# Meeting the Brief

## Introduction

My project is an interactive stock market analysis system that collects, cleans, and visualizes stock data while allowing user interaction and predictions. In this video, I will demonstrate how it meets both the basic and advanced requirements of the coursework brief.

## Basic Requirements

### Collect and Prepare the Data

Basic requirement 1 is to collect and prepare the data. We start with selecting the dataset we will use, which comes from the Nasdaq’s historical API. This allows us to take data from whatever date range we like, I have chosen Year-to-date, because it provides a clear starting point. The data is collected as raw JSON, which we then convert to CSV, from there, in another file, we will convert the CSV into lists, we then clean the data by stripping off the dollar signs change the date format, and removing commas.

### Data Analytics and Visualization

The next step is data analytics and visualisation. We use a Python algorithm to calculate regression lines that show trends and will allow us to make our recommendations later on, in the meantime we also create two graphs, a line graph showing both the price history, and the trend lines. And a bar chart, with the slopes of our four regression lines calculated. These are, year to date, lowest point to date, last seven days to date, and lowest point in the last seven days to date. These are all calculated through array indexing and searching.

### Create a Basic Interactive Information System Interface

Next is to create an interactive information system interface. To create my information system, I have used NodeJS, a JavaScript environment used for executing JavaScript code outside of a browser, and ReactJS, which is a frontend library for NodeJS. There are two graphs shown here, a line graph similar to the one we saw in python, and a candlestick graph, which clearly shows trends in stocks.

## Advanced Requirements

### Ensure that users can interact with the data visualisations on the information system.

We also have to ensure that users can interact with the data visualisations on the information system. Using this information system, users can change data range, click to filter data and hover to see the exact price of a stock at that time

### Create a form or poll relating to your chosen dataset on your information system that collects and stores data from the users.

Next I created a form related to the dataset to collect and store user data. Here we can see the form I created, using ReactJS to render a user interface, and MongoDB to store the data. Multiple different datatypes are collected, a string for the email, multiple numbers that can be either floats or integers, there is a Boolean variable depending on whether or not the stock prediction was accurate.

### Create a separate area on your information system that makes recommendations or helps inform decisions based on your analysis of your chosen dataset.

Finally, the recommendations, The goal of this project was to predict the stock market as accurately as reasonably possible, and we do this by getting the mean slope of the regression lines. This is then converted into a human readable sentence, some examples you can see now. This mean slope is also used in a y=mx+c equation to estimate a price one week from that day.

## Conclusion

In conclusion, this project meets all of the basic and advanced requirements requested, it collects and cleans stock data, it analyses trends, provides interactive graphs, allowed user interaction, used a form or poll, and provided recommendation. It is fully functional and demonstrates a real-world application of data analytics

# Investigation

The idea of predicting stock markets has intrigued researchers and financial analysts for decades. Notably, in 1973, economist Burton Malkiel posited in his book "A Random Walk Down Wall Street" that stock prices follow a “random walk”, which means that fluctuations are random, and cannot be predicted solely by analysing past price movements. This hypothesis challenged conventional technical analysis and spurred further investigation into market behaviour. My project aims to test this hypothesis by leveraging modern data analytics and machine learning techniques to analyse stock market data, specifically focusing on the Nasdaq index.

My interest in stock market prediction was initially sparked while investigating Bitcoin’s volatile price fluctuations. However, Bitcoin's up-to-date value and historical data are often locked behind APIs, limiting accessibility for analysis. In contrast, the Nasdaq offers a free API that allows users to download historical data from any point in time. This accessibility makes the Nasdaq an ideal candidate for this project, as it ensures that all datasets are publicly available and can be thoroughly analysed.

An inspiration for the design of this project comes from existing analytical solutions, such as Apple's Stocks app on iOS, Yahoo Finance, and the Nasdaq’s own indexes. These platforms provide valuable insights into market trends and investor behaviour, but my goal is to take the analysis a step further by testing the validity of Malkiel's hypothesis. By employing advanced statistical models and machine learning algorithms, I aim to determine whether patterns and trends can be identified that could potentially forecast future price movements.

This project aligns closely with the field of financial analytics, a branch of mathematics that has led to highly lucrative careers in finance and investment banking. By diving into this project, I will gain practical experience in data analysis, statistical modelling, and machine learning, all of which are crucial skills for a career in Data Science and Analytics—a field I am considering studying next year.

This project not only allows me to explore a fascinating area of financial mathematics but also serves as a practical stepping stone toward my future studies in Data Science and Analytics. With publicly available datasets. I am eager to put Malkiel's hypothesis to the test and potentially uncover hidden patterns in the Nasdaq stock market.

# Plan and Design

For the backend I have chosen to use NodeJS, there were a few main reasons for this. Firstly NodeJS is portable, meaning that an examiner can run the project on their own computer, provided they have the libraries installed. Secondly it is extremely performant. NodeJS has an extremely efficient way of rendering, that far outpaced the competition. And finally, it made it extremely easy to create both the back, and frontend of the project with compatible technologies

NodeJS is ideal for operations such as:

1. Fetching live stock data
2. Rendering interactive visualisations, (when combined with Plotly and ReactJS)

I did also consider Flask as an alternative, however there were a few reasons why I did not chose to use it. Firstly there is much more setup required, you are supposed to use a python virtual environment, and importing libraries is somewhat more complex. Secondly, because we need to use a virtual environment, it would not have been portable at all. Thirdly, it is far slower, and far less efficient. And finally, in the advanced requirements, it is specified the form must be created with JavaScript, and this would not have been possible with Flask, because it is a Python environment

For my database, I chose to use MongoDB, this was simply because I am using it for another project and felt that sticking to what I knew was best, I also had experience with the Mongoose JavaScript library required to connect to the database.

ExpressJS was used as a straightforward way to connect to the database from the backend server.

There are two different NodeJS “projects” within this project. One, a UI frontend built using the react framework, and Plotly. And a backend server, built using ExpressJS and Mongoose, it also handles and executes all the Python Scripts. Mostly, I really wanted to keep only one project, and this worked great for a long time. Untilz I had to submit form data. It was at that point it became clear a second server was needed, you see, react projects are client-side, and unable to interact with either local files, or databases. The proper way to handle this is to send data from the frontend to an API, and from there to the hosted database. This is how the server functions.

An important part of this project is JSX, this is an extension of normal JavaScript, and it allows for the HTML you see in most JavaScript files in the project. It shares features of both a programming language, and a markup language, allowing for a “best of both worlds” situation when trying to create dynamic, updating webpages.

A diagram of a website

AI-generated content may be incorrect.

Above is the list of important files regarding this project, all other files are either for NodeJS to handle the packages and libraries I am using, e.g. package. Json node\_modules etc. Or are automatically installed by react, and were not used in the project, e.g. app.test.js.

One thing you may notice is that there are many .css files, and the reason for this is you cannot edit every element from the index.css file. This is because of JSX, index.css has no ability to see the HTML within JSX, and so cannot change it.

Pseudocode for regression lines

Function slopefunction(all x values, all y values):

Number of elements on x axis

Get the mean of both axis

Calculate covariance of x and y

Calculate covariance of x

Calculate slope (M=covariance of X and Y divided by covariance of X)

Y intercept equals mean of Y minus the slope times mean of x

Return Slope and Y Intercept

Plot the regression line using the number of values in the x-axis, the starting point of the regression line, the y-intercept, and the slope

# Create

## Week 1

* Made mind map of project brief and requirements
* Began working on an idea based around stock predictions

## Week 2

* Decided Flask would be a smart and easy way to create an interactive entertainment system
* Attempted to find API to pull live data from the NASDAQ
* Attempted to create a system to recognise the last time the local database of a stock was updated, and fill in all blanks

## Week 3

* Began work on the report
* Decided to remove Flask and instead use NodeJS due to improved speed and portability of the project.

## Week 4

* Streamlined system for obtaining data
* Styling of the report starting, improved the look of the heading and menu objects.

## Week 5

* Continued working with NodeJS, installing react etc.
* Improvements to main.py

## Week 6

* Improvements to report.
* Creating graphs with Python

## Week 7

* Removed API temporarily to better satisfy basic requirements
* CSV handling, taking data from a database
* Began working on a graph of stock Opening, Closing, High and Low points from the first of January onwards.
* Issue with graph, was reversed because csv from Nasdaq placed newest dates at the top
* Working on styling of the report body

## Week 8

* Completed linear regression from YTD low
* Graph is exported to JSON file

## Week 9

* Interactive information system created, using NodeJS and ReactJS.
* Graphs working, and update live

## Week 10

* Backend Server created to handle user submission data.
* Pie chart done of responses

## Week 11

* All python moved to the server to make it callable correctly.
* User can view and get predictions on whichever stock they prefer, instead of just Apple
* Styling done for the information system

## Week 12

* Completed Report

## Issues

While working on the regression lines, I discovered a catastrophic error that would occur under very specific conditions. For context, there are four regressions calculated, firstly a year-to-date linear regression. A regression from the lowest point this year, a regression over the last seven days, and finally, a regression from the lowest point in the last seven days. You may realise, it is not possible to calculate those, if the lowest price offer was the most recent data source. This issue was only discovered because the Stock Market crashed recently, had that not happened, it is likely this issue would not have been discovered. Thankfully the issue was easily fixed, if the dataset for regression is only one value long, the slope is calculated to be zero.

A graph with a line going up

AI-generated content may be incorrect.

There was however another error with the regression lines too. In the above graph, you can see a stock that is clearly falling, at an extremely quick rate. However, the prediction being given was extremely positive, to the point of suspicion, as it turned out, having each regression line given equal weight in the prediction equation was a mistake. The two “low” regressions need to be valued a lot less, as they were predicting wild changes in stocks, because of a mild day-to-day rebound. Weighing these for less resulted in a far more accurate prediction

Below is an overview of how every part of this project works.

## Backend

The main functions of the backend, is to connect, collect and send data to the database. It also is responsible for calling the python files. However, it does not handle the data collection from those files, which is done by the python files themselves.

### Index.js

This file mostly handles routing. Its most important job is to connect the User, to the databases, and the python files that calculate and handle all the logic. There are four main parts to this file. Firstly, it defines and connects to the database. The schema (or structure) all the data takes is defined here.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.The second important job is to post data from the feedback form to the database. This uses the above schema to ensure data coming from the user is formatted correctly. It is then sent and stored in the database, with errors handled correctly in both the frontend and the backend.

A screen shot of a computer code

AI-generated content may be incorrect.Thirdly it gets the accuracy report from the database, this is like the previous step but in reverse. Returning data to the user, from the database. In this case two variables, true or false, with which our pie chart is made.

Its final job is the most important, as it is the backbone of the entire project. It takes input from the user in the form of a stock code, calls the python file, using that stock code as an input. With correct error handling, and formatting

A screen shot of a computer program

AI-generated content may be incorrect.

### Main.py

Main.py is a crucial part of this project, with its most important tasks including.

calling webscrape.py,



converting the CSV to JSON, cleaning the data as it goes

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

calculating graphs,

A screen shot of a computer program

AI-generated content may be incorrect.

doing linear regression calculations to provide predictions, where arr [0] is a point on the line, and arr [1] is the slope.

A black screen with white text

AI-generated content may be incorrect.

and estimating the price a week from now.



Saves graphs calculated in python as images in that folder, for debugging.

### Webscrape.py

Webscrape.py has somewhat of an inaccurate name. While the initial plan was, in fact for it to be a webscraper, this ended up not working for reasons I explained earlier. Now it simply calls an API, which returns the needed data,

A black background with text

AI-generated content may be incorrect.

And converts the JSON into CSV,

A screen shot of a computer code

AI-generated content may be incorrect.

In theory, this could be done as function returns instead of file writes.

## UI

### Public

The public folder contains what is called “Static HTML” nothing here updates dynamically, it serves only as a container for all the JavaScript in the source folder.

### Source

This is where all the complex JavaScript that the user interacts with is. Much of the data analysis and complexities are here too.

### Index.js

A screen shot of a computer program

AI-generated content may be incorrect.This index serves, similarly to the last one solely to route functions to their correct place.

There is little complex JavaScript here, mostly just consolidating all the other files.

### Data.js

This file handles the user input, the user submits a stock code, the company name is then rendered as the title, and the code is sent as a parameter in a POST request to execute the python file.

### App.js

In this project, app.js serves to render all the graphs that are seen by the user, using a library called Plotly. It reads the JSON data provided by the python scripts I mentioned earlier, in short, it reads the data, and displays the line graph, and the candlestick graph that the user can see. Along with the dynamic features asked for in the advanced requirements.

A screen shot of a computer program

AI-generated content may be incorrect.

### Form.js

Form.js handles the client side of the form, it renders the HTML, ensures all fields are filled, sends the submitted data as a POST request, and then clears the fields for another submission

### Accuracyreport.js

This file submits a get request for the “accurate” variable mentioned above, and compares the number of accurate guesses, versus inaccurate ones.

# Evaluation

While this project does work and does its job well, there are a few areas in which it could improve. Firstly, as it turns out Mr. Malkiel was correct. It is impossible to predict future stock prices based on price history alone. As such, if I were to improve this project, the first place I would start is with how predictions are done. I would bring in an AI model, which could analyse articles about a company, and using machine learning, figure out, over time which words and sentences preceded an increase, or decrease in value. I would have loved to include this feature in the project. In fact, my first ALT was based on a python library called Spacy, which analysed and extracted information from articles or essays. Unfortunately, there was not enough time to include this feature in the project, much less train a machine learning algorithm too.

I would also improve the python, and the way data is sent and processed. As it is now, the data is taken from the Nasdaq API. Converted to CSV, converted to JSON, where it is then used in the Graphs. This, in practice, would not work, because the second a React project is built for production, it mostly loses the ability to read and write local files, and because it is simply a source for all users to interact with at once, you should not use local files anyway. When a second user logs on, the program breaks. The correct way to do this is the same way I handled the form, and the pie chart based on it. That data is sent to and collected from an external database. And if I could redo this project, that is how I would handle the stock data too. This would solve the multiple-user issue, lessen the complexity of the file structure. And the JSON/CSV requirement would still be satisfied because the CSV in between webscrape.py and main.py would not be changed.

# References

<https://nodejs.org/docs/latest/api/>

<https://react.dev/learn/describing-the-ui>

<https://plotly.com/javascript/>

<https://plotly.com/javascript/react/>

<https://stackoverflow.com/questions/51124721/calling-mongoose-from-react-client-side>

<https://mongoosejs.com/docs/guide.html>

<https://stackoverflow.com/questions/52303566/check-if-all-input-fields-are-valid-using-js>

<https://stackoverflow.com/questions/30151213/how-to-make-a-div-element-responsive>

<https://stackoverflow.com/questions/70591567/module-not-found-error-cant-resolve-fs-in-react>

<https://stackoverflow.com/questions/22786068/how-to-avoid-http-error-429-too-many-requests-python>

<https://stackoverflow.com/questions/25743994/syntaxerror-json-parse-unexpected-character-at-line-1-column-1-of-the-json-dat>