Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

An intelligent sentiment proxy analysis system for estimating price changes in financial markets

Michael McGuinness

April 19, 2021

Supervisor: Prof. Khurshid Ahmad

A Final Year Project submitted in partial fulfilment
of the requirements for the degree of
B.A. (Mod.) Integrated Computer Science

Declaration

I hereby declare that this Final Year Project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: _____ Date: _____

Abstract

TODO - 400 words

Acknowledgements

TODO - thank who you gotta thank

Contents

List of Figures

List of Tables

Listings

Nomenclature

TODO

1 Introduction

TODO what? why? how?

1.1 Motivation

TODO what? why? how?

1.2 Objective

TODO what? why? how?

1.3 Report Structure

TODO what? why? how?

2 Background

TODO what? why? how?

2.1 more subsections and subsubsections

TODO what? why? how?

2.2 Background Summary

TODO what? why? how?

3 Design

This chapter discusses the overall design of the project, it's architecture and the individual components.

3.1 Brief

The project is 'An intelligent sentiment proxy analysis system for estimating price changes in financial markets'. It aims to understand the relationship between sentiment proxy and the market through creating a tool which aids in the analysis of sentiment proxy and market price data. Secondarily, this information is used to create estimations of the next day's returns for a given stock. The data-sets required and organising of said data-sets was a very important part of the project, which had to be executed with a high regard for precision and speed. The project revolved around the sentiment proxy and market price data-sets.

3.2 Requirements Gathering for System Design

Gathering the correct requirements for the system was an essential part of development. It is important to understand the limits of such a project, as well as understand how similar projects have attempted solving the problems that may come up. For this reason the reading of various research papers which attempted to perform a similar task was essential. These helped give a guideline of what the project structure was going to look like. The design was considered in relation to the requirements of this project, however, these papers helped expedite some design choices. The final design required understanding what information is required as an input for the system, and how it is structured. Then it is important to understand what the outputs of the system are. This then allows the conceptualisation of a pipeline which will achieve the required results. Finally, the pipeline must be refined and broken down into it's various parts. This allows a comprehensive and understandable design for the project.

3.2.1 Obtaining & Understanding Data-sets

The data-set requirements for the project are stock prices and sentiment proxies, both over time, and a dictionary containing words and their corresponding sentiment attributes. In order to fulfil these requirements three different source were required.

Price Source

One source is one which allows the gathering of stock prices. The requirements for this source are that it:

- allows the extraction of prices for a specific company
- allows the extraction of daily prices over a large period of time
- gives pieces of information beyond closing prices such as the volume of trades on that day

The source chosen is IEX Cloud.

IEX Cloud IEX Cloud is a financial data infrastructure platform that connects developers and financial data creators. It provides an API which allows access to a large amount of data centred around stock prices, including minute by minute prices for a given stock. The particular endpoint that was required for this project returns historical prices. This endpoint was perfect as it allowed the return of up to 5 years of data-points within the free tier. If there were expansion to be made it would even allow the return of the entire lifetime of a stock if the tier were upgraded. A few more points in favour of the use of IEX Cloud are that it is available at any time of day any day of the week, and it it is simple to use with good customer support in case of issues.

Sentiment Proxy Source

Another source is one which allows the gathering of sentiment proxy data. The requirements for this source are that:

- allows the extraction of sentiment proxy for a specific company
- allows the extraction of sentiment proxy over a large period of time
- allows extraction of sentiment proxies in a batch manner
- gives differing kinds sources, for example newspapers, academic articles

The sources explored are LexisNexis and Proquest, with the final decision having been LexisNexis.

Proquest Proquest provides access to many different databases containing licensed scholarly journals, newspapers, wire feeds, reports, etc. Using a trinity account, access is allowed 30 of these databases. Some of the databases included are:

- European Newsstream
- ProQuest Historical Newspapers: The New York Times with Index
- ABI/INFORM Global

For a comprehensive and up to date list of databases, it can be looked up on the website itself with trinity credentials. Proquest even allows the selection of specific databases to be searched. Allowing for more specificity in data-sources. Depending on the database articles can go back a very long time, especially with the Historical Newspapers sources. Proquest can search for a specific company then allows the download of up to 50 articles at a time.

LexisNexis LexisNexis is a research tool for news, companies and markets insights, multiple legal practice areas, and business and science biographies. For the purposes of this project, it has an extensive, reliable and quite importantly licensed library with hundreds of different sources, containing many kinds of articles including newspapers, magazines and journals. An interesting point to make is that it is recommended by the Californian Supreme Court and published Court of Appeal opinions in the US as an accurate, authentic, up-to-date, and reliable source for citing and quoting. It allows for a very detailed search of it's sources, as in it allows to search by type of source as well as allowing the use of Boolean expressions within the search parameters. For a full list of sources it can be found under the sources tab once logged in with trinity credentials. For our purposes it allows the search of these articles for a given company. It then allows the batch download of up to 500 articles at a time.

Source Choice There are a few reasons for having chosen LexisNexis over Proquest as the final:

- the quantity of news sources
- the reliability of news sources
- the ability to download much larger batch sizes

All of these factors allow for a more reliable, and much larger set of articles available for the project, allowing for more accurate final results.

Dictionary Source

The final source is one which supplies words and their corresponding sentiment attributes. This is essential for being able to understand what sentiment proxies are indicating. The requirements for this source are that:

- it has an comprehensive set of words
- it has been built in relation to sentiment proxy analysis
- it has generic positive and negative sentiment attributes

The dictionary chosen is one provided by Rocksteady. It is a generic dictionary in relation to economic terms. As far as I have understood, the information gathered for it was based on the Inquirer newspaper. It has 11,788 word entries with over 183 attributes that may be assigned to each word, including the generic positive and negative. This makes it quite an extensive and deep dictionary, on top of it fulfilling the requirements for the project.

A potential expansion and focus to the dictionary may improve it. This would mean adding more words, and changing the attribute values to be more in line with the problem being looked at and even the company being examined. A more specific dictionary may be found, or even a mix of both of these suggestions may cause great improvement in the result accuracy.

3.2.2 Users Interacting with the System

It is important that the project has the ability to produce certain outputs. Many of these being various statistics and graphing elements. The focus was on making sure all the required pieces of information were available. Since the user base for this program is mainly computer scientists, the interface could be left in the command line. It is formed by a set of menus which allows many different all the required kinds of of operations. However, they are all executed in the command line.

3.3 System Architecture Design

The design of the system is very important when it comes to understanding it's functionality, and purpose. The general purpose as discussed is taking sentiment proxy data and price data, then analysing it and making estimations with it. The reasoning for having a daily separation between endpoints that the newspapers and articles are being used for the analysis. These come out on a daily basis, this would lead to a large amount of inaccuracies if broken down in to smaller chunks. It may be useful to consider breaking the data down on a weekly or monthly basic if there were to be future expansion, as this may give a longer term relationship view, however, this is unexplored within this project.

It can be broken down into 3 main components, with the arrangement seen in figure ??. These being:

- `The Price Gatherer` – Gathers of all of the required price data in a date sorted array
- `The Sentiment Gatherer` – Gathers of all of the required sentiment data in a date sorted array
- `The Analyser` – Takes the data from the other two components and analyses it in various ways as well as run it through an estimator

Each of component has a very specific role within the system, and will be broken down further.

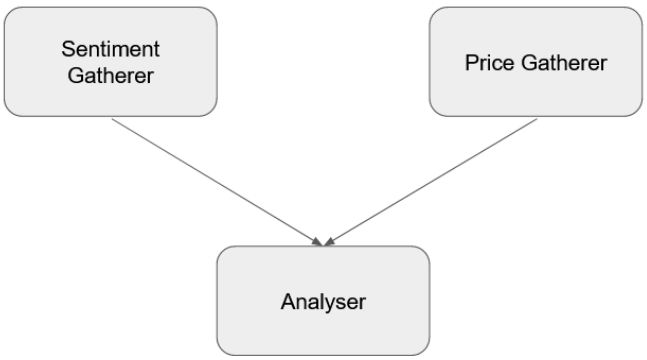

Figure 3.1: Overall Structure

3.3.1 The Price Gatherer

The Price Gatherer's purpose is to build a timeline of prices for a given company. It takes in data-points in reference to a given company's prices ordered by date and filters them in order to adapt them to the system. It can be broken down into 3 stages, arranged as seen in figure ??:

- `The Price Source` – Handles the gathering of price values
- `Key Filtering` – Filters the raw data and extracts the desired keys for each data-point
- `The Return Adder` – Adds returns to the data-points as extra keys

The overall output of this section is an array of JSON objects, ordered by the date key. Each of these objects containing the keys, the exception being the return keys as will be discussed:

- `date` – the date of the data-point
- `close` – the closing price for the date of the data-point

- `symbol` – the symbol representing the company this data is about
- `volume` – the volume of trades for the date of the data-point
- `return1Day` – the return in relation to 1 data-point previous
- `return7Day` – the return in relation to 7 data-points previous
- `return14Day` – the return in relation to 14 data-points previous
- `return21Day` – the return in relation to 21 data-points previous

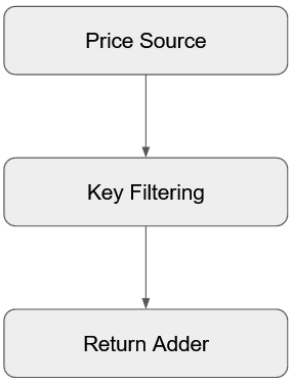

Figure 3.2: Price Gatherer Structure

The Price Source The purpose of the price source component is to contact the IEX Cloud API and collect daily price points for a given period. Due to the limitations of the tier chosen with the platform, the maximum period allowed is the past 5 years.

Key Filtering The IEX Cloud API returns an array of JSON objects, ordered by date, with each object containing the date, close, symbol and volume keys, as well as many others. This component filters out only the desired keys, removing the ones that are not required within the system. This allows the data to managed more easily, as well as any processes being executed faster, and anything that is stored requires less memory. This thus simplifies the process. However, it does create room for future expansion and potential improvement in understanding, as more specificity would be added.

The Return Adder The keys missing after the key filtering are those that determine the return values. This component adds them to each entry. The return for a given n determines the difference in closing prices between the current entry and the entry n days before. The returns require previous entries, therefore the first n entries will not have a return given n, the number of previous entries required. The reasoning for adding returns is due to the fact that it allows to examine the change between days rather than the full days themselves, since this project is studying the change in market values.

3.3.2 The Sentiment Gatherer

The Sentiment Gatherer's purpose is to build a timeline of sentiment for a given company. It takes in articles and a dictionary and processes them together in order to create an array of sentiment values ordered by date. It can be broken down into 6 stages, arranged as seen in figure ??:

- `The Article Source` – Handles the gathering of articles
- `The Article Parser` – Parses the raw articles into a format usable by the system
- `The Dictionary` – Handles the extraction of the dictionary from it's source
- `Key Filtering` – Filters the dictionary entries and extracts only the desired keys
- `The Sentiment Extractor` – Uses the dictionary entries in order to extract frequencies of sentiment from the articles
- `Z-Scores` – Modifies the sentiment extracted from absolute values to relative values using z-scores

The overall output of this section is an array of JSON objects, ordered by the date key. Each of these objects containing the keys:

- `date` – date of the data-point
- `articles` – z-score of number of articles
- `totalWords` – z-score of number of total words
- `positiveSentiment` – z-score of number of positive sentiment words
- `negativeSentiment` – z-score of number of negative sentiment words

The reasoning for using z-scores is due to the fact that it allows to examine the change between days rather than the full days themselves, since this project is studying the change in market values.

TODO - insert diagram

The Article Source The purpose of the article source is to provide the articles that will be used in the system.

The Article Parser The purpose of this component is to import the files and parse them into a format usable by the system, this being JSON. This is required for LexisNexis as the articles downloaded have the rich text format. The output of this component returns an array of JSON objects, this allows metadata to be stored alongside the body of the article itself.

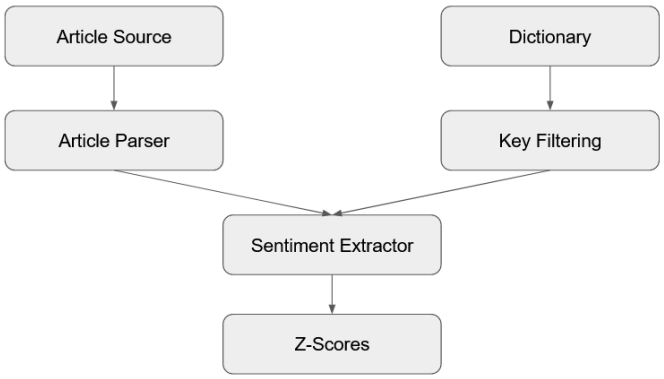

Figure 3.3: Sentiment Gatherer Structure

The Dictionary The purpose of this component is to handle the extraction of the dictionary into a JSON object array from the excel sheet it is stored in.

Key Filtering The purpose of this component is to filter only the desired keys from the dictionary, in order to decrease processing times and memory requirements. This quite importantly simplifies the analysis significantly. However, it does create room for future expansion and potential improvement in understanding, as more specificity would be added.

The Sentiment Extractor The purpose of the component is to extract sentiment frequencies from the articles, using the dictionary as a reference. What this means is that it tallies the number of words belonging to each desired attribute for each article. The articles are then joined by date, tallied appropriately, and ordered by date. The output is an array of JSON objects which represent the absolute values of the different attributes required.

Z-Scores The purpose of the component is to take the JSON object produced from the previous component and get the z-scores of the number of articles, of the total number of words in relation to the number of articles, and of each of the sentiment attributes in relation to the total number of words.

3.3.3 The Analyser

There are three components that are predecessors of the main analyser components. These are:

- `The User Interface` – A command line menu was created in order to organise and utilise all of the various components in a fast and easy way
- `The Joiner` – It joins the sentiment and prices data-sets where they overlap. Creating two new data-sets.

- `The Period Selector` – This allows for different periods to be explored within the dataset

The Analyser's purpose is to take the timelines produced by the Price Gatherer and the Sentiment Gatherer and analyse them in various different ways. It is a set of unrelated components which allow for the full analysis required in order to understand the data in the way desired for this project. The sub-components build for this component are the following:

- `Return Vs Sentiment Grapher` – Allows the graphing of sentiment keys and price keys in a time-series
- `Single Point Estimator` – Uses machine learning model to estimate next day returns
- `Autocorrelator` – Calculates correlation of key with itself with given lag
- `Return Vs Sentiment Correlator` – Calculates correlation of given keys with predefined lags
- `Descriptive Statistics` – Calculates and displays descriptive statistics of given key
- `Vector Autoregressor` – Calculate Vector Autoregression elements

The User Interface This is an essential part for the usability of the system create. It allows a user to navigate through the system, and analyse the data in any desired way, given the constraints of the system. It is a set of menu's which allow the selection of any given component and the selected that is desired in the analysis.

The Joiner For certain parts of the analysis the data being analysed should be overlapping appropriately. This is an important feature as the price and sentiment data-set have different start and end dates. As well as this not all dates that are covered by one are covered by the other, therefore data must be inserted in these cases, with the assumption that there was no activity for the data that did not exist previously. This has two main cases the first being a day with not sentiment, the assumption can be made that the day has zero sentiment, allowing an appropriate object to be inserted. As for days with no price the assumption can be made that the prices did not vary and the return was 0. However, this case is rare, and no examples of such a situation have been seen.

The Period Selector The purpose of this component is quite straight forward. It allows for the narrowing of scope. This is done by allowing for the selection of subsets of the main datasets by selecting a shorted period to examine.

Return Vs Sentiment Grapher The purpose of this component is to allow the creation of graphs which compare various columns between the prices data-set and the sentiment data-set. It however also allows the graphing of multiple columns in the one data-set, as well as graphing columns from only one data-set. This allows the creation of any graph desired. The graphs are graphed against the date of the data-point.

Single Point Estimator The purpose of this component is to explore the applications of machine learning models in this area. The models take in data from a given date, this being sentiment and price data and estimate whether the nest day would have a positive or negative return. This allows for the comparison between machine learning methods and mathematical methods. The output of this component is two fold. The first item returned is the accuracy of the most accurate model. The second item returned is said model, thus allowing for potential future use and/or storage. This area could be explored in a lot more depth through the use of a more substantial amount of models, hyperparameters and the like.

Autocorrelator The autocorrelator allows for the calculation of the correlation of a given column in a given data-set with itself, and it displays the information with n days of lag, where n is selected by the user. This is to help understand how closely related the data is with itself in nearby days, allowing for insight into whether there is a possibility of this data affecting itself. It is important to note, however, that high values of correlation only indicate that there is a potential relationship between days, and does not indicate necessarily causation.

Return Vs Sentiment Correlator The return vs sentiment correlator allows for the calculation of three kind of correlation. These are correlation between negative sentiment and 1 day return on the same day, the correlation between these when 1 day return is one day after the negative sentiment, and the correlation between these when negative sentiment is one day after the 1 day return. An expansion to this component would be to allow for different amount of lag between these data-sets. Similarly to the autocorrelator it is important to note, however, that high values of correlation only indicate that there is a potential relationship between elements, and does not indicate necessarily causation.

Descriptive Statistics The purpose of the component is to gather a column from the data-set and explore it's descriptive statistics, in order to understand said column better. On top of the descriptive statistics, this component prints out a graph which allows for a visual element. This can be of great aid when trying to understand the significance of the descriptive statistics.

Vector Autoregressor The purpose of this component is to explore causation between return and negative sentiment column using vector auto-regression. This adds a layer of rigorousness to the previously explored correlations, as it allows for the understanding of not just what is correlated but which columns cause data in other columns to change. This removes elements such as coincidence.

3.4 Scope of Project

The scope of the project was mainly limited by the data being explored. Many columns were removed from the various data-sources in order to simplify this. The data-set being explored is therefore limited it a general analysis. This can be seen through the use of the generic sentiment columns as well as the basic price columns. On top of this only a certain amount of depth was explored with the machine learning models, correlation and causation. For these reasons I believe there is very large amount of potential expansion to the project, and therefore the understand of the data and its relationships.

3.5 Design Summary

This chapter provides a brief overview of the project, the system design, the handling of the required data and its sources, user interaction with the system, it's design architecture, an explanation of the system components and the scope of the project.

4 Implementation

This chapter outlines and covers the process of implementation of the project. This includes the technologies used for the project, as well as the breakdown of the structure and implementation of the different parts of the project.

4.1 Technology Used

There are various technologies used within this project. These are Python 3.9, Visual Studio Code, Git and Github, Command-Line Interface, AntConc 3.5.9, Rocksteady 0.4 and Gretl. Each of these had a different function within the project.

4.1.1 Python 3.9

Python is an interpreted, high-level and general-purpose programming language. The version used was version 3.9.

Python was used in this project to develop and execute the procedures outlined in the implementation. This is due to the fact that python is a very useful language for handling projects which require a large amount of various different features as it is general purpose. For that reasons it has many publicly available libraries which help for many of the scenarios come across throughout development.

The libraries used had three main purposes: file handling, data tidying and mathematical operations.

File Handling

In order to handle the files downloaded from *lexisnexis* and *proquest*, various libraries and modules had to be used. The libraries and modules being `sys`, `os` and `striprtf`.

The `sys` module provides functions and variables used to manipulate different parts of the Python runtime environment. It is used to create a global variable which allowed the setting of a source for files, and allowed this to be used throughout the various files in the program. The source being the choice between using *lexisnexis* and *proquest* files.

The os module provides functions and variables used to perform operating system tasks. It is used to access environment variables, as well as to help parse through files in a given directory.

The `striprtf` library is used to translate rtf to a python string. When files are downloaded from *lexinexis* they are in rtf format. This library is used to help parsed the information in these files into a usable format.

Data Tidying

In order to tidy up the data extracted from articles and make it usable in the context required the use of some libraries is needed. These libraries and modules are `pandas`, `json`, `copy`, `datetime`, `operator`, `matplotlib`, `seaborn` and `warnings`.

The `pandas` is an open source data analysis and manipulation tool. This library is used to aid in the tabling of data. This was useful in order to use this data within graphs and to create an excel spreadsheet. For graphing timeseries, this library was especially useful with it's `to_datetime()` function which allowed the dates to be appropriately used as indices. This is significant especially when comparing two separate time series, as it spaced the values according to date and not which datapoint it is along the sequence.

The `json` library is used to dump json data from files and the extract it back from the file in order to cache the information extracted for future use.

The `copy` library is used to create deepcopies of json objects. This is necessary as the copies had to be completely separate from the original version, and due to how python works this cannot simply be done through a shallow copy. Therefore the library was used to facilitate this.

The `datetime` module supplies classes for manipulating dates and times. As they are usually stored in strings they can be complex to perform operations with. The library makes this process a lot more straightforward.

The `operator` module exports a set of efficient functions corresponding to the intrinsic operators of Python. It is used sort arrays of json objects that contain a given key. This was very useful when ordering data entries by date.

The `matplotlib` library provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. It therefore aids in the creation and display of graphs within the system.

The `seaborn` library is a data visualization library based on matplotlib. It helps the graphs look more aesthetically pleasing as well helping them become cleaere, therefore easier to read.

The warnings module is used in order to supress warnings. This helps make the program be easier to read for an end user.

Mathematical Operations

In order to perform mathematical operations appropriately multiple libraries are used. This is to avoid the recreation of tested and efficient functions, and avoid any potential errors when recreating them. These libraries and modules are numpy, statistics, scipy, math, sklearn and time.

The numpy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.

The statistics is a built-in Python library for descriptive statistics.

The scipy is a collection of mathematical algorithms and convenience functions built on the numpy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data. Specifically the scipy.stats module was used. Similarly, to the statistics library it was used to gather various types of statistical information.

The math module is a built-in module that you can use for mathematical tasks. It is used specifically for the log() function contained within.

The sklearn library is an incredibly useful machine learning library. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction. It is used for the machine learning functionality required for the singlePointEstimator. These being various trainable models, appropriately metric measuring functions, and some utility functions.

The time module provides various time-related functions. Most of the functions defined in this module call platform C library functions with the same name. This important because it is used to time some elements of the system, and these timings must be precise.

4.1.2 Visual Studio Code

Visual Studio Code is a freeware source-code editor made by Microsoft for Windows, Linux and macOS. Features include support for debugging, syntax highlighting, intelligent code completion, snippets, code refactoring, and embedded Git. It was used to ease the development of the codebase.

4.1.3 Git and Github

Git is a version control system. Git tracks the changes you make to files, so you have a record of what has been done, and you can revert to specific versions should you ever need to. Git allows changes by multiple people to all be merged into one source. This can be used to create features then once they are complete merge them into a final version of the system.

GitHub is a provider of Internet hosting for software development and version control using Git. It offers the distributed version control and source code management functionality of Git, plus its own features.

4.1.4 Command-Line Interface

A command-line interface processes commands to a computer program in the form of lines of text. It is used to execute the programs developed, git commands and any other required commands.

4.1.5 AntConc 3.5.9

AntConc is a freeware corpus analysis toolkit for concordancing and text analysis. It is used for preliminary text analysis of the articles downloaded from *lexisnexis* and *proquest*. It can be used to create words lists, n-grams, concordance plots, amongst other useful tools that may be used to examine word choices in texts.

4.1.6 Rocksteady 0.4

Rocksteady is a sentiment analysis tool. It creates a timeline of sentiment, and allows the filtering as well as visualisation of this data. It is used within this project to understand how the sentiment proxy extraction process works.

4.1.7 Gretl

Gretl is an open-source statistical package, mainly for econometrics. The name is an acronym for Gnu Regression, Econometrics and Time-series Library. It has both a graphical user interface and a command-line interface. It was mainly used for vector autoregression within this project.

4.2 Setting up the System

The technologies that have been used to implement the system have been covered. This section therefore covers the details of the implementation of the indivdual components laid out in the design, using these technologies.

The system is run as a script using the terminal. The commmand for initiating the program being `python IntelligentAnalysis.py`. This is assuming that the default version of python running is version 3.9, otherwise the command may have to be modified slightly to accomodate for this. The full codebase can be found at the following url: `https://github.com/DaVinciTachyon/FinalYearProject`. This command is run from the root directory of this git repository.

4.2.1 The Price Gatherer

The Price Gatherer has a straightforward flow. Thus making the understanding of it's implementation quite simple.

The Price Source

The Price Source component has one main function. That is to contact the IEX Cloud API and return the past 5 years of price points for a given stock.

This component does require a bit of set up in order to have access to the api:

1. Register for IEX Cloud
2. Gain access to and retrieve an API key
3. Insert API key into environment variables

Once these steps are done the component itself can be used. The component can be divided into a few different steps:

1. Check if the cache exists
    - If cache exists, load items from cache
    - otherwise, continue
2. Load API key from environment
3. Create API request, with certain important information:
    - URL – This contains the API Key and Stock Symbol
    - API Key

- Stock Symbol
- Period desired – This is set to 5 years given the payment tier chosen on the website

4. Gather API response into JSON String
5. Sort the JSON array by date
6. Cache the data

An important item to highlight within this procedure are the use of a caching system. This allows for a large speed up in the process of gathering prices, when the program is run for a second time. As well as this it save a lot of money in API fees and network access fees. The caching system in this case is the creation of a json file, however, it may be optimised to use a database.

In order to get the cache, the program check whether the file containing th edata exists, if it does it opens the file and loads the data as a json string. It can be seen below:

Listing 4.1: Loading Prices Cache

```
if os.path.isfile(pricesFilename):
    with open(pricesFilename) as json_file:
        data = json.load(json_file)
```

The cache is the created by opening a file and simple dumping the json string within, as can be seen here:

Listing 4.2: Creation of Prices Cache

```
with open(pricesFilename, 'w') as json_file:
    json.dump(data, json_file)
```

Another item to not is the sorting of the JSON array. This is usually a more complex process, however, the `operator` module eases this process significantly.

Listing 4.3: Sort JSON Array

```
sorted(data, key = operator.itemgetter('date'))
```

Key Filtering

The key filtering process takes in the full data-set and the keys desired for each element of the data-set. It then creates a new data-set with only the desired keys extracted. This is

currently a `for` loop which goes through the entire data-set adding each entry individually.

Listing 4.4: Filtering Desired Keys

```
for entry in originalDataset:
    filteredEntry = {}
    for key in keys:
        filteredEntry[key] = entry[key]
    newDataset.append(filteredEntry)
```

The current solution has a time complexity of `O(N)`, where N is the length of the data-set. Some research has been done in order to find a more efficient solution, however, so far this has been unsuccessful.

The Return Adder

This component requires an array of numbers for the returns to create. What this means the that it will reveceive a 1 in the array if it wants to calculate 21 days returns. It then loops through the data-set adding the appropriate returns to the given entry.

The return cannot be added to the first n days, this is simply done using an `if` statement to make sure to not attempt this before it is possible.

The formula used for return is $log(\frac{r_t}{r_{t-l}})$ where r is the return, t is the current day, and l is the number of days being used.

The final code put this all togther in the following way:

Listing 4.5: Adding Returns

```
for t in range(len(prices)):
    for l in returnLengths:
        if(t >= l):
            prices[t][f"return{l}Day"] =
                math.log(prices[t]['close']/prices[t-l]['close'])
```

4.2.2 The Sentiment Gatherer

The Sentiment Gatherer can be divided into three main subsections, each one of those being divided into two components. It is important to not that the first two main subsections can be run in parallel, and must be run before the last. The Article Source and Article PArser compose this first subsection. This can be run in parallel to the Dictionary and Key Filtering

components. However, they must all be run before the Sentiment Extractor and consequently the Z-Scores components.

The Article Source

The Article Source component has one main function. That is to gather the dowloaded articles and import them into the system.

This component requires a bit of set up:

1. Log into LexisNexis with Trinity credentials
2. Search for a given company
3. Download articles, this part has a few steps:
   (a) Select rtf format
   (b) Remove formatting from files
   (c) Download 500 files by giving an appropriate range – this is the maximum available for downloading at once
   (d) Repeat from step (a) 5 times
4. Wait 2 hours – Once 5 downloads have been executed, LexisNexis does not allow any more dowloads for 2 hours
5. Repeat from step 1 as many times as necessary

Using this process 7000 articles were dowloaded for this iteration of the system, for the given company, which will be discussed later on. The reason for the choice of 7000 article is due to the fact that they cover the 5 year range covered by the Price Source quite thoroughly. They are a significant amount of articles, however more could be dowloaded. That said they covered the requirements for this project.

There was though of automating this procedure in order to improve up this process. This would expedite the process and lead to the ability to download a lot more articles. There were a few reasons this was not done:

- The 2 hour wait is still required, therefore the process would not be expedited significantly
- The amount of articles gathered manually were sufficent for this project
- There was a focus on creating other parts of this project, as the creation of this automation would require a significant amount of time

Once these step are executed and the downloaded files are placed in the correct directory the component functionality is complete.

The Article Parser

The Article Parser Component gathers the articles and turns them into a format usable by the system. The component can be broken down similarly to the Price Source component with some differences:

1. Check if the cache exists
    - If cache exists, load items from cache
    - otherwise, continue
2. Find all files with articles
3. Iterate through all the files, doing the following:
    (a) Read the contents of the file and assign them to an rich text format (rtf) string
    (b) Parse the rtf string into a legible string using the `rtf_to_text()` function
    (c) Split the string into the articles contained
    (d) Parse the article strings into a JSON object containing the required content and metadata
    (e) Append the articles to a JSON array
4. Cache the article JSON array

The caching system is identical to the Price Source, the JSON array is dumped to a file and extracted from the file when required.

The extraction of the articles from the file into an rtf string has a point that should be highlighted. The way this works requires each line to be extracted separately, then concatenated to the previous, until the full file is extracted. This part of the program was given in order to aid in the project, however it was quite inefficient. The original code looked like this:

Listing 4.6: Slow Text Extraction

```
fileContent = ''
for line in file:
    if line.strip() != '':
        fileContent += line + "\n"
    else:
        fileContent += line
```

The improvement made to the code then looked like this:

Listing 4.7: Optimised Text Extraction

```
fileContent = "".join([f"{line}\n" if line.strip() != '' else f"{line}" for
    line in file])
```

Beyond reducing the number of lines in the program and looking cleaner, this change creates a radical change in the speed of the program. The program was run on the 7000 articles, and originally was let run for 12 hours before being terminated. Then once the change was made the program consistently finishes within 5 minutes. The reason for this being such a big deal is that if there were more articles added, the entire program would have be run again. The reason for the for the speed difference explains why there is such a large timing difference.

The original way uses the += operator, what this means is that an array is created for the original string, as a string is a character array, then a new character array is created with the new appropriate length in order to allow for the change. This is quite a costly process both timewise and spacewise, and it is repeated for every line in every file. It is important to remember that each time this process is executed the array becomes bigger, making it a slower process each time.

The optimised method used the the `"".join()` function. The way works is that it creates a set of strings one for each line, then does the joining only once. Therefore the large array array has to be created only once and does not have to have its size change.

The final point to discuss is the parsing of the python string into a JSON array. In order to utilise the information in an easier manner, some metadata is extracted alongside the content of the article itself. Each article is represented by a JSON object with the following keys:

- `title` – The article title
- `source` – The name of the publication the article was published in
- `date` – The date the article was published
- `copyright` – The type of copyright of the article
- `length` – The length of the body of the article
- `section` – If the article is divided into parts, this will indicate which part this is
- `language` – The language the article is in, for this project, the articles are in English
- `pubtype` – The type of publication the article was published in - newspaper, magazine, etc.

- `subject` – Key words which describe the article contents
- `geographic` – The locaiton of publication
- `loaddate` – The date the article was uploaded to the the Article Source
- `byline` – The author(s) of the article
- `body` – The contents of the article

These metadata items are extracted into the array from the original string by using certain indicators withing the text. An example of such an indicator would be that the line containing the source thatsrts with `"Source:"`. This is executed in the code the following way:

Listing 4.8: Source Extraction

```
if tempLine.startswith('Source:') and extractValue(line) != '':
    document['source'] = extractValue(line)
```

The `extractValue` function simple removes the indicator, allowing for the extraction of the value only.

The special case is the body. This still has indicators. The indicator of the start of the body is simply a line only containing `"Body"` and the end of it is simple represent by `"End of Document"`. The lines within these two indicatos are simply concatenated with each other and assigned to the correct key at the end. The += operator is used for this. There is room for creation of efficiency, however, due to the relatively short nature of these articles, this is efficient enough for use within this process currently. In the future this would be an area to explore the creation of such efficiencies.

The Dictionary

The dictionary is extracted from the Rocksteady files. If there were future development to this project it may be worth spending time adding more entries, as well as making the sentiment choice more specific to the context of the articles that are chosen. An example of such a thing would be that for a company such as Gamestop the word game may have positive sentiment, even though in general it may have no sentiment attached.

The extraction of the entries from the excel file is very simply achieved with the use of the pandas library. The following function extracts all the entries and associates each item to it's correct row and column.

Listing 4.9: Dictionary Extraction

```
data = pd.read_excel(r"./dictionaries/inquirerbasic.xls")
```

Key Filtering

This process of extracting only the desired sentiment attribute columns is achieved through the use of a dataframe. It takes the data-set from the previous component and an array containing the titles of the desired columns as inputs, and returns only said columns.

Listing 4.10: Key Filtering Dataframe

```
df = pd.DataFrame(data, columns= ['Entry', 'Positiv', 'Negativ']).to_numpy()
```

The dataframe is then turned into a dictionary object. What this does is it assigns the values of the desired sentiment attribute columns to the word itself. The object will have the following format: `{ word: [ sentimentColumnValue ] }`. This is achieve with the following segment of code:

Listing 4.11: Dictionary Creation

```
dictionary = {}
for index, item in enumerate(df):
    dictionary[item[0]] = item[1:]
```

It is important to note that `item[0]` is the word itself and the rest of the items in the array are the sentiment attribute column values.

The Sentiment Extractor

This component can be broken down into two steps:

1. Extraction
2. Date Joining

Extraction This step takes in the articles from the Article Parser and the Dictionary from the Key Filtering. It then creates a new array of JSON Objects. Each object represent an article. Each object has the following keys:

- `date` – The date key from the article
- `totalWords` – The length key from the article
- `positiveSentiment` – The number of words that have the `Positiv` attribute determined by the dictionary
- `negativeSentiment` – The number of words that have the `Negativ` attribute determined by the dictionary

If more sentiment attributes were to be included they would have an extra key assigned to them. The sentiment attribute keys are found by iterating through all the words in the body of the article and tallying the words according to which columns are labeled true in the dictionary.

Date Joining This step takes the array created in the previous step and merges all of the elements on any given day together. This means the final JSON object will be an array ordered by date with the following keys:

- `date` – The date of the articles
- `articles` – The total number of articles on said day
- `totalWords` – The total number of words in all the articles
- `positiveSentiment` – The number of words that have the `Positiv` attribute determined by the dictionary in all the articles
- `negativeSentiment` – The number of words that have the `Negativ` attribute determined by the dictionary in all the articles

Similarly to the Price Gatherer, the array is ordered by date in the following way:

Listing 4.12: JSON Array Ordered by Date

```
sentimentByDate = sorted(sentimentByDate, key = operator.itemgetter('date'))
```

Z-Scores

This component can be divided into four steps:

1. Column Separation
2. Percentage Calculation
3. Z-Score Calculation
4. Object Assignment

Column Separation The array created in the Sentiment Extractor is iterated through and an array is created for each key within the objects. This means that each key will have its own array.

Percentage Calculation In this project it is important to undesrtand relative amounts. Therefore a few different percentages must be calculated. The sentiment columns are replaced with the percentage of sentiment words in relation to the total number of words for

that article. The totalWords column is replaced with the percentage of words in relation to the number of articles.

Z-Score Calculation The `scipy.stats` module is then used to calculate the z-score for each in element in each of the arrays, excluding the `date` array. An example of this is:

Listing 4.13: Z-Score Calculation

```
articles = stats.zscore(articles)
```

Object Assignment The final step in the component is create an array from the separated array with the following keys:

- `date` – date of the data-point
- `articles` – z-score of number of articles
- `totalWords` – z-score of number of total words
- `positiveSentiment` – z-score of number of positive sentiment words
- `negativeSentiment` – z-score of number of negative sentiment words

This will still be ordered by date, as the order was never changed.

4.2.3 The Analyser

The Analyser is composed of various unrelated components, which are preceded by three components which allow the other to achieve their tasks efficently, and in an easy to execute manner. The inital three components are the following:

- The User Interface
- The Joiner
- The Period Selector

The User Interface

The User Interface is essential towards creating a navigatable environment for users of the system. In a very basic sense it is a set of menus which allows the selection of which tool to use within the Analyser, and selected the desired parameters for these tools.

These menus work by showing a user a set of options, and allowing the user to put in a number in order to select one of these. This methods allows for full usage of the program, and allows limitations to be imposed so that it may not be abused. The steps to achievinng one of the menus is the following:

1. Print out the options so that the user may know what they are
2. Request the input
3. Ensure the input is an integer, and convert the string input to one
4. Select the appropriate the appropriate function to run given the option selected, or repeat the question if the input was erroneous

Step 1 The options can be laid out with the following code:

Listing 4.14: Menu Options

```
print("1: 1 year", "2: 2 year", "3: 3 year", "4: maximum available", sep="\t")
```

Step 2 Python has a function to simplify the process of gathering the input. The input has the `strip()` function run on it in order to remove beginning and ending whitespace, just incase this is done by mistake.

Listing 4.15: Get Input

```
sampleSize = input("Please select the length of your sample size: ").strip()
```

Step 3 In order to ensure the input is a valid integer the follwing statement is run. It simply checks if the input is a digit, and if it is it converts it to an integer.

Listing 4.16: Check if Integer

```
if sampleSize.isdigit():
        sampleSize = int(sampleSize)
```

Step 4 The correct function is selected in one of two methods.

1. A set of `if/elif` statements, since a `switch-case` statement does not exist in python

   Listing 4.17: If/Elif Statements

   ```
   if section == 1:
       dataset = prices
   elif section == 2:
       dataset = sentiment
   ```

2. Check if the input is an integer and in the desired range, and can be given into the function

Listing 4.18: Integer Range Choice

```
if isinstance(column, int) and column > 0 and column <= len(keys):
    column = keys[column - 1]
```

The `isinstance` function allows the system to determine if a variable is an instance of a given type. In this case if it is an `int`.

Both of the allow for the situation where an input is entered incorrectly. In these cases the `else` option is chosen which asks you to try again, and the boolean which determines whether the item has been chosen is set kept as false, since it is set to true in other cases.

The Joiner

The Joiner wants the prices and sentiment data-sets to overlap. This has a requires a few step to be excuted in ored to be achieved:

1. Select start date – this will be the start date of the data-set that starts later between the two
2. Select end date – this will be the end date of the data-set that ends earlier between the two
3. Find the element indexes for start and end dates for both data-sets
4. Create data-sets with new start and end indexes
5. Add empty elements to days where the datset does not have a day the other does, in order to have fully overlapping data-sets

For step 4, this can be achieved using the following notation within Python: `prices[priceStart:priceEnd]`, `sentiment[sentStart:sentEnd]`, where the indexes have been previously discovered.

For step 5, the empty objects are the following:

- For a day with no sentiment

Listing 4.19: No Sentiment Day

```
{'date': prices[i]['date'], 'articles': 0, 'totalWords': 0,
    'positiveSentiment': 0, 'negativeSentiment': 0}
```

- For a day with no price activity

Listing 4.20: No Price Day

```
{'date': sentiment[i]['date'], 'close': prices[i - 1]['close'],
    'symbol': prices[i]['symbol'], 'volume': 0}
```

It is important to note that this step is done on the the full values of these arrays, then the z-scores are calculated afterwards.

The Period Selector

This item has it's own menu that runs after the tool to use has been selected. It allows the choice between 4 different periods. These being:

- 1 year – The data-sets returned will start 1 year before the end date of the data-sets, and have all the following values
- 2 years – The data-sets returned will start 2 years before the end date of the data-sets, and have all the following values
- 3 years – The data-sets returned will start 3 years before the end date of the data-sets, and have all the following values
- maximum available – The full data-sets will be used

This is done by getting the date of the last element, and subtracting the appropriate time from it. The date is an instance of `datetime`, not a string.

Listing 4.21: Subtract years

```
date - relativedelta(years=years)
```

All of the elements with a date including and past the start date are then return as the data-set to be used.

Return Vs Sentiment Grapher

The menu for this component allows the user to select the prices columns and sentiment columns they which to visualise.

The first step towards creating these graphs is inserting the objects into dataframes, where the indexes are the sets of dates as `datetime` instances. This enables them to be used as timeseries.

The next step is to use the `matplotlib` library in order to create the graphs themselves. The figure on which the graph will be displayed needs to be split into two subplots. These will be overlapping each other, with the x-axis being the date, and the y-axis being on one side representative of the price and the other the sentiment.

Listing 4.22: Split Window into Multiple Subplots

```
fig, ax = plt.subplots()
```

Listing 4.23: Create Overlapping Subplot

```
ax2 = ax.twinx()
```

Listing 4.24: Plot Dataset

```
ax.plot(dataset, color=colour, marker="o")
```

Listing 4.25: Plot Legend

```
fig.legend(legend)
```

Listing 4.26: Save picture of plot

```
fig.savefig('priceVsSentiment.jpg', format='jpeg', dpi=100,
    bbox_inches='tight')
```

TODO sample graph

Single Point Estimator

The Single Point Estimator can be broken down into 4 main steps:

1. Create input and output series
2. Input/Output Series Shuffling
3. Model Testing
4. Return Highest Accuracy Model and its accuracy

Create input and output series The first step changes the prices and sentiment JSON object arrays into an array containing all of the desired columns, excluding date, as an array for each entry. As well as creating a second array with Boolean values for whether the next day's return is positive or negative. An example of this is if the close and volume columns are desired from the prices data-set and the negative sentiment column is desired from the sentiment data-set, the input series array elements would look like this `[ closeValue, volumeValue, negativeSentimentValue ]`. If the next day's return for a given entry is positive, the output series element would be `True`.

Input/Output Series Shuffling The shuffling simply randomises the order of the entries, so that there is no bias based on date. This is done using the function given by the `sklearn.utils` module in the following way:

Listing 4.27: Shuffle series

```
X, y = shuffle(X, y)
```

This function shuffles the order of both series while mainting the correspondece between the two sets.

Model Testing The model testing itself can be broken down into a discrete set of steps common accross all models. The difference being that some models require hyperparameter training, meanwhile others do not. These steps are:

- `Hyperparameter Training` – A hyperparameter is a parameter whose value is used to control the learning process. Ranges of values are given and iterated through, the highest accuracy value is stored and used to create a final model.
- `K-Folds Cross-Validator` – It provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default). Each fold is then used once as a validation while the k - 1 remaining folds form the training set. This implementation uses 2 folds. This is an arbitrary value, further exploration may lead to finding a more suitable value. The `kf.split(inputDataset)` is the function which excutes the split on the dataset.
    1. `model.fit()` – Train the model on the training dataset
    2. `model.predict()` – Create a predicted output set given the test input dataset
    3. `accuracy_score()` – Calculate the accuracy of the values predicted from the model against the test output dataset
- `mean(kFoldAccuracies)` – The mean accuracy of the accuracies calculated for each of the folds is set as the accuracy for the given hyperparameter
- `max(hyperparameterAccuracies)` – The maximum accuracy of the accuracies calculated for the various hyperparameters is set as the defined accuracy for the model
- `timer` – The `perf_counter()` function from the `time` module is used to time the training and discovery of hyperparameter for the model. The start time is set to before the iteration through the hyperparameters, and the end time is just after.
- `max(modelAccuracies)` – The accuracy of the highest accuracy model as well as the trained model itself are returned

An example of the steps being executed on the KNeighborsClassifier is following:

Listing 4.28: KNeighborsClassifier Example

```
kf = KFold(n_splits=2)

startTime = perf_counter()
K = range(1, 10)
highestAccuracyK = 0
highestKNNAccuracy = 0
for k in K:
    accuracies = []
    for train, test in kf.split(X):
        model = KNeighborsClassifier(n_neighbors=k).fit(X[train], y[train])
        pY = model.predict(X[test])
        accuracies.append(accuracy_score(y[test], pY))
    accuracies = np.array(accuracies)
    accuracy = np.mean(accuracies)
    if(accuracy > highestKNNAccuracy):
        highestKNNAccuracy = accuracy
        highestAccuracyK = k
print("Processed: ", model.__class__.__name__)
print("Time: ", round((perf_counter() - startTime) * 1000), "ms")
```

TODO hyperparameter training choice of accuracy vs mean squared error graphs

TODO explain all models, differences, why chosen

- KNeighborsClassifier - k hyperparameter
- DecisionTreeClassifier
- GaussianProcessClassifier
- AdaBoostClassifier
- RandomForestClassifier - n hyperparameter

Return Highest Accuracy Model and its accuracy The accuracy of the different models is compared and the highest accuracy model is returned to the user.

Autocorrelator

TODO what? why? how?

Return Vs Sentiment Correlator

TODO what? why? how?

Descriptive Statistics

TODO what? why? how?

Vector Autoregressor

TODO what? why? how?

4.3 Implementation Summary

TODO what? why? how?

5 Results Evaluation

TODO what? why? how?

5.1

TODO what? why? how?

5.2 Results Evaluation Summary

TODO what? why? how?

6 Discussion & Conclusion

TODO what? why? how?

6.1 Meeting the Project Objectives

TODO what? why? how?

6.2 Further Challenges

TODO what? why? how?

6.3 Future Work

TODO what? why? how?

6.4 Discussion & Conclusion Summary

TODO what? why? how?

A1 Appendix

TODO what? why? how?