from start to finish. Both will be delivered as pdf documents for users to browse as necessary while using the packages. This deliverable will also be delivered on May 10th, 2019 as the Spring semester ends.

2 Specifications and Analysis

2.1 Proposed Design

To come up with our design, our team first spent a significant amount of time researching our client's situation to establish their needs by interviewing employees. We carried out several interviews with various kinds of employees and found that their workflow is in serious need of automation for the less skilled employees, as well as for the many students and interns they work with. Specifically it would be helpful if we could automate any of the following tasks:

- Retrieving and formatting data
- Preprocessing data for the model
- Constructing accurate predictive models
- Postprocessing the model for validation
- Visualizing the result

The DRP 2.0 pipeline helps with all of these tasks by providing a framework for users to use when writing scripts in Python or R. The pipeline provides hooks along with a standardized system of inputs and outputs that allow users to plug in the scripts they'd like to use for testing with little hassle. We're developing the pipeline using Python, a common data science language.

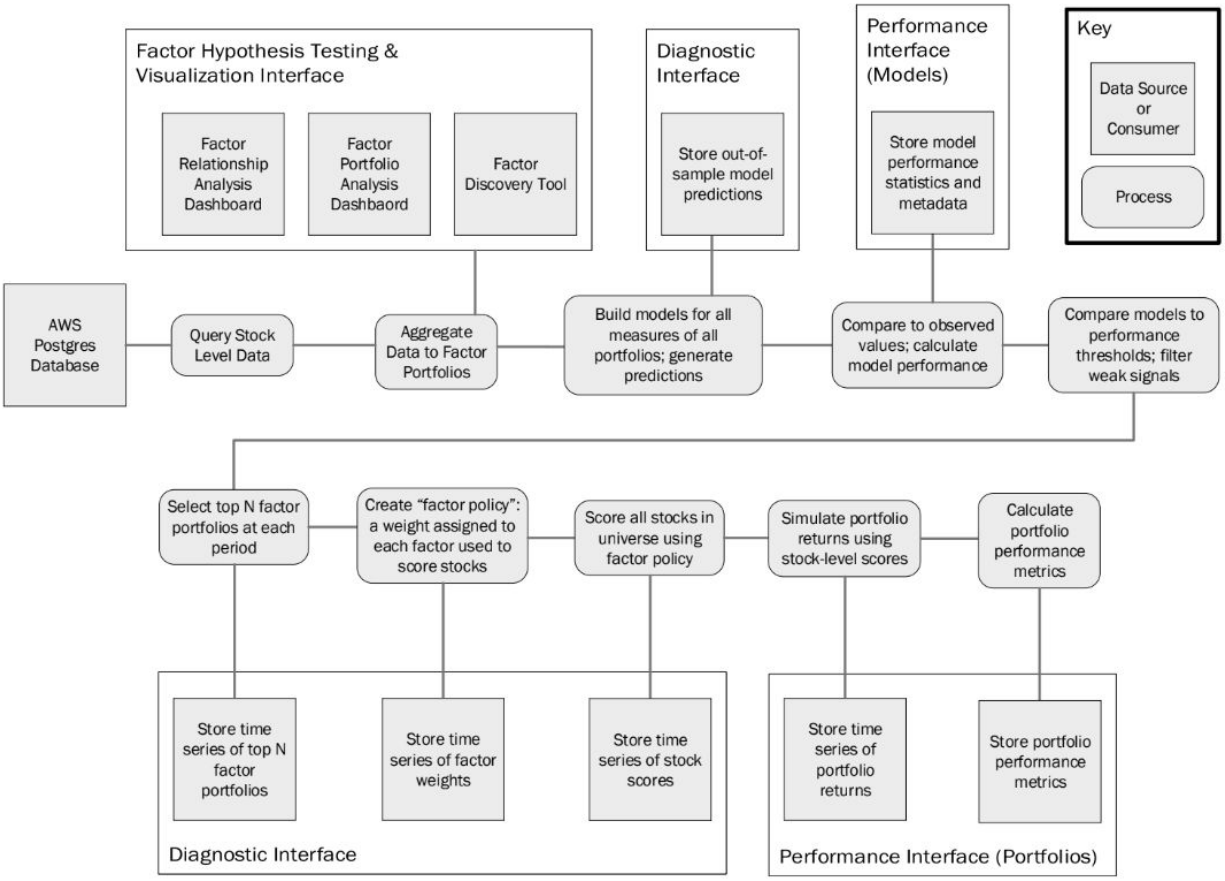

Figure 1: Pipeline Process Diagram with Interface Hooks

The internals of the pipeline make heavy use of the libraries our client is already making use of including numpy, pandas, sklearn, and matplotlib. Each of these already implements a lot of very useful functionality, so our pipeline is mainly focused on combining them to perform larger tasks.

It consumes stock level data from an AWS Postgres database, meaning weekly or monthly factor data for various stocks. This data initially came from FactSet, a service providing financial data to our client.

The pipeline offers several classes to be used as arguments and return values. These represent objects such as Factors and Factor Portfolios, and are made to make the use of the functions as intuitive as possible by reducing the number of arguments by encapsulating the relevant data.

The pipeline is also split into many function calls. This makes it so that a user can easily only make use of the parts they need for their specific project. A user could even just make use of a small portion of pipeline without intending to use the whole thing if they want to. This ensures that the pipeline is versatile and modular, which also makes future maintenance and iteration easier on developers.

To allow users a closer look into the internal processes of the pipeline, it also offers hooks for interfaces to connect to and examine or display data as it is processed. Figure 1 above shows several of these points and what they could be used for. The pipeline may also store such data in csv files or in a database for interfaces to read from at a later time.

The following are our functional and non-functional requirements.

Functional Requirements:

1. The pipeline will consume raw stock level data from an AWS Postgres database
2. The pipeline will be able to aggregate data to factor portfolios
3. The pipeline will be able to build models on factor portfolios, generate predictions, and calculate model performance
4. The pipeline will be able to score stocks using a factor policy
5. The pipeline will be able to simulate portfolio returns
6. Users will be able to customize the stock universe, factor portfolio strategies, model algorithms, and factor policy of the DRP 2.0 pipeline
7. Each function of the pipeline shall be able to be called and run independently
8. The pipeline will be able to handle missing or invalid stock level data
9. It will be able to interface with the DRP 2.0 dashboards
10. It will be able to run in both Python and R

Non-functional Requirements:

1. (*Performance*) The pipeline will be able to handle up to several GB of data
2. (*Performance*) The pipeline should be able to handle millions of observations and hundreds of factors
3. (*Maintainability*) The pipeline will have documented and standardized inputs and outputs for each of its functions and interfaces
4. (*Usability*) The pipeline interfaces will be intuitive to a novice data scientist
5. (*Accessibility*) The pipeline will be useable by novice programmers
6. (*Security*) The pipeline will not leak confidential data to outside sources
7. (*Compatibility*) The pipeline will be able to perform the same functionality as provided existing scripts

2.2 Design Analysis

Strengths of proposed design

One of the most obvious and great strength of our design is the language itself. Python is a great general purpose language with great support. It is lightweight, requires very little overhead for the code, and it is also used in the data science field. All this combined makes using Python as our language of choice for our project perfect. And since most of our client's employees are familiar with Python, we do not have to worry that they will not be able to not use our pipeline.

With our proposed approach, our pipeline is of a module design. Currently there are parts of the pipeline which are not completely finished yet, thus we have no idea how they work, or what their inputs and outputs look like. With that in mine, a module design would make the most sense. The individual parts that our client has made can be swapped in and out as needed depending on what model they need. Since the goal of our project was to create a unified pipeline for our client to use, this module design with libraries suits our client's needs perfectly. It will also allow the easy changing of our library itself, so our pipeline can be adapted to any other project, not just the DRP 2.0.

Also, thanks to the versatility of Python, we found an easy way to wrap our library in R. The majority of our client's data scientist team is proficient in R with some knowledge in Python. Writing our original project in R would've been harder than to do in Python. Thanks to the versatility and the various libraries in Python, we can easily wrap our project in R and people with both R and Python backgrounds can easily use our project.

Weakness of proposed design

The current main weakness of our proposal is that most of the components are not fully defined as of the current date. Due to this limitation, there are only a number of particular ways which we can solve some problems. The design choices we make must always have this in mind, limiting our future options.

Another problem with this proposal is that the user verbose nature of our functions. The combination of using Python and the nature of the arguments needed to be passed makes this inevitable. Our clients might find it less than ideal to write out long arguments for every step of the pipeline. While Python does make some of our programming work easier, for non-software, computer engineers, this design can make it possible for our client to be hesitant in accepting our design choice.

2.3 Alternative Designs

The following consist of a few approaches we considered but eventually rejected in favor of our current plan.

Standalone Application

One way we considered meeting our objective was by creating a standalone application that could run various data manipulation processes through a UI. The user would feed data to the application and choose the processes to be run on it. The UI would also allow for a significant amount of custom configuration for users to adjust as needed.

Strengths:

1. Doesn't require any coding to process data
2. Able to keep track of user preferences and state between sessions
3. Intuitive to use
4. Can easily save and reload models and results
5. Doesn't require installation of dependencies
6. Doesn't require user to use a particular language for other data manipulation

Weaknesses:

1. Difficult to feed results back into code
2. Unfamiliar concept for client's employees
3. Additional functionality requires building more UI
4. Less platform independent
5. Doesn't update along with packages automatically

We didn't choose this approach mainly because our client wants the end product to be available to many users easily and without a high learning curve. This application would require users to learn an entirely new interface that doesn't even mesh well with their existing workflows. Normally data is passed almost exclusively through code, so adding an application that has to import and export the data into the mix is somewhat awkward.

Browser Application

Another approach we considered was a browser application that could run all of the data manipulation processes remotely. The user could upload data and choose processes to be run on it. The application would provide a UI that could configure the processes as necessary. These processes would then be run on a server and the results sent back to the user.

Strengths:

1. All processes can be run on a server with above average processing power and memory

2. Results can be saved remotely and shared with other users
3. Doesn't require the user's computer to be available while running
4. Doesn't require coding to process data
5. Able to keep track of user preferences and state between sessions
6. Intuitive to use
7. Doesn't require installation of dependencies
8. Doesn't require user to use a particular language for other data manipulation

Weaknesses:

1. User must be online initially and to get the results
2. Big data must be uploaded and downloaded often
3. Difficult to feed results back into code
4. Unfamiliar concept for client's employees
5. Additional functionality requires building more UI
6. Requires a server to be accessed from

Similar to the previous approach, the browser application was not chosen because of the learning curve and adjustment to workflow required. With the added complication of uploading and downloading the data, this approach could serious disrupt the flow of data without significant benefits.

3 Testing and Implementation

3.1 Interface Specifications

Due to the modular nature of our pipeline, specific interfaces for testing will not be necessary. Function calls are atomic and don't modify hidden internal state, so they can simply be unit tested as written without needing extra scaffolding. Additionally, our classes also do not have hidden state because the user may need to construct or deconstruct objects on their own to use only a specific part of the pipeline. Thus they also do not need extra interfaces and can simply be tested as they are.

The public interfaces to be used for functions and classes are still primarily unknown at this time. They will be developed further as various portions of the pipeline are addressed and implemented. Our work on the project so far is not yet at the level of detail where we know the specific arguments and objects to be used yet.

3.2 Hardware and software

Testing hardware will only require our own personal computers.

- Using our own personal computers allows us to regularly test our scripts.

Testing software will require writing Python test scripts.

- Our GitLab has Continuous Integration set up to automatically run all test scripts we write and place in the correct GitLab folder.
- The tests run automatically with every push made to GitLab.

3.3 Functional Testing

Unit tests will be thoroughly written and reviewed at each stage of the pipeline to ensure that the results are what is expected. We are able to compare the results of the unit tests with the old code previously given to us by Principal.

By implementing Continuous Integration in GitLab, regression testing is implemented as we add the unit tests to each stage of our development. We are able to use the tests that were written for each part of the pipeline. Any code that is merged to master has to pass all previous tests for each stage of the pipeline. Once the new code is merged to master successfully, the unit tests written for the new code are added to the list of tests that must run successfully in order for the next part of the pipeline to be added. This prevents any new additions to the project from breaking anything that has already been completed.

Once the the entire program is assembled together, acceptance testing will be conducted. We will first conduct the acceptance test ourselves to make sure all the requirements for the program have been met. Then during the usability test, acceptance testing will be carried out together, so the user can also confirm everything is in order.

3.4 Non-Functional Testing

Performance testing will be carried out by comparing our new program to the old program/scripts. We will be running our program with the same configurations as the old ones. These results will show how our program performs compared to the old way Principal was doing things.

As our project will be used by the data science team, tests for usability will be extremely important. Once our prototypes are finished, we will hand our prototypes to Principal for user testing. We will observe in person how our program works under normal working conditions. Based on the results we have obtained from the users, changes will be made accordingly. This phase of testing will be done around the last phase of our project.

Compatibility testing will be done during the usability testing. Since the computers that will be primarily using our program are at Principal, the only way to conduct compatibility testing will be during the actual usability tests.

3.5 Model and Simulation

```
In [4]: import time

        # benchmarking runtimes
        factor_list = [Factor("prev_1m_ret", 98, 100), Factor("prev_3m_ret", 98, 100),
                       Factor("prev_6m_ret", 98, 100), Factor("prev_9m_ret", 98, 100),
                       Factor("prev_12m_ret", 98, 100), Factor("prev_ret12m_l1m", 98, 100),
                       Factor("p_volume_1w", 98, 100), Factor("p_volume_1m", 98, 100),
                       Factor("sales_g", 98, 100), Factor("earning_risk", 0, 2)]
        explanatory_factors = ["us3mo", "mcap", "gross_mgn"]
        predicted_factors = ["div_yid", "prev_12m_ret"]
        start_time = 19980101
        end_time = 20000101
        period = 'W'
        identifier = "ticker"

        start = time.time()
        test = create_factor_portfolios(factor_list, start_time, end_time, explanatory_factors, predicted_factors, period, identifier)
        end = time.time()

        print("%.2f seconds" % (end-start))

        5.07 seconds
```

Figure 2: Factor Portfolio Aggregation Efficiency Test

The above picture is sample of one of our completed parts, the data retrieving aspect of our pipeline. Here we are querying stock level data with specific factors. Here we are testing how fast it can retrieve our data. We have made improvements over the semester in order to speed up getting our data from our database.

```
In [5]: test[0].display()
        test[1].display()

        # directory must already exist
        #test[0].export_csv("./exports")
        #test[1].import_csv("./exports/prev_1m_ret_-_98_100_19980101_20000101_W.csv")

        test[0].display()
        test[1].display()
```

prev_1m_ret

	div_yid	gross_mgn	mcap	period_yyyymmdd	prev_12m_ret	\
0	0.000000	34.965637	18510.50000	19980102	99.724520	
1	0.000000	13.365541	8129.27500	19980102	212.894730	
2	0.869565	9.913704	766.55554	19980102	-13.148147	
3	0.000000	27.969145	26099.65800	19980102	19.675087	
4	1.136767	20.967411	6925.19870	19980102	111.134125	
5	0.000000	11.899323	12145.32800	19980102	80.337080	
6	0.000000	32.645294	821.49100	19980102	NaN	
7	0.000000	24.641940	1294.33410	19980102	NaN	
8	2.494401	36.087418	1679.82200	19980102	54.062020	
9	0.000000	27.248787	3110.67900	19980102	NaN	
10	0.000000	17.454561	6114.09770	19980102	33.740830	
11	0.000000	73.839960	2183.06000	19980102	28.968262	
12	0.000000	56.768560	3550.29760	19980102	-5.750799	
13	0.165975	31.419807	2338.33280	19980102	36.784386	
14	0.000000	30.619450	2403.29370	19980102	147.619050	
15	2.000000	7.779056	4833.75630	19980102	42.627823	
16	0.533333	39.015804	5425.06540	19980102	44.557476	

Figure 3: Partial Factor Portfolio Displayed

This is the same factor portfolio made above, except here we are simply displaying the data. Both of these simulations were ran in Jupyter Notebook.

3.6 Implementation Issues and Challenges

There were several challenges we faced during the beginning phases of our project. One of the early challenges we faced was with the project scope itself. In the beginning of the project, one of our team's main task was deciding what to focus on. To accomplish this, our team held interviews with various employees of Principal over the course of three weeks. The employees we have interviewed were from different teams in Principal. This caused the problem of how each employee saw the problem, and each had a different opinion of what to do. Sorting out this information we have obtained from the interview took a while, and it also delayed the start of the actual project. In the end we have decided to concentrate our scope on the data scientist team and focus on one of their many projects, the DRP 2.0.

The second challenge we have faced was the understanding of the DRP 2.0 itself. Our entire team consists of software and computer engineers. None of us had a background in either data science or finance. Understanding the DRP 2.0 took a good chunk of of our time too. There were many concepts we were not familiar with, and the research phase to help with our understanding of DRP 2.0 took longer than we would've liked it to be. Thankfully, through the continuous weekly meetings with Principal, our understanding of the DRP 2.0 grew and we managed to have a good understanding of it currently.

An implementation issue that we had and are still having is regarding with the module design of our project. As mentioned previously, many modules of the DRP 2.0 are still not fully defined yet. Due to this, our team has to make educated guesses based on our previous knowledge and with meeting with our clients, on how to define our interfaces. Creating an improper interface for one module could cause problems later down the pipeline, and our team is being careful so this won't happen. Due to this issue, we are having trouble implementing some of our parts, and also makes planning some of our future design a little hard for some interfaces.

4 Closing Material

4.1 Conclusion

The DRP 2.0 is an application for developing and testing predictive models for equity data which can be used by Principal Financial to make informed investment decisions. It is based around a core set of functions for aggregating stock data into portfolios, making and testing predictions, and combining factors into new signals. By designing these functions as a modular, intuitive, and flexible library for either Python or R, we enable efficient research amongst data scientists at Principal Financial and student groups who they work with. Additionally, this pipeline unifies the common processes used by several analytical dashboards used within the Global Investments Department, making them more easily maintainable and protected against errors.

Our technical progress on the pipeline thus far includes querying our Postgres database for stock information and aggregation to factor portfolios. This represents two of the ten steps in our core pipeline, but is quite significant considering the extensive research we did at the beginning of our project. Furthermore, we have nearly finished setting up a continuous integration framework which we will use for regression testing through the remainder of development.

4.2 References

We currently have no references for this design document.