University of Brighton
CI601 ~ The Computing Project
2022/23 Semesters 1 & 2

Computer Science with Artificial Intelligence BSc(Hons)

First Reader
Nik Polatidis

Second Reader
Andrew Montgomery

Can Machine Learning Forecast Exchange Rates Within a 5% Margin of Error?

An Agile and Cloud-Hosted Approach

David Alexander Beesley
Student Number: 19821050

Product source code is available on GitHub.

Submitted by the 5th of May 2023.

Abstract

This paper investigates the potential of Machine Learning (ML) techniques, specifically Long Short-Term Memory (LSTM) and Fully Connected Neural Networks (FCNN), in accurately forecasting exchange rates. Exchange rate forecasting is a complex and highly volatile time-series problem, with socioeconomic and political factors contributing to its difficulty. The present paper compares the performance of LSTM and FCNN models in predicting exchange rates within a 5% margin of error. To facilitate the research, a software suite, referred to as ExRate, was developed. ExRate consists of three components: the ExRate Service, a Python-based ML project responsible for procuring historical exchange rate data and training time-series models, using TensorFlow, Keras and Scikit-Learn; the ExRate API, a .NET API that automates the execution of the ExRate Service to facilitate user interactions; and the ExRate Frontend, a Single Page Application built using React, which serves as a showcase for the ExRate API. The ExRate product incorporates these technologies to create a comprehensive platform which can access model performance for exchange rate forecasting. The results of this investigation show that the LSTM model outperforms the FCNN model; the LSTM model often produced predictions within a 5% margin of error. By adopting Agile principles and DevOps practices, the software suite demonstrates the advantages of following industry software development standards to deliver exchange rate forecasting as a service. Overall, the findings of this study contribute to the understanding of ML techniques in forecasting exchange rates and provide a framework for future research and practical applications in the field.

Contents

Introduction

Foreign exchange rates, the value of one currency expressed relative to another currency, are important economic indicators for investors to assess a country (or region). The ability to accurately predict the future state of these rates would have a profound impact on international trade, investment, and various fiscal activities. Accurately forecasting exchange rates has long been a subject of research in economics, however, by utilising recent advancements in computer science, especially in Machine Learning (ML), there is potential for breakthroughs in this area.

ML employs a range of algorithmic and statistical functions to derive meaning from data, which can be applied to solve various problems including data classification, natural language processing, computer vision, anomaly detection, and time-series forecasting (Sarker, 2021). Exchange rate forecasting is fundamentally a time-series problem, much like other applications that have benefited from ML, such as road traffic (Awan, Minerva & Crespi, 2020), healthcare service demand (Piccialli, et al, 2021), and weather forecasting (Karevan & Suykens, 2020). However, unlike these examples, predicting exchange rates is considerably more complex due to external factors. This complexity is also seen in other financial data streams such as predicting stock prices (Althelaya, El-Alfy & Mohammedl, 2018) and inflation (Peirano, Kristjanpoller & Minutolo, 2021). The rate at which one currency gains or loses value against another heavily depends on socioeconomic and political factors, which are both difficult to extrapolate and model (Singh, Bhatia & Sangwan, 2007).

A closer look at these factors reveals a dichotomy between evolving socioeconomic trends, international and national politics. When a particular country or economic region experiences low inflation and rapid growth, foreign investment would increase, thus increasing the relative value of its currency. In contrast, when a country or economic region experiences political instability, economic downturn, unemployment and conflict then foreign investment would falter (Hopper, 1997) and the value of its currency decline. The sharp drop in the value of the Pound Sterling, by approximately 15%, following the United Kingdom's referendum on European Union membership is a prime example of this interplay (Korus & Celebi, 2019); showing how confidence in an economy can quickly change following a sudden change in a country's political landscape despite no material change in this immediate economic standing.

The non-linear and capricious nature of financial data associated with such phenomena has been noted in previous research (e.g., Meese & Rogoff, 1983; Babu & Reddy, 2015), leading to a vacuum where no particular approach for forecasting has yet been universally accepted within academic and enterprise circles (Yasir, et al, 2019). Despite the apparent intractability of forecasting exchange rates, the pursuit of precise and reliable forecasting remains a compelling objective due to the potential financial gains and profitability that a breakthrough in this area could return (Shen, et al, 2021).

Literature Review

To further contextualise this research, it is worth examining the history of exchange rate forecasting. With the advent of the computer, the 1970s and 80s saw the first research conducted around computer-aided forecasting. An example of this research was a paper by Stephen H. Goodman, which compared the latest techniques used by various companies and organisations at the time, many of which used rudimental computer processes to derive their forecasts (Goodman, 1979). At the time, Goodman evaluated the performance of ten services which provided exchange rate forecasting. All of these performed poorly by modern standards, as they failed to capture patterns in historic rates and the methods used did not account for the vast array of factors. Goodman concluded that although computers have made considerable progress in generating predictions, they have not yet reached a level where their predictions can be considered significantly valuable. Time-series forecasting techniques have seen significant growth since Goodman's work, with incremental innovations built on a growing body of work within the emergent field of ML. Among these techniques are Long Short-Term Memory (LSTM), Fully Connected Neural Networks (FCNN), Convolutional Neural Networks (CNN), and Support Vector Machines (SVM).

LSTM models, a type of Recurrent Neural Network (RNN), were specifically designed to capture long-term dependencies in time-series data while mitigating the vanishing gradient problem (Graves, 2012) present in traditional RNN models. They accomplish this through the use of gating, a process by which irrelevant data is retained while less relevant or anomalous data points are dropped (Yunpeng, et al, 2017). This design choice sets LSTM apart from traditional RNNs and makes it particularly useful for tasks involving time-series data and sequences, such as natural language processing and speech recognition (Kim, El-Khamy & Lee, 2017; Preethi, 2021), as well as financial forecasting. By retaining older yet relevant data, LSTM models can better recognise underlying patterns in time-series datasets, making them a popular choice in many ML applications.

The advantages of LSTM usage are put forward in a paper by Mang Ma and Zhu Mao which detail the positive impact of the LSTM structure on delivering accurate predictions. Their research built on an understanding of LSTM put forward in a paper on long-term recurrent convolutional networks for visual recognition and description (Donahue, et al, 2015). Ma and Mao further refine the LSTM technique to address a task involving the degradation process of ball bearings, a problem which mirrors the complexity of exchange rate forecasting. However, their research did not put forward any alternative approach or comparison, an area they identified for further research (Ma & Mao, 2020).

In addition to LSTMs, FCNN models are widely used in ML for various tasks and are often compared with LSTM models, even though they are not specifically designed for natural language processing and time-series predictions. FCNNs are suitable for datasets with fixed sizes and well-defined structures, making them useful for tasks like exchange rate forecasting (Ajit, Acharya & Samanta, 2020). These models are flexible and can be applied to a variety of

ML problems, including time-series forecasting, but the optimal hyperparameters depend on the problem and the dataset characteristics. The configuration of FCNNs involves a chain of nodes forming a layer, which can be stacked in different ways to improve accuracy (Peng, et al, 2021).

A study on the effective use of FCNN for time-series forecasting, led by Borovykh, draws a focus on generalisation techniques used to finetune an FCNN model for the task of time-series forecasting. This research found that over-parameterised networks may only fit a complex function on the past, rather than extrapolating patterns in past data to make informed decisions on the future of that data. Their research concluded that a simple and optimised model would often garner positive generalisation and mitigate the dangers of overfitting (Borovykh, Oosterlee & Bohté, 2019). Overfitting occurs when an ML model learns noise in the training data rather than the true underlying patterns and relationships between features, resulting in models which, on the surface, perform well yet fail to react as expected to unseen data. Despite this and as FCNN models are widely used and easily curated with modern development tools, they can present a benchmark for comparison.

Although CNN and SVM models have been successfully used in various applications, they are not always the best choice for time-series forecasting. This is because CNNs are primarily designed for image recognition tasks where the input data has a spatial structure (Evans, et al, 2018), while SVMs are most effective in text-based classification problems (Joachims & Thorsten, 2005) where there is no temporal dependency among the input data. As such, time-series forecasting requires models that can capture the temporal dependency and trends in the data, such as LSTM and FCNN.

Therefore, the current paper aims to compare the effectiveness of LSTM and FCNN models for exchange rate time-series forecasting. This study will conduct a comprehensive comparison between these two techniques to determine the superior method. To ensure a fair and rigorous comparison, the empirical analysis will consider various model configurations and hyperparameter tuning, with only the most effective combinations selected for analysis. The study's objective is to investigate whether either of these algorithms can predict exchange rates within a 5% margin of error.

To conduct this investigation a system is necessitated to run proposed models on historical exchange rates and provide a forecast. Functional requirements for this system are defined as the minimum technology needed, while non-functional requirements can aid accessibility, usability, and extend capabilities. An Agile development process was followed, with a focus on incorporating DevOps (a methodology which combines development and operations, both required for end-to-end software engineering) to ensure a smooth flow between development and deployment. The resulting software suite showcases cloud hosting services and online interconnectivity, while also being scalable, robust, secure, and professionally executed, following industry standards.

Methodology

Deliverables

For the purpose of this project, the following requirements have been identified which will act as the guiding principles for development:

Functional Requirements:

- The software artefact should be able to take two given exchange rates as inputs and generate a week-long forecast for their exchange rates.
- It should achieve this by utilising LSTM and FCNN models. The artefact should be able to employ these models to procure a forecast.

Non-Functional Requirements:

- The software artefact should be designed in a modular way that allows computer scientists to create and run any other ML model on the platform to enable further research.
- The forecast generated by the software artefact should be made available through an online system. This will allow users to access the forecast and use it in their own applications.
- The software artefact should also have a frontend to demonstrate how developers might use the platform in their own applications.

Links to the source code (hosted on GitHub) and frontend demonstration (hosted on Netlify) are in *Appendix 1*.

Technical Overview

System Architecture and Components

The software suite produced for this project consists of three distinct components, herein collectively referred to as ExRate. The centrepiece of this suite is the ExRate Service, a Python-based ML product designed to procure the necessary historical exchange rate data and train a time-series model on this data for forecasting purposes. The ExRate Service accomplishes this by gathering data from an open Application Programming Interface (API) that returns exchange rates between a base and target currency, within a defined range, in a JavaScript Object Notation (JSON) format (See *Appendix 2a*). For the purposes of development, the range of exchange rates collected is restricted to the past year from the point at which it is invoked, but this range can be easily changed to include more or fewer data points for training. After gathering this data, it is processed, normalised, and shaped to fit the

model's parameters, after which the training process can commence, and the predicted values acquired.

The ExRate service can be executed as a standalone console application without arguments, which enables a Command Line Interface (CLI) for collecting user inputs for the desired base, target currencies and model selection. However, if arguments for these inputs are provided when executing the script, the CLI is bypassed and only the forecast is returned. The benefit that standalone functionality enables allows for direct debugging and testing of features while streamlining the communication between the other components of ExRate.

The ML component of the ExRate Service leans heavily on the functionality provided by these external libraries:

- TensorFlow, an open-source ML framework developed by Google, allows developers to compose, build and train custom models for a variety of ML tasks such as image classification, natural language processing and time-series forecasting.
- Keras, built on top of TensorFlow, provides a simplified interface by providing common pre-compiled layers and popular activation functions, which can be chained together to build complex models for a given use case.
- Scikit-Learn aids developers with data-related tasks such as data mining and analysis, including algorithms for classification, regression, clustering and dimensionality reduction. Together, these tools lay the foundation of the ExRate Service by providing the tools used to build and train the models used.

In addition, the Service also employs other libraries for different tasks. For instance, Request is employed to fetch data from the open API source, while NumPy is used to access the necessary mathematical operations for data normalisation and reshaping. Moreover, Matplotlib is utilised to generate evaluation and prediction graphs, which help to analyse the model's performance and identify areas that require further optimisation and tweaking.

The second component of ExRate is a .NET API, written in C#, which automates the execution of the ExRate Service, where the script's output is passed to the API and returned as JSON. This API comprises two endpoints: a POST request that provides an instant process token, and a GET request that initially returns an `HTTP 404 - Not Found` response until the execution of the ExRate Service script is complete. Once the script has been executed, the GET request will return the forecast values in JSON format. This design choice was made to remove the risk of encountering an `HTTP 524 - A Timeout Occurred` response due to the Python script taking longer than the default 100 seconds to complete. By implementing tokenisation, the script can run in the background and respond only when the end user retries with their token, thereby ensuring a more stable and reliable response mechanism.

To mitigate the limitations of local development where the variability of root folder location of user systems can disrupt the functionality of the .NET API if not correctly configured, the ExRate Service and API are containerised using Docker technology. Docker, a software

platform, allows developers to build and run applications inside a portable environment, not unlike a virtual machine, specifically to standardise behaviour between applications and to create a deployable artefact for cloud-hosting. The initial benefit of containerisation for this product is to insulate the API's dependence on the Service. As the API expects to find the Service in a specific location, this dependency can break if a user does not properly configure the `appsettings.json` file with the correct paths for their local environment. To avoid this, containerisation can be incorporated to hardcode the location of the Service within the container itself. Beyond this initial benefit, containers are easily deployed to cloud hosting platforms. Ultimately, the use of containers ensures the behaviour between the two components is as intended and reduces the complexity of cloud deployment, justifying its use within the project.

The third and final component of the ExRate suite is its Frontend. The ExRate Frontend is a Single Page Application (SPA) built using the popular JavaScript framework, known as React. For this project, React, developed by FaceBook in 2011, has been paired with TypeScript, which is a strongly typed superset of JavaScript. The use of TypeScript further enhances the reliability and robustness of the SPA, as it is able to identify potential errors at compile-time, resulting in a more secure and maintainable codebase. This is a well-established practice in modern web development and serves to optimise the workflow of the application's development (Choi, 2020). The Frontend serves as a showcase which calls the ExRate API and demonstrates how it can be used to display the forecast in a modern use case, following industry standards for web development (see *Appendix 7a*).

Technology Stack Justification

Python was chosen for the ExRate Service due to its vast ecosystem of tools and wide range of supporting libraries. The ExRate Service was developed using the JetBrains Integrated Development Environment (IDE) for Python known as PyCharm, which has a suite of features to support development such as syntax highlighting, library management, environment configuration and debugging capabilities. The Python community is very active, with a vast amount of documentation and resources on popular libraries. One alternative to Python considered for this task was the R language, primarily used for statistical computing, data analysis and graphics. R has a variety of libraries for ML operations, including Caret, RandomForest and Glmnet. However, R lacks the flexibility and execution speed of a fully featured high-level interpreted programming language like Python, and its use in the project would restrict the functionality and efficiency of the ExRate Service component (Topçuoğlu, et al, 2021).

The ExRate API utilises .NET which has excellent tools and established design patterns which streamline development and integrate well with other Microsoft services such as Azure. This part of the project was developed using Visual Studio (VS), an IDE also developed by Microsoft, specifically for .NET C, C++, C# and F# applications. VS provides a wide selection of tools, including syntax highlighting, IntelliSense, a solution explorer, a test

runner, and a package manager with the most crucial among them being the built-in debugger. This debugger allows developers to jump into sections of code using breakpoints and step through and into relevant methods and classes to analyse the current state of objects and types in real time. This gives developers insight as to what may be causing bugs within the code.

While an API, especially one with RESTful architecture, can be developed in a wide variety of languages (e.g., Ruby on Rails, Python with Flask and Golang), .NET was selected for its cross-platform support and it being a strongly typed language, meaning there is no dependence on and uncertainty of type inference, unlike many of the alternative languages. .NET is also known for its high-performance due to its utilisation of Just-In-Time (JIT) compilation (Sarcar, 2022). Building on these benefits, the .NET framework was an advantageous choice given the extensive documentation published by Microsoft, alongside an active community of developers ready to aid and communicate design patterns used in enterprise-level applications (see *Appendix 2b*).

As for the Frontend, similar weight was placed on strongly typed languages, hence the decision to use React with TypeScript. Beyond this, frontend competitor JavaScript frameworks such as Angular, Vue and Svelte can also be paired with TypeScript. However, React is known for its superior performance, large community and component-based architecture, which further justify its use within ExRate. While there are further alternatives, outside of JavaScript frameworks, such as traditional Model-View-Controller (MVC) web applications made with .NET or Java, these are better suited to high-level complex solutions requiring a stable and maintainable backend. In the context of this particular software implementation, React's absence of backend functionalities may be deemed acceptable since it serves primarily as an illustrative example of how the ExRate API can be leveraged in a contemporary computing environment, where client-side execution and the resulting performance enhancements take precedence.

The chosen IDE for this React Frontend was VS Code as it is widely adopted among frontend developers, this has led to an ecosystem of extensions for developing a multitude of languages and frameworks. Alternatives to VS Code, such as JetBrains' WebStorm, Brackets and Vim, do not have the same flexibility and support for producing React SPAs.

The project's development relied heavily on Git for version control. Git is the de facto tool for tracking change and ignoring auto-generated or system-specific files in a project, whilst including features like branches, merges and conflict resolution. While Git is designed for development teams to work on the same repositories simultaneously and merge changes seamlessly, it is good practice even for solo developers.

While there are some alternatives, Docker is the industry standard for containerisation and provides a comprehensive set of tools, features and capabilities that are not present in other containerisation platforms such as Podman. Podman, built to be Docker-compatible, is a security-focused containerisation tool which would not be suitable for this project as its complexity and lack of community support would make developing with it more challenging

and waste time and resources to set up. Docker, on the other hand, has detailed documentation and an abundance of community support, making it a better choice for the goals of this project.

Development and Deployment

Legal and Ethical Considerations

It is imperative that the ExRate application complies with relevant laws, specifically the General Data Protection Regulation (GDPR) within the European domain and the California Consumer Privacy Act (CCPA) for North American compliance. However, as there is no system within the product which collects or manages sensitive data, this requirement is satisfied. Moreover, the ExRate application is published in the public domain under the General Public License (GPL) v3.0 to offer freedom for members of the public to run, study, share and improve the software. This ensures transparency and, given the nature of financial forecasting, does not gatekeep or paywall any possible advancements made, in keeping with the academic spirit of this project. It must also be made clear that the forecasts it generates do not constitute financial advice nor act as a recommendation for investors.

Risk Analysis

Bar world-changing events, the significant risks which may have caused delays were identified as the health of the candidate, time mismanagement and external factors such as the candidate's personal circumstances regarding their ability to output work this academic year. Other considerations are technical; the cloud-hosting service chosen for this project is a reliable platform for deployment, but uptime cannot be guaranteed. The risk of data loss is also present. To mitigate against the risks identified as within the candidate's remit, the candidate will ensure that core functionality was prioritised and done well before the second-semester deadline. Once functional requirements were met, the candidate focused on the non-functional requirements. Moreover, while cloud-hosting services are beyond the candidates' control, the likelihood of a failure on a major platform is considered low given the many industries and organisations which depend on major cloud-hosting platforms daily. To ensure the integrity of data, documentation was stored in the cloud (Google Drive) and the up-to-date source code was hosted on GitHub. All data was also stored locally and backed up weekly.

Agile Development

The principles delineated in the Agile Manifesto (Beck, et al, 2001) set out guidelines for project management. Software development is one such area where Agile techniques have been applied and have become a standard for development teams across the industry, moving away from traditional Waterfall methodologies (Dima & Maassen, 2018). The rise of Agile software development can be attributed, in part, to technical advancements in the distribution

of digital products. The replacement of legacy physical storage media, such as Floppy Disks and CDs, has been supplanted by Software as a Service (SaaS) models, where digital products are distributed via an advanced computing infrastructure (Dubey & Wagle, 2007). These technological progressions have facilitated the shift towards Agile methodologies, which promotes adaptability to changing requirements and unforeseen technical limitations, enabling flexible and rapid product delivery through continual improvement cycles.

One key principle of Agile is 'release early, release often' (Koch, 2004). This guideline endorses software products to be tested and delivered frequently in their Minimum Viable Product (MVP) state and iterated upon post-launch with new features or bug fixes. However, a disadvantage of this is that many end-users may feel the product or service lacks the promised quality at launch, this is particularly prevalent in the video game industry (Mertens, 2022).

The Agile approach is a mindset that does not require specific tools. However, in practice, Agile development often includes the use of planning tools such as Kanban Boards (Polk, 2011). Kanban Boards break down features into user stories and assign points based on their time complexity using the Fibonacci sequence. Developers then choose stories from the backlog (see *Figure 1*) and progress them through different stages, typically including analysis, development, peer review, testing, and deployment. Each stage is represented by a 'doing' and 'done' column, with a 'definition of done' provided for each.

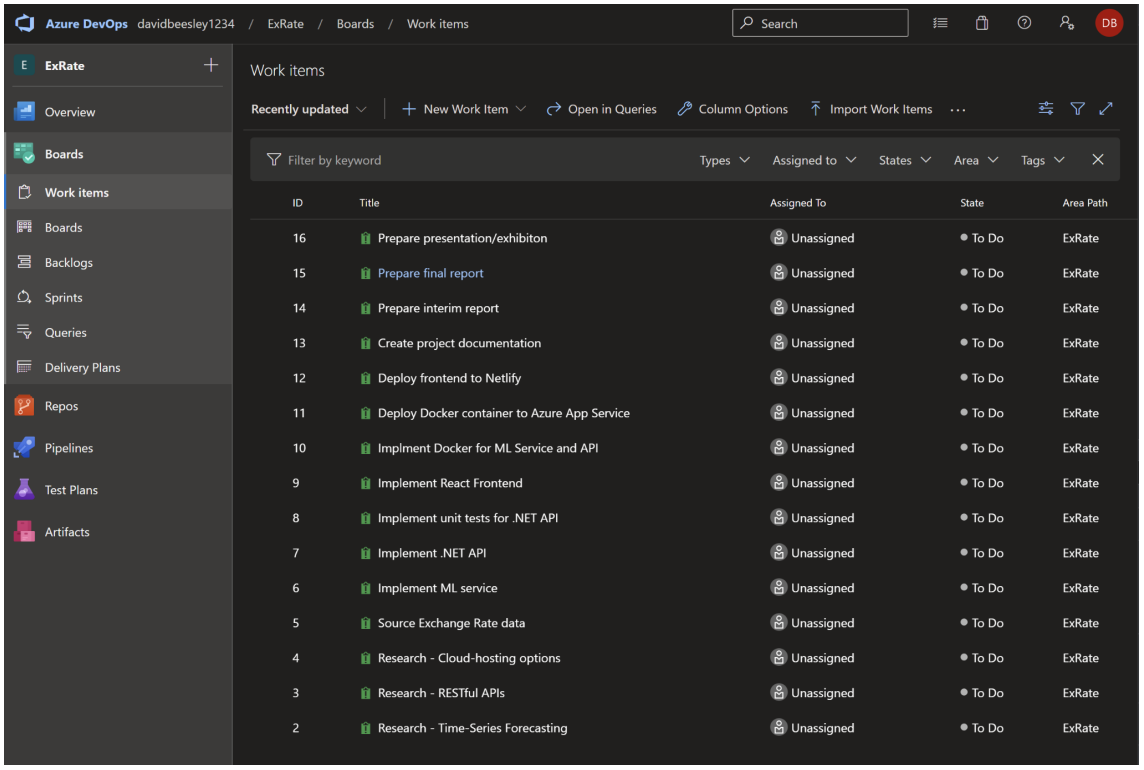


Figure 1 - Azure DevOps backlog for the ExRate project at the start of Development in September 2022.

For this project, Microsoft's Azure DevOps was used as it provides a platform for hosting and managing git repositories, as well as tools for supporting Agile development, including a Kanban Board, shown in *Figure 2*. Azure DevOps also supports Pipelines for interacting with other Azure cloud services to create a structure for Continuous Integration/Continuous Deployment (CI/CD) practices. It is important to note that the product's MVP will be the functional requirements previously mentioned. The development will prioritise MVP delivery over non-functional requirements, as per the teachings of agile.

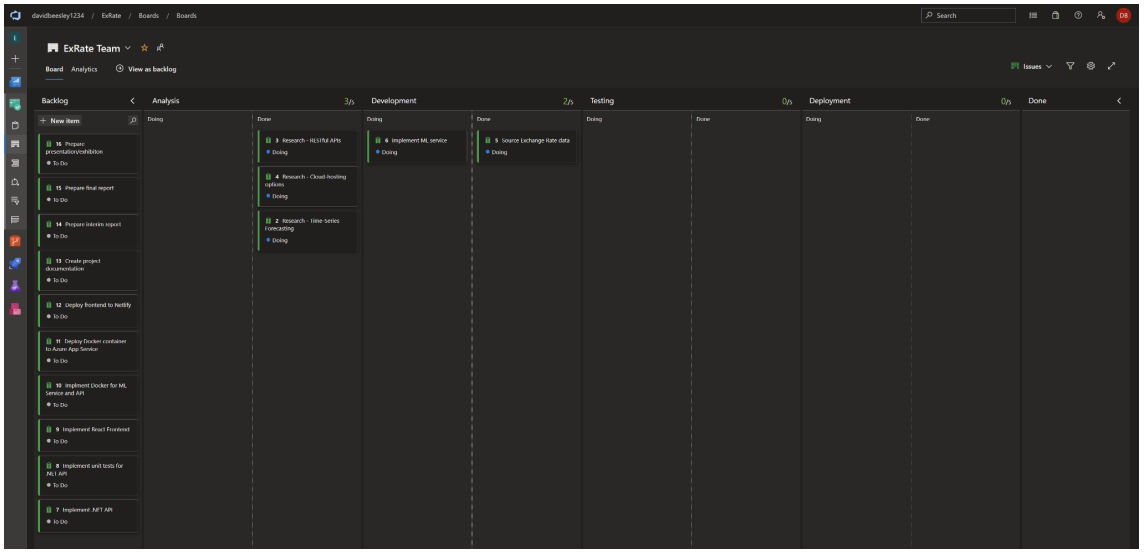


Figure 2 - The state of the Kanban board as of 15th October 2022.

To demonstrate how work items move across the board, *Figure 2* displays a snapshot from mid-October 2022, shortly after initial development for the project commenced. Further snapshots of this board, documenting monthly progress are provided in *Appendix 3*.

Containerisation

To set up a container for the ExRate Service and API, Docker and the Docker Desktop Client were installed as these provide the tooling to build and run containers locally. To build this container, a **`Dockerfile`** was added to the project with relevant instructions. The syntax for composing a **`Dockerfile`** can be found in the documentation provided by Docker Docs (see *Appendix 2b*). *Figure 3* shows the **`Dockerfile`** placed in the root folder for the ExRate project.

```
FROM mcr.microsoft.com/dotnet/sdk:6.0 AS package
WORKDIR /app

COPY ExRate_API .
RUN dotnet publish -c Release -o out

FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS final
WORKDIR /app

RUN apt-get update && apt-get install -y --no-install-recommends \
    python3.9 \
    python3-pip \
    && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

RUN pip install numpy pandas requests python-dotenv tensorflow scikit-learn
keras matplotlib seaborn

COPY ./ExRate_Service ExRate_Service
COPY --from=package /app/out ExRate_API

ENV TF_CPP_MIN_LOG_LEVEL=1

ENTRYPOINT ["dotnet", "ExRate_API/ExRate_API.dll"]
EXPOSE 80
```

Figure 3 - The `Dockerfile` used to containerise the ExRate API and Service.

Breaking down this **`Dockerfile`**, the first step uses the .NET 6 base image from Microsoft, the relevant files for the .NET API are copied into the working directory. Following this, the `dotnet publish` command is executed to build and publish the API ready for release. Once completed a new .NET 6 image is pulled from Microsoft where the publish directory is copied too and Python, along with the ExRate Service's required packages, are installed. The Service scripts are then copied to the working directory and, finally, an entry point is set and port 80 is exposed to access the contained API. This **`Dockerfile`** can then be used to build and run the container locally, using the appropriate console commands, or built and deployed in Azure.

Continuous Integration and Manual Deployment

A core benefit of Azure DevOps is the utilisation of automation throughout the development process. Continually integrating features and improvements from development branches into production is an essential part of the modern software development cycle (Airaj, 2017). Within Azure DevOps, this is managed by so-called 'Pipelines', which are instructions contained in **`.YML`** files.

```yaml
trigger:
- main
resources:
- repo: self
variables:
  imageRepository: 'exrate'
  containerRegistry: 'exrate.azurecr.io'
  dockerfilePath: '$(Build.SourcesDirectory)/Dockerfile'
  tag: '$(Build.BuildId)'
  vmImage: 'ubuntu-latest'
stages:
- stage: Test_Stage
  displayName: Test Stage
  jobs:
  - job: Test
    displayName: Test
    pool:
      vmImage: $(vmImage)
    steps:
    - task: DotNetCoreCLI@2
      displayName: Run .NET Tests
      inputs:
        command: 'test'
- stage: Build_Docker_Image_Stage
  displayName: Docker build and push stage
  jobs:
  - job: Docker_Build
    displayName: Docker build and push
    pool:
      vmImage: $(vmImage)
    steps:
    - task: Docker@2
      displayName: Docker Build and push image to container registry
      inputs:
        command: buildAndPush
        repository: $(imageRepository)
        dockerfile: $(dockerfilePath)
        containerRegistry: $(dockerRegistryServiceConnection)
        tags: $(tag)
```

Figure 4 - The azure-pipelines.yml pipeline which automates continuous integration.

For this project, a pipeline was set up which runs whenever a new git commit is made to the main branch. *Figure 4* above shows this automation script, which is split into stages; each stage having a particular job. This script will run the .NET unit tests, failing if they fail, this guards production against broken code, then builds and deploys the Docker container to the appropriate Azure Container Registry. *Figure 5* shows a successful pipeline execution and *Figure 6* displays the test results for that run.

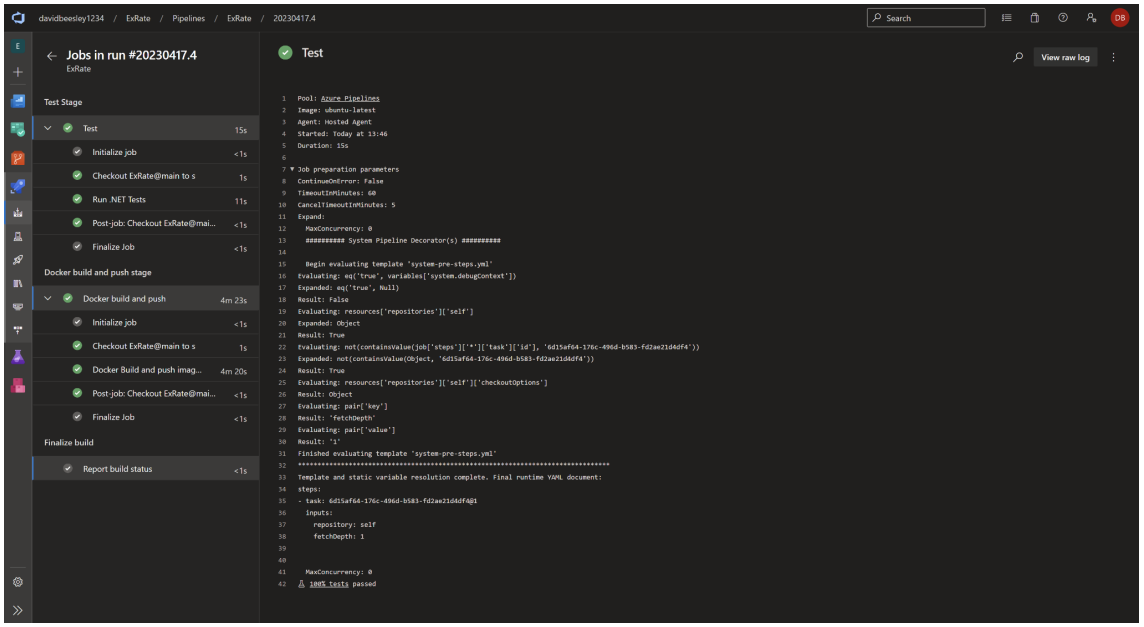


Figure 5 - A successful pipeline execution for the ExRate Service and API container.

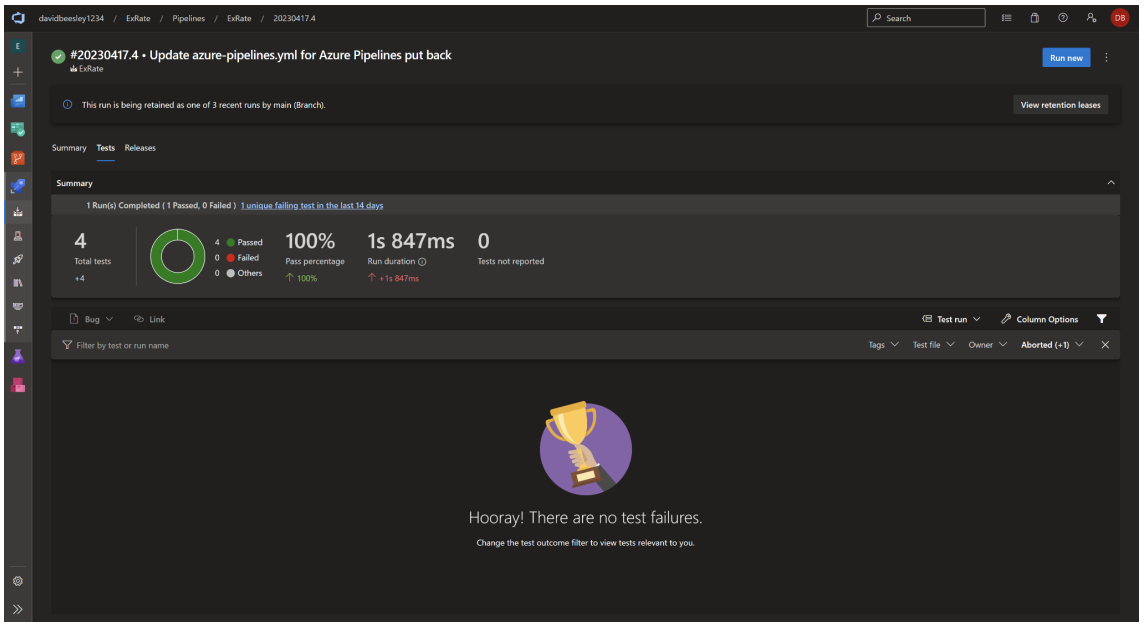


Figure 6 - The test results of the .NET unit tests in the ExRate API application.

The artefact produced by the pipeline can then be manually released to the desired Azure App Service where the API will be hosted. *Figure 7* shows a successful manual release to the App Service and *Figure 8* shows the overview tab for this App Service.

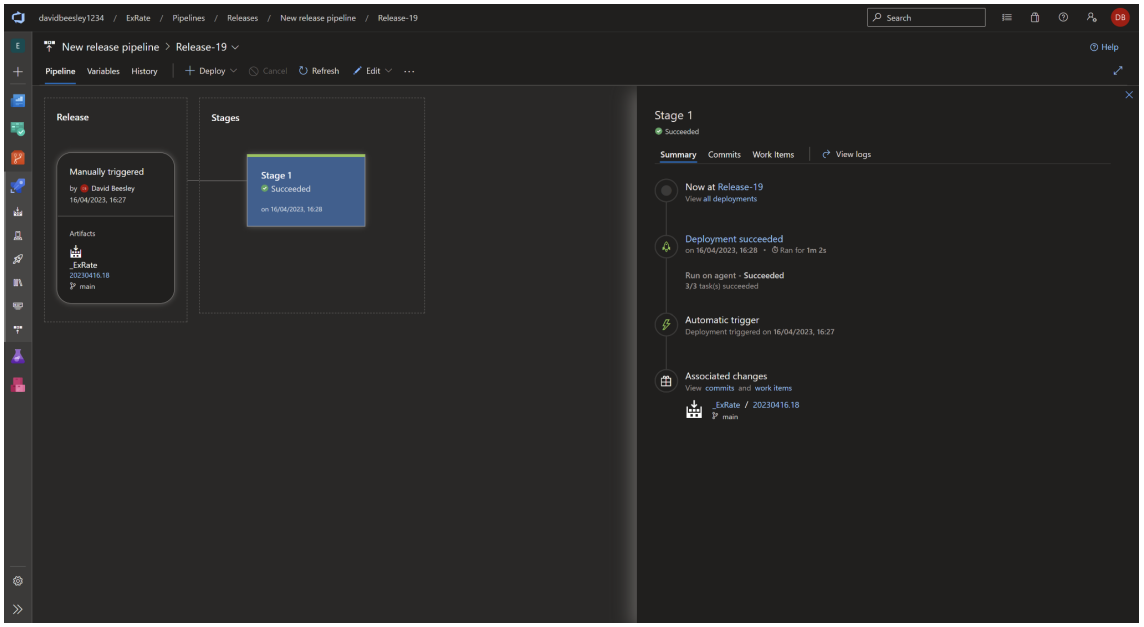


Figure 7 - A successful release using the artefact produced by the most recent pipeline run, this is a container image which is pushed to the App Service.

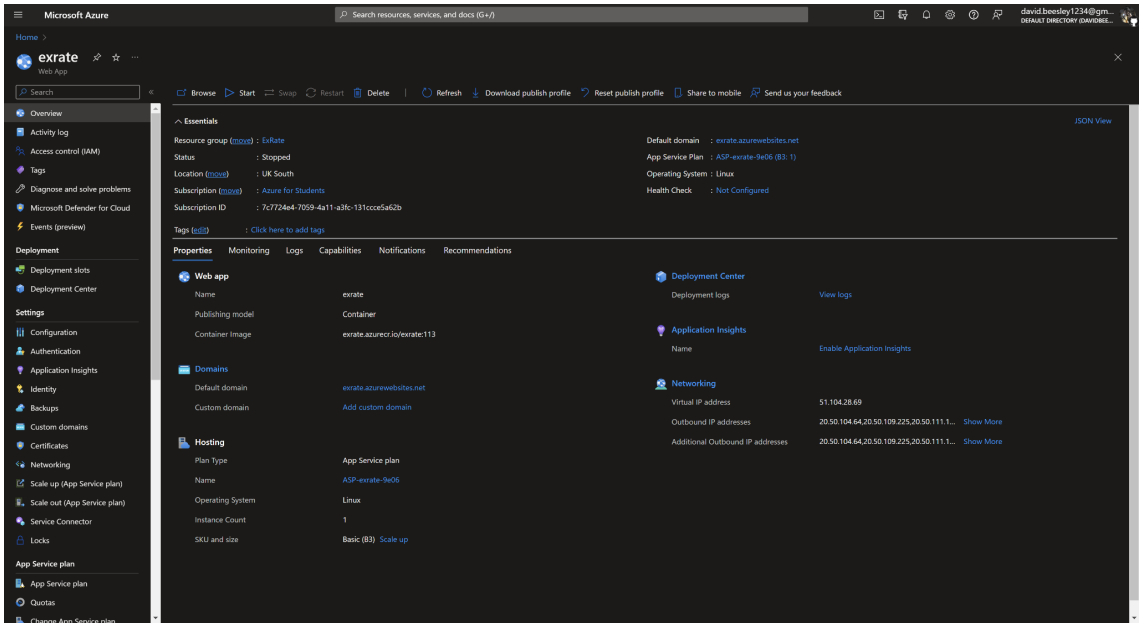


Figure 8 - The overview page for the ExRate App Service where the API is hosted once released to production. This service will run the most recent Docker image which has been manually released.

Testing

Throughout development a key part of the process is testing. Testing can take various forms depending on the project requirements and complexity of the code. Some projects may only rely only on informal testing, this is where the developers themselves run the code and validate correct functionality through observation. However, informal testing like this is often

imperfect as human error can be introduced which may result in bugs and malfunctions, due to developer oversight. To prove the functionality of a program, unit tests can be written that isolate a particular code component (such as a class or method) usually by mocking out external dependencies to create a controlled environment. Assertions can then be made that the behaviour of the component under numerous settings (usually following a true positive, true negative, false positive and false negative design) returns as expected and intended. These are called unit tests and within the ExRate API, a test suite was written using Nunit, a NuGet package for testing.

Figure 9 shows a unit test for the **`GetExRateForecastController`** class which handles the API endpoints, following the TripleA design pattern. This tests that the controller correctly returns an **`HTTP 400 - Bad Request`** when the endpoint is called without correct parameters.

```
[Test]
public async Task
StartProcess_ShouldReturnBadRequest_WhenInputParametersAreInvalid()
{
    // Arrange
    var baseCurrency = "";
    var targetCurrency = "";
    var modelType = "";

    // Act
    var result = await _controller.StartProcess(baseCurrency, targetCurrency,
modelType);

    // Assert
    Assert.IsInstanceOf<BadRequestObjectResult>(result);
}
```

Figure 9 - The true negative Nunit test for the `StartProcess()` method in the controller.

It should be mentioned that unit tests are not the only tool available for proving the functionality of a given code base, they should be used in conjunction with integration and system tests as well as user testing. However, given the scope of this project, testing was not fully implemented due to its rapid development, evolving nature and focus on ML components.

Machine Learning Methodology

Preprocessing

The ExRate Service gathers exchange rate information from an open API, which is stored in JSON format as key-value pairs. The date is the key and the exchange rate, between the base currency and the target currency for that day, is the value (shown in *Figure 10*).

```
{
        'Base': 'GBP',
        'Target': 'USD',
        'Rates':
        {
                '2022-04-01': 1.31139,
                '2022-04-02': 1.31145,
                ...
                '2023-04-01': 1.310333,
                '2023-04-02': 1.311406
        }
}
```

Figure 10 - An example of what JSON the API used to collect exchange rate data, hosted by https://api.apilayer.com, returns.

An example of these normalised values is shown in *Figure 11*. Normalisation is an important step as it scales down and standardises input features, between 0 and 1, which helps to reduce overfitting (Mohamed & Schuller, 2022) when training the model.

```
[0.0019998]
[0.0017932]
[0.0056148]
[0.0019402]
[0.0142575]
[0.0170797]
...
```

Figure 11 - The first six rates post-normalisation.

The equation in *Figure 12* represents the logic behind the normalisation process. This functionality is imported from the **`MinMaxScaler`** class located in the **`sklearn.preprocessing`** external module.

$$\mu = \frac{x - \min}{\max - \min} \quad \text{and} \quad \beta = \mu \cdot (\max - \min) + \min$$

Figure 12 - An equation for MinMax Normalisation.

The operations behind the **`MinMaxScaler`** process are shown in *Figure 12*, where x is the input data, min and max are the minimum and maximum values in the input range, therefore β is the scaled output of values between 0 and 1. After normalisation, the input data is reshaped as $[365, 7, 1]$. This is required as the model built by the service expects NumPy arrays of this shape.

This process is implemented in the **`DataNormaliser`** class shown in *Figure 13*; the **`normalise()`** function takes in a list of daily exchange rates as input and first converts it into a NumPy array with the shape of $(-1, 1)$ using the reshape function. The `-1` argument in

`reshape` indicates that the size of that dimension should be inferred from the length of the array and the other dimension, which is 1 in this case.

```
class DataNormaliser:
    def __init__(self):
        self.scaler = MinMaxScaler()

    def normalise(self, rates):
        rates_array = np.array(rates).reshape(-1, 1)
        return self.scaler.fit_transform(rates_array)

    def denormalise(self, rates):
        return self.scaler.inverse_transform(rates)
```

Figure 13 - The DataNormaliser class from the ExRate Service, written in Python 3.10.9.

Once the **`rates_array`** is created, it is passed to the **`fit_transform()`** function of the **`MinMaxScaler`** object, which scales the values to a range of 0 to 1. This scaled array is then returned as the output of the **`normalise()`** method.

After model training, normalised predictions are returned. To convert these back to the input scale pre-normalisation the **`denormalise()`** function can be used to invert the transformation and return the data to its intended scale.

Model Selection and Tuning

The ExRate Service was built to be modular so users can choose which model is used to train and make predictions. For this project, two models were selected for comparison and to demonstrate the modularity of the Service, namely LSTM and FCNN.

LSTM algorithms consist of three essential gates. The initial gate manages the input, the second gate oversees the forget mechanism, and the third gate controls the output. These gates play a crucial role in regulating the information flow within the LSTM cell, enabling it to selectively retain or discard information from the preceding time step and determine the information to be transferred to the subsequent time step.

In *Figure 14*, i is the input, f is the forget and o is the output gate; t represents the current time step in a sequence, W is the weight matrix, h is the hidden state at the previous time step (denoted by $t - 1$), x is the current step input and b is the bias vector for the gate.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

Figure 14 - The three LSTM gates.

These gates work together to control the flow of information within each time step. The use of a sigmoid function in each gate helps to determine the relevance of information, essentially providing a gating mechanism, indicating the degree of importance of that information for the next step in the sequence (Gers, Schmidhuber & Cummins, 2000).

Alternatively, the typical algorithmic structure of an FCNN model involves three operations; matrix multiplication, non-linear activation and element-wise addition, traditionally followed by backpropagation and optimisation steps. *Figure 15* shows the first step within a typical FCNN model, where z is the weighted sum of inputs, W is the weight matrix of the first hidden layer, x is the input vector and b is the bias vector (Liu, et al, 2015). Functionally, the input layer is linked to the first hidden layer by weights, each node (or neuron) of the hidden layer takes the input sum of the previous layer, which is calculated by taking the dot product of the vector and weight (including the bias addition).

$$z = W \cdot x + b$$

Figure 15 - The matrix multiplication expression.

The next step is to perform an activation function, this filters out certain results from the previous step. Introducing non-linearity at this stage allows the model to identify complex data patterns and relationships between inputs. In *Figure 16*, a represents the output of this activation function, which uses the exponential of $-z$ derived from the matrix multiplication, depicted in *Figure 15*.

$$a = \frac{1}{1 + \exp(-z)}$$

Figure 16 - The nonlinear activation function.

The product of a can then be applied to the final step in the FCNN architecture: element-wise addition. This produces the final output of the model once all hidden layers have been stepped through. *Figure 17* shows how a is used along with the model's weights and bias to drive an output, which is finally put through an activation function, commonly the Rectified Linear Unit (ReLU) function which is shown in *Figure 18*.

$$z = W \cdot a - 1 + b$$

Figure 17 - Element-wise addition.

$$f(x) = \max(0, x)$$

Figure 18 - The ReLU activation function.

Model Architecture

In order to incorporate an LSTM model into the ExRate Service, a custom class is defined that makes use of the Keras deep learning library. The class, shown in *Figure 19*, is designed to provide a simple and flexible interface for creating, training, and evaluating LSTM models for time-series prediction tasks.

```
class LSTM:
    def __init__(self, input_shape):
        self.model = Sequential()
        self.model.add(KerasLSTM(64, input_shape=input_shape, activation='relu',
kernel_regularizer=regularizers.l2(0.001), return_sequences=True))
        self.model.add(KerasLSTM(32, activation='relu',
kernel_regularizer=regularizers.l2(0.001), return_sequences=True))
        self.model.add(KerasLSTM(16, activation='relu',
kernel_regularizer=regularizers.l2(0.001)))
        self.model.add(Dense(1))
        self.model.compile(optimizer=RMSprop(learning_rate=0.001), loss='mse',
metrics=['mae'])

    def get_model(self):
        return self.model
```

Figure 19 - The LSTM class from the ExRate Service project.

First, a sequential model is created (since the historical exchange rates used for training are a sequence). Then three LSTM layers are added, the first is given 64 units and its output is fed into the second LSTM layer of 32 units and then the third with 16 units, all using ReLU activation. The output of the third LSTM layer is then given to a dense layer, which derives the activation state of each neuron. Next, these are compiled with backpropagation using the learning rate of 0.001. The loss metric has been set to Mean Absolute Error (MAE), which is a typical selection for time-series problems. Finally, the model is then returned via the **`get_model()`** function.

After defining the input shape, an instance of the LSTM class can be created and trained using the **`fit()`** function from Keras. This method requires training data and labels, as well as the

number of epochs and batch size to use during training. Once the training is finished, the model can make predictions on new data by utilising the **`predict()`** function.

The FCNN model was implemented in a similar structure, which ensured the modularity of the ExRate Service, keeping in line with the idea that models can be created and tweaked independently of the rest of the application.

Figure 20 shows the implementation of the FCNN model within the ExRate Service. The model starts in the same manner as the LSTM model from *Figure 19*, with an initial sequence model being created, this is then followed by a flattening layer which is used to convert the input data to a one-dimensional array. The model has three hidden layers which follow, each consisting of dense, batch normalisation and dropout layers, the tooling for which comes from Keras. The dense layers have 128, 64 and 32 neurons, respectively, with the ReLU activation function used to introduce non-linearity. The batch normalisation layers are used to normalise the inputs of each layer, and the dropout layers are used to randomly drop some of the neurons during training to prevent overfitting. The final step to compile is identical to that of the LSTM model for accurate comparison between the two.

```
class FCNN:
    def __init__(self, input_shape):
        self.model = Sequential()
        self.model.add(Flatten(input_shape=input_shape))
        self.model.add(Dense(128, activation='relu',
kernel_regularizer=regularizers.l2(0.001)))
        self.model.add(BatchNormalization())
        self.model.add(Dropout(0.5))
        self.model.add(Dense(64, activation='relu',
kernel_regularizer=regularizers.l2(0.001)))
        self.model.add(BatchNormalization())
        self.model.add(Dropout(0.5))
        self.model.add(Dense(32, activation='relu',
kernel_regularizer=regularizers.l2(0.001)))
        self.model.add(BatchNormalization())
        self.model.add(Dense(1))
        self.model.compile(optimizer=RMSprop(learning_rate=0.001), loss='mse',
metrics=['mae'])

    def get_model(self):
        return self.model
```

Figure 20 - The FCNN class from the ExRate Service project.

Model Analysis Method

To evaluate each model's performance the MAE which is a loss metric will be used to measure the margin of error between predicted and actual values, which indicates model performance. A higher MAE value suggests that the model is less accurate and makes larger

errors in its predictions, potentially due to overfitting. Conversely, a lower MAE value indicates that the model is more accurate and makes smaller errors, which is generally preferred. However, it is important to consider the context of the problem being solved as a high MAE may be acceptable in some cases where small errors are not critical or where there is high variability in the data, while a low MAE may be necessary in cases where small errors can have a significant impact.

To calculate the margin of error as a percentage between the actual and forecast rates, the function from *Figure 21* will be used. It takes two arrays and returns the percentage difference between them, this will be used to evaluate the results against the hypothesis posed by the current paper.

```
def percentage_difference(self, arr1, arr2):
        total_diff = sum(abs(a - b) for a, b in zip(arr1, arr2))
        avg = (sum(arr1) + sum(arr2)) / (len(arr1) + len(arr2))
        return (total_diff / avg) * 100
```

Figure 21 - A function written to evaluate the margin of error between the actual and forecast rates.

Results

Model Performance and Evaluation

To provide analysis on the performance of two models in forecasting the exchange rates, converting from United States Dollar (USD) to Great British Pound (GBP) was used as an example. This training was conducted throughout April 2023.

Figure 22 shows the MAE for the LSTM model. The LSTM model exhibited an initial sharp decline for both the training and validation sets, eventually stabilising around the 10th epoch. Although early stopping was enabled, a maximum of 100 epochs were used, indicating that the overall loss continued to make gradual improvements throughout training. The LSTM model achieved a low MAE of approximately 0.025 on the validation set.

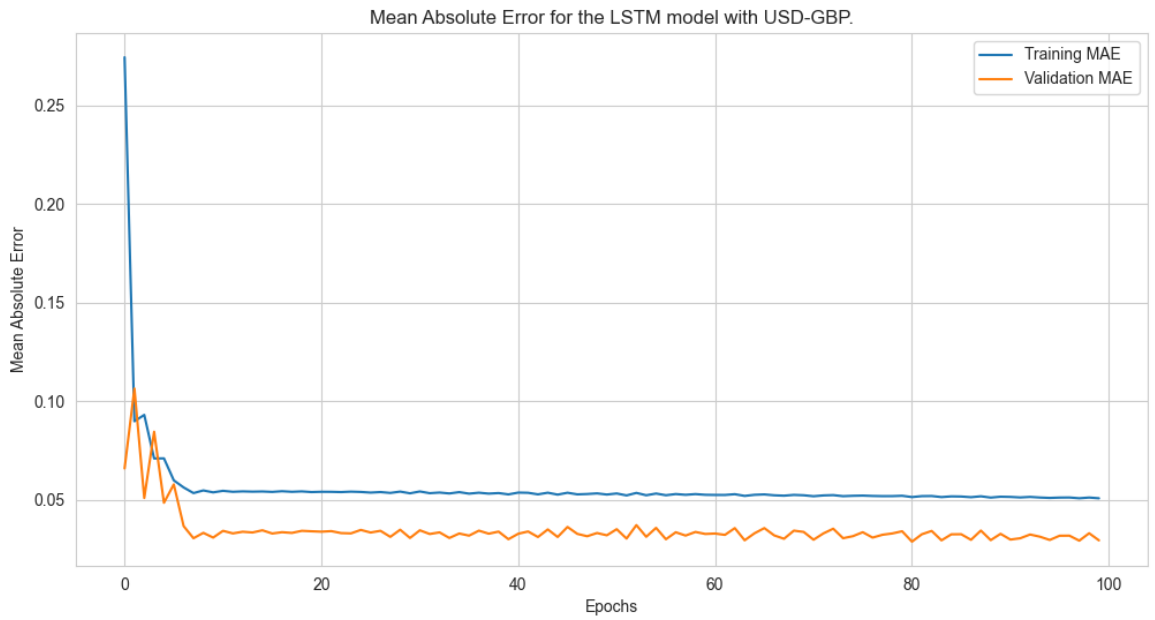


Figure 22 - A line graph showing the MAE loss metric over the course of the LSTM's training delineated by epoch iterations.

Focusing on the FCNN model, *Figure 23* shows a training loss with a downward trend, while the validation loss remained somewhat consistent. The latter implies that the model was overfitting its training data. This is evidenced by the early stopping feature terminating training prematurely at the 30th epoch, indicating that the model was not generalising well to unseen data. However, this shows that the FCNN model can be trained in less time, which may be an argument for its usage within an API context.

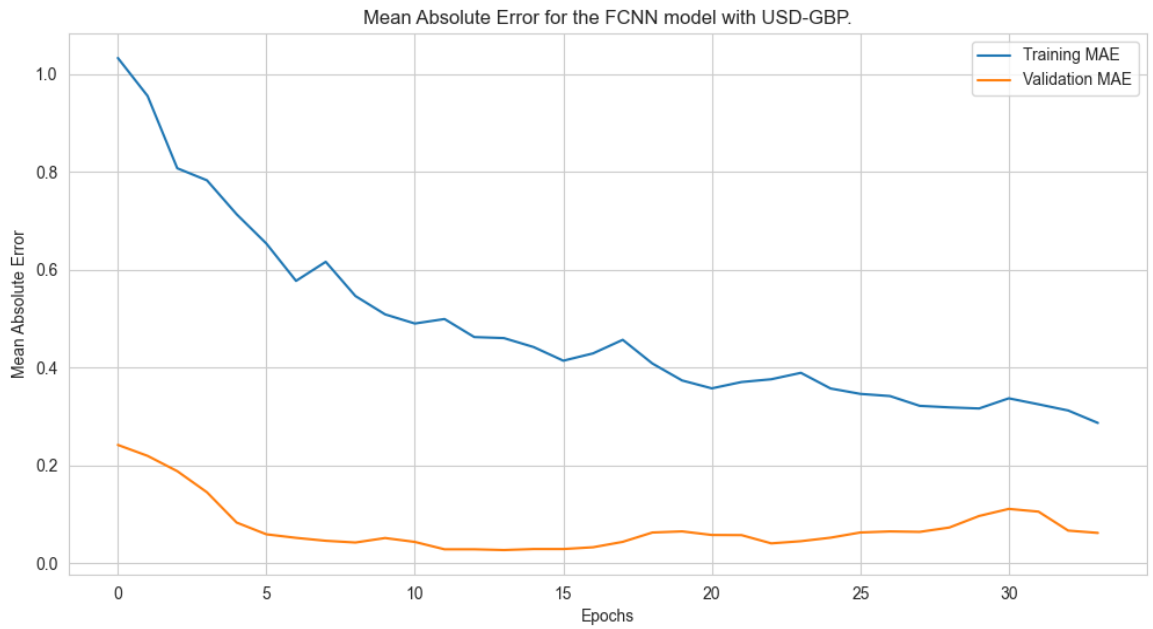


Figure 23 - A line graph showing the MAE loss metric over the course of the FCNN's training delineated by epoch iterations.

Model Forecasts

In *Figure 24*, it can be observed that the LSTM model has provided a forecast fairly close to the actual data for the predicted time horizon, whilst the FCNN model's forecast was considerably less accurate.

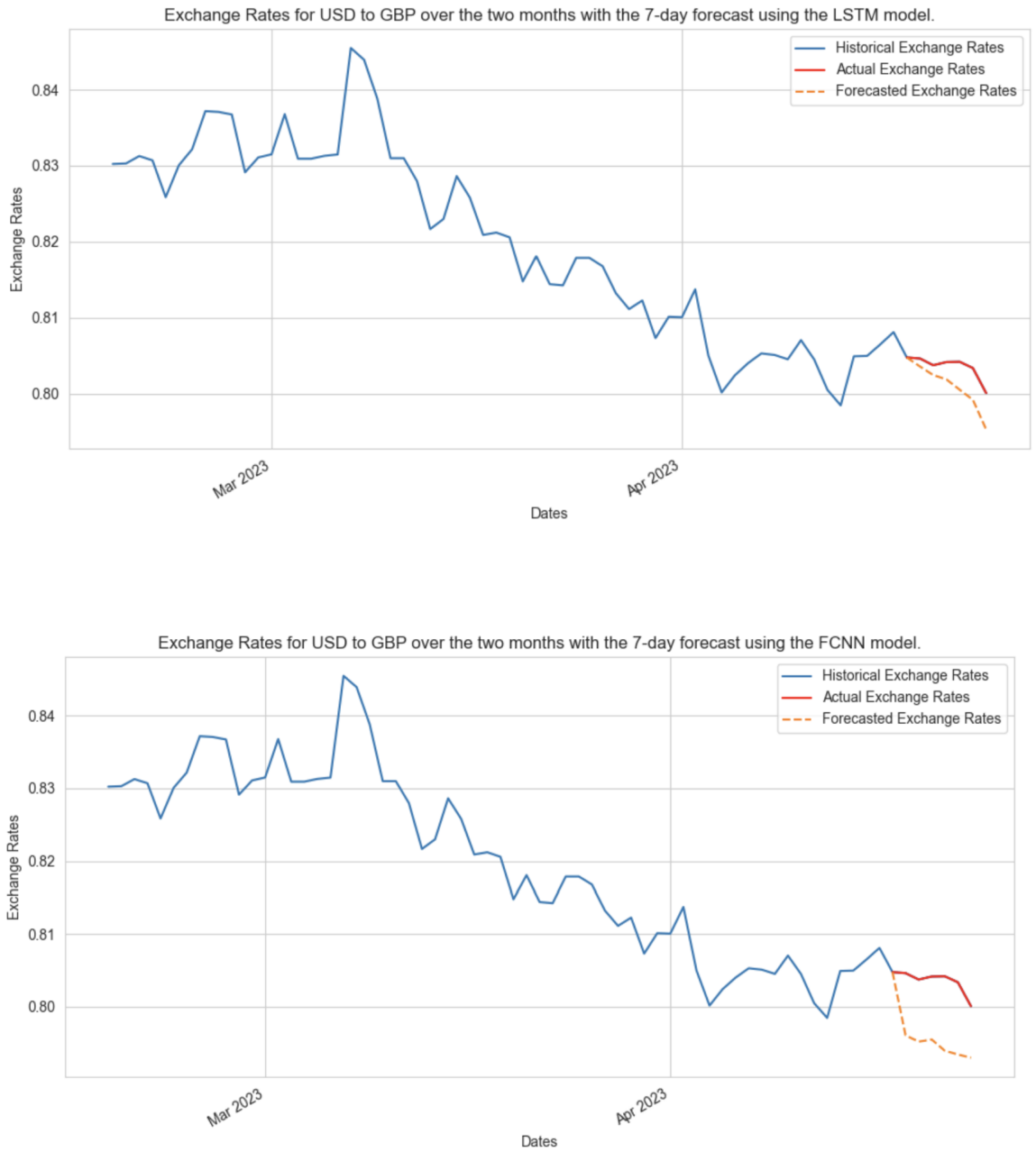


Figure 24 - A line graph showing the two model forecasts with two months of historical exchange rates for context.

Figure 25 shows a closer comparison between the actual rates and the predicted rates for each model for the given forecast horizon. The LSTM model somewhat follows the trends of the actual values, losing accuracy as the time horizon increases. The FCNN model bears very limited relation, if any, to the actual rates for this given horizon.

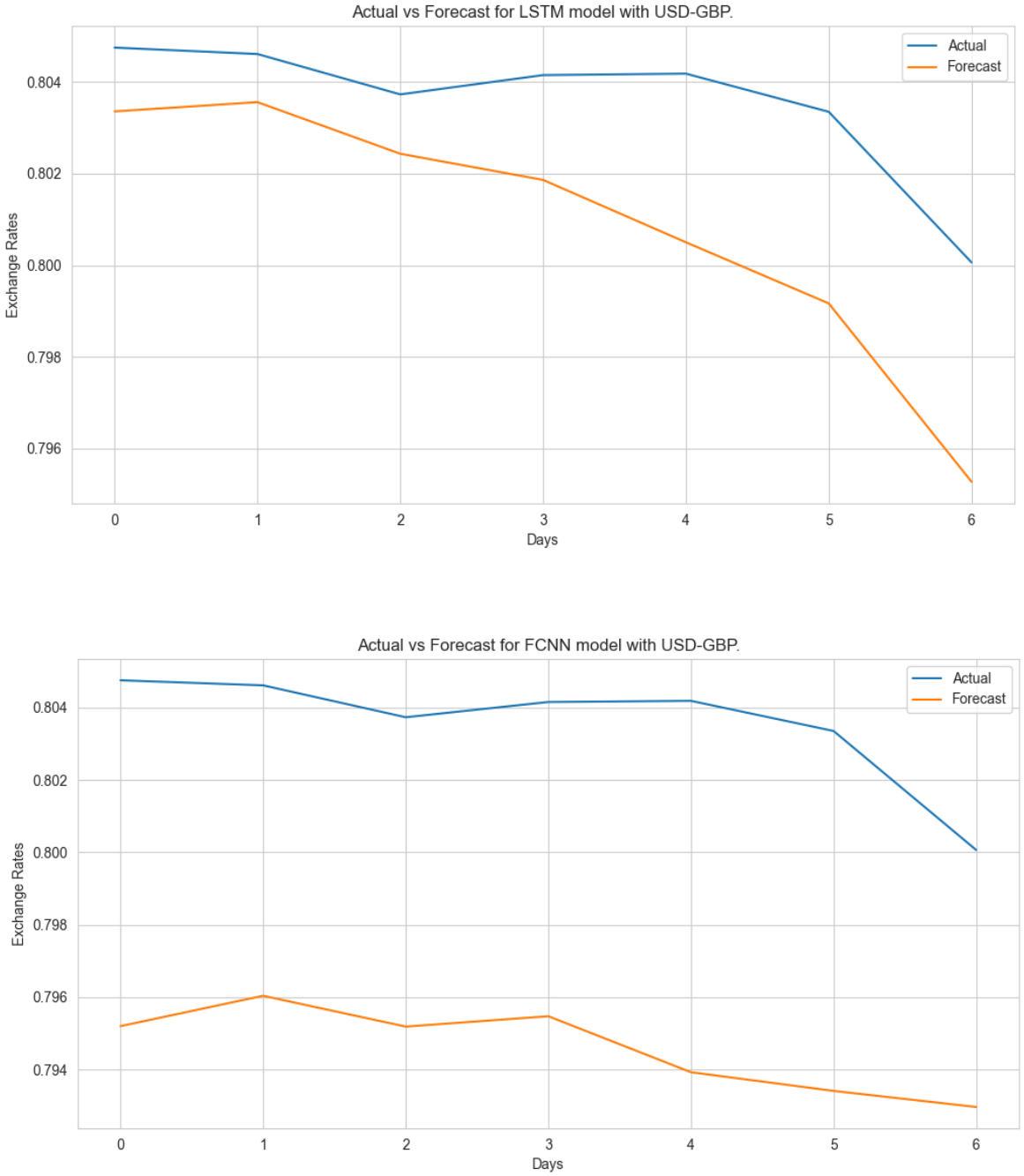


Figure 25 - Two line graphs showing the LSTM (top) and the FCNN (bottom) respective predictions compared to the actual rates observed over the same range of time.

To calculate the margin of error expressed as a percentage the `percentage_difference()` function from *Figure 21* is used. For the LSTM model, the input rates were:

```
Actual:      [0.80415, 0.80418, 0.80335, 0.80006, 0.80548, 0.80189, 0.80018]
Forecast:    [0.80723, 0.80791, 0.80809, 0.80836, 0.80865, 0.80894, 0.80947]
```

Which gave a percentage difference of 4.89%. The FCNN inputs were:

```
Actual:      [0.80415, 0.80418, 0.80335, 0.80006, 0.80548, 0.80189, 0.80018]
Forecast:    [0.81193, 0.81235, 0.81165, 0.81257, 0.81338, 0.81223, 0.81273]
```

This resulted in a percentage difference of 8.37%.

While these percentages reflect each model's best performance, it is important to consider the range of the margin of error when evaluating the performance of these ML models as their performance can differ significantly with each execution. For example, the range of the margin of error for the LSTM model would vary from 4% to 6% while the FCNN model would vary between 8% and 11%.

Discussion

To first compare the loss metrics of both the LSTM and FCNN models, the former demonstrated superior performance to the latter in terms of MAE, with a lower value achieved on the validation set. This indicates that the LSTM model is better suited for this forecasting task than the FCNN model. The MAE loss metric for the LSTM model suggests its ability to forecast the target variable accurately, while the loss metric for the FCNN model suggests that the model is fitting well to the training data but is not generalising well to new unseen data.

To further evaluate the efficacy of each model's predictive capability for exchange rates, a comparison between the two of their respective forecasts can be made with the predicted values against the actual rates for the relevant time period. Such a comparison is illustrated in *Figure 24*, where it is apparent that the LSTM model outperforms the FCNN model in terms of predictive accuracy, with the former exhibiting a 4.89% margin of error to the actual exchange rates. Conversely, the predictions of the FCNN model deviate considerably from the actual rates (8.37% margin of error), indicating its inferior performance. This disparity could be explained by two possible contributories; overfitting and FCNN's inability to capture temporal dependencies in time-series data due to its design. Unlike LSTM, FCNN is primarily prone to overfitting caused by suboptimal hyperparameters (i.e., learning rate and number of layers etc.), feature selection and small dataset (Sun, Brockhauser & Hegedűs, 2021).

This empirical investigation also reveals that a week forecast horizon produces more accurate predictions compared to longer forecast horizons such as two weeks or a month; the LSTM model's performance demonstrates forecasting accuracy decreased as the forecast horizon increased. This mirrors the work done by Ma and Mao (2020) who also found that predictions are less reliable the further from the end of the original time-series dataset forecasts are made. Thus, whilst it is feasible to make a week-long forecast, two weeks or more would likely hold no meaning. Consequently, it is necessary to make trade-offs between the forecast horizon and the accuracy of the predictions.

There were several challenges faced during development. One such challenge was the lack of documentation on setting up pipelines specific to the project requirements. This was in part due to the proprietary nature of Azure's customers where security considerations trump knowledge sharing within the community. Additionally, despite the advantages of automation, setting it up for a project can initially take time, especially if the developer is unfamiliar with the specific requirements for the chosen platform, be it Microsoft Azure, Amazon Web Services or Google Cloud - the three most popular cloud computing services used across the industry (Gupta, Mittal & Mufti, 2021) - which all vary in the way they require developers to setup automation for their projects.

Beyond the challenges presented by the continuous integration process, local development was at times less productive than desired due to changing requirements as the project matured

and new challenges were uncovered that required dedicated effort to overcome. One such example was when a new feature, which added a second ML model for training and predictions within the ExRate Service, became a requirement. This then triggered a refactor of not only the Service itself, to become modularity, separation of concerns and decoupled components, but which has knock-on effects on the API since the script arguments changed. This type of refactoring, while essential for the purpose of this project, was unforeseen at the start of development and came quite late in the development process which setback development across the project. However, this highlights the need for a development cycle like that of Agile (Almeida, 2021) where change is expected and embraced by the systems which support it.

The emergence of unforeseen challenges had an adverse impact on the project, resulting in a month-long delay (see *Appendix 5a*). The evolving nature of the requirements impeded the achievement of optimal velocity, rendering it infeasible to meet the March deadline. Nonetheless, it was fortuitous that the deadline was not a hard constraint, as the actual project deadline was at the end of April, at the very latest. It is worth noting that the completion of the current paper itself was also requisite and as such the initial March deadline was established at the commencement of development in September. The tracking of the burndown rate throughout the development phase facilitated the identification of areas that required improvement. However, non-functional tasks that involved the polishing and fine-tuning of technical components, such as the ML models, were typically left towards the end of the project and proved to be the most time-consuming.

As for the limitations of this project, the small dataset was identified retrospectively as having an effect on the robustness of the models and their ability to generalise to new data. With only as many data points as days in a year, there may not be enough examples to capture the full complexity of exchange rate movements. The univariate nature of the dataset can result in oversimplification and may not fully capture the dynamics of the market. Incorporating other variables could potentially provide a more complete picture of the market and improve the accuracy of the model. Furthermore, the exclusion of non-socioeconomic factors can also limit the usefulness of the model. Natural disasters or geopolitical events can have a significant impact on exchange rates, and not accounting for these factors can lead to inaccurate predictions, and the product may lack ecological validity. In real-world applications, it is crucial to consider all possible factors that can influence exchange rates to make informed decisions. Another limitation originated from the lack of integration and system tests developed for the ExRate software suite, while unit tests were developed for the ExRate API, the Service and Frontend lacked any. The lack of tests for the Service was in part due to its evolving nature and constant development, where tests would have added further complexity and overhead. Ultimately, some of these limitations could have been resolved had the time frame for the project been longer. A longer development window may have provided an opportunity to model some of the external factors mentioned and further refine and optimise the architecture of both models.

Future research should address the current limitations by increasing the size of the dataset or investigating alternative models which can better handle long-term predictions. One potential model alternative is the Autoregressive Integrated Moving Average (ARIMA) approach. ARIMA takes a different approach to time-series forecasting based on statistics (Rubi, et al, 2022), which may indeed perform better than our LSTM model based on recent findings (Pandey, et al, 2023). Another approach to improving predictive forecasting accuracy could be to ensemble a hybrid model which pulls on the strengths of LSTM and modularity provided by an FCNN model-type architecture. This may increase the overall effectiveness and accuracy of predictions through a combination of strengths. Further research should also focus upon pioneering new techniques which can incorporate ecological factors known to influence the volatility of exchange rates.

In conclusion, this paper has presented a comprehensive investigation into the application of LSTM and FCNN models, for forecasting exchange rates. The results demonstrated the superiority of the LSTM model, which could often produce results which fell below a 5% margin of error. However, the limitations of the study should be taken into account when interpreting the findings. Furthermore, this project underscores the importance of continuous integration and Agile methodologies in addressing evolving requirements and unforeseen challenges during the development process. Despite the setbacks faced, these methodologies facilitated adjustments and improvements throughout the project's development, allowing for the successful completion of the current paper and supporting software artefact. Overall, this paper contributes to the growing body of knowledge in the field of exchange rate forecasting using ML techniques, paving the way for more advanced and accurate models that can be utilised in a diverse range of fiscal applications. The importance of achieving long-term forecasting cannot be understated given the implications for economic, business and investment circles.

References

Airaj, M., 2017. Enable cloud DevOps approach for industry and higher education. Concurrency and Computation: Practice and Experience, 29(5), p.e3937.

Ajit, A., Acharya, K. and Samanta, A., 2020, February. A review of convolutional neural networks. In 2020 international conference on emerging trends in information technology and engineering (ic-ETITE) (pp. 1-5). IEEE.

Almeida, F.L., 2021. Management of non-technological projects by embracing agile methodologies. International Journal of Project Organisation and Management, 13(2), pp.135-149.

Althelaya, K.A., El-Alfy, E.S.M., and Mohammed, S. (2018) 'Evaluation of bidirectional LSTM for short-and long-term stock market prediction', in 2018 9th International Conference on Information and Communication Systems (ICICS), pp. 151-156. IEEE.

Awan, F.M., Minerva, R., and Crespi, N. (2020) 'Improving road traffic forecasting using air pollution and atmospheric data: Experiments based on LSTM recurrent neural networks', Sensors, 20(13), p.3749.

Babu, A.S., and Reddy, S.K. (2015) 'Exchange rate forecasting using ARIMA', Neural Network and Fuzzy Neuron, Journal of Stock & Forex Trading, 4(3), pp.01-05.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R. and Kern, J., 2001. The agile manifesto.

Borovykh, A., Oosterlee, C.W. and Bohté, S.M., 2019. Generalization in fully-connected neural networks for time series forecasting. Journal of Computational Science, 36, p.101020.

Choi, D., 2020. Full-Stack React, TypeScript, and Node: Build cloud-ready web applications using React 17 with Hooks and GraphQL. Packt Publishing Ltd.

Dima, A.M. and Maassen, M.A., 2018. From Waterfall to Agile software: Development models in the IT sector, 2006 to 2018. Impacts on company management. Journal of International Studies, 11(2), pp.315-326.

Donahue, J., Anne Hendricks, L., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K. and Darrell, T., 2015. Long-term recurrent convolutional networks for visual recognition and description. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2625-2634).

Dubey, A. and Wagle, D., 2007. Delivering software as a service. The McKinsey Quarterly, 6(2007), p.2007.

Gers, F.A., Schmidhuber, J. and Cummins, F., 2000. Learning to forget: Continual prediction with LSTM. Neural computation, 12(10), pp.2451-2471.

Goodman, S.H. (1979) 'Foreign exchange rate forecasting techniques: implications for business and policy', The Journal of Finance, 34(2), pp.415-427.

Graves, A., 2012. Long short-term memory. Supervised sequence labelling with recurrent neural networks, pp.37-45.

Gupta, B., Mittal, P. and Mufti, T., 2021, March. A review on Amazon web service (AWS), Microsoft azure & Google cloud platform (GCP) services. Proceedings of the 2nd International Conference on ICT for Digital, Smart, and Sustainable Development, ICIDSSD 2020, 27-28 February 2020, Jamia Hamdard, New Delhi, India.

Hopper, G.P., 1997. What determines the exchange rate: Economic factors or market sentiment. Business Review, 5, pp.17-29.

Karevan, Z., and Suykens, J.A. (2020) 'Transductive LSTM for time-series prediction: An application to weather forecasting', Neural Networks, 125, pp.1-9.

Kim, J., El-Khamy, M. and Lee, J., 2017. Residual LSTM: Design of a deep recurrent architecture for distant speech recognition. arXiv preprint arXiv:1701.03360.

Koch, S., 2004. Agile principles and open source software development: A theoretical and empirical discussion. In Extreme Programming and Agile Processes in Software Engineering: 5th International Conference, XP 2004, Garmisch-Partenkirchen, Germany, June 6-10, 2004. Proceedings 5 (pp. 85-93). Springer Berlin Heidelberg.

Korus, A. and Celebi, K., 2019. The impact of Brexit news on British pound exchange rates. International Economics and Economic Policy, 16, pp.161-192.

Liu, B., Wang, M., Foroosh, H., Tappen, M. and Pensky, M., 2015. Sparse convolutional neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 806-814).

Ma, M. and Mao, Z., 2020. Deep-convolution-based LSTM network for remaining useful life prediction. IEEE Transactions on Industrial Informatics, 17(3), pp.1658-1667.

Meese, R.A. and Rogoff, K., 1983. Empirical exchange rate models of the seventies: Do they fit out of sample?. Journal of international economics, 14(1-2), pp.3-24.

Mertens, J., 2022. Broken Games and the Perpetual Update Culture: Revising Failure With Ubisoft's Assassin's Creed Unity. Games and Culture, 17(1), pp.70-88.

Mohamed, M.M. and Schuller, B.W., 2022. Normalise for fairness: A simple normalisation technique for fairness in regression machine learning problems. arXiv preprint arXiv:2202.00993.

Pandey, A., Singh, G., Hadiyuono, H., Mourya, K. and Rasool, M.J., 2023, January. Using ARIMA and LSTM to Implement Stock Market Analysis. In 2023 International Conference on Artificial Intelligence and Smart Communication (AISC) (pp. 935-940). IEEE.

Peirano, R., Kristjanpoller, W., and Minutolo, M.C. (2021) 'Forecasting inflation in Latin American countries using a SARIMA–LSTM combination', Soft Computing, 25(16), pp.10851-10862.

Piccialli, F., Giampaolo, F., Prezioso, E., Camacho, D., and Acampora, G. (2021) 'Artificial intelligence and healthcare: Forecasting of medical bookings through multi-source time-series fusion', Information Fusion, 74, pp.1-16.

Polk, R., 2011, August. Agile and Kanban in coordination. In 2011 agile conference (pp. 263-268). IEEE.

Preethi, V., 2021. Survey on text transformation using bi-lstm in natural language processing with text data. Turkish Journal of Computer and Mathematics Education (TURCOMAT), 12(9), pp.2577-2585.

Rubi, M.A., Chowdhury, S., Rahman, A.A.A., Meero, A., Zayed, N.M. and Islam, K.A., 2022. Fitting Multi-Layer Feed Forward Neural Network and Autoregressive Integrated Moving Average for Dhaka Stock Exchange Price Predicting. Emerging Science Journal, 6(5), pp.1046-1061.

Sarcar, V., 2022. Fundamentals of. NET and C#. In Test Your Skills in C# Programming: Review and Analyze Important Features of C# (pp. 3-32). Berkeley, CA: Apress.

Sarker, I.H., 2021. Machine learning: Algorithms, real-world applications and research directions. SN computer science, 2(3), p.160.

Shen, Mei-Li, Cheng-Feng Lee, Hsiou-Hsiang Liu, Po-Yin Chang, and Cheng-Hong Yang. "An effective hybrid approach for forecasting currency exchange rates." Sustainability 13, no. 5 (2021): 2761.

Singh, Y., Bhatia, P.K., and Sangwan, O. (2007) 'A review of studies on machine learning techniques', International Journal of Computer Science and Security, 1(1), pp.70-84.

Sun, Y., Brockhauser, S. and Hegedűs, P., 2021. Comparing End-to-End Machine Learning Methods for Spectra Classification. Applied Sciences, 11(23), p.11520.

Topçuoğlu, B.D., Lapp, Z., Sovacool, K.L., Snitkin, E., Wiens, J. and Schloss, P.D., 2021. mikropml: user-friendly R package for supervised machine learning pipelines. Journal of open source software, 6(61).

Yasir, M., Durrani, M.Y., Afzal, S., Maqsood, M., Aadil, F., Mehmood, I. and Rho, S., 2019. An intelligent event-sentiment-based daily foreign exchange rate forecasting system. Applied Sciences, 9(15), p.2980.

Appendices

Appendix 1: Source Code

ExRate's Github page: `github.com/AlexBeesley/ExRate`
ExRate Frontend (hosted on Netlify): `exrate-frontend.netlify.app`

Appendix 1a - Project Artefacts

Appendix 2: Supporting Materials, Documentation and Resources

Exchange rate data collection from apilayer: `apilayer.com/marketplace/exchangerates_data-api`

Appendix 2a - Project Dependencies

Machine Learning:
- TensorFlow: `www.tensorflow.org/api_docs/python/tf`
- Keras: `keras.io`

Back-end development:
- Python: `docs.python.org/3`
- C#: `learn.microsoft.com/en-us/dotnet/csharp`
- ASP.NET: `dotnet.microsoft.com/en-us/apps/aspnet/apis`
- Docker: `docs.docker.com`

Front-end development:
- React: `legacy.reactjs.org/docs/getting-started.html`
- TypeScript: `www.typescriptlang.org/docs`

Integrated Development Environments:
- Visual Studio 2022: `learn.microsoft.com/en-us/visualstudio/windows`
- PyCharm: `www.jetbrains.com/pycharm/guide/tips/quick-docs`
- Visual Studio Code for React: `code.visualstudio.com/docs/nodejs/reactjs-tutorial`

Agile Development:
- Agile Methodology and Documentation: `www.nuclino.com/articles/agile-documentation`
- Microsoft Azure Pipelines: `learn.microsoft.com/en-us/azure/devops/pipelines`

Appendix 2b - Online Learning Resources

Clean Code: A Handbook of Agile Software Craftsmanship by Robert Martin (2013)

Agile Principles, Patterns, and Practices in C# by Robert Martin (2006)

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow 3e: Concepts, Tools, and Techniques to Build Intelligent Systems by Aurelien Geron (2022)

Exchange Rate Forecasting: Techniques and Applications by Imad Moosa (2000)

Machine Learning for Time Series Forecasting with Python by Francesca Lazzeri (2021)

C# Programming in easy steps, 3rd edition: Modern coding with C# 10 and .NET 6. Updated for Visual Studio 2022 by Mike McGrath (2022)

Full-Stack React, TypeScript, and Node: Build cloud-ready web applications using React 17 with Hooks and GraphQL by David Choi (2020)

Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development by Prem Ponuthorai (2022)

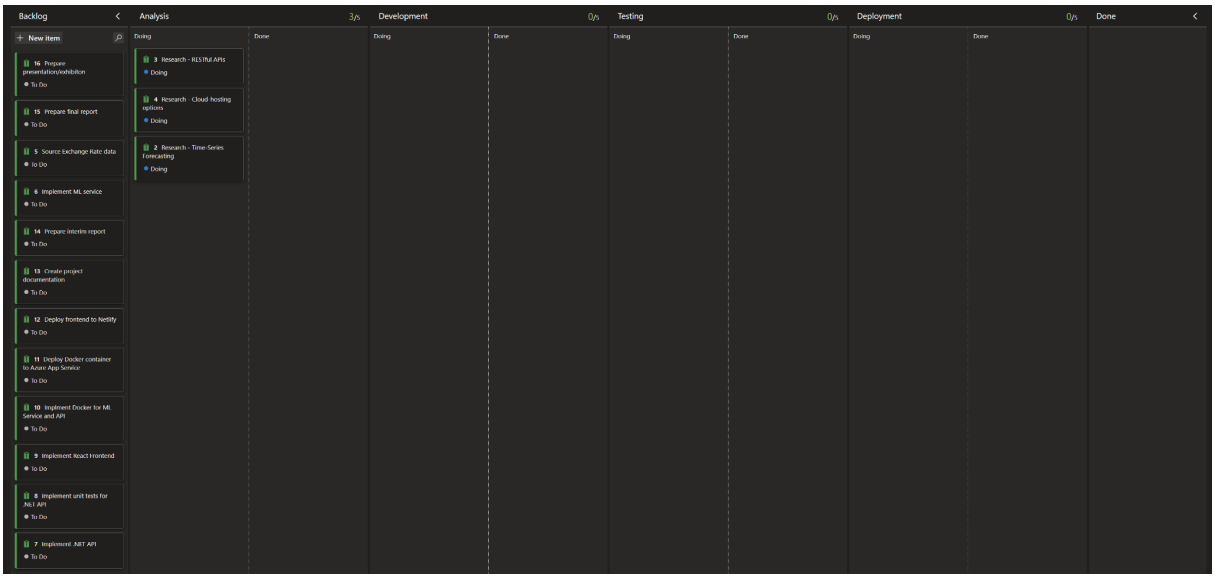
Hands-on Azure Pipelines: Understanding Continuous Integration and Deployment in Azure DevOps by Chaminda Chandrasekara (2020)

Docker Deep Dive by Nigel Poulton (2017)

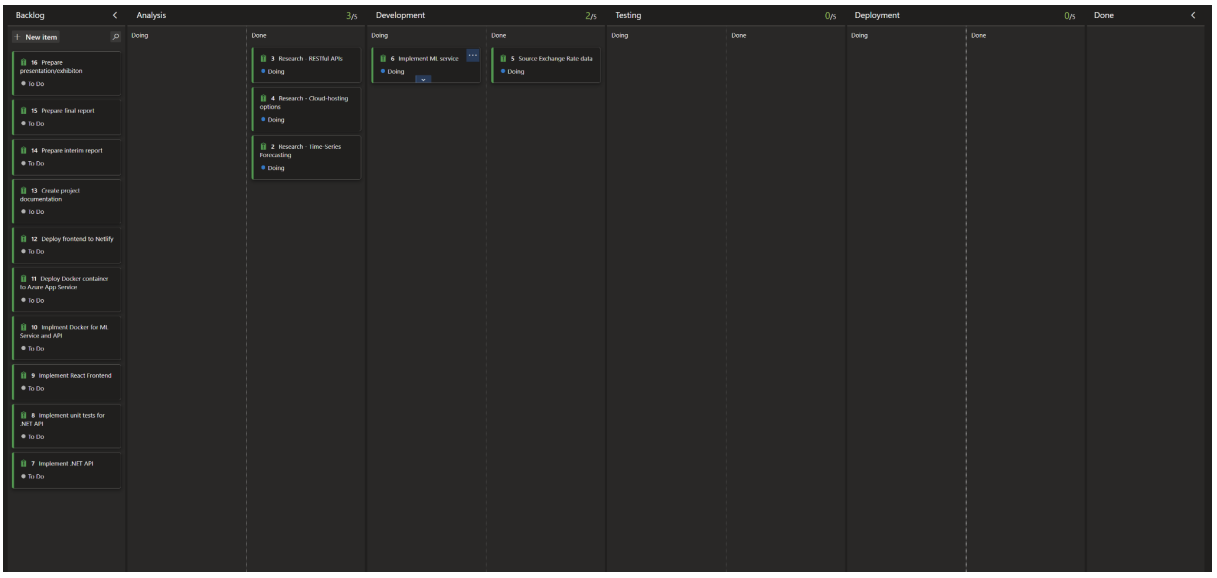
Appendix 2c - Literary Materials

Appendix 3: Kanban Board

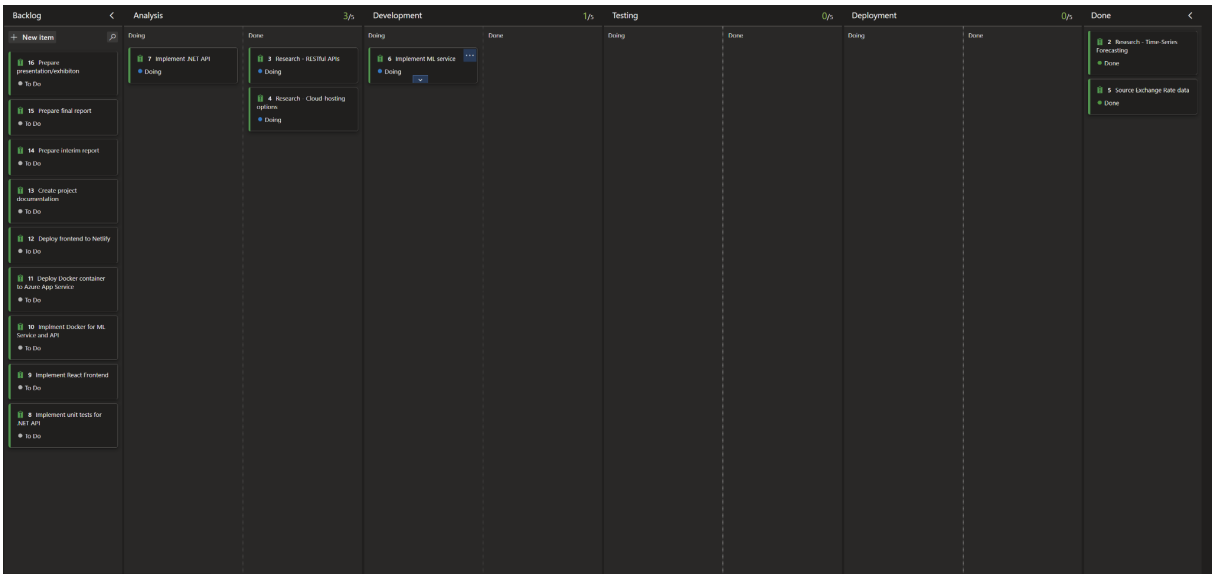
Screenshots of the Kanban Board from Azure DevOps of the projects progress from September 2022 to April 2023.



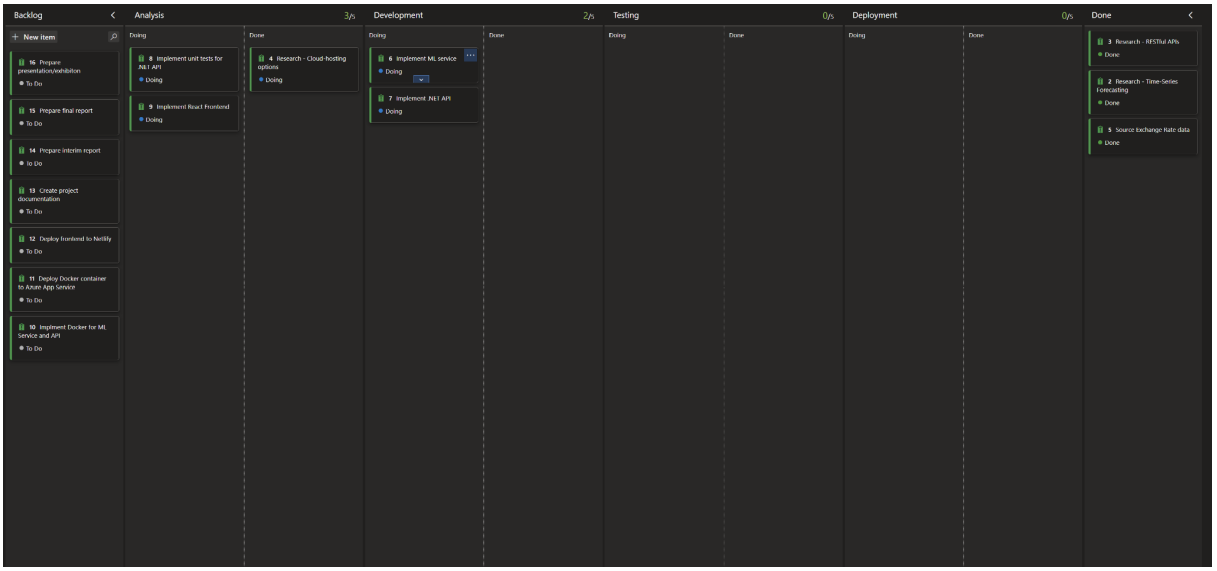
Appendix 3a - September



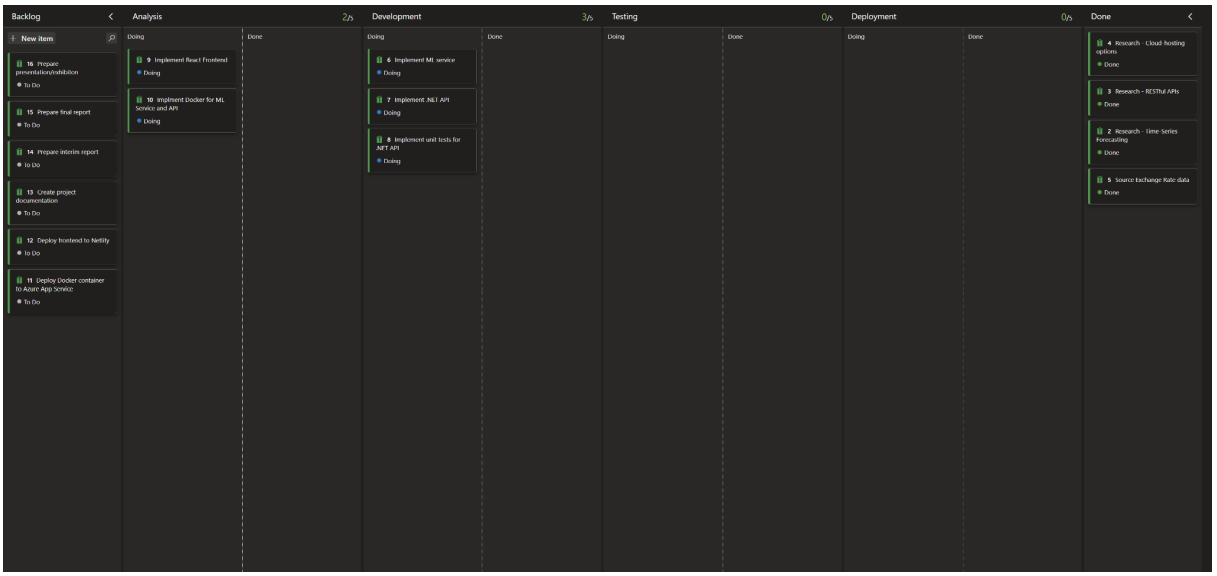
Appendix 3b - October



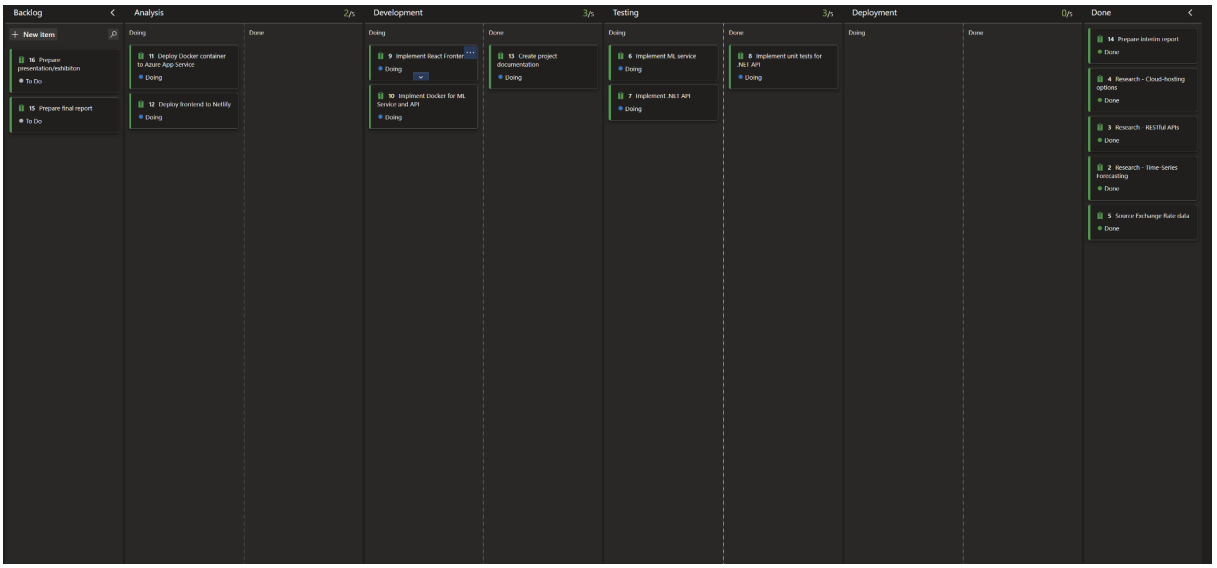
Appendix 3c - November



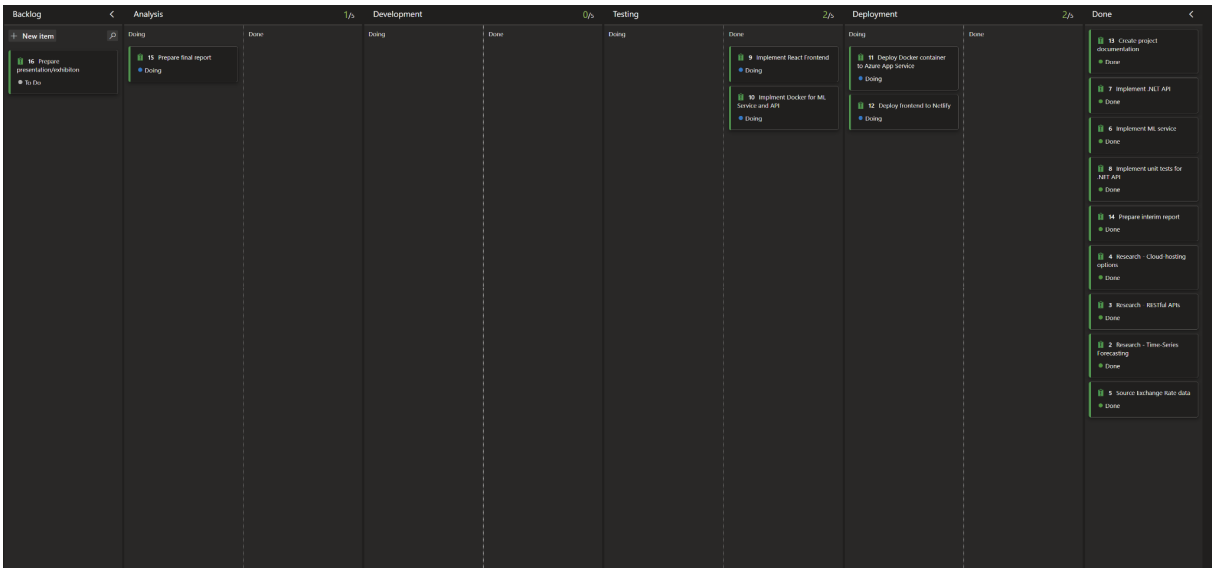
Appendix 3d - December



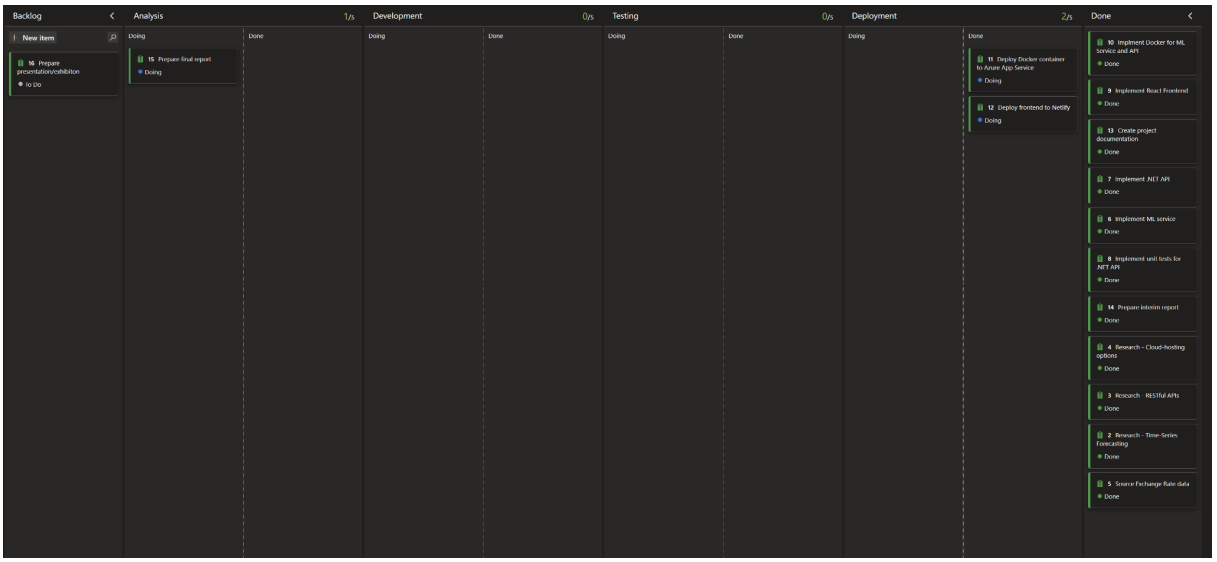
Appendix 3e - January



Appendix 3f - February



Appendix 3g - March



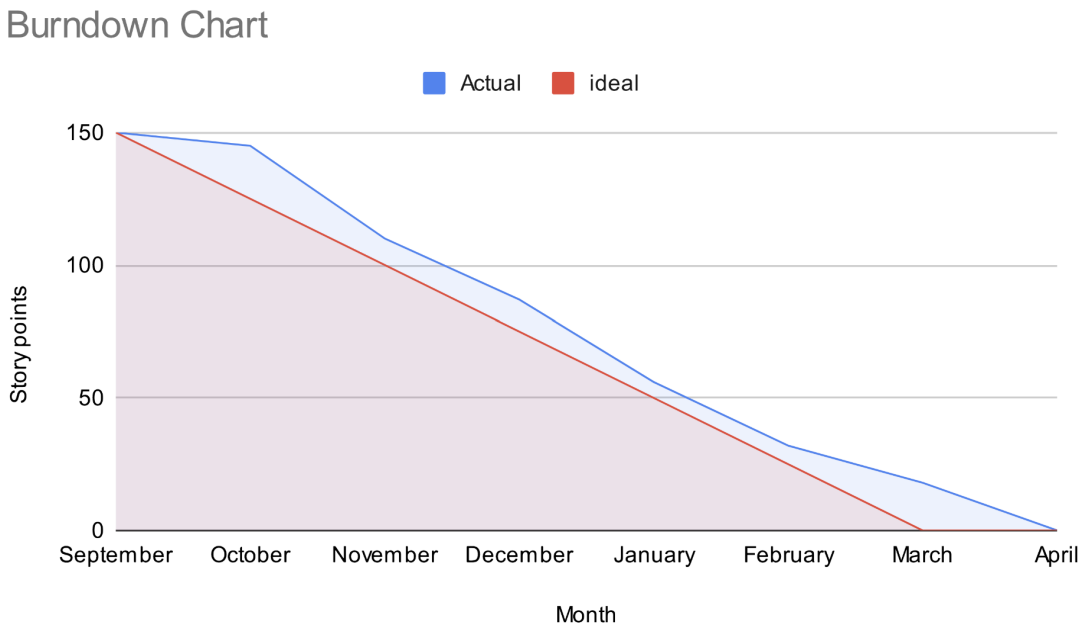
Appendix 3h - April

Appendix 4: Record of Meetings with Supervisor

10th October 2022	-	initial meeting and discussion
2nd December 2022	-	discussion for interim report
8th December 2022	-	discussion with first and second reader
14th February 2023	-	catchup and demonstration

Appendix 4a - Meetings with Nik Polatidis.

Appendix 5: Project Management



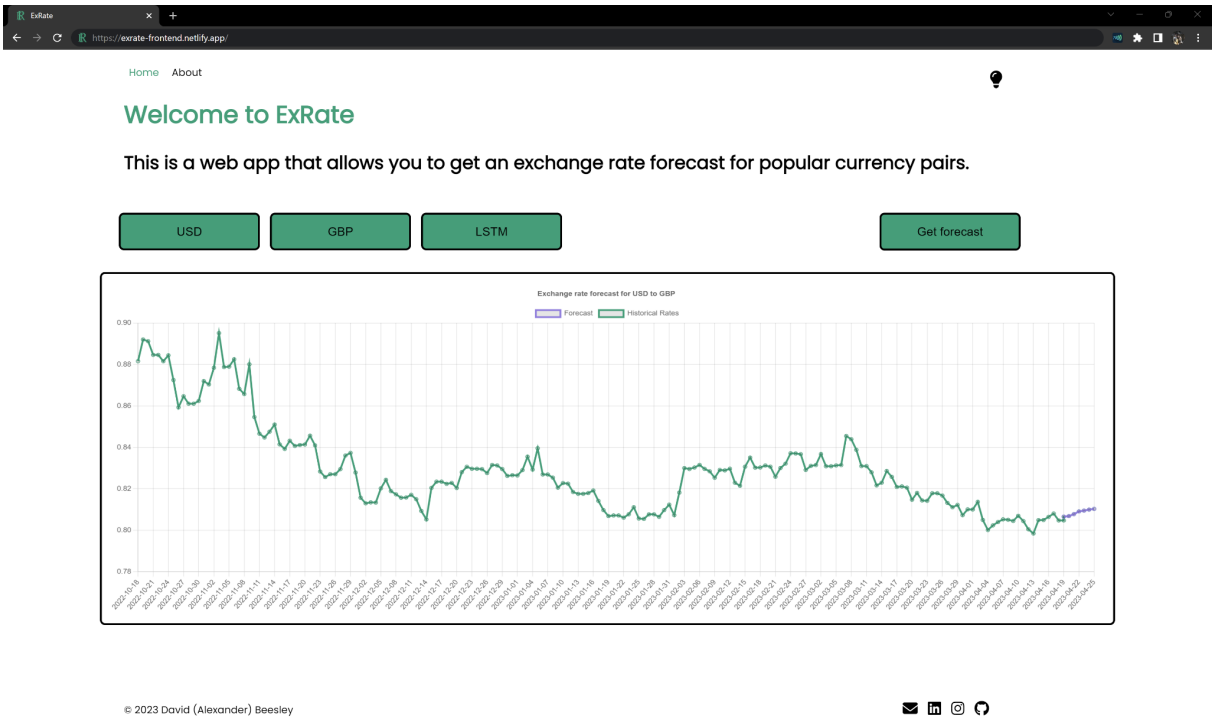
Appendix 5a - The Burndown Chart for this project at the end of April 2023.

Section 6: Commit Logs

```
2023-04-30 00:57:41 - Update README.md
2023-04-30 00:55:52 - Update README.md
2023-04-30 00:53:07 - .env check for service
2023-04-30 00:36:18 - removed files from repo which are now in
gitignore.
2023-04-30 00:18:24 - updated gitignore #2
2023-04-30 00:01:54 - gitignore update.
2023-04-27 16:09:12 - minor fixes, adds precentage difference to
service
2023-04-26 12:26:28 - new icon/logo
2023-04-25 10:54:10 - updated forecast graph
2023-04-24 15:58:07 - added lookback to service
2023-04-24 14:40:10 - Merge branch 'main' of
https://dev.azure.com/davidbeesley1234/ExRate/_git/ExRate
2023-04-24 14:39:57 - API ERROR PASSING and frontend fixes
2023-04-17 12:46:28 - Update azure-pipelines.yml for Azure
Pipelines put back
2023-04-17 12:25:12 - Updated azure-pipelines.yml for CD
2023-04-16 15:21:53 - Dockerfile update for seaborn
2023-04-16 15:15:35 - non azure setup
2023-04-16 14:53:16 - fix for azure.
2023-04-16 14:42:21 - added hook
2023-04-16 14:41:42 - frontend fix #2
2023-04-16 14:37:37 - fix for forntend
2023-04-16 14:26:12 - hooks fix
2023-04-16 14:23:13 - commit again
2023-04-16 14:22:42 - commit delete for github name bug
2023-04-16 14:17:17 - frontend fix #8
2023-04-16 14:00:12 - yarn rebuild
2023-04-16 12:07:24 - frontend fix and readme update
2023-04-15 12:46:28 - Service update
2023-04-14 12:35:35 - About page and README.md updates.
2023-04-14 00:15:15 - README.md update #2
2023-04-14 00:06:03 - README.md update
2023-04-13 22:47:21 - using .env for key for reliablity.
2023-04-13 22:41:33 - Frontend fixes and improvements.
2023-04-11 16:52:19 - home.tsx refactor and update to API changes
(not working yet)
2023-04-11 14:46:54 - updated tests
2023-04-11 14:43:24 - token arch update
2023-04-11 12:02:18 - Refactor and added more unit tests
2023-04-11 10:02:53 - using python naming conventions, deleted
code dump script
2023-04-10 22:01:41 - working api #2
2023-04-10 22:00:35 - Merge branch 'main' of
https://github.com/AlexBeesley/ExRate
2023-04-10 22:00:17 - working api #2
2023-04-10 22:00:00 - Working API
2023-04-10 14:21:45 - Update README.md
2023-04-10 14:18:35 - Service refactor
2023-04-08 15:42:23 - Muilt-model overhaul
2023-04-04 21:34:38 - About page added
2023-04-03 16:09:17 - fix for api
2023-04-03 14:57:54 - fix for dropdowns on netlify
2023-04-02 14:54:03 - README update
2023-04-02 14:46:41 - README updated. + some small fixes to
frontend footer
2023-04-01 12:24:31 - frontend styles work
2023-03-21 01:09:21 - test update for azure
2023-03-21 00:54:35 - Merge branch 'main' of
https://dev.azure.com/davidbeesley1234/ExRate/_git/ExRate
2023-03-21 00:54:27 - test fix for azure
2023-03-20 16:37:37 - Update azure-pipelines.yml for Azure
Pipelines
2023-03-20 16:13:18 - Updated azure-pipelines.yml
2023-03-20 16:05:15 - Merge branch 'main' of
https://dev.azure.com/davidbeesley1234/ExRate/_git/ExRate
2023-03-20 16:05:05 - removed unused test
2023-03-20 13:34:05 - Updated azure-pipelines.yml
2023-03-20 13:08:31 - fixed failing tests
2023-03-19 18:37:43 - codemaid cleanup
2023-03-19 18:33:52 - refactored API #2
2023-03-19 16:20:46 - Merge branch 'main' of
https://github.com/AlexBeesley/ExRate
2023-03-19 16:20:27 - refactored API
2023-03-19 15:26:50 - Update README.md
2023-03-14 14:45:14 - small fixes and rensorflow warning
supression.
2023-03-14 13:38:40 - az auth for key vault fix
2023-03-14 13:23:16 - azure auth setup
2023-03-14 12:12:05 - az auth fix
2023-03-14 11:35:12 - azure auth for keyvault
2023-03-14 11:18:43 - reverted log changes for the moment
2023-03-14 11:10:54 - log fix for azure
2023-03-11 23:56:10 - styles on frontend update
2023-03-11 13:11:05 - updated dockerfile
2023-03-11 12:46:02 - working locally using azure key
2023-03-10 23:24:46 - removed key from gh
2023-03-10 23:12:07 - Pretty cool right!
2023-03-09 00:06:13 - key fix #2
2023-03-08 23:51:57 - key inside!!!
2023-03-08 23:38:29 - Merge branch 'main' of
https://dev.azure.com/davidbeesley1234/ExRate/_git/ExRate
2023-03-08 23:38:20 - add .env for debug
2023-03-06 18:14:31 - Updated azure-pipelines.yml
2023-03-06 18:02:05 - Set up CI with Azure Pipelines
2023-03-06 17:56:13 - move docker ignore
2023-03-06 17:18:49 - Working Docker!!! yay
2023-03-05 14:09:14 - spelling error
2023-03-05 13:57:40 - docker #2
2023-03-05 01:51:13 - weird docker bugs.... getting there tho
2023-03-04 17:29:22 - front end work, docker work etc..
2023-03-02 09:58:03 - Frontend work #2
2023-02-27 13:18:21 - frontend dev #1
2023-02-25 22:07:35 - frontend work #1
2023-02-25 21:31:02 - updated tests #2
2023-02-25 21:25:52 - updated tests
2023-02-25 21:20:58 - added api tests. tests pass.
2023-02-21 18:01:21 - working chart
2023-02-20 02:11:59 - Working frontend api call
2023-02-19 15:38:29 - semi working frontend ;)
2023-02-19 14:44:07 - API is perfect
2023-02-18 11:44:33 - API imporvements
2023-02-18 11:22:15 - Working API.... ish
2023-02-14 09:54:19 - .vs folder deleted
2023-02-14 09:52:44 - project clean up #2
2023-02-14 09:51:03 - project clean up
2023-02-14 09:46:43 - added README.md
2023-02-14 09:34:36 - deleted T3 app in favour of React
2023-02-14 09:32:41 - clean up python code and added react
frontend template
2023-01-29 13:55:21 - model progress
2022-11-08 16:36:35 - model progress
2022-11-08 12:53:29 - program.py tweaks
2022-11-08 12:41:36 - added csv file with all available currency
abvs for UI idiot proofing.
2022-11-01 16:32:08 - ML work
2022-10-25 16:21:28 - Added forecasting example from keras.io to
reverse engineer
2022-10-25 15:56:06 - refactor #5
2022-10-25 15:54:15 - refactor #5
2022-10-25 15:52:19 - refactor #4
2022-10-25 12:18:46 - Merge pull request #1 from
AlexBeesley/refactor
2022-10-25 12:15:42 - refactor #3
2022-10-25 12:11:09 - Delete ExRate_Service/__pycache__ directory
2022-10-25 12:10:37 - refactor #2
2022-10-25 12:09:40 - refactor #1
2022-10-17 20:59:08 - Added .env for API Key
2022-10-10 09:42:07 - Clean code
2022-10-08 13:58:04 - added some further neural network work
2022-10-04 15:44:10 - added some initial neural network work
2022-10-04 12:57:15 - Added graph generation and UI
2022-10-04 11:17:17 - further progress
2022-10-04 10:57:32 - further progress
2022-10-04 10:23:19 - initial commit
```

Appendix 6a - Git commit messages to demonstrate progress.

Section 7: Frontend Demonstration



Appendix 7a - The ExRate Frontend home page shows forecast data for USD and GBP using the LSTM model.

~ END OF DOCUMENT ~