Asset Management

FINAL REPORT

Team 13
January - December 2018
Principal Global Investors - Client
Chinmay Hegde – Faculty Advisor

Carter Scheve — *Communications Lead*
Nathan Hanson — *Project Progress Tracker/Manager*
Caleb Utesch — *Meeting Scribe*
Jack Murphy — *Research Analyst*
Samuel Howard — *Lead Engineer*
Alex Mortimer — *Project Manager*

sddec18-13@iastate.edu
http://sddec18-13.sd.ece.iastate.edu/

Published: 12/3/2018, Version 1.0

Table of Contents

List of Figures

List of Definitions

PGI: Principal Global Investors

BK_P: Book to Price - used to help demonstrate value

X12M_Ret: Investment’s Twelve-Month Return - used to demonstrate momentum

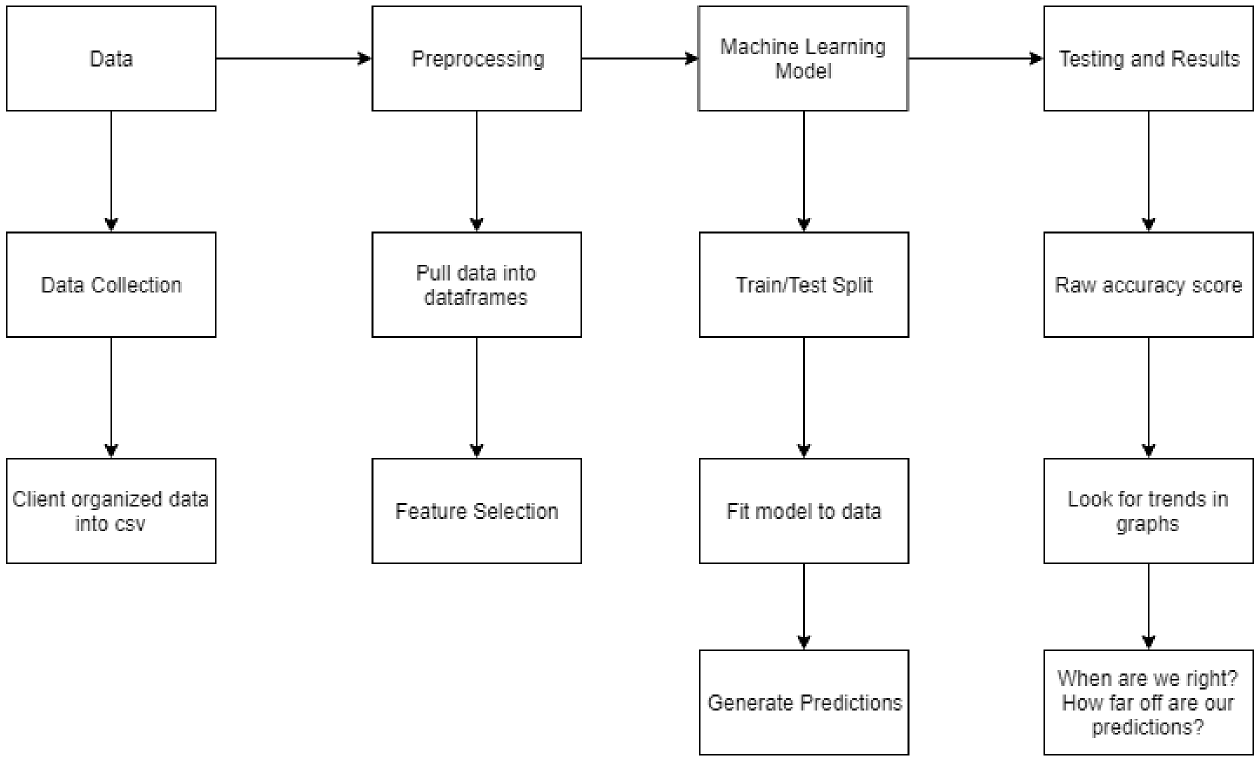
Predictor: The property or value being predicted

Feature: a descriptive property or value of an object, not the predictor

1 Project Design

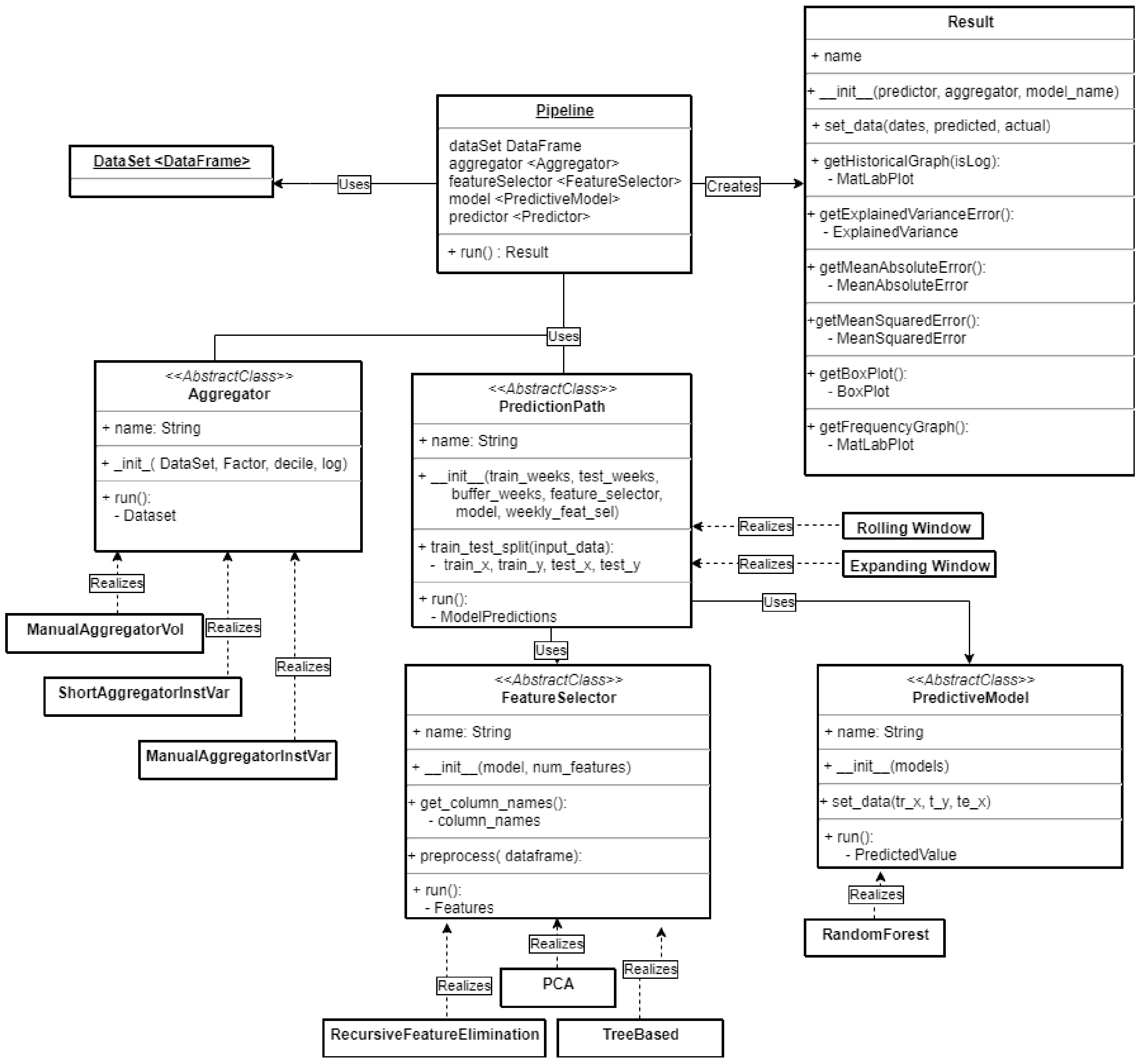
Investment analysis at Principal Financial Group currently relies on human calculation, using a variety of models and inputs. These statistical models are proven and effective, although the dependence upon human-given inputs and calculations is both inefficient and unreliable. Various steps of the statistical analysis process can be automated through software. This would remove most of the potential for human error, expedite the decision making process, and reduce overhead costs by making accurate statistical modeling and prediction more accessible.

Our solution makes use of our background in computational sciences to implement a software approach to multi-factor statistical analysis. We created a system which aids in the creation and management of an investment portfolio based upon well-defined statistical models and machine learning algorithms, which consistently outperforms the market. Such a system not only increase profits for portfolio owners, but it would also reduce risk by eliminating erroneous human action and increasing decision-making speed in a volatile stock market.



Our final software solution consists of a pipeline meant for easy integration and extensibility. When fed stock level data indexed on date, the data is first preprocessed and aggregated to ensure smooth integration with later features. The data then undergoes feature selection, which selects and modifies features about the data to better enable prediction. A predictive model then takes this data, and attempts to predict some feature about the data. A prediction path handles the specifics on which data is used to teach the predictive models, and which the model should test on. The results are then collected and analyzed to advise investment decisions. All components are encapsulated in interfaces enabling future developers to modularly implement new versions of each component.

2 Implementation Details


Result
+ name
+ __init__(predictor, aggregator, model_name)
+ set_data(dates, predicted, actual)
+ getHistoricalGraph(isLog): - MatLabPlot
+ getExplainedVarianceError(): - ExplainedVariance
+ getMeanAbsoluteError(): - MeanAbsoluteError
+getMeanSquaredError(): - MeanSquaredError
+ getBoxPlot(): - BoxPlot
+ getFrequencyGraph(): - MatLabPlot
Pipeline
dataSet DataFrame
aggregator <Aggregator>
featureSelector <FeatureSelector>
model <PredictiveModel>
predictor <Predictor>
+ run() : Result
DataSet <DataFrame>
Uses
Creates
Uses
<<AbstractClass>> Aggregator
+ name: String
+ _init_(DataSet, Factor, decile, log)
+ run(): - Dataset
<<AbstractClass>> PredictionPath
+ name: String
+ __init__(train_weeks, test_weeks, buffer_weeks, feature_selector, model, weekly_feat_sel)
+ train_test_split(input_data): - train_x, train_y, test_x, test_y
+ run(): - ModelPredictions
Realizes
Rolling Window
Realizes
Expanding Window
Uses
Realizes
ManualAggregatorVol
Realizes
ShortAggregatorInstVar
Realizes
ManualAggregatorInstVar
Uses
<<AbstractClass>> FeatureSelector
+ name: String
+ __init__(model, num_features)
+ get_column_names(): - column_names
+ preprocess(dataframe):
+ run(): - Features
<<AbstractClass>> PredictiveModel
+ name: String
+ __init__(models)
+ set_data(tr_x, t_y, te_x)
+ run(): - PredictedValue
Realizes
RandomForest
Realizes
PCA
Realizes
Realizes
RecursiveFeatureElimination
TreeBased

2.1 Prediction Pipeline

The prediction pipeline is essentially implemented as an ordered list of pipeline components. It holds references to different concrete implementations of pipeline components and manages input and output data from each, passing data along the chain in order.

2.1.1 Pipeline

The Pipeline class is an abstract class to represent the pipeline and flow of our final deliverable. Key components to this class include the predictive model, data aggregator, feature selector, and prediction path. The final Pipeline Result is collected through the pipeline.

2.1.2 Pipeline Component

The Pipeline Component is our highest level superclass, from which several of the following classes inherit logic. It encapsulates the fundamental idea of the Pipeline, requiring that each component within has a simple *run()* method, taking no arguments, which performs the task of that component. This makes it simple to abstract the logic out of the Pipeline class and simply specify which concrete class should perform each task.

2.1.3 Aggregator

The aggregator class is responsible for taking in the stock level data indexed by date, and returning factor level features of a certain decile organized by date with respect to a given factor. Additionally, this class handles any computations necessary to produce the predictor.

2.1.4 Feature Selector

The feature selector is the abstract component for the implementation of the various feature selection techniques the team developed throughout the year. It is used to preprocess the data set and format a subset of selected features for machine learning models to work with.

2.1.5 Predictive Model

The Predictive Model superclass subsumes much of the logic involved with a machine learning model, and also allows for simple subclassing by other classes. This handles the logic of setting the testing and training data for models, as well as limiting the available packages to use to a subset of scikit-learn libraries commonly utilized in machine learning applications.

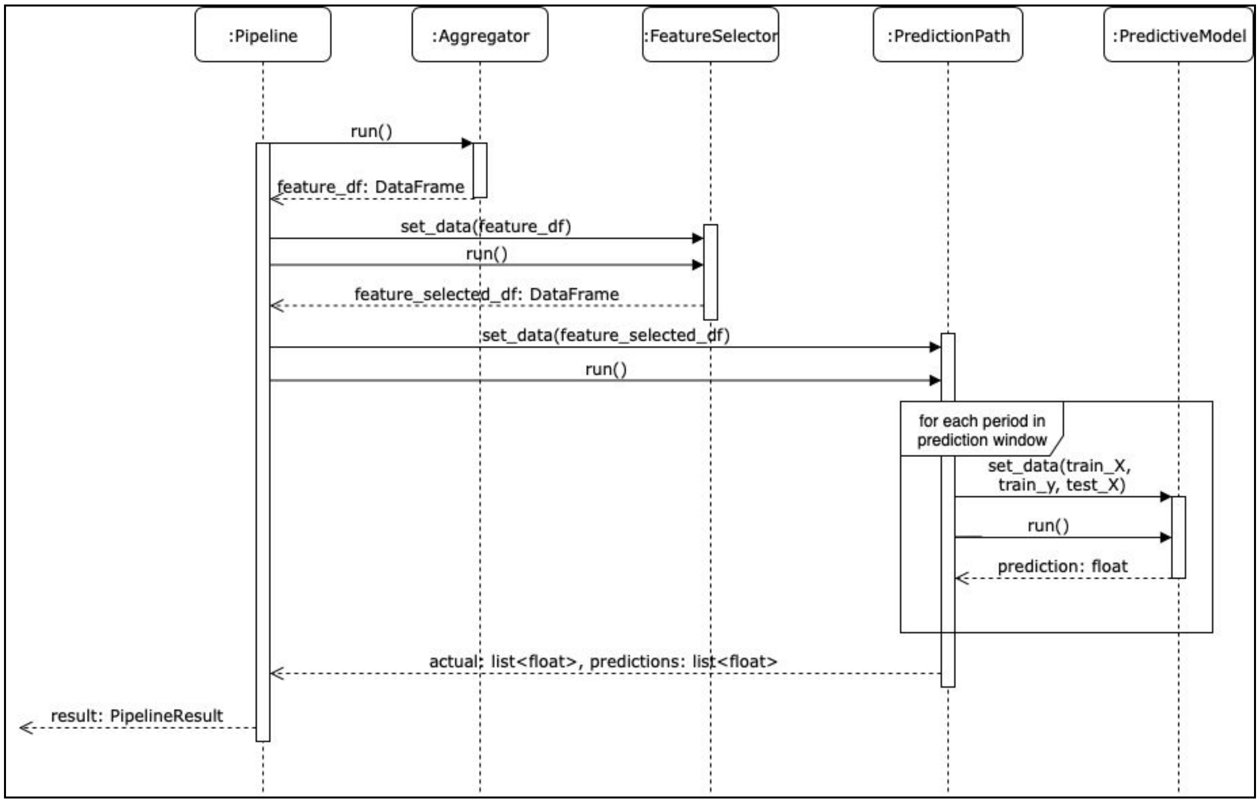
2.1.6 Prediction Path

The prediction path class is responsible for the main control flow of the prediction process. It controls which weeks are used to train the model, which weeks to test, how those weeks are selected, and how often feature selection is used.

2.1.7 Pipeline Result

The result class holds the prediction results and the actual values. It also contains functionality to perform statistical and graphical analysis of the data provided, with an emphasis on human-readable formats and results.

2.2 Execution Sequence


:Pipeline
:Aggregator
:FeatureSelector
:PredictionPath
:PredictiveModel
run()
feature_df: DataFrame
set_data(feature_df)
run()
feature_selected_df: DataFrame
set_data(feature_selected_df)
run()
for each period in prediction window
set_data(train_X, train_y, test_X)
run()
prediction: float
actual: list<float>, predictions: list<float>
result: PipelineResult

Sequence Diagram: Describes a general use case with the sequence of pipeline components. PipelineResult is formatted and returned.

3 Testing Procedure and Results

Testing is an important part of any software. This makes sure that the software delivered fulfills requirements and works with uses cases described in a proposal. Our software is an outlier in this fact. Testing is still important, but a standardized environment and processes was not compatible with our software as much as we would've liked. Some of our project dealt with data research, analysis, and modeling. This was more of a science, where documentation and findings was the requirement and not so much the code or end software. The pipeline for our project was a more formal software deliverable and testing was more feasible.

Because of the aforementioned status and nature of our project, we decided that our testing was going to be informal, but still achieve a set of goals. We didn't have enough time to implement a more formal testing process. This is because of our expertise in this area and our deliverables taking priority, which testing and testing results were not a part of. The first goal we wanted to accomplish was to make sure pipeline components work in an isolated situation. This being that they can take the specified inputs and give back outputs according to our pipeline standards. The second being to make sure all pipeline components work within the standardized pipeline (i.e. integration in the pipeline).

3.1 Testing Environment

Our testing environment, since it was an informal process, used the same environment as our development. This being our AWS EC2 instance loaned to us by our client. Hosted on that instance, we used Jupyter notebooks with python to run our testing process.

3.2 Regression Testing

Regression testing, in our project, was used to a certain extent to achieve the first goal stated in this section: make sure pipeline components work in an isolated situation. Since our process was informal, this part of testing was fully defined as regression testing, but the implementation of our process was within that process of testing. Our specific implementation of this process was as follows. First, we created a python environment (main script) and setup imports that we would need to run all of the components. From there, we standardized data to input into these components. Then, for each component, one at a time, we as many different viable use cases of data input and parameters into the component. We then analyzed the output data and tested it against expected output to see any discrepancies and errors. If there was errors, the process was to fix these errors iteratively (i.e. test, then fix, test, fix, etc.)

3.2 Integration Testing

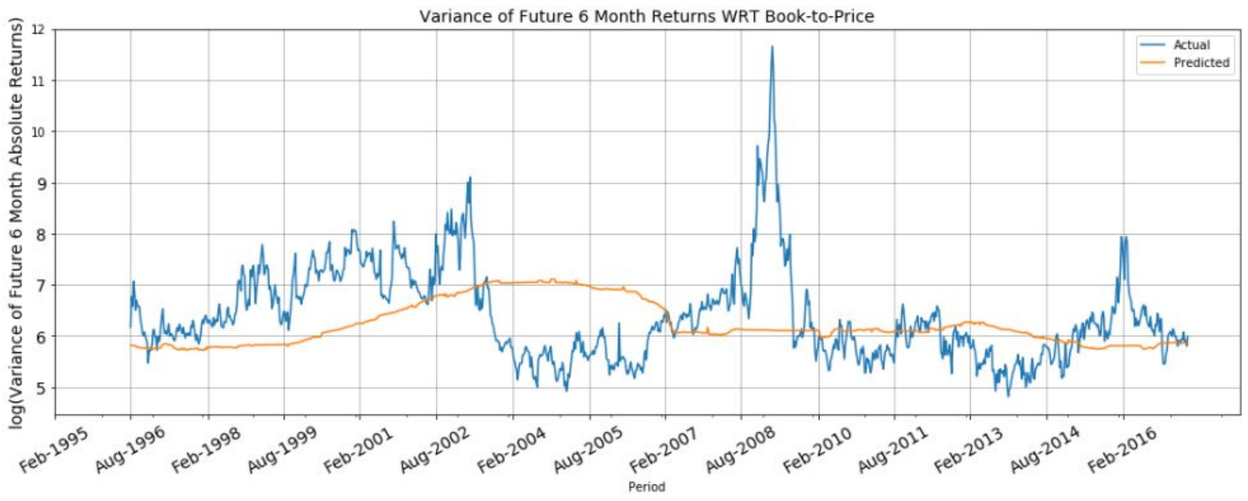
Our integration testing was used to achieve our second goal: make sure all pipeline components work within the standardized pipeline. We only used integration testing in this manner. The software environment was very similar to our regression testing process. We setup a main method with standardized data input ready and all necessary imports. The specific process was to setup an example pipeline. This followed our informal use case and standardized pipeline setup. From there, the same iterative process of testing and fixing would occur. Once a stable pipeline was in place, each concrete component, one at a time, would be replaced with a new one. From there, the same iterative process would be used to fix any errors and test integration. This was done until all components were tested and their integration success was verified.

3.3 Process Documentation

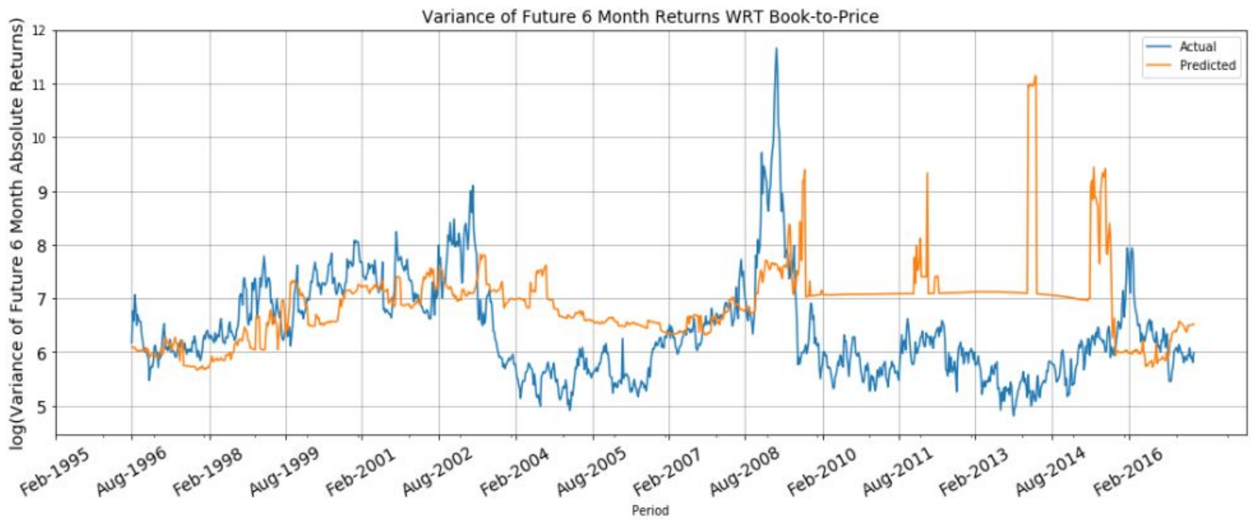
Throughout the project, documentation has been generated about our progress and our overall design process. There was an emphasis on this to accompany our technical work as this is a deliverable to our client. This documentation has taken the form of weekly updates for our client. This was decided on as the best way to go about this as we were also able to get feedback and change our process or direction if need be. Our client has an account on Dropbox and we uploaded our weekly presentation, findings, and code to the specified folders for our client.

3.4 Historical Analysis

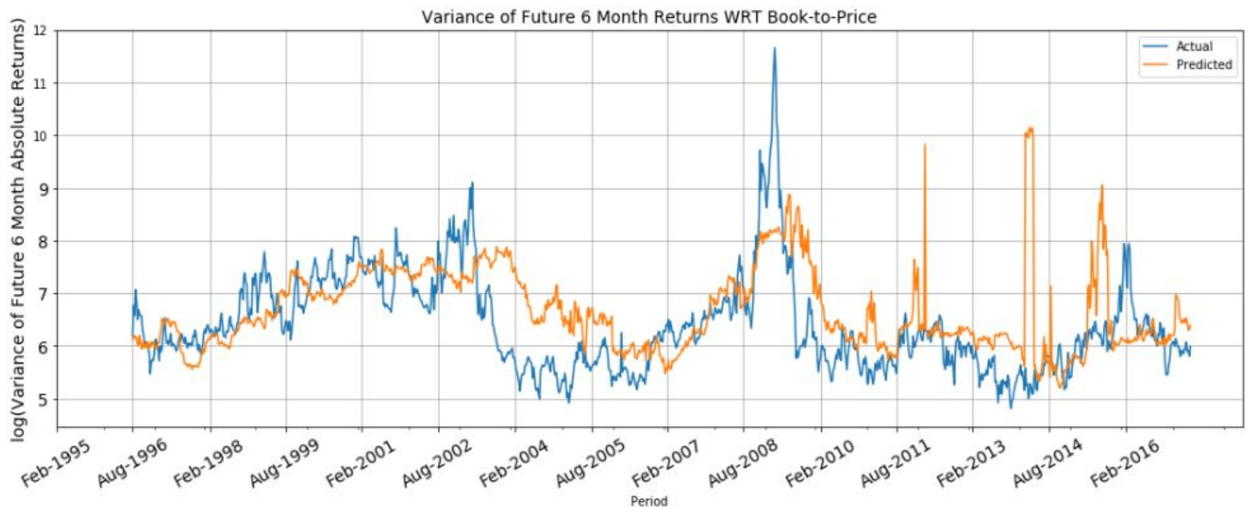
Our historical analysis was brief, but useful preliminary knowledge about how our modeled factors perform over time. We wanted to test the models on one factor, with one predictor, to have control in our process. This also gave us more interpretable results when differentiating between the success of different models. There are graphs shown below, demonstrating our models historical performance. All of the graphs shown have the same inputs. The factor level feature data has not been altered with feature selection or anything else. The predictor, in blue, is the 6 month cross-sectional variance of decile 1 scaled through a log function.



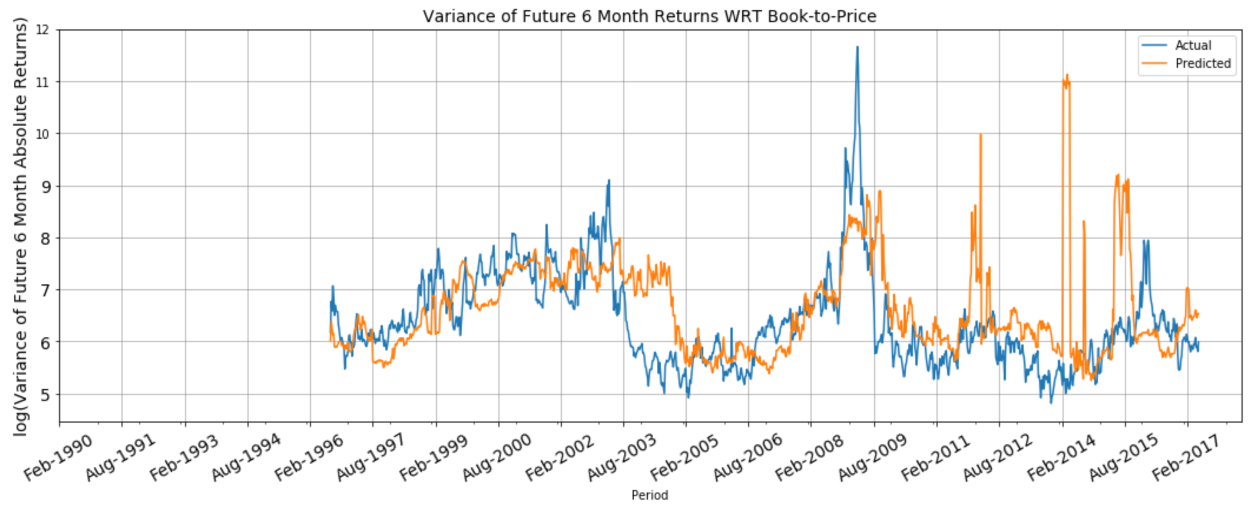
Gradient boosting model with loss calculation of least absolute deviation with a learning rate 0.001.



Gradient boosting model with loss calculation of least squares and a learning rate of 0.01



Random forest model with number of estimators set to 100



Extra trees model with number of estimators set to 100

4 Related Products and Literature

Investment analysis is a task that has been taken on by many intelligent minds over the course of time. Every investor that has worked in the stock market has tried to find the pattern of growth and success to gain an advantage in future deals. From what we have learned from research on the topic, other companies in the industry have software systems to help predict the behavior of stocks (Robert) but, like Principal, still rely on human analysis for their larger-scale decisions. Our product is different from projects like these in the sense that ours analyzes the success and overall performance of individual factors of investments, instead of an entire investment's overall success. In addition to the individual models, we will deliver analyses for each model, describing its effectiveness in the market along with its advantages and shortcomings. Much of the knowledge we have gained has been through tutorial websites, such as kaggle.com (Sehgal) or datacamp.com. These have been effective in enhancing our comprehension of machine learning because they allow for practice with realistic problems. This research enabled us to eliminate certain machine learning models that proved to not be feasible for our project.

Our research has resulted in a more targeted approach to data analysis. For example, we chose to implement a custom train-test split method, including a buffer between the end of the data used for algorithm training and the beginning of the data used for testing the trained algorithm. Through research and advice from our client, it was clear the continuous chronological nature of the data would have a significant effect on the accuracy of the trained algorithm. Our research did not include any domains pertaining to finance, stocks, investments, or other related fields. This was for two reasons; the main reason was per request from our client. They stated that it would be best for the project if we didn't research into the metrics and notions behind the data given. Their reasoning was that they wanted a fresh set of eyes that didn't have bias toward any piece of the data. The second reason was that we needed to tune our focus to other places. We didn't have any substantial experience or knowledge on the subject of data analytics and machine learning. We only had a limited amount of time for research, and delving further into machine learning instead of the financial domain was going to have a bigger impact on the success of our project.

5 Appendices

Appendix I. Operations Manual

In order to take advantage of the system we have developed, a user would have to develop concrete implementations of each of the Pipeline Components explained in section 2.1. This would all be done in one main python file, which would also contain the necessary imports for the data loading process, initializing the components, and displaying or storing outputs. Those components must then be passed into a Pipeline object for use, then call the *run()* method on that pipeline. The results come back in the form of a Result class, which is used to access several graphing functions, accuracy results, or the raw predicted and actual values.

The list of steps to setup and demo our system are as follows:

1. Download and install the most recent release of the Anaconda distribution
2. Clone the repository that contains the code for the system
3. Implement concrete classes that extend each pipeline component superclass.
4. Create a main.py file to instantiate the components and call the .run() method on the Pipeline object
5. Open a shell window
6. Navigate to the directory which contains the source code .py files from the repository
7. Run the command "python main.py"

Appendix II. Alternative/Unused Versions of Design

Alternative versions of our product were developed prior to our client adjusting the scope and specifications of our end goal. The development in the first semester of working on this project involved a focus on using machine learning and feature selection to predict specific factors of the data set. Developing accurate scores for predictions of these factors was where our team oriented most of the code. We also developed an extensive data library for ease of access to correctly aggregated and formatted data with various machine learning models.

A version of the final product that was considered before the end goals were established was development of a web application. This would mostly be a dashboard for our client to be able to directly interface with our models. It would provide appropriate graphs, much like our current end product. This was not able to be fully fleshed out because of the time constraints for finishing our development of the machine learning models.

Appendix III. Miscellaneous/Other Considerations