



**WPI**



**ANGELO,  
GORDON  
& CO.**

**DECEMBER 16, 2017**

A Major Qualifying Project Report  
Submitted to the Faculty of  
Worcester Polytechnic Institute  
In partial fulfillment of the requirements for the  
Degree of Bachelor of Science

Submitted to:  
Professor Kevin Sweeney  
Professor Michael J. Ciaraldi  
Professor Reinhold Ludwig

**PROJECT ID: KMS-ABGJ**

# **WALL STREET: ENGINEERING INVESTMENT PROFIT & LOSS REPORTING PIPELINE**

**AUTHORS: JAKE AKI & DAVID DEISADZE**  
**SPONSORED BY ANGELO, GORDON & COSPONSORED BY ANGELO, GORDON &  
CO.**



**WPI**



**ANGELO,  
GORDON  
& CO.**

---

## **ABSTRACT**

---

We sought to improve an existing multi-language financial reporting tool by unifying the pipeline into a single native Python environment and by providing a more intuitive user interface and enhanced documentation. We had to reverse engineer the legacy codebase and integrate a new dynamic engine for investor relations to reach our goal. Project management practices such as Scrum and Kanban as well as adhering to a well-maintained object oriented methodology were the core of our teamwork and productivity. Ultimately, the final product was a professional graphical interface in addition to the re-engineered pipeline, which unified the whole system.



---

## ACKNOWLEDGEMENTS

---

First and foremost, we would like to thank the whole Angelo Gordon team for hosting us and welcoming us to their unique culture. Specifically, we would like to thank Scott Burton for facilitating and guiding us through this project and the whole team for helping us on countless occasions. This project would not have gone as smooth if it were not for Mr. Burton's and the whole risk team's quick replies and determination to help. We would also like to thank our advisors who made time to visit from Boston and Worcester and gave us their valuable time and advice.

This project would not have been possible if it was not for all the open source forums and tools on the web. We would like to thank the whole Stackoverflow community for answering numerous questions and guiding us through bug resolution. We would like to thank the OpenPyXL, PyQt, and the Python distribution team, whose technology and API's we heavily relied on. Without their awesome documentation and tutorials, we would have had a much tougher time with our project.

We would also like to thank the WPI Interdisciplinary and Global Studies Division for handling all the logistics for the project center. We had a tremendous time in Manhattan and we gained valuable experiences through this MQP program.

Finally, there are countless other teams and individuals we could not list and we would like to extend our deepest thanks for their contributions.

Thank You,

David Deisadze and Jake Aki



---

## EXECUTIVE SUMMARY

---

We worked with an alternative investment fund Angelo, Gordon, & Co for this project. The term “alternative investment fund” means that the investments that Angelo, Gordon, & Co specialize in are not directly correlated with typical market indicators like the S&P 500. This is because Angelo, Gordon, & Co deals with long term investments like distressed assets, real estate, venture capital, and private equity (Angelo, Gordon, & Co, 2018).

Investment funds use a variety of data analytics, reporting tools, and financial indicators to track the success or failure of their financial strategies. One ubiquitous report is the profit and loss (P&L) statement. Angelo, Gordon, & Co had software for generating P&L reports, but the workflow could be convoluted, and the interface was unintuitive. This MQP team was brought on board to simplify the workflow, improve the interface, and provide documentation.

For the organizational aspects of the project, we used a combination of Scrum and Kanban project management styles to stay on top of workflow efficiency. For the technical aspects of the project, it was determined by the sponsor that Python should be the native language for the project. Therefore, we relied heavily on the open source community for a variety of modules and packages, like OpenPyXL and PyQt5, to replicate some of the functionality that was originally done in C# in the legacy pipeline.

As development progressed, we found that we had time to develop a graphical user interface layer on top of the report generation backend. So, the remaining time of the project was spent developing this interface and adding different features at the request of the sponsor, which culminated with the addition of an installer and executable version of the GUI.

Ultimately, we left the sponsor with a unified Python pipeline, streamlined workflow, a modern user interface, installer, documentation, and a design that should make supporting new templates easier in the future.



# Table of Contents

ABSTRACT ..... i

ACKNOWLEDGEMENTS .....ii

EXECUTIVE SUMMARY.....iii

Table of Figures .....vii

1 INTRODUCTION ..... 1

    1.1 Winners and losers report ..... 1

        1.1.1 Why it exists ..... 1

        1.1.2 Origin of the report..... 2

    1.2 Drawbacks to the winners and losers report ..... 2

        1.2.1 Difficulty of use ..... 2

        1.2.2 Complexity of pipeline..... 3

        1.2.3 Lack of extensibility ..... 4

        1.2.4 Problem summary ..... 4

    1.3 Our proposed solution ..... 4

2 BACKGROUND ..... 6

    2.1 Different reporting needs..... 6

        2.1.1 Closed funds..... 6

        2.1.2 Open funds ..... 7

    2.2 Scripting languages and why Python ..... 7

        2.2.1 Other possible languages ..... 7

        2.2.2 Keeping a native Python pipeline ..... 8

        2.2.3 Strengths of Python ..... 9

    2.3 OpenPyXL..... 9

        2.3.1 What it can do..... 9

        2.3.2 What it cannot do..... 11

    2.4 PyQt5 ..... 12

        2.4.1 Model, view, controller pattern ..... 12

        2.4.2 How the designer works ..... 13



- 3 METHODOLOGY ..... 15
  - 3.1 Project management ..... 15
    - 3.1.1 Kanban ..... 15
    - 3.1.2 Scrum ..... 16
    - 3.1.3 Source control ..... 16
  - 3.2 Development ..... 19
    - 3.2.1 Object oriented analysis & design ..... 19
    - 3.2.2 Design patterns ..... 19
  - 3.3 Dynamic report generation ..... 21
    - 3.3.1 Open funds vs closed funds ..... 21
    - 3.3.2 Multiple templates ..... 21
    - 3.3.3 Dynamic Column Strategies ..... 23
  - 3.4 GUI development ..... 25
  - 3.5 Documentation ..... 25
    - 3.5.1 Writing the README ..... 25
    - 3.5.2 Different markup languages ..... 26
    - 3.5.3 Incorporating with the GUI ..... 27
- 4 RESULTS ..... 30
  - 4.1 Testing ..... 30
    - 4.1.1 Unit testing ..... 30
    - 4.1.2 Regression testing ..... 31
  - 4.2 Product ..... 32
    - 4.2.1 Overview ..... 32
    - 4.2.2 OpenPyXL tools ..... 33
    - 4.2.3 Logging tools ..... 34
  - 4.3 GUI ..... 35
    - 4.3.1 GUI implementation ..... 35
    - 4.3.2 Generating executable for GUI ..... 41
- 5 CONCLUSIONS ..... 42



WPI



ANGELO,  
GORDON  
& CO.

5.1 What went wrong ..... 42

    5.1.1 Singleton design pattern ..... 42

    5.1.2 Hardcoded template references ..... 43

    5.1.3 Levels of encapsulation ..... 43

    5.1.4 PEP 8 code standard consistency ..... 44

5.2 What went right ..... 44

    5.2.1 Strategy design pattern..... 45

    5.2.2 Graphical front end ..... 45

    5.2.3 “Scrumban” ..... 45

5.3 Takeaways ..... 46

5.4 Recommendations ..... 46

APPENDIX A: STRATEGY ENTRY ..... 47

APPENDIX B: COLUMN STRATEGY ..... 51

APPENDIX C: START MAIN REPORT ..... 57

APPENDIX D: GRAPHICAL USER INTERFACE (GUI) ..... 59

APPENDIX E: UNIT AND REGRESSION TESTING ..... 67

Works Cited..... 74

## Table of Figures

---

Figure 1: Fund to strategy hierarchy.....	1
Figure 2 Legacy pipeline .....	3
Figure 3 Project Goals.....	5
Figure 4 OpenPyXL workflow.....	10
Figure 5 Qt Designer .....	13
Figure 6 Kanban board.....	15
Figure 7 GIT history as of Nov, 20th.....	18
Figure 8 Strategy design pattern .....	20
Figure 9 Closed vs open fund template.....	21
Figure 10 Closed fund template .....	22
Figure 11 Dynamic column mapping.....	23
Figure 12 Column strategy UML diagram.....	24
Figure 13 README in Markdown .....	26
Figure 14 Formatted Markdown .....	27
Figure 15 Pre-CSS README.....	28
Figure 16 Solarized README .....	29
Figure 17 Unit test structure .....	30
Figure 18 Regression testing workflow.....	31
Figure 19 Reporting tool overall structure .....	32
Figure 20 Insert rows workflow.....	33
Figure 21 Main GUI form view.....	35
Figure 22 Selecting an input file .....	36
Figure 23 Automatic file name parsing.....	37
Figure 24 Output directory link .....	38
Figure 25 Console log .....	38
Figure 26 Settings tab .....	39
Figure 27 Testing tab .....	40
Figure 28 Running the unit tests .....	41



---

# 1 INTRODUCTION

---

## 1.1 Winners and losers report

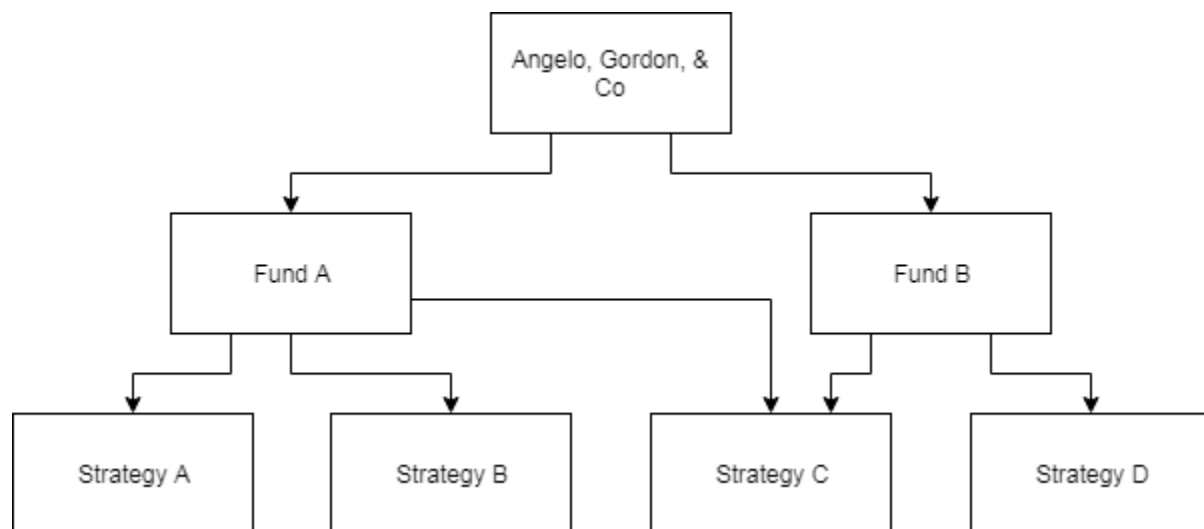
---

### 1.1.1 Why it exists

Before diving into technical details, it is important to understand the context of the project. A large number of investment funds generate multiple financial reports, some of which are circulated internally. However, other external reports, after heavy vetting, are sent out to clients. These reports are not just internal diagnostic tools, but also external marketing tools to show the fund's success. A common kind of report that all organizations publish is a profit and loss statement (or just P&L), which is typically a quarterly report. Investopedia (2017) describes this concept as:

... a financial statement that summarizes the revenues, costs and expenses incurred during a specific period ... These records provide information about a company's ability – or lack thereof – to generate profit by increasing revenue, reducing costs, or both.

As a hedge fund, Angelo, Gordon, & Co maintains several large investments called funds, which are composed of individual investments, which are lumped together as strategies.



**FIGURE 1: FUND TO STRATEGY HIERARCHY**



As you can see in the above figure, funds are comprised of multiple strategies, but an individual strategy might belong to multiple funds. Furthermore, a strategy represents a specific venture (venture refers to a position taken with an asset; this can be loans, stocks, or assets) with an investment, but that investment might be involved in multiple strategies. With this level of complexity, simple tools for evaluating a fund's success become increasingly valuable.

### 1.1.2 Origin of the report

Our sponsor, Scott Burton, developed an internal report for closed funds, which became informally known as the Winners & Losers report and will be referred to in the rest of this document as the W&L report. The W&L report is simply a specific kind of P&L statement. It breaks down the highest gross profit strategies within a fund along with the worst gross loss strategies. It was originally just an Excel spreadsheet with some clever logic and VLOOKUPS, which was populated with data by hand. It gained momentum as a valuable diagnostic tool on the success of a fund and began to be distributed to investors.

## 1.2 Drawbacks to the winners and losers report

---

The problem that we were given in the beginning soon became a complex system which needed further analysis to solve. Midway through, we as a team decided to add additional features to solve the problem. These events will be thoroughly discussed in the following sections.

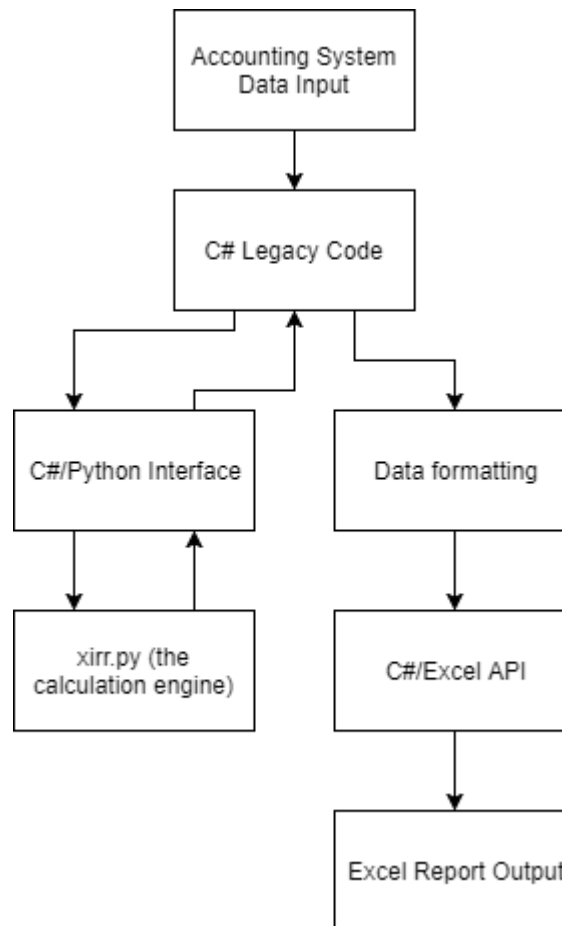
### 1.2.1 Difficulty of use

As described in the previous section, this report is valuable. However, building the report by hand was cumbersome and difficult. The fact that human beings were involved in directly manipulating and formatting the data allowed multiple points of failure to enter the system. Therefore, more controls and redundancies were needed to catch any mistakes.

To replace the human element, a consultant was brought on to build a report generation tool that would automate this formatting and data population step. This initial codebase was written in C#, and it was soon realized that it was an error-prone process that was difficult to use and broke frequently. Our sponsor desired a more streamlined and robust process.

### 1.2.2 Complexity of pipeline

Another issue with the tool developed by the consultant was the complexity of the pipeline. Prior to the construction of the new tool, there existed a calculation engine written in Python, which took in accounting data and performed certain computations to generate financial metadata like internal rate of return (IRR). Then, the old automated tool was built on top of this Python engine, but that old automated tool was written in C#. This led to an awkward interface involving batch calls to Python, which is shown in the figure below.



**FIGURE 2 LEGACY PIPELINE**

It is worth noting that C# was not a bad idea for this approach. There is an entire suite of Microsoft Office tools for C# that allow for interoperability between C# and Excel, which allows for a much more comfortable layer of abstraction between the data

wrangling and formatting and actually manipulating the Excel template. However, Python has excellent support through Numpy (NumPy developers, 2017) and pandas (NUMFocus, 2017) for the kind of data manipulation that our sponsor was doing, so the decision was made that it would be better to transition the whole codebase to Python instead of transitioning the file prep script to C#. This file prep script used some accounting math to calculate metadata like internal rate of return. Because this file prep was proprietary to Angelo, Gordon, and had less to do with the code and more to do with their specific algorithms, we were told not to modify it. Python is also much better for quickly getting projects up and running, so it was easier to try converting to Python first and seeing if it was an adequate replacement.

### 1.2.3 Lack of extensibility

A further advantage of rebuilding the reporting tool is that it provided us with the chance to build in a framework for extensibility. Our sponsor discussed the possibility of replacing the old Python calculation engine with a new and improved script. Other ideas for future extensions were different kinds of reports and differently formatted input files. So, the codebase needed to be able to adapt and grow.

### 1.2.4 Problem summary

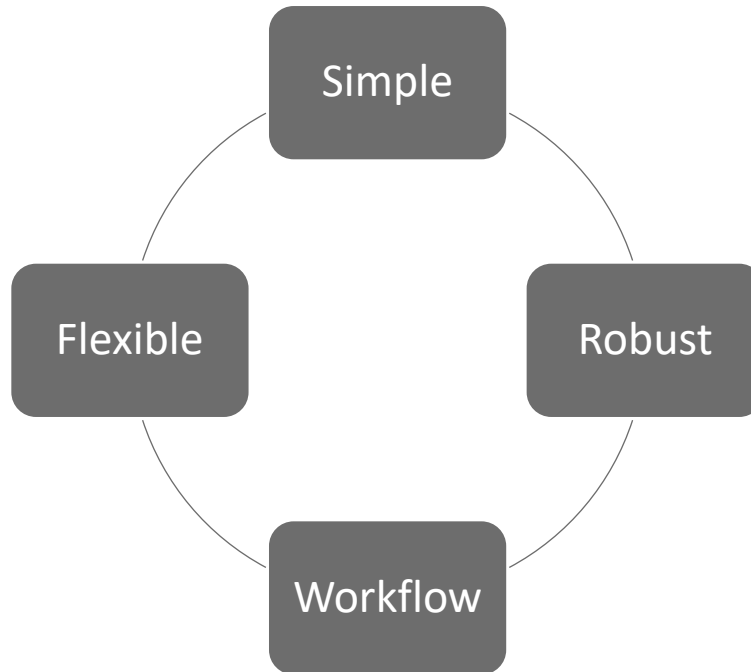
We were brought on board to replace the old report generation tool because of three major flaws:

- **Difficulty of use:** the code broke often. It was difficult to debug and had no meaningful interface.
- **Complexity of pipeline:** the use of two different languages aggravated the debugging issue and contributed unnecessary overhead.
- **Lack of extensibility:** the tool was not aging well and needed to be able to support new calculation tools, reports, and inputs.

## 1.3 Our proposed solution

---

Our project was very clearly defined. We were taking an old legacy codebase and converting it to Python to maintain a single pipeline. However, the project quickly grew from there, and it is important to understand how the goals of the project was divided into two end-goals: a code port and improved functionality for our end user as they achieve their goal. i.e., easy report generation. Fundamentally, this report generation tool is used by our sponsor Mr. Burton on a relatively infrequent basis (monthly or quarterly), but the accuracy of information is paramount. Hence, the process should be as free from human influence as possible while being simple to use. In trying to focus in on these ideas, automated and simple, we isolated some basic design goals.



**FIGURE 3 PROJECT GOALS**

- **Simple:** Mr. Burton only uses this software every few months, and it should not be a hassle to relearn it each time. It should be obvious how it is run and how to use it.
- **Robust:** We lean heavily on an Open Source library called OpenPyXL. As is always the case with such libraries, there is no guarantee it will keep working the same way. Some exposure is unavoidable, but we strived to maintain a layer of abstraction sufficient to minimize churn if there were any changes.
- **Workflow:** The previous solution was effective, but because of the disjointed workflow illustrated in Figure 2, as soon as something broke, it became a mess to debug. The goal with this single Python workflow is to make debugging both in finding errors in the code as well as wrong result data as painless as possible
- **Flexible:** Currently, Mr. Burton only supports one type of report for closed funds. However, in the future other kinds of reporting might become valuable. Since these reports have generally the same skeleton (e.g. Excel reports, data comes from tables of strategies, totaling and summation are common operations), it would be advantageous to make it as easy as possible to extend this software to support new reports.

The goals of simplicity and robustness were aimed at resolving the issue of difficulty to use. The workflow goal was intended to resolve the pipeline issues, and the flexibility was meant to improve extensibility.

---

## 2 BACKGROUND

---

### 2.1 Different reporting needs

---

#### 2.1.1 Closed funds

As mentioned in the introduction, a P&L report is an almost universally useful performance analysis tool. However, different investment strategies/funds have different metrics for success. The main example we used when trying to design our software to support various investment strategies was the distinction between a closed and an open fund.

Closed funds can be thought of as a long-term investment. Investors put their money into a box and then as far as they are concerned, it remains there for an extended period typically five to ten years or until the fund is closed. Now, the investment fund can move that money around, change strategies, and reinvest it, but the time horizon on the fund is longer, and thus it tends to be a safer and a larger return. Short of a massive systemic risk<sup>i</sup>, the money stays put until an agreed time frame passes (Boyd, 2001).

Due to this feature, an excellent metric of success for a closed fund is basis points<sup>ii</sup> of gross profits to the total size of the fund. This gives you an idea of how much profit a given strategy has generated relative to the initial investment.

---

<sup>i</sup> Kaufman in the Financial Markets journal provided an excellent definition for a systemic risk event. It is described as a “Big shock [with] direct causation (chain reaction) contagion (direct linkage among banks through interbank deposits, loans and clearings)” (Kaufman, 2000). The idea is that these chain reactions propagate throughout the entire financial market, and cause havoc. The financial crisis of 2008 is an example of such a system risk event.

<sup>ii</sup> Basis points are a frequently used measure for percentages in the investing. A basis point is equivalent to 1/100<sup>th</sup> of a percent or .01%, or .0001. The origin of this language and the reason for its pervasiveness is well-described on (Investopedia, 2017):

For example, if a financial instrument is priced at a 10% rate of interest and the rate experiences a 10% increase, it could conceivably mean that it is now  $0.10 \times (1 + 0.10) = 11\%$  OR it could also mean  $10\% + 10\% = 20\%$ . The intent of the statement is unclear. Use of basis points in this case makes the meaning obvious: if the instrument is priced at a 10% rate of interest and experiences a 100 bp move up, it is now 11%. The 20% result would occur if there was instead a move of 1,000 bps.

## 2.1.2 Open funds

An open fund is a more fluid investment. Investors are not restricted on when they can liquidate or reinvest more cash (Boyd, 2001). If a fund is doing well, investors might want to add more money, and if a fund is doing poorly, or an investor simply needs some more liquid assets, they can take money out. With an open fund, basis points relative to the total size of the fund is meaningless. An open fund could perform extremely poorly, but if a major investor pulls out, it might look like there were still decent returns because the total fund size would have shrunk.

As described before, a fund is composed of multiple financial strategies. In an open fund, the metric of success for an individual strategy is the strategy's gross profit relative to the total gross profit of the fund. Let us go through an example to clarify this. Imagine you have an open fund called WPI Investments. That fund is composed of three separate investments W, P, and I. These investments are the three different financial strategies. If the fund makes \$2 million in gross profit, but investment W only made \$3000, then its relative success in basis points is  $(\$3000/\$2000000)*100*100$  or 15 basis points. If we add the relative successes of all the strategies together, it should equal 10000 basis points or 100%.

## 2.2 Scripting languages and why Python

---

### 2.2.1 Other possible languages

The predecessor to this project was written partially in Python and C#. Our original goal was simply to convert the C# to Python, but a colleague of our sponsor asked the reasonable question of whether Python truly was the best language to convert to. As we have touched on, for matters of simplicity Python is attractive as that is already the language that the end user was working in. Logically, it made sense to use Python to extend the environment. As we worked on the project, we examined and discussed other possible languages that could have been used to achieve the same outcome.

- **C#:** This is the language we started in. C# has the advantage of being a truly object-oriented language, which makes maintaining SOLID<sup>iii</sup> design principles

---

<sup>iii</sup> SOLID is a set of five principles for object oriented programming. We loosely followed them for reference, but as Python is not a strictly object-oriented language, some of the criteria did not fit 100% of the time. But, Solid is essentially this:

- **Single responsibility:** a class should only have one reason to change so as to minimize churn. In other words, classes should only be responsible for one thing.

much easier. It is also supported by the .NET framework, which means it has better support for working with other Microsoft products namely Excel. Furthermore, when we started designing the GUI, C#'s Window Forms or Windows Presenter Foundation are in most ways better tools than Qt Core Designer or other Open Source tools.

- **Java:** Java has no relationship with the background for this project, but as a powerful, well-supported object-oriented language it feels necessary to mention it as an option. Java has an excellent Excel API and other numerical resources for the kinds of calculations that were done in Python, but it is slower to get running. Given that we had inherited code in C# and Python, it would not have been worth migrating all of it over to Java.
- **Python:** This is the language we ended up deciding on. Python is flexible, powerful, and easy to work with. The PyCharms environment used at Angelo, Gordon is a well-developed integrated developer environment (IDE), and makes installing modules, navigating and refactoring code a breeze. It also has fantastic auto-completion and syntax help.

### 2.2.2 Keeping a native Python pipeline

Keeping the pipeline consistent was one of the primary objectives for this project. Keeping the entire pipeline in Python was a major advantage since it simplifies the connections of each step in the pipeline and retains the maintainability for future teams. Prior to our arrival, the pipeline consisted of disparate codebases in C# and Python; connecting the two required adapters and patches. Maintaining these patches was difficult as well as adding new features. With the codebase being in one framework, it solved the problem of having a disparate unit testing as well as passing data between the frameworks. These separations created “hacky” clutter that was extremely difficult to understand as well as extend.

- 
- **Open/Closed:** classes should be easy to extend but not meant to be modified. i.e. you should be able to change the behavior of the code without having to open it back up and recompile it.
  - **Liskov Substitution:** If it walks like a duck, quacks like a duck, but needs batteries, it is probably not a duck. Sub-types ought to be valid substitutes for their base type. When this logic is violated, things get weird.
  - **Interface segregation:** Less applicable to Python, which has no strict interfaces, but basically no class should be contracted by an interface for methods that they do not use.
  - **Dependency inversion:** instead of directly instantiating subclasses, you should pass in abstractions of the class so that the subclass can change without breaking the hierarchy



### 2.2.3 Strengths of Python

One of the great strengths of Python is the sheer amount and diversity of open source libraries available. This is what made it so attractive to our sponsor initially. Two libraries were specifically used for the original file prep script: Numpy and Pandas.

Numpy is effectively a MATLAB package for Python. It provides an object for storing N-dimensional data, linear algebra tools, Fourier transforms, and more (NumPy developers, 2017). It is a simple but useful module with limited but powerful tools.

Pandas on the other hand has the lofty self-described goal of “...becoming the most powerful and flexible open source data analysis / manipulation tool available in any language” (NUMFocus, 2017). It is designed to work with “relational” data sets, which immediately draws allusion to a SQL-type structure, which is not an entirely unreasonable comparison. Pandas works with DataFrames, which are two dimensional arrays of data and basically provides all the sort of matrix manipulations and calculations that you might expect in a dedicated scripting language like MATLAB. It also brings with it the speed of pre-compiled C code and the ease of Python.

Given this history of leaning on strong Python libraries, we, with our sponsor’s support, looked into a variety of Python libraries for manipulating Excel files. One contender was XlsxWriter, which is a very robust and extensive library, but was discarded early on, because it only allows the creation of new Excel files; thus, it could not read in settings from a template file. Another tool that we use a little but not for the bulk of our report generation, is the win32com library, which provides a Component Object Model (or COM) layer over a Microsoft API. This allows us to effectively launch the Excel App from within our Python script. However, this is a very clunky way to do the kind of formatting that this report required, and ultimately lacked the finesse necessary.

So, we settled on a library called OpenPyXL, which is simply put a “...Python library for reading and writing Excel 2010 xlsx/xlsm/xltx/xltx files” (Clark & Gazoni, A Python library to read/write Excel 2010 xlsx/xlsm files, 2017)

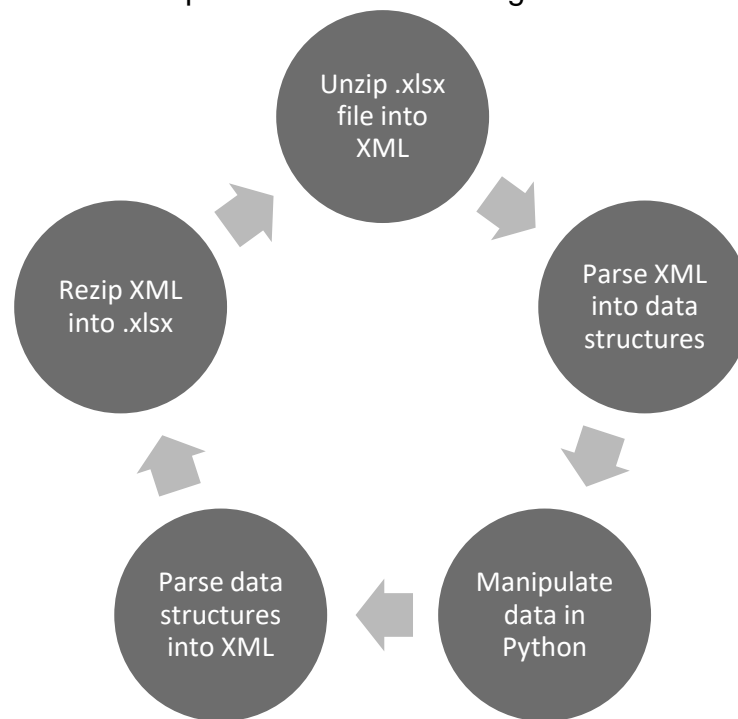
## 2.3 OpenPyXL

---

### 2.3.1 What it can do

OpenPyXL fundamentally works by unzipping and reading the raw XML files that all Excel XLSX (Excel XML based spreadsheet file format) files are composed of. It

parses all the information into what are more human-readable objects like cells, sheets, and workbooks. Next, OpenPyXL manipulates that data, not the actual Excel file, in memory until it is in a satisfactory state. OpenPyXL then reinterprets the data structures back into the XML, zips it back up, and labels it as XLSX. Assuming the data structures have not been given impossible or invalid values, the Excel document will open as intended. There are several important caveats to this general workflow.



**FIGURE 4 OPENPYXL WORKFLOW**

First, Python is already notoriously unreliable at keeping track of variable types. It uses what is colloquially referred to as “duck-typing”, which is attributed to the phrase “if it walks like a duck and quacks like a duck, then it is a duck”. This is just a clever way of describing dynamic typing, which is to say that when given a variable the Python interpreter will just treat it as whatever kind of data makes sense given the situation unless that raises an exception. A common example of this kind of exception was that occasionally the accounting data would provide the string “nan” in a numerical field like gross profit. When Python tried to read that as a float, it would raise an exception.

This issue is compounded in OpenPyXL because we are not directly manipulating the Excel worksheet but instead we are manipulating OpenPyXL’s abstraction of the worksheet. So, thanks to a combination of Python’s duck-typing and OpenPyXL’s abstraction it is entirely possible to do an entire array of impossible things

to these data structures, which will create a corrupted XLSX file when it is eventually re-zipped. OpenPyXL has made real efforts to provide significant checks on this. For example, when writing a value to a cell, OpenPyXL will check if that could be a valid input in a cell, e.g. a float, int, or a string.

Second, when we manipulate formatting (formatting such as inserting or removing rows or expanding or shrinking a table) it is possible to break the headers, which will lead to corrupting the file.

Thirdly, the code used to insert new rows uses a clever regular expression<sup>iv</sup> (regex) to dynamically update cell references. e.g. if a cell refers to I13, but we insert two rows, it will properly refer to I15. However, this means that if we have a named cell, which contains a string value that looks like capital letter plus a number, it will also auto increment that number. For example, a cell named FUND\_CAT50 would become FUND\_CAT53. This is best resolved by including underscores in any such names (e.g. FUND\_CAT\_50 would remain FUND\_CAT\_50).

### 2.3.2 What it cannot do

OpenPyXL has a few glaring issues that required us to design our own solutions. These issues all primarily arise from the fact the OpenPyXL aims to give very precise and exact control over the Excel sheet. Many operations are best utilized on the cell level as opposed to working an entire rows, columns, or tables.

First, OpenPyXL does not support inserting or deleting rows. This is initially surprising given how important an operation that is for working with spreadsheets, but makes sense historically within the module. OpenPyXL has gone through many changes in how cell references are represented, and most implementations of row

---

<sup>iv</sup> Regular expressions or regex is almost an entire coding language unto itself. They're used for very concise pattern matching. From *Master Python Regular Expressions* they have five primary functions (Lopez & Romero, 2014):

- Check if an input matches a pattern
- Look for the appearance of a pattern in a chunk of text
- Extract a specific pattern from a text
- Replace a specific pattern in text
- Split a large text into smaller pieces

Regex accomplishes this in Python by instantiating a pattern, which would be something like: look for any three character combination of decimals and uppercase letter or check if there is any occurrence of the pattern "b2C". Then this pattern can be used for any of the uses cases above.

insertion relied on these unreliable internals (Clark & Dallas, Insert row into Excel spreadsheet using openpyxl in Python, 2013).

Second, while it is possible to access named ranges in OpenPyXL, there are no built-in functions or methods that allow you to apply formatting. All OpenPyXL provides is the range of cells that the named range contains. This is further complicated by the fact that while the named range knows what cells belong to it, each cell does not know what named range it belongs to. This means as we insert cells above a named range, the named range will not correctly shift down, but the value in the range will.

Third, tables have very little support in OpenPyXL. They exist as data structures with all the correct values and options, but interacting with them can be very buggy and is easy to cause corruption.

Fourth, charts are completely unsupported. OpenPyXL cannot correctly parse them from the XML.

## 2.4 PyQt5

---

### 2.4.1 Model, view, controller pattern

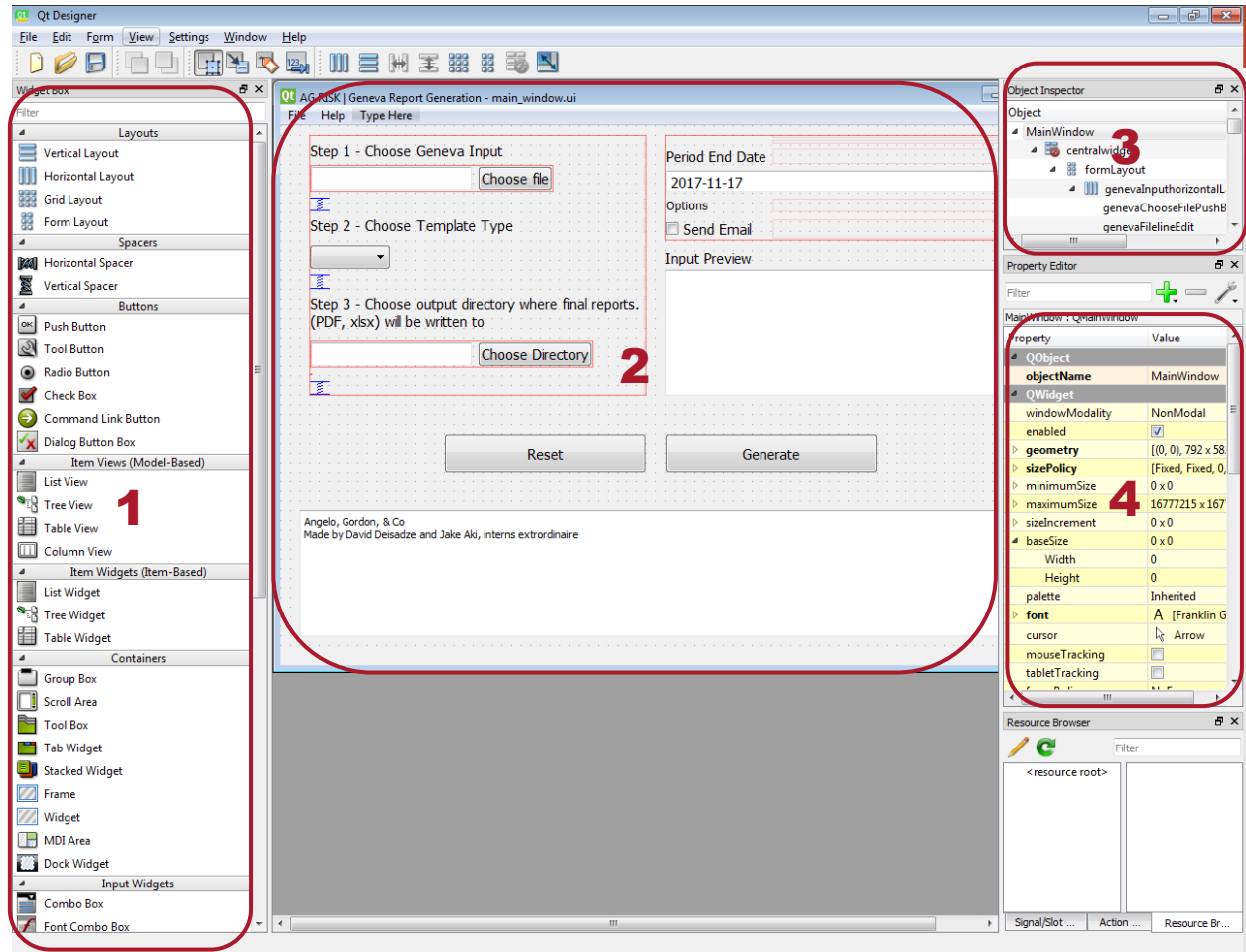
When it comes to designing a graphical user interface (or GUI), there are two popular design patterns. There is model view controller and model view presenter. In both patterns, the model is the data storage. It contains all the stored information and any methods pertaining to accessing or changing that data. However, the way in which the model interacts with the front end is different between the patterns (Qureshi & Sabir, 2013).

In model view controller (MVC), the view renders the model data and periodically requests updates directly from the model. It also is responsible for registering user input and sending that input to the controller to be handled. The controller gets to select the specific view. This means that the controller handles all the backend calculation as well as handling all the events raised in the view.

In model view presenter (MVP), the view handles no business logic and does not directly interact with the model. Instead, the presenter is responsible both for handling events from the view, but also for querying the model for information and sending that information to the view to be rendered.

Ultimately, MVC is simpler to implement and better suited for a lightweight application like ours, but if we had any intention of scaling the front end up into a larger project, MVP might be a good alternative.

## 2.4.2 How the designer works



**FIGURE 5 QT DESIGNER**

PyQt is another excellent open source software package, which acts as a Python API for a popular graphical software called Qt. It includes a tool which allows you to take Qt .ui files and convert them to the corresponding PyQt python files. This allowed us to use the Qt Designer pictured in Figure 5. The designer is like many other graphical software packages, and can be broken down into 4 significant regions. Region 1 is the toolbox. These are all the different widgets that can be dragged into the form displayed in Region 2. Region 2 shows a rough mock-up of how the GUI is going to look. However, since our CSS (or QSS as they are called for Qt Style Sheets) is applied programmatically, it is not an exact representation. Region 3 shows the hierarchy of

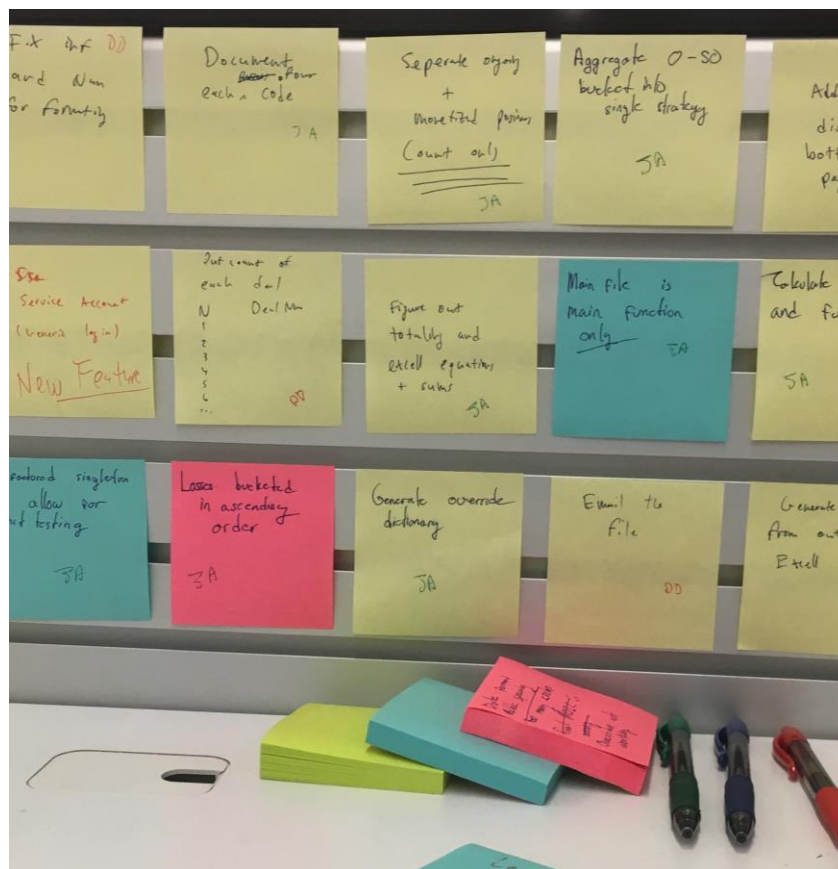


widgets in the form, and allows us to move widgets around to change inheritance. Lastly, region 4 shows all the specific attributes of the focused widget.

## 3 METHODOLOGY

### 3.1 Project management

#### 3.1.1 Kanban



**FIGURE 6 KANBAN BOARD**

Working in a professional setting, we were given a complex problem which had to be broken down to several goals. As a result, we decided to use the Kanban methodology to break down each goal into actionable steps and reasonably complete the steps. We set up a Kanban board in the cubicle and in the desks given as shown in Figure 6. The Kanban board consisted of 3 columns for task organization: to-do, in-progress, and done. The to-do column represented the tasks that needed to be completed; these tasks could be either bugs, features, or improvements. The in-



progress column represented the task was in progress and finally the done column marked the tasks done and ready to test. This method was extremely efficient since we were able to work on one actionable task and finish it thoroughly before starting a new task. Each member could work on a separate task as long as the two tasks were not interdependent.

Different swim lanes representing the tasks were also setup in the Kanban board. Tasks with yellow colors meant normal tasks to be done. Blue marked the task as a new feature and finally red marked the tasks as a bug. This way, our team was always aware of the type of task they were working.

Completing the tasks using the Kanban method improved our workflow as well as making communication easier for when things crashed or did not go according to plan. This organizational technique was a major factor in the success of this project. It also allowed us to communicate goals and storylines to the Angelo Gordon management, hence meeting all their requirements and fixes.

### 3.1.2 Scrum

Scrum is a popular buzz word that gets associated with Agile software development all the time. True Scrum has three roles: a product owner, Scrum master, and team. The product owner is responsible for the product's vision and future. The Scrum master keeps the team on track and facilitates Scrum principles. The team should self-organize to achieve their goals. (James, 2017).

In our miniature implementation of Scrum, we would loosely think of Mr. Burton as the product owner who kept our vision on track, and our project team was a blend of Scrum Master and developers. The scrum master role was less important because of the size of our team, but in terms of logistics, we split it up with one partner handling meeting planning and coordination and the other keeping the focus on our goals.

### 3.1.3 Source control

A version control system (VCS) is at the core of most modern software development. It helps on a myriad of fronts, which are well described on the GIT Tower website (Tower, 2017):

- **Collaboration:** When working on a project simultaneously, we inevitably need an answer to the question of “what happens when we edit the same thing at the same time”. Some VCS such as subversion resolve this issue by “locking” the repository such that only one coder can work on a section of code at one time.





WPI



ANGELO,  
GORDON  
& CO.

However, this is not scalable. More sophisticated VCS like GIT rely on “merging” which only comes into effect when two or more coders change the same piece of code, in which case the coders sit down, review the changes, and accept which one they want to use.

- **Storing old versions:** How do we keep track of changes and releases? How do we name our older versions? How do we know how much or how little information to keep track of? A good VCS allows us to decide these from the beginning and forces us to stick to those conventions automatically.
- **Restoring previous versions:** What do we do when inevitably a code-breaking change is pushed accidentally. Revert! Because VCS keeps track of all older versions of code, we can go through line by line to see what changes caused the break and revert the guilty code. It is a powerful debugging tool.
- **Backup:** These days everything lives in the cloud, and VCS assists with that. This way if one programmer’s computer explodes, only so much work is lost.

Clearly, even for projects consisting of one or two programmers, a strong VCS is vital to successful development. Angelo, Gordon & Co. uses the Microsoft branded version of GIT, so that is the VCS we used for this project.



**FIGURE 7 GIT HISTORY AS OF NOV. 20TH**

As a rule of thumb, we followed these guidelines:

- Branch for every major development goal
  - We branched for any new potentially breaking change, new feature, or new project that could be done in parallel to the current project.
  - We did several major code refactoring's (cleaning up code and reengineering code for less duplication) to enforce better code consistency and brevity, which always has the risk of breaking code.

- GUI development was a new feature entirely so it got a branch.
- Regression testing could be done in parallel, so it was a branch.
- Every stable build was tagged as a release in case we ran out of time on a new feature and needed a stable version to provide to our sponsor before leaving.
- When a new branch was well-tested it was merged back into the main development branch, and then finally back into the master branch.

## 3.2 Development

---

### 3.2.1 Object oriented analysis & design

When it made sense, object oriented principles were applied to this project. One good example of this is with regards to how we formatted the financial information in the software. In terms of our pipeline, raw accounting data is read in first by the data processing script provided by the sponsor (but still part of the pipeline) and then the processed data is spat back out in a CSV (Comma Separate Value file format). It is this CSV that is fed into the report formatter and generator. To store the data in the CSV a list of “StrategyEntry” objects are used (see reference code in Appendix A).

This object is responsible for handling the information and methods of a single financial strategy. It stores the raw CSV data in a dictionary and has methods for accessing that dictionary, performing additional processing on the data, and adding new data from SQL queries and similar.

### 3.2.2 Design patterns

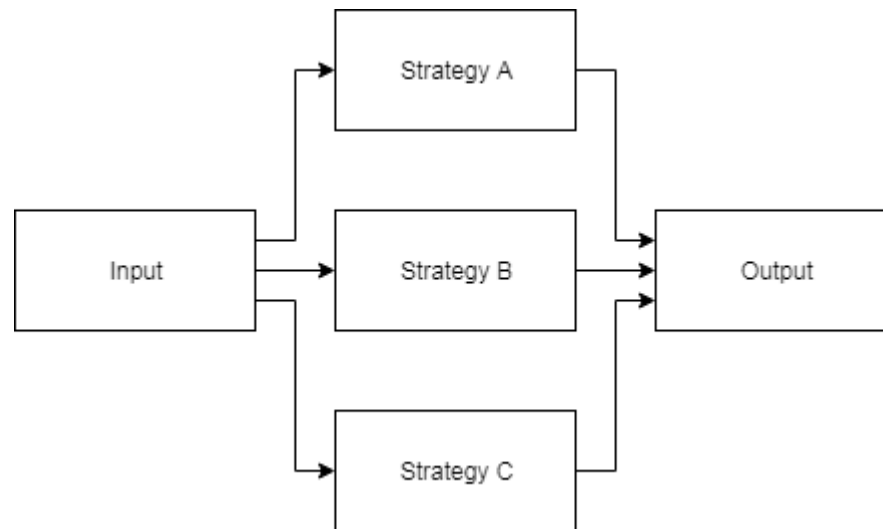
Another example of object oriented design is in how the dynamic columns and processing are done. These objects use a combination of the strategy design pattern<sup>v</sup> and the abstract factory design pattern.

According to an excellent definition on the Sourcemaking website, the strategy design pattern “...defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from the clients

---

<sup>v</sup> Design patterns are a powerful and somewhat controversial idea in computer science. The idea is that certain logical structures and patterns occur over and over in software design and that particularly effective patterns ought to be reused both for ease of comprehension (as experienced software architects will be familiar with common design patterns) and because of a sense of “why reinvent the wheel?” Critics of design patterns suggest that using a design pattern blindly without respecting basic good coding practices leads to bad code. They suggest that if a coder just follows good object oriented principles (see the endnote on SOLID) these patterns will arise naturally if they’re required (Rodriguez, 2016).

that use it.” (Shvets, Frey, & Pavlova, Strategy Design Pattern, 2017). Fundamentally, the idea here is that if we have a bunch of different operations that need to be performed on some input and always produce the same kind of output, we can encapsulate that logic as a “strategy” and then plug in whatever “strategy” we need.



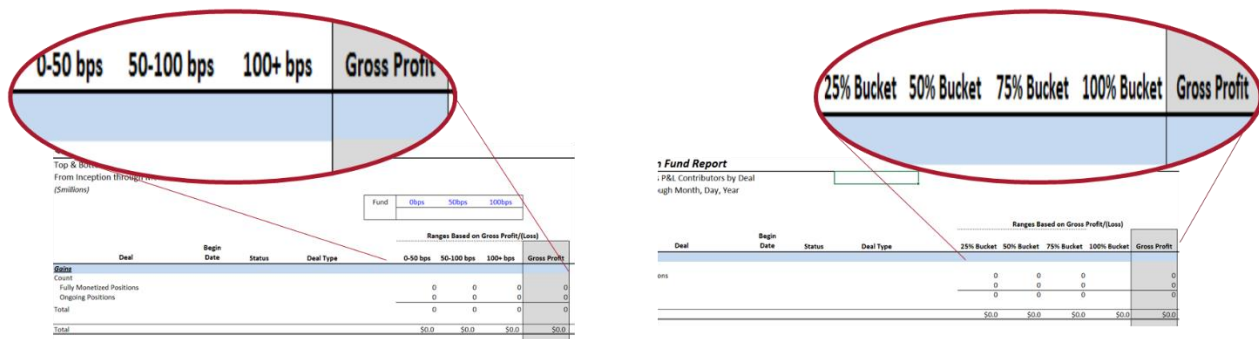
**FIGURE 8 STRATEGY DESIGN PATTERN**

As shown in figure 7, for a given input, it does not matter what path is taken (Strategy A, B, or C) we still arrive at the same type of output. Moreover, if the input type is the same as the output type, these strategies can be chained together to perform multiple operations on some data. When working on this project, a common confusion that arose was between a financial *strategy* and the *strategy* design pattern. These are completely unrelated ideas. A financial strategy is some investment plan that Angelo, Gordon uses. As far as our code is concerned, a financial strategy is simply a data point. These financial strategies are assigned a unique ID and when it comes time to use the strategy, Angelo Gordon’s codebase can look-up and perform the business logic based on that unique ID. The strategy design pattern is a way of designing our algorithms, and each algorithm designed in this pattern is referred to as a strategy.

The abstract factory design pattern is also described on Sourcemaking: “[to] Provide an interface for creating families of related or dependent objects without specifying their concrete classes.” (Shvets, Frey, & Pavlova, Abstract factory design pattern, 2017). The idea with an abstract factory is to have a class dedicated for creating related objects, and there is a perfect application for this factory with the strategy pattern, because strategies are all similar objects. So, the abstract factory churns out all the required strategies, which are then used in series to process the data.

### 3.3 Dynamic report generation

#### 3.3.1 Open funds vs closed funds



**FIGURE 9 CLOSED VS OPEN FUND TEMPLATE**

The first report template we worked with was for a closed fund. Because of the differences discussed in the background, a closed fund is primarily organized by a system of basis points relative to the fund size. A logical conclusion from this is that the fund size is also an important piece of information. Thus, the closed fund has a widget for displaying the fund size, what dollar amounts correspond to what basis point brackets, or buckets, and the parsed strategies should be sorted by their current basis point profits in relation to the total gross profit. As can be seen in Figure 9, three basis point brackets are used 0-50, 50-100, and 100+. All strategies in the lowest bucket are concatenated into a lumped “All Other Positions” for brevity.

The second report template was the open fund. This report gets to omit the fund size, because it can change from week to week. Furthermore, because basis points relative to fund size is not a useful measure, they are sorted by quartiles based on quarter percentage intervals as shown in Figure 9.

#### 3.3.2 Multiple templates

The best way to accomplish these distinctions between reports was by using fundamentally different Excel report templates.



3 is where all the totaling is done and provides a rough estimate of the total gross profits and losses by basis point bucket.

These reports could be done in any time frame (e.g. weekly, monthly, annually etc.). However, they were typically generated quarterly. Part of the impetus behind making these reports easier to generate was to allow them to be produced on a more spontaneous basis perhaps monthly or even weekly in the case of a systemic risk event.

### 3.3.3 Dynamic Column Strategies

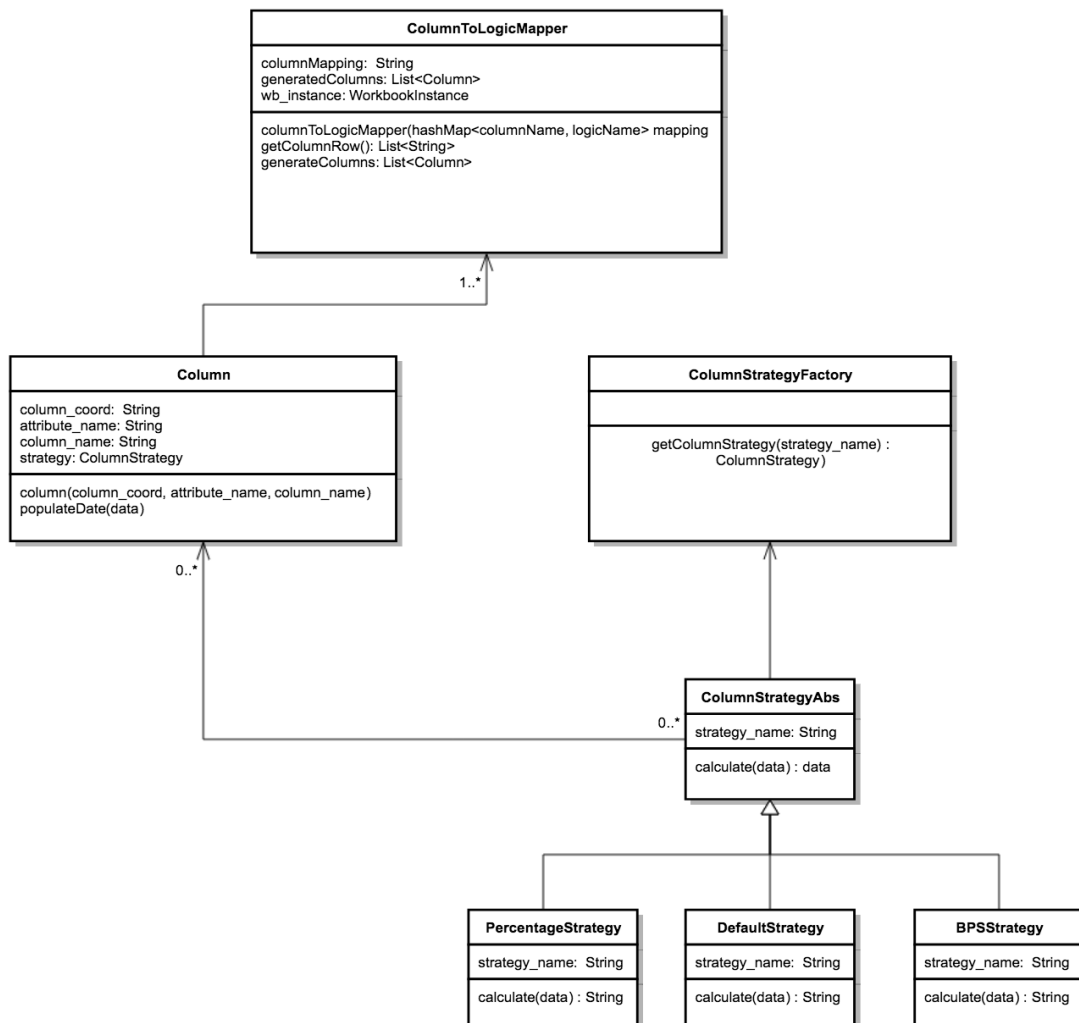
To support the different templates programmatically, we developed a system called dynamic columns. The system would have one entry point in the configuration files where excel columns would be mapped to a specific strategy (work done on the data) and a data attribute. The data attribute specified the portion of the data to be used for that specific column. As you can see in Figure 11, for the winners and losers closed fund report mapping, we specify the header row which is the row in Excel where the column headers are located. This allows the code to load in the column names and makes it easier to do the mapping.

```
winners_losers_column_mapping_closed_fund = {
  "header_row": 9,
  "column_mapping": [
    (1, "count", None),
    ("Deal", "default", "Deal"),
    ("BeginDate", "default", "first_cashflow"),
    ("Status", "monetized", None),
    ("Deal Type", "default", "Deal Type"),
    ("100+ bps", "bpp_in_mills", "cat3"),
    ("50-100 bps", "bpp_in_mills", "cat2"),
    ("0-50 bps", "bpp_in_mills", "cat1"),
    ("Gross Profit", "in_millions", "gross_profit"),
    ("% Total", "percent_change", None),
    ("Gross IRR", "float_default", "irr"),
    ("MOIC", "default", "moic"),
  ],
  "preprocessing": [{"overrides", {"workbook", "overrides"}},
    {"calculate_bps", {"fund_size",}},
    {"sort_by_gp", None},
    {"concat_lower_bps", {"keys",}},
    {"generate_gics", {"codes",}}]
}
```

**FIGURE 11 DYNAMIC COLUMN MAPPING**

The next argument is the `column_mapping`, which as mentioned before maps the column name (or index of the column), the strategy, and the parameter name; if no parameter name is needed, `None` is passed in. Finally, the last argument is the

preprocessing mapping. The preprocessing preps and manipulates the data with methods such as sorting, bucketing, and database queries. Both the column mapping and the preprocessing are highly dynamic and were built by us following the strategy pattern discussed in the Design patterns section 3.2.2. This allows the user to create custom strategies (for both columns and preprocessing) on the fly by extending the main strategy class and correctly implementing correct logic. The UML diagram for the column mapping is shown in Figure 12. For sample strategies see Appendix B.



**FIGURE 12 COLUMN STRATEGY UML DIAGRAM**



## 3.4 GUI development

---

The GUI was developed in the Qt's Python version called PyQt version 5. We chose this since it supported major user interface elements functionality as well as providing a simple user interface designer which was easy to use. The code followed a model view controller (MVC) paradigm which separates the design, business logic, and the data into separate components. This way there is a maintainable codebase as well as a stable functioning GUI which is enclosed in validators and exception handling.

The view was separated into two layers: application and widget views. The application view is the main view that relays all inputs and processes to widgets and popups. Widgets exist as elements on the application. For each widget, we created a separate view class and controller that encapsulated the visual aspects and business logic, respectively.

For application-wide data, we created a model which acted as a proxy between configuration files as well as storing globally available data. Controllers and view methods could subscribe to the model for changes, which allowed the widgets to “self-update” depending on data update, insertions, or deletion.

For features such as report generation and unit test simulators, we developed a threading layer which helped maintain usability in the GUI while allowing background processing.

## 3.5 Documentation

---

### 3.5.1 Writing the README

Initially, the README was intended as just a place to keep track of different dependencies that we had installed over the course of the project. However, we quickly realized that using a requirements.txt file was an obviously superior solution. The popular Python module installer pip supports a txt file, which lists all dependencies and allows the user to install said dependencies with a single command:

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

This allowed us to start using the README as a tool for leaving our sponsor with something resembling a user guide. We drafted up a first version with instructions for installing and running the program. With user feedback from Mr. Burton, we also added sections to explain how to modify the code to support a new template, how to run tests, and some potential runtime errors.

### 3.5.2 Different markup languages

Initially the file was written in raw text. However, Pycharms and GIT both provide excellent Markdown support. Markdown is light-weight markup language that allows for easy conversion to HTML. It was developed by John Gruber and Aaron Swartz in 2004 (Gruber, 2004). Over the years it has gained popularity and slightly different “flavors” of Markdown have arisen. GIT has specific Markdown conventions, which is how we styled our README, and it can be quite powerful. What we started with is shown below in Figure 13.

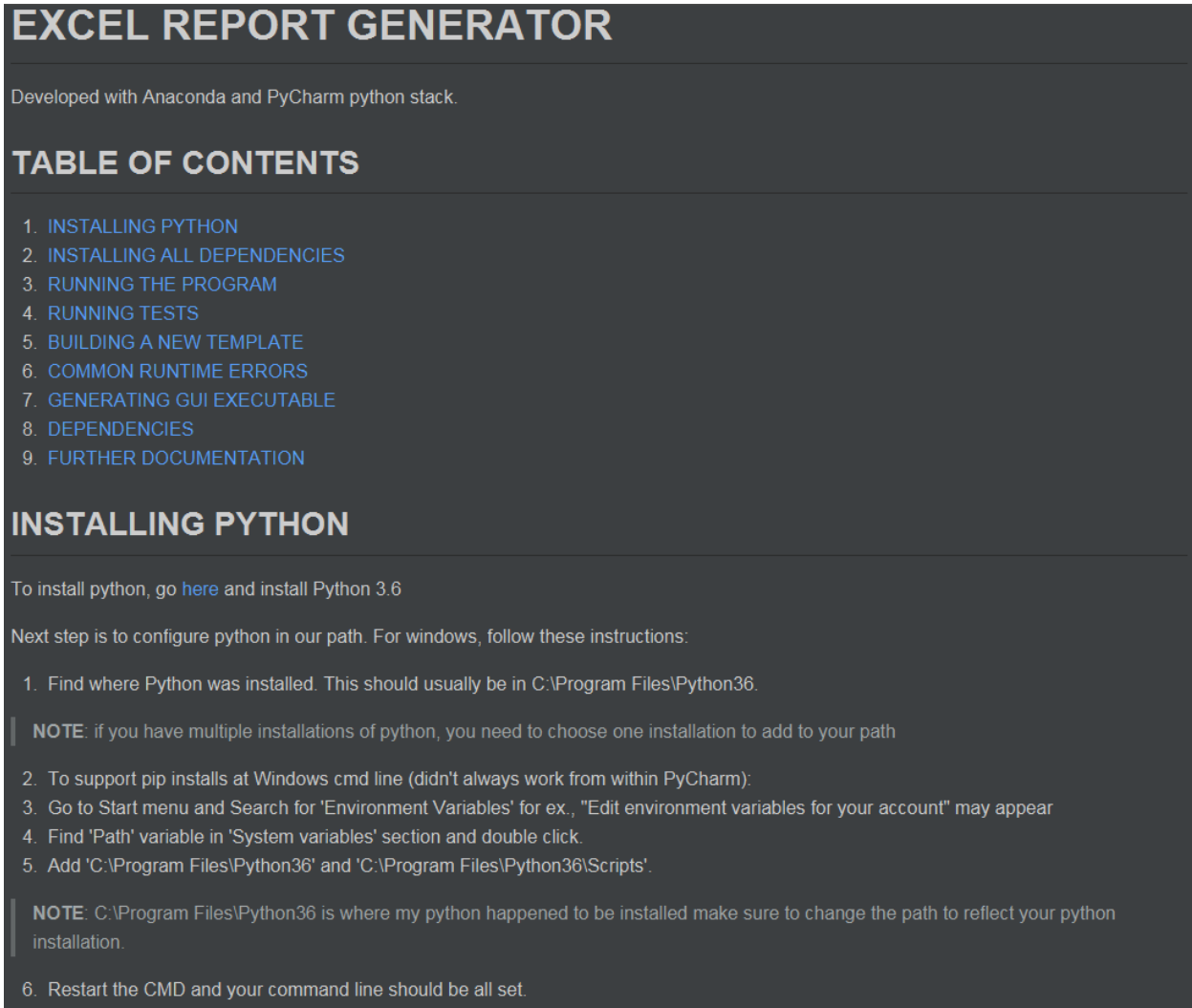
```

1 # EXCEL REPORT GENERATOR
2 Developed with Anaconda and PyCharm python stack.
3
4 ## TABLE OF CONTENTS
5 1. [INSTALLING PYTHON] (#1)
6 2. [INSTALLING ALL DEPENDENCIES] (#2)
7 3. [RUNNING THE PROGRAM] (#3)
8 4. [RUNNING TESTS] (#4)
9 5. [BUILDING A NEW TEMPLATE] (#5)
10 6. [COMMON RUNTIME ERRORS] (#6)
11 7. [GENERATING GUI EXECUTABLE] (#7)
12 8. [DEPENDENCIES] (#8)
13 9. [FURTHER DOCUMENTATION] (#9)
14
15 ## <a name="1"></a>INSTALLING PYTHON
16 To install python, go here (https://www.python.org/) and install Python 3.6
17
18 Next step is to configure python in our path. For windows, follow these instructions:
19 1. Find where Python was installed. This should usually be in C:\Program Files\Python36.
20 >***NOTE** if you have multiple installations of python, you need to choose one installation to add to your path
21 2. To support pip installs at Windows cmd line (didn't always work from within PyCharm):
22 3. Go to Start menu and Search for 'Environment Variables' for ex., "Edit environment variables for your account" may appear
23 4. Find 'Path' variable in 'System variables' section and double click.
24 5. Add 'C:\Program Files\Python36' and 'C:\Program Files\Python36\Scripts'.
25 >***NOTE** C:\Program Files\Python36 is where my python happened to be installed make sure to change the path to reflect your python install
26 6. Restart the CMD and your command line should be all set.
27
28 ## <a name="2"></a>INSTALLING ALL DEPENDENCIES
29 ### General Information
30 The required packages are listed in the file "requirements.txt"
31 This file would normally be in the default home folder created by Pycharm (e.g., C:\Users\Sburton\PycharmProjects\Distressed)
32 After installing python and loading python into the path, run 'pip install -r requirements.txt'. This should install all dependencies.
33 If this fails, you can go through the fail and manually install by using 'easy_install <module>' or 'pip install <module>'
34
35 ### To Install All Packages
36
37 1. First, do all manual installations
38 1. Install email with: 'easy_install email'
39 2. Install pywin32
40 1. Navigate to the 'lib directory'
41 2. Depending on your python installation you will need to either install pywin32-221.win-amd64-py3.6
42 or pywin32-221.win32-py3.6
43 >***NOTE** Assuming you're using the normal Angelo, Gordon python stack, it should be pywin32-221.win-amd64-py3.6
44 3. Double click the executable to run it.
45
46 3. Install the rest of the requirements with: 'pip install -r requirements.txt'
47 1. If this fails to install a module, try installing the module directly with either 'pip install <module>' or 'easy_install <module>'
48 2. Then, rerun 'pip install -r requirements.txt'
49
50 ## <a name="3"></a>RUNNING THE PROGRAM
51 There are several main files which help run different excel sheets.
52 main_pipeline_cmd.py - this will run the full pipeline. Taking in the geneva file and running

```

**FIGURE 13 README IN MARKDOWN**

This Markdown can be converted straight to HTML in Pycharms, which gives this nicely formatted document.



**EXCEL REPORT GENERATOR**

Developed with Anaconda and PyCharm python stack.

**TABLE OF CONTENTS**

- 1. [INSTALLING PYTHON](#)
- 2. [INSTALLING ALL DEPENDENCIES](#)
- 3. [RUNNING THE PROGRAM](#)
- 4. [RUNNING TESTS](#)
- 5. [BUILDING A NEW TEMPLATE](#)
- 6. [COMMON RUNTIME ERRORS](#)
- 7. [GENERATING GUI EXECUTABLE](#)
- 8. [DEPENDENCIES](#)
- 9. [FURTHER DOCUMENTATION](#)

**INSTALLING PYTHON**

To install python, go [here](#) and install Python 3.6

Next step is to configure python in our path. For windows, follow these instructions:

1. Find where Python was installed. This should usually be in C:\Program Files\Python36.

**NOTE:** if you have multiple installations of python, you need to choose one installation to add to your path

2. To support pip installs at Windows cmd line (didn't always work from within PyCharm):
3. Go to Start menu and Search for 'Environment Variables' for ex., "Edit environment variables for your account" may appear
4. Find 'Path' variable in 'System variables' section and double click.
5. Add 'C:\Program Files\Python36' and 'C:\Program Files\Python36\Scripts'.

**NOTE:** C:\Program Files\Python36 is where my python happened to be installed make sure to change the path to reflect your python installation.

6. Restart the CMD and your command line should be all set.

**FIGURE 14 FORMATTED MARKDOWN**

This then also integrates with GIT allowing you to view the README online.

### 3.5.3 Incorporating with the GUI

As we started developing the GUI for this project, we realized it could be useful to include the README in the GUI. Initially we generated the HTML using a Markdown-to-HTML converter and then fed that into a text browser widget, but that generated the document in figure 14.



**FIGURE 15 PRE-CSS README**

But, we were advised by Professor Ciaraldi to move to the solarized stylesheet to both better match the rest of the front end, but also to be less harsh on the eyes, so we settled on this final design with a dark solarized stylesheet.



FIGURE 16 SOLARIZED README

---

## 4 RESULTS

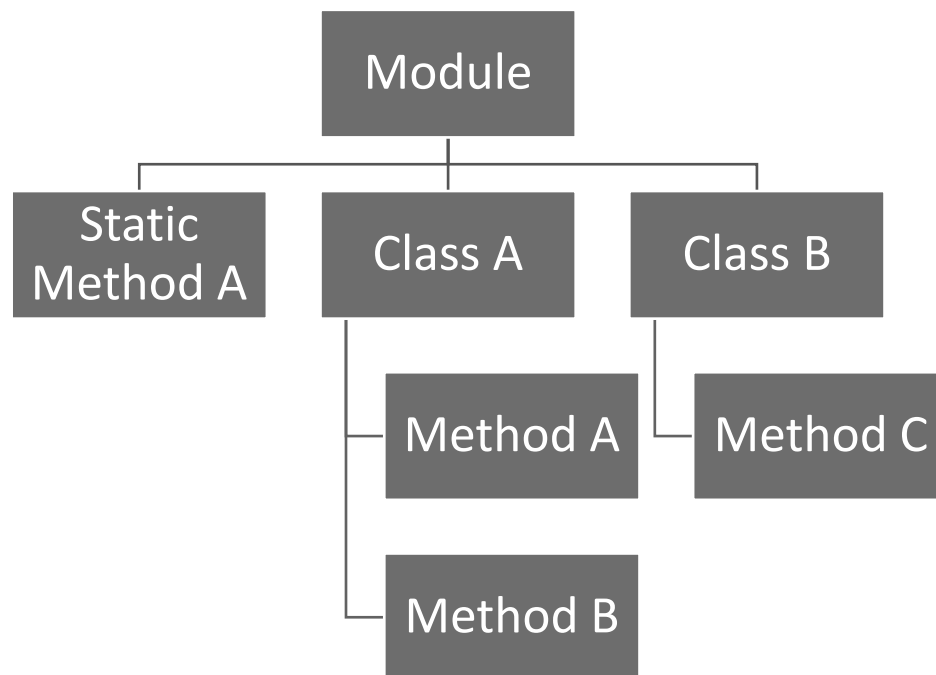
---

### 4.1 Testing

---

#### 4.1.1 Unit testing

The goal behind unit testing is to methodically compare the output of a given module with some kind of interface or blackbox specification. Usually a unit test is structured somewhat recursively.



**FIGURE 17 UNIT TEST STRUCTURE**

The idea is that a single test file is responsible for testing a single module, then individual test cases will be defined per class (or if there are any floating static<sup>vii</sup>

---

<sup>vii</sup> The idea of a static method in Python is a little trivial as it is entirely possible to define a function outside of a class anyways, and Python is not strictly object oriented either. However, there exists a Pythonic way of defining static methods if we wished to and it looks syntactically like this:

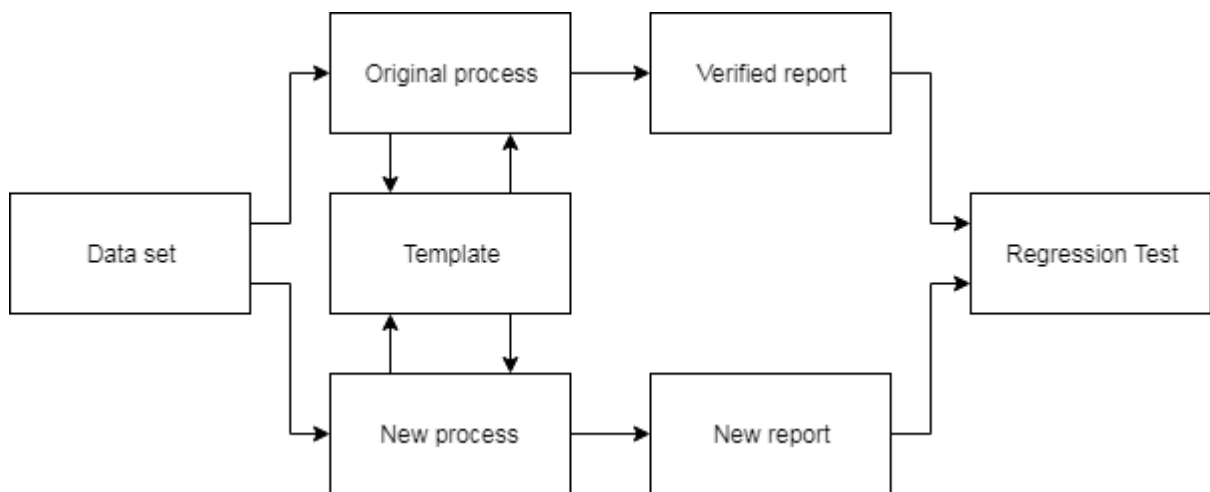
```
class ClassA():
    @staticmethod
    def method_a(a):
        return a
```

methods, per method) and then within each test case an individual test is written for each method.

Using Python’s unittest module, this structure is easy to follow by inheriting the TestCase object and implementing test methods that use methods like assertTrue. One element that was particularly important to test with unittests was the behavior of different comparison methods for OpenPyXL objects. As has been mentioned before, there is no guarantee of OpenPyXL’s behavior, and these comparison methods need to work for the sake of the regression testing explained below. Verifying that their functionality remains is paramount.

### 4.1.2 Regression testing

Regression testing is an important but oft underutilized tool for detecting problems before they even happen. A regression test is different from unit test in that it is not used to diagnose a part of the system. Instead, a regression test is run whenever new functionality is added or old functionality is changed to make sure that the changes have not accidentally broken anything that previously worked (Myers, Sandler, & Badgett, 2011).



**FIGURE 18 REGRESSION TESTING WORKFLOW**

The general model for our regression testing was simple. We started with a fixed data set of real but older data. We would run the data on a fixed template and generate a report, which would be verified via inspection. Then, in the future if we made changes to the code, we would run the same data set with the same source template, and compare the new report to the old verified report programmatically. If there were any differences, the test was failed and the inconsistency was flagged. If the purpose of the

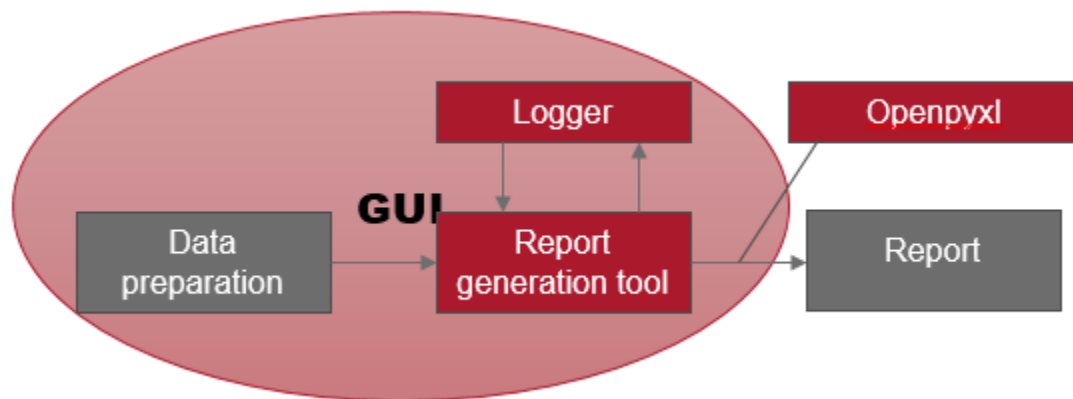
update was to change the code, the developer would need to remove the at fault unit tests.

This means that the regression test will diagnose any changes in the programmatic functionality, but will not work if the data set or template is changed. If either of those are changed a new reference report will need to be generated.

## 4.2 Product

---

### 4.2.1 Overview



**FIGURE 19 REPORTING TOOL OVERALL STRUCTURE**

This figure shows the final state of the project. The segments highlighted in crimson are the portions that we developed or built from the ground up. The segments in gray were either only slightly modified or left untouched entirely.

Initially, the scope of the project was just to develop the report generation tool. The goal was to convert that tool into Python, and so that necessitated also working with OpenPyXL, which came with its own set of issues and problems that required additional tools to be built. As those tools were brought online, different diagnostics became useful and a logging infrastructure was developed to work in parallel with the reporting tool. Finally, when the backend was finished, we took it upon themselves to develop a GUI to encapsulate the data preparation, logging, and report generation. Of course, all of these steps have some degree of a testing framework built with them.



### 4.2.2 OpenPyXL tools

As described in the background on OpenPyXL, the biggest weakness to OpenPyXL is also its greatest strength, which is its cell-centric design. Fundamentally, any attempt to manipulate the actual data in the workbook required work at the cell level. So, a suite of tools was built out to support working at a row level and at a table level. Furthermore, it was slowly realized that the ability to read and write from named cells was valuable, and so support for that was added as well.

The major row level operation that was needed was the ability to insert a row of new cells with matching formatting and some arbitrary point in the sheet. We borrowed a stackoverflow solution for row insertion, but as time went on and bugs were found with this solution modifications were made.



**FIGURE 20 INSERT ROWS WORKFLOW**

The main bug found with this code was in step 2. This is described in the Background section (section 2.0), but effectively, any cells with references to named cells that are named in a similar fashion as a cell reference (i.e. uppercase letter followed by numbers) that named cell would be incremented as well.

The next tool that was built was the ability to both find and move a named table. Furthermore, an additional layer was needed to be built on top of the row insertion tool in order to allow the insertion of a row into a table. The fundamental problem here was that the Cell data structure's internal reference does not match up to the Table data structures list of Cell references. So, when a new row was inserted the individual Cells were updated, but the table was not. If a row was inserted above the table or in the table, it would break the table. However, if the row was inserted below the table, everything was okay. So, tools were built to support that.

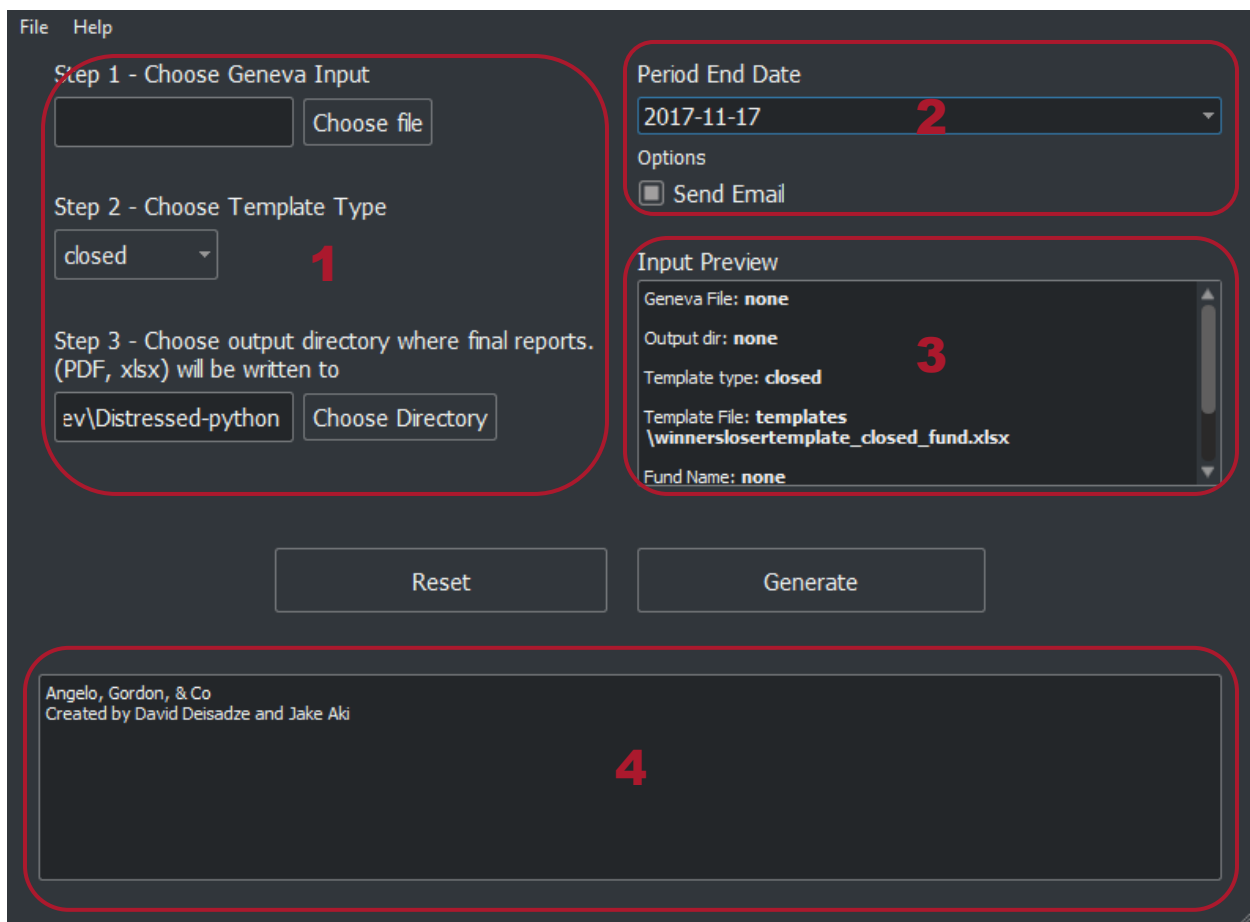


### 4.2.3 Logging tools

Python provides a powerful built in logging module, which is what we leaned on for the logging support in this project. We configured two separate loggers. One was the default logger, which logged everything into a file, which was stored with an ID unique to the day of the log. We assumed we would not need specificity down to the minute as this report is only run once a quarter. The second logger was the standard output logger and just created a clean format for the log message before streaming the messages to both `sys.stdout` and our GUI's console.

## 4.3 GUI

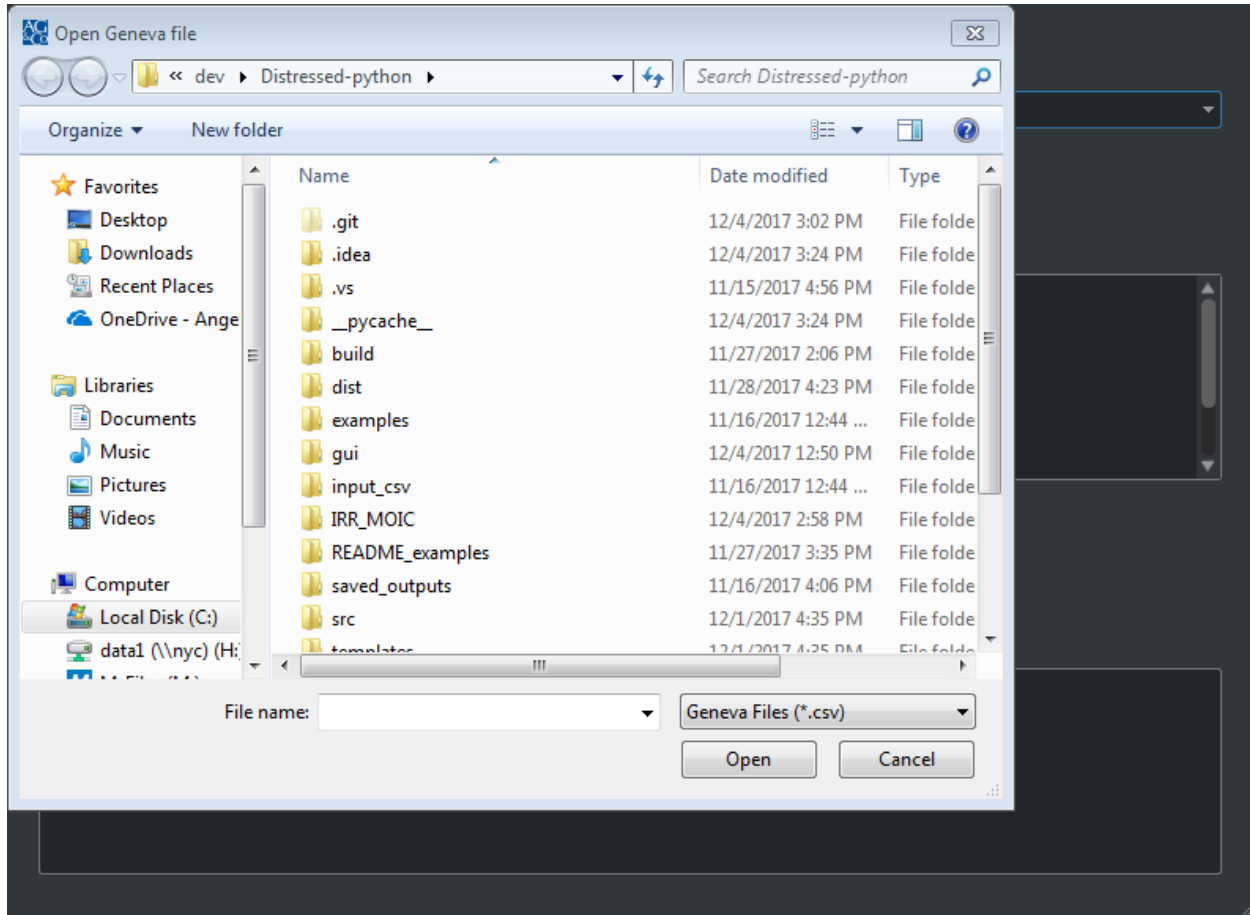
### 4.3.1 GUI implementation



**FIGURE 21 MAIN GUI FORM VIEW**

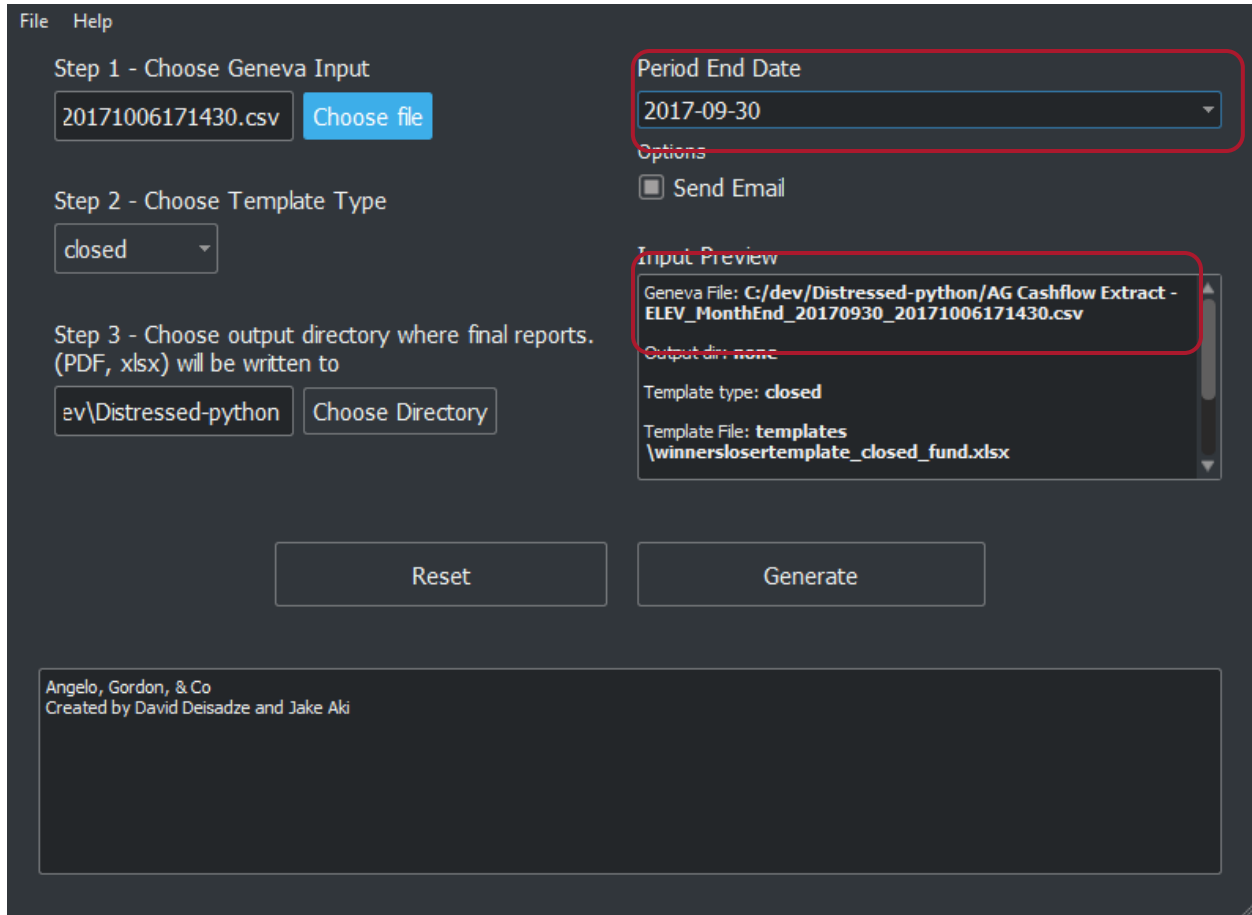
This figure shows the home page of the graphical user interface that was developed to support the report generation tool. Region 1 is the main inputs to the report. These must be selected to generate a report. There must be an input file, a template type, and an output directory. Region 2 are optional settings for the report, and they include changing the period end date and sending an email. Region 3 shows a preview of all the options currently selected, and region 4 is the console, which shows logging outputs. The two buttons are self-descriptive, but for completeness, the reset

button resets all settings and clears the console. The generate button kicks off the report generation based on the settings selected.



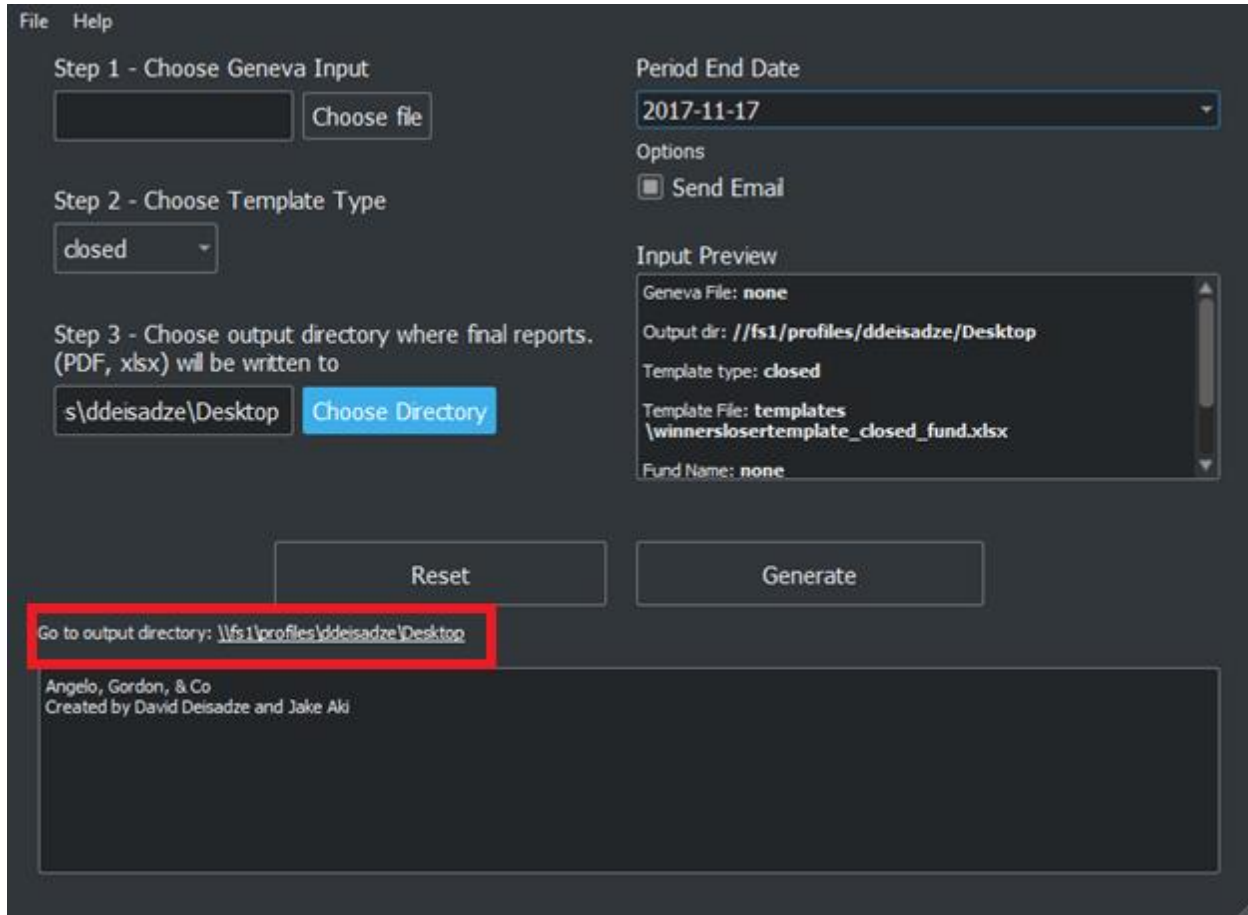
**FIGURE 22 SELECTING AN INPUT FILE**

Qt allows for a file explorer dialog input, which is what is used for selecting the input file. This explorer remembers the previous folder location, so if multiple reports are run back to back, it is easy to navigate to the same folder.



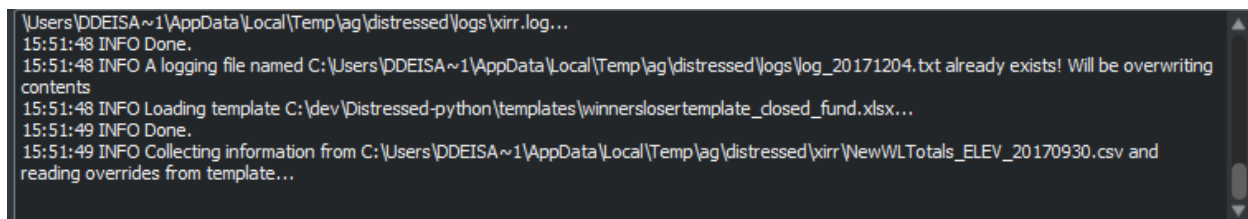
**FIGURE 23 AUTOMATIC FILE NAME PARSING**

Another note about selecting the input file is that the file name is automatically parsed by the GUI to extract the period end date and fund name. Furthermore, the input preview automatically updates the absolute path to confirm that you have selected the correct input file.



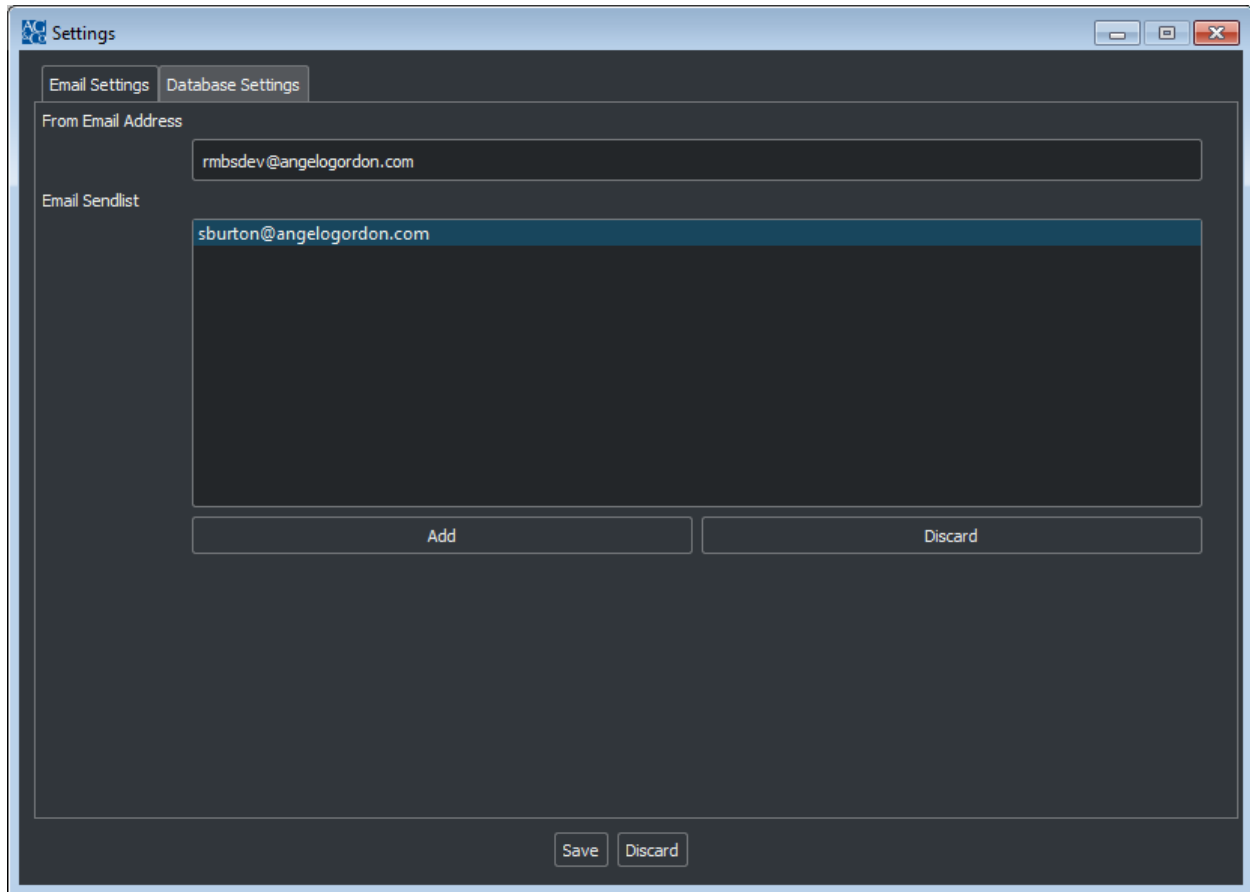
**FIGURE 24 OUTPUT DIRECTORY LINK**

Choosing the output directory uses the same file explorer as the input file, and when it is selected not only does the input preview update, it also provides a link (highlighted in red), which will automatically open a file explorer in the output directory so the user may easily check the result of the report generation.



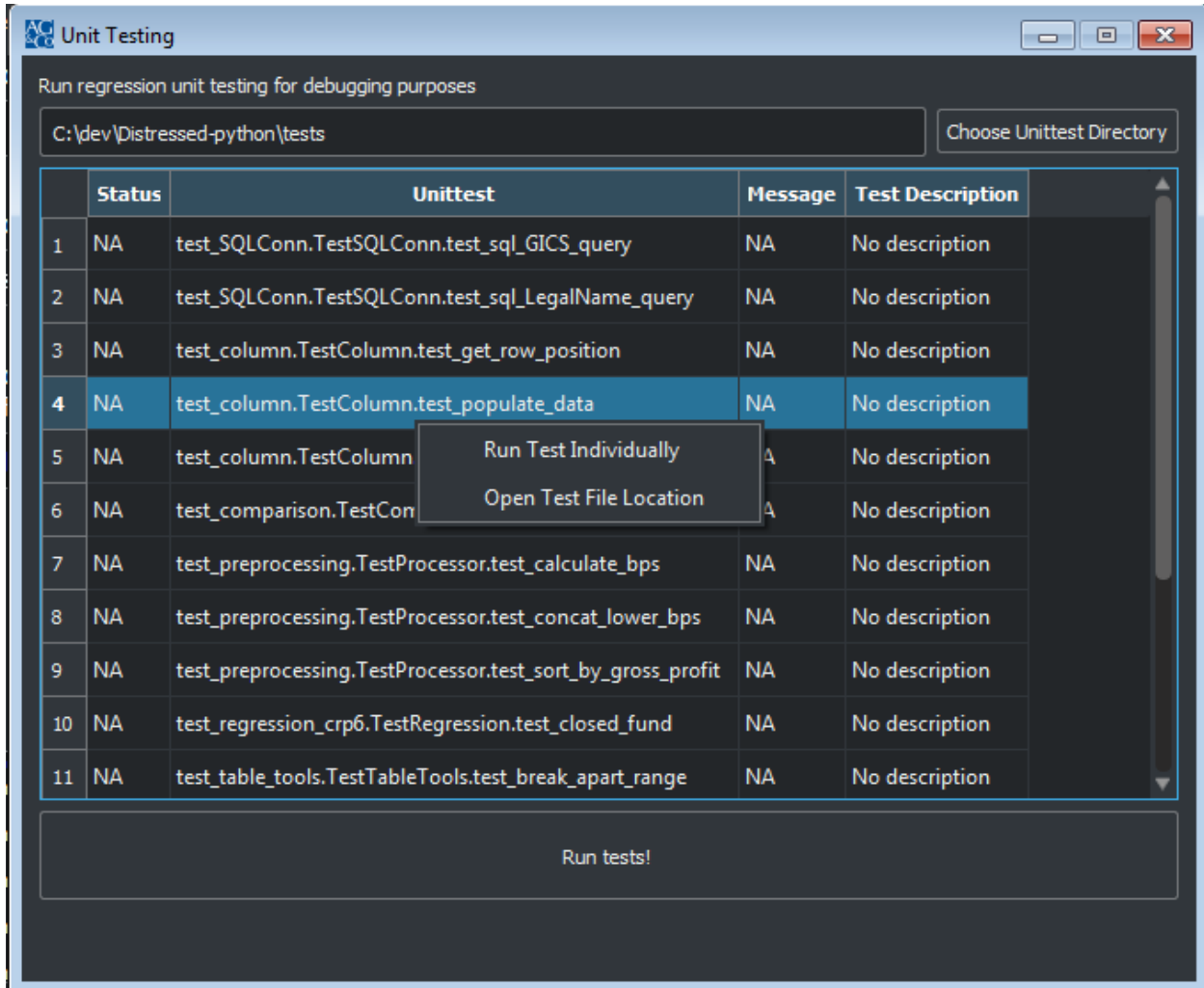
**FIGURE 25 CONSOLE LOG**

When the report is running, well formatted log messages are displayed to show the progress. Error messages and warnings will also be directed here.



**FIGURE 26 SETTINGS TAB**

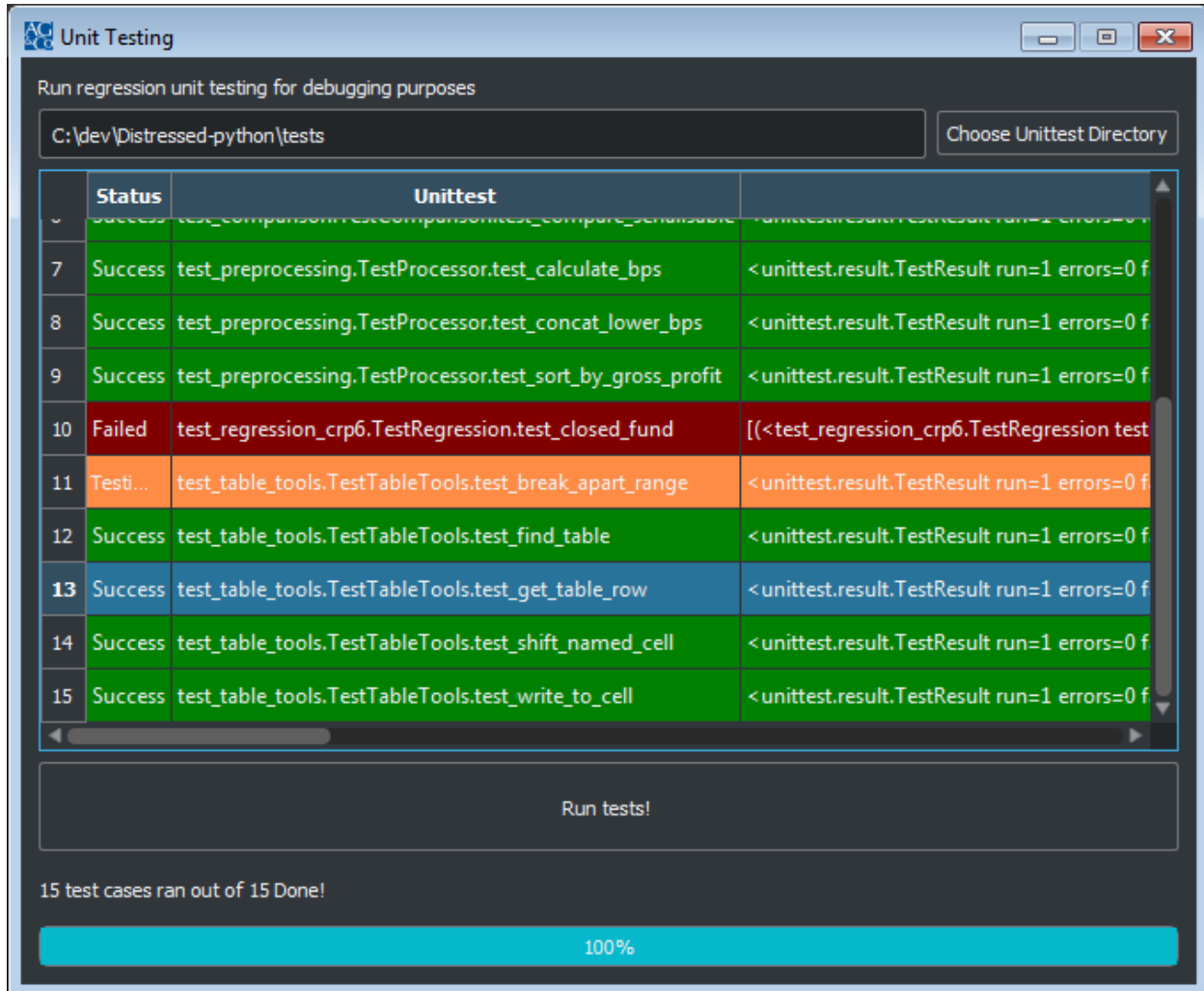
There is also a settings tab, which can be opened with keyboard shortcuts (Ctrl + S), or from the drop-down file menu. This allows for the configuration of other settings such as recipient email addresses, database settings, and sender addresses.



**FIGURE 27 TESTING TAB**

This testing tab is used to run diagnostics from within the GUI. It reads tests in dynamically from a selected folder allowing the user to add or remove test cases in the future without needing to manually update the GUI.





**FIGURE 28 RUNNING THE UNIT TESTS**

The testing interface uses a typical scheme of green for success, red for failure, and yellow for in progress. Furthermore, the error message can be double clicked to show more details, and by right clicking an individual test can be run by itself.

### 4.3.2 Generating executable for GUI

To further streamline the process of running the GUI, we also built an .msi style installer. This installer uses pyinstaller to build an executable version of the GUI, and then Inno Script Compiler (Russel, 2018) to build a setup.exe to install this executable. When correctly maintained, this provides a clean professional interface to install the executable.



---

## 5 CONCLUSIONS

---

### 5.1 What went wrong

---

While developing this project, it was important not to lose sight of the fact that this was a student project with the intention of learning and growing our skills as developers, engineers, and professionals. A key factor in learning is reflection, which is perhaps the most useful element of this report as it has given us the opportunity to look back, reflect, and consider what approaches worked well and what did not. Many of the mistakes that were made early on were fixed later in the lifespan of this project. Pycharm's refactoring tools were a godsend in the middle few weeks of project development allowing us to drastically change our approaches in several areas. But, we wanted to focus on a few specific errors, mistakes, and flaws and how we either resolved these issues or would resolve these problems in the future.

#### 5.1.1 Singleton design pattern

Early in the development of this project, we struggled with how to pass data around. There was a question of how the OpenPyXL workbook object should be handled. From a given run of the software, we only ever work with a single template, and it is important that all references to a workbook are to the same workbook. This seemed like an obvious application for another design pattern. This one is called the Singleton design pattern, once again referring to our friends at Sourcemaking (Shvets, Frey, & Pavlova, Singleton Design Pattern, 2017): "Ensure a class has only one instance, and provide a global point of access to it... Encapsulated 'just-in-time initialization' or 'initialization on first use'"

The idea is that an instance of a class is only ever made once, and then any references to that object in the future will simply return that object again. However, one important drawback to this is that we can never re-initialize this object. So, when we were simply running the report generation once, this was never an issue. However, in the GUI, you could generate multiple reports sequentially. Furthermore, in unit testing, we could not just set up and tear down the singleton, because once it was set up, it was there for good.

Therefore, after the first two weeks, we switched away from the singleton structure and simply passed around an instantiation of the workbook object itself. We risk more exposure to internals of the OpenPyXL library with this strategy, and it makes

some of the arguments for our functions more verbose, but ultimately it is a more flexible and logical approach.

### 5.1.2 Hardcoded template references

Ultimately, for this report generation to work based off a pre-existing Excel template, there must exist some degree of incestuous knowledge between the template and the report code. However, the project slowly evolved a more sophisticated method of interpreting the template as time went on.

Initially, the project had a large config file that contained enumeration<sup>viii</sup> style variables that had various cell references that were needed to know where different values in the template were held. For example, it had references for the different fields that needed to be populated, for the upper and lower corners of the gains and losses tables, and for the locations of different totaling cells. All of these references were hardcoded, which was a nightmare to update whenever the template changed and was generally bad practice.

So, we found a better way. By building on some of the OpenPyXL functionality, we were able to reference named ranges and tables. It was still necessary to hardcode the name of a table or the name of a range, but it was much more readable and more robust to changes in the template. This is the minimum level of knowledge that the software must have of the template to function.

### 5.1.3 Levels of encapsulation

This issue was less of a problem that came up and was more of a philosophical conversation that was ongoing during development. There are many definitions of object oriented programming. One popular definition is provided by Matt Weisfeld who described object oriented programming as adhering to four primary principles: encapsulation, inheritance, polymorphism, and composition (Weisfeld, 2013).

---

<sup>viii</sup> An enum, or enumeration, is a type of variable support in many modern languages such as C, C#, Java, and others. The idea behind an enum is to provide a typed structure for a list of related values. The common example is a list of valid display colors, which might be red, blue, and yellow. In code, we might represent red as 1, blue as 2, and yellow as 3. However, as far as most languages are concerned, that means that 4, 5, or any other integer would be valid. But, we only want red, blue and yellow. An enum restricts the valid values of that enum type to 1, 2, 3. i.e. it restricts it to a subset of valid ints. They also provide readability as it allows you to have a constant name and force the developer to think about all possible values for that enum (Bolton, 2017).

The main idea behind encapsulation is that a piece of code, specifically an object, should not reveal any of the internal workings as to how it specifically accomplishes a task. All an object ought to do is promise that certain methods will exist and perform a task as specified in an interface. This allows you to easily modify the core functionality later without breaking any code, which minimizes “churn”<sup>ix</sup>.

This is a great idea in principle, but in execution it can be both difficult and occasionally counterproductive. Thoughtless encapsulation does nothing to minimize churn while increasing complexity, so the thoughtful application of encapsulation was something that we kept in mind during development.

#### 5.1.4 PEP 8 code standard consistency

The PEP 8 style guide starts with a beautiful Ralph Waldo Emerson quote, which says “a foolish consistency is the hobgoblin of little minds.” The style guide continues to reinforce this point with the comment that “Consistency with this style guide is important. Consistency within a project is more important.” (van Rossum, Warsaw, & Coghlan, 2013) The sentiment here is like with most principles and guidelines, a style guide is only as valuable as the amount of thought that is applied when using it.

When we first wrote this codebase, we both used slightly different styles, and upon review, we decided that PEP 8 made the most sense. It is a well-known Python style. It was already built in to Pycharm as something to be highlighted, and it is simple to use. However, there were certain places where conventions were consistently ignored. For example, we stuck to a line length of 100 instead of 79 as that was well displayed on the monitors used at Angelo, Gordon.

## 5.2 What went right

---

Despite the focus on this project as a learning opportunity, we still wanted to create a product that they could be proud of and a unique opportunity presented by this project and this sponsor was that the completed software would immediately become an actual piece of the sponsor’s pipeline. This tool is and will continue to be actively used, and being able to provide that contribution was exciting. So, we also wanted to examine what was done right.

---

<sup>ix</sup> Churn is a popular software engineering term, which is basically, if I change this piece of code, how much stuff breaks because of that? How easy is code to modify and extend? Typically, good software practices attempt to minimize churn, but no churn at all is a pipedream.



### 5.2.1 Strategy design pattern

We used the strategy design pattern to great effect in two places in this project. The first use was for the dynamic columns that we used to provide support for multiple report templates. The second use was for the preprocessing pipeline that further streamlined different kinds of reporting. We had examined other approaches, but ultimately, this won out for the betterment of the project.

The process of selecting a design pattern was insightful as well. We leaned heavily on an online resource Sourcemaking (Shvets, Frey, & Pavlova, Strategy Design Pattern, 2017) for suggestions, and it was an invaluable development tool.

### 5.2.2 Graphical front end

The graphical user interface that we built out was not part of the original project description. It was a combination of the alacrity with which we finished the backend for the project and our enthusiasm to develop an interface that led to the GUI. Initially, we had floated the idea of developing a web app interface but the sponsor decided that the installable GUI was a more intuitive and useful interface.

Python is not the strongest language natively to develop a front end with. However, Python's strength is the breadth and power of the open source libraries available, so it was easy to find a module with strong graphical support.

Even though this part of the project was the least planned for aspect, it ultimately generated tremendous value. It was an exercise in demonstrating exactly what functionality we had built over the course of our project and inherently forced us to think visually about the workflow.

### 5.2.3 "Scrumban"

Part of what allowed us to develop this software so quickly was our application of different project management strategies. Our hybrid method of management, which we nicknamed "scrumban" was a combination of strategic and tactical planning. The "scrum" portion was tactical. It was how we managed the day-to-day objectives of our project. The "Kanban" portion was more strategic; it was how we delegated major goals and scheduled more long-term development. At the highest level of strategy was conversations with Mr. Burton, which helped us tie in the business goals and keep us centered on our ultimate goal: a simple more extensible codebase.

## 5.3 Takeaways

---

Software development is about so much more than being the best coder in the room. Good development sits at the confluence of so many factors: communication, coordination, strategic decision making, tactical problem solving, operations management, and design. If a team does not handle all these soft issues, there will never be time to dive into technical details.

This project was a phenomenal experience because it gave us the opportunity to not only see all these different aspects of development, but to also implement and run them. We were never given a bulleted list of exactly what to do, and Mr. Burton was not over our shoulders micro-managing every step. We were given a tremendous amount of responsibility over this project, which was a learning opportunity we could not have gotten anywhere else.

## 5.4 Recommendations

---

One of the major goals with this software was to make it extensible to future reporting needs. So, we hope to see that it gets used and reused in future reporting pipelines. To help support this goal, we recommend several things.

- **OpenPyXL:** This is a fantastic open source library but as we have noted in this paper, it has no fixed development. It is built on a piecemeal basis by a group of volunteer coders working on donations, but as new features are added they could obsolete some of the code we have built or even break code. Keeping a tag on OpenPyXL's development could save future headaches.
- **Template model:** The codebase is designed to be as flexible as reasonable with Excel reports that looked like an open or closed fund report. It is easy to tweak a widget here or add a column there, and we encourage that kind of reporting. Building an entire template from scratch would almost certainly be the least efficient way of using this codebase, though it is possible.



WPI



ANGELO,  
GORDON  
& CO.

## APPENDIX A: STRATEGY ENTRY

```
"""
This is the class definition for the StrategyEntry class.
The StrategyEntry contains all of the data necessary for generating the report and
some useful methods. It takes in a list of all the values read from the .csv file
and populates the StrategyEntry.
"""
from datetime import datetime
import logging

class StrategyEntry:
    def __init__(self, data_entry, keys):
        self.logger = logging.getLogger()
        self.data = {}
        for ind, key in enumerate(keys):
            self.data[key] = data_entry[ind]
        # GIC Defaults
        self.data["Deal"] = None
        self.data["Deal Type"] = None
        self.data["Country of Primary Business Risk"] = None
        self.data["GIC Sector"] = None
        self.data["GIC Industry"] = None
        self.data["Analyst(s)"] = None
        self.data["Publicly-Traded Equity"] = None
        self.data["Involvement with Company"] = None
        self.data["whitelist"] = []

    def __str__(self):
        return str(self.data)

    def __lt__(self, other):
        print("self: ", self.gross_profit(), "other: ", other.gross_profit())
        return self.gross_profit() < other.gross_profit()

    def calculate_bps(self, fund_size):
        """
        Calculates the strategies basis point (100ths of a percent) relative to the
        provided
        fund_size and adds it as a new data entry.
        :param fund_size: fund_size must be in the same units as gross profit (e.g. if
        gross profit
            is in millions fund_size should be too)
        """
        self.data["bps"] = abs((self.gross_profit() / fund_size) * 1e4)

    # PRIVATE
    def execute_override(self, workbook, override):
        """
        Applies overrides for the given override
        :param workbook: used to retrieve the sheet for app_logging
        :param override: the override that is related to this strategy
        :return: the overridden strategy
        """
```





# WPI



ANGELO,  
GORDON  
& CO.

```
"""
ws = workbook["OverrideLog"]
override_text = [self.strategy()]
if override.get("Show") == "n":
    # if we don't show this strategy, don't bother with the rest of the
filtering
    override_text.append("Don't show")
    ws.append(override_text)
    return None
elif override.get("Show") == "y":
    override_text.append("")
    if override.get("Begin Date") is not None:
        date = override.get("Begin Date")
        if isinstance(date, datetime):
            date = date.strftime("%m/%d/%Y")
        override_text.append("Begin Date overwritten from {0} "
            "to {1}".format(self.first_cashflow(), date))
        self.data["First Cashflow"] = date
    else:
        override_text.append("")
    if override.get("End Date") is not None:
        date = override.get("End Date")
        if isinstance(date, datetime):
            date = date.strftime("%m/%d/%Y")
        override_text.append("End Date overwritten from {0} "
            "to {1}".format(self.last_cashflow(), date))
        self.data["Last Cashflow"] = date
    else:
        override_text.append("")
    if override.get("Total Mkt Value") is not None: # Ask Scott about some of
these
        override_text.append("Total Mkt Value overwritten from {0} "
            "to {1}".format(self.remmv(),
                override.get("Total Mkt Value")))
        self.data["RemMV"] = override.get("Total Mkt Value")
    else:
        override_text.append("")
    if override.get("GrossIRR") is not None:
        override_text.append("GrossIRR overwritten from {0} "
            "to {1}".format(self.irr(),
override.get("GrossIRR")))
        self.data["Internal Rate of Return"] = override.get("GrossIRR")
    else:
        override_text.append("")
    ws.append(override_text)
    return self
else:
    self.logger.error("This override has no condition 'Show'")
    raise KeyError("This override has no condition 'Show'")

"""
All these getters are used to better support any changes to the input csv. In the
case of a
renamed column or added/removed column, the getters can be changed on a more
modular basis.
"""
```





# WPI



ANGELO,  
GORDON  
& CO.

```
def strategy(self):
    return self.data.get("Strategy")

def total_purchases(self):
    return float(self.data.get("TotalCost")) # Same as total cost apparently in
the C# code

def irr(self):
    return self.data.get("Internal Rate of Return")

def remmv(self):
    return float(self.data.get("RemMV"))

def gross_profit(self):
    return float(self.data.get("GrossProfit"))

def moic(self):
    return float(self.data.get("MOIC"))

def first_cashflow(self):
    return self.data.get("First Cashflow")

def last_cashflow(self):
    return self.data.get("Last Cashflow")

def bps(self):
    try:
        return float(self.data.get("bps"))
    except TypeError:
        self.logger.debug("bps was called before it was calculated")
        return None

def monetized(self):
    return self.remmv() == 0

def get_whitelist(self):
    return self.data.get("whitelist")

def get_attr_from_string(self, attribute_name):
    def switch(x):
        return {
            'bps': self.bps(),
            'monetized': self.monetized(),
            'last_cashflow': self.last_cashflow(),
            'first_cashflow': self.first_cashflow(),
            'moic': self.moic(),
            'gross_profit': self.gross_profit(),
            'remmv': self.remmv(),
            'irr': self.irr(),
            'strategy': self.strategy(),
            'total_purchases': self.total_purchases()
        }[x]

    try:
        return switch(attribute_name)
```



WPI



ANGELO,  
GORDON  
& CO.

```
except:
    self.logger.debug("{0} had no predefined getter, fetching directly from "
                      "dictionary".format(attribute_name))
    return self.data.get(attribute_name)

__repr__ = __str__
```



## APPENDIX B: COLUMN STRATEGY

```
"""
All column strategy and dynamic column creation classes
"""
import logging
from src.misc.utils import get_gross_profit_sum_from_list, conv_to_millions

class ColumnStrategyCreator:
    """
    Retrieves all strategies given the strategy name
    """

    def get_column_strategy(self, strategy_name):
        """
        Returns the correct strategy class based on the strategy provided
        :param strategy_name: strategy name, all strategies are found on the bottom
        :return:
        """
        class_ = STRATEGY_MAPPING[strategy_name]
        return class_

class ColumnStrategyAbs:
    """
    Abstract class for column strategies
    """

    def __init__(self, data, attribute_name=None):
        """
        :param attribute_name: extra variable, or attribute name
        """
        self.strategy_name = None # this should be changed
        self.attribute_name = attribute_name
        self.data = data

    def calculate(self):
        """
        This method is where the actual calculations should be performed to the data
        :param data: the list of Strategy points, ie our data
        :return: the new data set
        """
        pass

class DefaultStrategy(ColumnStrategyAbs):
    """
    The default strategy takes in an attribute name and
    returns a dataset with just the attribute name from the object
    """

    def __init__(self, data, attribute_name=None):
        super(DefaultStrategy, self).__init__(data, attribute_name)
        self.strategy_name = "default"
```



```
def calculate(self):
    """
    Grabs the attribute_name from the object and creates a new array
    :param data: the data set of StrategyEntry Objects
    :return:
    """
    new_data = []

    for data_point in self.data:
        attribute_obj = data_point
        new_data.append(attribute_obj.get_attr_from_string(self.attribute_name))
    return new_data

class FloatStrategy(ColumnStrategyAbs):
    """
    The default strategy takes in an attribute name and
    returns a dataset with just the attribute name from the object
    """

    def __init__(self, data, attribute_name=None):
        super(FloatStrategy, self).__init__(data, attribute_name)
        self.strategy_name = "float_default"

    def calculate(self):
        """
        Grabs the attribute_name from the object and creates a new array
        :param data: the data set of StrategyEntry Objects
        :return:
        """
        new_data = []
        logger = logging.getLogger()
        for data_point in self.data:
            attribute_obj = data_point
            val = attribute_obj.get_attr_from_string(self.attribute_name)
            try:
                val = float(val)
            except ValueError:
                logger.warning("Did not read a float. Read: {0} of type {1} "
                               "instead".format(val, type(val)))
            new_data.append(val)
        return new_data

class PercentChangeStrategy(ColumnStrategyAbs):
    """
    Calculates the percent change
    """

    def __init__(self, data, attribute_name=None):
        super(PercentChangeStrategy, self).__init__(data, attribute_name)
        self.strategy_name = "percent_change"

    def calculate(self):
        """
        Grabs the attribute_name from the object and creates a new array

```



```
        :param data: the data set of StrategyEntry Objects
        :return:
        """
        new_data = []
        total_gross_profit_sum = get_gross_profit_sum_from_list(self.data)
        for data_point in self.data:
            gross_profit = data_point.gross_profit()
            percentage = abs(gross_profit / total_gross_profit_sum)
            new_data.append(percentage)
        return new_data

class BPPStrategyRegular(ColumnStrategyAbs):
    """
    This strategy determines what category the bpp is
    the attribute it takes in the category either cat1, cat2, or cat3
    """

    def __init__(self, data, attribute_name):
        super(BPPStrategyRegular, self).__init__(data, attribute_name)
        self.cat1 = 0
        self.cat2 = 50
        self.cat3 = 100
        self.strategy_name = "bpp"

    def check_what_category_bps_is(self, bps):
        """
        Checks what category the bps is
        :param bps: the bps
        :return:
        """
        category = None
        if self.cat1 <= bps < self.cat2:
            category = "cat1"
        elif self.cat2 <= bps < self.cat3:
            category = "cat2"
        elif bps > self.cat3:
            category = "cat3"
        return category

    def calculate(self):
        new_data = []
        for data_point in self.data:
            bps = data_point.bps()
            if self.check_what_category_bps_is(bps) == self.attribute_name:
                new_data.append(data_point.gross_profit())
            else:
                new_data.append(None)
        return new_data

class AttributeInMillions(ColumnStrategyAbs):
    """
    Strategy to convert an attribute to millions
    """
```



```
def __init__(self, data, attribute_name):
    super(AttributeInMillions, self).__init__(data, attribute_name)
    self.strategy_name = "in_millions"

def calculate(self):
    new_data = []
    for data_point in self.data:
        attribute_obj = data_point
        val = attribute_obj.get_attr_from_string(self.attribute_name)
        new_data.append(conv_to_millions(val))
    return new_data

class BPPStrategy(ColumnStrategyAbs):
    """
    This strategy determines what category the bpp is
    the attribute it takes in the category either cat1, cat2, or cat3
    """

    def __init__(self, data, attribute_name):
        super(BPPStrategy, self).__init__(data, attribute_name)
        self.cat1 = 0
        self.cat2 = 50
        self.cat3 = 100
        self.strategy_name = "bpp_in_mills"

    def check_what_category_bps_is(self, bps):
        """
        Checks what category the bps is
        :param bps: bps of the data object
        :return: category string
        """
        category = None
        if self.cat1 <= bps < self.cat2:
            category = "cat1"
        elif self.cat2 <= bps < self.cat3:
            category = "cat2"
        elif bps >= self.cat3:
            category = "cat3"
        return category

    def calculate(self):
        new_data = []
        for data_point in self.data:
            bps = data_point.bps()
            if self.check_what_category_bps_is(bps) == self.attribute_name:
                gross_profit = data_point.gross_profit()
                gross_profit_in_mills = conv_to_millions(gross_profit)
                new_data.append(gross_profit_in_mills)
            else:
                new_data.append(None)
        return new_data

class PercentageBucketingStrategy(ColumnStrategyAbs):
    """
```



```
This strategy determines what category the bpp is
the attribute it takes in the category either cat1, cat2, or cat3
"""

def __init__(self, data, attribute_name):
    super(PercentageBucketingStrategy, self).__init__(data, attribute_name)
    self.cat1 = 25
    self.cat2 = 50
    self.cat3 = 75
    self.cat4 = 100
    self.strategy_name = "percentage_bucketing"

def find_max_gross_profit(self):
    """
    Finds the max gross profit from the entire data object list
    :return: float
    """
    return max([abs(i.gross_profit()) for i in self.data])

def check_what_category_percentage_is(self, percentage):
    """
    Check the percentage category for bucketing purposes
    :param percentage: percentage metric
    :return: bucketing string
    """
    category = None
    if 0 < percentage <= self.cat1:
        category = "cat1"
    elif self.cat1 < percentage <= self.cat2:
        category = "cat2"
    elif self.cat2 < percentage <= self.cat3:
        category = "cat3"
    elif percentage >= self.cat3:
        category = "cat4"
    return category

def calculate(self):
    """
    Calculates the percentage bucketing
    :return:
    """
    new_data = []
    max_gross_profit = self.find_max_gross_profit()
    for data_point in self.data:
        gross_profit = data_point.gross_profit()
        percentage = abs(gross_profit / max_gross_profit) * 100
        if self.check_what_category_percentage_is(percentage) ==
self.attribute_name:
            new_data.append(conv_to_millions(gross_profit))
        else:
            new_data.append(None)
    return new_data

class MonetizedStrategy(ColumnStrategyAbs):
    """
```



```
Strategy to determine if the strategy is monetized or ongoing
"""

def __init__(self, data, attribute_name=None):
    super(MonetizedStrategy, self).__init__(data, attribute_name)
    self.strategy_name = "monetized"

def calculate(self):
    """
    Checks if the strategy is monetized or ongoing
    :return:
    """
    new_data = []
    for data_point in self.data:
        new_data.append("Monetized" if data_point.monetized() else "Ongoing")
    return new_data

class CountStrategy(ColumnStrategyAbs):
    """
    Counts the data set and returns a list of enumeration -> [1,2,3,4,5...n]
    """

    def __init__(self, data, attribute_name=None):
        super(CountStrategy, self).__init__(data, attribute_name)
        self.strategy_name = "count"

    def calculate(self):
        new_data = []
        for ind, data in enumerate(self.data):
            new_data.append(ind + 1)
        return new_data

STRATEGY_MAPPING = {
    "default": DefaultStrategy,
    "percent_change": PercentChangeStrategy,
    "bpp_in_mills": BPPStrategy,
    "monetized": MonetizedStrategy,
    "in_millions": AttributeInMillions,
    "count": CountStrategy,
    "percentage_bucketing": PercentageBucketingStrategy,
    "float_default": FloatStrategy,
}
```





# WPI



ANGELO,  
GORDON  
& CO.

## APPENDIX C: START MAIN REPORT

```
def start_main_report(input_name, fund_name=None, verbose=False, save_path=None,
                      report_type="closed", template_path=None, starting_date=None):
    """
    Kicks off the main report and handles a lot of the business logic as to how
    various pieces of
    code should be called
    :param input_name: (string) the file name of the input .csv to be parsed. Can be
    relative or
    absolute path
    :param fund_name: (string) the 4 diGIT codename for the fund. Supported funds are
    in the config
    file in the FUND_AMTS dictionary. Default is parsed from filename
    :param verbose: (boolean) if True sends email to addresses specified in
    config.EMAIL_TO_ADDRESS
    :param save_path: (string) the relative path name for the file to be saved. If
    None, uses
    default format
    :param report_type: (string) the type of report to be generated.
    :param template_path: (string) the relative path to the excel template to be used.
    Must match
    the type specified.
    :param starting_date: default is to parse from input file, if set by user expects
    a datetime obj
    """
    file_logger = configure_file_logging()
    stdout_logger = get_stdout_logger()
    # Log warnings from opening workbook
    template_path_relative = os.path.join(config.BASE_PROJECT_DIR, template_path)
    with warnings.catch_warnings(record=True) as w:
        # initialize and read the workbook
        stdout_logger.info("Loading template {0}...".format(template_path_relative))
        workbook = OpenPyXL.load_workbook(template_path_relative)
    for warning in w:
        file_logger.warning(warning.message)
    stdout_logger.info("Done.")
    workbook.active = 0
    winners_sheet = workbook.active
    # Parse input filename
    input_name_file = os.path.basename(input_name)
    temp_fund, temp_date = parse_filename(input_name_file)
    if fund_name is None:
        fund_name = temp_fund
    if starting_date is None:
        starting_date = temp_date
    if save_path is None:
        save_path = config.TEMP_DIR_OUTPUT_PATH
    stdout_logger.info("Collecting information from {0} and reading overrides from "
                      "template...".format(input_name))
    legal_name = LegalName(fund_name).perform_query_single()
    load_in_name_date(workbook, name=legal_name, date=starting_date)
    # Open and load in data from input file
    array_data = read_csv_file(input_name)
    # Build StrategyEntry's from the input data
    keys = array_data[0]
```



# WPI



ANGELO,  
GORDON  
& CO.

```
strategies, strategy_codes = generate_data_points(array_data, keys)
output_strategies, strategy_codes = generate_data_points(array_data, keys)
overrides = generate_override_dictionary(workbook, fund_name)
# This is all the extra information that the preprocessors *might* need
extra_attributes = {"workbook": workbook,
                   "overrides": overrides,
                   "keys": keys,
                   "codes": strategy_codes,
                   "fund_size": get_fund_from_template(workbook, fund_name)}

stdout_logger.info("Done.")
stdout_logger.info("Processing data and populating worksheet...")
populate_winners_losers_worksheet(workbook, extra_attributes, winners_sheet,
strategies, fund_name,
                                report_type=report_type)

populate_output_worksheet(workbook, extra_attributes, output_strategies)
stdout_logger.info("Done.")
# Only select main page
workbook.active = 0
# format save path
save_path_excel = os.path.join(save_path, fund_name + "_Deal_Analysis_" \
                                + starting_date.strftime("%Y%m%d") + ".xlsx")

try:
    stdout_logger.info("Saving file to {0}...".format(save_path_excel))
    workbook.save(save_path_excel)
except PermissionError:
    file_logger.error(save_path_excel + " is open already in another editor
(likely Excel).")
    stdout_logger.error(save_path_excel + " is open already in another editor
(likely Excel).")
    stdout_logger.error(save_path_excel + " Please close this file and try again.")
    stdout_logger.error(save_path_excel + " is open already in another editor
(likely Excel).")
    stdout_logger.error(save_path_excel + " Please close this file and try
again.")

workbook.close()
return 0

workbook.close()
stdout_logger.info("Done.")
# excel_file = os.path.abspath(save_path)
stdout_logger.info("Converting file to .pdf...")
pdf_file = convert_excel_to_pdf(save_path_excel, save_path)
stdout_logger.info("Done.")
if verbose:
    stdout_logger.info("Sending files to {0}...".format(config.EMAIL_TO_ADDRESS))
    send_winners_losers_report(fund_name=fund_name, files=[pdf_file,
save_path_excel],
                              report_date=starting_date)

stdout_logger.info("Done.")
stdout_logger.info("Report generation done.")
```



## APPENDIX D: GRAPHICAL USER INTERFACE (GUI)

```
"""
Main view to encapsulate and build the view
"""
import ctypes
import os

from PyQt5 import QtWidgets, QtCore, QtGui

import config
from gui.utils import GUI_DIR, open_file_location_dir
from gui.view.designer.ui_main_window import Ui_MainWindow
from gui.view.main_form_view import FormView
from gui.view.readme_view import ReadmeWidgetView
from gui.view.regression_view import RegressionWidgetView
from gui.view.settings_view import SettingsWidgetView

class MainView(QtWidgets.QMainWindow):
    """
    Main application view which initializes all other views
    """
    def __init__(self, model, main_ctrl, app=None):
        self.model = model
        self.main_ctrl = main_ctrl
        self.app = app
        super(MainView, self).__init__(None, QtCore.Qt.FramelessWindowHint)
        # super(MainView, self).__init__(None)

        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.setFixedSize(self.size())
        self.gui_properties()
        # mouse events set up
        self.oldPos = self.pos()
        # separate view objects from the main view
        # define all views before show
        self.form_view = FormView(main_view=self, model=self.model)
        self.connect_menu_events()
        self.show()

    def gui_properties(self):
        myappid = u'angelogordon.risk.distressed.1.0' # arbitrary string
        ctypes.windll.shell32.SetCurrentProcessExplicitAppUserModelID(myappid)
        # icon for taskbar
        icon_path = os.path.join(GUI_DIR, "resources/resized_logo.png")
        self.setWindowIcon(QtGui.QIcon(icon_path))

    def mousePressEvent(self, event):
        self.oldPos = event.globalPos()

    def mouseMoveEvent(self, event):
        delta = QtCore.QPoint(event.globalPos() - self.oldPos)
        self.move(self.x() + delta.x(), self.y() + delta.y())
        self.oldPos = event.globalPos()
```



# WPI



ANGELO,  
GORDON  
& CO.

```
def connect_menu_events(self):
    self.ui.actionReadme.triggered.connect(self.readme_action)
    self.ui.actionReadme.setShortcut("Ctrl+r")
    self.ui.actionSettings.triggered.connect(self.settings_action)
    self.ui.actionSettings.setShortcut("Ctrl+s")
    self.ui.actionLogs.triggered.connect(self.open_logging_dir_action)
    self.ui.actionLogs.setShortcut("Ctrl+l")
    self.ui.actionClose.triggered.connect(self.exit_action)
    self.ui.actionClose.setShortcut(QtGui.QKeySequence(QtCore.Qt.Key_Escape))
    self.ui.actionMinimize.triggered.connect(self.showMinimized)
    self.ui.actionMinimize.setShortcut("Ctrl+m")
    self.ui.actionRegression_Testing.triggered.connect(self.regression_action)
    self.ui.actionRegression_Testing.setShortcut("Ctrl+t")

def exit_action(self):
    self.app.quit()

def readme_action(self):
    readme_widget = ReadmeWidgetView(self)
    readme_widget.open_dialog()

def settings_action(self):
    settings_widget = SettingsWidgetView(self)
    settings_widget.open_dialog()

def regression_action(self):
    RegressionWidgetView(self).open_dialog()

def open_logging_dir_action(self):
    log_dir = config.LOGGING_PATH
    open_file_location_dir(log_dir)
```

```
"""
This file will hold all application wide data.
Our "model"
"""
from src.misc.templates import get_template_based_on_input
import copy

class Model(object):
    """
    Taken from https://stackoverflow.com/questions/26698628/mvc-design-with-gtdesigner-and-pyside

    This is our model throughout the application, which contains all application
    settings.

    The data is held in the data variable
    After each change, the state is captured
    """

    def __init__(self):
```



# WPI



ANGELO,  
GORDON  
& CO.

```
self._update_funcs = []
self.states = []

# variable placeholders
self.data = {
    "geneva_file_name": "",
    "output_file_name": "",
    "template_type": "closed",
    "fund_name": "",
    "period_end_date": "",
    "send_email": True,
}
self.update_data_refs(self.data)
self.add_state(self.data)

def update_data_refs(self, data_to_update):
    """
    Updates any dependent data references
    :param data_to_update: the copy of the state to update
    :return: None
    """
    data_to_update["template_path"] =
get_template_based_on_input(data_to_update["template_type"])

def add_state(self, data):
    """
    Inserts data into the first index
    :param data: the data to insert into state
    :return: None
    """
    self.states.insert(0, data)

def get_state_head(self):
    """
    Gets the head
    :return: dict
    """
    return self.states[0]

def get_state_tail(self):
    """
    Gets the tail
    :return: dict
    """
    length = len(self.states) - 1
    return self.states[length]

def set_data(self, key, val):
    """
    Sets the data, updates state, and announces update to any
    of the callbacks
    :param key: the key of the dataset
    :param val: the value of the dataset
    :return: None
    """
    data_cp = copy.deepcopy(self.get_state_head())
```



```
try:
    data_cp[key] = val
except Exception as e:
    print(e)
self.update_data_refs(data_cp)
self.add_state(data_cp)
self.announce_update()

def get_most_recent_state_data_w_key(self, key):
    """
    Gets the most recent state
    :param key: key
    :return:
    """
    data = self.get_state_head()
    return data[key]

def get_data(self):
    """
    Returns data object
    :return:
    """
    return self.data

# subscribe a view method for updating
def subscribe_update_func(self, func):
    """
    Subscribes a function to whenever data is updated
    :param func: call back function
    :return:
    """
    if func not in self._update_funcs:
        self._update_funcs.append(func)

# unsubscribe a view method for updating
def unsubscribe_update_func(self, func):
    """
    Unsubscribes function from announcement
    :param func: callback function
    :return:
    """
    if func in self._update_funcs:
        self._update_funcs.remove(func)

# update registered view methods
def announce_update(self):
    """
    Announces there is a data update to any functions
    :return:
    """
    for func in self._update_funcs:
        func()

def reset(self):
    """
    Sets the most recent state to the default, and
```



# WPI



ANGELO,  
GORDON  
& CO.

```
announces update
:return:
"""
self.add_state(self.get_state_tail())
self.announce update()
```

```
"""
Controller for the main form view
"""
import os
from PyQt5.QtCore import QThread
from PyQt5 import QtCore
import config
import logging
from gui.controller.main_text_logger import QTHandler
from src.kick_off_pipeline import parse_geneva_filename
from src.kick_off_pipeline import kick_off_pipeline_call
from src.app_logging.configure_logging import configure_stdout_logging

class GenerateReportThread(QThread):
    """
    Generate report thread encapsulation
    """
    signalStatus = QtCore.pyqtSignal(str)

    def __init__(self, geneva_input=None, template_type=None,
                 verbose=False, output_dir=None, parent=None):
        """
        :param geneva_input: Geneva file input
        :param template_type: the template type to use
        :param verbose: send email
        :param output_dir: output directory of output files
        :param parent: parent thread
        """
        super(GenerateReportThread, self).__init__(parent)
        self.geneva_input = geneva_input
        self.template_type = template_type
        self.verbose = verbose
        self.output_dir = output_dir
        self.handler = QTHandler(self.signalStatus)
        formatter = logging.Formatter('%(asctime)s %(levelname)s %(message)s\n',
datefmt="%H:%M:%S")
        self.handler.setFormatter(formatter)
        self.logger = configure_stdout_logging()
        self.logger.addHandler(self.handler)

    @QtCore.pyqtSlot()
    def run(self):
        """
        The process which the thread encapsulates
        :return: None
        """
        kick_off_pipeline_call(geneva_input=self.geneva_input,
```



# WPI



ANGELO,  
GORDON  
& CO.

```
template_type=self.template_type,
                    verbose=self.verbose, output_dir=self.output_dir)

class MainFormController(object):
    """
    Controller to handle the form view
    """
    def __init__(self, model, view):
        self.model = model
        self.view = view
        self.worker = GenerateReportThread()

    def get_template_type_mapping(self):
        """
        Returns template type mapping keys
        :return: keys
        """
        mapping = config.TEMPLATE_TYPES
        return list(mapping.keys()), mapping

    def generate_report(self):
        """
        Kicks off the report generation thread configured based on all the preset
        values.
        It expects the call from the view to both handle any exceptions and do the
        validation
        """
        geneva_input = self.get_geneva_file()
        template_type = self.get_template_type()
        verbose = self.get_send_email_status()
        output_dir = self.get_output_dir()
        self.worker.geneva_input = geneva_input
        self.worker.template_type = template_type
        self.worker.verbose = verbose
        self.worker.output_dir = output_dir

        self.worker.start()

    def parse_date_and_fund_name(self, filename):
        """
        Parses the filename for the data and fund name
        :param filename: filename of the
        :return: fund_name, month_end_str, period_end_date, knowledge_date
        """
        return parse_geneva_filename(filename=filename)

    def get_attr_from_model_html(self, attr):
        val = self.model.get_most_recent_state_data_w_key(attr)
        if not val or val == "":
            val = "none"
        html = "<strong><span>{0}</span></strong>".format(val)
        return html

    def get_preview_html(self):
        html = ""
```





```
<div>
<p>Geneva File: {0}</p>
<p>Output dir: {1}</p>
<p>Template type: {2}</p>
<p>Template File: {3}</p>
<p>Fund Name: {4}</p>
<p>Period End date: {5}</p>
<p>Send email: {6}</p>
</div>
"".format(self.get_attr_from_model_html("geneva_file_name"),
self.get_attr_from_model_html("output_file_name"),
        self.get_attr_from_model_html("template_type"),
self.get_attr_from_model_html("template_path"),
        self.get_attr_from_model_html("fund_name"),
self.get_attr_from_model_html("period_end_date"),
        self.get_attr_from_model_html("send_email"))

    return html

def reset_model_to_default(self):
    self.model.reset()

def set_geneva_file(self, file_name):
    self.model.set_data("geneva_file_name", file_name)

def set_output_save_file_location(self, file_name):
    self.model.set_data("output_file_name", file_name)

def get_geneva_file(self):
    return self.model.get_most_recent_state_data_w_key("geneva_file_name")

def get_output_dir(self):
    dirname =
os.path.abspath(self.model.get_most_recent_state_data_w_key("output_file_name"))
    return dirname

def set_fund_name(self, fund_name):
    self.model.set_data("fund_name", fund_name)

def get_fund_name(self):
    return self.model.get_most_recent_state_data_w_key("fund_name")

def set_period_end_date(self, period_end_date):
    self.model.set_data("period_end_date", period_end_date)

def get_period_end_date(self):
    return self.model.get_most_recent_state_data_w_key("period_end_date")

def set_send_email_status(self, status):
    self.model.set_data("send_email", status)

def get_send_email_status(self):
    return self.model.get_most_recent_state_data_w_key("send_email")

def set_template_type(self, type):
    self.model.set_data("template type", type)
```



**WPI**



ANGELO,  
GORDON  
& CO.

```
def get_template_type(self):  
    return self.model.get_most_recent_state_data_w_key("template_type")
```



## APPENDIX E: UNIT AND REGRESSION TESTING

```
import os
from unittest import TestCase
import OpenPyXL
import winners_losers_closed_fund
import config
from src.app_logging.configure_logging import configure_file_logging
from src.excel.comparison_tools import cmp_serialisable

configure_file_logging()

class TestRegression(TestCase):
    """
    Uses a pre-generated Winners & Losers report from a fixed data set to compare
    against the
    output report from the current codebase and compares errors
    """
    def setUp(self):
        expected = os.path.join(config.TEST_DATA_PATH,
"CRP6_Deal_Analysis_20170930_expected.xlsx")
        print(expected)
        self.expected_closed_output = OpenPyXL.load_workbook(expected)
        self.input = os.path.join(config.TEST_DATA_PATH,
"NewWLTotals_CRP6_20170930.csv")
        self.actual_closed_output = os.path.join(config.TEMP_DIR_OUTPUT_PATH,
"CRP6_Deal_Analysis_20170930.xlsx")

    def tearDown(self):
        self.expected_closed_output.close()
        self.expected_closed_output = None

    def check_cells(self, expected, actual):
        """
        Checks all the cells in the current worksheet by comparing all the style
        objects as
        Serialisable objects.
        :param expected: the expected worksheet object
        :param actual: the actual worksheet object
        :return:
        """
        for ex_row, ac_row in zip(expected.iter_rows(), actual.iter_rows()):
            for ex_cell, ac_cell in zip(ex_row, ac_row):
                self.assertEqual(ex_cell.coordinate, ac_cell.coordinate)
                self.assertEqual(ex_cell.value, ac_cell.value)
                self.assertEqual(ex_cell.number_format, ac_cell.number_format)
                self.assertTrue(cmp_serialisable(ex_cell.font, ac_cell.font))
                self.assertTrue(cmp_serialisable(ex_cell.fill, ac_cell.fill))
                self.assertTrue(cmp_serialisable(ex_cell.border, ac_cell.border))
                self.assertTrue(cmp_serialisable(ex_cell.alignment,
                                                    ac_cell.alignment))

    def check_ranges(self, expected, actual):
        """
        Checks the defined names as Serialisable objects
        """
```



```
    :param expected: list of DefinedNames
    :param actual: list of DefinedNames
    :return:
    """
    for ex_defn, ac_defn in zip(expected, actual):
        self.assertTrue(cmp_serialisable(ex_defn, ac_defn))

    def check_images(self, expected, actual):
        """
        Currently just checks that the expected number of images are there, as
        comparing images
        is beyond the scope of this test
        :param expected: list of expected images
        :param actual: list of actual images
        :return:
        """
        self.assertEqual(len(expected), len(actual))

    def check_tables(self, expected, actual):
        """
        Checks the tables as Serialisable objects
        :param expected: list of Tables
        :param actual: list of Tables
        :return:
        """
        for ex_table, ac_table in zip(expected, actual):
            self.assertTrue(cmp_serialisable(ex_table, ac_table))

    def check_sheets(self, expected, actual):
        """
        Checks all the sheets in the current workbook, failing if they don't match in
        name, and
        calling check_cells, check_images, and check_tables to check all of those sub-
        objects
        recursively.
        :param expected: the expected workbook object
        :param actual: the actual workbook object
        :return:
        """

        for ex_sheet, ac_sheet in zip(expected._sheets, actual._sheets):
            try:
                self.assertEqual(ac_sheet.title, ex_sheet.title)
                self.assertEqual(ac_sheet.max_row, ex_sheet.max_row)
                self.assertEqual(ac_sheet.min_row, ex_sheet.min_row)
                self.assertEqual(ac_sheet.max_column, ex_sheet.max_column)
                self.assertEqual(ac_sheet.min_column, ex_sheet.min_column)
                self.check_cells(ex_sheet, ac_sheet)
                self.check_images(ex_sheet._images, ac_sheet._images)
                self.check_tables(ex_sheet._tables, ac_sheet._tables)
            except AssertionError as e:
                print("ac_sheet: {0} ex_sheet: {1}".format(ac_sheet.title,
ex_sheet.title))
                raise e

    def test_closed_fund(self):
```



# WPI



ANGELO,  
GORDON  
& CO.

```
winner_losers_closed_fund.main(self.input)
actual = OpenPyXL.load_workbook(self.actual_closed_output)
self.check_sheets(self.expected_closed_output, actual)
self.check_ranges(self.expected_closed_output.defined_names.definedName,
                  actual.defined_names.definedName)

actual.close()
```

```
import os
import OpenPyXL
from OpenPyXL.worksheet.table import Table
from unittest import TestCase
from src.excel import table_tools
from src.app_logging.configure_logging import configure_file_logging
import config

configure_file_logging()
TEST_TEMPLATE_CLOSED_FUND = os.path.join(config.BASE_PROJECT_DIR,
"templates\winner_losers_template_closed_fund.xlsx")

class TestTableTools(TestCase):
    """
    Testing the Excel table manipulation tools
    """

    def setUp(self):
        self.workbook = OpenPyXL.load_workbook(TEST_TEMPLATE_CLOSED_FUND)
        self.save_name = "test_table_tools_workbook.xlsx"
        self.workbook.create_sheet("Test_Sheet")
        self.worksheet = self.workbook["Test_Sheet"]
        table = Table(displayName="Test_Table", ref="A1:E5")
        self.worksheet.add_table(table)
        self.workbook.save(self.save_name)
        self.workbook = OpenPyXL.load_workbook(self.save_name)

    def tearDown(self):
        self.workbook.close()
        self.workbook = None
        if os.path.exists(self.save_name):
            os.remove(self.save_name)

    def test_write_to_cell(self):
        val = "Artichoke"
        coord = "A5"
        table_tools.write_to_cell(self.workbook, coord, val,
sheet=self.workbook["Test_Sheet"])
        self.workbook.save(self.save_name)
        self.workbook = OpenPyXL.load_workbook(self.save_name)
        self.assertEqual(self.workbook["Test_Sheet"][coord].value, val)

    def test_get_table_row(self):
        self.assertEqual(table_tools.get_table_row(self.workbook, "Test_Table"), 1)

    def test_break_apart_range(self):
        simple_range = "A5:B11"
```



```
spec_character_range = "$A$5:$B$11"
long_range = "AAA21:ADB44"
all_range = "$AAA$21:$ADB$44"

self.assertEqual(table_tools.break_apart_range(simple_range), [{"A", 5}, {"B",
11}])
self.assertEqual(table_tools.break_apart_range(spec_character_range), [{"A",
5}, {"B", 11}])
self.assertEqual(table_tools.break_apart_range(long_range), [{"AAA", 21},
["ADB", 44]])
self.assertEqual(table_tools.break_apart_range(all_range), [{"AAA", 21},
["ADB", 44]])

def test_find_table(self):
    try:
        table_tools.find_table(self.workbook, "Test_Table")
    except KeyError:
        self.fail("Could not find test table")

def get_named_range_coords(self, name):
    """
    Takes in a name and tries to find it. Fails if it can't locate the named_range
    or if the
    named range is not a cell
    :param name: the name of the range
    :return: the coords and the worksheet
    """
    names = [defn.name for defn in self.workbook.defined_names.definedName]
    if name not in names:
        self.fail("Can't find {0}".format(name))
    defn = self.workbook.defined_names[name]
    count = 0
    for title, coord in defn.destinations:
        if count > 1:
            self.fail("This named range is more than one cell")
        worksheet = self.workbook[title]
        count += 1
        coord = coord.replace("$", "")

    return coord, worksheet

def test_shift_named_cell(self):
    # Check that the named test cell exists
    test_name = "FUND_CELL"
    change = 10
    coord, worksheet = self.get_named_range_coords(test_name)
    base_col, base_row = table_tools.get_col_row(coord)
    table_tools.shift_named_cell(self.workbook, test_name, 0, change, worksheet)
    self.workbook.save(self.save_name)
    self.workbook = OpenPyXL.load_workbook(self.save_name)
    coord, worksheet = self.get_named_range_coords(test_name)
    new_col, new_row = table_tools.get_col_row(coord)
    if new_row != base_row + change:
        self.fail("Failed to add {0} to row".format(change))
    table_tools.shift_named_cell(self.workbook, test_name, change, 0, worksheet)
    self.workbook.save(self.save_name)
```



```
self.workbook = OpenPyXL.load_workbook(self.save_name)
coord, worksheet = self.get_named_range_coords(test_name)
new_col, new_row = table_tools.get_col_row(coord)
if table_tools.get_col_num(new_col) != table_tools.get_col_num(base_col) +
change:
    self.fail("Failed to add {0} to col".format(change))
    table_tools.shift_named_cell(self.workbook, test_name, 0, -change, worksheet)
    self.workbook.save(self.save_name)
    self.workbook = OpenPyXL.load_workbook(self.save_name)
    coord, worksheet = self.get_named_range_coords(test_name)
    new_col, new_row = table_tools.get_col_row(coord)
    if new_row != base_row:
        self.fail("Failed to remove {0} from row".format(change))
        table_tools.shift_named_cell(self.workbook, test_name, -change, 0, worksheet)
        self.workbook.save(self.save_name)
        self.workbook = OpenPyXL.load_workbook(self.save_name)
        coord, worksheet = self.get_named_range_coords(test_name)
        new_col, new_row = table_tools.get_col_row(coord)
        if new_col != base_col:
            self.fail("Failed to remove {0} from col".format(change))
            table_tools.shift_named_cell(self.workbook, test_name, change, change,
worksheet)
        self.workbook.save(self.save_name)
        self.workbook = OpenPyXL.load_workbook(self.save_name)
        coord, worksheet = self.get_named_range_coords(test_name)
        new_col, new_row = table_tools.get_col_row(coord)
        if table_tools.get_col_num(new_col) != table_tools.get_col_num(base_col) +
change \
            and new_row != base_row + change:
            self.fail("Failed to add to both col and row")
```

```
import os
from unittest import TestCase

from src.column.column_generate import Column
from src.excel.table_tools import get_table_row
from src.app_logging.configure_logging import configure_file_logging
from src.misc.utils import read_csv_file
import config
from winners_losers_main import generate_data_points
import OpenPyXL

TEST_TEMPLATE_CLOSED_FUND = os.path.join(config.BASE_PROJECT_DIR,
"templates\winnerslosertemplate_closed_fund.xlsx")
TEST_INPUT_FILE = os.path.join(config.TEST_DATA_PATH, "NewWLTotals_CRP6_20170930.csv")

configure_file_logging()

class TestColumn(TestCase):
    """
    Tests the Column class located in src/column/column_generate
    """
```



```
@classmethod
def setUp(cls):
    print(TEST_TEMPLATE_CLOSED_FUND)
    cls._workbook = OpenPyXL.load_workbook(TEST_TEMPLATE_CLOSED_FUND)
    cls._workbook.active = 0
    cls._winners_sheet = cls._workbook.active
    array_data = read_csv_file(TEST_INPUT_FILE)
    keys = array_data[0]
    cls._data_points = generate_data_points(array_data=array_data, keys=keys)[0]
    cls._starting_row = get_table_row(cls._workbook, "OutputTable")
    cls._column = Column(data_container=cls._data_points,
starting_row=cls._starting_row,
                        attribute_name="GrossProfit",
                        col_coord="A1", column_name="Gross Profit",
strategy_name="default",
                        workbook=cls._workbook)

def tearDown(self):
    self._workbook.close()

def test_populate_data(self):
    """
    Test population of values in the excell sheet with gross profit
    :return: None
    """
    self._column.populate_data()
    max_row = self._winners_sheet.max_row
    cell_arr = self._winners_sheet[self._starting_row:max_row]
    cell_arr_col_1 = [i[0] for i in cell_arr]
    cell_arr_gp = [float(i.value) for i in cell_arr_col_1]
    correct_solution = [i.gross_profit() for i in self._data_points]
    self.assertEqual(cell_arr_gp, correct_solution)

def test_populate_data_whitelist(self):
    """
    Test whitelist data, meaning exlcusion of certain data types
    :return:
    """
    # check whitelist data works
    # remove gross profit as whitelist, the return of populate data should be
empty
    new_data_container = []
    for i in self._data_points:
        i.data["whitelist"] = ["None"]
        new_data_container.append(i)

    column = Column(data_container=new_data_container,
starting_row=self._starting_row,
                    attribute_name="GrossProfit",
                    col_coord="A1", column_name="Gross Profit",
strategy_name="default", workbook=self._workbook)
    column.populate_data()
    max_row = self._winners_sheet.max_row
    cell_arr = self._winners_sheet[self._starting_row:max_row]
    cell_arr_col_1 = [i[0] for i in cell_arr]
    cell_arr_gp = [i.value for i in cell_arr_col_1]
```





WPI



ANGELO,  
GORDON  
& CO.

```
finalarr = []
for i in cell_arr_gp:
    if not i:
        continue

    if i.isdiGIT():
        finalarr.append(i)

self.assertEqual(finalarr, [])

def test_get_row_position(self):
    self.test_populate_data()
    self.assertEqual(self._column.get_row_position(), len(self._data_points) +
self._starting_row)
```

## Works Cited

- Angelo, Gordon, & Co. (2018, January 2). *The case for alternatives*. Retrieved from Angelo, Gordon, & Co: [https://www.angelogordon.com/why\\_alternatives.aspx](https://www.angelogordon.com/why_alternatives.aspx)
- Bolton, D. (2017, July 20). *What is an enum?* Retrieved from Thoughtco.com: <https://www.thoughtco.com/what-is-an-enum-958326>
- Boyd, B. (2001). Closed-end funds open some new possibility. *Puget Sound Business Journal*, 41.
- Clark, C., & Dallas. (2013, July). *Insert row into Excel spreadsheet using openpyxl in Python*. Retrieved from Stackoverflow: <https://stackoverflow.com/questions/17299364/insert-row-into-excel-spreadsheet-using-openpyxl-in-python>
- Clark, C., & Gazoni, E. (2017, November 29). *A Python library to read/write Excel 2010 xlsx/xlsm files*. Retrieved from Openpyxl: <https://openpyxl.readthedocs.io/en/default/>
- Gruber, J. (2004, December 17). *Markdown*. Retrieved from Daring Fireball: <https://daringfireball.net/projects/markdown/>
- Investopedia. (2017, December 4). *Basis Point (BPS)*. Retrieved from Investopedia.com: <https://www.investopedia.com/terms/b/basispoint.asp>
- Investopedia. (2017, November 17). *Profit and Loss Statement (P&L)*. Retrieved from Investopedia.com: <https://www.investopedia.com/terms/p/plstatement.asp>
- James, M. (2017, December 7). *Learn Scrum*. Retrieved from Scrum Methodolgy: <http://scrummethodology.com/>
- Kaufman, G. G. (2000). Banking and currency crises and systemic risk: A taxonomy and review. *Financial Markets, Institutions & Instruments*, 76.
- Law, J. (2015). *A Dictionary of Finance and Banking*. Oxford University Press.
- Lopez, F., & Romero, V. (2014). *Mastering Python Regular Expressions*. Packt Publishing Ltd.
- Myers, G. J., Sandler, C., & Badgett, T. (2011). *The Art of Software Testing*. Hoboken: John Wiley & Sons.
- NUMFocus. (2017, November 29). *pandas: powerful Python data analysis toolkit*. Retrieved from pandas.pydata.org: <http://pandas.pydata.org/pandas-docs/stable/>



- NumPy developrs. (2017, November 29). *NumPy*. Retrieved from Numpy.org: <http://www.numpy.org/>
- Qureshi, R. J., & Sabir, F. (2013). A comparison of model view controller and model view presenter. *Science International-Lahore*, 3.
- Rodriguez, J. (2016, June 15). *Don't get obsessed with design patterns*. Retrieved from Simpleprogrammer.com: <https://simpleprogrammer.com/2016/06/15/dont-get-obsessed-design-patterns/>
- Shvets, A., Frey, G., & Pavlova, M. (2017). *Abstract factory design pattern*. Retrieved from Sourcemaking.com: [https://sourcemaking.com/design\\_patterns/abstract\\_factory](https://sourcemaking.com/design_patterns/abstract_factory)
- Shvets, A., Frey, G., & Pavlova, M. (2017, December 5). *Singleton Design Pattern*. Retrieved from Sourcemaking.com: [https://sourcemaking.com/design\\_patterns/singleton](https://sourcemaking.com/design_patterns/singleton)
- Shvets, A., Frey, G., & Pavlova, M. (2017). *Strategy Design Pattern*. Retrieved from Sourcemaking.com: [https://sourcemaking.com/design\\_patterns/strategy](https://sourcemaking.com/design_patterns/strategy)
- Tower. (2017, December 5). *Learn Version Control with Git*. Retrieved from Git-tower.com: <https://www.git-tower.com/learn/git/ebook/en/desktop-gui/basics/why-use-version-control>
- van Rossum, G., Warsaw, B., & Coghlan, N. (2013, August 1). *PEP 8 -- Style Guide for Python Code*. Retrieved from Python.org: <https://www.python.org/dev/peps/pep-0008/#a-foolish-consistency-is-the-hobgoblin-of-little-minds>
- Weisfeld, M. (2013). *The Object-Oriented Thought Process*. Addison-Wesley.