DEEP LEARNING PROJECT

AI Trading Bot

Due date:

13th March 2024

Submitted by:

André Galczynski, Aykan Güney, Djauschan Fedaie, Eira Hammerich, Jasmina Jäkle, Joel Wälchli, Johann Beaumont Grigjanis, Kevin Stelke, Luca Nickel, Niklas Kormann, Philipp Duwe, Thomas Parchmann, Umut Kurucay

Supervisor:

Prof. Dr. Hendrik Annuth and
Marco Pawlowski

Fachhochschule Wedel

Feldstraße 143

22880 Wedel

Contents

List of Figures

List of Tables

1

Introduction

1.1. Background

The financial market, especially the stock market, is a dynamic environment that is influenced by a variety of factors. These factors range from macroeconomic indicators to specific company news. Understanding and predicting the behavior of this market has always been the goal of professionals, traders, and investors. With the advancement of digitalization and the availability of large data sets from market activities, it has become possible to develop computer-based models and algorithms capable of analyzing market trends and making trading decisions in real time.
Despite advancements in algorithmic trading technology, there's still room for innovation, especially with the incorporation of modern artificial intelligence (AI) and machine learning techniques. By leveraging reinforcement learning, transformer models, and convolutional neural networks, it's believed that algorithmic trading could ascend to a new level through more accurate predictions and the development of more efficient trading strategies. (Joel Wälchli)

1.2. Project Goal and Vision

The primary objective of this project is the development of a bot capable of autonomously trading in the stock market and generating profits. Special emphasis is placed on time series analysis and AI-supported strategy development, as well as on implementing the bot within a microservice architecture and developing a user-friendly front-end.

However, our vision for the bot extends beyond the current project. We aim to create software that can actually connect to real stock exchanges and be tested in a live environment. Although realizing this vision within the current project timeline is not anticipated, it serves as a guideline and inspiration throughout all development phases.

The ultimate goal is to have a bot that operates independently of human intervention, makes decisions based on data rather than emotions, and achieves consistent profits, thereby paving the way for the next generation of algorithmic trading. (Joel Wälchli)

2

Foundations

2.1. How the Capital Market Works

Stocks represent shares in the equity of a company and are therefore a right of ownership. When investors buy stocks of a company, they acquire a part of the company and become shareholders. Stocks are traded on stock markets, where offer and demand determine the price of a stock. This price is a key indicator of the value of a company.

The stock price is created by the interaction of offer and demand. If the demand for a stock increases due to positive company news, good earnings prospects or other factors, the share price usually rises as well. Conversely, an oversupply of shares, i.e. when there are more sellers than buyers, usually leads to a fall in the stock price. The price of a stock is therefore constantly redefined by the transactions of market participants.

An important aspect of sh stock are trading is the “open“ and “close“ prices. The “open“ price is the price at which the first transaction of a share is executed at the beginning of the trading day. The “close“ price, on the other hand, is the price of the last transaction at the end of the trading day. These prices are important indicators of market movements and can be used by investors to identify trends and make trading decisions.

Another important term in the context of share trading is the “spread“. The spread is the difference between the purchase price (bid) and the selling price (ask) of a share. The bid price is the highest price a buyer is willing to pay for a stock, while the ask price is the lowest price at which a seller is willing to offer his stock. The spread is a measure of the liquidity of a stock; the lower the spread, the higher the liquidity as a rule. A low spread

means that the market for a stock is very active and transactions can be carried out close to the market price. A high spread, on the other hand, indicates lower liquidity, which means that buyers and sellers may have to accept greater price differences to make a transaction possible. (Eira Hammerich)

2.2. State of the Art

Modern capital markets are heavily dependend on advanced technology and financial theories to work efficiently. Algorithmic trading, where computers use algorithms to perform a high volume of trades very quickly, has increased transaction speed and volume, made markets more fluid and also lowered the cost of trading.

In addition to that, blockchain technology has also become significant in modern capital markets. It delivers a decentralised method of recording transactions, eliminating the need for traditional intermediaries, reducing costs and ensuring the accuracy and security of transaction records. Furthermore, robo-advisors have emerged as a notable development. Robo-advisers are digital platforms that offer investment management services by using algorithms. What they do is to adjust investments based on the user's risk tolerance and preferences, making personalised financial advice more accessible to a wider audience.

Another aspect is that the availability of financial information has improved. Investors can now easily access real-time data and analysis tools, which enables them to make better informed decisions. Online platforms also allow investors to share investment strategies and ideas.

Summing up it can be said that current capital markets use technology to enhance transaction efficiency, security, and access to investment information and advice. The investment process is now more streamlined, which results in a wider range of participants to take advantage of opportunities. (Umut Kurucay)

3

Data Analysis

3.1. Stocks and Indices Selection

In this section, we explain why we have selected which stocks and indices for this project. First, we discuss the stocks whose price performance we want to forecast. We then explain why we believe that the selected indices are useful inputs for the stock price prediction.

3.1.1. Stocks

Short explanation of why these companies were chosen ggf. Philip T. anfragen
Annotation: The cut-off date for market capitalization is Feb 19th 2024.

American Airlines Group Inc. (AAL):
American Airlines, a leading airline, operates an extensive international and domestic network with flights to numerous destinations worldwide. The company generates revenue through the sale of airline tickets, cargo services and other transportation services. With a market capitalization of USD 9.579 billion, American Airlines represents a significant portion of the airline industry.

Apple Inc. (AAPL):
Apple is a leading global technology company known for developing and selling consumer electronics, computer software and online services. Its best-known products include the iPhone, iPad, Mac computer systems, Apple Watch, Apple TV and the iOS, macOS, watchOS and tvOS software platforms. Apple generates revenue through the sale of these hardware

products, software licenses and services such as the App Store, Apple Music, iCloud and Apple Pay. With a market capitalization of USD 2.815 trillion, Apple is one of the most valuable companies in the world and plays a dominant role in the technology and consumer electronics sector.

Advanced Micro Devices Inc. (AMD):
AMD is a leading semiconductor company that develops and sells processors and graphics cards for computers, servers and embedded systems. The company generates most of its revenue by selling these microchips and technologies, which are used in a wide range of electronic devices worldwide. With a market capitalization of approximately USD 280.9 billion, AMD is a major player in the technology industry, known for its innovations in the high-performance computing and gaming markets.

Citigroup Inc. (C):
Citigroup is a global financial services company operating in over 100 countries. It provides investment banking, trading and credit card services in the United States. Citigroup is organized into two main segments: the Institutional Clients Group and the Personal Banking and Wealth Management Group. With a market capitalization of USD 104.385 billion, Citigroup is one of the largest banks in the world.

Moderna Inc. (MRNA):
Moderna is a biotechnology company specializing in mRNA therapeutics and vaccines, including the COVID-19 vaccine. The company derives revenue from the development and sale of these medical innovations. With a market capitalization of USD 33.694 billion, Moderna is a major player in the biotech industry.

NIO Inc. (NIO):
NIO Inc. is a Chinese company specializing in the development, manufacture and sale of electric vehicles. It is known for its innovations in battery swapping technology and autonomous driving. With a market capitalization of USD 12.966 billion, NIO positions itself as a leading player in the electric vehicle market.

NVIDIA Corporation (NVDA):
NVIDIA is a leading company in the development of graphics processing units (GPUs) and AI technologies. It generates revenue through the sale of GPUs for gaming and professional markets as well as artificial intelligence and data center offerings. With a market capitalization of USD 1.794 trillion, NVIDIA is a key player in the high-tech semiconductor sector.

Snap Inc. (SNAP):
Snap, known for Snapchat, operates a social media platform that is primarily focused on image and video communication. The company generates revenue through advertising, including sponsored content and filters. With a market capitalization of USD 18.438 billion, Snap Inc. is a major player in the social media sector.

Block Inc., formerly Square (SQ):
Block is a financial technology company that provides payment services for merchants and individuals, including the popular Cash App, a platform that enables peer-to-peer payments and allows users to trade stocks and cryptocurrencies. It earns money through transaction fees and financial services. Block's market capitalization is USD 40.309 billion.

Tesla Inc. (TSLA):
Tesla is a manufacturer of electric vehicles and renewable energy technologies. The company generates revenue through the sale of electric cars, solar panels and energy storage solutions. Tesla's market capitalization is USD 636.799 billion.
(Eira Hammerich)

3.1.2. Indices

Through an in-depth economic analysis, we have selected the indices that we believe will add the most value to the predictive quality of our models. In order to optimize the computational requirements, prioritize relevant data and avoid noise, the following indices were chosen:

NASDAQ Composite Index (COMP)
Represents a price index for over 3,000 tech stocks. It serves as a general sentiment indicator for tech-oriented stocks.

Dow Jones Commodity Indices (DJCI), each representing specific industrie or market segment

- **DJ Com. Index Gold (DJCIGC)**
  Gold represents a safe investment in times of crisis and acts as an alternative investment option to stocks. A rise in the price of gold may indicate increasing risk aversion among investors.

- **DJ Com. Index Silver (DJCISI)**
  Silver is an important raw material for industry, so the price rises with industrial growth. In addition, silver represents a safe investment opportunity, similar to gold.

- **DJ Com. Index Energy (DJCIEN)**
  This index tracks the performance of commodities in the energy sector (oil, natural gas, gasoline, heating oil). Fluctuations in energy prices can have a significant impact on the profitability and thus the share prices of companies, especially in energy-intensive sectors.

- **DJ Com. Index Nickel (DJCIIK)**
  Nickel is an important raw material for the production of stainless steel and batteries, e.g. for electric vehicles, and can therefore have a significant impact on the production costs and profitability of companies in these sectors.

- **DJ Industrial Average (DJI)**
  Dow Jones Industrial Average consists of 30 large, publicly traded companies in the US that are active in various industries. It is often seen as a barometer for the general direction of the US stock market.

- **DJ Internet Index (DJINET)**
  The Internet Index measures the performance of the 40 largest US companies in the Internet industry. It provides insights into performance and trends within the fast-growing and innovation-driven technology market, which has a significant impact on the global economy and thus on the financial markets.

U.S. Dollar Index (DXY)
The index measures the performance of the US dollar compared to the currencies of the USA's most important trading partners. The influence of the US dollar exchange rate on equities is complex (e.g. many commodities are traded in US dollars and it also has an impact on imports and exports). Unlike our other indices, the DXY is a foreign exchange index and not an equity index.

S&P 500 (SPX)
It represents the price index of the largest 500 companies on the US stock exchange. The S&P 500 provides a general "sentiment picture" of the stock market.

Overall, these indices provide a lot of valuable information. We found that the predictive quality of our models improved when the above-mentioned indices were included. As an outlook, there is further potential for improvement in the indices included. So far, there has been a strong concentration on the US market. In addition, further valuable information such as company data (sales, profits), inflation, interest rates and building applications could be included via APIs. Due to the limited time available for this project, implementation was not possible, but this could be part of an extension or continuation of this project. It is also important to note that the indices and shares are subject to various influences and multicollinearity. We use the indices as independent variables to better predict stock prices. Nevertheless, it should be noted that the indices are also influenced by the stocks. For example, Apple accounts for 9.21% of the NASDAQ index and therefore has a significant influence on it. The same applies to the S&P 500, DJ Industrial Average and DJ Internet Index. Furthermore, stocks prices also influence each other. They are therefore both dependent and independent variables. For this reason, we have carried out a detailed exploratory analysis of our data. (Eira Hammerich)

3.2. Exploratory Data Analysis

3.2.1. Correlation Analysis

Correlations between indices and stocks as well as correlations among stocks themselves are desirable for our model in order to improve the prediction quality. However, we would like to avoid correlations within the indices (independent variables) in order to avoid multicollinearity in our model. As shown in Figure 3.1, we have almost no correlation among the stocks. Nvidia is slightly correlated with Apple and Advanced Micro Devices. This is in line with our expectations, as Nvidia and AMD operate in very related fields and Apple also operates in a very similar business field in the computer industry.
By contrast, we unfortunately have strong correlations among the indices as shown in Figure 3.2. COMP, DJINET, DJI and SPX in particular show a strong correlation.

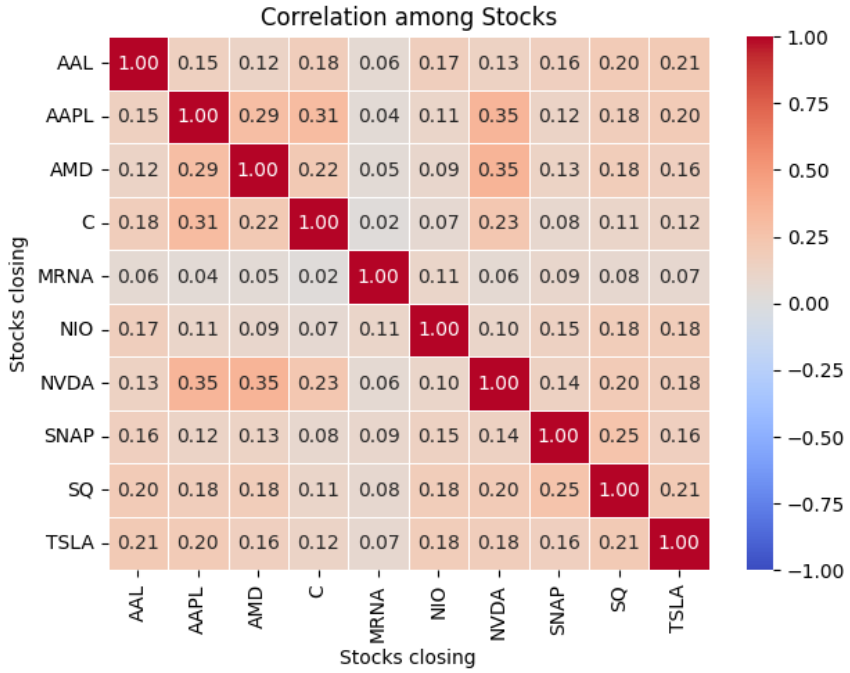


Figure 3.1.: Correlation among Stocks

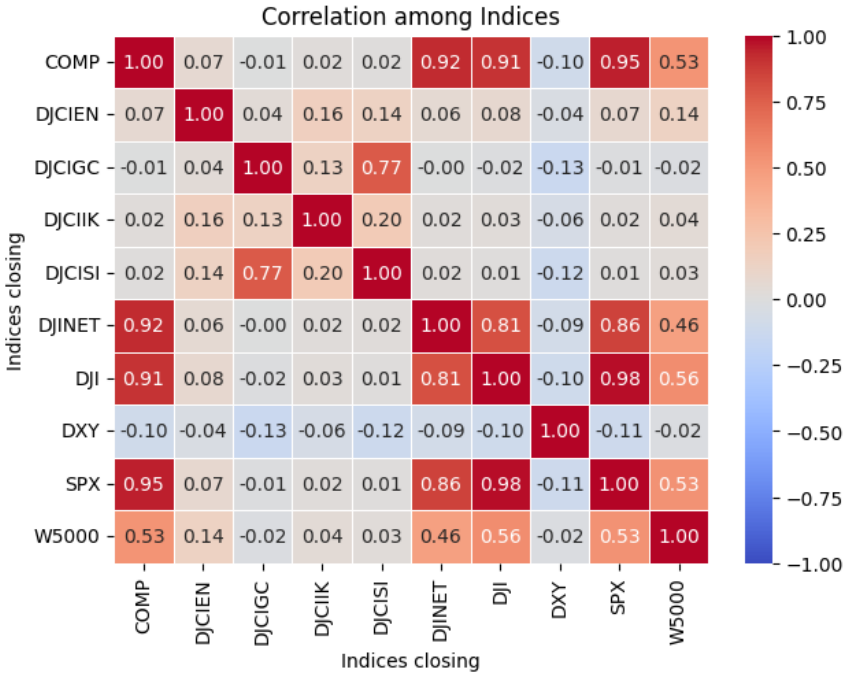


Figure 3.2.: Correlation among Indices

In addition, Figure 3.3 shows the correlations between indices and stocks. The correlation analysis can also be used to better diversify stock investments. However, this was not pursued further in our project due to time constraints.

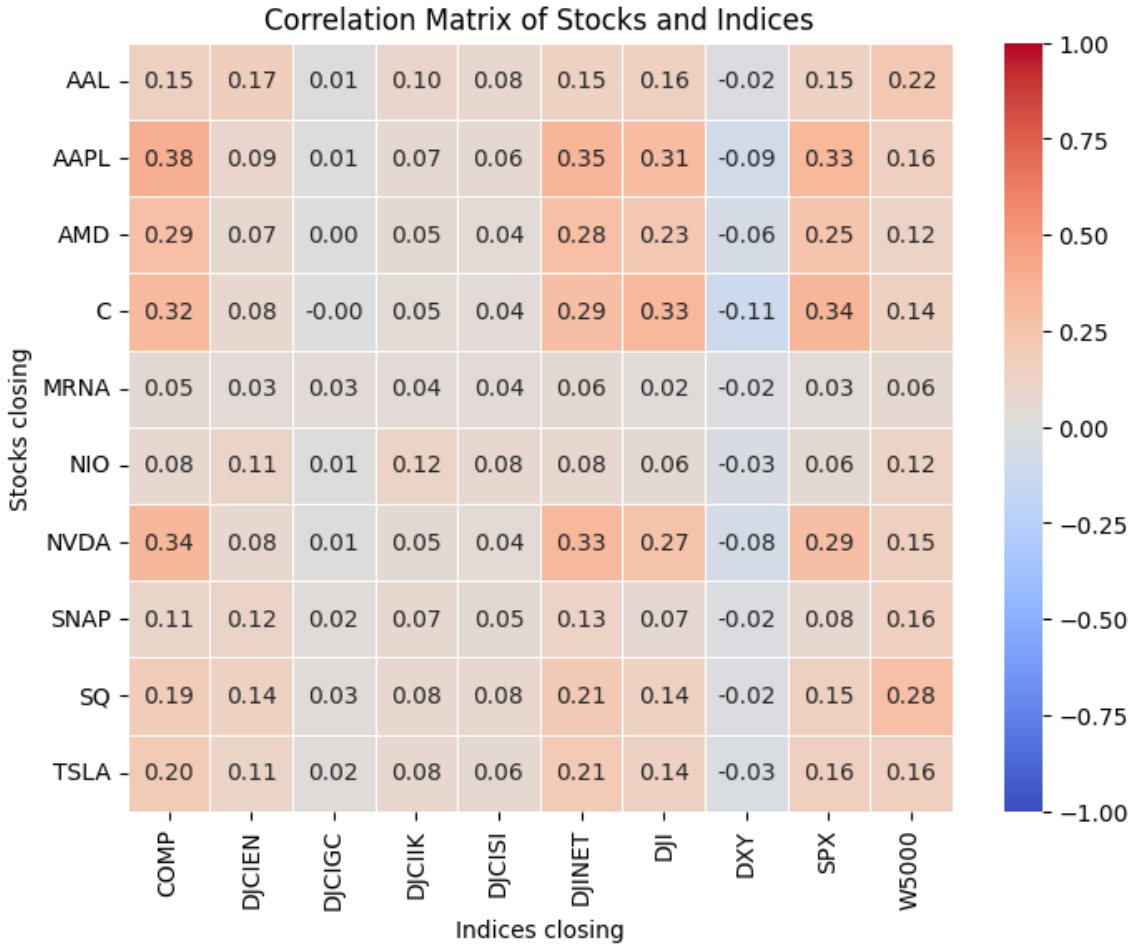


Figure 3.3.: Correlation Matrix of Stocks and Indices

3.2.2. Volatility

The volatility of a share is based on the changes in its price and provides an insight into the risk and stability of the share over time. These price changes reflect the short-term upward and downward movements of the share price and are a key indicator of market dynamics. In order to obtain a quantitative measure of the share's volatility, we calculate the standard deviation of the differentiated closing prices in the period from 1 January 2020 to 1 March 2021.

The standard deviation measures the dispersion or dispersion rate of the price changes from their mean value. A higher value of the standard deviation indicates higher volatility, which means that share prices fluctuate strongly over the period under consideration. In contrast, a lower standard deviation indicates a more stable share whose prices deviate less strongly from their mean value.

High volatile stocks are particularly attractive for day trading, where stocks are held and traded within short periods of time The reason for this lies in their potential for significant profits, which can be realised through targeted speculation on short-term market movements. This strategy takes advantage of the sharp price fluctuations of volatile stocks to optimise buy and sell decisions within a trading day.

However, it is crucial to be aware of the increased risk associated with high volatility. The increased uncertainty and unpredictability of price movements can also lead to significant losses. Therefore, successful day trading requires not only a careful selection of volatile stocks, but also a sound risk management strategy e.g. setting stop-loss orders and diversifying the portfolio. For this reason, we have analysed the volatility of our stocks.

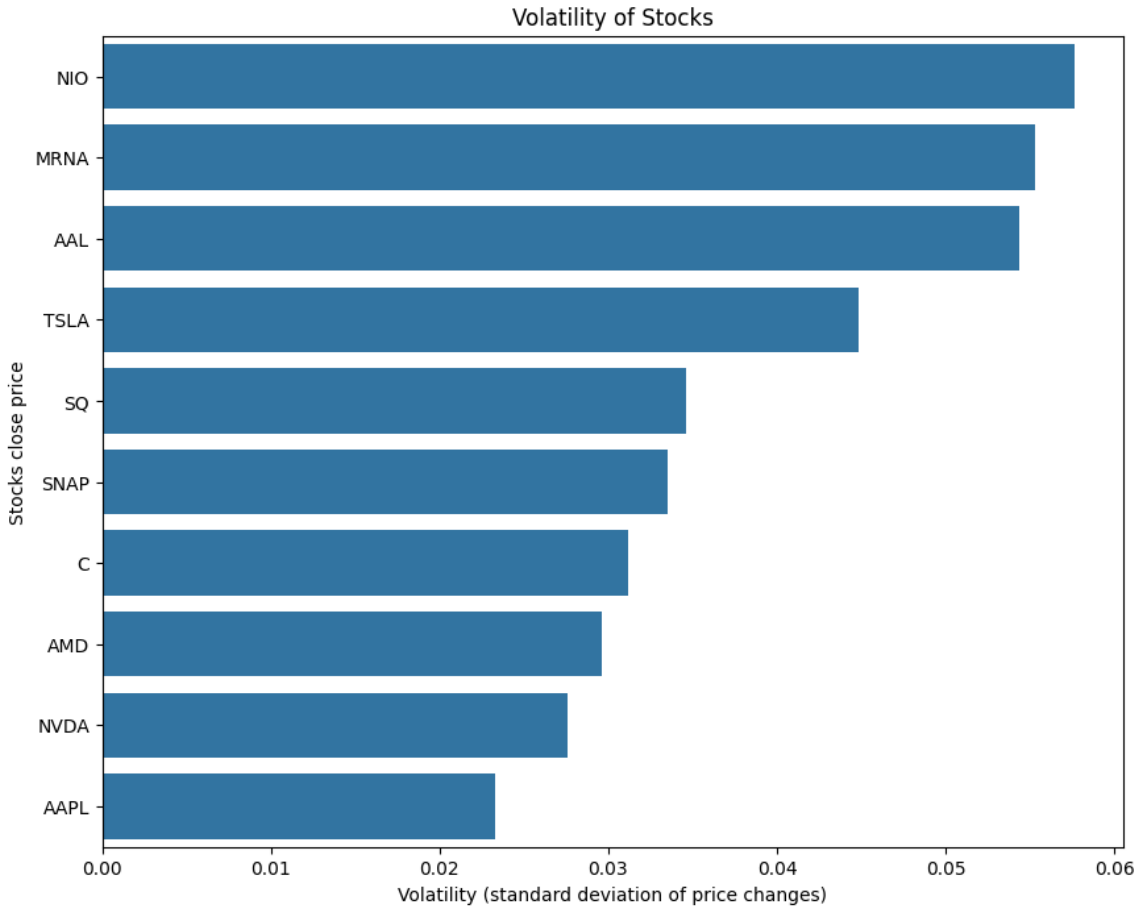


Figure 3.4.: Volatility of Stocks

3.2.3. Missing Values

The records of the stock and index prices start at different times (see Figure 3.5). To provide a better overview, we have depicted the time periods in a Gantt chart. The stocks are recorded until 19 March 2021, while the indices are only recorded until 5 March 2021. We have taken this into account when training our models by only including the data up to 1 March 2021 in our models. We used the data from the days after that to test the models in order to create comparability of performance.

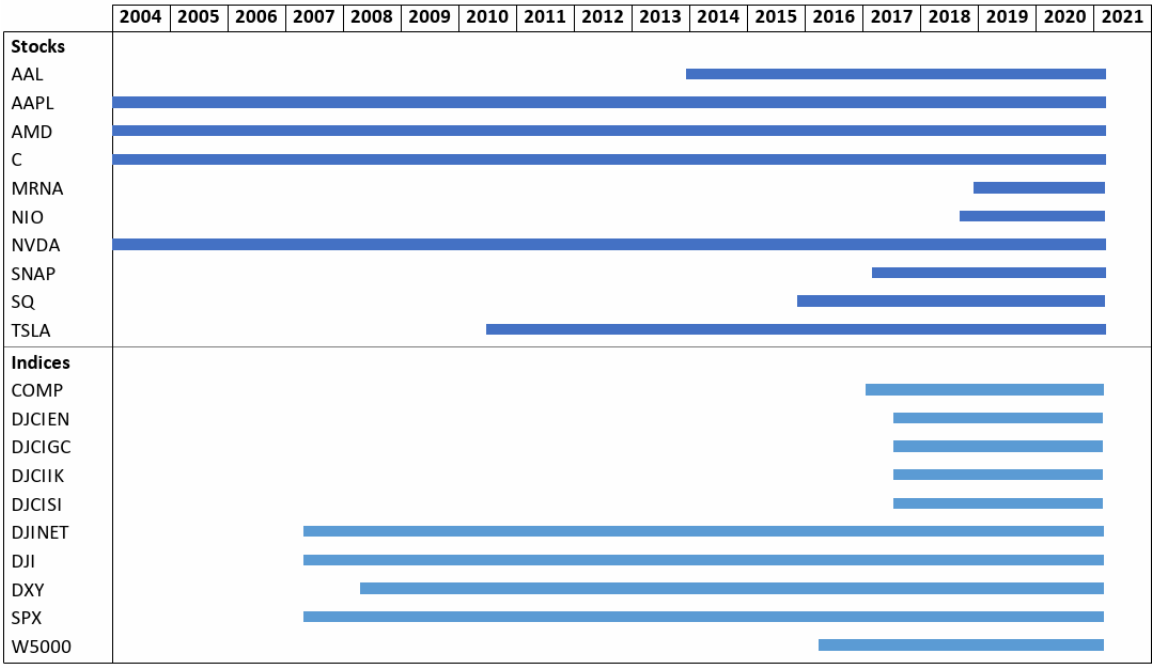


Figure 3.5.: Period for which the data is available

The trading times of the stocks and indices have an influence on which times we include in our training. We have therefore looked at the times at which we have non-zero data. As with the correlation analysis, we also take the per minute closing price as the basis. In line with our expectations, the figures show that the shares are only traded during the trading hours from Monday to Friday from 9.30 to 17.00 o'clock. The indices are also traded almost exclusively from Mondays to Fridays, although some of them have very different trading hours. It is particularly striking that DXY, i.e. the dollar exchange rate, is traded around the clock. This is also in line with our expectations, as money is always exchanged.

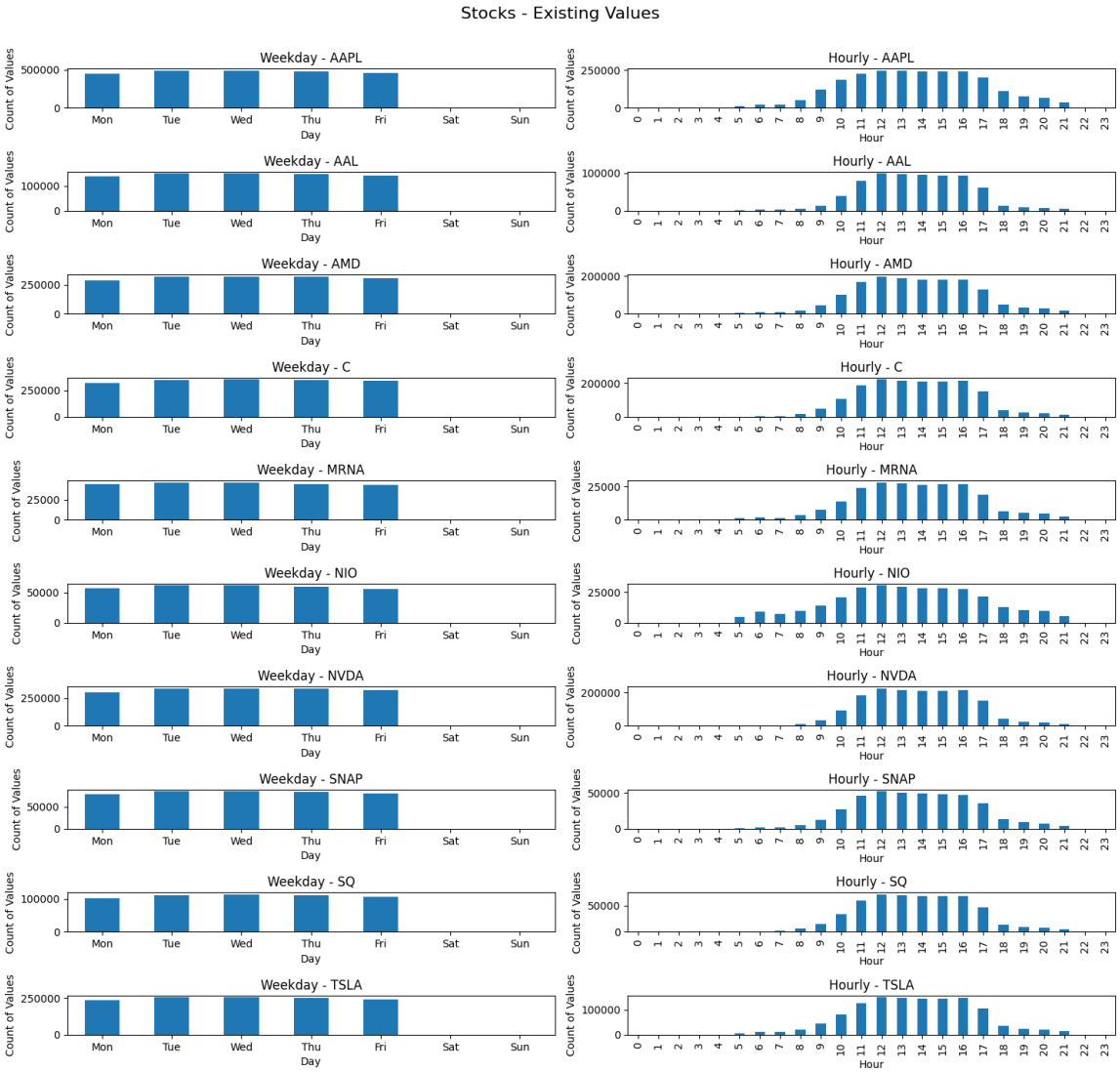


Figure 3.6.: Existing Values Stocks

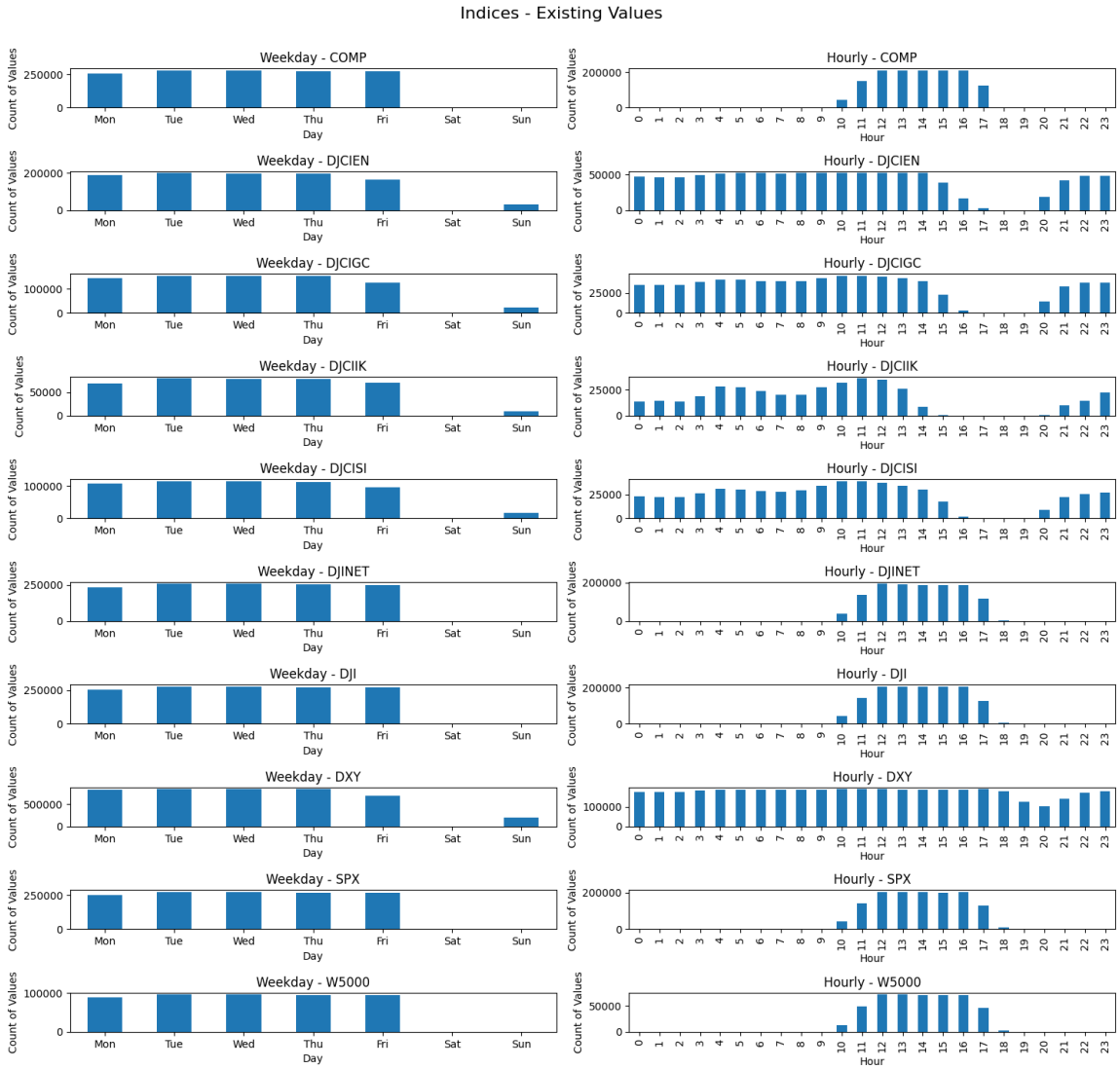


Figure 3.7.: Existing Values Indices

4

General Model Approach

This chapter explains the general approach we developed to use machine learning to implement the stock trading bot.

4.1. Multiple Models

The base of our trading bot is a reinforcement agent that decides on the trading actions. The inputs for the agent are price predictions from multiple models that forecast stock prices. In this way, we also used the ensembling approach, where you use different models to predict the same values and use another model, here the reinforcement agent, that learns in which cases which model produces better predictions. We had 3 main approaches for the development of the forecast models:

1. Statistical time series analysis, ANN and other machine learning models
2. CNN
3. Transformer

In the following there is a whole chapter dedicated to each of these approaches. To develop the models, we assigned a group to each of the approaches.

Besides the different machine learning approaches, we also developed models for different trading modes regarding the time frame. After we did some initial experiments with the different approaches, we agreed on testing the performance of all approaches on the following three modes:

Table 4.1.: Trading modes

Mode	**Time resolution**	**Prediction horizon**	**Remark**
Day trading	1 minute	20 minutes	Open at 10:00 close at 20:00
Swing trading	2 hours	2 days	Trading over night possible
Longterm trading	1 day	30 days	Trading over night possible

4.2. Model comparison

To compare the performance of the different models that predict stock prices, multiple metrics were developed and implemented. The evaluation of the models with these metrics is also implemented and visualised in the front-end to enable the user to choose a model for his investment strategy. For example, the user could select a specific model that performs better on stocks he wants to trade.

To evaluate the models we selected or developed three metrics:

- Mean absolute error
- Mean error
- Simulated portfolio performance

The mean absolute error is a common error measure that is easy to interpret. It is given by the formular:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{4.1}$$

The MAE is a measure for the variance of a model. It can be interpreted as the expected error a prediction might have, positive or negative, in $. This is easier to interpret than the MSE, which uses the sum of the squared errors. The unit of the values would be square dollar, which is not an useful unit, as it is difficult for the human brain to interpret polynomial growth.

The absolute error can be used as a measure for a bias in the prediction, i.e. a systematical error in positive or in negative direction. It is calculated by the formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} y_i - \hat{y}_i \tag{4.2}$$

The simulated portfolio performance is a more complex evaluation measure and was developed by ourselves. The idea behind it is to have a measure that gives an indication how a stock portfolio would have performed if you had based all trading decisions on the evaluated model. This is supposed to give a intuitive and simple indication to the user about the model's performance.

The difficulty in developing this measure was to translate pure price predictions into a trading policy. This kind of measure is not only sensitive to the performance of the model, but also to the design of the trading strategy that we implemented in this measure and how well it fits to a specific stock.

The measure is implemented by an algorithm that simulates trading activities based on the prediction of the model for a given time window. The inputs are the absolute predictions at each time step in the time window for each prediction time step and for each feature, i.e. stocks, and the true values for each stock at each time step in the evaluation window. At each time step the algorithm decides whether to buy 5 shares, to hold or to sell all shares for each stock individually. The algorithm keeps track of the money invested, the number of shares in the portfolio for each stock and the return from the cash-outs. At the end, the relative performance is calculated by the following formula:

$$Performance = \frac{Return - Moneyinvested}{Moneyinvested} * 100 \tag{4.3}$$

We made some simplifying assumptions that limit the meaningfulness of the measure based on this algorithm:

- The amount of shares that could be bought in one time step is limited to 5 per stock
- The cash to use for investments is unlimited
- There are no trading costs

- We can buy a stock to exacly the current price

As stated before, the trading strategy when to sell, buy or hold was the key challenge, as the models do not only predict a single step ahead. We decided for the most simple strategy:

- Buy: If all predicted values are higher than the current value
- Sell: If all predicted values are lower than the current value
- Hold: In all other cases

The idea behind this strategy is to only sell or buy in the worst or best case prediction. This strategy ignores all scenarios, where the price is predicted to fall first before rising again or vice versa, which could also be traded on. For example, one could argue that it was also profitable to buy when the predicted values fall before rising. Under the assumption that there is a limit to buy-orders, this could increase the amount of stocks to profit from the rise. With our strategy, the algorithm would only buy stocks after reaching the local minimum. We decided against a more granular trading strategy, because the limitations mentioned above imply that none of these strategies would represent the reality completely accurately anyway. Therefore, this measure should primarily be used to compare our models with each other. *(Niklas Kormann)*

5

Model: Reinforcement Learning

5.1. Inputs for an ensemble model

The decision to use an ensemble model for the trading bot is based on the premise that combining multiple indicators and models can provide valuable information for a collective trading decision. In addition to the models from other fields, indicators such as ma5, ma30, ma200, and RSI are selected because they are frequently mentioned in the literature, leading to the assumption that many investors pay attention to them. These indicators are intended to help better capture the relevance of price movements, thereby enhancing the predictive power of the model.

The selection of specific moving averages (ma5, ma30, ma200) reflects different time horizons and is applicable to various time scales. The hypothesis is that significant deviations from these average values, both positive and negative, can provide essential information about trend behavior („the trend is your friend“). It also allows for differentiation between short-term and long-term trends. The Relative Strength Index (RSI) was added to identify the strength of the current market trend and potential reversal points.

5.2. Use Reinforcement-Learning

Given the assumption that stock prices are mathematically challenging to predict, if not impossible, and that stock prices are formed through a broad spectrum of differentiated market participants with varying interests and views, an in-depth analysis of indicators is bypassed. Consequently, the judgment on the added value of models or indicators is determined through reinforcement learning. This approach relies on the use of an aggregation

strategy capable of consolidating indicators and outputs from various independently trained models. This strategy takes into account the performance of different models and indicators to achieve a more comprehensive assessment of the market situation. By aggregating decisions, the bot aims to make a more profitable trading decision.

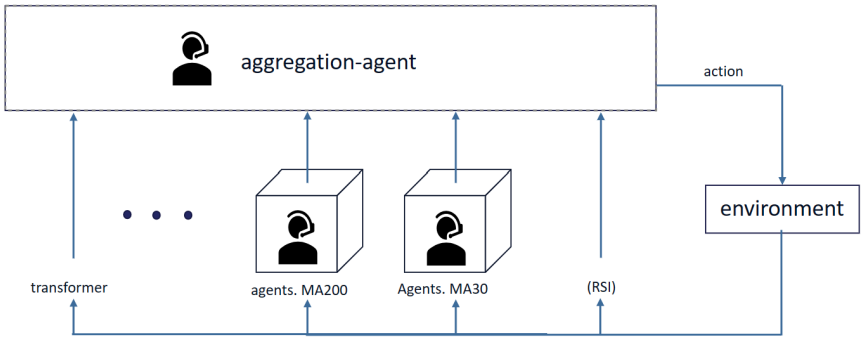


Figure 5.1.: aggregation-function

Additionally, several individual Q-Learning agents were deployed to learn specific tasks within the trading environment. Figure 5.1 illustrates the structure of collective decision-making.

5.3. Integration of individual agents

Individual agents are utilized in the trading model for specific tasks, learning from market data and acting based on these insights. Their main responsibilities include analyzing trends, detecting overbought or oversold conditions, and identifying potential turning points. Each agent focuses on particular indicators or market phenomena, allowing for a deeper analysis of market dynamics.

As input parameters, a discretized value of the standard deviation from the last periods for each indicator is used. This helps to reduce the state spaces and makes the model more adaptable to market volatility. By setting price deviations in relation to the standard deviation, the system responds more flexibly to significant market movements without being affected by daily fluctuations. This takes into account the increase in market volatility with rising market capitalization and avoids the use of absolute prices in the Q-learning table, which enhances the efficiency of the learning process.

5.4. Initial approaches

Several initial approaches, such as implementing stop-loss and limit orders, did not achieve the desired success. The challenge lay in pinpointing the correct entry and exit points without allowing normal volatility to trigger the orders undesirably. The approach suggested by our professor to experiment with distributions was not pursued further due to time constraints.

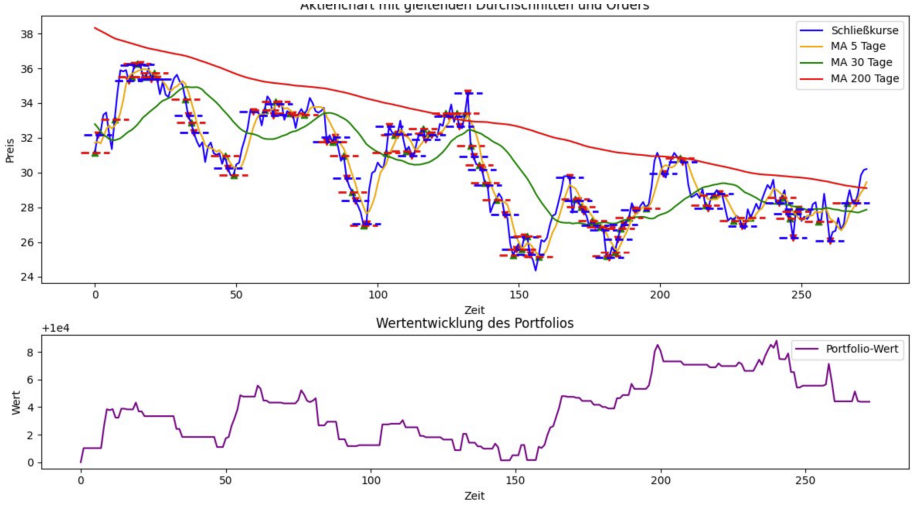


Figure 5.2.: Trial: stop and limit orders

Figure 5.2 illustrates such an attempt, aiming to maximize the portfolio value using stop-loss and limit orders. The dashed lines in the upper chart visualize the set stop-loss and limit orders (optimized through manual adjustment). The lower chart demonstrates how, despite a long-term negative trend, a return can be achieved through long positions. Unfortunately, the outcome, as shown in Figure 5.2, is not sustainably reproducible with this performance on other courses.

5.5. Reward System

The reward system in the Q-learning model is designed to encourage correct trading decisions while simultaneously minimizing risks. The system rewards actions that lead to a positive financial outcome and penalizes decisions that result in losses.

Specifically, a positive reward is given if the trading action—whether buying or selling—results in a price gain. Conversely, an action that leads to a price loss incurs a negative reward (penalty).

The reward system was optimized by considering the losses/gains not just as a binary function but also incorporating the portfolio value. This was initially implemented in a trial with an iterative approach per period. The final portfolio value punished or rewarded the episode. However, this learning process proved to be very slow. To optimize this, rewards/penalties were given based on the percentage change in the portfolio value after each decision. Substantial positive changes received additional rewards, while significant losses were penalized more heavily. This approach led to an increase in the learning curve by equally valuing portfolio value maximization throughout the period with each action.

5.6. Conclusion

The project has demonstrated that utilizing Q-Learning agents is a promising approach in the domain of algorithmic trading. These agents were able to trade by learning from market data and adjusting their strategies based on a predefined reward system. This highlights the Q-Learning method's capacity to identify and leverage complex patterns and trends within the financial market. Whether this approach can yield an excess return will be revealed in the live presentation.

Looking forward, an intriguing prospect for future developments is the exploration of strategies that optimize stop and limit orders through the application of distributions. This could allow for a more nuanced control of order execution, thereby enhancing the effectiveness of the trading strategy.

Another consideration is transitioning from a simple aggregation function to a Multi-Agent Reinforcement Learning (MARL) strategy for collective decision-making. This approach could enable multiple agents to learn to act together and coordinate their decisions in a manner that maximizes the overall success of the system.

Finally, the implementation of Deep Q-Learning (DQL) was contemplated as an advanced extension of the project. Although preliminary steps were taken in this direction, this approach was not fully realized due to a realignment within the team. DQL holds the promise of deeper analytical capabilities and better adaptation to the dynamic conditions of the financial markets through the use of deep neural networks.

Overall, the project has provided significant insights into the opportunities and challenges of deploying Reinforcement Learning in the context of algorithmic trading. (Joel Wälchli)

6

Model: Transformer

This chapter explains the work of the Transformer Group. The group consists of Luca Nickel, Philipp Duwe, Niklas Kormann and Eira Hammerich. As we worked together on all tasks within the group, we did not assign the subsections to authors.

6.1. Internal group communication and organization

In the Transformer group, we placed great value on very close cooperation. We continuously worked together on a joint software and a joint backlog. This section is subdivided into two sections: technical teamwork and interpersonal collaboration.

6.1.1. Technical teamwork

> The real measure of success is the number of experiments that can be crowded into 24 hours.
>
> *Thomas Alva Edison*

Due to the small amount of existing research in the field of stock price forecasting with transformers, there was uncertainty as to which technical approach was the right one. We therefore decided to adopt an agile way of working as a team. We used a Kanban board to distribute our tasks and organize group work. We developed the software iteratively using lean startup-like iterations.

Figure 6.1.: Workflow of the transformer group

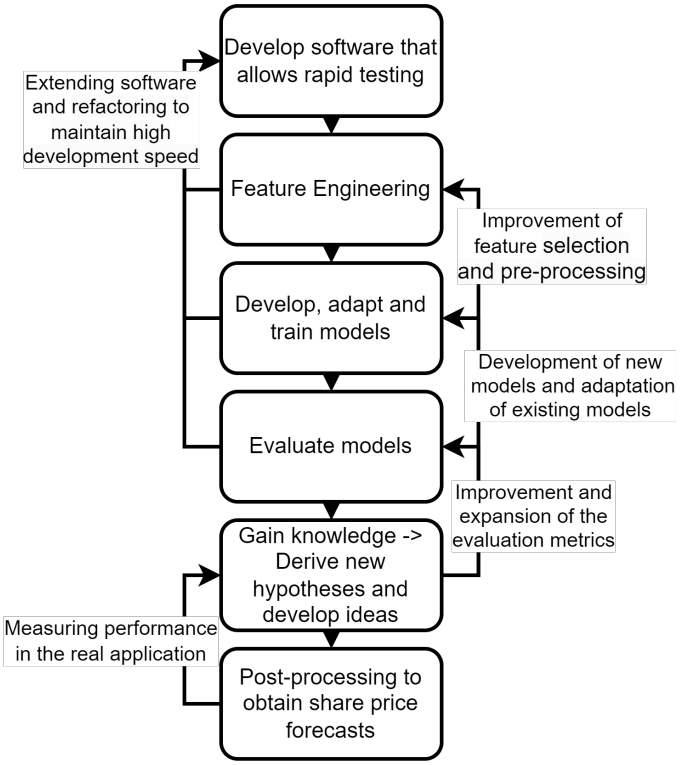


Figure 6.1 shows our technical project flow. As described in the previous quote, our goal was to be able to experiment as quickly as possible in order to learn as much as possible and to take further development steps in the right direction. Section 6.2.1 describes the software design that allowed us to iterate quickly.

6.1.2. Interpersonal collaboration

The group work in the transformer group was characterized by very regular communication. During the semester, we had 3 weekly group meetings plus the group leader meeting and the official meeting with the lecturers. These regular meetings with the well-maintained Kanban board and protocols of the meetings, including the distribution of tasks, enabled us to have an efficient and effective workflow. This meant that everyone knew what needed to be done to move the project forward. We regularly shared insights and hypotheses, and we explained technical concepts to each other. In particular, our discussions often helped to resolve

errors in thinking and to come up with new ideas as a group. We also asked each other for advice when we were stuck on implementing a feature. For particularly tricky problems, we often made appointments for two group members to work on the problem together. For quick interim questions, we used our Teams channel, in which we also communicated with other stakeholders. We also used a WhatsApp group to enable very fast communication when necessary. In summary, the work in the transformer group was characterized by a goal-oriented, motivated way of working and a lot of learning from each other.

6.2. Research

This chapter describes the work of the Transformer group within the research project.

6.2.1. Software Framework

The agile way of working and the number of experiments conducted were enabled by the software framework described in this section.

1. Project Structure

This chapter highlights how the Transformers group built their project, meaning the code repository which is used to train transformer models and deliver predictions to the backend. The structure of the Transformers group's project directory is shown below. Afterwards, the role of each file or directory will be shortly explained.

```
transformers/
├── data/
├── notebooks/
├── scoring_functions/
├── src_transformers/
├── api.py
├── compose.yaml
└── Dockerfile
```

```
├── README.md
├── requirements_torch.txt
├── requirements.txt
└── setup.cfg
```

The `data/` directory covers all aspects of data management for our project. This includes the input data for model training, output data such as scaler and model files, configuration files for the training pipeline, and miscellaneous data such as index mappings. The `notebooks/` directory contains a notebook dedicated to analysing data distribution. It covers temporal distribution, the impact of normalization, and the outcomes of outlier detection techniques. Furthermore, the `scoring_functions/` directory contains specialized evaluation functions for comparing model performance and assessing sample data to validate their functionality. Lastly, the `src_transformers/` directory contains the core codebase that is essential for both model training and prediction. The structure of this folder is described in more detail at the end of this chapter.

The `Dockerfile` used to build the transformers Docker container is essential for including the dependencies and configurations needed for our project. The `compose.yaml` file, which is part of our containerized infrastructure, is used to run the train and prediction pipelines. It also facilitates the startup of Tensorboard, providing a visual representation of the training processes. The `api.py` file serves as the central API within the transformers prediction container. This container is built in the `docker-compose.yaml` file, which is located one level above the transformers group directory. The API receives requests from the web application backend and generates model predictions that are then returned seamlessly.

The `README.md` file serves as a detailed guide, providing information on the project's structure, functionality, installation, and usage instructions. The `requirements_torch.txt` file specifies the PyTorch requirements and the Index URL from which to download them, ensuring seamless compatibility and setup for the PyTorch framework. The `requirements.txt` file lists all additional Python libraries essential for the project, streamlining the setup process and ensuring a consistent environment. At last, the `setup.cfg` file specifies the configuration settings used by autopep8 to improve code readability and ensure adherence to style guidelines.

In the following, the src_transformers directory is explored in more detail, as it contains the codebase that is actually used to train models and produce predictions.

```
src_transformers/
├── models/
├── pipelines/
├── preprocessing/
├── test/
├── utils/
├── abstract_model.py
├── main.py
└── prediction_interface.py
```

The `models/` directory stores all model classes that encapsulate the core components of our predictive models. It is essential for organizing and storing various model implementations. The 'pipelines/ directory includes the trainer class and relevant helper classes that orchestrate the end-to-end training process, providing a streamlined workflow for model development. The `preprocessing/` directory contains dataset classes, preprocessing code, and helper classes. This section is crucial to ensure that input data is correctly processed and ready for model ingestion. The `test/` directory contains Pytest test cases to validate the robustness and accuracy of the implemented functionality. However, we have not consistently created test cases for our entire codebase. This is an area for improvement in future projects. The `utils/` directory includes classes and scripts that offer supportive tools for different aspects of the project, such as the logger class and visualization scripts. These tools improve the overall utility of our codebase.

The `abstract_model.py` file defines the abstract class that acts as the foundation from which `prediction_interface.py` inherits its core functionality. `main.py` serves as the entry point for running pipelines within the transformers codebase. It acts as a central component for initiating and managing various project workflows. To run a pipeline, the user would call main.py in Python, passing a path to a config file and the chosen pipeline as arguments on the command line. The config file contains important information for the pipeline, such as model, training and dataset parameters. More information about config files is given in the next section. Finally, `prediction_interface.py` represents the Transformer interface class, which is responsible for generating predictions and passing them to the Transformer API.

2. YAML Configuration Files

We use YAML files to adjust the settings and parameters necessary for model training. The configuration files contain all the necessary information, including model parameters (such as transformer encoder and decoder length), training parameters (including batch size and the number of epochs), and dataset parameters (such as the start and end dates of the input data and the specific stock symbols used). Our code dynamically extracts information from these files, creating instances of the model and dataset classes and configuring their attributes based on the parameters specified in the YAML configuration. The trainer instance also uses the training parameters from the configuration file to orchestrate its methods. Below this paragraph you can see a shortened version of one of our transformer model training configs.

```yaml
use_gpu: true
model_parameters:
  torch_transformer:
    num_heads: 4
    num_layers: 2
    d_ff: 80
    ...
training_parameters:
  batch_size: 10000
  epochs: 600
  learning_rate: 0.00125
  ...
dataset_parameters:
  read_all_files: false
  first_date: 2000-01-01 # YYYY-MM-DD First date of the data to be read
  last_date: 2021-01-03 # YYYY-MM-DD Last date of the data to be read
  ...
  encoder_symbols:
    - "AAPL" # Apple
    - "AAL" # American Airlines
    - ...
  decoder_symbols:
    - "AAPL" # Apple
    - "AAL" # American Airlines
    - ...
```

By selecting YAML as our configuration file format, we obtain several benefits over other formats like JSON. YAML's human-readable and intuitive syntax makes it easy to understand, while its structured organization enables the logical grouping of related configurations. Additionally, YAML is easy to edit and maintain, allowing for adjustments to be made using a simple text editor. YAML supports a variety of data types, including lists and dictionaries. This increases flexibility and allows different configurations to be displayed with ease. YAML supports comments within the file. This enables clear communication of the meanings of parameters. YAML is widely supported across programming languages, ensuring compatibility and ease of integration. Python allows for easy conversion of YAML files into dictionaries. These dictionaries can then be unpacked to easily set class attributes of an instance in its constructor.

In our project, YAML configuration files offer even more advantages. They make it easier to start new experiments by allowing users to copy the configuration from a previous experiment and modify only the necessary parameters. The dedicated config directory and appropriately named files provide an easy-to-navigate overview of configurations, establishing a historical record of experiment settings over time. The historical record not only makes it easier to reuse parameters but also allows for quick adjustments for new experiments. Using YAML configurations is especially advantageous when code changes occur, as explained in the following section on expandability.

3. Expandability

The expandability of our project is a fundamental aspect inherent in its design concept. This is particularly apparent in connection with our configuration system. When a new attribute is added to a class, such as in the dataset class, the only requirement for the code to seamlessly run is to ensure that the used configuration file reflects these changes, incorporating the new attribute as a parameter. This is because the configuration files are simply read into dictionaries. Then, when an instance of a class is created the parameters of the config, now key-value pairs of the dictionary, can simply be unpacked in the constructor call. Therefore, if the new attribute is a parameter in the config, an instance of the updated class can be initialized succesfully. This streamlined process extends to the expansion of model or dataset classes and even the trainer class.

Furthermore, the inclusion of a completely new model in the project can be simply achieved by updating the model mapping in `src_transformers/pipelines/constants.py`. This ensures that the model name in the YAML configuration file can be accurately matched to the corresponding model class. Similarly, introducing a new dataset involves adding it to the preprocessing package, just like adding a model to the models package. To distinguish the chosen dataset in the configuration, a dataset mapping, similar to the model mapping in `constants.py`, must be created. Finally, the project supports the addition of new pipelines in a straightforward manner. By adding them to the pipelines package and adjusting the arguments recognised by the argument parser in `src_transformers/main.py`, they can be immediately integrated into the project. This inherent flexibility allows developers to easily extend and enhance the project as their requirements evolve.

4. Reusability

In summary, it can be noted that our framework does not restrict the user to training transformer models for stock price forecasting. The inherent flexibility of our project structure allows different models to be trained on different datasets, making it a valuable resource for a wide range of projects. The lack of barriers to incorporating new models and datasets underlines the adaptability of the project, ensuring that it can be easily reused for a range of applications.

In particular, the training pipeline, exemplified by our trainer class, demonstrates a robust level of abstraction that makes it indifferent to the specific model or dataset used. This abstraction enhances the overall reusability of the project, as developers can use the training pipeline for different models and data types without extensive modification. Regarding reusability, our dataset class, originally designed for stock price data, transcends its original scope and proves applicable to a broader context of multivariate time series data. This means that our project provides an easily adaptable solution for those who wish to forecast time series with transformers.

6.2.2. Training

In this chapter we describe how we improved the training of our models to get the best results as fast as possible by optimising the learning rate and the sampling process.

1. Learning Rate

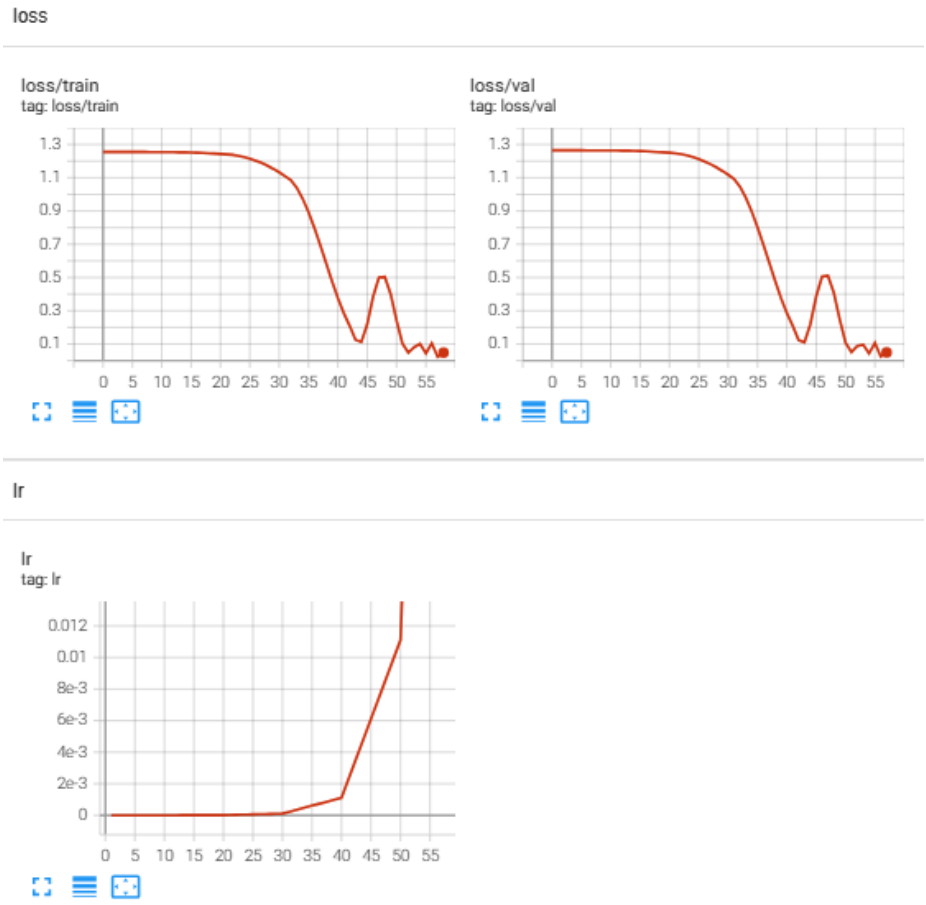
To crowd as many experiments as possible into 24 hours, like Edison recommends, we tried to find the optimal learning rate. This means the learning rate should be as high as possible to decrease the train duration on the one hand, but must be small enough to find a good optimum in the loss landscape on the other hand. To optimise the training in terms of the learning rate, we had two approaches:

1. Testing for optimal learning rate
2. Using learning rate scheduler

1.a Optimal learning rate

To find the optimal learning rate, Jeremy Howard recommends a method where you test for the optimal learning rate by increasing the learning rate while training [5]. We implemented this method directly in the training function to test for optimal the learning rate with one model. For the test, we started with an initial learning rate of 0.0000001 and increased the learning rate with each epoch. The results of the experiments can be seen in the figure below. The optimal learning rate is where the slope of the loss chart is the steepest. For our experiment, the optimal learning rate would be around 0.001.

Figure 6.2.: Results from test for optimal learning rate.



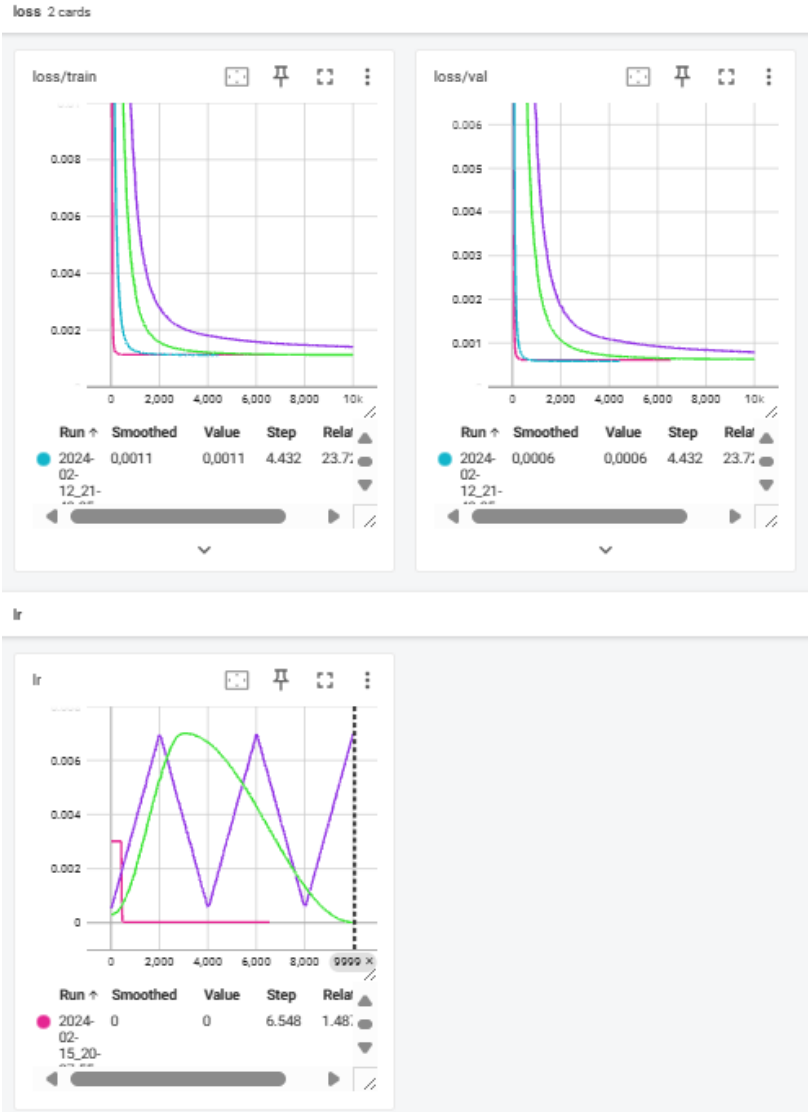
1.bLearning rate scheduler

Another approach to optimise the training is to use learning rate schedulers, which adjust the learning rate over time while training the model. We implemented three scheduler:

- One cycle learning rate
- Cyclic learning rate
- Reduce on plateau

We used the pytroch functions for all three scheduler in our implementation. The cyclic scheduler reduce the risk of suboptimal initialisation of the model parameters, as they start with a slow learning rate, before increasing it [6]. The scheduler *reduce on plateau* reduces the risk of jumping over the local minimum due to a high learning rate. All three scheduler did not increase the performance of our models nor the speed of the training compared to a fixed learning rate. Therefore, we did not use them for the rest of our training.

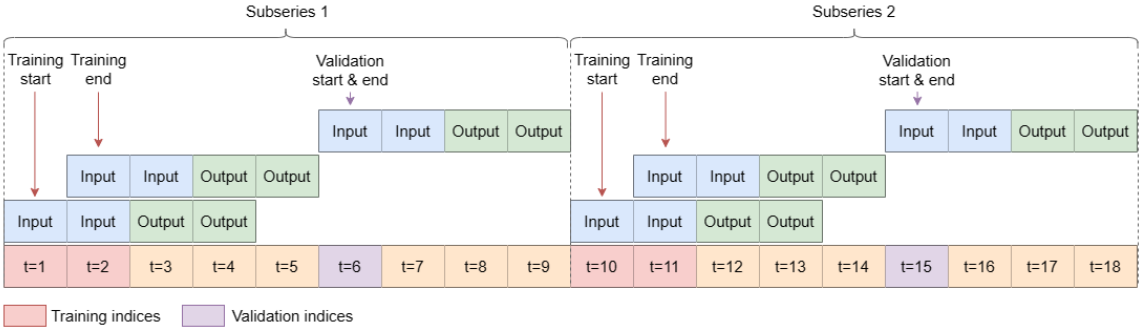
Figure 6.3.: Results from tests with learning rate scheduler.



Subseries

One dilemma that comes with applying machine learning to time series, is the importance of the last samples for the training and the validation. Whenever there is a trend in the data, most current samples are structurally most similar to the samples in the future, we want to predict. Therefore, these samples add most value to the training but also must be included in the validation set to give an accurate measure of the performance on future samples. So, we tried to avoid having one large validation set at the end of the whole dataset by splitting the dataset into subsets. As we always predict a sequence of values, we had to make sure that each timestep is exclusively included in the train or the validation set to avoid data leakage. To do so, we wrote a function in the dataset class, that determines all indices in the dataset that can be pulled for the train and the validation set. The fig. 6.4 shows how this would work on an exemplary dataset.

Figure 6.4.: Example of sampling the data in sub series



For the training set, samples can be pulled from the indices 1, 2, 10 or 11. For each of the indices, the input and output sequences are sampled from the data as shown above. This also has the advantage, that we can sample randomly for batch training.

6.2.3. Data

The transformer group uses an 9-stage process for data preprocessing which is described below.

1. **Reading the data:** Import of the relevant data per share or index for pre-processing.
2. **Union of the time stamps of the features used:** Different time stamps are available for the data used. A union of all timestamps available in the data is formed.
3. **Merging the features into a data frame:** A data frame is created that contains the information of all features.
4. **Fill in missing values in the data frame:** As the features do not all have data for all time stamps, the data is completed with data imputation.
5. **Adjust time resolution of the data:** Data is aggregated to a time resolution chosen by the user.
6. **Apply the differencing to the closing prices:** It is not the absolute price data that is used, but the percentage change in the price data.
7. **Add additional time features:** Additional time features are created to give the model additional information.
8. **Outlier detection:** There are outliers in the trading volumes. Therefore, these outliers are detected and clipped, to make the predictions more robust.
9. **Scale trading volumes:** The trading volums differ from stock to stock. They are therefore scaled in order to improve training and prediction quality.

1. Reading the data

The data used in this project are provided as CSV files. The structure of all raw data, each of which represents a price history, corresponds to fig. 6.5. The information for the symbol and type columns is contained in the file name or folder name. They are stored in the data frame so that it contains all relevant information for data preprocessing.

Figure 6.5.: Input Data

timestamp	open	high	low	close	volume	symbol	type
2013-12-09 08:36:00	22.8 $	22.8 $	22.6 $	22.6 $	1,600	AAL	stock
...	...	...	...	...	...	...	...
2020-12-31 19:59:00	15.8 $	15.8 $	15.8 $	15.8 $	8,580	AAL	stock

Only the closing prices per minute for stocks, indices and ETFs are used for the next steps. The trading volumes are also used for stocks and ETFs. Indices cannot be traded directly but only via ETFs and therefore don't have trading volumes.

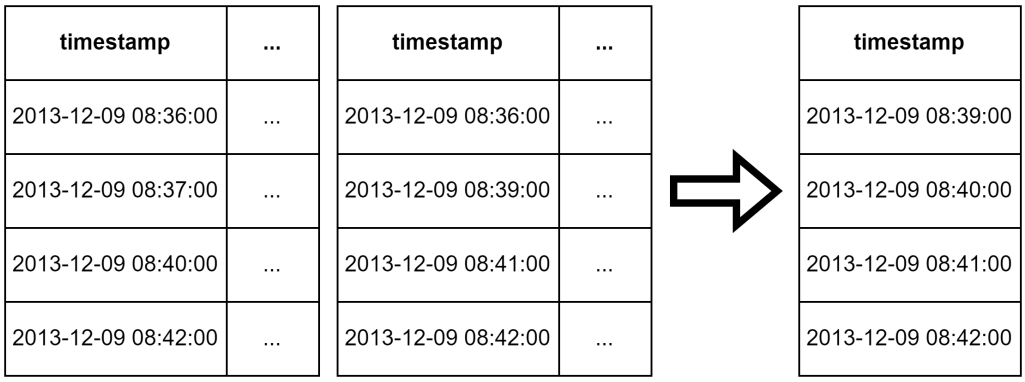
2. Union of the time stamps of the features used

As illustrated in fig. 6.6, there is data for the available price data at different time stamps. In order to be able to train machine learning models with the data, it is important that the intervals between the time stamps are equidistant. Only then can the models learn temporal relationships properly. In addition, the prediction can only be assigned to specific points in time if the input data and the target data in the training can be assigned to specific points in time.

The mechanism we use to generate a complete equidistant time series from the different, not always equidistant time series is very reliable and robust. The process is illustrated by the arrow in fig. 6.6. To the left of the arrow is the input data and to the right is the result data of the mechanism. While data is only available for the individual shares and ETFs when they are traded, price data is always available for the indices. We create a union of all available timestamps of all features. Since we always train with several stocks and indices, we always have all timestamps in the relevant time interval in practical use. Compared to a seemingly more robust software-based replenishment of the time series, this has the great advantage that we can perfectly reproduce the real opening hours of the stock exchanges under consideration without manual feature engineering. We also map off-exchange trading. If the exchange is closed due to public holidays or other events, this is automatically taken into account in this approach.

One disadvantage of this approach is that it always generates the maximum amount of data by using a union set. This is the desired behavior when using stock indices and stocks and ETFs. However, if we use a foreign exchange index as a feature, then our data will also contain points in time when no stocks and ETFs were traded. This is because forex is always traded. However, equities are not always traded. We therefore no longer use foreign exchange indices as a feature, because otherwise the model would have too many points in time without a change in share prices in training and would therefore systematically underestimate the volatility of share prices at trading time.

Figure 6.6.: Merge timestamps of the different features



In addition to creating an equidistant time series, we also use this mechanism to select the quantity of data used. An example of selecting a recent subset of the available data is shown below.

all_timestamps = {2013-12-09 08:36:00, 2013-12-09 08:37:00, 2013-12-09 08:39:00, 2013-12-09 08:40:00, 2013-12-09 08:41:00, 2013-12-09 08:42:00}

data_usage_ratio = 0.67

used_timestamps = {2013-12-09 08:39:00, 2013-12-09 08:40:00, 2013-12-09 08:41:00, 2013-12-09 08:42:00}

The set *all_timestamps* is the set of all timestamps that we have obtained by applying the mechanism to the input data. With the parameter *data_usage_ratio* we can select what percentage of these timestamps we want to use. It is not possible to use all the data at a high time resolution because we do not have the computing capacity to train a model with such a large amount of data. The set *used_timestamps* contains the result data set that is used to train and evaluate the models. The most recent data is always used and older data is eliminated by using a *data_usage_ratio* of less than 1, as the more recent the data points in a time series are, the more relevant they are.

3. Merging the features into a data frame

Now that we have defined the timestamps we want to use, we create a data frame that contains all the data we want to use. Figure 6.7 shows the data frame containing the time points we want to use on the left. The two data frames to the right of the time points in fig. 6.7 contain the data of two stock prices that we want to predict. These three data frames are combined into one data frame in the next step. This combined data frame will be used for training and forecasting.

Figure 6.7.: Features and timestamps that are to be combined into a data frame.

timestamp
2013-12-09 08:39:00
2013-12-09 08:40:00
2013-12-09 08:41:00
2013-12-09 08:42:00

AAPL

timestamp	close	volume
2013-12-09 08:36:00	20.1 $	100
2013-12-09 08:37:00	20.3 $	50
2013-12-09 08:40:00	20.7 $	200
2013-12-09 08:42:00	21.0 $	10

NVDA

timestamp	close	volume
2013-12-09 08:36:00	40.5 $	200
2013-12-09 08:39:00	40.1 $	100
2013-12-09 08:41:00	39.8 $	50
2013-12-09 08:42:00	40.0 $	500

Figure 6.8 shows an example of the result of merging the three data frames from fig. 6.7. We use the sorted set of all time points that we want to use as the time index for the result data frame. In the next step, for each feature in the period under consideration, we insert all data points in the results data frame. The result is a data frame that contains the data of all features for a defined time period. Since not all stocks and ETFs are always traded, this data frame contains holes, as can be seen in fig. 6.8.

Figure 6.8.: Combine all input features into one data frame.

timestamp	close AAPL	volume AAPL	close NVDA	volume NVDA
2013-12-09 08:39:00			40.1 $	100
2013-12-09 08:40:00	20.7 $	200		
2013-12-09 08:41:00			39.8 $	50
2013-12-09 08:42:00	21.0 $	10	40.0 $	500

4. Fill in missing values in the data frame

The data frame shown in fig. 6.8 is now filled so that it no longer contains any holes. In the close columns, we first perform a forward fill. This corresponds to reality, as the last traded price has not changed. Then we do a backfill to fill empty fields. This is necessary because, as can be seen in fig. 6.8, there may be no data for some prices up to a certain point in time. The closest price was the most realistic estimate for us to fill these gaps. We fill the gaps in the trading volumes with zeros. This also corresponds to reality. If a share or ETF has no data at a point in time, then it was not traded at that time. The result of the filling can be seen in fig. 6.9. The bold values come from the raw data. The other values were imputed according to the procedure described. The trading volumes of indices are generally ignored as the indices are not traded.

Figure 6.9.: Feature Imputation

timestamp	close AAPL	volume AAPL	close NVDA	volume NVDA
2013-12-09 08:39:00	20.7 $	0	**40.1 $**	**100**
2013-12-09 08:40:00	**20.7 $**	**200**	40.1 $	0
2013-12-09 08:41:00	20.7 $	0	**39.8 $**	**50**
2013-12-09 08:42:00	**21.0 $**	**10**	**40.0 $**	**500**

In addition, nights can be ignored in this step via a parameter. This is advantageous because there is no trading during the nights. This reduces the volatility in the data and the trained models systematically underestimate the volatility of the prices under consideration at trading times. The start and end of a night can be configured. Explicitly ignoring nights is only necessary if foreign exchange indices are used as a feature. Otherwise the nights are not included in the data.

There is also the option to explicitly ignore weekends. This is also only necessary if foreign exchange indices are used as a feature. Otherwise the weekends are not included in the data.

5. Adjust time resolution of the data

The raw data is available with a time resolution of one minute. With a minute resolution, the signal to noise ratio of stock market data is very unfavorable. In order to make good predictions, we therefore wanted to be able to test our models with different time resolutions. The time resolution in minutes can be set by a parameter in our code. Figure 6.10 shows a data frame that results if we only want to use Apple's share price as the independent and dependent variable. The time aggreagation, which we use to adjust the time resolution, is explained below using fig. 6.10 and fig. 6.11. In this example, the time resolution is changed from every minute to every 5 minutes.

Figure 6.10.: Data before aggregation

timestamp	close AAPL	volume AAPL
2013-12-09 08:30:00	120.0 $	**200**
2013-12-09 08:31:00	120.1 $	**400**
2013-12-09 08:32:00	120.3 $	**1,000**
2013-12-09 08:33:00	120.2 $	**200**
2013-12-09 08:34:00	**120.5 $**	**10**
2013-12-09 08:35:00	121.1 $	**50**
2013-12-09 08:36:00	121.0 $	**2,000**
2013-12-09 08:37:00	121.2 $	**50**
2013-12-09 08:38:00	120.6 $	**100**
2013-12-09 08:39:00	**120.3 $**	**300**

The bold values in fig. 6.10 are the values that are taken into account in the aggregation. The result timestamp for 5-minute intervals is the start of each interval. The result closing price is the last price value in an interval. The trading volumes are added up for each interval. This aggregation reflects reality as closely as possible. It shows the trading volumes and price changes in intervals as they occurred. The result of the aggregation can be seen in fig. 6.11. The time resolution was reduced from a minute resolution to a 5-minute resolution. In addition to the better signal-to-noise ratio, this has the advantage that we can use longer periods of time for training, as the number of samples for a period is reduced by the aggregation.

Figure 6.11.: Data after aggregation 5 minutes aggregation

timestamp	close AAPL	volume AAPL
2013-12-09 08:30:00	120.5 $	1,810
2013-12-09 08:35:00	120.3 $	2,500

Values that were added by the aggregation but for which there are no data points are removed. These can be nights and weekends, for example. The aggregation generates values for all time points between the first and the last time point in the input data.

6. Apply the differencing to the closing prices

The aim of machine learning models is to learn patterns in data. The models make predictions based on these patterns. The advantage of differencing is that the data is independent of absolute prices. This allows the model to better recognize recurring patterns in different features and at different price levels than the same pattern. This improves the results of the model. Therefore, it makes sense to predict the percentage change of stock prices instead of predicting the absolute prices. [1]

Therefore, we apply percentage differencing to the price data. Through percentage differencing, we convert the absolute price data into percentage price changes since the last point in time. Figure 6.12 shows the data from fig. 6.11 after applying percentage differencing.

Figure 6.12.: Apply percentage differencing to the data.

timestamp	close AAPL	volume AAPL
2013-12-09 08:35:00	-0.17 %	2,500

7. Add additional time features

In the exploratory data analysis we found that stock trading behaves differently at certain times. For example, most of the indices were not traded before 10am or after 6pm. Also, the trading volume increased towards the middle of the day. To enable the model to learn these patterns, we introduced the time features hour, day of the week and month. These features are being added to the dataset after differencing. We used the Posix time to derive these features from the time series. In order to make the different expressions of the features equidistant, we one-hot-encoded these features.

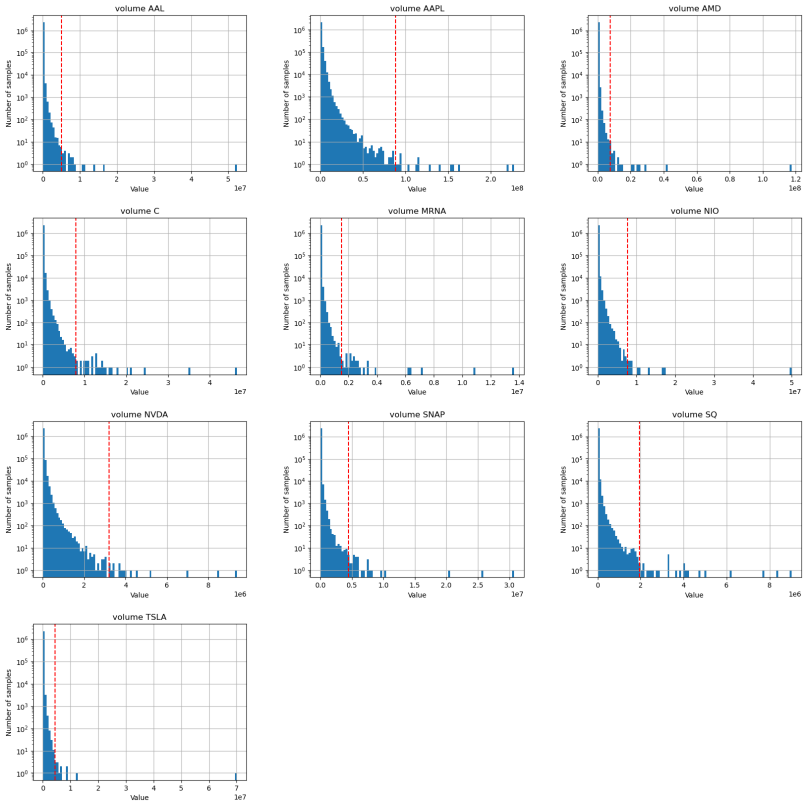
It should be noted that the introduction of the time feature greatly increased the training duration compared to the training without time features, as the one-hot encoding leads to many additional features. Therefore, we have not used these features in one of the final models.

8. Outlier detection

To detect outliers and exclude them from the training is important to train a robust and generalisable machine learning system. A widely used approach is the Z-score, which is the deviation of a sample to its mean, measured in standard deviation. Usually, samples with a Z-score of 3, meaning three times the standard deviation away from the mean, are treated as outliers. [8]

We discovered some outliers in our data, especially in the volumes. When we analysed the outlier, we found that the volume values are not distributed normally but follow a gamma distribution with alpha = 1, which is basically an exponential decline [15]. Therefore, we have implemented a method that is an adjusted version of the z-score. We defined a percentile and clipped all values above the doubled value of that percentile. For the data without aggregation, we defined to use the 0.99995-percentile as default. We doubled the value, because we discovered that the outlier detection itself is more robust against deviations in the shape of the distribution, when we define a smaller percentile and use a tolerance factor (here 2) than when we increase the percentile. This is an idea we used from the z-score where you also use a tolerance factor based on the standard deviation. The figure below shows the results, which values are above the limit for outlier detection.

Figure 6.13.: Distribution of volume values with limit for outlier detection.



9. Scale trading volumes

The last step of our data preprocessing is the normalisation of the volume values. This is required, because these values are much higher than the differentiated stock values and can reach values up to 100.000.000 as shown in Figure 6.13. This would lead to an over-representation of these features in the training process. Therefore, we used scaler from sklearn, to normalise these values to a range from 0 to 1. We tested min-max-scaling, standard-scaling, power transformation (to normal distribution) and quantile transformation (to normal distribution). In the end, we only used min-max-scaling, because this keeps the distribution the same. The standard scaling should only be used for values that are already normally distributed, because it scales with the mean and the standard deviation. The power and quantile transformation convert the values into a normal distribution. Usually,

this should improve the performance of machine learning models, but in this case the results were not usable due to a high number of zero-values, which causes a gap in the distribution after scaling. Figures of the distribution after the different scaling methods applied can be found in the appendix.

6.2.4. Models

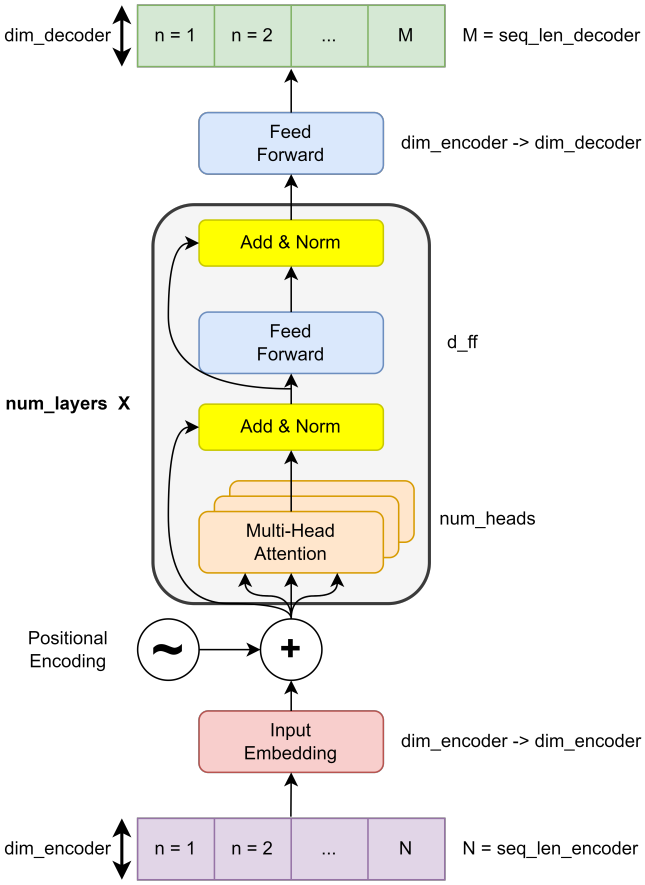
In this chapter, we describe the models developed by the Transformer Group. To enable fast training, all models can be trained with CUDA on the GPU.

1. Encoder Transformer Pytorch Model

The model described in this section is nicknamed *torch_transformer* within the Transformer group. It is, like *BERT* [3], an *encoder-only* transformer architecture. The model consists almost exclusively of a *TransformerEncoder* [11] from *Pytorch*. The *TransformerEncoder* is used nearly entirely with its default settings. The vanilla transformer was only minimally adapted in order to be able to work with the time series in this project. The model therefore impresses with its very simple implementation, which almost eliminates errors in the implementation of the model. It was therefore used as a benchmark for our group's own transformer developments. However, its performance is so good that we were rarely able to beat the model with our own developments.

Figure 6.14 is a visualization of the torch_transformer based on [13]. The parameters used to annotate the model can be set in the configuration file. These parameters can be set for all models in the transformer group. As the model only consists of stacked encoder blocks, the sequence length of the input sequence and the output sequence must not differ by default. However, we have resolved these restrictions by only outputting the first encoder_length elements of the output sequence. The remaining elements are discarded. This means that the output sequence can be shorter or as long as the input sequence. Since we have always used a significantly longer input as the output sequence in order to make successful predictions, this restriction is practically no problem.

Figure 6.14.: Structure of the Encoder Transformer Pytorch Model.



As we started the development with a classic transformer architecture, the following nomenclature has also become established for the *encoder-only* transformers. The dimension, i.e. the number of features in the input, is defined by the parameter *dim_encoder*. This is determined automatically when the data is pre-processed. The dimension, i.e. the number of predicted features in the output, is determined by the *dim_decoder* parameter. It is also determined automatically when the data is preprocessed. The length of the input sequence is determined by the parameter *seq_len_encoder*. The length of the target sequence is determined by the parameter *seq_len_decoder*. The independent variables are described by the *encoder_symbols*. The dependent variables are described by the *decoder_symbols*.

The *torch_transformer* is used in the prototype for the time resolutions of 2 hours and days. Table 6.1 shows the configuration for the *torch_transformer* for the 2-hour resolution. Table 2 contains the configuration for the daily resolution.

Table 6.1.: Best performing parameter configuration for the *torch_transformer* with 2 hours time resolution.

model_parameters	
num_heads	4
num_layers	2
d_ff	80
seq_len_encoder	96
seq_len_decoder	16
dropout	0
training_parameters	
batch_size	10000
epochs	2000
learning_rate	0.00125
loss	MSE
dataset_parameters	
first_date	2000-01-01
last_date	2021-01-03
data_usage_ratio	0.35
subseries_amount	4
validation_split	0.15
scaler	Min-Max-Scaler
time_resolution	120
ignore_nights	20:00 - 04:00
outlier_quantile	0.99995
encoder_symbols	AAPL, AAL, AMD, C, NVDA, SNAP, SQ, TSLA SPX, DJCIGC, DJCISI, DJCIEN, DJCIIK, DJI DJINET, COMP, W5000
decoder_symbols	AAPL, AAL, AMD, C, NVDA, SNAP, SQ, TSLA

Table 6.2.: Best performing parameter configuration for the *torch_transformer* with daily time resolution.

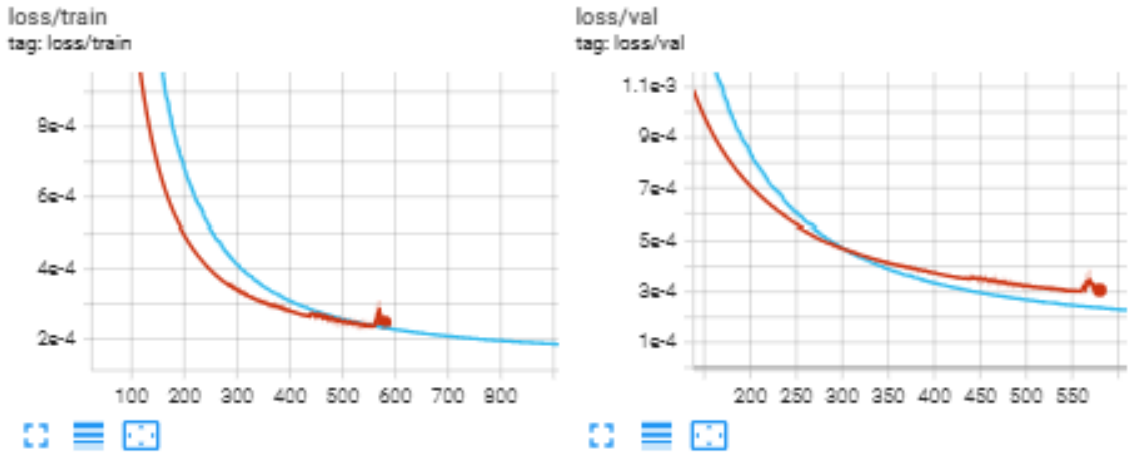
model_parameters	
num_heads	4
num_layers	2
d_ff	80
seq_len_encoder	90
seq_len_decoder	30
dropout	0
training_parameters	
batch_size	10000
epochs	800
learning_rate	0.00125
loss	MSE
dataset_parameters	
first_date	2004-03-01
last_date	2021-01-03
data_usage_ratio	1
subseries_amount	4
validation_split	0.15
scaler	Min-Max-Scaler
time_resolution	1440
ignore_nights	20:00 - 04:00
outlier_quantile	0.99995
encoder_symbols	AAPL, AAL, AMD, C, NVDA, SNAP, SQ, TSLA SPX, DJCIGC, DJCISI, DJCIEN, DJCIIK, DJI DJINET, COMP, W5000
decoder_symbols	AAPL, AAL, AMD, C, NVDA, SNAP, SQ, TSLA

2. Encoder Transformer Pytorch Model with Batch-Norm

In addition to the torch_transformer we implemented the same architecture again, but without using the transformer layer class from *Pytorch*. Instead, we implemented an additional option for this model, where we could use batch norm instead of layer norm. Zerveas et al. found out that this can improve the performance of transformers used for time series forecasting, as this can mitigate the effect of outlier values [18].

In a direct test with all parameters fixed, but one with batch norm and one with layer norm, the training with the batch norm was slightly better, as shown in the following figure. The loss curve of the training with batch norm (blue) falls a little bit slower in the beginning, but also flattens slower. Therefore, we used batch norm with this model for the following experiments.

Figure 6.15.: Loss curves: Batch norm (blue) vs. layer norm (red)



This model, which we have nicknamed the *transformer_encoder* within the transformer group, is used in the prototy for the minute time resolution. Table 3 shows the parameters with which we achieved the best performance with this model.

Table 6.3.: Best performing parameter configuration for the *transformer_encoder* with minute time resolution.

model_parameters	
num_heads	2
num_layers	3
d_ff	80
seq_len_encoder	60
seq_len_decoder	20
dropout	0
norm	batch normalization
training_parameters	
batch_size	10000
epochs	10000 (training ended earlier due to time constraints.)
learning_rate	0.002
loss	MSE
dataset_parameters	
first_date	2019-01-01
last_date	2020-12-31
data_usage_ratio	0.5
subseries_amount	4
validation_split	0.1
scaler	Min-Max-Scaler
time_resolution	1
ignore_nights	18:00 - 10:00
outlier_quantile	0.9999
encoder_symbols	AAPL, AAL, AMD, C, MRNA NIO, NVDA, SNAP, SQ, TSLA DXY, COMP
decoder_symbols	AAPL, AAL, AMD, C, MRNA, NIO, NVDA, SNAP, SQ, TSLA

@TODO Niklas: Final results

3. Time Series Transformer

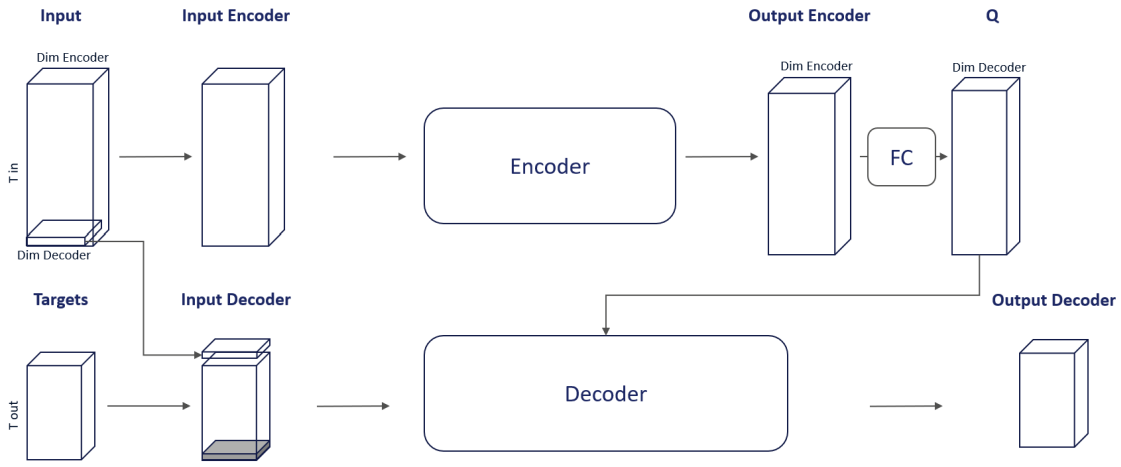
Our first approach was to adjust a complete transformer architecture which was designed for NLP based on Vaswani et al.[13] for processing time series. To process time series instead of language, we first got rid of the word embedding. Secondly, we had to adjust the architecture so that the encoder and decoder could use inputs of different lengths and different numbers of features. The idea was that the encoder uses a long time series with additional features as input data from past trades. The decoder output is shorter, uses only the features to predict, and is used as prediction for a given number of time steps in the future. The training would be done with a masked decoder input, like in NLP. For inference, the output of the decoder has to be recycled as decoder input for the next time step. The rest of the decoder input length is filled with zeros, until the whole prediction horizon is filled with predictions.

There where two small challenges we had to solve for this architecture:

1. As there is no start token like in NLP, we needed another first value for the decoeer input.
2. The output of the encoder has not the same number of dimensions as the decoder and cannot be used directly in the cross-attention block

The first problem we solved by using the last values of the time series of the encoder input as first values of the decoder input. For the adjustment of the feature dimensions of the encoder output, we used a simple fully connected layer to create the Q values for the cross-attention block. The whole architecture is shown in fig. 6.16.

Figure 6.16.: Timer Series Transformer Architecture



After a while, we dicarded this approach, as all resulting models only predicted zero change after the training conveged.

6.2.5. Validation

Logging Implementation

In our logging implementation, we drew inspiration from the logger class that Marco kindly provided. The logger includes important features that improve project monitoring and transparency. Leveraging the summary writer class from TensorBoard in the background, our logger provides a robust foundation for tracking and visualizing essential metrics. To improve the user experience during training, we incorporated the tqdm library and utilized its progress bar feature for loops, which is particularly beneficial in our training loop. The tqdm library's write method ensures a clean display of information without interrupting the progress bar, eliminating the shortcomings of native Python print statements.

To enhance console print clarity, we include "[LOGGER]" as a prefix, along with the name of the code file that calls the logger. This approach offers users understanding of the code's current execution point. Our logging system goes beyond textual logs. During validation steps, it dynamically generates and saves prediction charts to TensorBoard. In summary,

our logging system enables a clean and transparent textual log in the console, providing real-time insight into ongoing processes. Additionally, it generates a TensorBoard file, offering comprehensive information and visualizations to understand and analyze the entire training process thoroughly.

Loss Functions

In our search for effective loss functions for training and validation, we implemented a variety of errors to determine their performance. Mean Squared Error (MSE) was one of the main choices:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{6.1}$$

Another option was the Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{6.2}$$

We also tried the Root Mean Squared Logarithmic Error (RMSLE):

$$RMSLE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(y_i) - \log(\hat{y}_i))^2} \tag{6.3}$$

Furthermore, we experimented with the Exponential Mean Squared Error (ExpMSE):

$$ExpMSE = \exp\left(\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2\right) \tag{6.4}$$

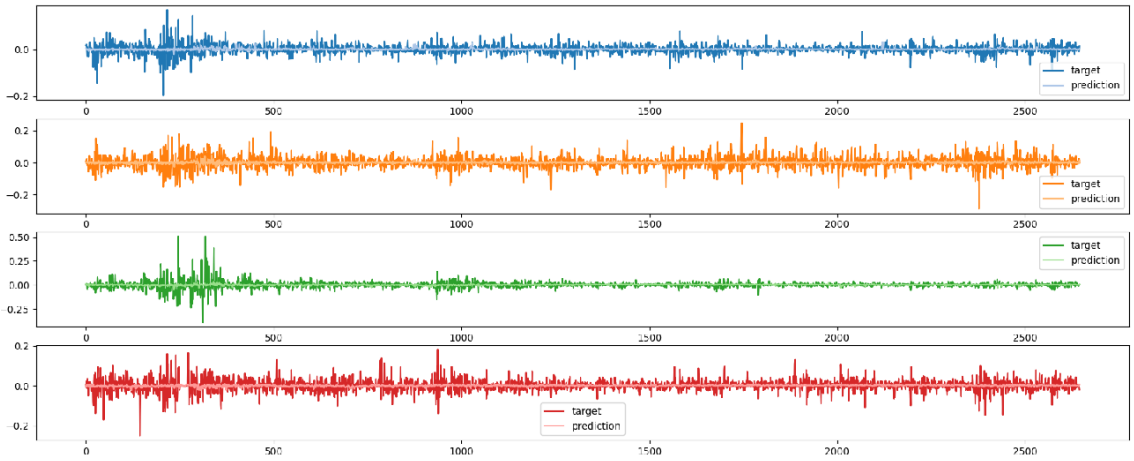
After much experimentation, the consensus was to use MSE and RMSE as the primary measures. RMSE, known to be less sensitive to outliers, proved valuable in scenarios where extreme values could affect model performance. However, MSE ultimately gave the best results, balancing sensitivity to outliers with overall predictive accuracy. Its ability to penalise outliers more severely than RMSE also proved beneficial in mitigating the impact of extreme data points.

Visualisation

To get deeper insights into the results of the model, we used multiple visualisations, as the pure observation of the loss could not give a clear indication of what the predictions look like. These visualisations were generated during the training for the train and validation set and could be inspected in the tensorboard software.

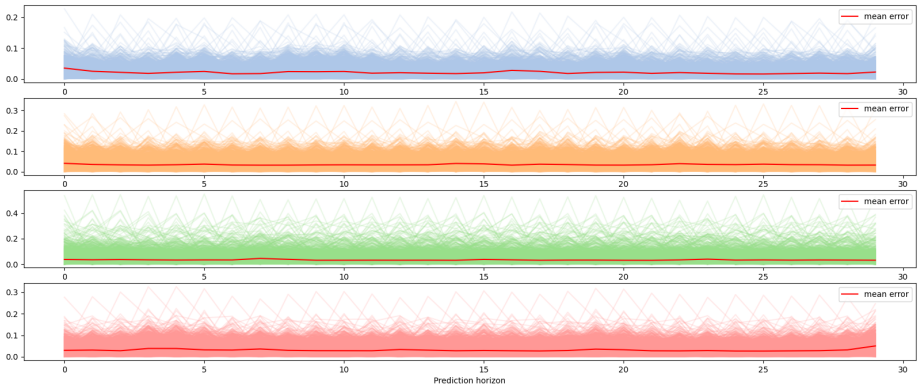
The first visualisation we implemented was the plot of the differenced predictions against the target for all output features. As we predicted for each sample a series of value, we visualised only the prediction one step ahead.

Figure 6.17.: Logging image 1: Plot of differenciated predictions vs. targets.



Secondly, we needed a plot to get an overview of what the predictions further then one step ahead look like. For that, we plotted the absolute errors for each sample as a separate line into a run chart for each feature. On the x-axis is not the number of samples, but the prediction horizon i.e. the steps ahead the model predicts. The mean of all these samples, the mean absolute error, is plotted as a red line along the prediction horizon.

Figure 6.18.: Logging image 2: Plot of absolute errors along the prediction horizon.



Finally, we implemented a plot of the absolute predictions, as the final goal is to predict the direction in which a stock price develops over time and not the exact growth rate for each time step. For example, this could visualise if the model predicts a long time direction but not local volatility. For this, the differenced values are being cumulated over all samples starting with 100, so that each point on the curve is the relative price in percent compared to the start of the train or test set. These results are plotted for the actuals and for the predictions one step, 14 steps and 29 steps ahead.

Figure 6.19.: Logging image 3: Plot of absolute predictions.

6.3. Conclusion

To summarise, our work has resulted in the successful creation of a practical library capable of training a variety of deep learning models for time series forecasting. A key advantage is the flexibility of the training process, enabled by our simple configuration file system. This flexibility enables users to conduct numerous experiments efficiently, fostering a quick and iterative approach to model development. The library's high degree of configurability, driven by the use of configuration files, also accommodates various customizations to input data, allowing for versatility in addressing different use cases.

The logging system enhances the training process by providing insights, transparency, and aiding in result interpretation. The library's extensible design allows for easy modification and expansion, and the project maintains clarity through well-documented code and comprehensive guidelines for contributors. In addition, our adherence to coding standards, such

as typing conventions, reflects our commitment to quality. The Transformers group invested time in acquiring domain knowledge, which we implemented during the development and model training. Throughout the project, team members learned collaboratively, addressing each other's weaknesses and fostering a cooperative environment.

7

Model: Statistical and Machine Learning-Based Time Series Analysis

The following will outline the Time Series Analysis- Group's work.

7.1. Internal Group Communication and Organization

In addition to the communication and organization of the entire group and between the individual groups, an internal group communication and organization structure was essential. The centerpiece of internal group communication was the weekly meeting that took place every Friday. The aim of this meeting was to discuss the current statuses of the individual group members and to support each other in facing challenges simultaneously. Additionally, the next steps for the group were discussed during this meeting, where task allocation was determined. At the same time, deadlines were set for the individual tasks, which aimed to give a certain commitment to the task allocation. In support of this conversation, a type of Scrum board was created using Notion, where a common backlog was created and regularly maintained. This was meant to make it immediately apparent which tasks had been completed, which tasks were pending, and which tasks were currently being addressed and when these tasks were due. On Notion, interesting and helpful articles and publications were also shared among the group members to improve knowledge transfer. Additionally, this facilitated the introduction to the various topics. Besides the weekly meetings, there were also appointments as needed. These appointments, for example, took place before presentations to discuss the content and design of the slides. Additionally, these extraordinary meetings also occurred when there were acute challenges. In such cases, a meeting was spontaneously arranged to tackle the challenges together. Such discussions were usually attended by only those group members who had found time spontaneously. At these meetings,

primarily code-related challenges were addressed, and coding was done together. Especially considering that the team members did not pursue the classical statistical methods after creating the MVP, such meetings occurred where Ms. Jäkle, who was primarily responsible for implementing the machine learning approaches, and Mr. Fedaie and Mr. Stelke, who were mainly responsible for implementing the statistical models, participated. Reasons why the implementation of the statistical models was not pursued after the MVP will be explained later in the project report. The focus of the meetings between Ms. Jäkle, Mr. Stelke, and Mr. Fedaie was on solving challenges, with one meeting involving several hours of debugging and providing feedback. Here, Mr. Fedaie and Mr. Stelke tried to give tips and suggestions to further optimize the code. In addition to the conversations, communication also took place via chat, with particular use of WhatsApp and Teams. The chats primarily involved coordinating the discussions and work distributions, and plots, documents, and code snippets were shared in the chats for subsequent discussion. Overall, the group's communication and organization were designed to work efficiently, learn from each other, and do so in the fastest way possible.

7.2. Statistical Models

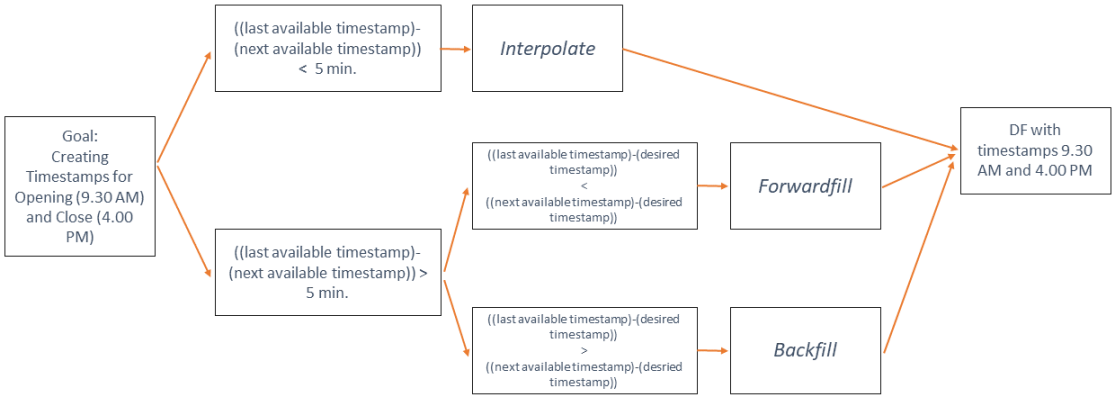
This chapter aims to present how the group implemented the statistical models. The process starts with describing the data preprocessing steps, followed by an in-depth discussion on the specific models implemented. The initial phase involved implementing relatively simple models such as Naive Forecasts, Window Average models, etc., which were intended to serve as baseline models. This was followed by the development of more complex statistical models, including ARIMA, Theta, etc. Alongside the statistical models, Deep Learning approaches were also explored, notably n-Beats and n-HiTs. The implementation of the preprocessing is outlined first.

7.2.1. Preprocessing

As already outlined in the previous chapters of the project report, the timestamps in the provided stock data have irregular intervals. Since the statistical models to be implemented require regular time intervals, preprocessing was necessary. This should ultimately result in a dataframe with uniform time intervals. In addition to the requirement for uniform time

intervals, the implemented statistical models are mostly applied for predictions on a daily basis at most, as making predictions on a minute basis is very challenging with the statistical models implemented and would also significantly limit the meaningfulness of the predictions. For example, one could cite the Naive Forecasts to be discussed later. These assume that the best prediction for the next timestamp is the previous one. Making such a prediction on a minute basis is very limited in its meaningfulness. Applying it on a daily basis already makes more sense. To implement the preprocessing with the requirements outlined above, the following approach and logic were implemented. The goal of the preprocessing was to create a dataframe that only contains the stock opening price at 9:30 AM and the closing price at 4:00 PM. The off-exchange trading, which is also captured in the data, was not to be considered in this process, and this approach made predictions on a daily basis possible, while also ensuring that the timestamps in the dataframe have uniform time intervals. The issue at this point was that often there were no data for the timestamps at 9:30 AM or 4:00 PM, so these data had to be filled in using a defined logic. This logic was implemented as follows. First, the time difference between the last available timestamp before the market opening or closing and the next available timestamp was considered. If this difference was less than five minutes, the missing timestamp for the market close/open was determined by interpolation. The reason for this approach is that with a time difference of under five minutes, no large price jumps are expected, which would mean that an interpolation would inadequately represent the price movement within five minutes. If the time gap between the last and next available timestamps was greater than five minutes, the data for the market close or opening were generated using a back or forward fill. If the last available timestamp was closer to the desired timestamp at 9:30 AM or 4:00 PM, a forward fill was performed. If the next available timestamp was closer, a backfill was performed. No interpolation was applied at these two points, as a time difference of five minutes at the stock exchanges can lead to such large fluctuations that an interpolation would not generate an adequate representation. Ultimately, this implemented logic resulted in a dataframe that only contained the timestamps for the market opening and closing. This approach is additionally illustrated schematically in the figure below.

Figure 7.1.: Preprocessing



7.2.2. Implementation of the Statistical Models

After the preprocessing has been presented, this section will explain how the statistical models have been implemented. For the implementation of the statistical models, the Nixtla library was utilized, which contains all the statistical models that were intended to be implemented. Nixtla impressed with its intuitive usage, comprehensive documentation, fast runtime, and the numerous options available, eliminating the need for additional libraries for each statistical model. The implemented statistical models can be divided into three main categories. These are the baseline forecasts, ARIMA, deep learning approaches, and other approaches, which will be summarized under ”other approaches” in this project report. First, the procedure for implementing the baseline forecasts will be outlined.

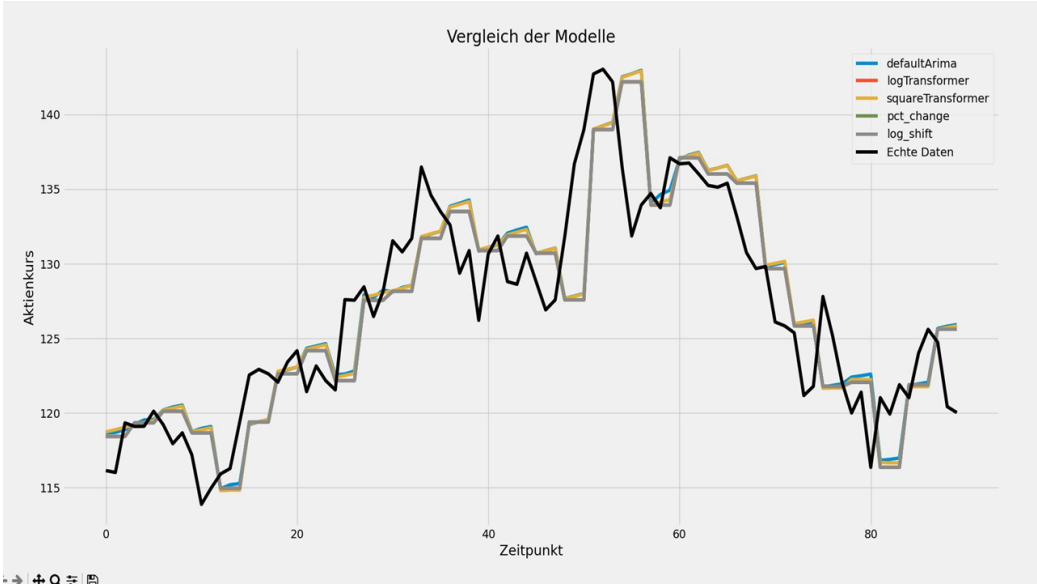
Implementation of Baseline Forecasts

For the baseline forecasts, three different models were created. Firstly, there is a Historic-Average model, which simply takes the average of the entire historical course and uses this value as the prediction. Secondly, a Naive model was created, which assumes that the best prediction for the next timestamp is the last timestamp from the past. Lastly, a Window-Average model was created, which takes the average from a time window of a given period in the past and uses this value as the prediction for the future. The primary goal of creating these models was not to expect particularly high accuracy from them. However, they were intended to serve as a reference and quality benchmark for the subsequent models.

Implementation of ARIMA

A model that is already more promising for predicting stock prices is the so-called ARIMA model. An ARIMA model, short for "Autoregressive Integrated Moving Average," is an advanced statistical analysis method for predicting and analyzing time series data. It combines autoregressive elements - hence the AR in ARIMA - which use past values of the series, with moving average components - hence the MA in ARIMA - that consider past errors, and integrates these. As a result, ARIMA is particularly well-suited for modeling and forecasting time series that exhibit complex patterns such as trends and seasonal fluctuations. [17] A prerequisite for the application of ARIMA models is the presence of stationary data. Since stock prices are inherently non-stationary, these data had to be transformed to ensure that stationary data were present. The simplest method is to integrate the time series. The advantage here is that the AutoARIMA method of the Nixtla library automatically finds the correct degree of integration and subsequently back-transforms the data, so the predictions can be immediately interpreted. However, this method is often not the optimal solution, and the data often remain non-stationary, or only insufficiently stationary, after integration. For this reason, other transformations were also tested. These included a transformation where the logarithm function was applied to the data, a transformation where the data were squared, a transformation that merely considered the percentage change in the price and then trained an ARIMA model on it, and a "shift-log transformation," which applies the logarithm function to shifted data. This approach made the implementation significantly more complex, as it made the predictions of the ARIMA model not directly interpretable. Instead, the predicted values had to be back-transformed to make them interpretable again.

Figure 7.2.: ARIMA-Outputs for Apple Stock as an example



The graph above, using the example of Apple stock, shows that the different transformations only made relatively small differences in the forecasts. It is important to note from the graph that the models only predict two days ahead and are then refitted. In general, it can be said about the ARIMA model that these models can capture basic fluctuations and even more complex patterns and can create reliable forecasts based on these. However, stock prices sometimes do not show any concrete patterns, which is where ARIMA models reach their limits. Especially considering that the other models in the group were significantly more performant, pursuing this approach was suspended after the creation of the MVP. For the MVP however an ARIMA-Model was created for every stock individually, these Models were saved as a pickle file and were eventually used in the interface to generate prediction for a given time horizon. Those predicitons were then used in the frontend.

Implementation of Deep Learning Approaches

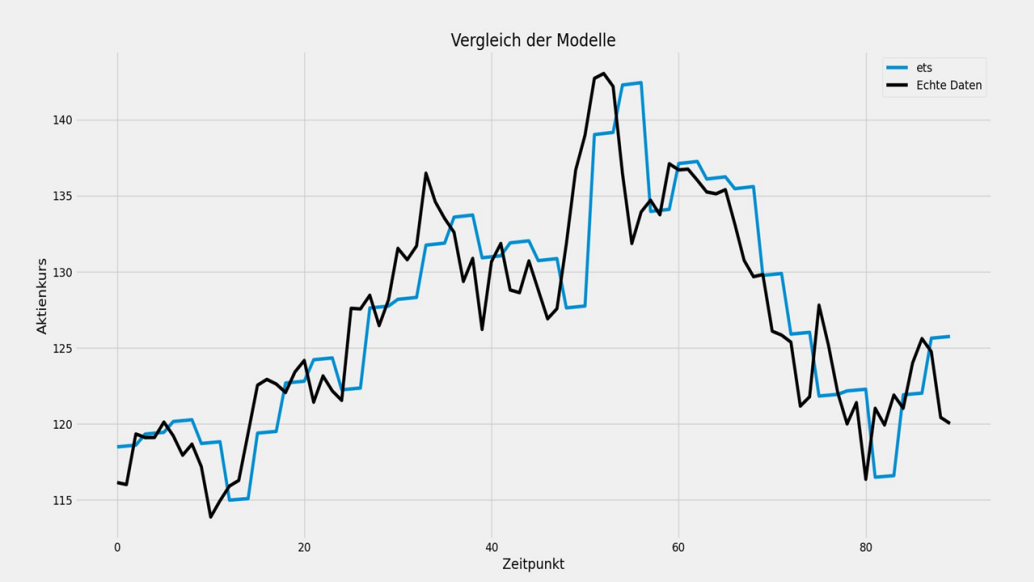
In addition to the ARIMA models, attempts were also made to predict stock prices using two deep learning approaches specifically developed for time series forecasting, namely n-Beats and n-HiTs, applying the Nixtla library once again. However, it quickly became evident that the forecasts from these models were not promising, as they performed worse than ARIMA and even worse than a simple naive forecast. Even after extensive parameter

tuning, there was no improvement in the results. Consequently, the Darts library, which also includes implementations of n-Beats and n-HiTs, was utilized, but the outcome was similarly disappointing. Therefore, this approach was discarded and not considered further in the development of the MVP.

Implementation of other Approaches

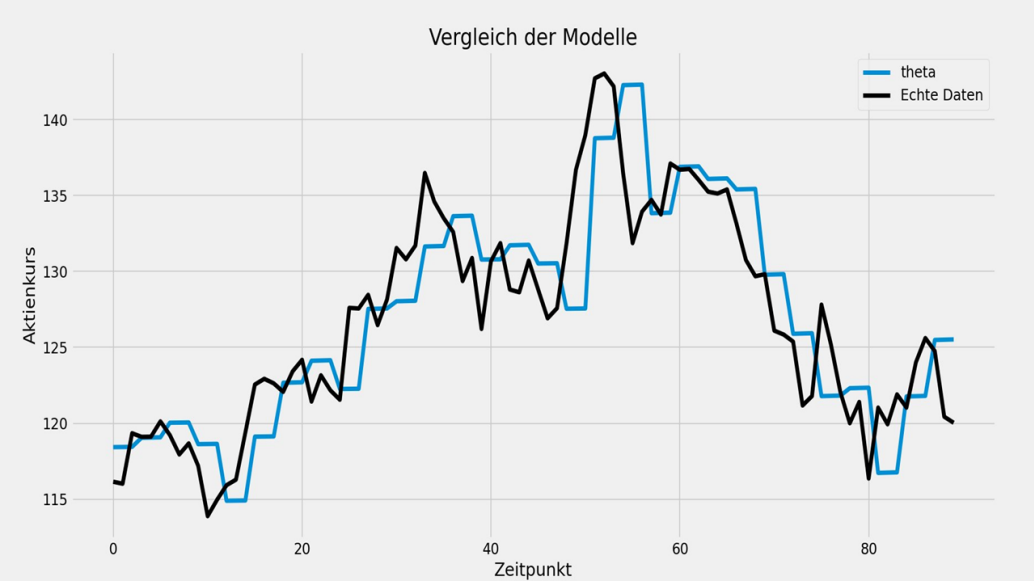
Among the other approaches implemented within the project are the ETS models, Theta models, and GARCH/ARCH models. Since the GARCH and ARCH models that were implemented early on showed that they could not produce reliable forecasts, they will not be further elaborated upon here and were therefore not considered in the creation of the MVP. Thus, attention can now be directly given to the ETS and Theta models. These were also implemented using the Nixtla library. ETS models, short for "Error, Trend, Seasonality," systematically model error, trend, and seasonality components within time series data. These models are based on exponential smoothing, where each component is treated specifically to improve the accuracy of the forecasts. ETS models can use smoothing parameters to forecast time series that exhibit clear trend or seasonal patterns.[9]

Figure 7.3.: ETS-Outputs for Apple Stock as an example



As shown above in the figure with the course of Apple's stock as an example, an ETS model cannot predict the course better than an ARIMA model. It should also be noted again that the models were updated every two days, yet it is still evident that the model is not capable of accurately reflecting the movements of the stock's course. Therefore, the ETS model was not included in the final prototype. In addition to the ETS model, a Theta model was also implemented. Theta models are an approach to time series forecasting that generates predictions through a specific treatment of the trend within the data. They are based on the decomposition of the time series into at least two components representing different characteristics of the time series, such as the trend and seasonal fluctuations.[10]

Figure 7.4.: Theta-Outputs for Apple Stock as an example



As can be seen in the graph above, this model too can be deemed unsuitable for predicting complex stock prices. This model will also not be considered further in the final prototype.

In summary, it can be stated that statistical models can only serve as a benchmark for the prediction of stock prices and do not generate sufficiently accurate forecasts.

7.3. Machine Learning Models with Feature Engineering

In this part of the Deep Learning project, the focus was on time series analysis, particularly in the area of feature engineering, as well as the development and optimization of machine learning models (ML-models).

7.3.1. Beginning to Machine Learning Models with Feature Engineering

At the beginning of the Deep Learning project, the initial task was to load, read, and gain a first overview of the data. In this process, I examined the fundamental characteristics of the data, such as the size of the dataset, the number of columns and rows, and various other aspects. Following this, I created simple plots to visually comprehend the data. This initial overview provided me with a general impression of what to expect in this project, without immediately delving into the complexity of the details.

Missing Data and Preprocessing

After gaining some insight into the data, I focused on missing data, examining the absence of data points. Initially, I looked closely at only one stock dataset, starting with Apple's dataset.

In preparing the dataset for my analysis, I first standardized it. I limited my attention to the daily trading hours of the New York Stock Exchange, which run from 9:30 AM to 4:00 PM. My focus was exclusively on weekdays, the days when stock market activities take place, which amounts to approximately 262 days per year.

Upon a closer examination and visualization of this data, it was noticed that data points or days were missing at certain times. On closer inspection, it turned out that these missing data corresponded to public holidays in the USA. These are stock exchange holidays when the market is closed and no trading takes place. Therefore, these missing data are not errors in the dataset, but reflect the actual trading breaks. To confirm this finding, additional stock datasets were analyzed, and here too, the regular absence of data on stock exchange holidays was observed.

Feature Engineerings

After completing the analysis of missing data, the focus shifted to my part of the work: feature engineering for time series. The initial step involved experimenting to discover which additional information could be extracted from the data.

To work with a complete dataset and avoid missing values or NaN, I performed a simple data preprocessing. Missing values were filled using a forward fill method. The precise alignment and refinement of data preprocessing will be conducted in a later phase, once it is clearly defined which specific models are required, particularly in terms of prediction models and their requirements. However, for this first pass, to identify features from the dataset through feature engineering, a simpler preprocessing is sufficient.

The first step was to integrate basic **DateTime-features**. These features utilize the intrinsic timestamp information of the datasets to extract temporal aspects such as the weekday, hour, or month. Such information can be particularly valuable in identifying potential seasonal patterns and trends that could be important for forecasts.

Following that, **Lag features** were developed, which use the time sequence of the data. A Lag feature represents the value from a previous point in time in the dataset, which is useful for understanding dependencies and the influence of past values on current forecasts. For example, a Lag feature could represent the previous day's closing price of a stock to analyze its influence on today's price.

In addition to Lag features, **Window features** were implemented. These features calculate statistical metrics over a defined period – a "window" in the time series. For instance, a Window feature could represent the average price of a stock over the last seven hours, the past 30 days, the last 12 months, or even just the last 5 minutes, which can smooth out fluctuations and highlight longer-term trends.

In developing the many different features, the objective was to integrate them into a pipeline. This structured approach allows for the seamless combination of various features and systematic processing through the pipeline. The design of this pipeline is conceived to be flexible, allowing for the selection of currently desired features and their efficient application to models. This ensures easy adaptability of feature selection and straightforward application of features to different models, significantly simplifying the process of model building and evaluation.

ML-Models

After creating various features and completing the feature pipeline, I turned my attention to the creation and evaluation of the ML-models. My approach started with the implementation of classic models such as Linear Regression, Random Forest, Gradient Boosting Machines, and Support Vector Machines.

- Linear Regression is a simple statistical model used to predict a dependent variable (here the closing price) based on one or more independent variables.

- Random Forest is an ensemble model consisting of many independent decision trees that deliver average predictions. Instead of relying on a single model, Random Forest utilizes the strengths of many trees to achieve more precise and robust results.

- Gradient Boosting Machines are also ensemble models, but they correct their errors sequentially through successive trees. Each tree in Gradient Boosting Machines is built in succession, attempting to correct the errors of its predecessor. This differs from Random Forest, which creates independent trees.

- Support Vector Machines are powerful models, especially used in classification tasks, but can also be applied to regression. Support Vector Regression thus attempts to find a function that best describes a given set of data points with a continuous target variable, aiming to minimize the error within a certain threshold. Unlike classification, where the focus is on the optimal separation of classes, Support Vector Regression aims to make the most precise prediction of a continuous value.

The focus of my initial prediction models was on predicting the daily closing price at 4 PM, from which I would forecast the prices of the following days.

Data splitt

Before I can develop predictive models, the data must be divided into training and testing sets. For the initial split, I have allocated the data in an 80-20 ratio, with 80% of the data used for training and the remaining 20% for testing the model. However, this is not yet the final division of the data.

A particularly crucial detail in this division is the maintenance of the temporal order of the data. This means that the data is not split randomly but in chronological sequence. By using the temporal order, it is ensured that the training set contains data from earlier periods, while the test set consists of subsequent data.

After splitting the data into training and testing sets, they are processed through the previously designed pipeline. This process ensured that both the training and testing data were enriched with the features developed during the feature engineering phase. This methodical approach guarantees that the data are consistently prepared and that the models can be trained and tested on a solid foundation of features.

In a further step, the training and testing data are divided into dependent and independent variables, which is crucial for training the ML-model. This division precisely defines which data serve as input (independent variables) and which as output (dependent variables). The independent variables act as input data used to predict the target variables. The dependent variables, also known as target variables, represent the outcomes or outputs of the model that are to be predicted.

Subsequently, the training data is used to train the corresponding model. Once training is complete, the model is tested with the test dataset.

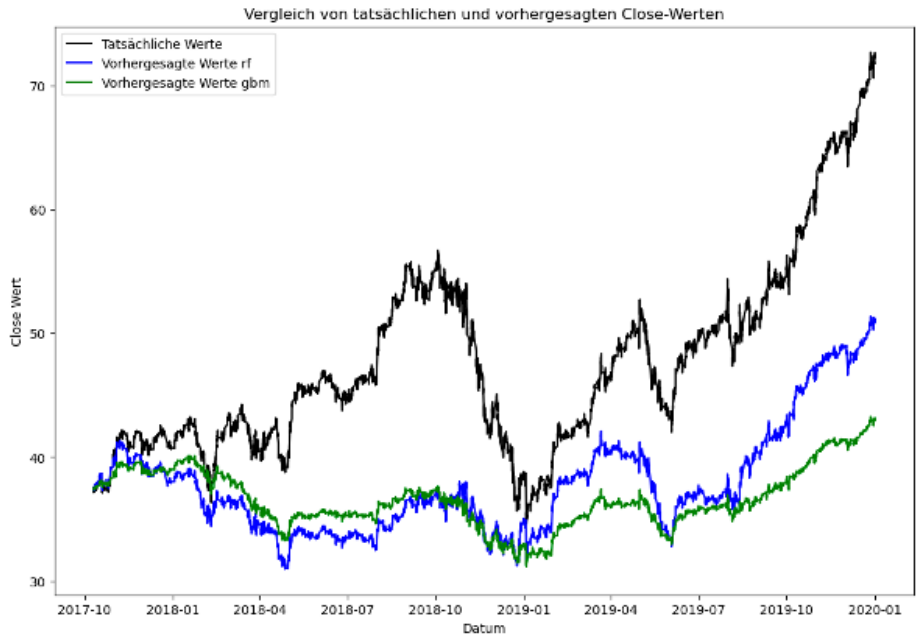


Figure 7.5.: RF and GBM Models prediction

Referring to the first result in Image 7.5, the prediction results mistakenly mirrored the swings of the actual values closely. These favorable outcomes, however, were the result of an error in data processing: the Open, High, and Low Value variables were inadvertently not removed, despite the model's focus being solely on the Closing values. The inclusion of these additional variables resulted in an overly optimistic skew in the predictions, as they inadvertently captured the same stock movements.

After correcting this error, the model displayed more realistic results (see Image 7.6), and compared to the initial assessments, it was less good. This correction led to a more accurate representation and brought forth interesting insights. Despite the initial overestimation, the predictions made by the classical ML-models were considered good for the first iteration. This adjustment allowed for a deeper understanding of the data and its dynamics.

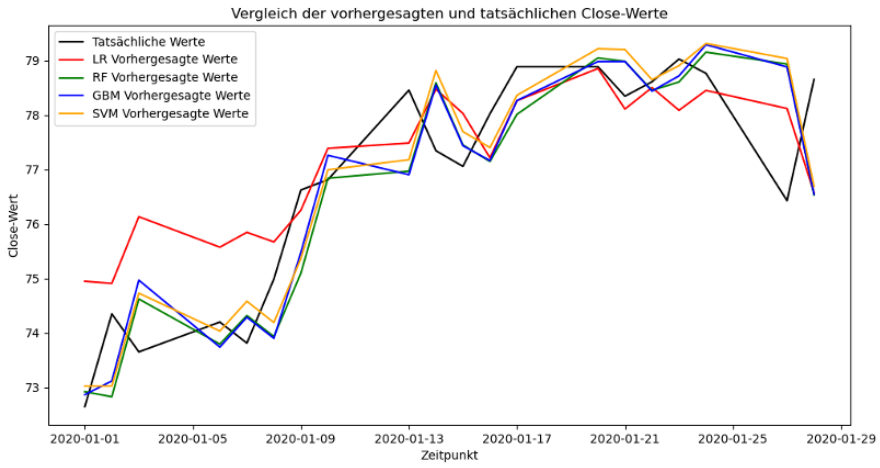


Figure 7.6.: ML-Models prediction

Adding Percent Change Features

To enhance the performance of the models and facilitate their learning of daily variations, an additional feature, the percent change, was introduced. This feature was calculated by determining the difference in the closing value from the previous row and then computing the percentage change based on that difference. The rationale behind using this percent-change feature is to enable the prediction of the next growth in the closing value.

After the creation of this new feature, it was integrated into the existing pipeline. The data was then split accordingly and processed through the pipeline. The newly trained ML-model now focused on predicting the percent-change values of the closing price. The predictions of these percent-change values were subsequently back-transformed to determine and analyze the forecasted stock prices.

The final outcome of this method proved to be very promising, as it yielded good results.

Prediction Sequential Adjustment

In the next step, the models prediction function was adjusted to make forecasts over several days. An iterative prediction procedure was implemented: each forecast builds upon the previous one, allowing the model to autonomously generate subsequent predictions from known data up to a certain Day X, and incorporate these into further analysis.

An example of this process would be: the model makes a prediction for Day X. The forecasted value for Day X is then saved and factored into the calculation for the prediction of Day X+1. Thus, the forecast for Day X+1 includes both the original data up to Day X and the predicted value for Day X. This process is continuously repeated, with each new forecast building on the previous one, enabling dynamic and continuous analysis of future values.

Grid Search

The final critical step for the ML-models is selecting and setting the right hyperparameters. To optimize the performance of the ML-model, the Grid Search method was employed to identify the best hyperparameters. This method allows for a systematic search and evaluation of various combinations of hyperparameters. Different values are tried for each hyperparameter. In the end, Grid Search provides the optimal combination of hyperparameters that maximizes the model's performance.

Completing the first final model

Following the successful completion of the first final prediction model, which is now operational and equipped with the optimal hyperparameters, the model is being prepared for deployment. This preparation involves ensuring that the model is clean and complete, both in terms of model structure and the results achieved.

As part of these preparations, the associated code was revised. This revision included reorganizing and cleaning up various components, including preprocessing, feature creation, the pipeline, data partitioning, and the scripts for the ML-models.

The first final model is designed as a daily forecasting model capable of predicting closing values for the upcoming days. During the preprocessing stage, the data for the ML-model was prepared with particular attention to the closing prices at 4 PM of each trading day. To exclude weekend irregularities, only business days were considered. Any potential gaps in the data were addressed using a forward-fill method.

During the feature engineering phase, features relevant to the final model were selectively chosen. These included datetime features, the percent-change feature, and a lag feature covering the last 20 days (equivalent to one business month, as 20 days equals 4 weeks multiplied by 5 business days). The pipeline was then constructed from these features, ensuring standardized data processing.

A different method was chosen for splitting the data into training and testing sets, moving away from the previous 80-20 percent distribution. Instead, a specific date, January 3, 2021, was set as the dividing point. All data up to this date were used as training data, while testing data began from January 4, 2021.

To ensure the correct application of the pipeline and the adequate creation of the lag features over 20 days, the test data was expanded to include the period of 20 days before the separation date. This ensured that when running through the pipeline, the lag features were accurately formed up to the cut-off date of January 4, 2021. This means that after splitting the data into training and test sets and processing through the pipeline, the training data extends up to and including January 3, 2021. The test data begins from January 4, 2021. Thus, the model is capable of making predictions from January 4, 2021, based on the knowledge and data available up to January 3, 2021.

Before the models can be trained, a renewed search for the optimal hyperparameters is conducted using Grid Search to ensure the best possible performance of the model. Only after this step are the models finally trained. This involves developing models for Linear Regression, Random Forest, Gradient Boosting Machines, and Support Vector Regression.

Following the development and training of the ML models, sequential forecasts were carried out. The four models yielded good results, with the Random Forest and Gradient Boosting Machines models, in particular, providing the most accurate predictions.

Models saved and abstract class created

The development process for the daily forecasting model has been completed, and the code is now ready. In preparation for the upcoming MVP (Minimum Viable Product) milestone, a few additional steps have been taken. Initially, an abstract class was created to ensure a consistent structure across all models. This required adapting the existing code for the ML-models to be compatible with the abstract class.

Furthermore, the trained models were saved as .pkl files to facilitate their integration into the backend system. This step significantly simplifies the use of the models as they do not need to be regenerated with each use. Finally, an API was developed to allow the frontend to access the models.

Next Steps

Following the completion of the MVP phase, we are now entering the final stretch, focusing on the development of three different prediction models. For the final product, three specific models have been established, each targeting different prediction intervals:

1. A Daily model, providing daily forecasts for a period of up to 30 days.

2. A Two-Hourly model, generating predictions every two hours.

3. A Minutely model, which creates forecasts every 20 minutes.

Each of these models is designed to meet distinct requirements and time horizons, offering a comprehensive range of predictions.

7.3.2. Daily Model

The Daily model, which has already been developed, required only minor adjustments to fit the final product. Additionally, the model was finalized for each stock, secured, and saved as a .pkl file. Once these steps were completed for the Daily model, the work shifted focus to the development of the other two prediction models.

7.3.3. Two-Hourly Model

The Two-Hourly model, which delivers a prediction every two hours, required more comprehensive considerations and adjustments, starting with the preprocessing, selecting necessary features from feature engineering, refining the pipeline, splitting the data, and retraining models. The Two-Hourly model is designed to generate forecasts based on an hourly dataset and deliver them in a bi-hourly cadence. This means it produces forecast values for the subsequent hours at two-hour intervals.

Data preprocessing

In the preprocessing step for the Two-Hourly model, specific adjustments were necessary that went beyond the simple carryover of the closing value at 4 PM, as was the case with the Daily model. It was crucial for the hourly dataset to reflect the stock market's opening hours from 9:30 AM to 4:00 PM. Consequently, the data was filtered to consider only the values at the full hours — 9:30, 10:00, 11:00, 12:00, 13:00, 14:00, 15:00, and 16:00. This approach ensured that the dataset was precisely prepared for the hourly prediction.

Features & Pipeline

In the feature engineering process for the Two-Hourly model, the DateTime features and the percent-change feature were used once again. However, in contrast to the 20-day retrospective utilized for the lag features in the Daily models, the Two-Hourly model considered the last seven hours. This means that each row of the dataset additionally contains information

about the past seven hours. This approach was chosen because the trading day on the stock market lasts from 9:30 AM to 4:00 PM, which corresponds to approximately seven hours, the lag features of seven hours represent a complete trading day. After the creation of the selected features, the pipeline was adjusted accordingly.

Data splitt

Similar to the Daily model, when adjusting the split between training and test data for the Two-Hourly model, an appropriate amount of historical data had to be included to generate the lag features. Therefore, an additional seven hours of historical information was incorporated for the test data. This step is crucial to ensure that when the test data is processed through the pipeline and the lag features are being generated, the split is exact and consistent with the dates. This ensures that the test data contains a complete set of historical features without any null values. According to the implemented pipeline, the training data extends up to January 3, 2021, at 4 PM and the test data begins on January 4, 2021, starting at 9:30 AM.

Training models

The Two-Hourly Prediction model, which is intended to generate forecasts for every other hour, has been meticulously prepared and accordingly adjusted. The prediction code was also revised to ensure that the forecasts are delivered in the desired two-hour intervals and do not produce hourly predictions.

Before the models underwent final training, another grid search was conducted to finely tune the hyperparameters. Following the adjustment and optimization of the hyperparameters, the models were trained. Finally, the completed models were saved as .pkl files to secure the achieved results and make them available for future applications.

7.3.4. Minutely Model

The third and final predictive model, the Minutely model, which is based on minute-level data and designed to create forecasts every 20 minutes, required a further fundamental redesign. Decisions had to be made regarding the type of preprocessing to apply, which features from feature engineering to adopt and create, the adjustment of the pipeline, and how the data should be split.

The Minutely model is trained with minute-by-minute collected data and is engineered to generate forecasts at 20-minute intervals.

Data preprocessing

In the preprocessing for the Minutely model, the dataset was brought to a minute-level resolution to create a continuous data series from the start to the end of the selected time period. Subsequently, the dataset was restricted to the hours between 9:30 AM and 4:00 PM, reflecting the official stock market trading hours, consistent with the other prediction models. To ensure that there are no data gaps within the minute resolution, a forward-fill method was also applied to fill any potential empty time intervals.

Features & Pipeline

For the feature selection of the Minutely model, as with the other models, the datetime feature and the percent-change feature were utilized. In the design of the Lag feature, due to the minutely data basis and the 20-minute forecast intervals, a more extensive Lag range of 60 minutes was integrated. This allows for the consideration of three complete 20-minute segments in each forecast, which means the Lag feature encompasses a total of 60 time intervals. Following the selection and creation of the relevant features, the pipeline was adjusted accordingly.

Data splitt

In adjusting the data split for the Minutely model, it was also crucial to partition the data so that, after processing through the pipeline, they are complete and begin or end on the correct date. While the training data were prepared as usual, it was necessary for the test data to include an additional last 60 minutes of historical data so the pipeline could effectively utilize these data. The outcome was a training dataset that extends up to January 3, 2021, at 4 PM, and a test dataset that begins on January 4, 2021, at 9:30 AM, including the complete 60 minutes of Lag features.

Training models

The next step in the process involved training the models. However, before this, a grid search was conducted to determine the optimal hyperparameters for the model. Subsequently, the prediction function was adjusted to ensure that the model generates forecasts in 20-minute intervals, instead of producing minute-by-minute predictions. After this modification, the models were finalized as usual, trained, and then saved as .pkl files for further use.

Interface and API

After the completion of the models, the code was prepared for integration into the interface. This included adjusting the code to ensure compatibility of the abstract class with all three prediction models. Furthermore, the API was modified to allow the frontend and backend to retrieve the models.

Final Product

For the final product, however, only the Random Forest and Gradient Boosting Machines models were selected. This decision was based on insights gained during the MVP phase, which showed that these two ensemble methods yielded the best results. The Linear Regression and Support Vector Regression models were not included in the final product due to their lower performance.

7.4. Long Short-Term Memory (LSTM)

7.4.1. Introduction

At the heart of modern time series analysis and sequential data processing lie advanced neural network architectures like LSTM. They were developed to overcome the vanishing gradient problem and also enable capturing long-term dependencies in data by utilizing specialized memory cells and gates (input, output, and forget gates). In contrast, GRU networks offer a simplified yet effective alternative to LSTMs by reducing the number of gates, resulting in a less complex model that is still capable of learning long-term dependencies. SimpleRNNs, the simplest form of recurrent networks, though intuitive and straightforward in their approach, hit limitations with long sequences due to the vanishing or exploding gradient problem. With the help of TensorFlow and the high-level Keras API, creating an LSTM is more accessible.

7.4.2. Model Structure and Architecture

The developed LSTM model consists of several layers to maintain the balance between learning capability and generalization. The architecture is composed as follows:

Input Layer:
The input layer is responsible for feeding the time series data into the model. The dimension of this layer is determined by the number of features (e.g., historical prices, volume, technical indicators) and the length of the time window over which predictions are to be made.

LSTM Layers:
Following the input layer, several LSTM layers are added. For this project, two LSTM layers with 128 and 64 neurons were chosen. This configuration was considered a good compromise between model complexity and avoiding overfitting.

Dense Layers:
After the LSTM layers, dense layers (Fully Connected Layers) follow, which serve to condense the information obtained from the LSTM layers and generate the final prediction. A dense layer with 32 neurons was used, followed by an output layer with a single neuron representing the prediction of the next stock price (or price change).

Activation Functions:
Within the LSTM and dense layers, the standard tanh (Hyperbolic Tangent) activation function was used. Models were also trained with the ReLU (Rectified Linear Unit) activation function and compared with the sigmoid activation function. Based on a scientific article on LSTM stock price prediction, it was confirmed that tanh is best suited for this application. For the output layer, a linear activation function was chosen since the goal is continuous prediction. (c.f. Determining the best activation functions for predicting stock prices, H.M. Sami)

Experimenting with other RNNs:
Exploring various recurrent neural network (RNN) architectures, including BiLSTM (Bidirectional Long Short-Term Memory), GRUs (Gated Recurrent Units), simpleRNNs, and combinations thereof, offered a new approach to enhance the results and to learn the behaviour of the models. BiLSTMs extend traditional LSTMs by processing data in both directions, capturing complex dependencies.

7.4.3. Rationale for Decisions Made

The decision for the specific model architecture and layer configuration was based on several considerations:

Avoidance of Overfitting: By selecting a moderate number of neurons and layers, efforts were made to avoid overfitting. After experimenting with multiple LSTM layers and significantly more neurons, the number of hyperparameters was adjusted due to overfitting.

Efficiency and Computational Resources: Additionally, the architecture should be efficient enough to be trained with the available free computing resources on Google Colab and a MacBook without compromising model performance. Particularly, the HPT without Keras took a lot of time, as exploratory application of various combinations of hyperparameters was necessary.

7.4.4. Hyperparameter Tuning

Basics of Hyperparameter Tuning Hyperparameters are the configuration settings that are set before the training process and influence the structure of the model as well as the training process itself. Unlike model parameters, which are adjusted automatically during training, tuning hyperparameters requires an external strategy.

Manual vs. Automated Hyperparameter Tuning (HPT) To save time in tuning the network, after a rough narrowing down and playing with the HP, traditional manual tuning using the Keras Tuners was employed, which unfortunately was discovered only at a later stage due to lack of experience with TensorFlow and ML.

Keras Callbacks In the TensorFlow Keras ecosystem, callbacks have proven invaluable for streamlining the training process, enhancing efficiency, and improving model performance. Key callbacks such as ModelCheckpoint allow for the automatic saving of models at various stages of training, ensuring that progress is not lost and that the best performing model is retained. EarlyStopping prevents overfitting by halting training when a monitored metric ceases to improve, optimizing both time and computational resources. Unfortunately not all Callbacks have been implemented since it is a wide topic. I.e. the ReduceLROnPlateau callback adjusts the learning rate dynamically based on performance, helping to fine-tune models for better accuracy. Collectively, these callbacks not only simplify the model training workflow but also significantly contribute to achieving optimal model outcomes with minimal manual intervention. Also TensorBoard from tensorflow, an interactive Web-Layout to improve HP and see the results in dynamic live plots, can help a lot in ML.

Application of Keras Tuner in the Project The Keras Tuner is a powerful tool for automated hyperparameter tuning. It allows systematic exploration of the parameter space through methods such as Random Search, Hyperband, and Bayesian Optimization to find the best possible configuration for a given model. In the project, the Keras Tuner was used to optimize the hyperparameters for the LSTM model, including the number of layers, neurons per layer, and learning rate. Through automated search, an optimal configuration was identified, leading to a significant improvement in model performance.

Advantages of Automated Tuning The main advantage of using the Keras Tuner lies in significant time savings and objective, data-driven decision making. Additionally, it enables more effective exploration of complex hyperparameter spaces, leading to better model configurations.

Challenges and Limitations Despite its advantages, the use of the Keras Tuner also brings challenges. These include the need for extensive computational resources and the potential risk of overfitting if the model is too closely tailored to the training dataset.

7.4.5. Univariate vs. Multivariate LSTM

The decision between univariate and multivariate LSTM models is very important for prediction accuracy in projects with diverse time horizons, ranging from minute-level forecasts for day trading to daily forecasts for long-term investments. Univariate LSTM is appreciated for its simplicity and effectiveness in focusing on a single variable, while multivariate LSTM allows for a more comprehensive data analysis and greater flexibility in prediction by incorporating multiple variables. However, a significant limitation of multivariate models is that they can only predict a single future point in time. To leverage all input variables for an improved forecast of a specific feature, additional features would need to be predicted, increasing model complexity and the risk of overfitting, potentially reducing reliability. Therefore, utilizing multivariate models requires careful consideration of complexity versus benefit, especially when taking into account different forecast horizons and data availability. Due to these reasons, only univariate LSTMs were finalized, as the challenges associated with multivariate LSTMs could not be adequately addressed.

7.4.6. Benefits of Jupyter Notebooks in Development

Jupyter Notebooks have proven to be a helpful tool in learning and developing the model. The ability to execute code in separate cells facilitates rapid iterations and experiments with different model configurations, preprocessing steps, and algorithms. Additionally, the combination of code, results, and Markdown notes in a notebook was very helpful in maintaining an overview in this multifaceted subject.

7.4.7. Utilization of Google Colab

Google Colab expands the capabilities of Jupyter Notebooks by providing a powerful, cloud-based environment specifically designed for machine learning and data analysis. Especially the access to powerful resources that Colab provides like free access to GPUs and TPUs, significantly reduced training time for the LSTMs. Moreover, since most Coursera machine learning courses used Colab or Jupyter Notebooks, this was a familiar development environment.

7.4.8. Learning process

Data Leakage
An unexpectedly high performance level of a model suggested the occurrence of data leakage, a condition where information from the test dataset is accidentally utilized during the training process. This scenario often results in overestimated performance metrics that do not accurately reflect the model's true capability to make predictions on unknown data. To remedy this problem, the data pipeline was carefully reviewed to ensure a strict separation between training and test data. Such corrections led to more realistic performance evaluations and contributed to the development of a more robust model.

Predicting Price Changes Instead of Absolute Prices
Focusing on predicting actual stock prices initially led to difficulties in interpreting the outcomes due to the influence of numerous external factors on stock prices. A shift towards predicting price changes rather than absolute prices proved to be a more effective approach, as it better captures market volatility and dynamics. This adjustment enhanced the predictive accuracy and applicability of the model for trading strategies.

Overfitting

Overfitting represents a frequent challenge in the training of machine learning models, where a model becomes excessively tailored to the training data, diminishing its ability to generalize to new data. Techniques such as the introduction of dropout layers and adjustments to the model's complexity were implemented to counteract overfitting. Cross-validation was also applied to more reliably evaluate model performance and enhance generalization capabilities.

Incorrect Application of Pre-trained Models

The improper use of the predict method led to powerful models failing to deliver satisfactory results. This issue was addressed by a thorough revision and adjustment of the application code. Corrections in the implementation of the predict method ensured that the models could unleash their full performance potential, resulting in significantly better predictions.

7.4.9. Common Misconceptions and Pitfalls

Common misconceptions and errors frequently encountered online regarding LSTM models, especially in resources such as Medium articles, Stack Overflow, YouTube, and GitHub, include:

Misuse of MinMaxScaling

It's a widespread practice to apply MinMaxScaling to financial time series, such as stock prices, aiming to normalize the data. However, this approach may not be ideal. Specifically, using raw prices is often discouraged; instead, focusing on price changes or returns is recommended. This adjustment becomes crucial when a stock price in the training set never surpasses a certain threshold (e.g., €40), making it challenging for the model to predict prices beyond this range accurately. Adapting the model to understand and predict changes or movements in prices rather than absolute values can significantly improve its applicability and performance.

Incorrect Train-Test Split
A typical mistake involves using data from the test set inadvertently during the training phase or for making predictions, which can lead to overly optimistic performance evaluations. The correct approach is to ensure a strict separation between training and testing datasets, where the model is trained on one set of data and tested on another, entirely unseen set to accurately assess its predictive capability.

Using LSTMs with a Sequence Length of 1
Employing LSTMs with a sequence length of 1 fundamentally misunderstands the purpose and strength of these models. LSTMs are designed to capture long-term dependencies in sequence data, making them particularly suited for time series analysis. Using a sequence length of 1 turns these models into something akin to a standard feedforward neural network, negating the benefits of LSTMs in understanding temporal dynamics. A more appropriate use of LSTMs involves setting a longer sequence length to leverage their capacity to process and learn from sequences of data over time.

Additionally, the prevalence of plots showing perfect predictions on the internet complicates the search for high-quality code and reliable modeling practices. Such representations can be misleading, as they often do not reflect the true predictive capabilities of a model, especially in the inherently unpredictable domain of stock prices. This can set unrealistic expectations and hinder the understanding of LSTM models' practical applications and limitations. It's essential to critically assess the credibility of online resources and understand that in the realm of financial time series forecasting, perfect predictions are more an exception than the norm.

7.4.10. Documentation of the Coding Process

The development process of the code was iterative, starting with a basic structure and evolving as understanding and competency grew. Learning the importance of structuring and documenting the code effectively to ensure maintainability and scalability of the project became clear. Particularly in unfamiliar territory, it was evident how quickly code could devolve into chaos. Utilizing Jupyter notebooks in the initial phase allowed for the documentation of code alongside visual data analyses and theoretical explanations, providing a comprehensive overview of the development process. Moreover, without the need for debuggers and breakpoints, it was possible to isolate specific sections and execute

them individually, which proved to be very helpful. As the project progressed, the code was organized into reusable functions and modules, implemented in Visual Studio Code. This enabled a clean separation between data processing, model training, and evaluation. Emphasis was placed on writing clear and informative comments that elucidate the logic behind key decisions.

7.4.11. Future Perspectives and Enhancements for LSTM Models

Looking ahead, there are some areas for future exploration to enhance the strength of LSTMs:

Artificial Rabbits Optimization (ARO):
The ARO algorithm, inspired by the foraging behavior of rabbits, could offer a novel approach to optimizing LSTM networks. Its potential for efficiently searching the hyperparameter space warrants investigation, potentially leading to improved model performance and efficiency.

Incorporating Momentum:
Momentum, a critical factor in stock markets, represents the tendency of stocks to continue moving in the same direction. Incorporating momentum-based features into LSTM models could significantly enhance their ability to extrapolate future market movements, offering a more nuanced understanding of market dynamics.

Leveraging More TensorFlow Features:
TensorFlow offers an extensive array of functionalities, such as scaling data to a Gaussian distribution, which could further refine the LSTM model's input data for better normalization and potentially improved predictions. Given the vastness of TensorFlow's ecosystem, exploring additional features and functionalities could unlock new levels of LSTM performance.

Exploring Mogrifier LSTMs:
Mogrifier LSTMs, a recent innovation in the LSTM architecture, have shown promise in enhancing model performance by dynamically adjusting the input and recurrent connections. Testing and integrating Mogrifier LSTMs could offer further advancements in handling complex time series data.

In conclusion, while the current LSTM models have demonstrated considerable capabilities in time series analysis and prediction, the potential for further enhancements remains vast. By exploring novel optimization algorithms like ARO, incorporating momentum and additional TensorFlow features, integrating unexplored data features, and experimenting with advanced LSTM variants such as Mogrifier LSTMs, future research could significantly advance the state of the art in LSTM applications, particularly in challenging domains like stock market forecasting.

8

Model: CNN

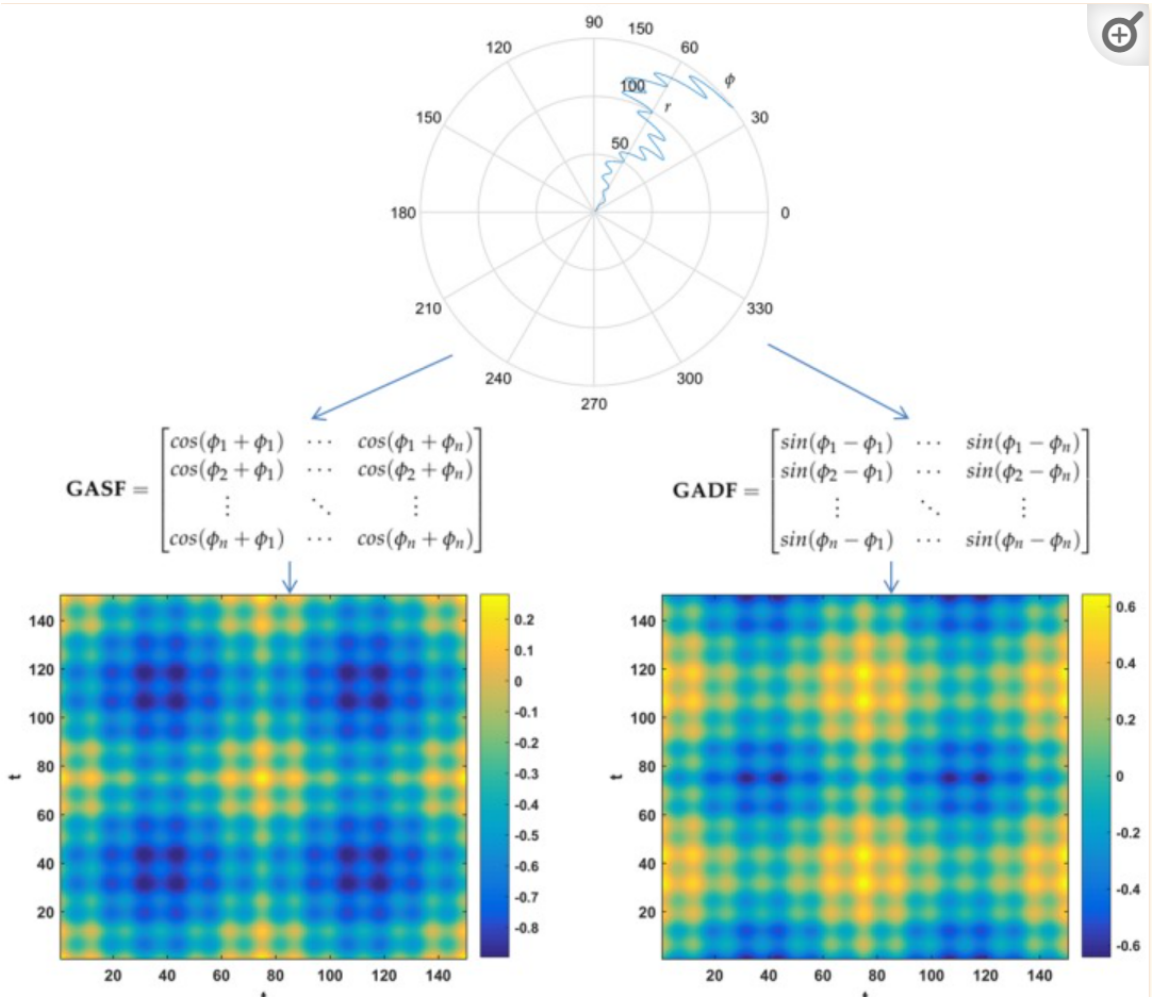
8.1. Solution Methodology

To solve the Problem defined in Chapter 1.2, this chapter is dedicated to explain the approach of using CNNs to contribute to the overall Project Solution. There are two main approaches we choose to follow; one is based on the identification of specific Trading Patterns with the help of Visual Pattern Recognition and the other one is based on imaging the timeseries followed by the use of a CNN. For the use of a CNN the 2-Dimensional timeseries data, consisting of (timestamps, feature values) will be transformed into 3-Dimensional Data consisting of (timestamps, feature values, feature values) where the additional feature Dimension includes the temporal relation between different the time intervals within one series. This in enables the CNN to better process the newly represented Information, due to the detection of relevant relations.

Unfortunately, the method of identifying Trading Patterns is not described in the following due to the retirement of Philip Thielges. So, the main topic of this Chapter is to describe the process of predicting the change in value of a chosen stock with the use of a CNN Model.

The approach is based on the use of Gramian Angular Summation Fields [14] which is a technique that encodes the time series signals into an image. The concept here is to transfer the time series to a polar coordinate space. The Gramian matrix is then formed where each element is calculated by the cosine of the summed angles for GASF or the sine of the subtracted angles for the GADF. This process captures the relationships between different segments of the signal. As this is not a scientific Paper and not the main topic of the project I will not go into more detail. Important is that the method will allow us to use Stock Prices given as Timeseries Data, as Image Data. The chosen Library for using GAFs is from “pyts” [12]

Figure 8.1.: Image overview



8.2. Implementation

As the task may intuitively lead to the choice to predict all defined stocks at multiple timesteps all within one model (Matrix – to – Matrix prediction) in the following I elucidate the reasons for the decision to implement the Problem with a (Matrix – to – Scalar prediction). The limitation of features in a Neural Network is crucial for the training duration and complexity (Curse of Dimensionality). As the use of a Gramian Angular Field multiplies the count of features the inclusion of all stocks as features would dramatically increase the

training data and consequently the training duration and complexity. This fact in addition to the circumstance that the project group has no relevant external resources available leads to the decision to depict the problem as a Matrix – to – Scalar Problem. So, a Model is defined with a Matrix Input and a Scalar output. The Input, as previous described is composed of (length of a single series, count of features, count of features) and the output value represents an enhanced percentage-based change for a specific time in the feature. The Enhancement is a functional unary operation to increase the granularity of the prediction, this will be further explained in the chapter 8.2.1 "Preprocessing" einfügen. To finally realize the prediction of a Matrix, in order to fulfil the problem description, multiple independent CNN Models were created, each predicting a scalar at a different time. To realize this concept a software component imports the relevant Models, executes them iteratively and merges the results to a prediction series, this process will be further explained in the chapter 8.3 "Software Architecture".

8.2.1. Preprocessing

The Preprocessing is the backbone of the CNN Method and consists of multiple steps which are executed sequentially. Each preprocessing Step is implemented as a service that takes in the output data of the previous executed preprocessing step. The First step is to load the given csv raw data, filter the relevant feature columns and parse the Date given as a string to pandas DateTime format.

Data Integration

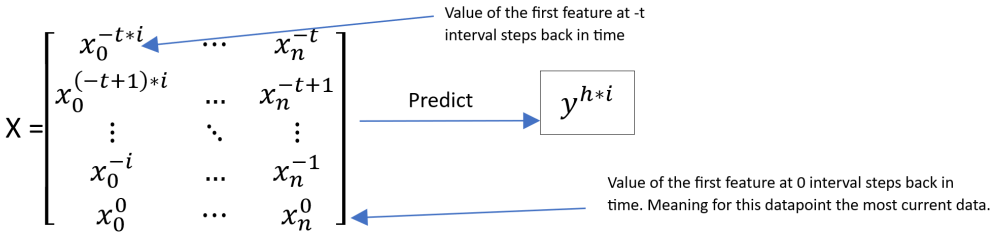
After Loading the data we use the existing stock-data to cross-reference other databases and integrate their data entries (ETF, Index files). For the prcocess of imaging with the use of a GAF the limit of Features is crucial as in the process of the image data creation, the number of features will be doubled. This is because each feature will be included in the width and height of the image. The chosen features for the CNN are "Open, Volume, MCHI, IVV, avgOpen". The previous Open values are obvious as it represents the history of the stock itself, the Volume aims to include the relation between the amount of trades and the future price, MCHI is designed to track the performance of the MSCI China Index so that the Economic Rivalry and Suppliers are taken into account and finally the IVV aims to mirror the performance of the SP 500 Index as each chosen stock is in the US.

As Features are derived from ETF Files as well as the respective Stock File a merge process for the different data sources must be executed before building the datapoints to ensure that the data for the respective timeseries is extracted at the same timestamp. If data from a feature (ETF File) is not available at the exact time as the data from the stock file, the timestamp is being considered invalid as this may otherwise produce NaN errors or inaccuracies. So, this multiplies the invalid timestamps in the building process which makes the tolerance and "retryThreshold" mechanism even more important. The transformation from the continuous database to a Set of datapoints where each element represents a multivariate timeseries, will be described below:

Timeseries Builder

This section aims to explain a fundamental component of time series data processing. It will outline the key parameters of the temporal scope of the data. The Data to train the CNN on and consequently the input data to predict a Stock Price is defined with 4 parameters. The first parameter denoted as "length" defines, given a timestep t=0, how many timesteps (intervals) back each datapoint includes. The second parameter will explain the temporal granularity of each datapoint, termed the "interval". The "interval" defines the temporal spacing between consecutive timesteps, it ensures appropriate temporal resolution. Lastly, the prediction horizon, delineating the temporal gap between the final timestep of the input and the initial prediction output.

Figure 8.2.: Image overview



$$X = \begin{bmatrix} x_0^{-t*i} & \cdots & x_n^{-t} \\ x_0^{(-t+1)*i} & \cdots & x_n^{-t+1} \\ \vdots & \ddots & \vdots \\ x_0^{-i} & \cdots & x_n^{-1} \\ x_0^{0} & \cdots & x_n^{0} \end{bmatrix} \rightarrow y^{h*i}$$

$t = timesteps\ back, \quad i = interval\ (time\ spacing),$
$h = timesteps\ ahead, \quad n = count\ of\ features$

For Example:

$$X = \begin{bmatrix} 122{,}23^{8:00}_{Open} & \cdots & 8201^{8:00}_{Volume} \\ 121{,}89^{8:10}_{Open} & \cdots & 1102^{8:10}_{Volume} \\ \vdots & \ddots & \vdots \\ 124{,}45^{9:30}_{Open} & \cdots & 1602^{9:30}_{Volume} \\ 125{,}61^{9:40}_{Open} & \cdots & 8000^{9:40}_{Volume} \end{bmatrix} \rightarrow 126{,}49^{10:00}_{Open=Y}$$

Predict

In this example a timeseries with the interval of 10*minutes,*
the length of 10 *timesteps and*
a prediction horizon of 2 *is shown*$(2 * i = 2 * 10minutes = 20\,\text{min}\ in\ the\ future)$

Example 1: As the nature of stock trading data is not completely consistent in the meaning of missing data points because of weekends, holidays, technical error and so on, a mechanism is needed to bridge the gap between two entries within a timeseries where one timestamp is not consistent. For example, given a timeseries with the length of 10 entries, an interval of 120min and a prediction horizon of 5 (5 * 120 = 700min ahead). Starting at 9:30am as this is the time ETFs and Indices can be traded, the following structure clearly states the problem: [9:30-01:01:2010, 11:30-01:01:2010, 13:30-01:01:2010, 15:30-01:01:2010] as the x would logically be 17:30-01:01:2010, but at this time there are no longer measured trades. So different solutions were found to fill up the remaining 6 missing entries. I chose in arrangement with the Transformer-Group the approach of going to the next day, so the timeseries will continue with [9:30-02:01:2021, 11:30-02:01:2010, 13:30-02:01:2010, 15:30-02:01:2010, 9:30-03:01:2021, 11:30-03:01:2010]. For interval definitions of 10min for example in the day trading area, filling up lacking timestamps immediately with the data of the new day, is not that optimal as for example even though the timestamp at 5pm is not available 16:59pm is and may be semantically more accurate. So, I chose to build an algorithm that

aims to generate consistent timeseries as a machine learning model highly rely on patterns and are sensitive to noise within the data therefore a consistent temporal resolution is extremely important. Depending on the interval aka the resolution of the timeseries two parameters additionally to the interval, length and horizon can be configured during the built of the training data. The “tolerance” and the “retryTreshold”, the tolerance enables the previous highlighted situation as the tolerance defines how much difference is allowed between two adjacent timestamps to be considered valid e.g. tolerance of 1 would allow the values 4:59pm and 5:01pm to be valid even though the timestamp 5pm is being looked for. The retryTreshold allows to handle holidays, weekends and other situations where the tolerance is exceeded and data of the next day is selected. As the data of the next day is also not every time available for example on Fridays or holidays, this will be configured in the retryTreshold as it determines in the algorithm how often to iterate to the next day to find a valid successor.

Time Series Transformations

For the CNN to better identify the patterns within the data and enhance the temporal relations as well as dependencies of the features, two Time Series Transformations were executed.

Moving Average

During preprocessing each time series will be extended with a Moving Average feature of the “Open” Value to mitigate noise and outliers present in the data. Additionally, this process allows the model to identify and capture trends, by smoothing out short-term fluctuations.

Differencing

Having added the moving average, another time series specific method is executed. To avoid a situation where the model tries to learn absolute pattern regarding specific prices, the differencing method is used to generate a series of differences rather than a series of absolute prices. This allows the elimination of trend such as Inflation. Most importantly, after having extracted the difference in value the percentage-based change of value will be derived from the difference, this allows the identification of patterns that are relative in time and comparable across the years.

Functional Transformations: Unary operations

One key operation during the preprocessing that had a great impact on the model performance is the use of a multiplication unary operation. After calculating the percentage-based difference, a multiplication by the factor 10 is executed on the difference to increase the granularity and the distance between prediction and label. As the change in value in stock trading, especially when taking adjacent or close distance samples, is often marginal. So, to increase the distance and accordingly also the loss, a desired change value of 1,2

Normalization

The Gramian Angular Field Method requires the values to be within a range of [+1, -1] so a min-max scaling is executed to standardizes the chosen features by mapping them to a common scale.

Gramian Angular Field Transformation

The last step in the preprocessing is to transform each single preprocessed timeseries into a gramian angular summation field. This is explained mathematically as follows. First, let vectors be denoted by bold lower case, scalars by lower case and matrices by bold upper case. For n real valued observations in a time series x = x1, x2, ..., xn, the latter are first normalised between 1 and 1 as

Figure 8.3.: GAF Transformation

$$\bar{\mathbf{x}} = \frac{x_i - max(\mathbf{x}) + (x_i - min(\mathbf{x}))}{max(\mathbf{x}) - min(\mathbf{x})}.$$

This provides angular values in the range [0,], which will aid in obtaining information granularity in the GAF. The next step is to obtain the polar coordinates which are the cosine angle, , from the normalised amplitude values and the radius, r, from the time stamp t, as presented in Equation:

Figure 8.4.: GAF Transformation, Cosine

$$\begin{cases} \phi = \cos^{-1}(\bar{x}_i), & -1 \leq \bar{x}_i \leq 1, \bar{x}_i \in \bar{\mathbf{x}} \\ r = \frac{t_i}{N}, & t_i \in \mathbb{N} \end{cases}.$$

features: (a) a one-to-one mapping of the time series to the polar coordinate results, so it is bijective, and (b) temporal relations are preserved. The polar coordinates of the normalised time series in the [1, 1] interval fall in the angle boundaries [0, π]. This provides different concentrations of information in the GASF, which should benefit any classification task [7] To conclude, the now created GASF will be the input for the CNN to identify relevant parts for the prediction.

8.2.2. Visual Illustration of the Preprocessing

Figure 8.5.: Merging Stock and Feature Data

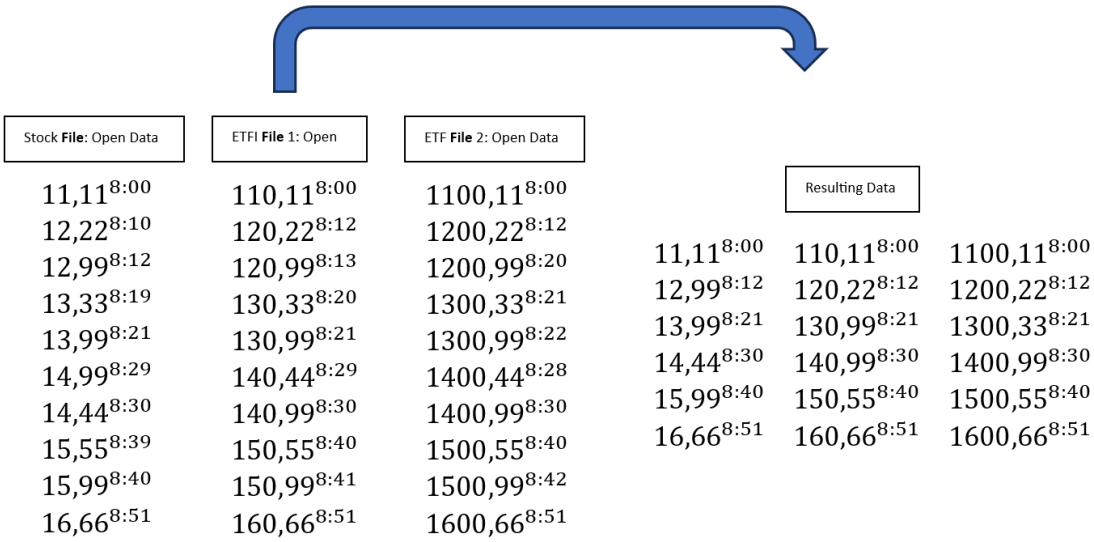


Figure 8.6.: Building the series

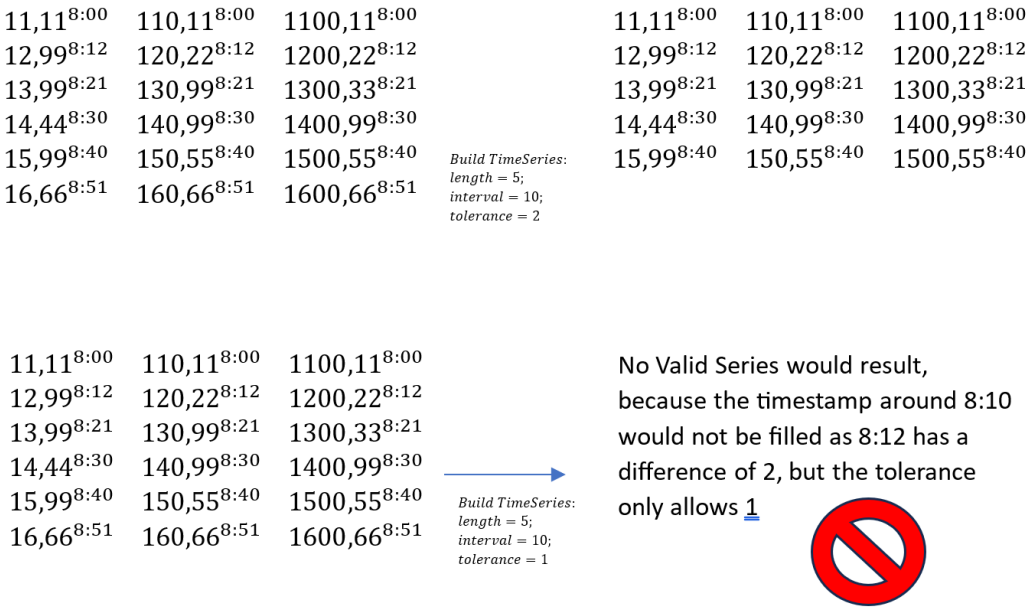


Figure 8.7.: Calculate Difference, Change and Multiplikation

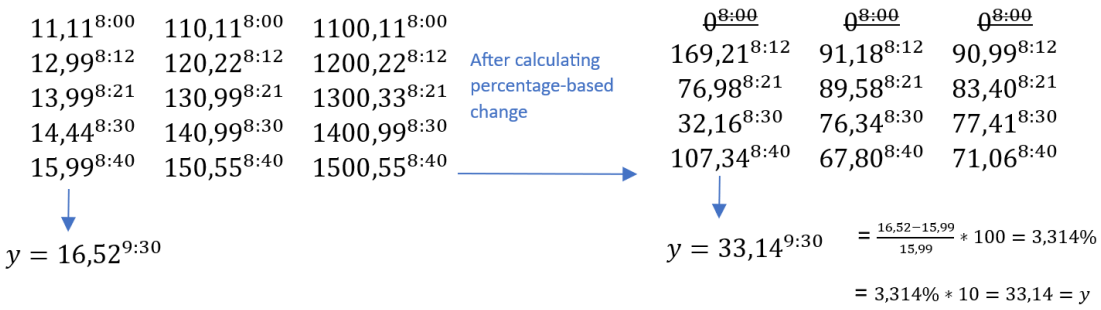
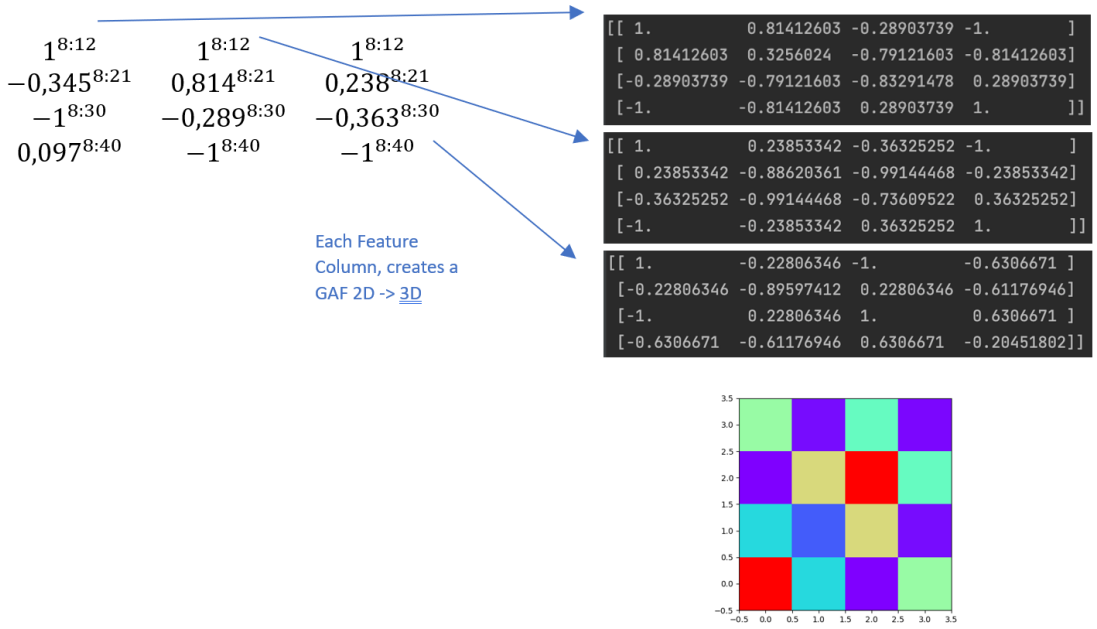


Figure 8.8.: Scaling with Min-Max [-1, 1]

Figure 8.9.: 2D -> 3D GASF Data



8.2.3. Training

For the Training process a Model Architecture consisting of a total of 6 Layers were chosen. Starting with 2 Convolutional Layers and 1 Pooling Layer followed by 2 additional Convolutional Layers with 1 Pooling Layer. I chose this architecture based on a test series. Although extending the Architecture with additional Layers increases the model performance, the trade off is not worth taking, as the increase is only marginal but the computation time

skyrockets. As the project does not have external resources available, I chose the smaller version. The training – test ratio is 80 – 20 with 5 Epochs, were after each epoch a test loop is being executed to assess the performance and a potential stagnation. Dropout and L2 Regularisation did not have any positive impact and were felt out in a later stage of the project. For the main part of the Training a MSE (Mean Square Error - Loss) and a Adam optimizer ("Adaptive Moment Estimation") was used.

8.2.4. Predictions

In Order to predict live values, a combination of task needs to be executed. The Process is as follows: Frist the models for the chosen TradingType and Stocks will be loaded. Based on the chosen Time Range the live data will be loaded from the source e.g. the validation files. Having the loaded the correct models and the data, a preprocessing pipeline for each stock will be executed in order to correctly parse the live data for each stock into GASF ImageData. Each GASF Image Data will be fed into all models for the same stock (as models are stock and time specfic). Finally, all predictions for all filtered models will be mered into a dataframe and returned to the frontend

8.3. Software Architecture

The following chapter describes the chosen Software Architecture to reliably predict the requested stock prices at a specific time. The Project Structure is subdivided in 3 components that are individually designed and executable. All Components are configured and controller with .yml files to flexible adjust settings. The backbone of the CNN Project is the preprocessing Component, following the preprocessing the generated data will be used in the training Component to train a CNN Model with specific features and specific hyperparameters. Finally, to generate the predictions the prediction Component will be executed. The prediction Component enables the import of the necessary models, combines the predictions, and parses the result to the defined pandas Dataframe 1.) `src.preprocessing`:

- **Input** = .yml file defining, used features, location of resources, timeSeries parameters (interval, length, prediction horizon)

- **Use** = processes the raw data, as described above and stores the processed data in as a .npy file.

2.) `src.training`:

- **Input** = = .yml file defining where to find the preprocessed data, the model hyperparameters, logging settings.

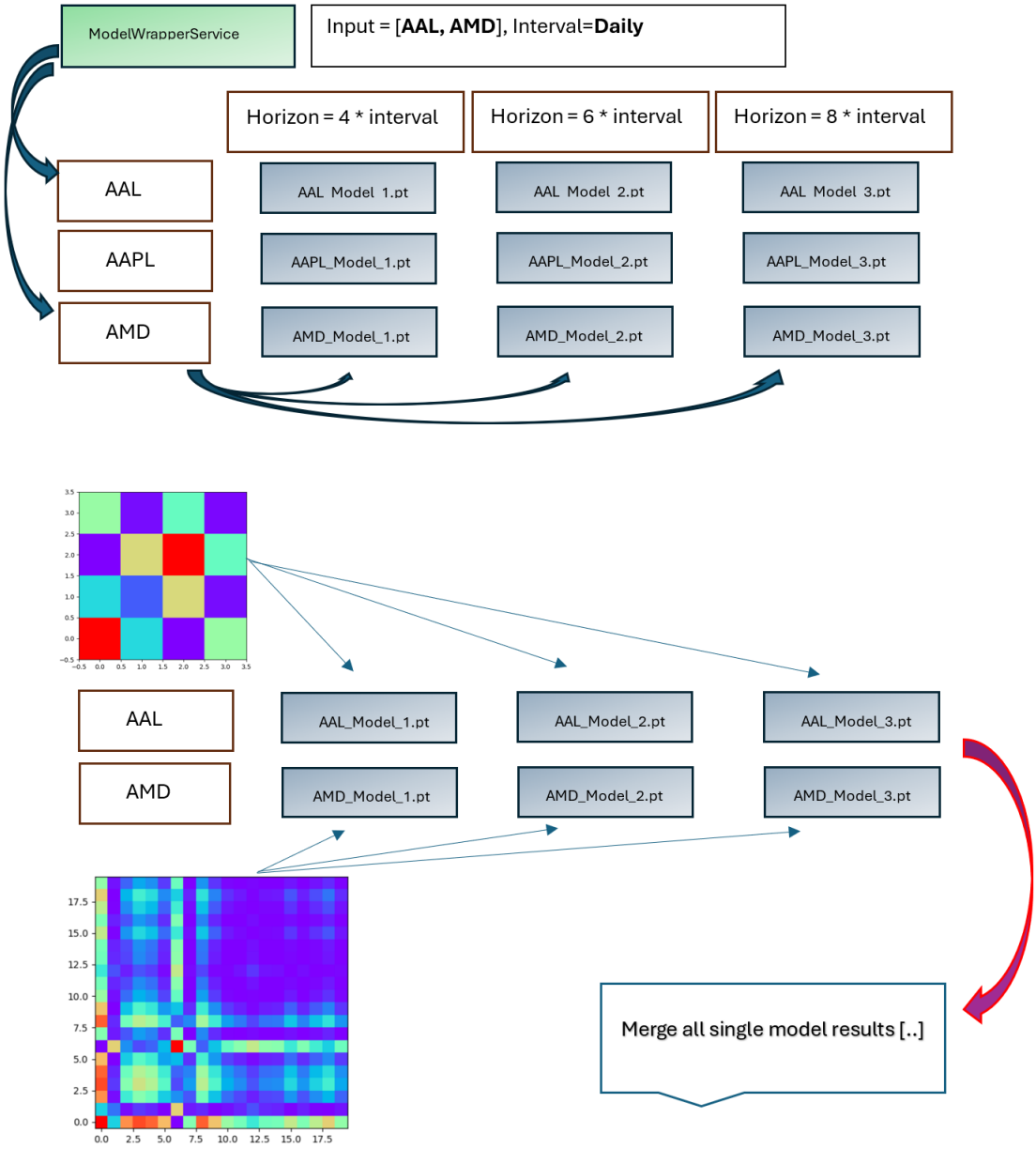
- **Use** = creates a Trainer, a pytorch Model, traines the model and export the model as .pt file to the results folder

3.) `src.prediction`:

- **Input** = the request from the backend server using the defined api. The request contains a list of stocks as well as a date range

- **Use** = Given the requested stocks and date range, the prediction component, loads the necessary models from the result folders, calls a preprocessing pipeline, executes all relevant models, and assembles the result to the defined data frame format

Figure 8.10.: ModelWrapperService



8.3.1. Performance vs Scalability

As described above the Preprocessing pipeline is very scalable as new steps can be just added, but in contrast the runtime for the execution of the preprocessing component is extremly high. As this would make the chosen architecture inefficient, preprocessed data is serialized and saved. So for cahnging hyperparameters the preprocessing can be skipped entirely. In addition a other way to enhance the preprocessing execution time would be to loop 1 time thorugh the data and execute all steps for the respective dataPoint during that loop instead of looping 1 time entriely then returning the processed data to the next service and looping over it again. This method on the other hand would lead to more complexe code and would be harder to understand for new developers.

8.4. Results

In this section the results of the AAPL stock, for the 3 Trading Types are displayed. This models are only Matrix - to - Scalar Models and therefore only predict specific timesteps ahead. The prediction Horizon for the images are stated above and the displayed loss on every image is a "MSE". It is clearly visiable that the splitting of train and test data was not optimal was the test predictions are to accurate, but the images also show that the avarage loss decreases steadily over each epoch.

8.4.1. Daily Model

The Image Displays a Model with a Interval(Temporal Spacing) of 1440min = 1Day and a Prediction Horizon of 5 (5*Interval = 5*1440min => 5 Days)

Figure 8.11.: LongrangeTrading: AAPL

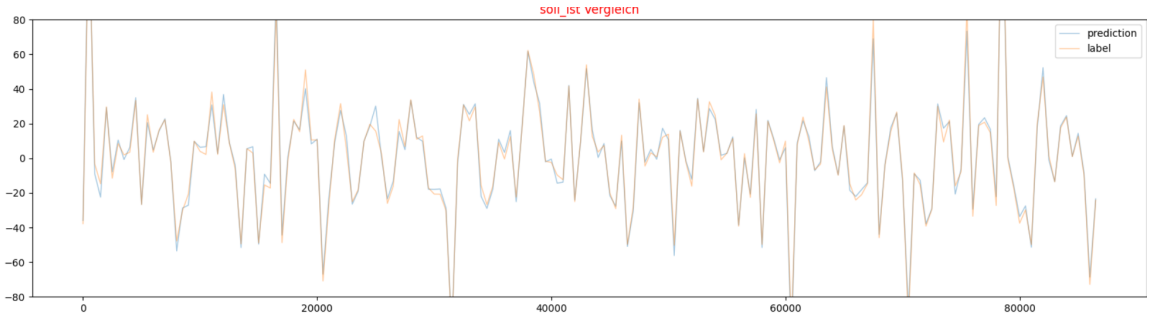
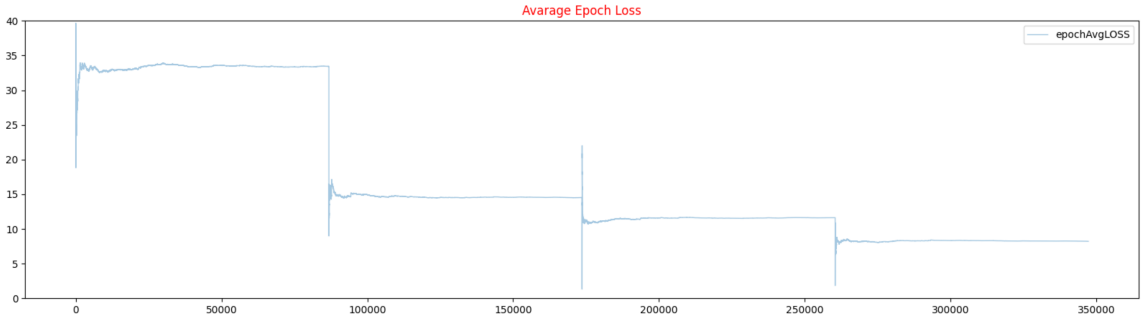


Figure 8.12.: LongrangeTrading: AAPL



8.4.2. Two-Hourly Model

The Image Displays a Model with a Interval(Temporal Spacing) of 120min = 2Hours and a Prediction Horizon of 5 (5*Interval = 5*120min => 6Hours)

Figure 8.13.: SwingTrading: AAPL

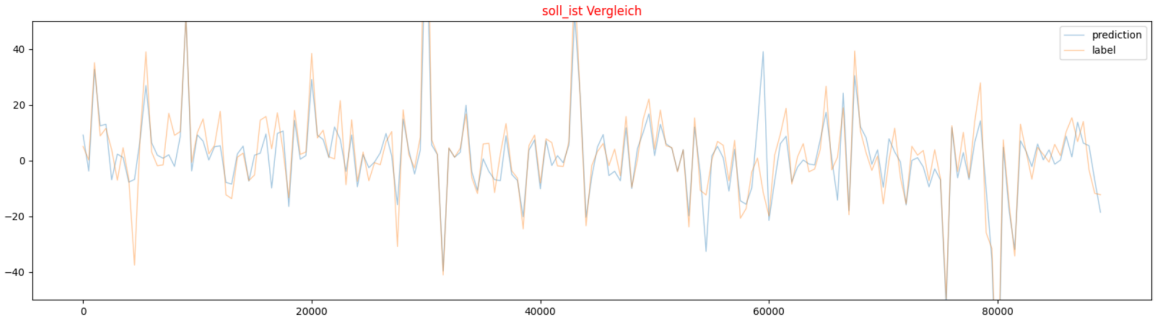
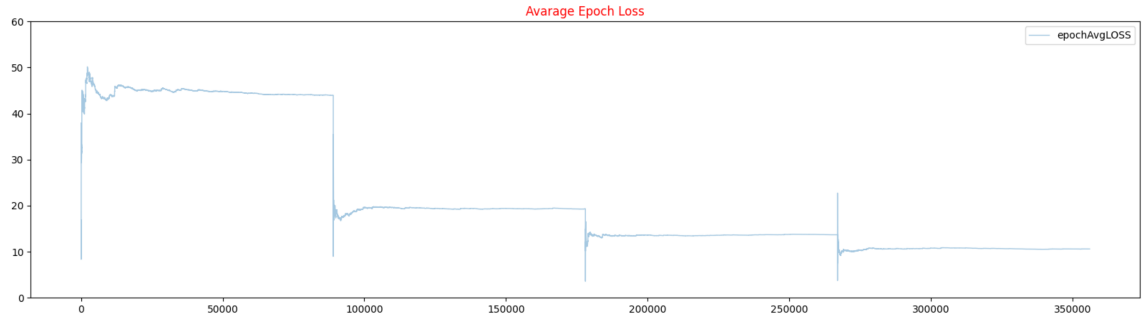


Figure 8.14.: SwingTrading: APPL



8.4.3. Minutely Model

The Image Displays a Model with a Interval(Temporal Spacing) of 1min and a Prediction Horizon of 5 (5*Interval = 5*1min => 5min)

Figure 8.15.: DayTrading: APPL

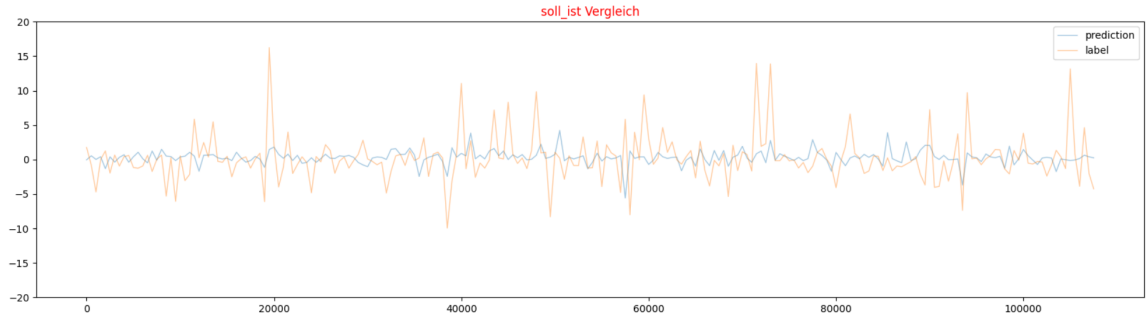
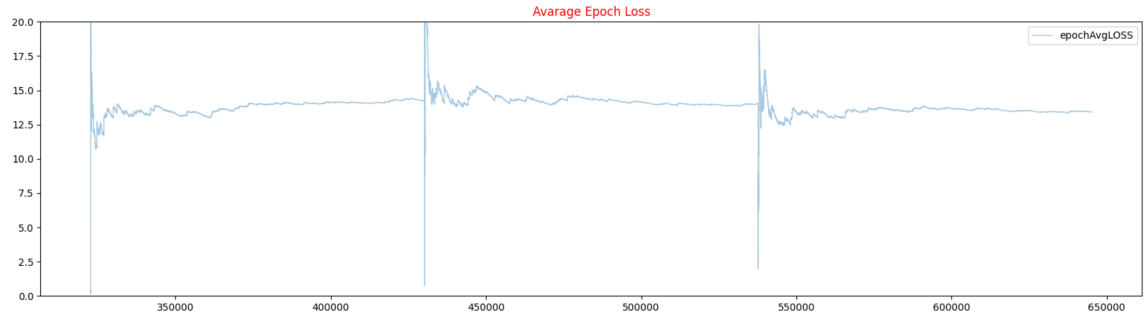


Figure 8.16.: DayTrading: APPL



8.5. Conclusion

To continue the CNN Component multiple approaches can be taken. One of the most promissing aspect would be the adjust the Matrix - to - Scalar prediction to a Matrix - to - Vector in Order to predict more values. In addition to that a other method for the creation of timeserieses can be taken, for example taking only values at the excat same timeStamp or always data at the end of the day. Lastly, more features can be included such as industry indicies and a score for positive or negative analyst omments.

9

Model comparison

To compare the performance of the different models that predict stock prices, multiple metrics were developed and implemented. The evaluation of the models with these metrics is also implemented and visualised in the front-end to enable the user to choose a model for his investment strategy. For example, the user could select a specific model that performs better on stocks he wants to trade. *(Niklas Kormann)*

9.1. Selection of evaluation metrices

To evaluate the models we selected or developed three metrices:

- Mean absolute error
- Mean error
- Simulated portfolio performance

The mean absolute error is a common error measure that is easy to interprete. It is give by the formular:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{9.1}$$

The MAE is a measure for the variance of a model. It can be interpreted as the expected error a prediction might have, positive or negative, in $. This is easier to interpret than the MSE, which uses the sum of the squared errors. The unit of the values would be square dollar, which is not an useful unit, as for the human brain it is difficult to interpret polynomial growth.

The absolute error can be used as a measure for a bias in the prediction, i.e. a systematical error in positive or in negative direction. It is calculated by the formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} y_i - \hat{y}_i \tag{9.2}$$

The simulated portfolio performance is a more complex evaluation measure and was developed by ourselves. The idea behind it is to have a measure that gives an indication how stock a portfolio would have performed if you had based all trading decisions on the evaluated model. This is supposed to give a intuitive and simple indication to the user about the model's performance.

The difficulty in developing this measure was to translate pure price predictions into a trading policy. This kind of measure is not only sensitive to the performance of the model, but also to the design of the trading strategy that we implemented in this measure and how well it fits to a specific stock.

The measure is implemented by an algorithm that simulates trading activities based on the prediction of the model for a given time window. The inputs are the absolute predictions at each time step in the time window for each prediction time step and for each feature, i.e. stocks, and the true values for each stock at each time step in the evaluation window. At each time step the algorithm decides whether to buy 5 shares, to hold or to sell all shares for each stock individually. The algorithm keeps track of the money invested, the number of shares in the portfolio for each stock and the return from the cash-outs. At the end, the relative performance is calculated by the following formula:

$$Performance = \frac{Return - money_invested}{Money_invested} * 100 \tag{9.3}$$

We made some simplifying assumptions that limit the meaningfulness of the measure based on this algorithm:

- The amount of shares that could be bought in one time step is limited to 5 per stock
- The cash to use for investments is unlimited
- There are no trading costs
- We can buy a stock to exacly the current price

As stated before, the trading strategy when to sell, buy or hold was the key challenge, as the models do not only predict a single step ahead. We decided for the most simple strategy:

- Buy: If all predicted values are higher than the current value
- Sell: If all predicted values are lower than the current value
- Hold: In all other cases

The idea behind this strategy is to only sell or buy in the worst or best case prediction. This strategy ignores all scenarios, where the price is predicted to fall first before rising again or vice versa, which could also be traded on. For example, one could argue that it was also profitable to buy when the predicted values fall before rising. Under the assumption that there is a limit to buy-orders, this could increase the amount of stocks to profit from the rise. With our strategy, the algorithm would only buy stocks after reaching the local minimum. We decided against a more granular trading strategy, because the limitations mentioned above imply that none of these strategies would represent the reality completely accurately anyway. Therefore, this measure should primarily be used to compare our models with each other. *(Niklas Kormann)*

10

Frontend

10.1. Frontend Development Phases

10.1.1. Phase 1: The Concept

At the start of the project, the main task was to brainstorm and prototype some ideas for the frontend layout. This included the general look as well as the possible functionalities the frontend should be capable of. A part of this was doing some internet research to gain ideas from already existing products. This was especially helpful, since through comparison of different approaches, a clear idea could be generated that was a sum of the pros of the found existing solutions (most important being easy to understand and use). With this, some ideas about possible layouts were generated using picture combinations, and the most promising ones were chosen as pre-layouts. Before beginning the frontend development, some meetings were held where the key points of the application were discussed (mainly with the backend) to ensure that the desired functions were well chosen and possible to realize. This included initial ideas of which API interfaces were necessary. Also, the frontend was integrated early into Docker to be easier to use and review from different platforms.

10.1.2. Phase 2: Proof of Concept

While discussing the most important points, the basis for the frontend was developed. For the development, Vue.js was chosen due to the fact that it is easy to use, performant, good documentation is available for it, and it was already known by the backend developer. During this early phase, different components were tested. The first components programmed for the frontend (and Backend) were the main graphs, the login, the header, and some info

subpages. The graph page was designed to be able to compare models but also to be able to see the model information individually. To achieve this, a combo-chart approach with single charts underneath was used. Thanks to this, only charts checked in the menu appear in the combo chart, which makes the choice of charts to compare easy. Also, the single chart of the model only appears when the model is checked, so the page is not flooded with unnecessary information. Some other parts like a progress bar and a Dark mode were also implemented but taken out later in the project.

10.1.3. Phase 3: The MVP

For the MVP phase, the main task was to get a prototype that had all the basic functionalities included. This included having a home page for trading (wasn't working with real data), having a subpage to view the model predictions (was working with real data), and having some subpages with information about the models and stocks (just had some pretty basic information). To achieve this, a lot of pair-programming and meetings were used so the software communication between the backend and frontend would function as expected. In this stage, a decision for the final layout was made by talking to the team about the changes they found most visually appealing and user-friendly.

10.1.4. Phase 4: The Prototype

For the prototype, the biggest challenge was to implement the reinforcement learning and finalize the pages to be usable as expected. Since the reinforcement learning model takes into account the predictions of all other models, it is included on the home page. For the subpages, the decision was made to create 2 subpages for the predictions. One where all the models can be compared by predicting the same stock. The other page is the corresponding opposite, where a model is chosen and different stocks can be predicted. This choice was made because combining those pages resulted in a combined page with both functionalities that was not user-friendly. Additionally, errors were included in the models to get a better insight into the prediction quality (absolute mean error for the quality, mean error to recognize drift). Another change is that the colors of the series in the combo and single chart match, to make it easier to find the single model chart after looking at the combo chart.

10.2. Final "as is" summary with explanations of the design choices

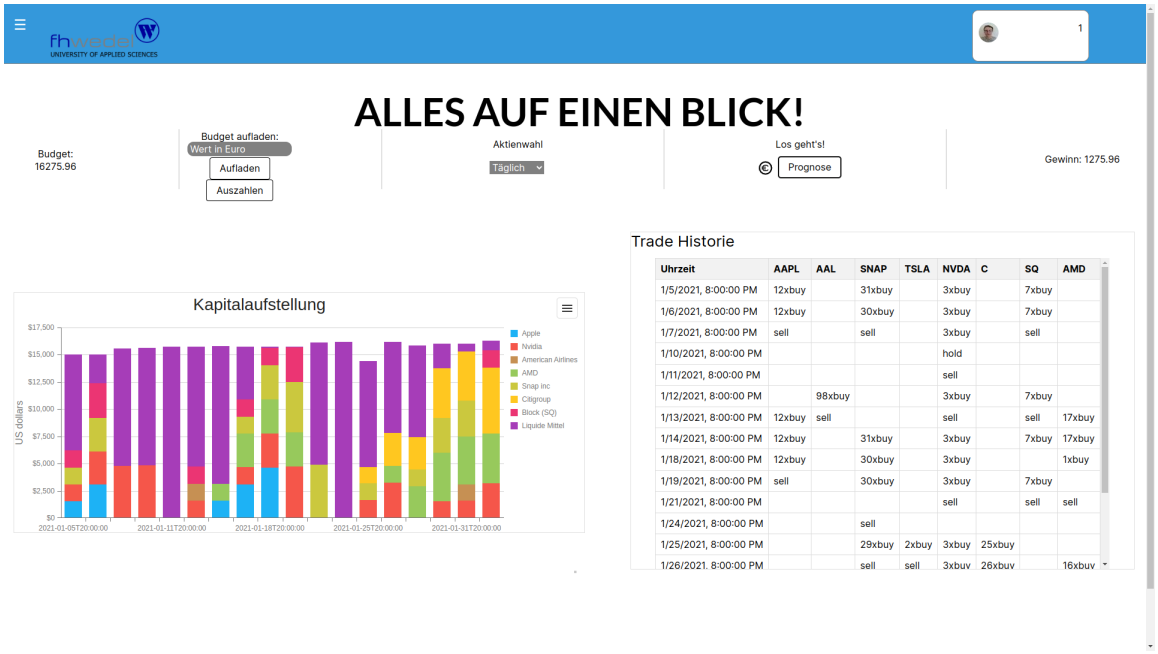
In this section, the pages will be explained individually, and design choices that were not explained before will be discussed deeper. Since the compare and info pages don't differ that much in the user workflow, they are discussed together with the individual characteristics pointed out.

10.2.1. General Information about the pages

When doing any prediction, some parameters have to be chosen (more detailed information about why the parameters are chosen this way is handled in another section). The design goal was to make it as intuitive as possible. The typical flow to choose everything from left to right was chosen, and the parts to choose something were differentiated by separators. Single answer choices like the time interval were implemented using a dropdown, everything with more answers was designed to be checkboxes. The general color scheme was chosen to fit together with the colors of the FH Wedel logo. The Burger/Dropdown menu was chosen to be able to navigate between the pages.

10.2.2. The Homepage

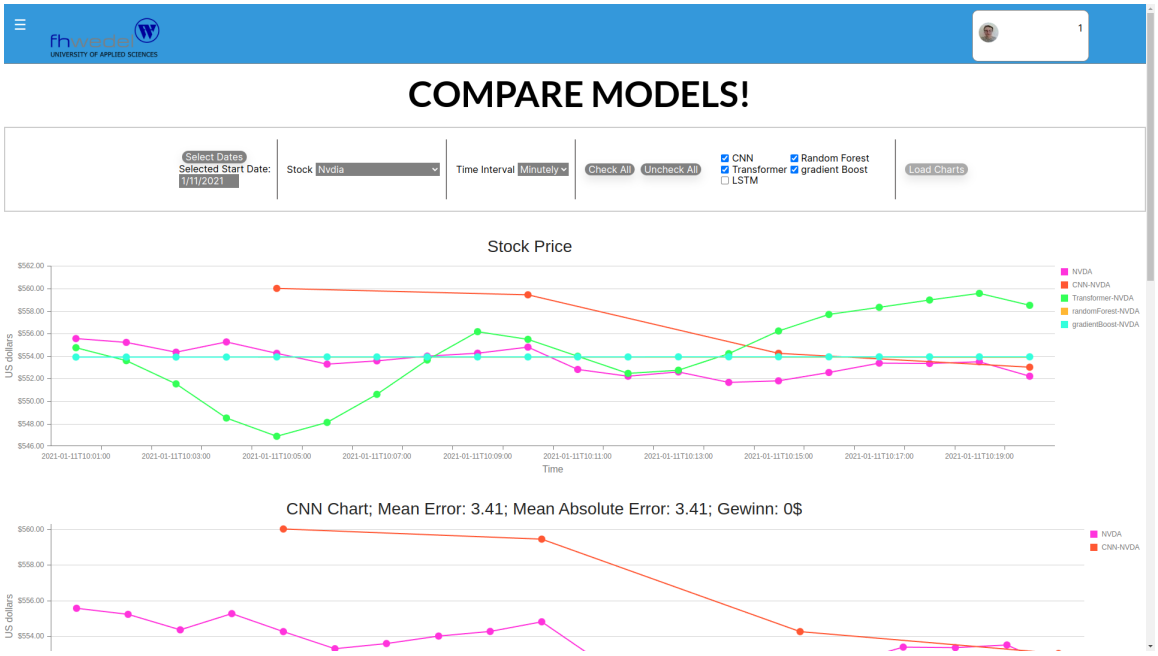
Figure 10.1.: The Homepage



The homepage was designed to be a few-click solution. A user can choose his budget and a time interval to trade and start the trading simulation. To achieve even more simplicity, the selection of the time and stocks is done automatically (if wished, this can be adjusted by implementing the date picker component from the comparison sites). The budget of the reinforcement agent is then displayed as a stacked bar chart, to be able to easily see how the budget of the agent is spread. The actions taken by the agent are shown as a log table on the right. This way, the trading done by the agent is as transparent as possible.

10.2.3. The CompareModel and CompareStock Pages

Figure 10.2.: The Model Comparison Page



These sites are the key pages to get information about all the prediction models. They were designed to be able to compare each model with the others, as well as compare the different stock predictions done by the model. The errors of each model-stock combination are calculated, as well as the earnings that could be achieved by trading a stock with a model. They are displayed in the title of the chart to be able to see all the necessary information at one glance. The most individual part of the compare model site is the combo chart, where the different model predictions are combined in a single chart for comparison. Also, the option was given to check or uncheck all models, to save time for bigger selections and in preparation for possible upcoming additional models/stocks. The line colors were chosen to be visually appealing and rich in contrast to each other to be able to see the different lines easily. For the resolution of the axis, there are no big surprises, meaning the date was simply printed and the value is shown with two decimal places.

10.2.4. The ModelInfo and StockInfo Pages

These pages are simply to show some information about the models and stocks. The information about the individual models was created by the groups, to point out the most important and individual points of their model. The information about the tradable stocks was simply created using an AI Chat model, since the priority to fill these with deeper information was decided to be pretty low. Due to this, the search function for the stocks is a simple letter matching.

10.2.5. Not implemented features

The biggest part that was not implemented was a more detailed user profile with individual settings. This has the simple reason that the only important part here was the budget, which is individual to the user. The profile part on the top right was left there to give possible following groups a potential starting point for more individualization. Another feature that was cut out due to low importance was (as already mentioned above) the dark mode. It was already implemented for the home page but was not important enough to be transferred to the other pages, so it was removed completely. Also, progress bars were removed and replaced by a loading text due to the problem that the loading progress was hard to estimate.

10.3. Further Development of the Frontend

In this section, some ideas are featured which have been thought about but not implemented yet due to different reasons. If the number of stocks increases, there are some things that should be considered to be changed. One is the selection of the stocks at the comparison site. If the number of stocks increases, this should either be changed to a multi dropdown menu or a search. The search on the comparison page (as well as the search on the info page) should then use a smarter algorithm to be able to find the wanted stock easier (e.g., the Levenshtein distance). In a future project, the responsitivity can be improved aswell, to be able to view the page coorect on smartphone devices (the responsitivity got lost in the prototype phase, since it was a low prio). Possible other improvements could include a profile for individualizing the home screen and implementing a trade history

memory. For the model-stock information like error and value, another section or page can be created where the errors and the trading results can be showed in a table or heatmap, to make the comparison of the different model-stock combinations better comparable. When implementing live data, another page can be added to view the stock prices of the currently purchased stocks with a live feed of the value change and profit for now. The display on the home page can then be step by step updated with the new live data. (Developed by André Galczynski, Johann Beaumont Grigjanis, Thomas Parchmann; Section written by Thomas Parchmann)

11

Backend

11.1. Introduction

The backend plays a crucial role as an intermediary between the frontend and the AI interfaces that are responsible for executing tasks and algorithms. It forms the foundation of the system and enables seamless integration and communication between user interactions and the underlying AI models.

The importance of the backend is to provide a robust and efficient interface that receives and processes user requests and forwards them to the appropriate services. In doing so, it ensures reliable execution of the AI API and the return of relevant results to the front end. In addition, the backend is responsible for processing and storing data required for the AI models, as well as managing user accounts and authentication information. It acts as the central database for the system and ensures the security, integrity and confidentiality of the information.

By examining the functionality and architecture of the backend in detail, readers gain an insight into the complex interaction between users and the backend. This provides a better understanding of the importance of the backend as a critical component in a software project and emphasizes its role in providing a powerful and scalable platform for users.

11.2. Requirements analysis

General requirements: The backend should serve as the core component of the system and provide the main functionality. It must be able to receive and process requests from the front end. The integration of API interfaces for accessing AI models is required. The implementation of a user administration with login option and storage of (user) data is necessary.

Front-end interaction: The backend must be able to receive and respond to requests from the frontend. Communication between backend and frontend should take place via HTTP(S) to enable efficient data transfer. Support for various data formats such as JSON or XML for communication between backend and frontend is required.

API interfaces for AI models and methods: The backend must provide API endpoints to access different AI models and methods. The interfaces should enable simple integration of different AI algorithms. The API should be robust and scalable to handle a high number of requests while ensuring good performance.

User management: The backend must provide user management functionality, including new user registration and user profile management. User data should be securely stored and protected from unauthorized access, for example by using encryption techniques.

Database integration: A database connection is required to store user data as well as possibly other system data. A relational database such as MySQL or PostgreSQL or SQLite can be used to store user profiles, login information and other relevant data. The database should be scalable and allow for efficient data management to optimize system performance.

11.3. Architecture and design

Backend (FastAPI): We chose FastAPI to create a robust and efficient RESTful API. FastAPI is a modern framework for building APIs with Python that works on the basis of type annotations and standard Python types. It offers high performance and easy to read code.

The architecture of the application is based on the MVC (Model-View-Controller) pattern, with FastAPI taking over the controller area. The core logic is organized in separate modules responsible for database operations, data models and validation of user input. This keeps the code clean and well structured.

The database interaction is abstracted by the SQLAlchemy ORM (Object-Relational Mapping), which makes it easy to work with the SQLite database. Using SQLAlchemy also has the advantage that database operations are platform independent and can be easily migrated to other databases should this be required in the future.

An important aspect of the code is the implementation of security measures such as password hashing with bcrypt and the use of dependency injection for the database connection to ensure that each request gets its own database connection and database resources are managed efficiently.

The REST endpoints are clearly named and follow the RESTful principle of resource orientation. Each endpoint performs a specific action and returns the corresponding data, with error handling ensured through the use of HTTP status codes and user-defined error messages.

The integration of CORS (Cross-Origin Resource Sharing) via middleware allows the API to be accessed from different sources, which is important for accepting requests from a separate front-end application.

Overall, the combination of FastAPI, SQLAlchemy and well-structured code provides a solid foundation for developing a scalable and secure API that meets the requirements of modern applications.

Database (SQLite): This database schema uses SQLite as the backend database to manage the ”users”, ”login” and ”stocks” tables.

SQLite is a lightweight, serverless database engine written in C that is supported in most operating systems and programming languages. It provides an easy way to integrate a relational database into applications without the need for a dedicated database server. SQLite databases are stored in individual files, making them portable and easy to manage. They are well suited for applications with low to medium data volumes.

The ”users” table is used to manage user data. It contains information such as the user’s email address, username and password, as well as an indicator of whether the user is active and their current budget. This table is crucial as it stores the users’ identification and account information, which is essential for authentication and authorization in the application.

The ”login” table is used to log login information. It stores timestamps for user logins, which is useful for analyzing login behavior or performing security checks. This table is important to collect information about the use of the application and to monitor user behavior.

The ”stocks” table contains information on various stocks, including their trading data such as prices and volumes during a specific time period. This table is relevant for applications that work with financial data as it provides a basis for analyzing stock prices and tracking trading activity. This table is especially interesting for future features that implement predictions on real time data.

The ER diagram, which visualizes these tables and their relationships, is critical to understanding the data model and providing a clear idea of the structure of the SQLite database. It helps developers and database administrators to recognize the relationships between entities and to design and manage the database efficiently.

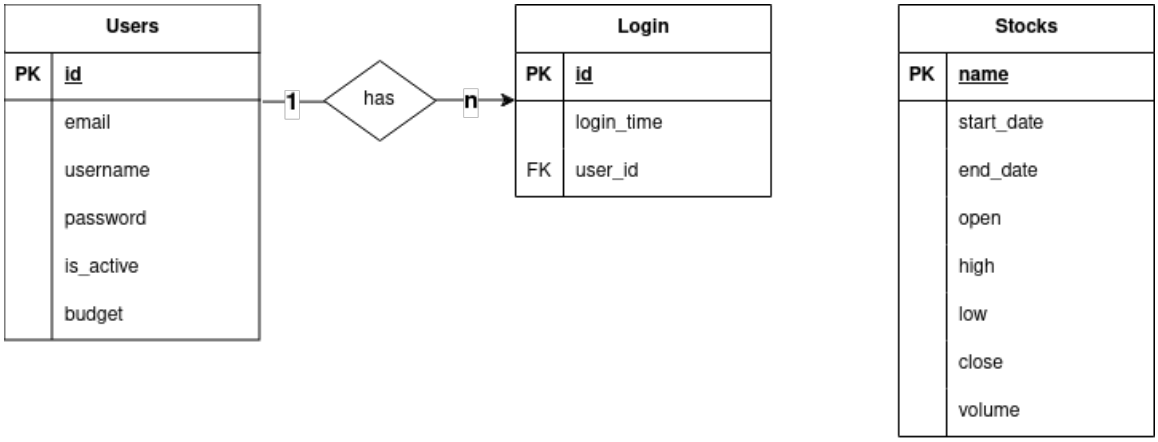


Figure 11.1.: Backend ER-Diagram

Communication between backend and frontend: Communication between the backend and frontend takes place via HTTP requests and responses. The frontend sends requests to specific API endpoints in the backend to perform the desired action, such as retrieving data, saving data or updating data. The backend processes these requests and sends back corresponding responses, which may contain the requested data or confirmation of the success of the action.

Communication follows the principles of Representational State Transfer (REST), a widely used architectural style for designing networked applications. In a RESTful architecture, resources are identified by unique URLs (API endpoints) and clients interact with these resources using standardized HTTP methods (GET, POST, PUT, DELETE).

HTTP is a stateless protocol, which means that each request from the frontend to the backend is independent and not based on previous interactions. This allows for scalability and simplicity in the design of distributed systems, but may require additional mechanisms (e.g. session management, authentication tokens) to maintain application state across multiple requests.

The exchanged data is typically formatted in standardized formats such as JSON (JavaScript Object Notation) or XML (eXtensible Markup Language). JSON is the most commonly used format due to its simplicity, ease of use and widespread support in web development frameworks and libraries.

Effective error handling mechanisms are crucial for robust communication between backend and frontend. The backend should provide meaningful error responses when requests fail due to invalid input, authorization issues or other errors. Error responses should include appropriate HTTP status codes (e.g. 400 for a failed request, 401 for unauthorized access) and descriptive error messages to help frontend developers diagnose and resolve issues.

Integration of AI models and methods: The backend provides API endpoints to call AI model methods and make requests for data processing. The AI models can be hosted on external servers or locally and are called via the API interfaces to perform predictions or analysis. The results of the AI models can then be returned to the front-end application and displayed.

The architecture of the system is based on a clear separation of front-end and back-end, which enables a strict separation between the user interface (front-end) and the application logic and database access (back-end). This architectural structure offers numerous advantages that have a positive effect on the performance, maintainability and security of the overall system.

11.3.1. Backend components

The backend consists of various components that work together to ensure the functionality of the system:

FastAPI Server: The FastAPI server forms the heart of the backend and acts as an interface between the frontend and the various backend services. It receives HTTP requests from the frontend, processes them and sends back the corresponding responses.

Database Access Layer: This component is responsible for interacting with the SQLite database. It enables the storage, retrieval and updating of data in the database according to the requirements of the applications.

Authentication and Authorization Component: This component provides mechanisms to authenticate and authorize users. It verifies the identity of users and grants access based on their authorizations and roles.

Controllers and Services: These modules contain the actual business logic of the application. They process incoming requests, retrieve data from the database as required, perform necessary operations and send the results back to the FastAPI server for forwarding to the front end.

12

Microservice Architecture

> When services are loosely coupled, a change to one service should not require a change to another.
>
> *Sam Newman*

This chapter briefly explains the implemented microservice architecture and highlights its advantages.

12.1. Advantages of microservice architectures

This section outlines the key benefits of implementing a microservice architecture. Firstly, it is important to examine the key attributes of microservices. They are independent, as aptly summarized in the quote above. Each component of a microservice architecture can be developed, deployed, utilized, and scaled without reference to other services that make up a product. Services do not have to share code with other services and will communicate with each other via API endpoints.

In addition to independence, microservices are specialised. Each microservice has a unique set of functions required to solve a specific problem. If a service grows over time, it can be split into multiple services again. Working with smaller services is often easier, particularly for continuous development and debugging. This is because smaller services can be more manageable and less complex. It is important to consider the benefits of smaller services when designing a system architecture.

12.1.1. Agility

Microservices support the organization of small and independent teams, each working on individual services. Teams work in a small and defined context and have the opportunity to work self-responsibly and more efficiently. Due to that development cycles will be shortened.

12.1.2. Flexible scaling

Microservices offer a significant advantage because each service can be scaled independently to provide an ideal solution for its task. This will allow for adjustments to be made to the infrastructure requirements as necessary. Also, the costs of a function can be measured correctly. Therefore, availability can be guaranteed even as demand for a service increases.

12.1.3. Straightforward deployment

Microservices offer a means of achieving continuous integration and deployment. It is easier to test new concepts and discard them if they do not work. The limited amount of downtime costs create an ideal environment for experiments, simplify code updates, and reduce the time required to release for production.

12.1.4. Technological versatility

Microservices do not offer universal solutions as teams are free to use the best tool to solve their specific problem. It is also important to note that each service can use a different version of the tool without interfering with other services. A large project may require the use of multiple versions of a tool for different services.

12.1.5. Reusable code

When utilising microservices, it is feasible to use the same function for multiple purposes if it is its own microservice. A service written for a specific task can also be used for another task. Therefore, a project team can provide a new product with hardly any new code if most of the required services have already been developed for another product. A metaphor that aptly describes this situation is that of a puzzle. If the team already has access to all puzzle pieces, they can solve the puzzle without developing new pieces.

12.1.6. Resilience

The independence of services also increases the resilience of a software product. In a monolithic architecture, an uncaught error in a single service can lead to the failure of the entire software product. If an error occurs within a microservice, instead of causing the entire system to fail, it will only affect the functionality that the microservice provides to the software product.

12.2. Splitting the code into containers

Initially, the architecture consisted of two components, which remained unchanged until after the MVP presentation. However, it was later decided to split the backend into multiple components. It was realized that a backend containing every possible model type had multiple limitations. The service was too large to maintain properly. Additionally, it was difficult to synchronize the requirements for different model types. It is important to ensure a manageable size for ease of maintenance and to establish clear and consistent requirements for all model types. Some model types require a specific version of a package, which may be impossible to achieve if another model requires a different version of the same package. However, it is important to ensure that all models have access to the required package version. It also was found that it was also very difficult to run TensorFlow and PyTorch in the same environment without certain conflicts. As a result, an architecture was developed consisting of a Frontend-Service, a main Backend-Service, and smaller services for each model type. This Section will explain the main services of this architecture.

12.2.1. Model container

The following model containers are currently implemented:

- ANN (Artificial Neural Network) - contains the model types GradientBoostedTree and RandomForest
- CNN (Convolutional Neural Network)
- LSTM (Long Short-Term Memory)
- Reinforcement Learning
- Transformer

The Predictioninterface of the included model is contained within the containers, making them suitable only for inference. The container communicates via an API endpoint, which can typically be accessed at http://predict_<modelname>:8000/predict, while being a member of the network specified in the Docker Compose file. To call the CNN Container, use the following format: http://predict_cnn:8000/predict.

Endpoints can also be accessed by using the following URL on the machine where the Docker Compose is run: http://localhost:<mapped port specified in the Compose file>/predict.

12.2.2. Backend container

The Backend container serves as the primary communication device between the Frontend and models. It forwards requests from the Frontend to the desired model API endpoint and includes endpoints for user management, data loading, and other use cases.

Additional information can be found in section 11.

12.2.3. Frontend container

The Frontend container serves as the primary interface for users to interact with the proposed Trading Bot. The platform consists of multiple subpages that facilitate reinforcement learning simulations, model and stock comparisons, and provide information on the available model types and stocks for use with the Trading Bot.

Additional information can be found in section 10.

12.3. Docker-Compose files

This table displays the relevant docker compose keywords that have been used and their corresponding meanings:

Docker-Compose keywords	
Keyword	**Description**
network related keywords	
http:// networks	Specifies the network used by the Docker containers to communicate with each other.
driver	Specifies the network driver for the Docker network.
service related keywords	
services	The 'Services' section in docker-compose is used to define the various services that constitute your application. Services are essentially containers in production.
build	The 'build' command is utilised to construct an image from the specified build context.
context	This directory contains the Dockerfile and is used as the context during the build of the Docker image.
dockerfile	The docker file name to use should be be specified relative to the build context.
ports	ports is used to map a port from the local machine to inside the docker container. The syntax for this command is <host-port>:<container-port>.
volumes	Paths or named volumes can be mounted using Volumes. The syntax follows the pattern of <host-path>:<container-path>.
entrypoint	The entry point specifies the command that is executed when the container starts. This is a command to run your program.
depends_on	The 'depends_on' keyword expresses dependency between services, resulting in two effects: docker-compose up starts services in dependency order, and docker-compose up SERVICE automatically includes SERVICE's dependencies.
networks	This keyword is used to specify the networks a service is connected to.

Please refer to the 'docker-compose.yml' file in the project's root directory for an example of how to use it.

12.4. Connecting new models to the existing code, a guide.

To add more models, follow these steps: Please refer to the 'readme.md' file located in the root directory of the project repository. This file contains useful information about the project. The directory containing the codebase for the new model should be named after the model being developed and located The directory should be located in the project's root directory. After developing and training the model, it must be connected to the microservice architecture. There is a prediction interface available in its dedicated branch in the git repository. This interface was used by all implemented models.

After developing the prediction interface, it is necessary to specify an API endpoint. The use of the Python package fastAPI is recommended for development. The API file should import the previously specified prediction interface.

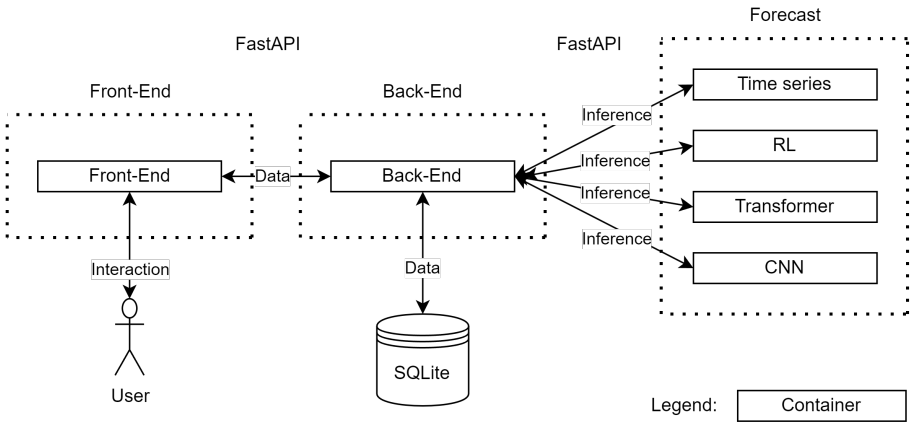
When the prediction endpoint is present, add an additional API endpoint to the main backend API that forwards the request to the specified endpoint.

The final steps involve adding the model results to the frontend to display the prediction outcomes and incorporating these predictions into the reinforcement learning ensemble to enhance the trading bot's capabilities.

Please refer the following table which shows useful links to debug existing and future API endpoints.

Helpful links for API development	
URL	**Description**
http://localhost:8080	URL of Frontend
http://localhost:8000/docs	FastAPI generated Site to test main backend API Endpoints
http://localhost:8001/docs	FastAPI generated Site to test transformer container API Endpoints
http://localhost:8002/docs	FastAPI generated Site to test CNN API Endpoints
http://localhost:8003/docs	FastAPI generated Site to test ANN API Endpoints
http://localhost:8004/docs	FastAPI generated Site to test reinforcement learning API Endpoints
http://localhost:8005/docs	FastAPI generated Site to test LSTM API Endpoints
Note: The docs site will be automatically generated by FastAPI. It can always be accessed by the mapped port in the Docker Compose file or on port 8000 when run outside of a Docker container.	

Figure 12.1.: Containerization



The microservice architecture was designed and developed by Philipp Duwe, André Galczynski, and Luca Nickel.

Chapter 12 was authored by André Galczynski.

13

Conclusion and Outlook

13.1. Benefits

The benefits of this research project are examined in two dimensions. The practical benefit describes the suitability of the implemented software to trade profitably on the stock exchange. The scientific benefit describes the suitability of the findings from the project and the implemented software to forecast time series with machine learning and to make decisions based on the predictions with reinforcement learning. *(Written by Philipp Duwe)*

13.1.1. Practical benefits

The RL agents based on the forecasts of the models appear to be able to trade economically. In the simulations that we can run at the moment, profit is being made. However, the conditions are not close to reality at the moment. In order to make a more realistic assessment of the software, it would have to be used to trade on a demo portfolio.

A decisive factor that we have not taken into account in this project is the trading costs. [16] The spread in particular plays a major role here. The spread is the trading margin between the buy and sell price. As the securities considered in this project are all very liquid, the spread is manageable, but the costs add up. Outside of trading hours, higher spreads may occur in off-market trading. In addition, there are fees for buy and sell orders, which also incur trading costs.

13.1.2. Scientific benefits

From a scientific point of view, this project is an advanced approach to developing a trading bot. State-of-the-art deep learning and machine learning models are used to predict share prices. These currently generate time series predictions of a reasonable quality. Since stock price developments are characterized by an unfavourable signal to noise ratio, this is a convincing result. The procedure of using the developed models as an ensemble also corresponds to the current state of research in order to improve the prediction quality. Finally, the decision is made by a reinforcement learning agent. The project combines classical supervised learning to train the models for the predictions with reinforcement learning to make the decisions. The project is therefore an exciting experiment in the very active field of artificial intelligence-based trading algorithms. Above all, the ease with which the project can be expanded makes it very suitable for further research and experiments.

13.2. Outlook

In the future, other project groups could further develop this project. This could include adding more models and stocks that are state-of-the-art. The reinforcement learning model can be extended with new methods for trading like deep Q-learning (DQN). Other ideas would be to implement real live trading and using real money, to have a working trading website. More information could be delivered about the companies like live news that could have an impact on the price and value of the stock. A possible extension of the project would be a portfolio feature instead of just trading single stocks. Completely new additions to the website with other AI approaches could enhance the user experience. AI features like an AI chat to help the user with questions or automated data storytelling, to explain users the stock behavior could be implemented. More transparent decisions of the model with explainable AI approaches could improve trust in the whole trading process. A feature could be to upload own models to the website and have something like a store, where you can view and purchase models of other users. This could be combined with the option to choose some parameters and train your own models with the help of the website. The list of possible extensions to the website is pretty big and since the base for the website is implemented now, it would be an interesting project to be further developed by future project groups.

Written by Thomas Parchmann

A

Appendix

A.1. Plots of distributions of volume values after normalization

Figure A.1.: Distribution of volume values after applying min max scaling.

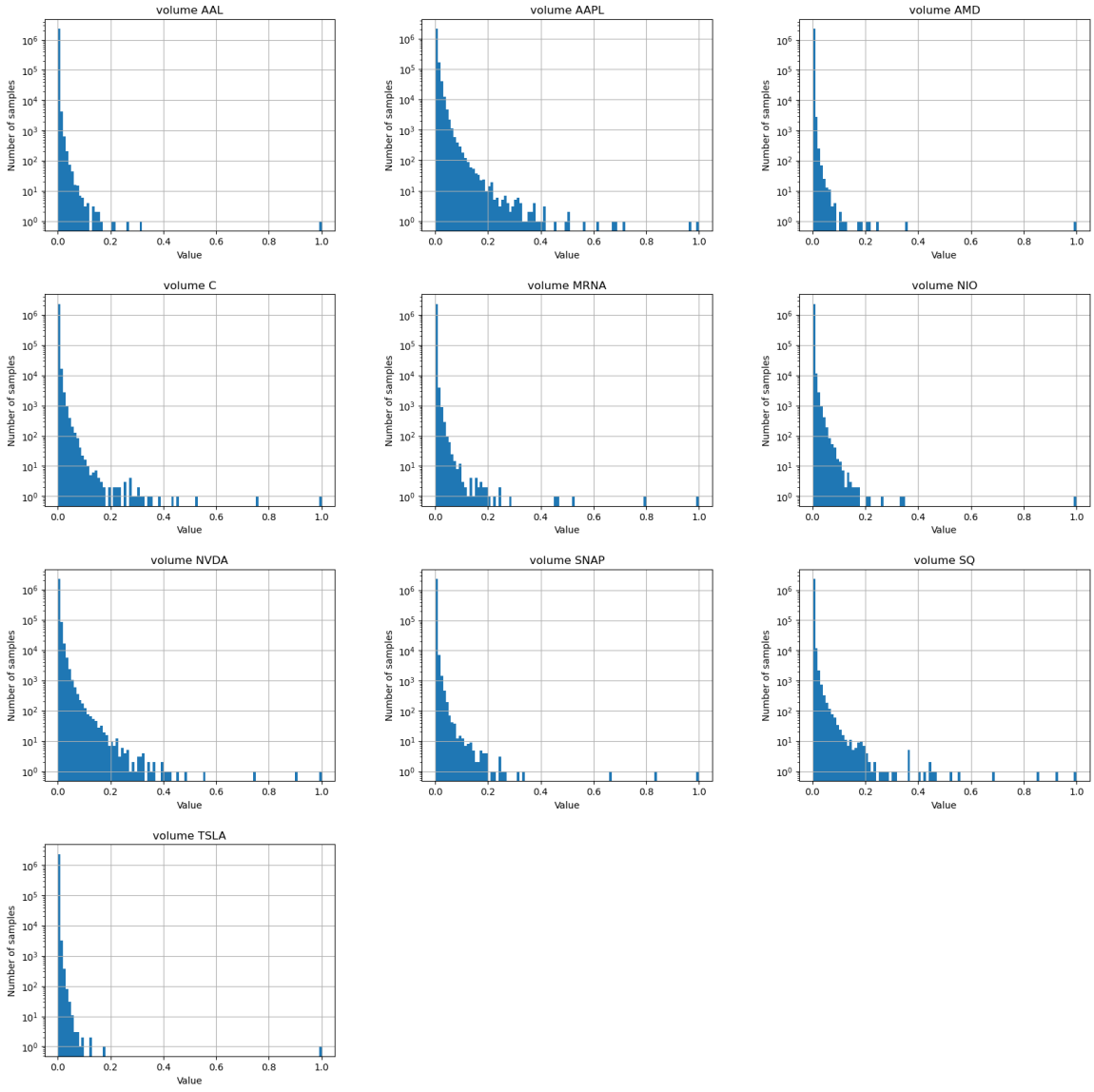


Figure A.2.: Distribution of volume values after applying standard scaling.

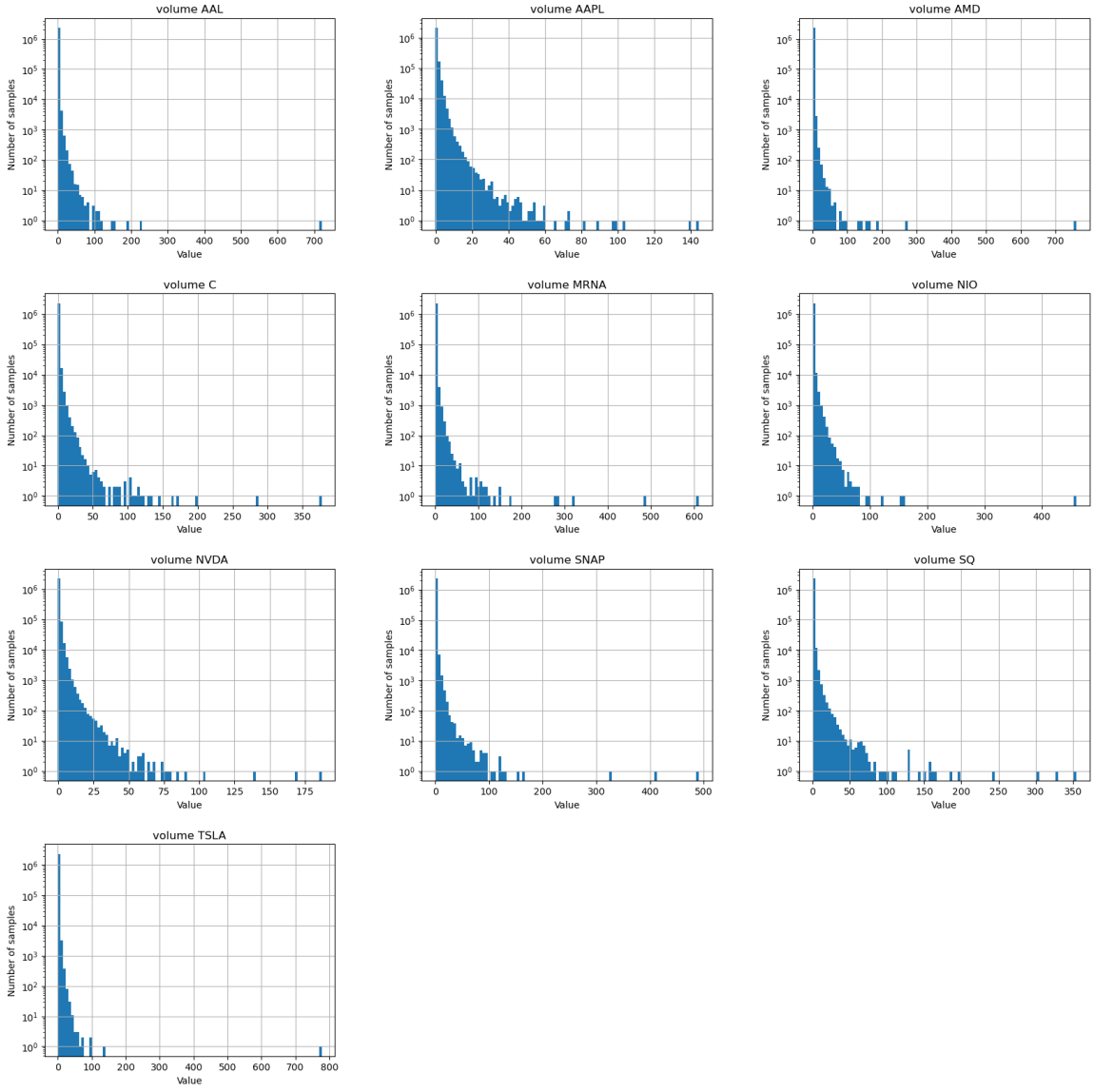


Figure A.3.: Distribution of volume values after applying power transformation.

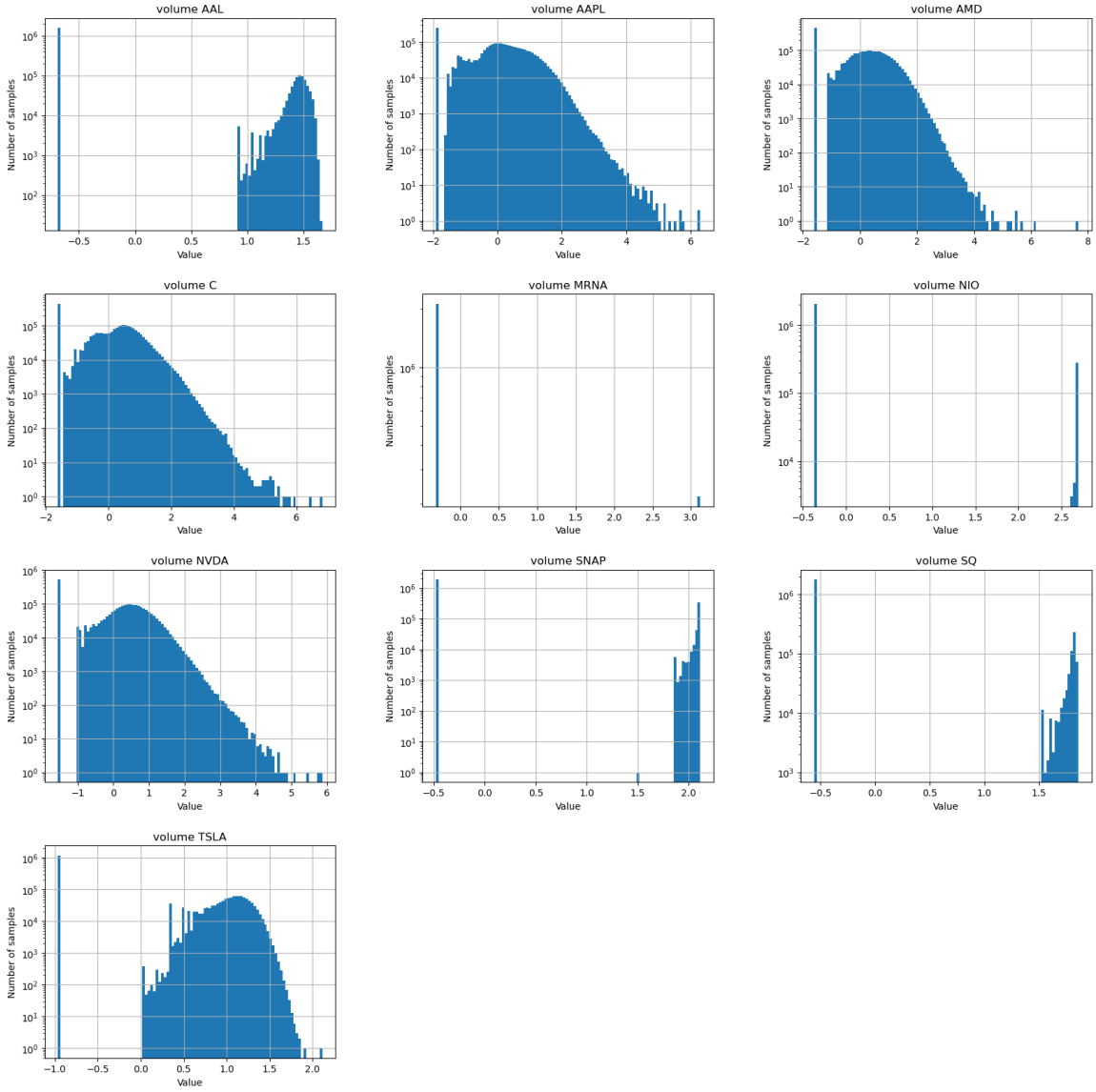
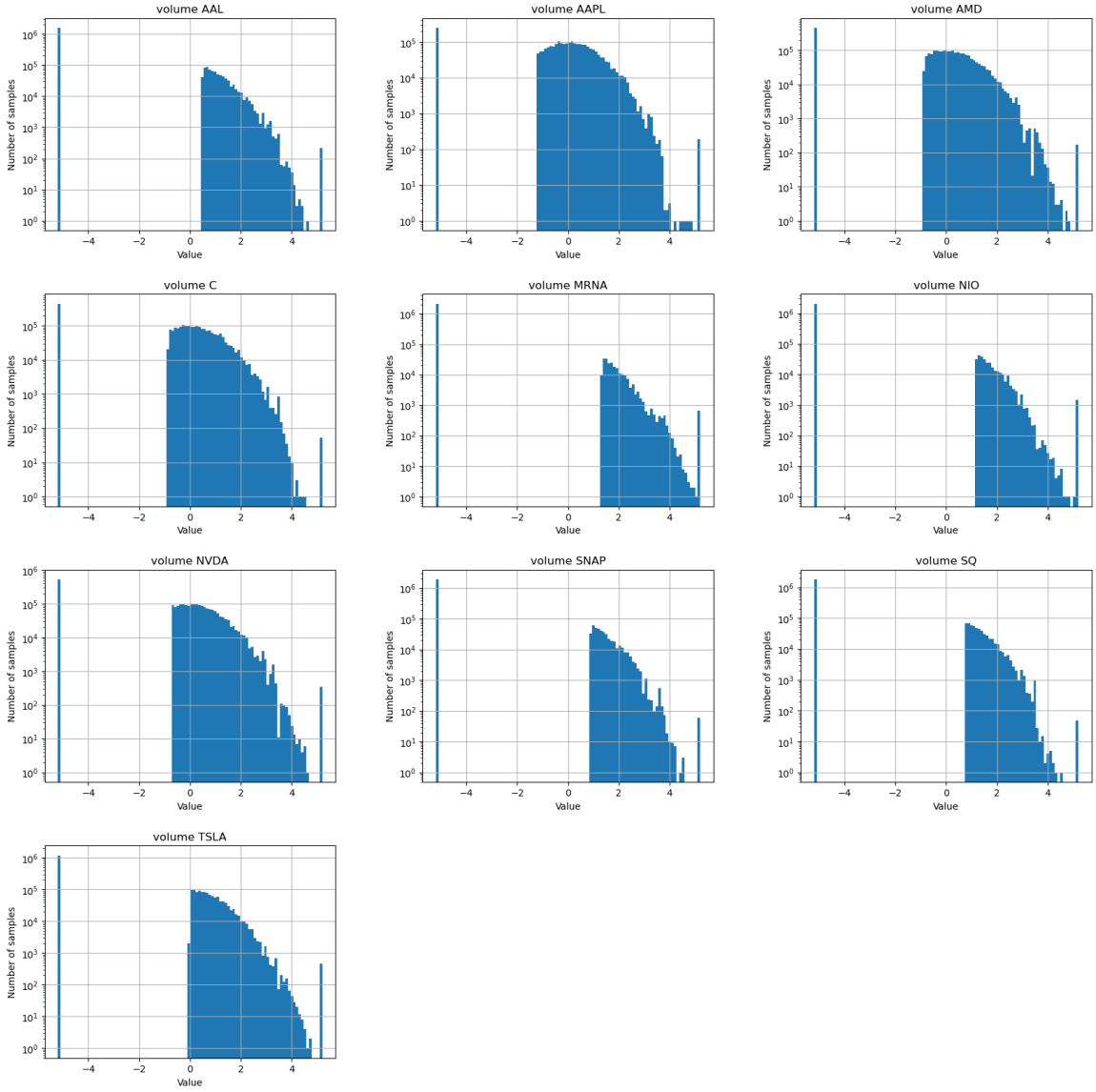


Figure A.4.: Distribution of volume values after applying quantile transformation.



Sources

[1] Prof. Dr. Hendrik Annuth. *Time Series*. Presentation slides. Part of the lecture Machine Learning. 2022.

[2] *Chat GPT von Open AI*. URL: https://openai.com/blog/chatgpt.

[3] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[4] *Github-Repository als Anhang des Programmcods*. URL: https://github.com/.

[5] Jeremy Howard. *fast.ai course lesson 16*. URL: https://course.fast.ai/Lessons/lesson16.html.

[6] Jeremy Howard. *fast.ai course lesson 18*. URL: https://course.fast.ai/Lessons/lesson18.html.

[7] ... Imene Mitiche Gordon Morison. *Imaging Time Series for the Classification of EMI Discharge Sources*. Sept. 4, 2018. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6163566/ (visited on 03/11/2024).

[8] Prakhar Mishra. *5 Outlier Detection Techniques that every "Data Enthusiast" Must Know*. July 2021. URL: https://towardsdatascience.com/5-outlier-detection-methods-that-every-data-enthusiast-must-know-f917bf439210.

[9] NIXTLA. *Auto-ETS Model*. URL: https://nixtlaverse.nixtla.io/statsforecast/docs/models/autoets.html#model.

[10] Marco Peixeiro. *Theta Model for Time Series Forecasting*. Nov. 2, 2022. URL: https://towardsdatascience.com/theta-model-for-time-series-forecasting-642ad1d00358.

[11] *TransformerEncoderLayer*. https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoderLayer.html. Accessed: 2024-02-21.

[12] Unkown. *GramianAngularField*. Jan. 4, 2023. URL: https://pyts.readthedocs.io/en/stable/generated/pyts.image.GramianAngularField.html (visited on 03/10/2024).

[13] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[14] Zhiguang Wang and Tim Oates. *Imaging Time-Series to Improve Classification and Imputation*. June 1, 2015. URL: https://arxiv.org/pdf/1506.00327.pdf (visited on 03/11/2024).

[15] Eric W. Weisstein. *Gamma Distribution*. URL: https://mathworld.wolfram.com/GammaDistribution.html.

[16] *Wie funktioniert der ETF-Handel?* https://www.justetf.com/de/academy/wie-funktioniert-der-etf-handel.html. Accessed: 2024-02-23.

[17] Tony Yiu. *Understanding ARIMA (Time Series Modeling)*. Apr. 26, 2020. URL: https://towardsdatascience.com/understanding-arima-time-series-modeling-d99cd11be3f8.

[18] George Zerveas et al. "A transformer-based framework for multivariate time series representation learning". In: *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery and data mining* (2020). arXiv: 2010.02803. URL: https://arxiv.org/abs/2010.02803.

Declaration of Honor

We solemnly swear that we have independently authored this present work and have not used any resources or aids other than those specified. All direct or indirect quotes or thoughts from external sources are appropriately marked as such.

Sources from the lecture content at FH-Wedel by Professor H. Annuth have not been cited. Self-created graphics and recordings are provided without source citation. The developed program code can be viewed on GitHub at source [4]. All other sources used – including books, online sources, and others – are listed in the bibliography.

For the grammatical review of the text and support in the creation of source code, we have used AI-based tools in accordance with [2].

André Galczynski, Aykan Güney, Djauschan Fedaie, Eira Hammerich, Jasmina Jäkle, Joel Wälchli, Johann Beaumont Grigjanis, Kevin Stelke, Luca Nickel, Niklas Kormann, Philipp Duwe, Thomas Parchmann, Umut Kurucay