Is incorporating data from various sources beneficial for time series predictions on stock prices using neural forecast methods?

Jan Koos Assheuer (8424110)
Henricus Pieter Bracht (8447941)

Frankfurt School

A thesis presented for the degree of
Bachelor of Science

Frankfurt School of Finance and Management
Frankfurt am Main
15 January 2024

Acknowledgements

We would like to express our gratitude towards our Professors, Prof. Dr. Florian Ellsäßer and Levente Szabados, for supervising the thesis and assisting us in preparation. We also extend our gratitude to Gerhard Müller for his inspiring music, which provided continual motivation and encouragement during the challenges encountered in our technical development.

Abstract

In this thesis, we investigate the impact of incorporating data from various distributions on forecasting accuracy on stock prices using neural forecast methods. Our study primarily investigates the application of transfer learning in time series forecasting to facilitate stock market price predictions. We have trained several models, built up using distinct architectures and enriched with differing data including weather and traffic based time series data. We compare them to the baseline models ARIMA and TimeGPT. In this fashion, we conducted extensive hyperparameter tuning and a comprehensive analysis of model performance based on various metrics. Key findings give insights into the benefits and limitations of integrating diverse data sources in predictive modeling for financial time series.

Contents

Chapter 1

Introduction

Uncertainty has always been a challenge for humans to overcome. On a societal level, uncertainty can lead to economic and political instability. When individuals and institutions are unable to predict or manage the future, it can interfere trust and confidence. Mitigating uncertainty is therefore incessant. Time series analysis is a method to analyze and interpret data points over time and can be traced back to early pioneers like Yule 1927. Time series forecasting, often serving the purpose of predicting economic indicators or business dynamics, has proven thereby to be crucial to economic and business analytics.

Early applications were in the fields of studying dynamic structures, investigating relationships between variables and performing adjustments for seasonal variations of economic data, especially cyclical developments, for instance recession or expansion. A milestone was the publication of "Time Series Analysis: Forecasting and Control" by Box and Jenkins 1970 introducing a systematic approach to analyze time series, namely the ARIMA model that revolutionized time series forecasting and is still widely adopted today.

Originally, there have been two schools of thought in the history of time series analysis. One approach was the frequency domain approach, the other was time domain approach (Tsay 2000). In a nutshell, the time domain approach, which ARIMA belongs to, analyzes signals based on their values over time. The primary focus is on how the signal changes over time. The frequency domain approach, on the other hand, focuses on the rates of change that constitute the signal.

Up until recently, it seemed that Deep Learning models were not feasible for time series forecasting as the winners of forecasting competitions were predominantly statistical models (Makridakis, Spiliotis, and Assimakopoulos 2018). With the introduction of the NBEATS model (Oreshkin et al. 2020), there has been a Deep Learning model that can consistently outperform the purely statistical or hybrid statistical/Machine Learning or statistical/Deep Learning model.

With the current development of foundation models that are particularly strong because of their ability to use Transfer Learning (Bommasani et al. 2022) and that have proven successful with Transformer models (Vaswani et al. 2023) particularly for Natural Language

Processing, it is worthwhile to also investigate Transfer Learning capabilities in the field of time series analysis.

The potential integration of Transfer Learning in time series forecasting can display a valuable development to mitigate contemporary challenges. Transfer Learning, in a nutshell, is using knowledge acquired in the training of one source domain to apply in a new target domain.

In this thesis, we aim to explore the application of Transfer Learning in time series forecasting in the context of stock price predictions. We incorporated data from various sources, including weather data, traffic data, energy consumption data and more. We will first review research that has been conducted to this point, then explain how we built the model and what metrics we used, further explicate how we trained the models for our experiments and finally present the results with our conclusions.

We do so by training 5 different models that are to different degrees enriched with additional data, accompanied with extensive hyperparameter tuning. Moreover, we compare these models mutually, but also to two other baseline models, ARIMA and the recently published foundation model TimeGPT.

Chapter 2

Literature Review

2.1 NBEATS (Oreshkin et al. 2020)

2.1.1 Context

For a considerable long time, it seemed like machine learning and/or deep learning models had issues with outperforming the classical statistical ones (Makridakis, Spiliotis, and Assimakopoulos 2018) in time series forecasting. Models, like ARIMA (Ho and Xie 1998) or the Theta Model (V. Assimakopoulos and Nikolopoulos 2000) were performing better and up until recently in 2020, still the winner of big forecasting competitions like for instance M4 was a hybrid approach of a statistical and a machine learning model (Smyl 2020). An additional challenge arises in the nature of deep learning models: It is hard to interprete them and to understand the rational of the model behind decision driving factors (Castelvecchi 2016). The paper introduces a new model, **N**eural **B**asis **E**xpansion **A**nalysis for Interpretable **T**ime **S**eries Forecasting, for short NBEATS. Thus, the paper's objective is to solve two problems: Create a pure deep learning model that:

(A) outperforms statistical and hybrid models and

(B) is interpretable and understandable.

To measure the success of the model, it is imperative to compare the forecast denoted as $\hat{y}$ to the original values y. There are four metrices that are used in the paper to measure that difference i. e., the loss metrices they are relating to when comparing their model to other models. They can be found in our appendix A.1.1.

2.1.2 Architecture

The model has 3 different objectives:

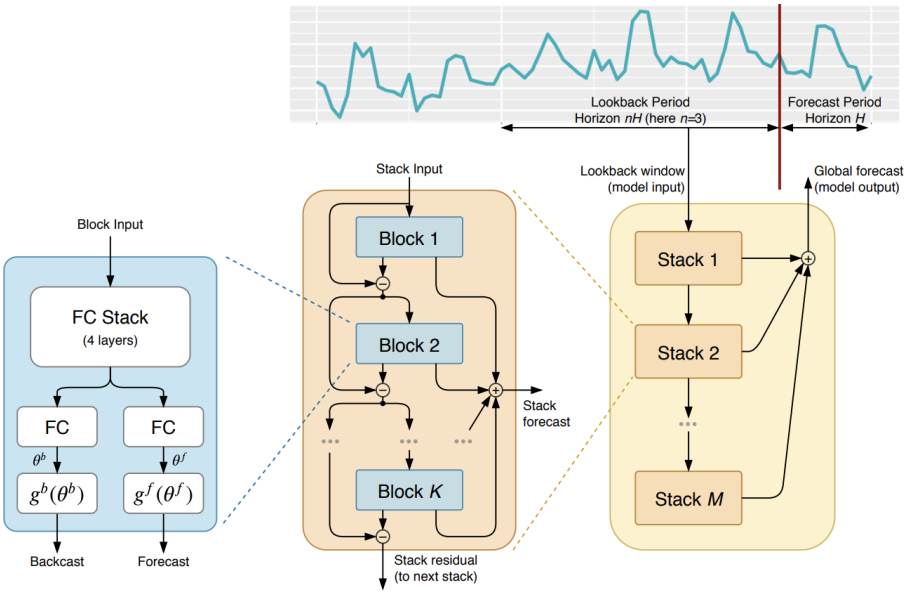

Figure 2.1: NBEATS Architecture visualized (Oreshkin et al. 2020)

1. It should be simple and generic, but also potentially deep.
2. Generality and flexibility. It must **not** be time-series specific.
3. Interpretability.

The superordinate component of the model are stacks. Each stack has two outputs: A residual, that will be passed on into the next stack and the forecast of the stack itself, which is summed up as the model output.

$$\hat{y}_{\text{Model}} = \sum_{s}^{S} \hat{y}_s$$

Going deeper into the model, each stack consists of blocks. Blocks, as well, have two outputs adhering to the double residual stacking principle introduced in the paper. In contrast to the hard-to-interpret classical residual networks, where the input is added to the output of a layer before passing it on to the next component, we have a partial result after each block, making the result better interpretable. The doubly residual stacking works the way that that the input of the block is deducted by the backcast of the block. The output of all blocks are summed up resulting in the complete stack forecast.

$$x_\ell = x_{\ell-1} - \hat{x}_{\ell-1} \qquad \hat{y}_s = \sum_{\ell} \hat{y}_\ell$$

Moreover, the block forks the input into two parts: One forecast and one backcast. In both cases, a fully connected layer estimates the expansion coefficients θ_ℓ^f for the forecast and θ_ℓ^b. These expansion coefficients are then plugged into a function, $\hat{y}_\ell = g_\ell^f(\theta_\ell^f)$ and $\hat{x}_\ell = g_\ell^b(\theta_\ell^b)$ respectively, that then outputs the forecast/backcast. More accurately:

$$\hat{y}_\ell = \sum_{i=1}^{\dim(\theta_\ell^f)} \theta_{\ell,i}^f \mathbf{v}_i^f \qquad\qquad \hat{x}_\ell = \sum_{i=1}^{\dim(\theta_\ell^b)} \theta_{\ell,i}^b \mathbf{v}_i^b$$

$\mathbf{v}_i^f$ and $\mathbf{v}_i^b$ are the basis vectors for forecast and backcast,
$\theta_{\ell,i}^f$ is the i-th element θ_ℓ^f.

It is noteworthy that the backcast's purpose is not to estimate the already given data, but to help the blocks further in the stack to disregard components unhelpful for their forecasting estimate.

In terms of interpretability, it refers to the ease of understanding and extracting meaningful insights from the model's predictions. The paper therefore creates two models, one is generic and one is interpretable. The latter incorporates domain-specific knowledge (trend and seasonality) to provide clearer and more transparent outputs, in contrast to the generic model, which lacks these specific constraints and may produce less interpretable results. As a regularization technique, ensembling is used.

2.1.3 Results

The model was compared to different approaches:

- Statistical models
- Pure machine learning
- Hybrid of machine learning and time series analysis methods
- Deep learning and time series analysis combinations

The N-BEATS model exhibits state-of-the-art performance on diverse time series datasets, including the M4 dataset, surpassing models with hand-crafted features and architectural adjustments, demonstrating that deep learning can achieve impressive forecasting results without the need for extensive prior knowledge or domain-specific engineering. The outputs of a proposed model in both generic and interpretable configurations are compared, demonstrating that by incorporating specific constraints, such as polynomial and Fourier basis structures, the interpretable architecture produces more interpretable outputs without compromising performance.

Interestingly, N-BEATS can perform "meta learning": Meta learning is a machine learning paradigm where models are trained to learn from various tasks in a way that enables them to quickly adapt to and generalize well on new, unseen tasks. The outer learning i. e., generalization learning, takes place in learning the parameters of the whole network with gradient descent. The inner learning captures the learning within the basic building blocks, particularly by changing the expansion coefficients. This effect of the meta learning can be proved with better generalization performance upon adding more stacks and blocks. The findings of the paper support two key hypotheses: generic deep learning performs exceptionally well on heterogeneous time series forecasting without domain knowledge, and deep learning models can be constrained to provide interpretable outputs. Additionally, the study suggests that NBEATS may exhibit a form of meta-learning, opening possibilities for further exploration.

2.2 Transfer learning (Hosna et al. 2022)

2.2.1 Intro

Transfer learning has received a lot of attention from researchers. Traditional machine learning algorithms use a limited data distribution and often solve the problem suboptimally. This issue can be mitigated using transfer learning, where we add more data to our dataset resulting in better output with efficient results. Analogous to normal learning, certain concepts or patterns of behavior can be applied also with two different exercises e. g., balancing your weight on a motorbike or on a bicycle. This aspect of learning inspired transfer learning. Usually, machine learning tasks have been addressed in isolation, transfer learning tries to bridge source and target task. Thereby, it attempts to understanding a new task based on a related one performed in the past. Transfer learning is most worthwhile to apply when there is a limited supply of data. Possibly, a generalization takes place, which makes the model feasible for solving multiple types of exercises. However, one should be careful: If the target and source tasks are not compatible, there is a negative transfer leading to worse performance.

2.2.2 Related work

Data mining and classical machine learning have been widely applied for different tasks, using training and testing data on similar data distribution inputs. A loss metric is applied to alter the model in its parameters, such that we achieve a better performance measured on this parameter. It is a challenging and costly task to find appropriate training data to match the algorithm's actual purpose. Consequently, the adoption of transfer learning first became evident in a top-level learner, that has already been trained and improved on a related field. In the next step, this top-level learner can then be specifically trained upon our required task. There have been multiple attempts to describe transfer learning since

1995. However, transfer learning became more attractive when large-scale and free data sources became available.

2.2.3 Definition

According to the paper, the definitions of transfer learning comprise the reuse of already existing models in order to address new challenges, including using previous knowledge to improve performance in new tasks. Contrary to common machine learning models that are trained and developed from scratch, transfer learning utilizes knowledge from related tasks. Furthermore, transfer learning can be otherwise defined as training new models with already trained models of previous tasks. Transfer learning can be conceived as a complement to machine learning. There is no need to start from scratch, increasing training speed, increased accuracy when resources are limited and enabling simulation based training.

2.2.4 Techniques in transfer learning

The paper now investigates the questions:

- What data should one transfer: Maybe certain concepts are not feasible to transfer.
- How should one transfer: Choice of a learning algorithm.
- When to transfer: When does the transfer of knowledge of other domains facilitate the models training and performance capacity?

Furthermore, the techniques are divided into certain categories:

- Inductive transfer learning: Consists of two subcategories based on similarity of source and target domains and availability of labeled information:
  - Inductive multi-task learning: Source and target domain of the data are the same and there is a lot of labeled data. However, it differs insofar from traditional multi-task learning, that only performance of the target domain is taken into account.
  - Self-taught inductive learning: Source and target domains are related, but there is no labeled data available.
- Transductive transfer learning: The data domains are different, yet the task is the same.
- Unsupervised transfer learning: Target and source tasks differ, both target and source domains consist fully of unlabeled data.

2.2.5 Sample selection

For transfer learning, it is crucial to select the right data samples to build a successful model. Firstly, one needs to ensure that there exists a logical connection between data used for training and target tasks. Secondly, it is important to be cautious not to overload the model with too many parameters from the source data. While there might be many attractive and potentially useful variables in the source data, it is crucial to focus on what is truly important for the target task.

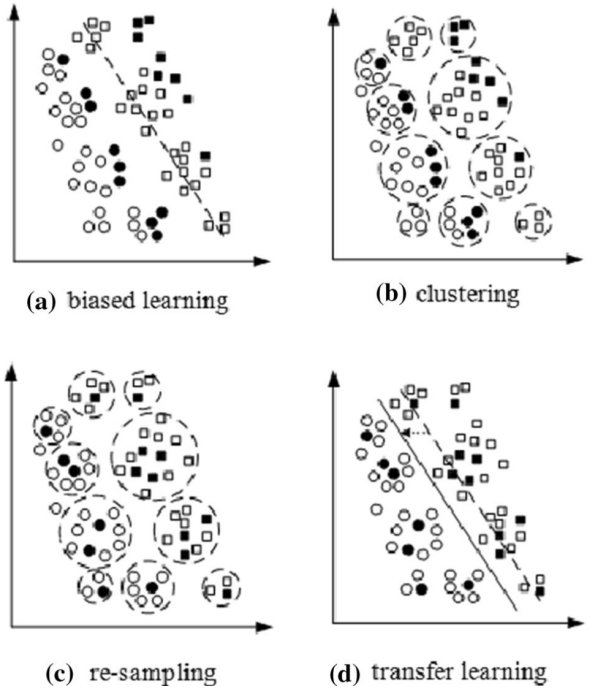

Figure 2.2: Sample selection by transfer learning (Hosna et al. 2022)

Sample selection bias

One of the most complex challenges when applying transfer learning is the sample selection bias that could be inferred in differences of present data in comparison to future data. Small sampled groups can therefore imply certain patterns for larger groups, which will cause bias suffering datasets. There are some strategies that can be used to mitigate that issue, in particular between target and source tasks, however, bias remains an issue impacting transfer learning. Due to the significance to build effective models, sample selection and its bias received more attention from various research disciplines in the recent years.

Sample selection algorithms

One mitigation to address the sample selection bias are estimation models. In particular, there is the kernel mean matching model with which one can mitigate bias from external data. The objective of such models is to minimize overfitting and improve the effectiveness of estimating data that is diverse in density and/or information. Kernel mean matching is among other models like the unconstrained least-square importance fitting an advancement in covariate shift estimation models. It considers a labeled data distribution, estimates prediction loss between source and target data and employs penalty criteria to incorporate relevant information. The workflow comprises transductive transfer learning (see subsection 2.2.4), empirical risk minimization and optimizing parameters for the target domain.

Applications of sample selection

Sample selection has diverse fields of application to match requirements that samples are balanced among groups if different gender, race, and health conditions. In tech, sample selection can help in detecting evolving spam patterns or cyber-attacks by adapting to new threats. Old data can create inefficiencies because it might not be able to cope with modern threats. In social science surveys, different belief systems can be investigated with the help of sample selection. A significant challenge in these sample selection applications is the deviation from a fundamental machine learning principle: the training and testing data should come from the same distribution. Because sample selection is so widely used, it implies many potential biases in real-world datasets. Although a lot of research has been conducted on sample selection bias, there is yet no way to quantify and rectify it accordingly. (Zadrozny 2004)

2.2.6 Domain adaptation

Domain adaptation as a part of transfer learning is the concept of using a model trained on a specific domain to work on another domain. The challenge hereby is to bridge the gap between the data distributions and enhancing generalization capabilities.

2.2.7 Applications of transfer learning

- Manufacturing and Processing: Efficiency can be increased with lower costs and higher product quality.
- Robotics and Simulation: Simulated data can facilitate the training of robots, the "transfer" takes place from simulation to real world.
- Gaming: AlphaGo, for instance, was trained on multiple games and was not restricted to a single one, therefore it is transferring knowledge from one game to the other.

- Image classification: Pretrained models are adapted to specific tasks, saving time and resources.
- Zero-shot translation: Transfer learning can translate between languages it has not been trained on by pivot languages for indirect translations.
- Sentiment classification: Sentiment analysis can take place by models that were trained on large text datasets.

2.2.8 Contributions of transfer learning

- Medical sciences: Enhancing MRI and imaging analysis by adapting models trained on large annotated datasets.
- Bioinformatics: Analyzing biological data by adapting models across different organisms to address data sparsity.
- Transportation: Transfer learning adapts models trained on specific conditions to work accurately in varying conditions in traffic scene analysis.
- E-Commerce: Improves recommendation systems by transferring knowledge from dense to sparse datasets.

2.2.9 Future of transfer learning

There are multiple ways that transfer learning can be further developed. One of which is expanding application areas, whose feasibility we test in this Thesis. Handling negative transfers is another area that must be investigated to avoid negative impact on performance. Moreover, transfer learning must be investigated on its effectiveness and reliability, which is also subject to the research in this thesis. Finally, further theoretical development can be conducted, maybe researching for specific techniques. Transfer learning is a promising concept in machine learning that might make us able to leverage on pre-existing knowledge, enhancing efficiency and dealing with data scarcity.

2.3 Baseline Models

2.3.1 ARIMA

The ARIMA(Box and Jenkins 1970) model was first proposed in 1970 and is now a widely adapted statistical approach for time series forecasting. It consists of and combines AutoRegressive (AR), Integrated (I), and Moving Average (MA) components. The functionality of the parts can be described as follows:

- Auto-Regressive: Deriving the current value from the development of prior values.
- Integrated: This refers to the differencing process to account for non-stationary data i. e., data with trends, seasonal patterns and cycles. It could otherwise potentially lead to unreliable results.
- Moving Average: Accounts for the dependency between an observation and a residual error by applying a moving average to prior observations.

The model has three parameters:

- p (lag order): The number of lag observations used for prediction.
- d (degree of differencing): The number of times the data is differenced.
- q (order of the moving average): The size of the moving average window.

2.3.2 TimeGPT

TimeGPT(Garza and Mergenthaler-Canseco 2023) is a novel approach to bring a universal pre-trained foundation model into existence that is designed for time series forecasting. It should be capable of predicting accurately time series data for a multitude of domains without requiring additional training. The aim is to have a new paradigm that is more accessible, more accurate and faster.

Background

It is currently in discussion whether deep learning approaches are effective in time series forecasting as historically, statistical models have been reliable. Recently, some deep learning models have gained popularity through simplicity and better results nevertheless (Kunz et al. 2023). The skepticism towards deep learning foundation models in time series forecasting could stem from the lack of publicly available datasets.

Foundation models are defined by their capability to generalize well across domains. There are two approaches:

1. Zero shot learning: Without training specific to the dataset that is to be predicted.
2. Fine-tuning: An already pre-trained model is fine-tuned to the domain it works in.

Model

The model takes a window of values to make the forecast and enriches it by local positional encoding. The transformer has an encoder-decoder structure. A final linear layer maps the

output to the forecasting window dimension. The target hereby is to obtain a flexible model that can deal with heteroeneity. The model was trained with over 100 billion datapoints consisting of data from numerous domains. Apart from standardization and interpolating missing values, no other preprocessing has been applied.

Evaluation

For evaluation, relative Mean Absolute Error (rMAE) and relative Mean Squared Error (rMSE) were used normalized against the seasonal naive model. As a result, TimeGPT outperforms a wide range of statistical, machine learning and deep learning models in different frequencies (hourly, daily, weekly, monthly). This is particularly evident in zero-shot inference. Moreover, TimeGPT offers significant advantages in computational efficiency and simplicity of the model.

Finally, the model's impact on forecasting practices is reflected. Foundation models can significantly change the typical forecasting processes. Furthermore, two main areas of future exploration are mentioned: Informed forecasting, incorporating knowledge about the underlying process and time series embedding, where the similarity between series are measured.

Chapter 3

Methodology

3.1 Data Pre-processing

3.1.1 Financial Data

The type of data we are utilizing is time series data: It comprises values recorded at specific time intervals, uniformly spaced with consistent time gaps between each data point in one time series. Our primary dataset comprises the daily closing prices of publicly traded stocks and ETFs. Each time series encapsulates data from the respective asset's IPO or other significant event up to a cutoff point in early 2023. Consequently, the length of each time series varies. The interval between observations aligns with the logical gap between two consecutive trading days. On weekdays this gap is 24 hours long, but one some days the gap is longer, most prominently weekends see no trading, some bank holidays like Christmas and Thanksgiving Day also see no trading. On days where there is no trading no new data is generated and datasets do not provide values for these days. To tackle this problem there are two main approaches:

- Forward filling missing dates: This entails adding empty rows for missing dates into our dataset and filling these rows with the last known value.
- Dropping missing dates: This means excluding non-trading dates from the dataset, in each time series a trading days is succeeded by another trading day immediately, regardless of how many actual days passed.

Forward filling comes with the advantage that it most accurately represents the real world, as actors can observe the prices of assets on non-trading-days, but cannot influence them. However this feature also makes the model somewhat less stable as during inference (i.e. using the finished model to make predictions on real data) the model must not only predict the new value in the time series but also that the next day is a non-trading day. Further an underlying assumption of technical analysis (i.e. studying historical market data to identify

patterns, trends, and signals that might indicate potential future price movements) is that the price of financial assets can be determined from past data as tomorrows prices are correlated to yesterdays prices. This suggests that on non-trading days no new data is generated and thus there is no need to include this data for training and inference. We experimented with both approaches and decided to use the latter strategy of dropping missing values, however we consider both approaches valid for financial time series predictions.

In total, we received daily price and volume data for 124,118 financial assets from Yahoo Finance in .csv format (Yahoo Finance 2023). Out off all data we retained only the data from 57,927 assets. This filtration aimed to ensure that the data quality is as high as possible, which is essential to obtain high performance from any AI model. Each file contains a date column and for the respective days opening price, maximum price, minimum price, closing price, volume, dividends and stock splits. The prices are adjusted for stock splits. Out of these columns we kept only the closing price time series. Among the exclusions were 1,830 time series which have data available for less than 200 trading days. These shorter time series typically originate from instruments who recently had an IPO and are unsuitable for the model's training. Because we need to sample several subsets from each time series in our dataset, these time series are too short to be usefull for training.

An additional 2,768 time series were omitted for containing zero values, which contradicts the nature of stocks or ETFs that inherently possess nonzero prices, as no rational actor would sell instruments for no financial benefit.

We also excluded 1,514 time series wherein more than 2% of values across open, high, low, and close time series were unknown or non-numeric. Linear interpolation was applied to the close time series to rectify missing data, while open, high, and low time series were disregarded as we do not save these columns. We nevertheless dropped assets where these columns had poor data availability to increase the remaining data quality.

Furthermore, 27,979 time series were removed due to inconsistencies in logic across high, low, open, and close values within a single day. Even though only the Close time series were utilized, this step ensured overall data quality.

An additional 16,711 time series were eliminated for excessive volatility, where day-to-day stock price changes exceeded 250%. This step aimed to mitigate data errors or unforeseeable events affecting the model's predictive accuracy. During early experimentation, we could observe that keeping these time series causes the model to frequently predict significant jumps in stock prices and unreasonably high volatility, which also increased the validation and training loss significantly.

Furthermore, 15,388 time series were discarded due to excessive illiquidity, where some assets remained stagnant without any price changes for extended periods, in some cases up to several years. Time series with illiquid periods of at least 35 trading days were removed to prevent the model from predicting prolonged illiquid periods erroneously.

Finally, the csv file for one asset is corrupted, preventing us from using it in the analysis.

Using both the open and close time series of the same asset would be redundant as they

are conceptually the same data, just offset by one day, as today's closing price is very close to tomorrows opening price. The volume and stock split time series are too volatile and mainly filled with zero values respectively, these attributes makes them too different from daily price data to make meaning full transfer learning between patterns from both data distributions possible.

By solely utilizing the close time series, we aimed to streamline training by minimizing redundant patterns, thereby significantly accelerating the process as the model does not have to consider an instrument's close, high and low separately. We considered an altered methodology where we treat one assets close, high, low and open time series as one joint multivariate (i.e. one point in time is associated with several variables) time series. However, we decided against this because even though our implementation of the NBEATS model through the darts library supports multivariate forecasting the original paper proposing NBEATS only details singlevariate forecasting. Further, this approach would complicate integrating data from other distributions, as they too would need to be multivariate which necessitates similar multivariate patterns with four attributes per time point.

After excluding all inappropriate time series we have arrived at a total of 57,927 time series, with an average length of 2758 trading days, summing up to 159.78 million data points related to the stock market.

3.1.2 Weather Data

In addition to the stock market data, we integrated information sourced from the US National Oceanic and Atmospheric Administration (NOAA 2023). Their public database comprises daily measurement records from 124,946 weather stations worldwide. These stations exhibit varied lengths based on their installation dates, spanning over a century in some instances. The recorded fields encompass a range of measurements, such as daily maximum, minimum, and average temperatures, precipitation levels including rainfall and snowfall, among other metrics. Notably, stations differ in the extent of data captured, with some recording all available measurements while others might have more limited data. Additionally, many stations have short or prolonged periods of time where they do not record some or all measurements.

A significant portion could not be utilized for training purposes. The semi-discrete nature of rain and snowfall data made it dissimilar when plotted alongside stock market data, leading us to exclude it from our analysis. On the other hand, temperature data exhibited the closest resemblance to stock market data when plotted, and a majority of stations had recorded this particular metric.

From the weather stations' daily recordings of minimum, maximum, and average temperatures, each station potentially yields three distinct univariate time series. For each weather station, we examine all three of these time series and insert NaN values for missing days. We then cut off all NaN values at the start and beginning and dropped all time series where more than 1% of the data consisted of NaN values. For the remaining time series,

linear interpolation was applied. We could observe numerous data errors characterized by abrupt deviations interrupting otherwise continuous trends, most often temperatures would decrease to 0 degree Fahrenheit even during summer months before continuing on the previous trend at much warmer levels. We replaced this outliers through linear interpolation when the difference between a day's value and the 10-day moving average exceeded 20% of the range between the highest and lowest measurements. However, if more than 3% of values would need to be replaced using this method, the entire time series was disregarded. Out of all time series that were not dropped we would then choose the longest one to include in our curated dataset, provided that the length corresponds to at least 20 days.

In total we have time series from 6331 weather stations, each describing daily temperature over time. The average length of a time series is 12,512 days and thus we have 82.97 million individual data points, somewhat less than from stock data but from significantly less individual stations.

3.1.3 Other Data

Among our potential data sources to enrich our model, we considered utilizing measurement data captured during physical exercises conducted by trial subjects, such as pulse and accelerometer sensor readings. However, these time series often exhibited prolonged constant intervals or repetitive patterns, leading us to anticipate adverse effects on the model's performance. Similar challenges were observed with illiquid stock market data, as elaborated in section 3.1.1, thus we decided against incorporating this physical exercise data. With the same reasoning we also excluded a dataset with semi-distinct time series concerning app usage hours.

We leveraged several datasets available in the darts library in Python. We chose all available datasets which look similar to stock market data when plotted and have sufficient length. This encompasses:

1. ETTh1Dataset and ETTh2Dataset (Zhou et al. 2021), which includes data from a single Electricity Transformer. The dataset is sampled hourly from 2016/07 to 2018/07 and includes load and oil temperature measurements amongst other variables. For consistency with the other singlevariate datasets, we exclusively utilized the Oil Temperature time series.

2. ETTm1Dataset and ETTm2Dataset (Zhou et al. 2021). This dataset is akin to the prior datasets but sampled every 15 minutes. As before we selected only the Oil Temperature time series from these measurements.

3. ElectricityDataset (Trindade 2015). Like the four prior datasets this dataset includes electricity data, specifically the consumer electricity usage of several households sampled every 15 minutes.

4. EnergyDataset (Jhana 2019) containing aggregated energy generation data categorized by energy type. Out of all time series we saved the time series for: biomass, fossil gas, fossil oil, other renewables, waste, fossil hard coal, hydro run-of-river and poundage, other energy, wind onshore and the total load forecast. We excluded time series that are too close to any of those named time series as we want to avoid duplicated data.

5. ExchangeRateDataset (Lai et al. 2018), featuring the daily spot exchange rate of the currencies Australian Dollar, British Pound, Canadian Dollar, Swiss Franc, Japanese Yen, New Zealand Dollar, and Singapore Dollar against the US Dollar. Due to illiquidity and some data errors, we excluded the Chinese currency from this dataset.

6. ILINetDataset (Zeng et al. 2022) sourced from the Centers for Disease Control and Prevention of the United States, portraying weekly data on influenza-like illness patients. We saved the time series containing infected cases, patient numbers, and healthcare provider figures.

7. USGasolineDataset (EIA 2023), which contains the weekly supply data of Finished Motor Gasoline in the United States from 1991-02-08 to 2021-04-30, recorded in gallons.

8. TrafficDataset (Lai et al. 2018), which contains data captured by the California Department of Transportation, describing the road occupancy rate measured by 862 sensors across San Francisco Bay area freeways. The dataset is sampled hourly over a period of two years.

We performed linear interpolation where necessary while ensuring that the data is of sufficient quality and free of obvious errors by plotting the data and handpicking the data distributions used. From the darts datasets, we obtained 893 singlevariate time series with a mean length of 17,636, this adds up to a total of 15.75 million data points, making this the smallest out of all distributions.

3.2 Data Usage

The N-Beats model requires a fixed window size and a forward window. These values denote the lengths of the individual features and target time series crafted for the training process, respectively. We have decided to run our training with different window sizes between 252 and 3000 and forward window of 14. Thus, during training, the model sequentially considers a time series of length $window\ size$ and computes the loss based on the subsequent 14 values for backpropagation.

The training function of the darts NBEATS model instance necessitates a list of time series. From each time series (which each corresponds to one financial asset, one weather station or one time series from the darts datasets), it automatically samples all available sub time series

of length $(window\ size + forward\ window)$. This approach leads to multiple instances of most data points appearing in the training or validation sets. Given our hyperparameters, this means the model encounters numerous exposures to very similar trends. While this approach is designed to minimize loss, it also leads to excessively prolonged training periods. With our hardware setup and data, training a moderately sized model for a single epoch would take several days. To address this, we specified a limit on the number of samples per time series. However, the default implementation in the darts library is somewhat limited as it selects only the n most recent samples. We adapted this sampling method to uniformly select samples from the entire length of a time series. This sampling strategy exposes the model to a broader array of data beyond recent samples, aiming to enhance generalization. The limiting factor with this approach is the break between the start indices of two consecutive time series. We experimented with this value and set it in the range of 100-300 for optimizing the hyperparameters and to 300 during final training and validation, lower values would correspond to more individual time series and slower training. We set it to 300 with regards to our hardware constraints.

Our model underwent several runs: One solely using stock market data and several others where we incorporated additional data. In both instances, the validation set comprised solely stock market data. However, for the latter experiments, the training set was enriched with time series of non-financial data of weather and the darts datasets.
To ensure an unbiased assessment of our model's performance on unseen data, we set aside 8,689 out of 57,927 stock market time series, equating to 15%, constituting the exclusive test set.

Before passing each sub time series to the models train function, we normalized it between zero and one. By performing this normalization we lose some information, namely whether the asset in question is a penny stock (i.e. stocks that are valued bellow 1 unit of the local currency) or a more valuable asset. This information could influence the prediction as cheaper assets are generally more illiquid and still have greater upwards potential than already valuable stock. We decided to perform the normalization regardless as the relationship described above does not hold in many cases and the nature of the instrument might elsewise be obtained from the price development. We also hope that this enables the model to better transfer knowledge from patterns learned from assets which are several magnitudes more expensive or cheaper than other in useful ways.

To prevent data leakage (i.e. providing the model with knowledge during training that cannot be assumed to be available during deployment), we normalize the sub time series suing the following procedure:
Let $x = [x_1, x_2, x_3, \ldots, x_n]$ be a sub time series, and $features\_len$ denote the number of features in the sub time series where $features\_len = |x| - 14$. The minimum value within the features is min value $= \min(x_1, x_2, \ldots, x_{\text{features\_len}})$ and the maximum value within the specified range is max value $= \max(x_1, x_2, \ldots, x_{\text{features\_len}})$. The normalization process can then be expressed mathematically as:

$$\text{Normalized\_List} = \begin{cases} x, & \text{if min\_value} = \text{max\_value} \\ \left[\frac{x_i - \text{min\_value}}{\text{max\_value} - \text{min\_value}} \text{ for each } x_i \text{ in } x \right], & \text{otherwise} \end{cases}$$

If we were to normalize the sub time series while considering the entire length for taking the max_value and min_value we can cause the model to gain information about the future price development of the instrument.

If a value in the target timeseries exceeds two or negative one (e.g. there is a very strong change in the assets value during the 14 days feature timeseries), we drop this features-targets combination from the training or validation data. This mitigates spikes in train and validation loss, presumably caused by data errors or exploding gradients and only applies to far below one percent of timeseries.

3.3 Model

We are using the generic architecture for our model, enabling the incorporation of multiple stacks, albeit sacrificing some interpretability compared to the architecture highlighted in the reference paper described in section 2.1, which distinctly separates the trend and seasonality stacks. All remaining model parameters underwent optimization using the Optuna library (Akiba et al. 2019) in the Python programming language. This optimization process involved iterative training on a subset of our dataset including 5% of the total data, incorporating extended breaks between window starts (300 time steps) across 20 epochs, followed by test loss computation using the trained model on the test set. Each of these iterations is referred to as a trial in the optuna library.

The input window, denoted as "input chunk length" in the darts implementation dictates the length of each time series passed to the model for training predictions. It is crucial to strike a balance here: Enough data points should be included to provide ample context for accurate predictions. However, an excessively long input window may overly weigh historical trends which may compromise the model's adaptability to current patterns. Additionally, during inference, the features time series length passed to the model cannot be shorter than the input window used for training. Consequently, if this value is too large, predictions for instruments listed for shorter periods become impossible which limits the model's applicability.

To mitigate these challenges, we conducted optimization to determine an optimal input window size from a set of possible values: [252, 504, 1000, 1500, 2000, 3000]. The choice of 252 days aligns approximately with the number of trading days within a year, ensuring a balance between capturing sufficient historical context and preserving model adaptability to shorter-term instrument listings. We also experimented with shorter window sizes but noticed that the models predict too linearly and we want the models to have enough context to find both a trend and a seasonality in the time series.

The window size parameter determines the model's prediction horizon into the future. In

alignment with our study's objectives, we've configured this parameter to forecast 14 days ahead for all models.

To accommodate our hardware and time constraints while ensuring robust model training, we fixed the maximum number of epochs to train the model before evaluation at 20. Through plotting training and validation loss at each epoch, we could observe that the majority of models converge and reach a plateau in loss within this epoch range given the smaller data sample. This epoch count allows the model to undergo more trials within our resource constraints while ensuring convergence and stability in the learning process. We hope that by performing as many trials as possible we can find a better local minimum or potentially even the global minimum of the loss landscape.

The learning rate determines the optimization process's step size. If the learning rate is too small, convergence takes longer and it risks missing the global minimum, while a learning rate that is too large might also hinder reaching the global minimum. To optimize this parameter, we set the optimizer to explore values between 1e-6 and 1e-2. After the hyperparameter tuning with darts we re-optimized this parameter using the learning rate finder in PyTorch lightning (Smith 2017).

The batch size determines the number of simultaneous time series the model considers during training. Our experimentation with various batch sizes led us to settle on 500 as a compromise between accuracy and computational efficiency. Smaller batches generally have the potential to decrease loss but at the cost of significantly longer processing times.

The number of stacks within the network fundamentally influences the architecture. For our optimization process, we explored this parameter's range between 4 and 20. The interpretable model explained in the NBEATS paper (Oreshkin et al. 2020), which serves as a reference, consists of only 2 stacks, however as we train the generic model we can employ more layers, which can improve performance. We aimed at a somewhat less complex model given our hardware constraints, so choose to only allow the model up to 20 stacks, which is still quite complex.

The number of blocks within each stack is optimized between 2 and 8. The increased amount of blocks brings high complexity and may lead to overfitting, therefore we restricted the maximum to 8. We set the minimum to 2 as it is intuitively the lowest feasible number to accommodate the size of the dataset we are training it with.

Within each block, the number of layers also impacts the model's depth and complexity. We fine-tuned this parameter between 4 and 6, a range selected carefully because it vastly increases the number of trainable parameters and at the same time determines the expansion coefficients for back- and forecast.

The expansion coefficient dimension was optimized between 3 and 8. This hyperparameter needs to be selected with caution since it can affect overfitting immensely when too high, not only for the forecast but also for the backcast, since it relies on denoising the data.

The layer width represents the number of neurons within each fully connected layer. During our optimization process, we optimized this parameter in the range of 128 to 512 neurons.

As there is a large number of layers in the model, even a smaller layer width can cause the model to have many trainable parameters.

The chosen loss function for training is Huber loss with a delta value of 0.3. It exhibits a unique behavior: it increases exponentially for small differences and linearly for larger ones. This characteristic permits the model to explore larger movements and fluctuations in asset prices without excessively penalizing incorrect predictions. Such flexibility is crucial when dealing with financial data, notorious for its unpredictability and the ease of making inaccurate forecasts.

The selected optimizer for our model is Adam, a generally reliable choice that integrates seamlessly with the optuna library used for optimization. Although we considered experimenting with various optimizers like SGD and rAdam, the diverse behavior of different loss functions with the same hyperparameters, especially the learning rate led us to prioritize a cautious approach. Instead of potentially encountering unexpected behavior due to combinations of loss functions and optimizers, we opted for a safer choice by focusing on optimizing the learning rate instead of the optimizer.

We also incorporated dropout layers with a dropout probability optimized within the range of 0% to 50 % for all fully connected layers. This strategy helps prevent overfitting by determining the likelihood that a neuron's output is nullified during training.

For early stopping, we set the patience parameter to 10 epochs, coupled with a minimum delta of 0.00005. This configuration allows the model to halt training earlier for configurations showing little promise, thereby saving computational run-time and allowing us to explore more combinations of hyperparameters.

In our experiments, the inclusion of L2 Regularization within the range of 0 to 0.1 did not yield improvements in the model's performance. Consequently, we opted against incorporating L2 Regularization in any of our models. However, to mitigate overfitting, we still utilized early stopping and the integration of dropout layers

Regarding activation functions, we optimized the choice among 'ReLU', 'SELU', and 'Sigmoid' from the set of available functions. The activation function enables the model to capture and understand nonlinear relationships within the data.

In our experimentation where we optimized the hyperparameters, we conducted two runs: One focused solely on optimizing parameters mentioned above and another one that also included the optimization of the enrichment ratio, ranging from 10% to 50%. The enrichment ratio signifies the extent of additional data inclusion from alternative distributions into the training dataset, augmenting the financial data. For instance, a 5% ratio implies a 5% increase in the training dataset by incorporating data from the weather dataset and another 5% increase from the darts dataset. Notably, due to the abundance of stock data points compared to other distributions, particularly the darts dataset, setting the enrich ratio above 52% for weather data or above 10% for darts data does not yield added benefits as there is no data available to enrich the train set further.

To evaluate model performance consistently during hyperparameter tuning, we utilize an

input window of length 3000. Maintaining a constant window size is important as altering the input window during inference can skew loss comparisons unfairly. Moreover, the window size during inference must not be shorter than the training window size, hence we choose the maximum window size among all possibilities optimized for. Later we use a window size of 504 for all comparisons.

3.4 Metrics

To measure the performance of our models and that of our benchmark models we use several different metrics. Metrics in machine learning are designed to express how well the model performs on real data and returns a single floating point number to represent how good the model is to be considered. Any given predictive AI model is given a set of features (which in our case are the daily prices of a financial instrument over 504 trading days) and returns a prediction (in our case the closing price on the subsequent 14 trading days). How well the model performed can be determined if the actual values (which are called targets) for the prediction period are known, as the model is expected to make a prediction as similar as possible to the targets.

When we aggregate the loss over multiple time series, by default we take the arithmetic mean of the loss among each time series, however for some other metric we use median, in those cases it will be specified explicitly.

- **Mean Absolute Error (MAE)**: This is a well known metric that measures the average magnitude of the difference between each target and prediction data point. The formula is:

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

- **Mean Squared Error (MSE)**: This metric is very similar to MAE. It quantifies the average squared difference between the individual predicted and actual data points. It is calculated using the formula:

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

- **Median of Mean Absolute Error / Median of Mean Squared Error**: After employing the aforementioned metrics we discovered that our own, as well as our benchmark models are prone to occasionally predict very strong price increases or decreases which then do not materialize in the actual price development. These rare instances cause massive outliers in the loss values which the aforementioned metrices are susceptible to, as they compute the simple arithmetic mean. When deploying the models for actual trading usage, a human investor might look at these predictions,

determine that the predictions are unrealistic and decide not to invest in these assets. We attempt to capture the effect of this behaviour by aggregating using the median, as that is less sensitive to outliers. Thus, we created new metric functions based on the MAE and MSE functions to also show us the median loss value along all time series, a behaviour which we have also added to all later-mentioned metrices for better comparability.

- **Huber Loss**: This metric combines traits from both MSE and MAE. It increases exponentially for small differences and linearly for larger ones. This behavior reduces the impact of outliers, making it suitable for the unpredictable and error-prone nature of financial prediction. For that reason we also used this metric as a loss function for all training runs, with a delta value of 0.3, which determines from what threshold to start increasing the value linearly. The formula is:

$$\text{Huber Loss} = \begin{cases} 0.5(x_n - y_n)^2, & \text{if } |x_n - y_n| < \delta \\ \delta \cdot (|x_n - y_n| - 0.5 \cdot \delta), & \text{otherwise} \end{cases}$$

This concludes the pre-existing metrices we used. We also implemented several of our own metrics:

- **Mean of Last Value Error / Median of Mean Last Value Error**: We set up these metrics so they capture the absolute difference between the last value in the predicted and the last value in the target time series. The other values are disregarded and not used for this metric. We designed this metric with an investment strategy in mind where an asset is chosen with regard to the predicted price after 14 trading days and then sold. There, the intermediary price development does not matter, as profit or loss is only realized after 14 trading days. From the absolute differences between the two mentioned values for every time series, we aggregate once using mean and once using median. Mean Loss is more faithful to our envisioned investment strategy, but as mentioned above, to reduce the effect of extreme outliers we provide both averages.
- **Mean Total Return Error / Median of Mean Total Return Error**: When designing this metric we envisioned an investment strategy where an asset is selected based on the predicted performance but can then be sold at any future day based on the existing prediction or on predictions made on later days after knowing the actual price development since opening the position and including it in the features. After selling a position, a new asset can then be bought based on its predicted performance. The metric was setup such that for each predicted data point, the total return since inception is calculated for both the predicted and the target time series. We obtain the return of an instrument as the percentage change of the stock price. We then calculate the difference between each data point in both time series and aggregate using the mean and median function on all differences.

  - Mean Total Return Error: $MTRE = \frac{1}{n}\sum_{i_0}^{n} |\hat{r}_i - r_i|$
  - Median Total Return Error: $MedTRE = \text{med}(|\hat{r}_i - r_i|)$
  - With:
    * $r_i = \frac{y_i}{y_0} - 1$
    * and analogously $\hat{r}_i = \frac{\hat{y}_i}{\hat{y}_0} - 1$

- **Geometric Mean Daily Return Error**: Similarly to the prior one, this metric is meant to measure the performance of the model when following a strategy where assets can be sold at any point in time, however when envisioning this investment strategy, the actor might also decide to invest into an asset at a later date than the one where the model's prediction is made, possibly after the predicted price declines but before it increases. We programmed this function so that for each predicted datapoint the percentage change in price which corresponds to the daily return is calculated and the absolute difference between the prediction and target time series is calculated. Out of these differences we compute the geometric mean to obtain the functions value.

$$GMDRE = \sqrt[n]{\prod_{i=0}^{n} |\hat{r}_i - r_i|}$$

  With:

  - $r_i = \frac{y_i}{y_{i-1}} - 1$
  - and analogously $\hat{r}_i = \frac{\hat{y}_i}{\hat{y}_{i-1}} - 1$

- **Mean Final Return Error / Median of Mean Final Return Error**: This metric is very similar to the Last Value Error. The total percentage change of the asset is calculated, from the inception day which corresponds to the last value of the feature time series, to the last data point of the prediction and target time series. The absolute difference is then calculated and out of the absolute difference in final returns for all individual time series the mean and median value is selected respectively. These metrics would be appropriate for an investment strategy where an asset is bought and sold after the forecast horizon has elapsed.

- **Backtesting Profit Error**: To measure the performance of our models when deployed to follow the investment strategy outlined in the description for metrices "Last Value Error" and "Final Return Error" where an asset is selected based on the predicted performance and then held for the entire length of the forecast window which is 14 trading days for all our models we created this metric. The model is used to predict the stock prices for the next 14 trading days ahead. Out of all time series, which would correspond to recent historic daily prices for all assets in consideration, a subset of 5% is selected which includes those time series with the highest expected percentage increase in price. A hypothetical portfolio that is equally weighted between all the assets in the selected sample is then constructed and it's return is

compared to the return of the portfolio which consists of all assets. We expect this metric to be positive as the model is supposed to choose stocks with likely upward movement. However it is crucial to keep in mind when interpreting the magnitude of this functions result that by the design of our test set. It will receive time series from different points in time. So in addition to choosing time series from assets with upwards potential it will likely also choose time series from points time where markets where upwards trending. When deployed to trade on the stock market in real time, it will not have this ability and thus likely deliver lower performance.

3.5 Benchmark Models

In our study, the performance of our models was assessed by employing the ARIMA and TimeGPT models for comparative analysis. This approach facilitated the evaluation of our models' accuracy in relation to established benchmarks, enabling us to gauge the relative performance of our models pure and enriched states on a standardized scale.

3.5.1 ARIMA

We utilized the Arima(Box and Jenkins 1970) model outlined in section 2.3.1 as a benchmark for comparison with our model. Employing the AutoArima model from the Python darts library, we obtained predictions on our data by randomly sampling from each time series within the test set. Prior to fitting the AutoArima model, we normalized the sample using the procedure described in section 3.2 to obtain comparable results.

The AutoArima model uses the stepwise algorithm to iteratively explore various combinations of the parameters p, q, and d (see section 2.3.1) to minimize loss on the features. Subsequently, the model is refitted using the optimal parameters and predicts the subsequent 14 values, allowing us to compute the loss for evaluation purposes. Using the AutoArima model to optimize the hyperparameters for every sample of the test dataset instead of reusing the same optimized parameters for several samples increases the accuracy of the forecast but is computationally very expensive. For that reason we decided to only take one sample out of each time series in the test set, instead of taking all possible samples.

3.5.2 TimeGPT

To obtain the benchmark performance to measure our models performance against, we contacted Nixtla, the developer of TimeGPT(Garza and Mergenthaler-Canseco 2023), who graciously provided access to their model through an API endpoint along with instructions on how to use it and credentials. This allowed us to input our data and receive predictions for subsequent values. Given limitations on the number of predictions using the API, we

randomly sampled a sub time series from each time series within our test set and assessed the model's performance by calculating the loss on the ensuing 14 values.

Specifically, we utilized the "timegpt-1" model, recommended for predicting a limited number of future data points as per the documentation. Employing 100 fine-tuning steps, we refined the model by iterating through training on our input data, minimizing forecasting errors to enhance predictive accuracy. This allows the model to acclimate to stock data after being initially trained on different datasets by Nixtla.

Regarding data normalization, we adhered to the documentation's guidance, abstaining from normalizing the input data for TimeGPT. However, post-receiving predictions, we normalized the data using the previously described procedure to ensure consistency and facilitate further analysis, as our metrices explained above in section 3.4 rely on normalized input data.

3.6 Model training

We trained a total of five NBEATS models which we compare to two benchmark models (Arima and TimeGPT). Of our own models we trained one with only the financial dataset as the training set. Two where we used the financial dataset as well the weather data only and two where we also included data from the weather and darts dataset.

3.6.1 Training data analysis

We have three datasets, one with financial data, one with weather data and one with various smaller datasets included in the Python darts library. A time series conducted from the mean of each datapoint along all time series can be found in Figure 3.1 below. We can observe that stock are increasing on average over time while the temperature is near constant. The darts datasets is dominated by a dataset of traffic congestion over time, which follows a weekly pattern but is otherwise constant.

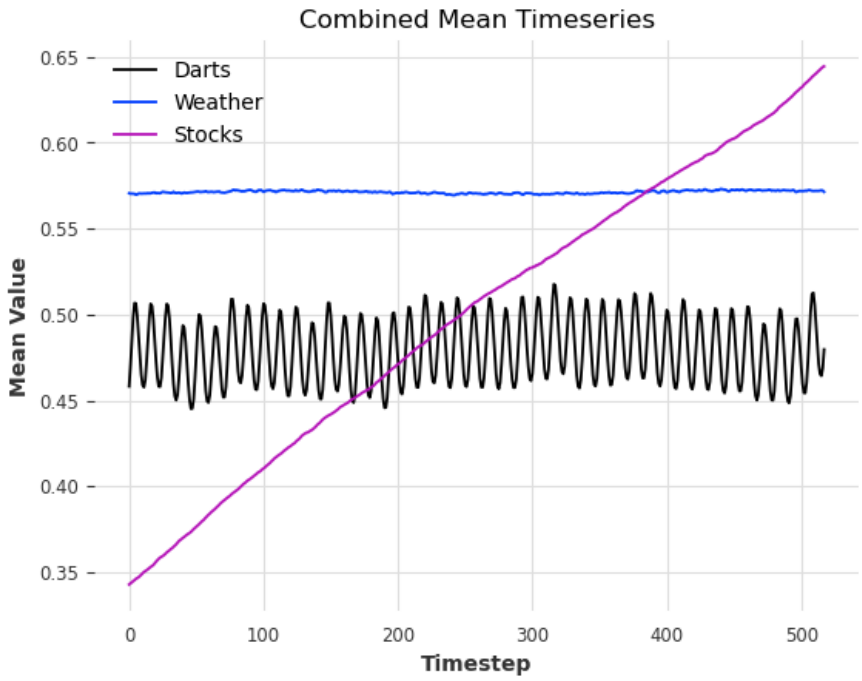

Figure 3.1: The average of all Time Series clusters combined

The correlation coefficient between the mean financial time series and the weather time series is 0.22149 while the coefficient between the financial time series and the darts dataset is 0.05886. Both correlations are quite weak, while the correlation with the darts dataset is especially weak. For that reason we decided to train models with both: Only weather data as well as weather and darts data. It is, however, noteworthy that the mean time series do not accurately represent the trends and seasonality of individual time series within the respective distribution.

For that reason we have conducted another experiment: we took a random sample of 5,000 time series, each of length 504 from each of the datasets. We calculated the correlation coefficients of all samples from the stocks dataset with all samples from the weather and darts dataset. From the resulting lists with the correlation coefficients of length 25,000,000 each we took the absolute values and calculated the arithmetic mean of the resulting list. The resulting values are a better approximation of the similarity of the datasets and bounded between zero and one just like the simple correlation coefficient.

- Average correlation coefficient between stock and weather data: 0.23654
- Average correlation coefficient between stock and darts data: 0.17710

We can thus conclude that the weather dataset is more similar to the stock data, than the darts data is, but that both dataset are considerably different to the stock dataset.

3.6.2 Only stock data

For our model with only stock data, we used all time series we could extract from the financial dataset where every time series has the length of 518 trading days which corresponds to 504 datapoints that are features and 14 that are targets.

We utilized Optuna library to optimize hyperparameters within the defined parameter ranges (refer to Section 3.2 for details).

The hyperparameter tuning involved 300 iterations using an iterative approach that explores various combinations of hyperparameters. The optimization followed a tree-structured Parzen Estimator algorithm. Throughout these iterations, we monitored the loss on the test set using different loss functions for each iteration. Ultimately, the hyperparameter combination resulting in the lowest Mean Absolute Error (MAE) Loss was selected. Finally we re-optimized the learning rate using the default PyTorch learning rate finder Figure A.1, as due to the lower amount of epochs and available training data during prior hyperparameter tuning, the Optuna library seemed to favour higher learning rates. We assume that this behaviour is related to the model attempting to make as much progress as possible before being stopped, so the next iteration of the tuner can begin. The corresponding parameters are:

- windows size: 504
- learning rate: 1.2022644346174132e-06
- number of stacks: 5
- number of blocks: 6
- number of layers: 5
- expansion coefficient dimension: 6
- layer widths: 129
- dropout: 0.138856
- activation function: Sigmoid

The model exhibited robustness across various hyperparameters. We performed a multivariate linear regression using mean absolute error as the dependent variable. The results are summarized in the following table:

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0298	0.004	8.426	0.000	0.023	0.037
window_size	3.244e-06	5.33e-07	6.092	0.000	2.2e-06	4.29e-06
learning_rate	-0.0252	0.223	-0.113	0.910	-0.464	0.413
num_stacks	0.0003	7.53e-05	3.600	0.000	0.000	0.000
num_blocks	0.0001	0.000	0.454	0.650	-0.000	0.001
num_layers	-0.0001	0.001	-0.215	0.830	-0.001	0.001
expansion_coefficient_dimension	5.975e-05	0.000	0.228	0.819	-0.000	0.001
layer_widths	4.672e-06	3.44e-06	1.358	0.176	-2.1e-06	1.14e-05

Table 3.1: Regression of hyperparameters on MAE Loss with only stock data

For some of the parameters, the p-value is quite large. However, the confidence interval does not deviate far from zero for all variables. This indicates that the model is robust against different values for the hyperparameters. The only coefficient which is distant from zero is that of the learning rate, which is however easily explainable as the changes in the learning rate are only minor and the learning rate is bounded in our experiment between 1e-6 and 1e-2. Thus any actual changes in loss, resulting from changes in the learning rate are much smaller. Further, since we re-optimized the learning rate later, interpreting the coefficient for the learning rate during Optuna optimization is unproductive.

Following the hyperparameter optimization, we trained the model using the identified optimal hyperparameters. The training employed the Adam optimizer with Huber loss, using $\delta = 0.3$. Multiple training runs were conducted, and the best-performing model on the test set was selected. Performing several complete training rounds was necessary because the models would sometimes explore exclusively into suboptimal directions of the loss landscape and thus the loss would plateau to different levels. The optimal model was trained for 78 epochs before being preemptively stopped due to signs of overfitting. The best weights were then restored, and the model's performance was evaluated on the test set. The training and validation loss during training from the best performing model can be found in figure 3.2 bellow. Please note that the y-axis is scaled logarithmically.

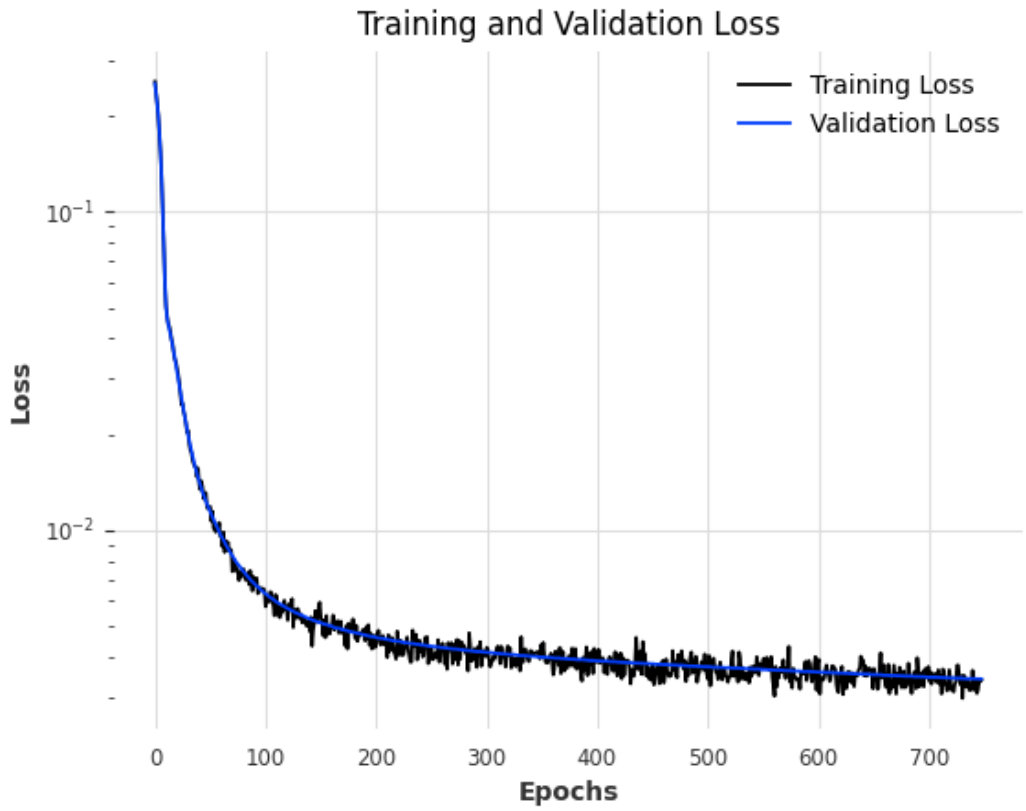

Figure 3.2: Loss convergence using only stock data (the y axis is scaled logarithmic)

The model's convergence pattern is visually apparent in the graph, indicating a rapid convergence initially, followed by a slow but steady convergence phase before being interrupted by the early stopping mechanism outlined before.

The test loss was then calculated on all possible sub time series that could be constructed from the 15% test sample of the entire stock data. Losses from all models are presented in Chapter 4.1.

3.6.3 Enriched data

Following a similar methodology, we conducted an evaluation of the model's performance using enriched training data. The procedure again involved optimizing the hyperparameters through optuna and then selecting the hyperparameter combination that exhibites the lowest Mean Absolute Error (MAE) loss for subsequent training. Combined with re-optimizing the learning rate Figure A.2. These hyperparameters are:

- windows size: 252

- learning rate: 0.0007585775750291836
- number of stacks: 4
- number of blocks: 4
- number of layers: 5
- expansion coefficient dimension: 4
- layer widths: 136
- dropout: 0.156693
- activation function: Sigmoid
- enrichment ratio: 10%

Please note that the enrichment ratio is 10%, this is the enrichment ratio for both weather and darts data. We choose to apply the same ratio to both data sets, to speed up the hyperparameter tuning process and because we expect that the ideal enrichment ratios for data from both data distributions are similar. The training dataset this model is fitted on is thus 20% bigger than that, the model for only stock data was fitted on.

We trained the model several times with these hyperparameters but found the model's loss to plateau to nearly identical every time.

Similar to the model trained exclusively on stock market data, Table 3.2 below demonstrates the model's robustness against hyperparameter variations. Notably, this table represents results derived from a training procedure utilizing a different random sample comprising only 5% of the overall data. Consequently, the observed higher constant can be attributed to the random selection of data and does not by itself hont towards any relative performance of the models. The final models are trained using the entirety of financial data alongside enriched data, where applicable. Moreover, the evaluation of test loss is conducted on a comprehensive 100% sample of all data present in the test set. This methodology ensures that any difference between the models losses can be attributed to the characteristics of the model and the data type and not random sampling bias.

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0426	0.005	7.792	0.000	0.032	0.053
enrich_ratio	0.0031	0.001	2.849	0.005	0.001	0.005
window_size	3.277e-06	4.16e-07	7.876	0.000	2.46e-06	4.1e-06
learning_rate	0.2130	0.172	1.237	0.217	-0.126	0.552
num_stacks	0.0001	4.3e-05	2.351	0.019	1.65e-05	0.000
num_blocks	0.0003	0.000	1.113	0.266	-0.000	0.001
num_layers	-0.0027	0.001	-2.606	0.010	-0.005	-0.001
expansion_coefficient_dimension	2.491e-05	0.000	0.198	0.843	-0.000	0.000
layer_widths	1.428e-06	3.7e-06	0.386	0.700	-5.85e-06	8.7e-06

Table 3.2: Regression of hyperparameters on MAE Loss with all data

The confidence interval for all parameters is closely bound around zero, which indicates that this model also is quite robust against changes in the hyperparameters. The coefficient for the enrichment ratio is positive, which indicates that increasing the enrichment ratio is associated with an increase in loss. However, this only considers a linear relationship and assumes the other parameters to be constant, while in reality we adjust them to fit the enrichment ratio. Further, this data has been obtained from training for a limited number of epochs on only a small sample of the full dataset. Thus from this figure alone, it cannot be determined whether adding data from other distributions is helpful or harmful to the models performance, that question will be answered by the relative performance of models which were trained for longer and on all available data.

Figure 3.3, below shows the convergence of the model. It was trained on the enriched train dataset for 141 epochs before being stopped by the early stopper and converged successfully before reaching it's capacity. Please also note that the absolute training and validation loss is not directly comparable as they have been obtained using different window sizes, where larger window sizes decrease loss further.

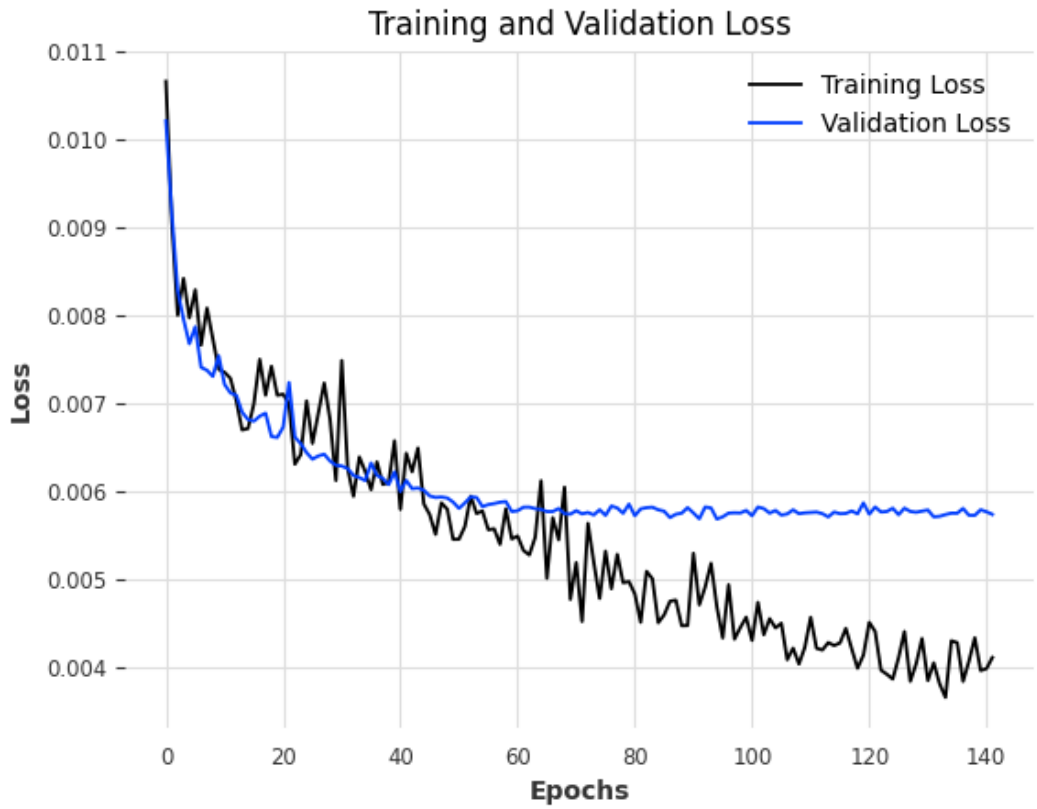

Figure 3.3: Loss convergence using data from all sources with enrichment ratio = 10%

We also trained a model with the same hyperparameters but without including the darts dataset. That model trained for 167 epochs, the convergence plot is shown in the appendix A.4.

3.6.4 Further Enrichment

We retrained both models several times and at each iteration the enriched model converged at approximately the same loss. The model trained on only stock market data however converges to drastically different losses each time.

We were surprised by the strong increase in stability. This means, that the model must not be retrained repeatedly because its validation loss remains stable. Therefore, we decided to experiment further with this behaviour. We again performed an experiment where we optimized the hyperparameters, while only allowing the hyperparameter tuner to select an enrichment ratio between 30% and 50%.

The optimal hyperparameters we discovered in this experiment (followed by re-optimizing the learning rate as seen in Figure A.3) are as follows:

- windows size: 252
- learning rate: 0.0010964781961431851
- number of stacks: 9
- number of blocks: 4
- number of layers: 6
- expansion coefficient dimension: 6
- layer widths: 379
- dropout: 0.340826
- activation function: Sigmoid
- enrichment ratio: 40%

Because of the high number of stacks and increased layer widths, this model has significantly more trainable parameters than the two prior models. As seen table 3.3 bellow, this model too is quite robust against changes in the hyperparameters, so would likely also perform well with a smaller architecture.

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0447	0.013	3.466	0.001	0.019	0.070
enrich_ratio	0.0096	0.014	0.684	0.495	-0.018	0.037
window_size	5.344e-06	1.04e-06	5.146	0.000	3.29e-06	7.4e-06
learning_rate	2.4400	0.413	5.908	0.000	1.622	3.258
num_stacks	4.901e-05	0.000	0.190	0.849	-0.000	0.001
num_blocks	0.0008	0.001	1.173	0.243	-0.001	0.002
num_layers	-0.0023	0.002	-1.426	0.157	-0.006	0.001
expansion_coefficient_dimension	-0.0014	0.001	-2.252	0.026	-0.003	-0.000
layer_widths	-3.737e-07	1.11e-05	-0.034	0.973	-2.23e-05	2.16e-05

Table 3.3: Regression of hyperparameters on MAE Loss with all data for enrichment ratio = 40%

From the plot for the model corresponding to the hyperparameters shown below with an enrichment ratio of 40% using stock, weather and darts data we can observe that most of the models progress was made within the first epochs. Further the training loss spiked shortly around epoch 88, before quickly recovering.

As in the two sections before, in figure 3.4 we can observe the training and validation loss of the model. We can observe a steady improvement in the validation loss before plateauing so the training procedure appears to have been successfully.

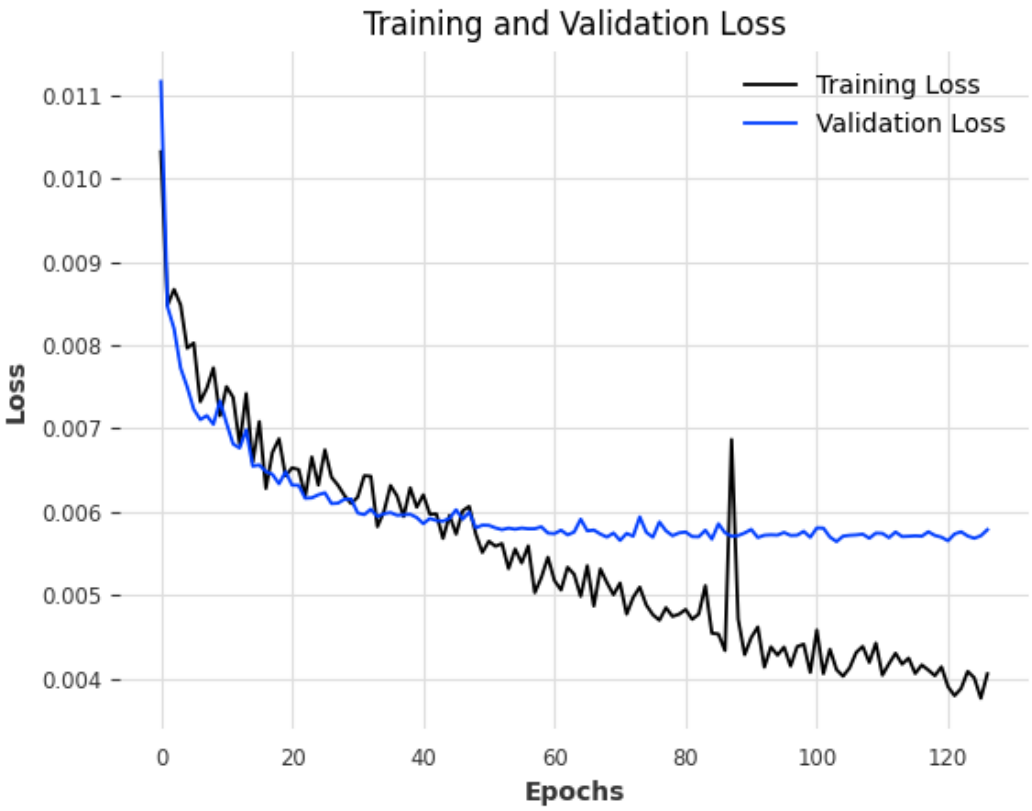

Figure 3.4: Loss convergence using data from all sources with enrichment ratio = 40%

As with the model with a corresponding enrichment ratio of 10%, we also trained a model with identical hyperparameters to this one, without including the darts data. The convergence plot can be found in appendix A.5.

Chapter 4

Results & Interpretation

4.1 Model performance

As already mentioned in 3.6, we trained five different models. These five models are compared to two baseline models, TimeGPT and ARIMA. Notably, TimeGPT is a foundation model that is pre-trained on a variety of different data. Unfortunately, we could not tune the model's parameters like fine-tuning steps to an optimum as we were constrained in the amount of API calls we could make to the TimeGPT endpoint per minute. This limits the interpretation of time series results as with more hyperparameter tuning we might have been able to improve the results further. However, the results can still be meaningfully interpreted. Furthermore, to evaluate the results, we evaluated each model on several metrics. The metrics are discussed in 3.4. For a table which shows all results, please refer to table A.3.

4.1.1 Mean of Mean Absolute Error & Median of Mean Absolute Error

As with most metrices in machine learning, the lower the value of Mean and Median of Mean Absolute Error, the better the model performed. The performance of the different models measured using the Mean Absolute Error, which is the function we optimized the model on for the hyperparameter tuning, can be seen in figure 4.1 below on the left-hand side.

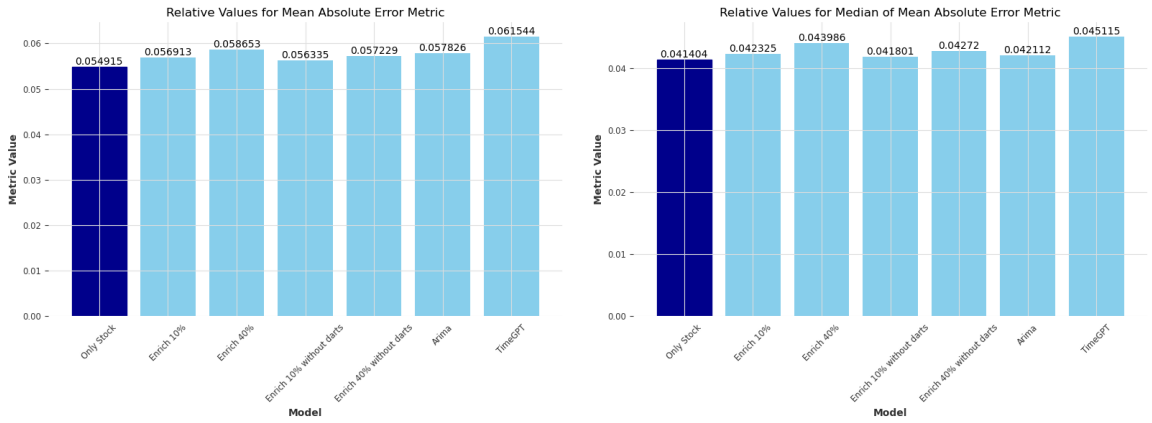

Figure 4.1: Charts for Mean and Median of Absolute Errors

We can observe that the worst performing model is the TimeGPT benchmark model with a corresponding loss of approximately 0.0615. On the other hand, we see the stock model to perform best with a Mean Absolute Error of 0.0549, the best performing model is also highlighted in dark blue for easier identification. When compared to the enriched models i. e., the models that were trained on additional data, we can see that the additional data, harmed the performance measured by this metric. Interestingly, the models that were only enriched with weather data and not with the remaining darts data, performed better than the ones that were also enriched with data from the darts datasets. As we will propose later in section 4.2, this may be related to the weather data being more similar to the stock data than the darts dataset is. We established that relationship in section 3.6.1.

While the Mean Absolute Error is an interesting metric to measure the deviation from the ground truth, to account for outliers and estimations in the wrong direction, we also considered the Median of Mean Absolute Error as seen in figure 4.1 above on the right hand-side. Although the error seems to have decreased for all models compared to Mean Absolute Error, it confirms the results of Mean Absolute Error. Significantly however, we can see the most drastic difference for the ARIMA model, that according to the Median of Mean Absolute Error performs slightly better against the others compared to the Mean Absolute Error. Still, TimeGPT remains the worst, while Stocks performs best.

4.1.2 Mean Squared Error and Median of Mean Squared Error

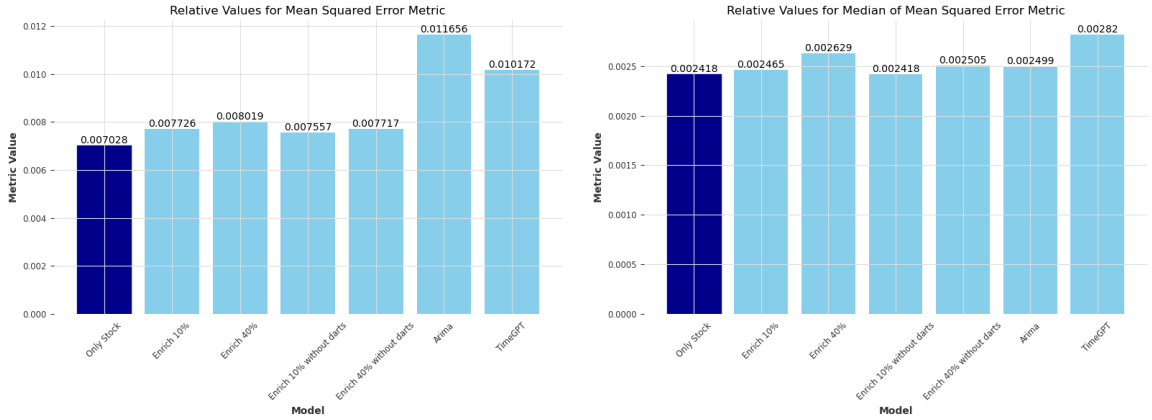

Figure 4.2: Charts for Mean and Median of Mean Squared Errors

The Mean and Median of Mean Squared Errors from the predicted to the actual values show a similar pattern. ARIMA seemingly is most influenced by the Mean Squared Error therefore indicating it has more or stronger outliers. Still, these Error metrics confirm the results we have seen before: Stock maintains its lead in performance and enrichment does not improve performance.

4.1.3 Huber Loss

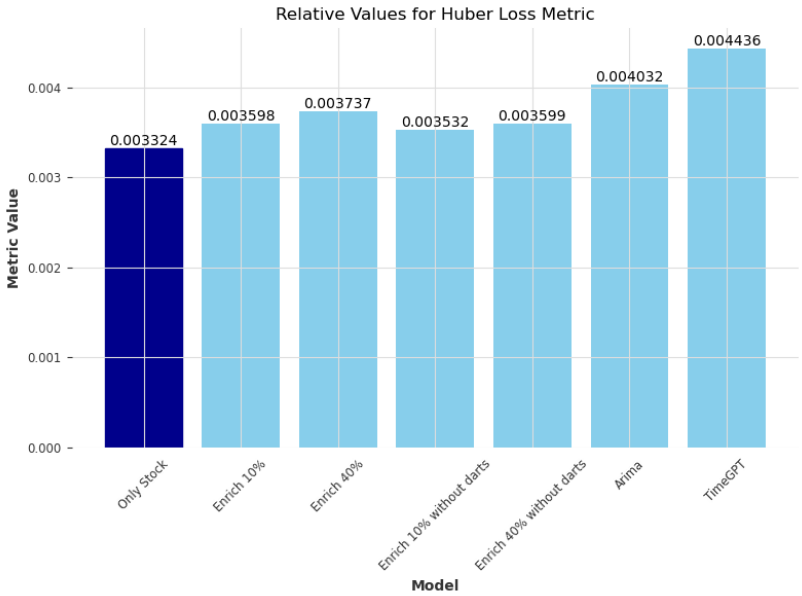

Figure 4.3: Chart for Huber Loss

The Huber Loss metric combines aspects of Mean Absolute Error and Mean Squared Errors making it less sensitive to outliers. We used the Huber Loss as the loss function during training of all models. The Stock model performs best and TimeGPT worst, giving a consistency across all the different metrics and underscoring the robustness of the stock model compared to its counterparts.

4.1.4 Mean Last Value Error and Median Last Value Error

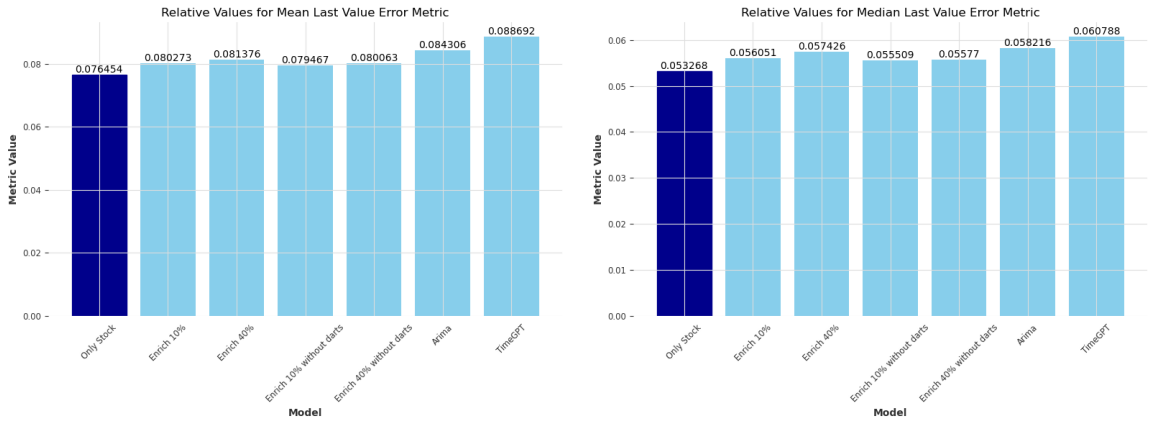

Figure 4.4: Charts for Mean and Median of Last Value Errors

The Mean and Median of Last Value Errors are used to indicate the models usefulness in predicting future prices where the values in between are of no importance. A practical application could be closed end fonds which cannot be sold before the maturity or termination date. We can observe that as before, the model trained without enrichment on stock data only, performs best. The model that is enriched with 10% weather data and no darts data is the next best model and the benchmark models perform worst.

4.1.5 Mean Total Return Error, Median of Mean Total Return Error and Mean Final Return Error and Median Final Return Error

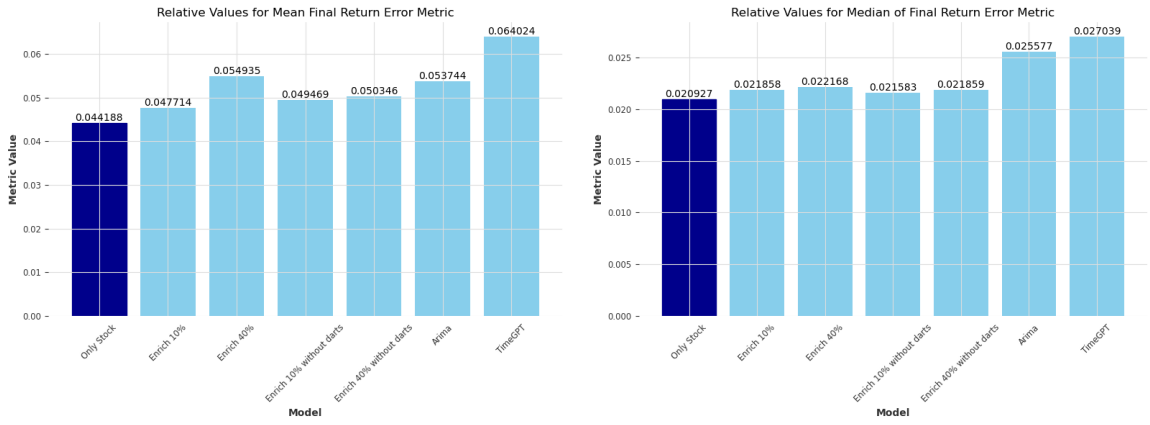

Figure 4.5: Charts for Mean and Median of Final Return Errors

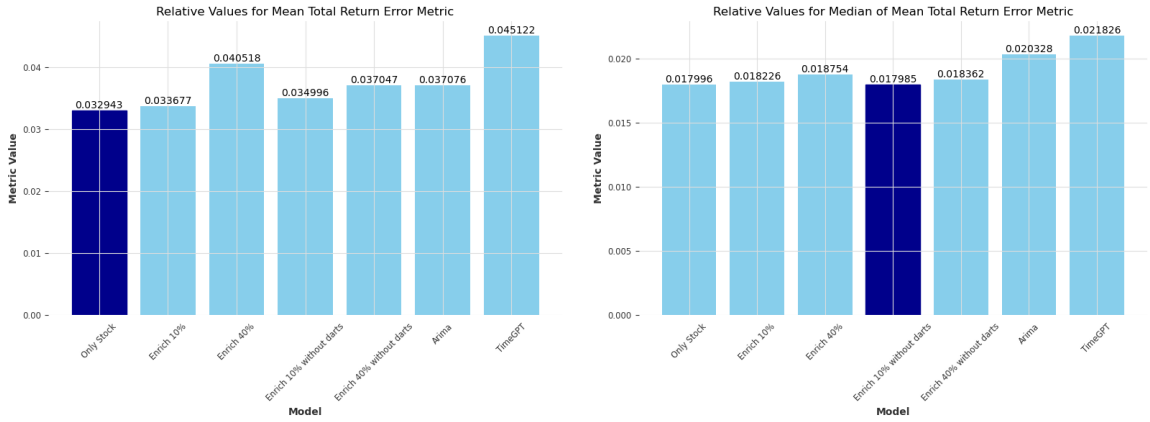

Figure 4.6: Charts for Mean and Median of Mean Total Return Errors

For both Mean and Median Final Return Error, TimeGPT again shows the worst performance, while stock is still superior over its enriched competitors. This is equally true for both Mean Final Return Error. Further, even though Mean/Median Total Return Error is conceptually much more close to Mean/Median Last Value Error, the plots are remarkably close to those of Mean/Median of Mean Total Return Error. We can however observe that for Median of Mean Total Return Error, the model trained with 10% enrichment with only weather data in the enrichment set has the lowest loss, beating the model trained on only stock market data by a tiny margin. This may be due to a better generalization over longer

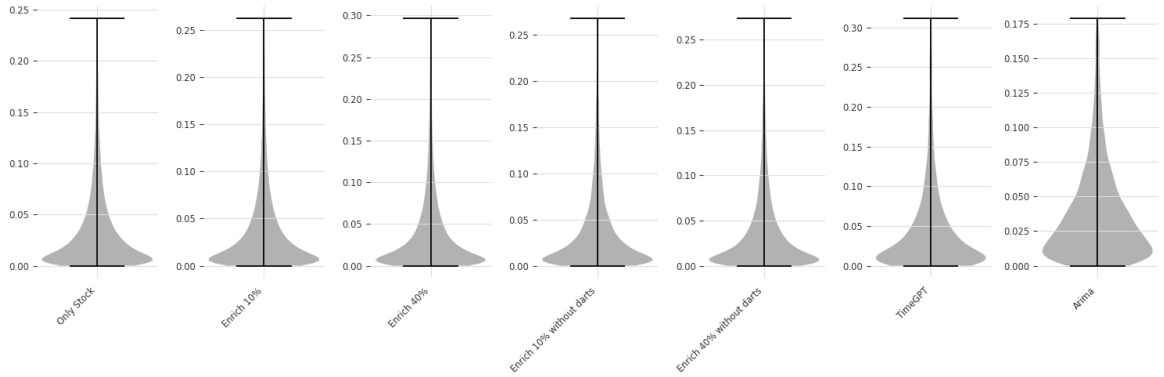

Figure 4.7: Loss distribution of Mean/Median of Final Return Error for different models.

time periods or due to random chance during selection of the enrichment data, which might correlate especially well with the test set.

The Violin Plot above shows a distribution curve of the Mean or Median of Final Return Errors for all time series. For each sub plot, the y-axis shows the loss for each individual predictions and how wide the curve is corresponds to how many times that loss occurred in the list of the losses for all individual time series. Please note that this plot was created using the Mean of Final Return Errors metric function in our related code, however as we calculated an individually loss value for every single prediction and because taking the Mean or Median of a list with only one value is equivalent, the plot would look identical if created using Median of Final Return Errors. Because there are some large outliers in the loss, we dropped the highest 1% of losses, which makes interpretation of the curves much easier.

We can observe from the plot that the Arima model tends to make predictions that results in higher losses more often and such with low losses less often. Further we can observe that the distribution of loss looks very similar for all other models, only the scale of the y-axis being apparently different.

4.1.6 Geometric Mean Daily Return Error

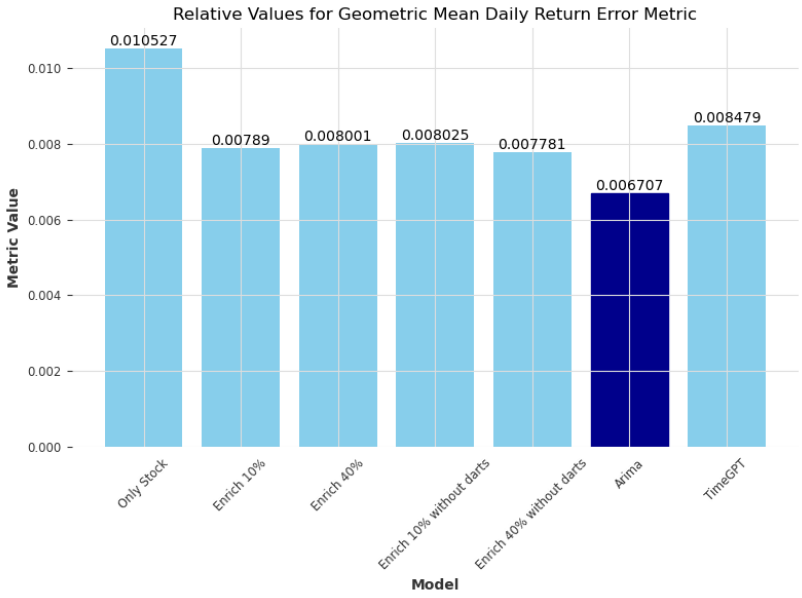

Figure 4.8: Chart for Geometric Mean Daily Return Error

Notably, the Geometric Mean Daily Return Error, that is crucial to evaluate daily performance of the models, was the best with ARIMA, showcasing its potential for short-term predictions. Contrary to the results from all prior metrics, the model trained on financial data is the worst performing model by quite a margin, while all enriched models beats it's performance. Thus, to enable a trading strategy where financial instruments may be bought or sold at any day, as described in section 3.4, it may be better to use an enriched model than a model trained on purely financial data.

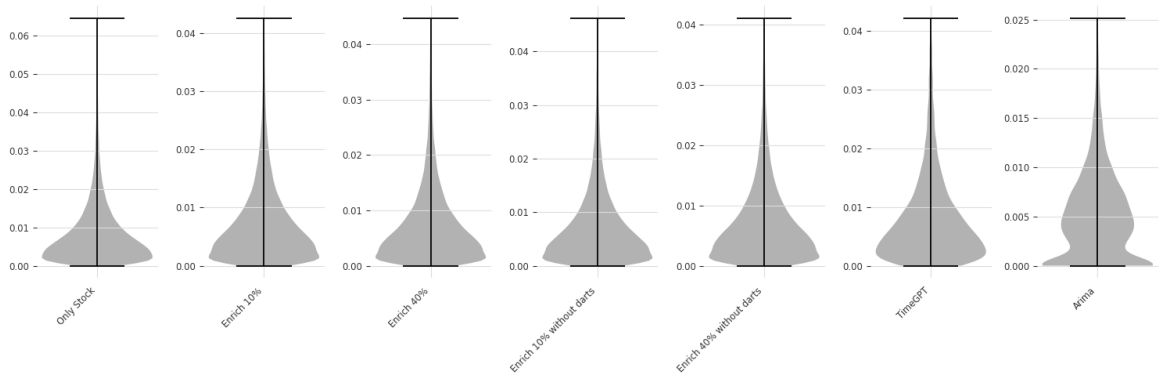

Figure 4.9: Loss distribution of Geometric Mean Daily Return Error for different models.

4.1.7 Backtesting Profit Error

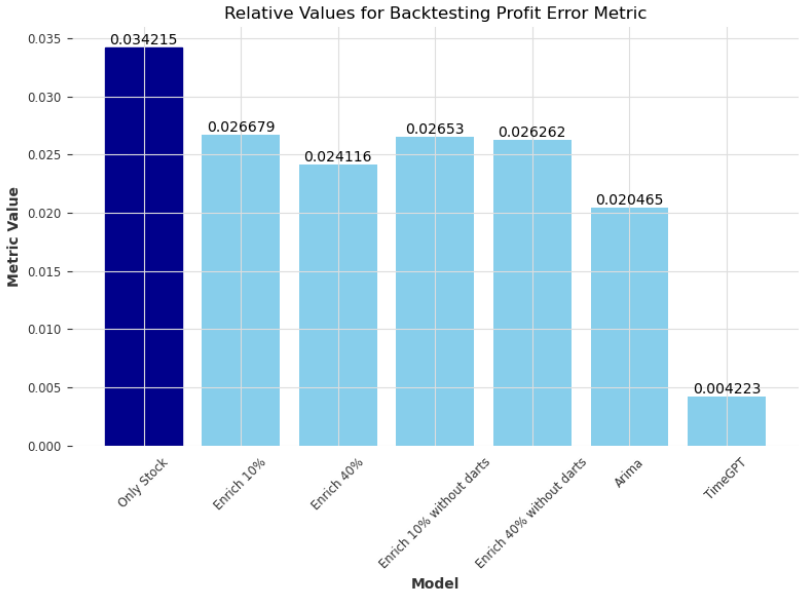

Figure 4.10: Chart for Backtesting Profit Error

The backtesting profit error is a metric we designed to evaluate the practical profitability of the models. Reversed to the other metrics, it is better to have a high value compared to a low value. Therefore, again, TimeGPT was performing the worst and stock the best. This hints at the fact that in a real-world trading scenario where we invest for a fixed number of days, we should employ the stock model to potentially obtain the best investment return.

4.1.8 Interpretation

In conclusion, this analysis reveals a clear pattern: The stock model consistently outperformed all its competing models across almost all metrics, which proves that the endeavor to improve performance by enrichment with other types of data has been unsuccessful. In contrast to the stocks model, TimeGPT, which is a foundation model that, except for the fine-tuning steps, completely relies on Transfer Learning, was performing worse than all other models across most metrics, despite having been trained on most data in total. Also, we can observe that the performance of the models generally becomes worse the more they are enriched. This further supports the hypothesis that enrichment negatively affects model performance and contradicts any randomness in sample selection involved as the reason for the worse performance. Further the regression analysis performed on the validation loss during hyperparameter tuning revealed positive coefficients for the enrichment ratio.

Interestingly, ARIMA was performing in some metrics well and in some badly. Intuitively, this is because ARIMA has more outliers making it prone to have higher values in Mean Errors compared to Median Errors. However, as already mentioned, the Geometric Mean Daily Return Error is lowest for ARIMA, making it reliable for short term predictions. Still, the most robust and reliable model for good model performance is the stock model as it performs well among all metrics. Especially considering Financial decision-making applications, we see the stock model to be outperforming in all important metrics, such as Mean/Median Total/Final Return Errors and the Backtesting Profit Error. A good performance in metrics like the Huber Loss, indicates a good resilience to outliers, because this metric is less sensitive to extreme values. Extreme outliers are likely to be found in real-world scenarios, especially in Economics and Finance.

4.2 Transfer Learning

The core reason why nowadays foundation models are so powerful is because they can make use of vast datasets enabling them to transfer learn (Bommasani et al. 2022). The ability to employ transfer learning can be decisive in dramatically improving the model's performance because with more data available, upscaling the model is facilitated. As previously shown in our results 4.1, enriched models i. e., models that were trained on differing source domains do not outperform the model that was only trained in the target domain which is stock prediction in this case. In almost all metrics, the purely on stock data trained model is better than the ones trained on multiple datasets including TimeGPT. Hence, the question arises why that is the case: As pointed out in the literature review 2.2 and in Zhuang et al. 2020, there are hurdles to overcome to make Transfer Learning successful. These are the core challenges:

- **Negative Transfer**: The relationship between source and target task is not compatible.

- **Sample Selection bias**: Inappropriate selection of control groups or samples can introduce bias, affecting the target task's performance.
- **Mismatch of probability distributions**: If the probability distributions of the source domain $P(DS)$ and target domain $P(DT)$ are not equal, it requires modifications to the optimization problem such that the generalization capabilties are improved, which may not always be successful or feasible.
- **Overfitting**: The model could become subject to overfitting because it now sees a source domain that is not the target domain. We have to ensure a balance between source and target domain. Therefore, techniques to prevent overfitting have to be applied.

In the case of this thesis, we assume **negative transfer** to be the issue why the model does not become more accurate upon giving it more training data. As we can also observe the models training loss to improve while the validation loss remains constant during training. As the validation set consists of data from only the target domain and the train set is enriched. The model might make progress towards better predicting weather data which does not benefit the performance in the target domain. We assume that this behaviour is not caused by overfitting or that overfitting is not a large factor as the validation is constant and would usually trend upwards if the model is overfitting.

4.2.1 Negative Transfer: Data Similarity

As shown in 3.1, the average of all Time Series do not correlate quite well. While the average stock time series is a strictly monotonically increasing quasi-linear function, the average weather time series is a quasi constant and the darts dataset, dominated by traffic data, is on average stagnating. This setup seems to hurt the model's performance in specializing on stock prediction. It does not bring the desired effect of performing better as one might assume because of expected better generalization and having seen more data. The fact that a model that has only been enriched with weather data and not with the darts dataset performs better than one that has been trained on both weather and darts data hints that the outcome could potentially improve when the source data is more similar to the target data because weather data is more similar. The difficulty specific to the problem of predicting stocks using Transfer Learning is therefore to gather stock-like data in reasonable quantities. As a consequence, we can see that similarity in data matters to the performance of the model. Luckily, we can calculate the similarity and draw conclusions whether or not to include the specific data in a specific context.

Chapter 5

Conclusion

5.1 Summary of Key Findings

This thesis had the purpose to investigate the feasibility of differently trained Deep Learning models, some were making use of transfer learning to different degrees while one remained trained solely on the target task. Specifically, the target was to accurately predict stock market trends. This analysis revealed several interesting insights. Firstly, the model trained exclusively on stock market data consistently outperformed the enriched models across almost all metrics that we tested them on. This finding is contrary to the initial inspiration to write the thesis that enrichment with different datasets could enhance the model's performance. Therefore, we have shown that enrichment with non-financial data, in our case weather and "darts" datasets, were harmful. We assume this to be because of negative transfer. Negative transfer is a phenomenon that can occur when additional data that is dissimilar impedes the learning process of the transfer learning model. Furthermore, the benchmark model "TimeGPT" also gives us this intuition since despite being trained on diverse data, it was performing worst in the specific task of predicting stock data. This emphasizes that the highly specific target domain of stock market data makes it hard to compete for generalized models against specialized models. Models particularly trained on stock market data are required for this task.

5.2 Implications & Future Research

The result of this study underscores the importance when selecting datasets to develop models for predicting stock markets. Tailored financial data can, as a consequence, deliver more reliable insights for the decision-making process. As we only enriched the data with weather and diverse datasets that were natively offered by Darts, one could in the future investigate on whether more similar data can have a positive transfer effect, as we already pointed out that data similarity can be a problem 4.2. In practice however, the nature of

the problem of forecasting stock data is not feasible for Transfer Learning due to the easy accessibility of stock data itself and the lack of similar data that is publicly available and easily accessible as a source domain. Transfer Learning could be helpful when the target domain lacks data and the source domain can complement this lack. Additionally, exploring more advanced neural network architectures and their adaptability in the field of financial forecasting could serve as a foundation for further future studies.

5.3 Final thoughts

In conclusion, this thesis contributes to the understanding of Transfer Learning in financial time series forecasting. We could show that **incorporating data from various resources is generally not beneficial** for time series predictions on stock prices using neural forecasting methods.

Bibliography

Yule, G. Udny (1927). "On a Method of Investigating Periodicities in Disturbed Series, with Special Reference to Wolfer's Sunspot Numbers". In: *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character* 226, pp. 267–298. URL: http://www.jstor.org/stable/91170 (visited on 01/03/2024).

Box, George E. P. and Gwilym M. Jenkins (1970). *Time Series Analysis: Forecasting and Control*. San Francisco: Holden-Day.

Ho, S.L. and M. Xie (1998). "The use of ARIMA models for reliability forecasting and analysis". In: *Computers & Industrial Engineering* 35.1, pp. 213–216. URL: https://www.sciencedirect.com/science/article/pii/S0360835298000667.

Assimakopoulos, V. and K. Nikolopoulos (2000). "The theta model: a decomposition approach to forecasting". In: *International Journal of Forecasting* 16.4. The M3- Competition, pp. 521–530. URL: https://www.sciencedirect.com/science/article/pii/S0169207000000662.

Tsay, Ruey S. (2000). "Time Series and Forecasting: Brief History and Future Research". In: *Journal of the American Statistical Association* 95.450, pp. 638–643. URL: http://www.jstor.org/stable/2669408 (visited on 01/03/2024).

Zadrozny, Bianca (Sept. 2004). "Learning and Evaluating Classifiers under Sample Selection Bias". In: *Proceedings, Twenty-First International Conference on Machine Learning, ICML 2004* 2004.

Trindade, Artur (2015). *ElectricityLoadDiagrams20112014*. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C58C86.

Castelvecchi, Davide (2016). "Can we open the black box of AI?" In: *Nature News* 538.7623, p. 20.

Smith, Leslie N. (2017). *Cyclical Learning Rates for Training Neural Networks*. arXiv: 1506.01186 [cs.CV].

Lai, Guokun et al. (2018). *Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks*. arXiv: 1703.07015 [cs.LG].

Makridakis, S, E Spiliotis, and V Assimakopoulos (2018). "Statistical and Machine Learning forecasting methods: Concerns and ways forward". In: *PLoS ONE* 13.3, e0194889.

Akiba, Takuya et al. (2019). *Optuna: A Next-generation Hyperparameter Optimization Framework*. arXiv: 1907.10902 [cs.LG].

Jhana, Nicholas (2019). *Hourly energy demand generation and weather Dataset*. Accessed on: 2023-11-23. URL: https://www.kaggle.com/datasets/nicholasjhana/energy-consumption-generation-prices-and-weather.

Oreshkin, Boris N. et al. (2020). *N-BEATS: Neural basis expansion analysis for interpretable time series forecasting*. arXiv: 1905.10437 [cs.LG].

Smyl, Slawek (2020). "A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting". In: *International Journal of Forecasting* 36.1. M4 Competition, pp. 75–85. URL: https://www.sciencedirect.com/science/article/pii/S0169207019301153.

Zhuang, Fuzhen et al. (2020). *A Comprehensive Survey on Transfer Learning*. arXiv: 1911.02685 [cs.LG].

Zhou, Haoyi et al. (2021). *Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting*. arXiv: 2012.07436 [cs.LG].

Bommasani, Rishi et al. (2022). *On the Opportunities and Risks of Foundation Models*. arXiv: 2108.07258 [cs.LG].

Hosna, Asmaul et al. (Oct. 2022). "Transfer learning: a friendly introduction". In: *Journal of Big Data* 9.

Zeng, Ailing et al. (2022). *Are Transformers Effective for Time Series Forecasting?* arXiv: 2205.13504 [cs.AI].

EIA, U.S. (2023). *Petroleum & Other Liquids*. Accessed on: 2023-11-23. URL: https://www.eia.gov/dnav/pet/hist/LeafHandler.ashx?n=PET&s=wgfupus2&f=W.

Garza, Azul and Max Mergenthaler-Canseco (2023). *TimeGPT-1*. arXiv: 2310.03589 [cs.LG].

Kunz, Manuel et al. (2023). *Deep Learning based Forecasting: a case study from the online fashion industry*. arXiv: 2305.14406 [cs.LG].

NOAA, United States (2023). *Global Historical Climatology Network daily*. Accessed on: 2023-11-23. URL: https://www.noaa.gov/data.

Vaswani, Ashish et al. (2023). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL].

Yahoo Finance (Nov. 2023). *Historic Financial EOD Prices*. Accessed on: 2023-11-21. URL: https://finance.yahoo.com/quote/.

Appendix A

Appendix

A.1 Literature Review

A.1.1 Loss Metrics NBEATS

$$\text{sMAPE} = \frac{200}{H}\sum_{i=1}^{H}\frac{|y_{T+i} - \hat{y}_{T+i}|}{|y_{T+i}| + |\hat{y}_{T+i}|} \qquad \text{MAPE} = \frac{100}{H}\sum_{i=1}^{H}\frac{|y_{T+i} - \hat{y}_{T+i}|}{|y_{T+i}|}$$

$$\text{MASE} = \frac{1}{H}\sum_{i=1}^{H}\frac{|y_{T+i} - \hat{y}_{T+i}|}{\frac{1}{T+H-m}\sum_{j=m+1}^{T+H}|y_j - y_{j-m}|} \qquad \text{OWA} = \frac{1}{2}\left[\frac{\text{sMAPE}}{\text{sMAPE}_{Naive2}} + \frac{\text{MASE}}{\text{MASE}_{Naive2}}\right]$$

A.2 Methodology

A.2.1 Learning Rate Finder

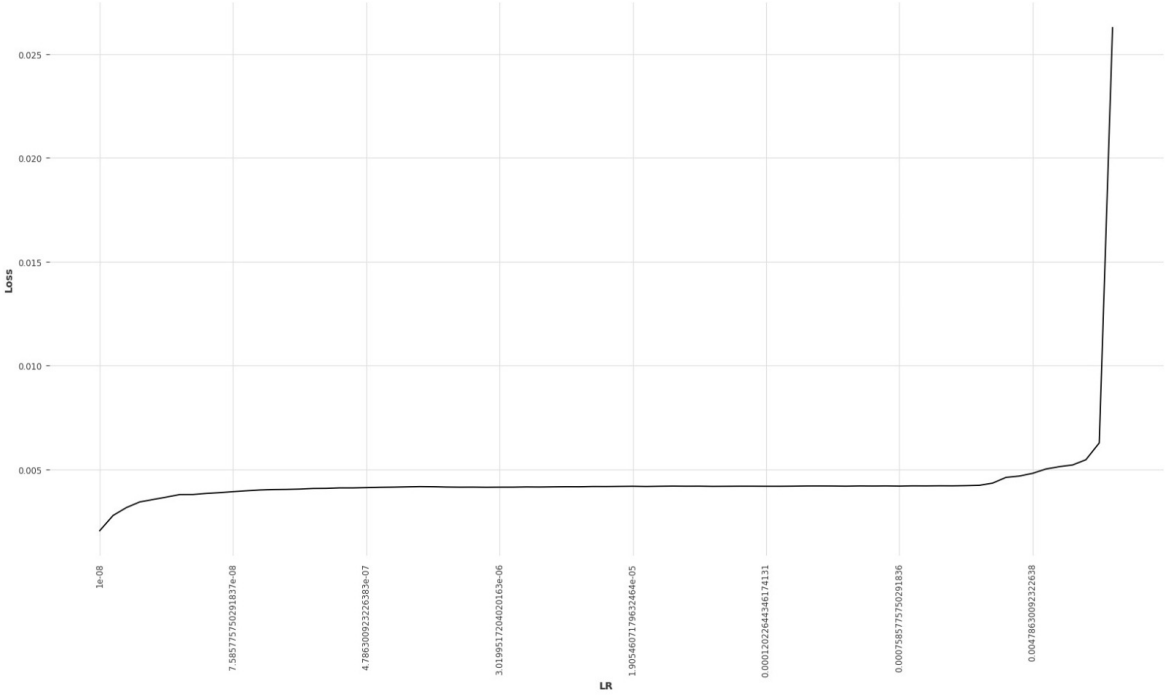

Figure A.1: The Learning Rate Finder Plot for finding the optimal Learning Rate with only Stock data

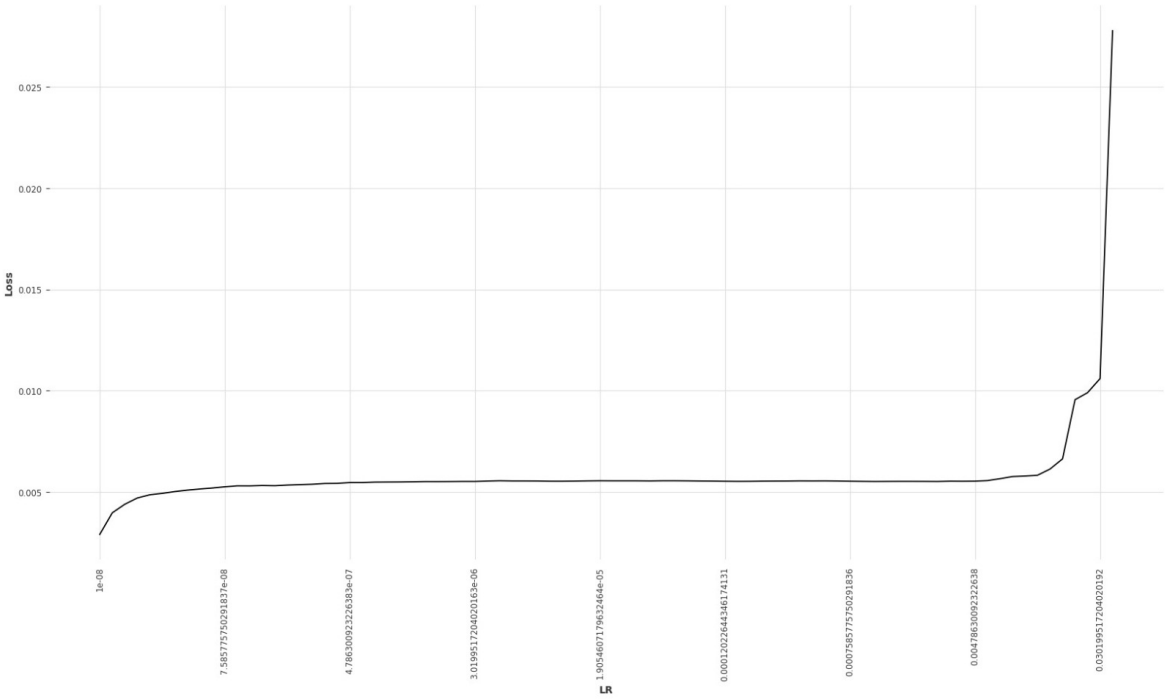

Figure A.2: The Learning Rate Finder Plot for finding the optimal Learning Rate with 10% enrichment

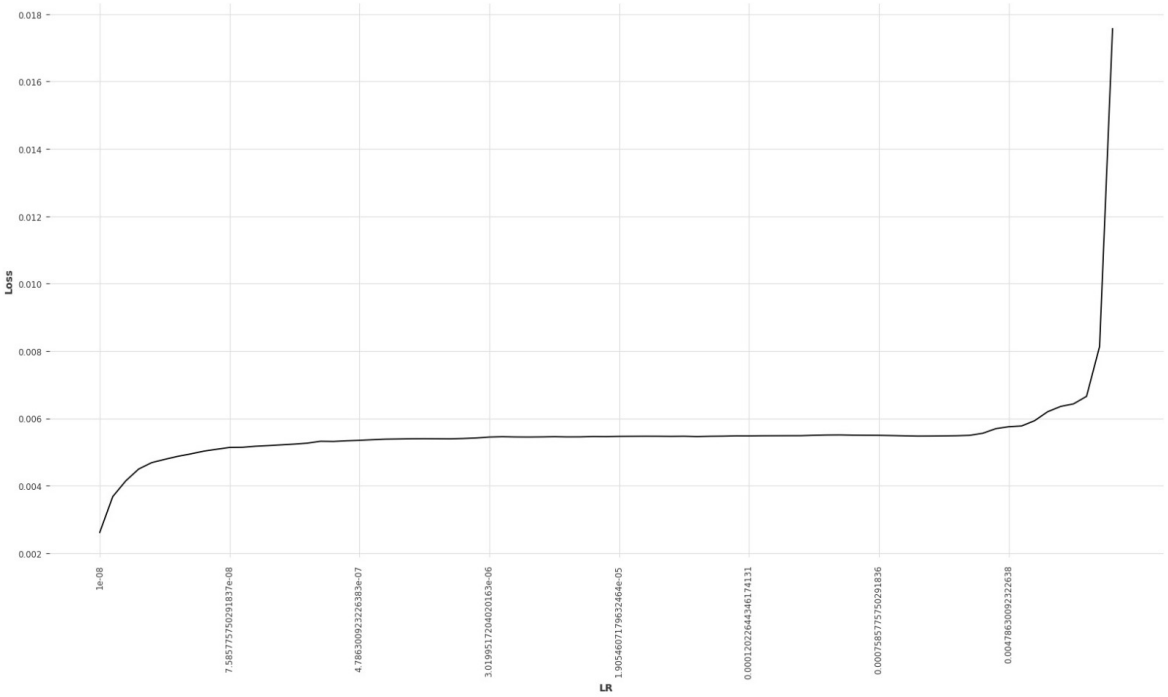

Figure A.3: The Learning Rate Finder Plot for finding the optimal Learning Rate with 40% enrichment

A.2.2 Other Convergence Plots

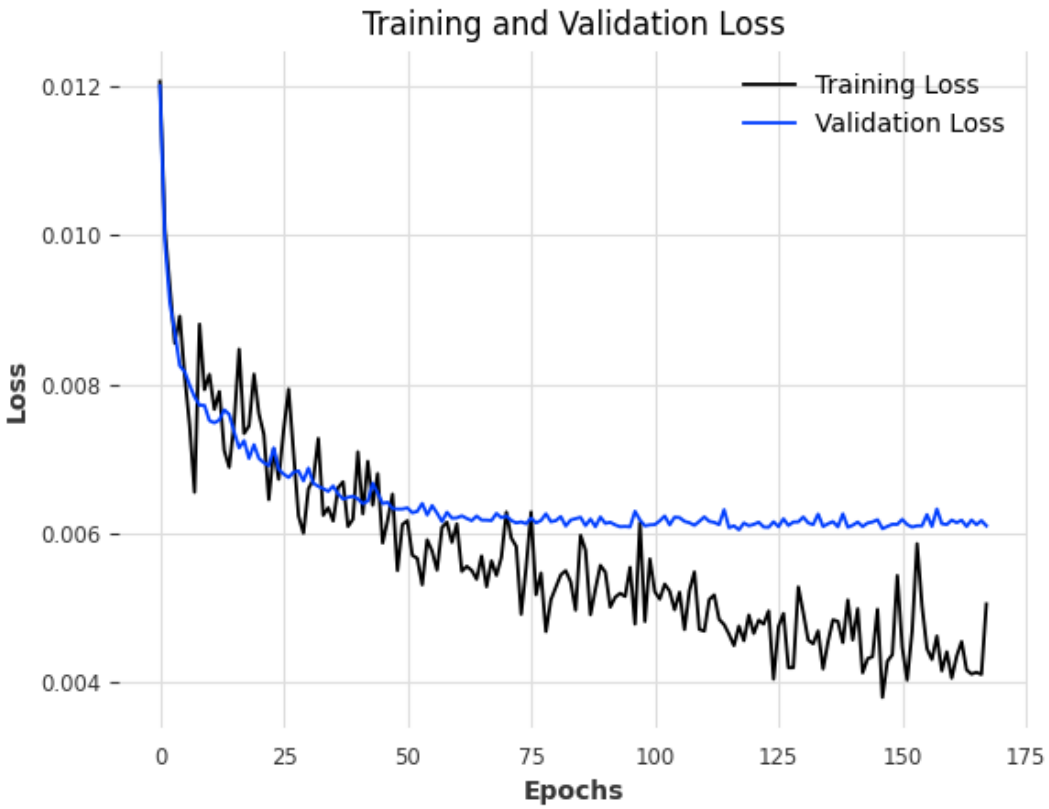

Figure A.4: Huber Loss convergence of using stock and weather data from all sources with enrichment ratio = 10% (without including darts data)

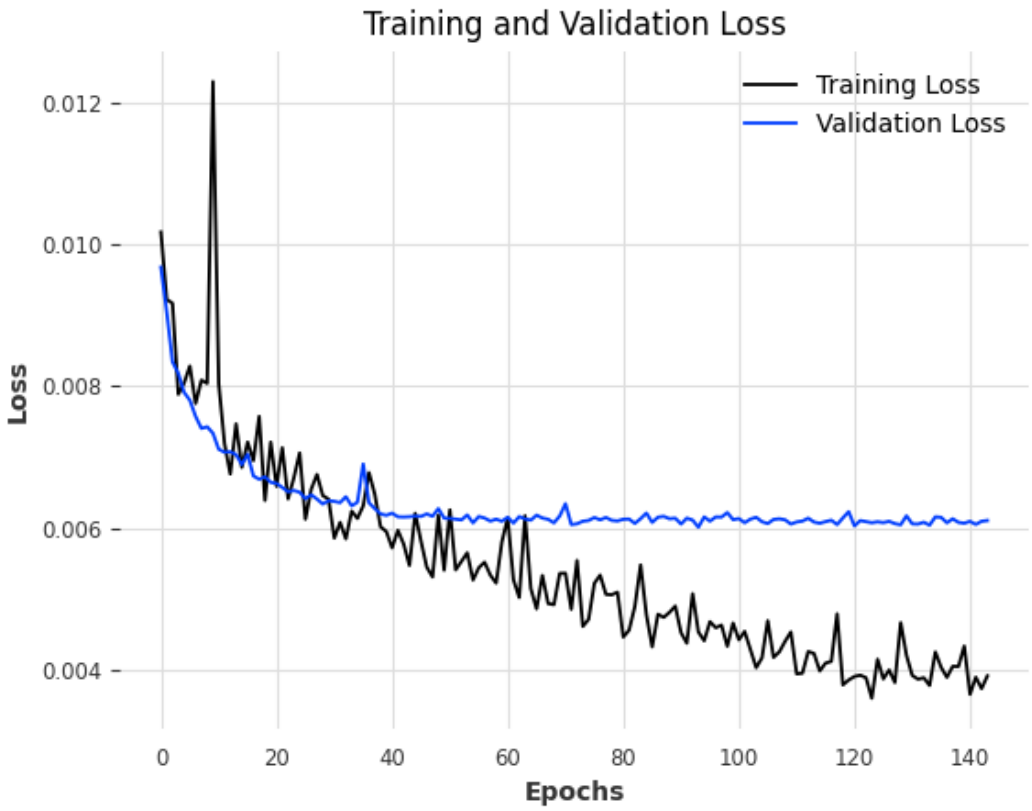

Figure A.5: Huber Loss convergence of using stock and weather data from all sources with enrichment ratio = 40% (without including darts data)

A.3 Results

Metric	stock	best_enrich_10	best_enrich_40	best_enrich_10_without_darts	best_enrich_40_without_darts	ARIMA	TimeGPT
MeanAbsoluteError	0.054915	0.056913	0.058653	0.056335	0.057229	0.057826	0.061544
MedianAbsoluteError	0.041404	0.042325	0.043986	0.041801	0.04272	0.042112	0.045115
MSELoss	0.007028	0.007726	0.008019	0.007557	0.007717	0.011656	0.010172
MedianSquaredError	0.002418	0.002465	0.002629	0.002418	0.002505	0.002499	0.00282
HuberLoss	0.003324	0.003598	0.003737	0.003532	0.003599	0.004032	0.004436
MeanLastValueError	0.076454	0.080273	0.081376	0.079467	0.080063	0.084306	0.088692
MedianLastValueError	0.053268	0.056051	0.057426	0.055509	0.05577	0.058216	0.060788
MeanTotalReturnError	0.032943	0.033677	0.040518	0.034996	0.037047	0.037076	0.045122
MedianTotalReturnError	0.017996	0.018226	0.018754	0.017985	0.018362	0.020328	0.021826
GeometricMeanDailyReturnError	0.010527	0.00789	0.008001	0.008025	0.007781	0.006707	0.008479
MeanFinalReturnError	0.044188	0.047714	0.054935	0.049469	0.050346	0.053744	0.064024
MedianFinalReturnError	0.020927	0.021858	0.022168	0.021583	0.021859	0.025577	0.027039
BackTestingProfitError	0.034215	0.026679	0.024116	0.02653	0.026262	0.020465	0.004223

Table A.1: Detailed table showing model performance measured by different metrics for all runs

A.4 Code

Our code to this project can be found in a GitHub repository, that can be found here.