Bayesian ARIMA Design Guide

This project represents a blend of classical time series analysis and modern Bayesian statistical methods, aiming to enhance forecasting capabilities by quantifying uncertainty and leveraging hierarchical modeling across multiple timeframes. In this design guide, I'll explain the mathematical foundations, the rationale behind design decisions, the challenges encountered, and the solutions devised.

Introduction

The Motivation

Time series forecasting has long been a pivotal tool in various domains, from finance and economics to meteorology and engineering. Among the myriad of models developed, the ARIMA (AutoRegressive Integrated Moving Average) model stands out for its simplicity and effectiveness. However, traditional ARIMA models operate within a deterministic framework, providing point estimates without quantifying the inherent uncertainty in predictions. This limitation often necessitates the use of supplementary methods to gauge prediction confidence, adding layers of complexity to the analysis.

Bayesian statistics is a paradigm that offers a probabilistic approach to inference, allowing for the incorporation of prior knowledge and the quantification of uncertainty through posterior distributions. By marrying Bayesian methods with ARIMA, we venture into a domain where forecasts are not just single-point estimates but probabilistic statements that capture the uncertainty and variability of future events. This integration paves the way for more informed decision-making, especially in fields like finance, where understanding risk is as crucial as predicting returns.

Background and Research Landscape

The foundation of this project is rooted in the mathematical formulations of Bayesian inference and Markov Chain Monte Carlo (MCMC) sampling techniques. Drawing inspiration from conference proceedings North Carolina State's conference paper on Bayesian auto regressors, the project leverages the capabilities of PyMC, a powerful Python library for Bayesian modeling. PyMC's flexibility and robust sampling algorithms, particularly the No-U-Turn Sampler (NUTS), facilitate efficient exploration of complex posterior distributions, making it an ideal choice for this endeavor.

While Bayesian methods have been extensively applied in various statistical modeling scenarios, their integration with ARIMA models, especially within a hierarchical structure that accommodates multiple timeframes, is relatively novel. Existing projects often treat Bayesian and ARIMA methodologies separately or focus on Bayesian extensions of simpler models. This project distinguishes itself by fusing Bayesian inference with ARIMA's time series forecasting prowess, augmented by a hierarchical ensemble framework that consolidates predictions across daily, hourly, and minute-level intervals.

Novelty and Significance

The novelty of this project lies in its comprehensive approach to forecasting by combining Bayesian ARIMA models with ensemble methods within a hierarchical structure. This amalgamation offers several advantages:

1. **Uncertainty Quantification**: Unlike traditional ARIMA models, Bayesian ARIMA provides probabilistic forecasts, enabling a nuanced understanding of prediction confidence.

2. **Hierarchical Modeling**: By accommodating multiple timeframes—daily, hourly, and minute-level—the model captures diverse temporal dynamics, enhancing forecasting accuracy and robustness.

3. **Ensemble Integration**: Combining forecasts from different models through ensemble methods like weighted averages and regression-based ensembles leverages the strengths of each model, mitigating individual weaknesses.

4. **Scalability and Modularity**: The project's design emphasizes modularity and object-oriented principles, ensuring scalability and ease of maintenance, especially when extending to additional tickers or timeframes.

In essence, this project not only advances the methodological landscape by introducing a Bayesian ARIMA framework but also provides practical tools and utilities that streamline the forecasting process, making it accessible and efficient for real-world applications.

Design Process

Conceptualizing the Bayesian ARIMA Framework

The inception of the Bayesian ARIMA model began with a thorough understanding of both ARIMA and Bayesian methodologies. ARIMA models, characterized by their autoregressive (AR), differencing (I), and moving average (MA) components, are adept at capturing temporal dependencies and trends in time series data. However, their deterministic nature often falls short in conveying the uncertainty inherent in predictions.

Bayesian statistics, on the other hand, excels in incorporating prior knowledge and quantifying uncertainty through posterior distributions. By treating ARIMA parameters as random variables with specified priors, Bayesian ARIMA models can provide a probabilistic interpretation of forecasts, enriching the decision-making process with a measure of confidence.

Mathematical Foundations

Traditional ARIMA Models

The ARIMA model is defined by three parameters: p (autoregressive order), d (differencing order), and q (moving average order). Its mathematical formulation is:

$$\phi(B)(1-B)^d y_t = \theta(B)\epsilon_t$$

Where:

- $\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \ldots - \phi_p B^p$ is the autoregressive polynomial
- $\theta(B) = 1 + \theta_1 B + \theta_2 B^2 + \ldots + \theta_q B^q$ is the moving average polynomial
- B is the backshift operator, defined as $By_t = y_{t-1}$
- ϵ_t is white noise, assumed to be normally distributed with mean zero and variance σ^2

The model captures both short-term dependencies (through AR and MA terms) and long-term trends (through differencing).

Bayesian ARIMA Extension

In extending ARIMA to a Bayesian framework, the parameters ϕ, θ, and σ are treated as random variables with specified prior distributions. The Bayesian ARIMA model can thus be represented as:

$$\phi_i \sim \mathcal{N}(0, 10) \quad \text{for } i = 1, \ldots, p$$

$$\theta_j \sim \mathcal{N}(0, 10) \quad \text{for } j = 1, \ldots, q$$

$$\sigma \sim \text{HalfNormal}(1)$$

$$y_t \sim \mathcal{N}(\mu_t, \sigma)$$

$$\mu_t = \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j}$$

This probabilistic formulation allows the incorporation of prior beliefs about the parameters and facilitates the estimation of their posterior distributions given the observed data.

Implementing the BayesianARIMA and BayesianSARIMA Classes

BayesianARIMA Class

The `BayesianARIMA` class serves as the backbone of the Bayesian ARIMA model. Its implementation required translating the mathematical formulation into a probabilistic model using PyMC.

Key Components and Design Decisions:

1. **Initialization Parameters**:

- **Name**: Serves as an identifier for saving and loading models.
- p**,** d**,** q: Define the ARIMA order, directly influencing the model's capacity to capture temporal dependencies.

2. **Training Method**:

- **Differencing**: Applied to achieve stationarity, a prerequisite for ARIMA modeling. The differenced series is then used for parameter estimation.
- **Priors**: Normal distributions with mean zero and large variance (sigma=10) are chosen for ϕ and θ\thetaθ coefficients to express minimal prior information, allowing the data to predominantly influence the posterior.
- **Noise Term (**σ**)**: Modeled using a Half-Normal distribution to ensure positivity.
- **Likelihood**: The observed data y_t is modeled as a Normal distribution with mean μ_t and standard deviation σ, where μ_t aggregates the AR and MA components.

3. **Prediction Method**:

- Utilizes posterior means of the parameters to generate forecasts.
- Incorporates sampled error terms (ϵ_t) from the posterior to account for uncertainty in MA components.

4. **Model Persistence**:

- Implements `save` and `load` methods using the `dill` library to serialize the model and its trace, ensuring efficient storage and retrieval.

Mathematical Rationale:

The choice of priors reflects a non-informative stance, allowing the data to shape the posterior distributions. The Normal priors for ϕ and θ are standard in Bayesian regression models due to their mathematical convenience and conjugacy properties. The Half-Normal prior for σ ensures positivity without imposing undue constraints, aligning with the nature of variance parameters.

The construction of μ_t as the sum of AR and MA components mirrors the deterministic ARIMA formulation but within a probabilistic framework, facilitating uncertainty quantification through the posterior distributions of the coefficients.

BayesianSARIMA Class

Building upon the `BayesianARIMA` class, the `BayesianSARIMA` class introduces seasonal components, accommodating models like SARIMA (Seasonal ARIMA) that account for periodic fluctuations in the data.

Key Enhancements:

1. **Seasonal Parameters**:

- P, D, Q: Define the seasonal ARIMA order, capturing periodic dependencies at lag multiples of the seasonal period mmm.

2. **Seasonal Differencing**:

- Applies seasonal differencing to remove recurring patterns, enhancing stationarity.

3. **Additional Priors**:

- **Seasonal AR and MA Coefficients (Φ, Θ)**: Modeled using Normal distributions, similar to non-seasonal coefficients, ensuring consistency in prior specification.

4. **Likelihood Augmentation**:

- Incorporates seasonal AR and MA terms into μ_t, enriching the model's capacity to capture complex temporal dynamics.

Mathematical Extensions:

The seasonal components are integrated into the model as follows:

$$\mu_t = \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \sum_{I=1}^{P} \Phi_I y_{t-I\cdot m} + \sum_{J=1}^{Q} \Theta_J \epsilon_{t-J\cdot m}$$

This extension allows the model to account for both non-seasonal and seasonal dependencies, providing a more nuanced representation of the underlying data patterns.

Hierarchical Modeling Across Multiple Timeframes

Financial markets exhibit behaviors that manifest differently across various timeframes. Daily, hourly, and minute-level analyses each capture unique aspects of market dynamics. Recognizing this, the `HierarchicalModel` class orchestrates Bayesian SARIMA models across these timeframes, enabling comprehensive forecasting.

Design Choices and Justifications:

1. **Multiple Timeframes**:

- **Daily**: Captures broader market trends and weekly seasonality.

- **Hourly**: Reflects intra-day patterns and volatility.
- **Minute-Level**: Targets high-frequency trading dynamics and rapid price fluctuations.

2. **Seasonality Considerations**:

- **Daily**: Weekly seasonality ($m = 5$) aligns with the standard trading week.
- **Hourly**: Intra-day seasonality ($m = 6$) captures hourly trading patterns.
- **Minute-Level**: Minimal seasonality ($m = 1$) due to the high-frequency nature of data.

3. **Ensemble Integration**:

- Combines forecasts from different timeframes to produce a consolidated prediction, leveraging the strengths of each model and mitigating individual biases.

4. **Memory Management**:

- The `memory_save` parameter allows for serializing trained models to disk, optimizing memory usage and enabling scalability.

Mathematical and Computational Considerations:

The hierarchical structure ensures that each timeframe operates semi-independently, allowing models to specialize in capturing the nuances of their respective temporal scales. The ensemble methods, whether weighted averages or regression-based combinations, mathematically integrate these diverse forecasts to yield a unified prediction.

Developing Ensemble Methods

Ensemble methods are pivotal in aggregating forecasts from multiple models, enhancing accuracy and robustness. Two primary ensemble approaches were implemented:

1. **Weighted Average Ensemble**:

- Assigns predefined weights to each model's forecast, computing a weighted sum to derive the final prediction.
- **Mathematical Representation**:

$$\hat{y} = w_1 \cdot \hat{y}_1 + w_2 \cdot \hat{y}_2 + w_3 \cdot \hat{y}_3$$

Where w_i are the weights assigned to each model's forecast $\hat{y}_i$.

2. **Regression-Based Ensemble**:

- Utilizes a regression model (e.g., Linear Regression) to learn optimal weights based on historical performance, allowing the ensemble to adaptively adjust weights based on data patterns.
- **Mathematical Representation**:

$$\hat{y} = \beta_0 + \beta_1 \cdot \hat{y}_1 + \beta_2 \cdot \hat{y}_2 + \beta_3 \cdot \hat{y}_3$$

Where β_i are the regression coefficients learned from training data.

Rationale Behind Ensemble Selection:

The choice between weighted average and regression-based ensembles hinges on the balance between simplicity and adaptability. Weighted averages offer computational efficiency and ease of interpretation, making them suitable for scenarios where model performances are relatively stable. Regression-based ensembles, while computationally more intensive, provide the flexibility to adjust weights based on evolving data patterns, potentially yielding superior forecasting performance in dynamic environments.

Utility Modules: Bridging Data and Models

The complexities of time series data handling and the need for precise time delta calculations necessitated the development of specialized utility modules. These modules ensure that data aligns seamlessly with model expectations, enhancing the overall robustness of the forecasting framework.

Mathematical and Logical Foundations:

The class employs logical constructs to determine trading periods and calculate overlapping intervals between start and end times. By iterating over trading days and summing active trading seconds, it ensures that time delta calculations are both accurate and contextually relevant to trading activities.

Preprocessing Module

Data preprocessing is a critical step in time series modeling, ensuring that the data adheres to the assumptions required by the model. The `preprocessor.py` module encapsulates a comprehensive pipeline for preparing raw stock data.

Key Steps and Design Choices:

1. **Data Loading and Cleaning**:

- Imports stock data from CSV files, focusing on the 'Close' prices to maintain consistency across different timeframes.
- Implements forward-fill strategies to handle missing values, preserving data continuity without introducing bias.

2. **Stationarity Checks and Differencing**:

- Utilizes the Augmented Dickey-Fuller (ADF) test to assess stationarity, a foundational assumption in ARIMA modeling.
- Applies differencing operations to achieve stationarity, incrementally increasing the differencing order until the series passes the ADF test or reaches a maximum threshold.

3. **Error Handling and Reporting**:

- Incorporates checks to prevent excessive differencing, which can lead to data sparsity and model instability.
- Provides informative error messages to guide users in resolving data-related issues.

Mathematical Rationale:

Stationarity is paramount in time series modeling, ensuring consistent statistical properties over time. The ADF test serves as a formal mechanism to assess this assumption, while differencing operations address non-stationarity by removing trends and seasonality. Log returns further enhance stationarity by normalizing price data, a common practice in financial time series analysis.

Given the high frequency of minute-level data, efficient data fetching and storage are critical to prevent memory overloads and ensure swift model training. The chunked fetching strategy balances the need for comprehensive data with computational feasibility, while also complying with limits imposed by the `yfinance` API for maximum data points in a single GET request.

Challenges

The development of the Bayesian ARIMA framework was characterized by a series of iterative refinements, each addressing specific challenges and enhancing the model's robustness.

Challenge 1: Mathematical Formulation and Tensor Alignment

Translating the ARIMA equations into a probabilistic model using PyMC introduced complexities related to tensor shapes and alignment. Misalignments between lagged terms and observations led to runtime errors and incorrect model specifications. I had many bugs with this issue, as initially I hadn't paid as close attention as I maybe needed to when it came to vector and tensor operations. Some of these misalignment issues, especially with the `BayesianSARIMA` class, would only show up on edge cases for certain orders for `p`, `q`, `P`, or `Q`, which made them difficult to identify.

To address this issue, I worked on using static indexing. I implemented precise indexing strategies to align lagged observations with the current time step, ensuring that each term in the summation accurately corresponds to the intended lag. This was fixed as a mathematical constant dependent on the input order. I utilized PyTensor's tensor operations to dynamically adjust tensor shapes, accommodating varying AR and MA orders. During debugging, I also conducted unit tests with synthetic data to validate tensor operations and ensure that the mathematical formulations translated accurately into the probabilistic model, and that I considered the ranges of possible orders well.

The main lesson I learned from this that I took on the rest of my code was when working with tensors, vectors and matrices, to very carefully follow the necessary and ideal shapes and sizes of each vector or matrix. To help with this in future code I wrote, I added comments denoting the shape and size of vectors and tensors, so that way any math operations I did between them would line up.

Example: When configuring the AR component, ensuring that y_{t-i} aligns correctly with $\phi_i$required slicing the differenced series appropriately. Misalignment here could distort the relationship between the AR coefficients and their corresponding lagged terms, as well as throw issues for trying to add together different sized tensors.

Challenge 2: Handling Long and Short Term Data

Minute-level data, while rich in information, introduced significant computational challenges due to its high frequency and volume. Processing such data within the Bayesian framework, which is computationally intensive, risked long training times and potential memory issues. In addition, small frequency data may run into numerical instability if trying to predict far in the future, however, long-interval data may miss out on a lot of the finer details within a timestep.

This was an issue I came up with during the system design phase, as time series models tend to make a tradeoff based on the period of time or range for which they are forecasting. However, I wanted to make a system that was efficient yet dynamic in the ranges of times it could predict. Therefore, I wanted to come up with a solution that was dynamic, and that it could either predict short-term trends as well as long-term trends. The solution I came up with this was to trade multiple models and then use ensemble methods to combine them. I believe this approach of taking multiple different resolution models and combining them allows the model to predict both the fine grain as well as large grain, future variations and data. Doing so was difficult, as I needed to balance computational complexity, as well as create methods by which I can combine these forecasts. To solve this issue, I created the ensemble abstract class. This abstract class provides a framework which can be easily extended and manipulated to combine forecasts easily. I also made this framework extensible, allowing for methods as simple as weighted averages to those as complex as machine learning methods.

Challenge 3: Selecting Optimal Hyperparameters for MCMC Sampling

The efficiency and accuracy of MCMC sampling are highly sensitive to hyperparameters like the number of draws, tuning steps, and target acceptance rate. Poorly chosen hyperparameters could result in sampler convergence issues or inefficient exploration of the posterior space. In addition, selecting the order of the ARIMA model was a highly manual process that was involved, as you had to look at the autocorrelation function and partial autocorrelation function plots and holistically select an order.

To fix this, I experimented with many different hyperparameters, trying to balance computational efficiency (since I was just running on a 5 year old laptop on the CPU) with accuracy. For the Bayesian order selection, I leveraged `pmdarima`, a library that wrote a wrapper for the `auto_arima` method from R, which automatically conducts tests to determine optimal orders. It runs tests like Kwiatkowski–Phillips–Schmidt–Shin, Augmented Dickey-Fuller or Phillips–Perron to find the optimal differencing `d`, and then automates grid search or optimization methods to select other hyperparameters. This made my program a lot more automated and hands-off, which helps with reusability. This also makes it more deterministic and less dependent on holistic guesses.

Challenge 4: Time Period Handling

One of the main issues that arose with the software that I didn't initially expect to be an issue was handling the time demarcations for the data. Handling time series data that only exists for periods of a day, or during trading hours, was difficult, as it meant that time labelling and propagating code that normally is fairly trivial to implement wouldn't work. You could not make the assumption that going forward 1 hour in time would be a valid trading hour, and dealing with the end of trading day meant that some time periods overlapped multiple days. In addition, with trading days and intervals not always easily dividing into the day (for example, there are 6.5 hours in a trading day from `9:00 am` to `4:30 pm`), these were also difficult edge cases I had to deal with.

The solution for this was the creation of the `TradingTimeDelta` utility. This class was created to handle the time periods and intervals that were not easily divisible by the day. It took in typical time deltas, and then calculated the time deltas that were valid for trading hours. This was then used in the `HierarchicalModel` class to calculate the time deltas for the different timeframes. This was a difficult issue to solve, as it required a lot of edge case testing and debugging to ensure that the time deltas were calculated correctly. In addition, I implemented a lot of auxiliary functionality through static methods to ensure that the time deltas were calculated correctly, and to determine time series labels and valid trading periods. By bundling all of this functionality into a single class, I was able to ensure that the time deltas were calculated correctly and that the time series data was correctly labelled and handled. This also increased the modularity of the code, as the time delta calculations were abstracted away from the main code, and helped make it easier to develop unit tests to ensure that edge cases were handled correctly.

Results

Comprehensive Forecasting Capabilities

The culmination of this design process is a sophisticated Bayesian ARIMA framework capable of delivering probabilistic forecasts across multiple timeframes. The hierarchical structure, encompassing daily, hourly, and minute-level models, ensures that the system captures diverse temporal dynamics inherent in financial markets.

1. **Probabilistic Forecasts**:

- Each Bayesian SARIMA model produces not just point estimates but entire posterior distributions for future values, encapsulating the uncertainty and variability in predictions.
- This probabilistic nature enhances decision-making, allowing stakeholders to gauge risks and make informed choices.

2. **Hierarchical Integration**:

- By operating across daily, hourly, and minute-level intervals, the framework provides a holistic view of market dynamics, capturing both macro and micro-level trends.
- The ensemble methods combine these forecasts, leveraging their collective strengths to produce more accurate and reliable predictions.

3. **Model Persistence and Scalability**:

- The implementation supports saving and loading trained models using the `dill` library, facilitating efficient storage and retrieval.
- The modular and object-oriented design ensures scalability, allowing for the seamless addition of new tickers or timeframes without overhauling the existing structure.

Example Runs

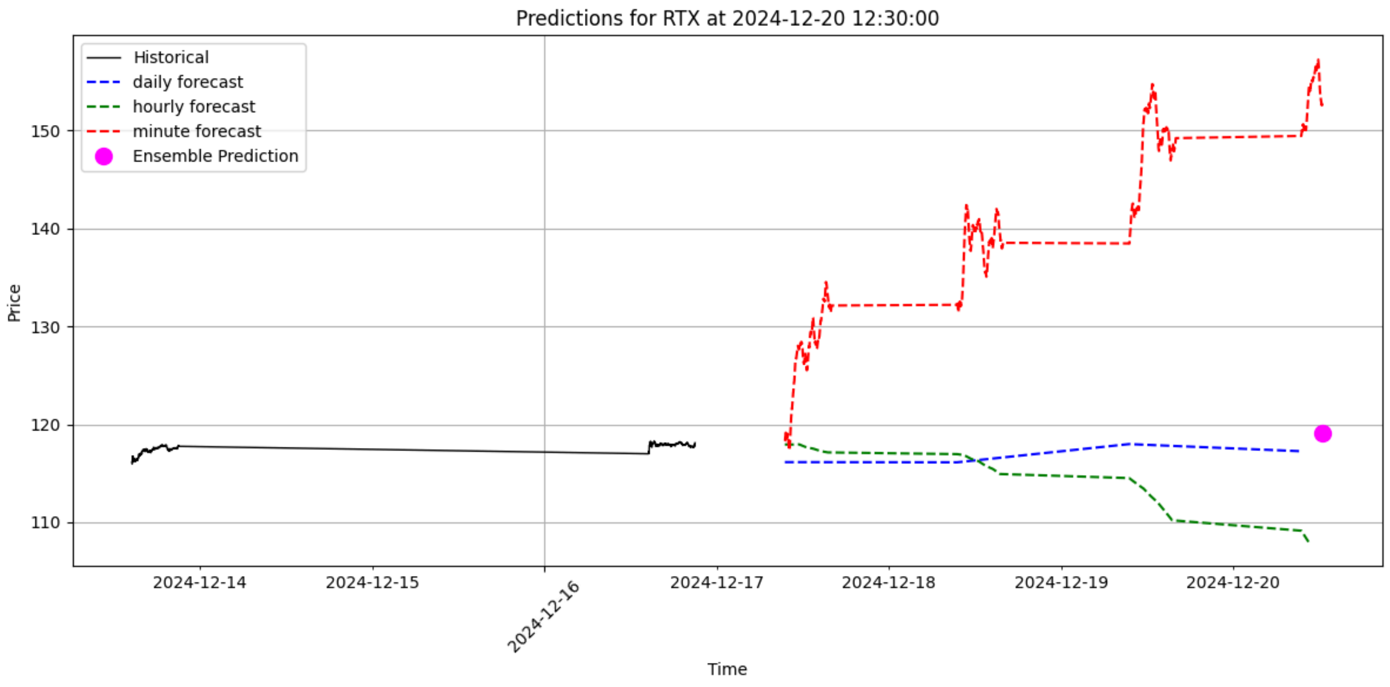
Drawbacks

One of the main drawbacks of the system is that it is computationally intensive. The Bayesian ARIMA model is already computationally intensive, and when you add in the fact that you are running multiple models at different timeframes, it can be very slow. This is especially true when you are running on a CPU, as I was. This is a drawback, as it means that the system is not as efficient as it could be, and that it is not as scalable as it could be. While I did implement some memory saving techniques, such as serializing the models to disk and splitting the NUTS sampling into multiple cores, this only goes so far. Ideally, if developed into an actual module and not just a proof of concept, it would be run on a GPU, which would speed up the process significantly.

What this meant for me however, is that the examples I ran were on a limited dataset (due to both computational complexity issues and yfinance's API limits), as well as on limited sample draws. This means that the results I got were not as accurate as they could be, and that the model was not as robust as it could be. However, the results I got were still promising, and showed that the model was able to predict the stock price with a reasonable degree of accuracy. In addition, by running the system twice on different sample numbers, it can be seen that the model is properly learning, as the more samples provided to it, the tighter the posterior distribution becomes, as well as the more accurate the predictions become.

4 Total Draws

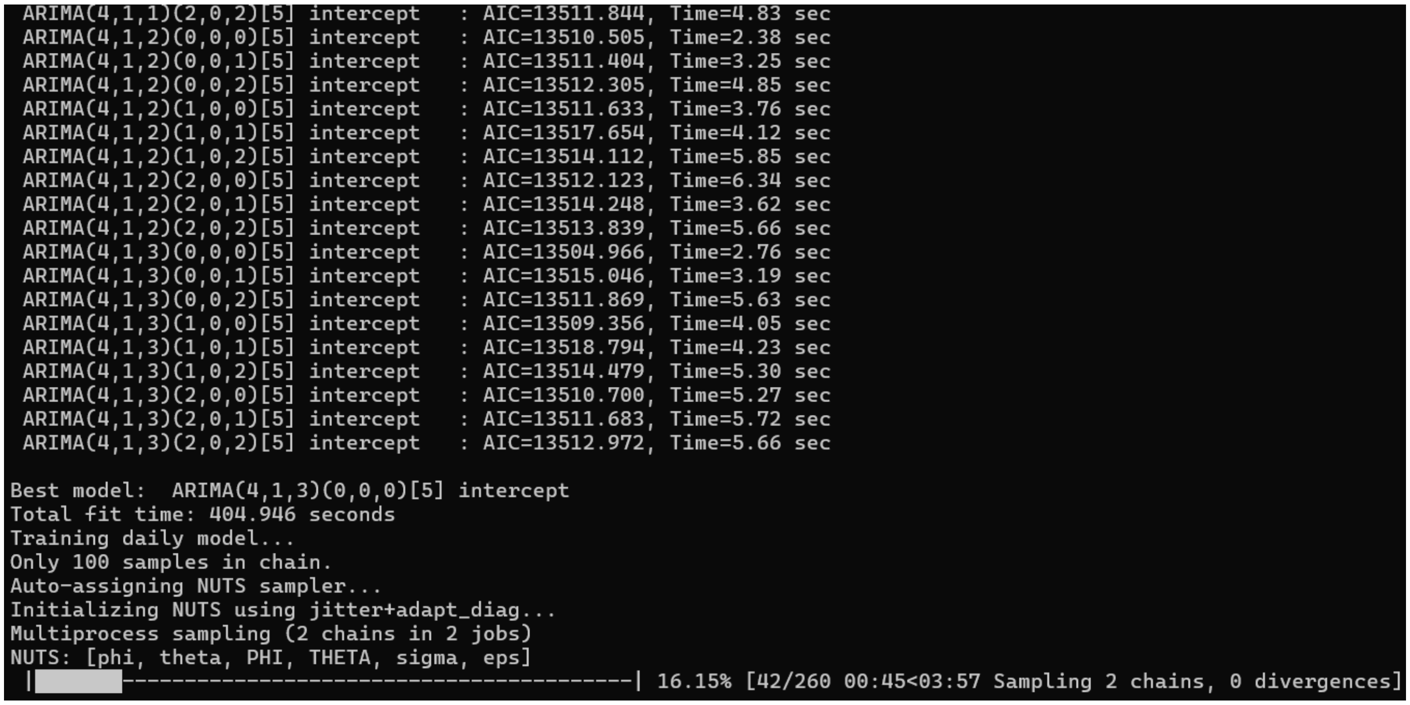
I ran an example on the `RTX` stock, with 2 draws and 2 tuning steps. After training the model on the data, I then forecasted the stock price for about 3.5 days into the future, for December 20th, 2024 at 12:30 pm. The result image is as follows:



As you can see, while the result isn't the most accurate, the model is able to predict the stock price with a reasonable degree of accuracy. In addition, the minute model seems to be learning some daily seasonality, which indicates that the seasonal part of the model is learning correctly.

260 Total Draws

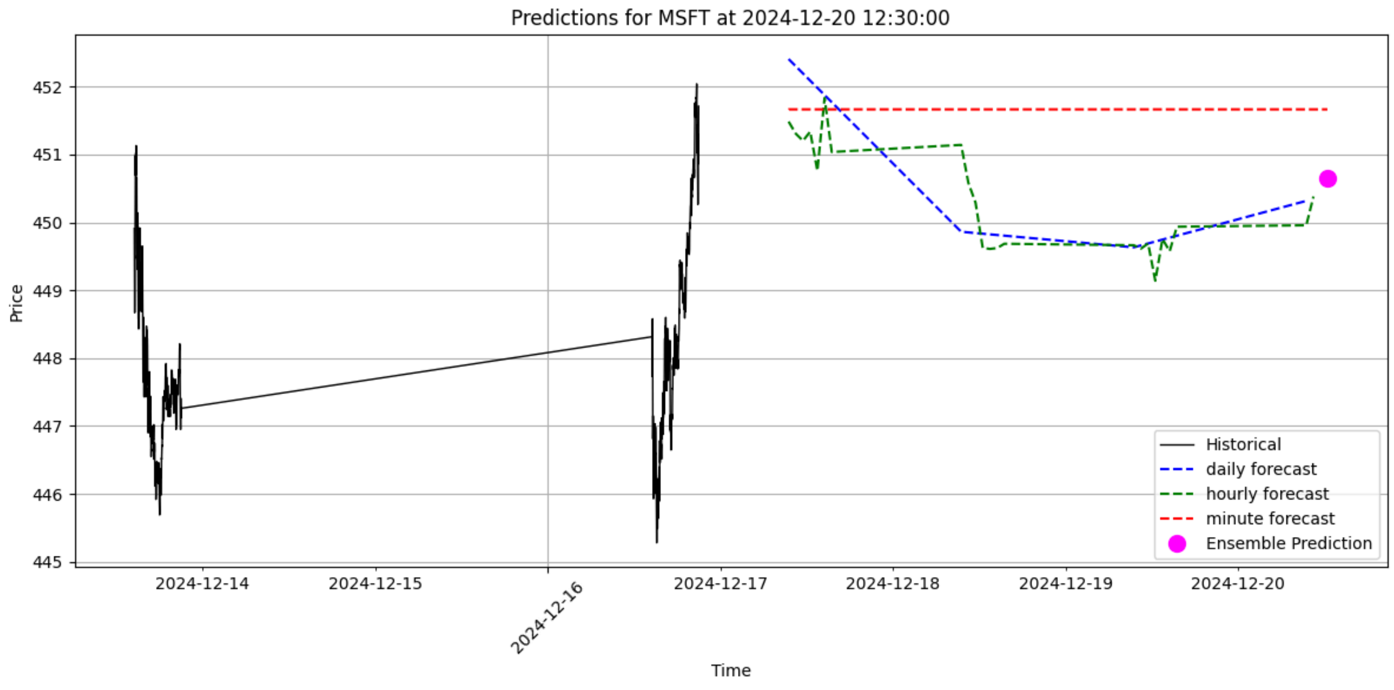
I ran another example on Microsoft's stock, with 130 draws and 130 tuning steps. After training the model on the data, I then forecasted the stock price for about 3.5 days into the future, for December 20th, 2024 at 12:30 pm. The training took about 10 hours and looks like this:



```
 ARIMA(4,1,1)(2,0,2)[5] intercept   : AIC=13511.844, Time=4.83 sec
 ARIMA(4,1,2)(0,0,0)[5] intercept   : AIC=13510.505, Time=2.38 sec
 ARIMA(4,1,2)(0,0,1)[5] intercept   : AIC=13511.404, Time=3.25 sec
 ARIMA(4,1,2)(0,0,2)[5] intercept   : AIC=13512.305, Time=4.85 sec
 ARIMA(4,1,2)(1,0,0)[5] intercept   : AIC=13511.633, Time=3.76 sec
 ARIMA(4,1,2)(1,0,1)[5] intercept   : AIC=13517.654, Time=4.12 sec
 ARIMA(4,1,2)(1,0,2)[5] intercept   : AIC=13514.112, Time=5.85 sec
 ARIMA(4,1,2)(2,0,0)[5] intercept   : AIC=13512.123, Time=6.34 sec
 ARIMA(4,1,2)(2,0,1)[5] intercept   : AIC=13514.248, Time=3.62 sec
 ARIMA(4,1,2)(2,0,2)[5] intercept   : AIC=13513.839, Time=5.66 sec
 ARIMA(4,1,3)(0,0,0)[5] intercept   : AIC=13504.966, Time=2.76 sec
 ARIMA(4,1,3)(0,0,1)[5] intercept   : AIC=13515.046, Time=3.19 sec
 ARIMA(4,1,3)(0,0,2)[5] intercept   : AIC=13511.869, Time=5.63 sec
 ARIMA(4,1,3)(1,0,0)[5] intercept   : AIC=13509.356, Time=4.05 sec
 ARIMA(4,1,3)(1,0,1)[5] intercept   : AIC=13518.794, Time=4.23 sec
 ARIMA(4,1,3)(1,0,2)[5] intercept   : AIC=13514.479, Time=5.30 sec
 ARIMA(4,1,3)(2,0,0)[5] intercept   : AIC=13510.700, Time=5.27 sec
 ARIMA(4,1,3)(2,0,1)[5] intercept   : AIC=13511.683, Time=5.72 sec
 ARIMA(4,1,3)(2,0,2)[5] intercept   : AIC=13512.972, Time=5.66 sec

Best model:  ARIMA(4,1,3)(0,0,0)[5] intercept
Total fit time: 404.946 seconds
Training daily model...
Only 100 samples in chain.
Auto-assigning NUTS sampler...
Initializing NUTS using jitter+adapt_diag...
Multiprocess sampling (2 chains in 2 jobs)
NUTS: [phi, theta, PHI, THETA, sigma, eps]
 |███████-----------------------------------------| 16.15% [42/260 00:45<03:57 Sampling 2 chains, 0 divergences]
```

Here is the result image:



As you can see, the result is much more accurate than the previous one. The model is able to predict the stock price with a much higher degree of accuracy, and the minute model seems to be learning some daily seasonality, which indicates that the seasonal part of the model is learning correctly. This shows that the model is learning correctly, and that the more samples provided to it, the more accurate the predictions become.

Concepts Learned

Time Series Modeling

One of the most rewarding aspects of this project was diving deep into the nuances of time series modeling. It marked a significant shift in my understanding—from a deterministic approach to a probabilistic mindset. This transition was critical as I explored how Bayesian methods could seamlessly integrate with ARIMA models, enhancing both flexibility and interpretability.

A key concept I tackled was stationarity, the backbone of time series analysis. I learned that stationarity ensures consistent statistical properties over time, a requirement for most forecasting models. Achieving this often involves differencing, and through experimentation, I developed a strong appreciation for the trade-offs involved when adjusting the differencing order. Too little differencing can leave trends in the data, while too much can strip away valuable signals.

Seasonal dynamics presented another fascinating challenge. I delved into seasonal ARIMA models, which are specifically designed to capture periodic fluctuations and recurring patterns within data. Understanding the mathematical representation of these seasonal components was particularly exciting when integrating them into a Bayesian framework. It became clear how the seasonal elements could enhance model performance when periodic behavior drives the underlying data.

Model selection and optimization were central to ensuring my models performed well. I used tools like pmdarima and statistical tests to identify the most appropriate model parameters. Balancing model complexity with predictive power turned out to be a nuanced process—while flexible models are capable of capturing more intricate patterns, they can also introduce computational challenges and risks of overfitting. Fine-tuning this balance was a valuable lesson.

Bayesian Inference

Combining ARIMA with Bayesian methods opened up an entirely new perspective on modeling uncertainty. To integrate these approaches, I first had to deepen my understanding of Bayesian inference. The concepts of prior and posterior distributions were pivotal here. Priors express our beliefs about parameters before observing the data, while posteriors update these beliefs as new information becomes available. It was both humbling and empowering to see how Bayesian methods allow uncertainty to be explicitly modeled and updated.

To estimate posterior distributions, I explored Markov Chain Monte Carlo (MCMC) sampling techniques, particularly Hamiltonian Monte Carlo (HMC) and the No-U-Turn Sampler (NUTS). These methods efficiently navigate high-dimensional posterior spaces where simpler approaches might fail. However, they come with their challenges—I had to address sampler convergence issues and experiment with hyperparameters to optimize performance.

PyMC became my tool of choice for implementing these Bayesian models. Its intuitive syntax and powerful probabilistic programming capabilities allowed me to define complex hierarchical models. Integrating ARIMA components into PyMC frameworks turned out to be a game-changer, enabling me to blend traditional time series forecasting with modern probabilistic methods. The hands-on experience I gained with PyMC was instrumental in bridging the theoretical aspects of Bayesian statistics with practical applications.

Data Handling

I also realized that effective time series forecasting depends heavily on how well the data is prepared. Data integrity was non-negotiable. Missing values, gaps, and inconsistencies in the time series could distort model predictions, so I spent considerable time mastering techniques for data cleaning. Ensuring the continuity of the time series and validating its consistency across different timeframes was essential.

Beyond cleaning, I explored data transformations to improve model performance. Applying log returns helped stabilize variance in price data, making patterns more discernible for the model. However, this introduced another layer of complexity—balancing transformations for statistical benefits while keeping forecasts interpretable and actionable for stakeholders.

Final Thoughts

This project not only strengthened my understanding of time series modeling but also pushed me to embrace the probabilistic nature of real-world forecasting problems. Combining Bayesian methods with ARIMA has opened the door to more robust, flexible, and transparent forecasts. Whether it’s capturing uncertainty through posterior distributions or fine-tuning seasonal components, this approach has proven both innovative and practical. The experience was a testament to how blending classic statistical models with modern probabilistic tools can take time series forecasting to the next level. I also learned a lot about software design and development, as I had to create a system that was both efficient and dynamic, and that could handle a lot of edge cases. I followed a lot of software design principles, such as modularity, abstraction, and extensibility, to ensure that the system was robust and scalable.

Future Improvements

If I were to continue this project, there are several areas I would focus on to enhance the model's performance and usability. First, I would explore more advanced ensemble methods, such as boosting and bagging, to further improve forecast accuracy. These methods could help mitigate individual model biases and enhance the robustness of the ensemble predictions. Having a more robust ensemble method would also help with the issue of computational complexity, as it would allow for more accurate predictions with fewer models, or models that had been trained less. More dynamic weighted averages can also help consider edge cases where one model may diverge and become invalid.

I would also like to explore more advanced Bayesian optimization techniques to fine-tune hyperparameters and model structures. By automating the hyperparameter tuning process, I could optimize model performance more efficiently and effectively. This would also help with the issue of computational complexity, as it would allow for more accurate predictions with fewer models, or models that had been trained less. Many of my hyperparameters were chosen manually, and I believe that automating this process would lead to more accurate and robust models. In addition, doing more case studies and analysis on outputs versus validation or test sets can help me better understand the model's performance and how it can be improved.

Finally, and probably most importantly, I would work on improving the computational efficiency of the model. This could involve optimizing the code for parallel processing, leveraging GPU acceleration, or implementing more efficient sampling algorithms. By reducing the computational burden of the model, I could scale it to handle larger datasets and more complex forecasting scenarios. This would also help with the issue of computational complexity, as it would allow for more accurate predictions with fewer models, or models that had been trained less. Many of the issues I faced with the model were due to computational complexity, and I believe that improving the efficiency of the model would lead to more accurate and robust predictions. My current model isn't parallelized, and I believe that parallelizing the model would help with the issue of computational complexity. I would also like to explore more advanced sampling algorithms, such as Hamiltonian Monte Carlo, to improve the efficiency of the model. By improving the efficiency of the model, I could scale it to handle larger datasets and more complex forecasting scenarios. This would also help with the issue of computational complexity, as it would allow for more accurate predictions with fewer models, or models that had been trained less. Many of the issues I faced with the model were due to computational complexity, and I believe that improving the efficiency of the model would lead to more accurate and robust predictions.

Overall, this project has been a valuable learning experience, and I look forward to further exploring the intersection of Bayesian statistics and time series modeling in the future. I believe that by addressing these areas of improvement, I can create a more accurate, robust, and scalable forecasting framework that can be applied to a wide range of real-world scenarios. It was a great proof of concept, and I believe that with more work, it could be a very powerful tool for time series forecasting.