Stock market analysis Project Log

**Project log 1 – 17/07/2025**

What I did today:

* Set up the main project folder and has been named “stock price analysis and prediction web app”
* Drafted a 3-week timeline up to Aug 3
* Researched free stock APIs (narrowed down to yfinance and alpha vantage)

What I learned:

* `yfinance` doesn’t need an API key and works directly with Python
* Alpha Vantage has more indicators but limits you to 5 API calls/min on free tier
* The basic flow of websites and how information flows through the system

Next steps:

* Set up the python environments
* Create a simple flask route that returns the JSON data

**Project Log 2 – 23/07/2025**

What I did today:

* Researched alternative APIs to yfinanace due to api unpredictability and unreliability
* Decided to implement LSTM model for advanced stock prediction and researched example implementations for it
* Decided alpha vantage API for beginner friendly and cost effective stock information
* Revised Flask syntax for endpoints and routes

What I learnt:

* Yfinance is not reliable
* LSTM example implementation
* Alpha vantage project to guide through implementaiotn
* Indpeth plan of project and the plan going forward

Next steps:

* Define all endpoints with accurate error handling (i.e. try-else blocks)
* Create file for LSTM implementation
* Research how to merge backend to stock prediction model
* Look into front technologies and design

**Project Log 3 – 26/07/2025**

What I did today:

* Implemented the search bar on the front end
* Styled using CSS
* Organised file structure to make it compatible with flask

What I learnt:

* HTML and CSS styling and syntax
* Flask requirements and structure

Next steps:

* Define end point in in app.py for summary calls from alpha vantage API
* Connect frontend and backend through JavaScript
* Learn JS basics and how to encode frontend functionality
* Error handling

**Project log 4 – 28/07/2025**

What I did today:

* Defined Flask end route for retrieving summary
* Linked with frontend search bar where opon search user can retrieve summary of stock company
* Error handling so that the code doesn’t crash upon malformed input

What I learnt:

* How to link to frontend and backend using JavaScript
* Core structure of implementing backend routes using flask API and key functionality

Next steps:

* Format the summary in the frontend for visual appeal
* Implement seach bar suggestions using API call from backed -- **bare in mind only 5 API calls a minute –**
* Look to impement LSTM model into backedn file and visualize using tools from libraires i.e. MATPLOTLIB

**Project log 5 – 02/08/2025**

What I did today:

* formatted the summaries of the stock upon calling it
* implemented the search bar suggestion using api call and linked it to the frontend

What I learnt:

* how to use query input in the frontend and how to use it in the backend to retrieve the suggestions

**Project log – 03/08/2025**  
What I did today:

Researched different deep learning models for stock prediction, focusing on recurrent architectures like LSTM. Explored tradeoffs between GRU and LSTM for time series.

What I learnt:

Time series models require sequential input windows. Understood that in our case, using a 30‑day lookback window predicts the next closing price. Referenced Alpha Vantage LSTM model – LSTMs usually used for active translations usually take on around 15-25 words to predict the next. I looked to push that limit.

Next steps:

Draft initial LSTM implementation and test with sample data retrieved from Alpha Vantage time series daily data. It has OHLCV where we will use univariate regression and use a single feature – daily close price.

**Project log – 05/08/2025**  
What I did today:

Today I focused on understanding how the input data needed to be shaped for the LSTM model. Stock prices start as a simple 1D array, but the model requires inputs in 3D: (batch\_size, sequence\_length, num\_features). To achieve this, I first created sliding windows of fixed length (e.g., 60 days) to represent the historical context. This made the data 2D in the form (num\_windows, window\_size). Since each timestep has features, I added a third dimension, giving (num\_windows, window\_size, 1) because we only used the closing price as a feature. During batching, the first dimension became the batch\_size. This reshaping was critical, as the LSTM only accepts 3D input, and it ensured that the model could learn the temporal relationships properly. I also learned that while our current setup uses one feature (closing price), the structure allows easy extension to multiple features like trading volume or technical indicators.

What I learnt:

I have learnt how to design and implement a full machine learning pipeline, from preprocessing and batching data to training an LSTM with attention. I now understand how data moves between 2D and 3D shapes (batch, timesteps, features) and the importance of structuring code with safeguards like if \_\_name\_\_ == "\_\_main\_\_":. I also gained experience integrating the model with a Flask backend and frontend for real-time predictions and visualisation, while applying hyperparameter tuning to balance accuracy and efficiency.

Next steps:

Run multiple epochs and tune hyperparameters like learning rate, hidden units, and dropout.

**Project log – 06/08/2025**  
What I did today:

Ran experiments on the LSTM model over multiple epochs. Tested hyperparameters systematically and compared training/validation losses. Changed features such as learning rate, epochs, batch sizes and window sizes. Saw various effects but largest improvement came from increasing the window size to 30 from 20 and reducing the learning rate from 0.001 to 0.0005

What I learnt:

From uni ML experience, saw how hyperparameter tuning significantly impacts model convergence. Learned how to balance overfitting and underfitting.

Next steps:

Add more layers for improved performance and consider advanced methods like attention.

**Project log – 07/08/2025**

What I did today:

Refactored the code into industry‑standard file structure, creating separate utils.py, lstm\_model.py, and checkpoints folder. Packaged functions using `if \_\_name\_\_ == '\_\_main\_\_':` to safeguard when importing.

What I learnt:

Best practices for modularising ML projects. The `main` guard ensures that training functions don’t run unintentionally when importing models.

Next steps:

Integrate attention mechanism to improve accuracy.

**Project log – 10/08/2025**

What I did today:

**Implemented an attention layer on top of the LSTM hidden states to allow the model to learn which past timesteps are most important for prediction.** This change was necessary because a standard LSTM treats all hidden states equally when producing an output, which can dilute the influence of the most relevant days in a stock sequence. By adding attention, the model assigns learned weights to each timestep, effectively “focusing” more on critical past movements while reducing noise from less informative days. The weights are generated by comparing each hidden state to a context vector, passed through a softmax function to ensure they sum to one. This makes the output a weighted sum of hidden states, optimised during training alongside the model’s parameters. As a result, the network captures long-range dependencies better and provides more accurate next-day predictions.

What I learnt:

Through implementing attention, I learnt how models can dynamically assign importance to different timesteps instead of treating all inputs equally. I also gained a clearer understanding of how attention weights are generated, normalised, and optimised during training to improve predictive accuracy.

Next steps:

Generate plots to visualise predictions against actual values.

**Project log – 11/08/2025**What I did today:

**Integrated ML pipeline with backend Flask app.** Connected the trained LSTM model to Flask routes so that predictions could be served dynamically for different stock tickers. Implemented a caching system where models train once per symbol and then load from saved checkpoints for efficiency. Created helper functions to preprocess data, scale values, and prepare unseen windows for next-day forecasting. Integrated plotting functions to generate time-series history, train vs validation comparisons, zoomed validation windows, and next-day prediction charts. Ensured all outputs were saved to static files and linked seamlessly to the frontend for user display.

What I learnt:

Complexities of connecting ML output to backend routes. Created helper functions to transform numpy arrays to torch tensors and scale data for consistent predictions.

Next steps:

Connect frontend to display these plots dynamically.

**Project log – 13/08/2025**  
  
What I did today:

Connected backend with frontend successfully. Stock summary now shows next‑day predicted close and 4 generated plots. Styled layout using CSS with hover effects and responsive design.

What I learnt:

How to use Jinja templating in Flask to pass variables and plot paths into HTML. Improved frontend presentation by creating distinct sections for company summary and prediction graphs.

Next steps:

Polish UI and finalise project documentation.