Predicting the Future and Quantifying the Uncertainity: Deep Dive into Predicting Price Movement Of Securities Using a Non Parametric Bayesian Approach

Aman Bhattacharyya
201711295

Supervised by Dr. Peter Gracar

Submitted in accordance with the requirements for the
module MATH5872M: Dissertation in Data Science and Analytics
as part of the degree of

Master of Science in Data Science and Analytics

The University of Leeds, School of Mathematics

September 2024

UNIVERSITY OF LEEDS

School of Mathematics

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

Academic integrity statement

Name Aman Bhattacharyya

Student ID 201711295

Abstract

Predicting movement of stock prices or other equities related to financial markets is crucial for making informed investment decisions and managing financial risks. Accurate predictions can not only guide trading strategies but also enhance market efficiency by reducing the risk of incorrectly pricing the assets.

This dissertation dives deep into the use of Gaussian Process Regression (GPR) for predicting the price or value of different securities, GPR is a method that provides not only point estimates but also confidence intervals with standard error offering a clear understanding of uncertainty while making predictions.

Through extensive literature review and analysis combined with exploration of various kernel functions, prior distributions, and training data time frames, the various possibilities were examined to optimize the GPR model. Additionally, the study also explores advanced applications of GPR in financial modeling by studying various literature, highlighting its potential to outperform traditional methods. This research contributes to the growing body of work on GPR, demonstrating its effectiveness in producing reliable predictions which are easy to understand.

Acknowledgement

I would like to express my deepest gratitude to Dr. Peter Gracar, whose guidance and support have been very important throughout the course of this dissertation. His insightful suggestions about the study materials and the topic along with his encouragement pushed me in the right direction, enabling me to navigate the complexities of this topic with confidence. His mentorship has been crucial in shaping the outcomes of this work, and I am sincerely thankful for his contribution to my academic journey.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Stock Market Price Movement

All major global economies have established stock markets where shares of publicly listed companies are traded between sellers and buyers at agreed-upon prices. The interaction of supply and demand for these equities results in price fluctuations. Stock price movements are influenced by various quantitative and qualitative factors, including company performance, large buy orders from institutional investors, speculative news, overall market conditions, global events, and rumors. Consequently, while it is impossible to predict stock prices with absolute certainty, it is feasible to make informed predictions based on historical trends, changes in trading volume, and other relevant indicators.
Stock price movements are subject to a complex interplay of factors. However, various models and analyses, such as technical analysis and fundamental analysis, are employed by investors to forecast price movements based on historical data and financial metrics (Murphy, 1999; Graham & Dodd, 1934).

1.2 Techniques used to Predict Stock market prices

Predicting stock market prices is a complex task that involves various techniques, ranging from traditional financial analysis to advanced statistical models to advanced machine learning algorithms. This section explores some of the prominent methods used in forecasting stock prices, including Gaussian Process Regression and deep learning models like LSTM.

1.2.1 Technical Analysis

(Murphy, 1999) suggests that technical analysis involves studying past market data, primarily price and volume, to forecast future price movements. It is based on 3 fundamental premises.

1. *Market Action Discounts Everything:* market actions reflects the shifts in supply and demand. Which simply means that the rise and fall of stock prices have already taken into account all the external factors affecting it which is reflected in the stocks market price hence studying the price charts using technical indicators can help us predict the movement of prices.

2. *Prices Move in Trends:* The markets will always move in trends and by charting price action it is possible to identify those trends.

3. *History Repeats Itself:* Chart patterns which have been witnessed in the past can be identified and categorized to indicate bullish or bearish movement of the stock price, the patterns identified and the associated trends do not change over time.

(Murphy, 1999) states that based on the mentioned fundamental premise tools such technical moving averages, relative strength index (RSI), and chart patterns are used in technical analysis to predict price movements

1.2.2 Fundamental Analysis

Fundamental analysis is used to find a stock's true value by examining related economic, financial, and other quantitative factors. Analysts look at financial statements, management quality, industry conditions, and economic indicators to assess whether a stock is undervalued or overvalued. Fundamental analysis usually gives a better prediction of long term price movements compared to short term price movements which can be affected by unrelated qualitative factors (Wright Hoffman, 1935).

1.2.3 AI Models

Machine learning models have become increasingly popular for stock price prediction. These models can handle large datasets and identify complex patterns that traditional methods might miss. Methods such as Structural support vector machines (SSVMs) (Leung, MacKinnon and Wang, 2014) and deep learning models like ANN(artificial neural network), MLP (Multilayer Perceptron) and LSTM (Long-short term memory) have shown to give promising results while predicting stock market prices (Kumari, Sharma and Chauhan, 2021)

1.2.4 Gaussian Process Regression

A Gaussian process is a type of stochastic process, characterized by a collection of random variables indexed by time or space, such that any finite subset of these variables follows a multivariate normal distribution. The distribution of a Gaussian process represents the joint distribution of an infinite number of these random variables, resulting in a distribution over functions defined on a continuous domain, such as time or space (Wang, 2023).
Furthermore Gaussian Regression Process is a non-parametric Bayesian approach to regression (Banerjee, Dunson and Tokdar, 2012) and therefore it provides a flexible method for modeling and predicting time series data compared to parametric approaches (Ghahramani, 2013).

1.2.5 Time Series Analysis

Time series analysis involves statistical techniques that model and predict future values based on previously observed values. Common methods include Autoregressive Integrated Moving Average (ARIMA) models, which combine autoregression, differencing, and moving averages to capture various patterns in time series data. ARIMA is a simple autoregressive model and is not suitable for non linear relationship or capturing time-varying volatility in financial markets. In contrast, the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model addresses some of these limitations by explicitly modeling the volatility of the time series. The GARCH model assumes that volatility changes over time and often exhibits clustering, making it more suitable for financial data where periods of high volatility tend to be followed by periods of low volatility, and vice versa (Bollerslev, 1986).GARCH models provide a more robust approach for modeling and forecasting time-varying volatility in financial markets.

1.2.6 Sentiment Analysis

Sentiment analysis uses natural language processing (NLP) to analyze news articles, social media, and other text sources to gauge public sentiment about a stock. This technique can predict

stock price movements based on the collective sentiment expressed in these sources. This can be very speculative in nature in contrast to fundamental and technical analysis(Bollen, Mao and Zeng, 2011).
Sentiment analysis offers several advantages over traditional methods but they come with their own challenges in predicting the financial markets. One key benefit is its ability to capture real-time public sentiment, providing insights into how news and social media discussions might influence stock prices. This real-time analysis can help investors make timely decisions based on the latest public perceptions. Additionally, sentiment analysis can process vast amounts of unstructured text data, uncovering trends and patterns that might not be evident through traditional financial metrics alone. However, it also faces limitations, such as the challenge of accurately interpreting the context of language, which can lead to misinterpretations (Saad and Saberi, 2017).

1.3 Problem Definition

This dissertation aims to quantify the uncertainty in price movements of different securities in the stock markets and to predict the future price or value of these securities based on their past movements. Two distinct scenarios will be explored:

- Predicting future security prices during stable, regular market conditions.
- Predicting future security prices during periods of market instability, such as crashes.

To achieve these goals, Gaussian process regression will be employed as the primary predictive model. This method not only allows for the estimation of future security prices but also provides a quantifiable measure of uncertainty through the calculation of standard error ranges for the predictions. The dissertation will further streamline and optimize the Gaussian process regression approach to enhance accuracy and efficiency, ensuring the model delivers the most reliable forecasts possible.
Additionally, the research will explore the comparative performance of the model under different market conditions, shedding light on its robustness and adaptability in both stable and volatile environments.
The dissertation will follow the following flow:

1. Literature Review: A broad review of relevant literature will be conducted, focusing on key concepts and theories that helped the development of the code and the subsequent analysis. This section will evaluate prior research, highlighting the foundational ideas that shaped the methodological approach of this dissertation.
2. Background of Gaussian Process Regression: This section will provide an in-depth understanding of Gaussian Process Regression (GPR), detailing its theoretical foundations and mathematical representations. The discussion will cover the Gaussian prior and posterior distributions, accompanied by the corresponding mathematical formulations. Additionally a detailed examination of various kernel functions, including their mathematical expressions and implications for model performance will be discussed.
3. Data Description: This section will provide a detailed description of the dataset utilized in the study, including an overview of the data characteristics and structure. The data cleaning and transformation processes undertaken to prepare the dataset for analysis will be also be thoroughly discussed

4. Methodology: This section will provide in-depth insights into the methodologies employed for model development, encapsulating all aspects of model development, from kernel and prior selection to the training data timeframe taken for analysis. A thorough discussion of the hyperparameter optimization techniques used will also be included, detailing the rationale behind each choice and its impact on model performance.

5. Results: This section will present a detailed discussion of the results obtained from the study, with a focus on interpreting the findings in relation to the objectives outlined in the dissertation. Each result will be analyzed in depth, providing insights into how they contribute to the overall goals of the research and the implications for future work.

2 Literature Review

2.1 Wang (2023)

Wang (2023) provides a thorough and detailed introduction to the subject of Gaussian Process Regression (GPR), which is a powerful non-parametric Bayesian approach used in machine learning for regression tasks.
The primary objective of Wang's tutorial is to present GPR in a manner that is understandable to a wide range of readers. The paper sets out to:

1. Introduce Fundamental Concepts.
2. Explain Gaussian Process Regression.
3. Demonstrate Practical Implementation.

Wang (2023) further states that multivariate normal distributions are central to the development of GPR. Wang (2023) then introduces the concept of kernels, which play a pivotal role in GPR. Kernels are functions that define the similarity between data points, and their selection significantly impacts the performance of a GPR model. The paper discusses the most common kernel, the Radial Basis Function (RBF) kernel, and its properties and shows that unlike parametric models, which have a fixed number of parameters, non-parametric models like GPR have an infinite number of parameters, allowing for greater flexibility.
The paper goes on to describes a Gaussian process as a distribution over possible functions that fit a set of data points. This distribution is governed by the prior assumptions encoded in the kernel and is updated as new data becomes available. A key feature of GPR is its ability to provide not just point estimates but also measures of uncertainty. The paper explains how the mean of the posterior distribution gives the predicted function, while the variance provides a measure of confidence in the predictions.
Towards the end of the paper Wang (2023) provides standard algorithm for GPR, this section is complemented by a practical example, where the implementation is demonstrated through code snippets.

2.2 Tu et al. (2024)

The paper presents a hybrid model that integrates the Autoregressive Integrated Moving Average (ARIMA) model with Gaussian Process Regression (GPR) to better the accuracy of stock price prediction. Tu et al. (2024) address the limitations of conventional linear models like ARIMA, which fail to capture the non-linearity in stock price movements, and propose a combined model (ARIMA + GPR) that can account for both linear and nonlinear trends in prices. The study uses daily stock price data from three companies listed on the Shanghai Stock Exchange (SSE) to evaluate the performance of the proposed model against ARIMA, Artificial Neural Network (ANN), and GPR models.
The hybrid model consists of a two-step process: first, the ARIMA model is applied to capture the linear trends in the stock price data. The residuals from this model, which represent the non-linear components, are then modeled using GPR with a combined covariance function (GPRC). This allows the model to handle the complex linear and nonlinear relationships that belong to the movement of stock prices over a period of time. The model's performance is evaluated using mean absolute error (MAE), mean absolute percentage error (MAPE), mean square error

(MSE), and root-mean-square error (RMSE).
The study finds that the hybrid ARIMA + GPRC model outperforms both the individual ARIMA and GPR models, as well as the ANN model, in predicting stock prices. The combined model's ability to capture both linear and nonlinear characteristics of stock price movements results in significantly improved predictions.

2.3 Kim, Mallick and Holmes (2005)

In this paper Kim, Mallick and Holmes (2005) introduce a new method for analyzing spatial data that doesn't stay consistent across different areas by using piece wise Gaussian processes. This approach is more suitable for overcoming the limitations of traditional stationary models, which often struggle to detect sudden changes in spatial relationships. The proposed model breaks the spatial area into separate regions, each considered stable, allowing it to handle sharp transitions in the way data points are related to each other.
The methodology presented in the paper centers around a Bayesian framework that uses separate (disjointed) priors for different regions within the spatial domain. The spatial area is divided into distinct regions using a method called Voronoi tessellation (a method of dividing a plane into regions based on the distance to a specific set of points), and each of these regions is modeled independently with its own stationary Gaussian process. By modeling each region separately, the model assigns unique priors to the parameters that define the covariance structure within that region, meaning the priors are not influenced by data from other regions. This allows the model to capture the variability in the spatial data more accurately. The process of determining the number, shape, and position of these regions is carried out using a Markov chain Monte Carlo (MCMC) method.
The piecewise Gaussian process model was tested through simulations and applied to real-world soil permeability data from the Schneider Buda oil field in Texas. The results confirmed that the model is effective in handling non-stationary behavior, especially in areas where the spatial structure changes between different regions abruptly. Compared to traditional stationary models, this approach provided better predictive performance by accurately showing the varying spatial characteristics. The use of disjoint priors allowed each region's unique features to be independently modeled, leading to more precise and dependable predictions.
The use of disjointed priors allows for a highly flexible model that can accurately capture sharp changes in data, but it increases the time complexity of the simulation.

2.4 Yi (2022)

This article provides an in-depth introduction to Gaussian Processes (GPs), making complex concepts easier to grasp. It covers the key principles of GPs, particularly their use in machine learning for tasks like regression and classification. A significant focus is placed on the Bayesian nature of GPs, highlighting how they enable predictions with quantified uncertainty.
The article begins by defining key components of the Gaussian Process (GP) model, including the GP prior, likelihood, marginal likelihood, mean functions, kernel functions, and posterior. It provides comprehensive mathematical explanations on how to select priors, compute the posterior and perform parameter learning through maximum likelihood estimation. Real-world examples, such as stock price prediction and regression tasks for understanding functions, are used to illustrate these concepts. A key focus is on how GPs model uncertainty, which is a

fundamental part of the Bayesian method.
The article does not provide experimental results but instead offers a detailed conceptual analysis of Gaussian Processes (GPs). It explores important techniques, such as the use of kernels to determine the covariance between data points and how this along with the value of the hyperparameter influences the smoothness of the resulting function.
The important takeaway of this article is that it offers detailed mathematical insights and formulas used to construct Gaussian process regression models. By breaking down complex concepts and providing step-by-step explanations, the article simplifies the approach to building and understanding these models. This focus on mathematical clarity is a key takeaway, as it makes the complicated methods of Gaussian process regression easier to implement practically.

2.5 Krauth et al. (2017)

This paper explores the strengths and weaknesses of Gaussian Process (GP) models, focusing on three main challenges: scalability and efficient inference, flexible kernels, and alternative hyperparameter learning objectives. It presents AutoGP, a framework that improves GP models by integrating scalable variational inference, advanced kernel flexibility, and optimized objective functions. The findings demonstrate significant performance improvement across various datasets, this showcases the potential of GPs to compete deep learning methods.
The authors address the scalability challenge of Gaussian Process (GP) models by making use of the computation power of GPU's alongside stochastic variational inference. The authors also enhance kernel flexibility through the use of ARD (Automatic Relevance Determination) kernels and deep kernels, such as arc-cosine kernels. For hyperparameter optimization, they suggest using leave-one-out cross-validation (LOO-CV) rather than the traditional marginal likelihood, especially in classification tasks where minimizing error is crucial. The framework is implemented in TensorFlow, which supports automatic differentiation and more efficient gradient computation.
The study's key findings include:

1. Performance on MNIST Dataset: The proposed AutoGP framework outperformed all previously reported GP-based methods and was competitive with other kernel-based approaches.

2. Scalability: AutoGP demonstrated unprecedented scalability by successfully applying GP models to datasets containing up to 8 million observations.

3. Kernel Flexibility: The introduction of ARD and deep kernels (e.g., arc-cosine) allowed for more flexible and accurate modeling of complex functions, especially in high-dimensional spaces.

4. Hyperparameter Optimization: The LOO-CV objective function led to significant performance gains, particularly in error reduction for classification tasks.

In conclusion the authors identify areas for future exploration, such as the need for application-specific kernels and further optimization techniques to reduce computational complexity. Additionally, despite improvements in scalability, the framework's performance on image classification datasets like CIFAR-10 and RECTANGLES-IMAGE still lags behind deep learning approaches.

2.6 Shi et al. (2021)

The paper explores the use of a deep kernel Gaussian Process (GP) model for predicting financial market returns and volatility. The deep kernel GP model leverages the power of long short-term memory (LSTM) networks to enhance traditional GP kernels, capturing complex temporal dependencies in sequential financial data. The primary contribution is the development of a GP-LSTM model combined with a grid search algorithm, which optimizes hyperparameters for predicting both conditional returns and volatility. The study tests this model on the Shenzhen Stock Exchange Component Index and uses the predictions to construct a long-short trading portfolio which is then evaluated by its conditional Sharpe Ratio (financial metric used to evaluate the performance of an investment by adjusting for its risk).
The study introduces a new approach by integrating a Gaussian Process (GP) model with a Long Short-Term Memory (LSTM) network, which combines to form a deep kernel GP model (GP-LSTM). In this model, the LSTM network is employed to capture and encode temporal features from the data, which are then utilized by the GP's kernel function to make predictions. This combination uses the strengths of both models: the LSTM's ability to handle sequential data and the GP's ability to model uncertainty.
To optimize the model's performance, the study uses a grid search algorithm for hyperparameter tuning. The model is tested on daily return data from the SZSE COMP constituent stocks, covering the period from January 1, 2013, to July 12, 2017. The effectiveness of the model is assessed through various performance metrics, including Mean Squared Error (MSE), accuracy, and Value at Risk. These metrics help evaluate the model's predictive accuracy and its capability to estimate financial risk.
The results show that the GP-LSTM model outperforms traditional GARCH models and other machine learning models in predicting both returns and volatility. The constructed portfolio based on GP-LSTM forecasts achieved higher Sharpe Ratios compared to benchmarks, indicating superior prediction accuracy. The model performs especially well in volatile periods, highlighting its robustness in capturing complex financial patterns. Sub-period analysis reveals that the model's advantage is most significant in market conditions characterized by high volatility.

2.7 Banerjee, Dunson and Tokdar (2012)

The paper attempts to solve the computational challenges associated with applying Gaussian Process Regression (GPR) to large datasets. GPR models are highly flexible but become computationally difficult to process as the dataset size grows due to the cubic time complexity involved while performing matrix inversion. The authors in their research utilizes linear projection to reduce the dimension of the data while not compromising on the models predictive accuracy. This method allows GPR to be applied more efficiently to large datasets, overcoming the difficulties related to computational inefficiency.
The proposed method involves projecting the data into a lower-dimensional subspace using random projections, this method involves a linear projection that approximates the Gaussian process. The projection method significantly reduces the computational complexity by approximating the covariance matrix in a more stable manner. The study compares this method against traditional subset-based approaches, demonstrating its superior performance in both computational efficiency and predictive accuracy.
The study evaluates in-depth the proposed linear projection method by applying it to both sim-

ulated and real-world datasets. This approach allows for a robust assessment of the method's effectiveness when faced with different scenarios. The results indicate that the linear projection method not only significantly reduces computational time but also maintains or even enhances predictive performance when compared to conventional techniques. By applying the method to diverse datasets, the study provides evidence of its practical advantages, showcasing its efficiency and reliability in handling complex data analysis tasks. This demonstrates the method's potential to improve computational efficiency while delivering high-quality predictions, making it a valuable tool in various analytical applications.

2.8 Park, Lee and Son (2016)

The authors in this paper investigate the use of nonparametric machine learning models—namely neural networks, Bayesian neural networks, Gaussian Process Regression (GPR), and support vector regression (SVR)—for predicting market impact costs in the U.S. stock market. The market impact cost is a significant portion of implicit transaction costs that influence overall trading efficiency but are difficult to measure directly. The study compares these nonparametric methods against the I-star model, a well-established parametric benchmark, demonstrating the superior predictive performance of nonparametric approaches across various metrics. The research utilizes transaction data from large, mid, and small-cap stocks, offering a comprehensive analysis of market impact prediction across different capital sizes.
The study employs nonparametric regression models to estimate market impact costs, focusing on key input variables, including normalized order size (Size), 30-day volatility (Vol), and percentage of volume (POV). The data used spans 17 representative firms from the S&P 500, MidCap 400, and SmallCap 600 indices, comprising approximately 18 million transactions. The models are trained using cross-validation and evaluated on multiple performance metrics such as mean absolute error (MAE) and root mean square error (RMSE). The Gaussian Process model is particularly highlighted for its ability to incorporate uncertainty in predictions through its probabilistic framework.
The results show that nonparametric models, particularly the Bayesian neural network and Gaussian Process, outperform the traditional I-star model in predicting market impact costs. The improvements are consistent across large, mid, and small-cap stocks, with the Bayesian neural network reducing prediction errors by up to 43% compared to the benchmark model. However, the study notes that while support vector regression (SVR) provides some improvements, it sometimes underperforms compared to both the I-star model and other nonparametric approaches.

2.9 Alibrahim and Ludwig (2021)

The paper explores the performance of three popular hyper-parameter optimization techniques—Grid Search, Bayesian Optimization, and Genetic Algorithm—applied to a neural network. The focus is on determining which method yields the best hyper-parameters to enhance model accuracy.
The research used the Santander Customer Transaction Prediction dataset, involving 200,000 records with 200 features. The study applied Grid Search, Bayesian Optimization, and Genetic Algorithm to optimize hyperparameters, such as the number of hidden layers, optimizer, activation function, and dropout rate.
The Genetic Algorithm outperformed the other two methods in most metrics, achieving an accuracy of 0.9059, a mean squared error (MSE) of 0.0744. Grid Search showed slightly higher

accuracy (0.8976) compared to Bayesian Optimization (0.8959) and lower mean square error (0.0924) compared to Bayesian Optimization(0.1041), Grid Search performs slightly better compared to Bayesian Optimization as suggested by the results of this paper.
The study is limited to a single dataset and model type, which may limit the generalizability of the results also the computational cost of each method is not deeply analyzed, which is crucial for real-world applications.
This study also indicates that grid search method can be comparable or provide better results than Bayesian Optimization provided search space is relatively small, and exploration of hyperparameters is possible.

3 Background : Gaussian Process Regression

Gaussian Process Regression (GPR) is a powerful non-parametric method for regression that leverages the properties of the Gaussian distribution. It provides a flexible approach to modeling data, offering probabilistic predictions to quantify uncertainty (Yi, 2022).

3.1 Gaussian Process Regression

A Gaussian Process (GP) is a collection of random variables of any number which have a joint Gaussian distribution. For regression, we model the outputs y_i as:

$$y_i = f(x_i) + \epsilon_i$$

where $f(x)$ is the latent function we want to understand, and $\epsilon_i \sim \mathcal{N}(0, \sigma_n^2)$ represents the Gaussian noise with variance σ_n^2.

3.2 Gaussian Process Prior

Before observing any data, we assume that the function $f(x)$ is drawn from a GP:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x'))$$

where $m(x)$ is the mean function, often assumed to be zero, i.e., $m(x) = 0$, and $k(x, x')$ is the covariance function or kernel, which defines the covariance between function values at different points. A common choice for the kernel function is the Radial Basis Function (RBF) or squared exponential kernel:

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{1}{2l^2}\|x - x'\|^2\right)$$

where σ_f^2 is the variance and l is the length-scale hyperparameter.

3.3 Posterior Distribution

Given the training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^n$, the joint distribution of the observed target values $\mathbf{y} = [y_1, y_2, \dots, y_n]^T$ and the function value f_* at a new test point x_* is:

$$\begin{pmatrix} \mathbf{y} \\ f_* \end{pmatrix} \sim \mathcal{N}\left(0, \begin{pmatrix} \mathbf{K} + \sigma_n^2 \mathbf{I} & \mathbf{k}_* \\ \mathbf{k}_*^T & k(x_*, x_*) \end{pmatrix}\right)$$

Here:

- $\mathbf{K}$ is the $n \times n$ covariance matrix with $\mathbf{K}_{ij} = k(x_i, x_j)$.
- $\mathbf{k}_*$ is the covariance vector between the test point and training points, $\mathbf{k}_* = [k(x_*, x_1), \dots, k(x_*, x_n)]^T$.
- $k(x_*, x_*)$ is the variance at the test point x_*.

3.4 Making Predictions

To predict the function value at x_*, we condition on the observed data $\mathbf{y}$ to obtain the posterior distribution:

$$f_* \mid x_*, \mathcal{D} \sim \mathcal{N}(\mu_*, \sigma_*^2)$$

where the posterior mean and variance are given by:

$$\mu_* = \mathbf{k}_*^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y}$$

$$\sigma_*^2 = k(x_*, x_*) - \mathbf{k}_*^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{k}_*$$

The computation of the mean has $O(N^3)$ time complexity hence heavy computation will be required for large datasets.

3.5 Hyperparameter Optimization

The kernel function $k(x, x')$ depends on hyperparameters (e.g., σ_f, l). These can be optimized by maximizing the marginal likelihood:

$$\log p(\mathbf{y} \mid \mathbf{X}) = -\frac{1}{2}\mathbf{y}^T(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K} + \sigma_n^2\mathbf{I}| - \frac{n}{2}\log 2\pi$$

where $\mathbf{X}$ is the matrix of training inputs.

3.6 Kernel Functions

The choice of the kernel function $k(x, x')$ is crucial in Gaussian Process Regression, as it encodes assumptions about the smoothness, periodicity, and other properties of the function $f(x)$. Below are some commonly used kernels and their formulas (Natsume, 2022):

3.6.1 Squared Exponential (RBF) Kernel

The Squared Exponential (SE) kernel, also known as the Radial Basis Function (RBF) or Gaussian kernel, is one of the most commonly used kernels. It is defined as:

$$k(x, x') = \sigma_f^2 \exp\left(-\frac{1}{2l^2}\|x - x'\|^2\right)$$

where:

- σ_f^2 is the variance parameter, controlling the vertical variation of the function.
- l is the length-scale parameter, controlling the horizontal variation and smoothness of the function.

This kernel assumes that the function $f(x)$ is smooth and that nearby points have similar function values.

3.6.2 Matern Kernel

The Matérn kernel is a generalization of the RBF kernel that introduces a parameter ν controlling the smoothness of the function:

$$k(x, x') = \sigma_f^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\|x - x'\|}{l} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}\|x - x'\|}{l} \right)$$

where:

- ν controls the smoothness of the function. When $\nu = 1/2$, the Matérn kernel corresponds to the exponential kernel. As $\nu \to \infty$, the Matérn kernel converges to the RBF kernel.
- K_ν is the modified Bessel function of the second kind.

The Matérn kernel is often preferred when the function is expected to be less smooth than what the RBF kernel assumes.

3.6.3 Rational Quadratic Kernel

The Rational Quadratic (RQ) kernel can be seen as a scale mixture of RBF kernels with different length-scales. It is defined as:

$$k(x, x') = \sigma_f^2 \left(1 + \frac{\|x - x'\|^2}{2\alpha l^2} \right)^{-\alpha}$$

where:

- α is a positive parameter that controls the relative weighting of different length-scales.
- σ_f^2 and l have the same interpretation as in the RBF kernel.

This kernel is useful for modeling functions with varying degrees of smoothness.

3.6.4 Periodic Kernel

The Periodic kernel is used for modeling functions that are periodic in nature:

$$k(x, x') = \sigma_f^2 \exp \left(- \frac{2 \sin^2 \left(\frac{\pi \|x - x'\|}{p} \right)}{l^2} \right)$$

where:

- p is the period of the function.
- l controls the smoothness of the periodic variations.

This kernel is particularly useful for time-series data with clear periodic patterns.

3.6.5 Linear Kernel

The Linear kernel is suitable for modeling linear relationships between input and output. It is defined as:

$$k(x, x') = \sigma_0^2 + x \cdot x'$$

where:

- σ_0^2 is the variance parameter that controls the offset of the regression.
- $x \cdot x'$ denotes the dot product between the input vectors x and x'.

This kernel assumes that the function $f(x)$ is a linear function of the input.

3.6.6 Constant Kernel

The Constant kernel represents a simple, constant function. It is defined as:

$$k(x, x') = \sigma_c^2$$

where σ_c^2 is a constant value.
This kernel is often used in combination with other kernels to model the bias term or when the function is assumed to have a constant bias over the input space.

4 Data Description

This section details the columns of the datasets used in the analysis. The dissertation focuses on the daily closing prices of two stocks with significant market capitalizations: Apple Inc. (AAPL), listed on the National Association of Securities Dealers Automated Quotations (NASDAQ), and Hindustan Unilever (HUL), listed on the Bombay Stock Exchange (BSE). Additionally, the daily value of the Standard & Poor's 500 (S&P 500) is included in the analysis, as it tracks the stock performance of 500 of the largest companies listed on U.S. stock exchanges.
Stocks with large market capitalization are considered because they are less susceptible to manipulation by individual investors or large fund houses. Similarly, the S&P 500, which tracks 500 of the largest companies in the U.S. stock market, is also resistant to such influences. This reduces the likelihood of unpredictable price movements that lack a solid basis.
Python version 3.12 is used to prepare the graphs with the help of plyplot from the matplotlib library, data manipulation and cleaning is done using the numpy and pandas library
The datasets have been downloaded from different sources and are as follows:

4.1 Standard & Poor's 500 (S&P 500)

The data for S&P 500 daily values is downloaded from www.marketwatch.com, Table 1 shows the data type and structure of the dataset before cleaning.

Date	**Close**	**Open**	**High**	**Low**
08/08/2024	5252.57	5328.03	5233.85	5319.31
08/07/2024	5293.13	5330.64	5195.54	5199.50
08/06/2024	5206.42	5312.34	5193.56	5240.03
08/05/2024	5151.14	5250.89	5119.26	5186.33
08/02/2024	5376.63	5383.89	5302.03	5346.56

Table 1: Pre-Cleaned Data S&P 500

The dataset in Table 1 is cleaned to perform analysis:

1. Date: This column indicates the day in which the data was recorded, the Date column is changed from mm/dd/YYYY fromat to YYYY-mm-dd format and then the rows are arranged in ascending order by date.
2. Close: This column shows the closing value of S&P 500 on the given date, this column is converted from a string to float datatype.
3. Open: This column shows the opening value of S&P 500 on the given date, this column is converted from a string to float datatype.
4. High: This column shows the highest value reached by S&P 500 on the given date, this column is converted from a string to float datatype.
5. Low: This column shows the lowest value reached by S&P 500 on the given date, this column is converted from a string to float datatype.

The dataset contains everyday values of the S&P 500 index from 1st January 2011 to 8th August 2024, 3434 rows and 5 columns.
Figure 1 shows the chart of the closing values for the recorded dates.

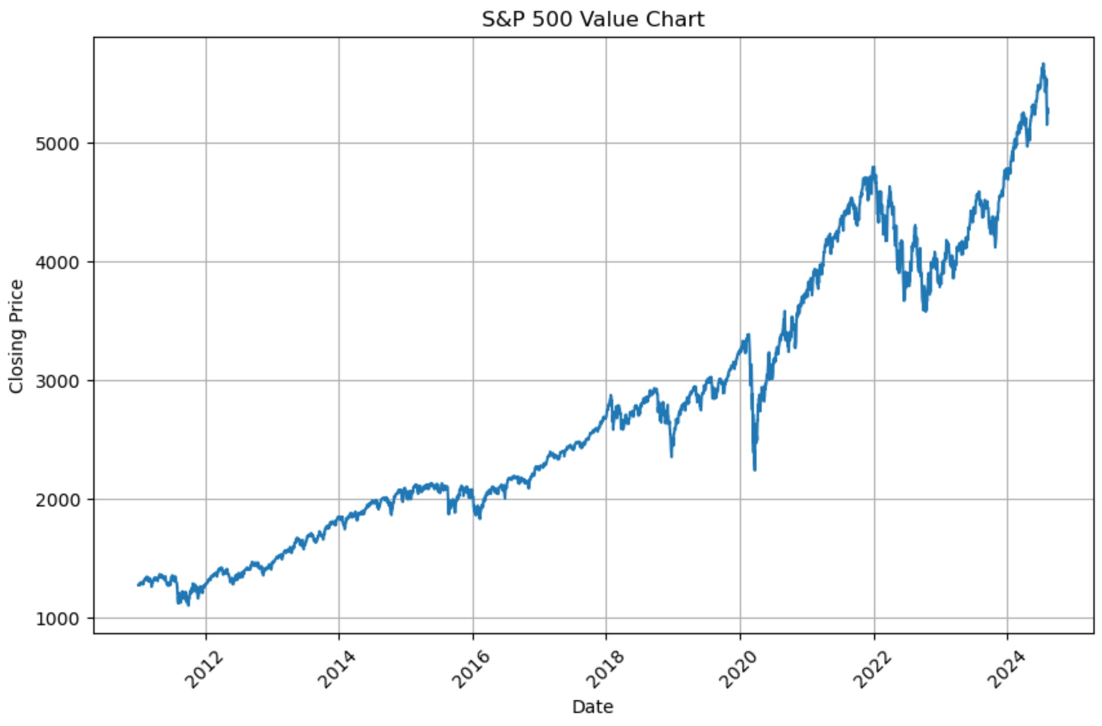


Figure 1: Value Chart for S&P 500 Index.

4.2 Hindustan Unilever (HUL)

The data for the daily stock prices of HUL has been downloaded from www.finance.yahoo.com, Table 2 shows the data type and structure of the dataset before cleaning The dataset in Table 2

Date	Open	High	Low	Close	Volume
2000-01-03	230.000000	230.755005	230.000000	230.755005	249030.0
2000-01-04	238.100006	238.100006	225.000000	228.104996	585140.0
2000-01-05	220.000000	225.000000	210.500000	219.990005	802960.0
2000-01-06	222.490005	236.000000	221.000000	228.820007	1900320.0
2000-01-07	230.000000	245.199997	228.225006	241.294998	1925820.0

Table 2: Pre-Cleaned Data HUL

is cleaned to perform analysis:

1. Date: This column indicates the day in which the data was recorded, the Date column is changed from string fromat to YYYY-mm-dd date format.
2. Close: This column shows the closing value of HUL share prices on the given date, this column is converted from a string to float datatype.
3. Open: This column shows the opening value of HUL share prices on the given date, this column is converted from a string to float datatype.
4. High: This column shows the highest value reached by HUL share prices on the given date, this column is converted from a string to float datatype.

5. Low: This column shows the lowest value reached by HUL share prices on the given date, this column is converted from a string to float datatype.

6. Volumne: This column shows the total volume of trades made, total shares bought and sold, on the given date.

The dataset after cleaning contains everyday share prices of HUL from 1st January 2000 to 8th August 2024, 6142 rows and 6 columns.
Figure 2 shows the price chart of the closing prices for the recorded dates.

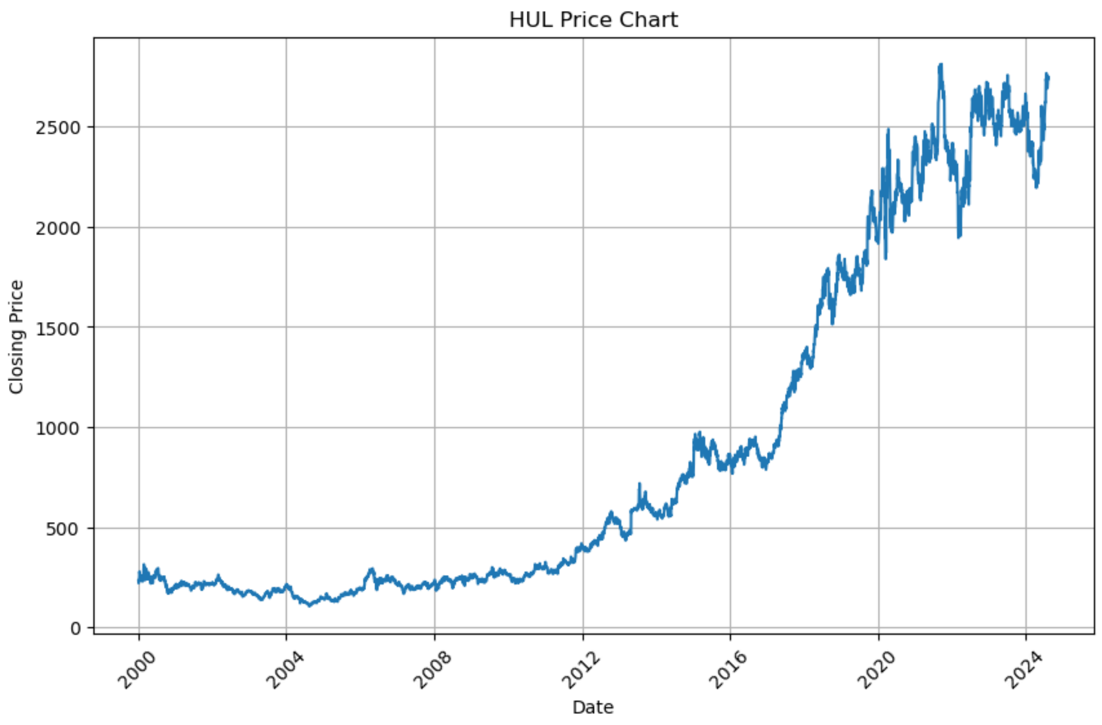


Figure 2: Price Chart for HUL.

4.3 APPLE Inc. (APPL)

The data for the daily stock prices of APPL has been downloaded from www.marketwatch.com, Table 3 shows the data type and structure of the dataset before cleaning The dataset in Table 3

Date	Close	Volume	Open	High	Low
08/09/2024	216.24	42201650	212.10	216.78	211.97
08/08/2024	213.31	47161150	213.11	214.20	208.83
08/07/2024	209.82	63516420	206.90	213.64	206.39
08/06/2024	207.23	69660490	205.30	209.99	201.07
08/05/2024	209.27	119548600	199.09	213.50	196.00

Table 3: Pre-Cleaned Data APPL

is cleaned to perform analysis:

1. Date: This column indicates the day in which the data was recorded, the Date column is changed from string mm-dd-YYYY to YYYY-mm-dd date format and then the rows are arranged in ascending order by date.

2. Close: This column shows the closing value of APPL share prices on the given date, this column is converted from a string to float datatype.

3. Open: This column shows the opening value of APPL share prices on the given date, this column is converted from a string to float datatype.

4. High: This column shows the highest value reached by APPL share prices on the given date, this column is converted from a string to float datatype.

5. Low: This column shows the lowest value reached by APPL share prices on the given date, this column is converted from a string to float datatype.

6. Volumne: This column shows the total volume of trades made, total shares bought and sold, on the given date.

The dataset after cleaning contains everyday share prices of HUL from 1st January 2011 to 9th August 2024, 3423 rows and 6 columns.
Figure 3 shows the price chart of the closing prices for the recorded dates.

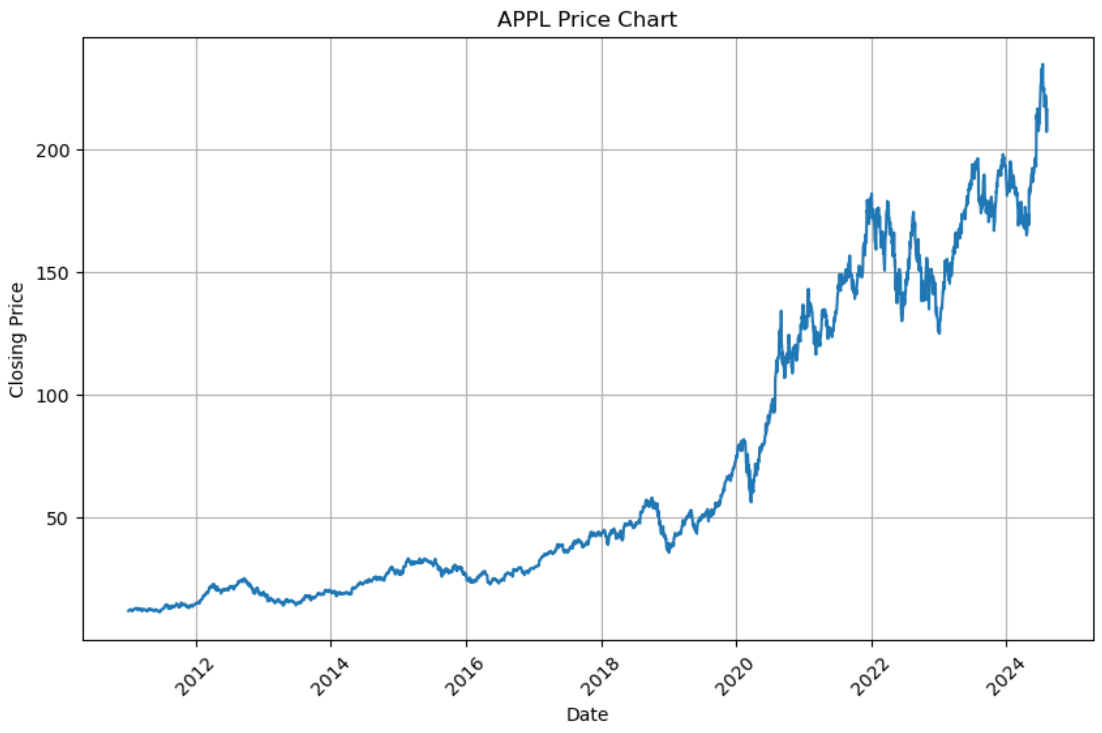


Figure 3: Price Chart for APPL.

5 Methodology

This section of the dissertation delves into the methodologies employed and the analyses conducted to forecast the future values of the following securities, Python version 3.12 is used to compile all the code, for the graphs pyplot from the matplotlib library is used, the numpy and pandas library is used for data manipulation and cleaning, the system used is Acer Predator laptop with a 9^{th} generation intel i7 processor and a NVIDIA GTX 1660Ti graphics card:

1. **Hindustan Unilever (HUL)**: A publicly traded company listed on both the Bombay Stock Exchange (BSE) and the National Stock Exchange (NSE) in India. HUL is a Fast-Moving Consumer Goods (FMCG) company with a substantial market capitalization of 77 billion USD (United States Dollars) as of 20th August 2024.
2. **Apple Inc. (AAPL)**: A publicly listed technology company on the NASDAQ in the USA. AAPL boasts a market capitalization of 3.4 trillion USD as of 20th August 2024.
3. **Standard & Poor's 500 (S&P 500)**: The S&P 500 is an index tracking the performance of 500 of the largest companies in the U.S. stock market, collectively representing a market capitalization of 46 trillion USD as of 20th August 2024.

5.1 Aim

The primary objective of constructing the predictive model is to forecast the price or value movement of a security one month into the future, based on its historical data. This prediction will be carried out under two different scenarios: during regular market conditions and during market crashes.

The following steps will be undertaken in this dissertation to develop optimal predictive models:

1. Identifying the most suitable kernel or combination of kernels to be used.
2. Determining the optimal prior mean function, by closely emulating the true underlying function with the prior to improve future predictions.
3. Selecting the appropriate time period of historical data to predict one month into the future.
4. Splitting the training data into training and validation sets to model the posterior function.
5. Utilizing grid search to find the optimal parameters for the validation set.
6. Applying the identified optimal parameters to test the model on the test set.

5.2 Selection of Kernel Function

There are various different kernels, some of which have been discussed in section 3.6, the kernels used to model smooth functions are RBF kernel, Matern Kernel and Rational Quadratic Kernel, periodic kernel isn't compared here as the values of the securities are not often periodic in nature (Li et al., 2022).

1. **RBF Kernel:** This kernel is widely used due to its smoothness and versatility in modeling non-linear relationships. It is particularly effective when the underlying function is smooth (Scikit-learn developers, 2024;Li et al., 2022).

2. **Matern Kernel:** The Matern kernel is more flexible than the RBF because it introduces a parameter (ν) that controls the smoothness of the function. It can model rougher or smoother functions depending on the choice of ν, making it a popular choice for functions with different degrees of smoothness. The inclusion of the additional parameter increases the computational time required for grid search (Scikit-learn developers, 2024 Li et al., 2022).

3. **Rational Quadratic Kernel:** This kernel can be seen as a scale mixture of RBF kernels with different length scales, making it capable of handling datasets with varying levels of smoothness. It is particularly useful when the data exhibits different patterns at different scales. The inclusion of additional parameter also increases the computational time in this case (Scikit-learn developers, 2024 Li et al., 2022).

The choice of kernel can significantly impact the performance of a GPR model. The RBF kernel is typically used for smooth data, while the Matern kernel can better handle rougher functions. The Rational Quadratic kernel's flexibility in handling multiple scales can make it a better fit when the data has multiple underlying processes occurring at different scales. These kernels can also be combined but combining the kernels could over-fit the training data and give huge errors while testing (Li et al., 2022).
The RBF Kernel combined with the linear kernel and constant kernel will be used for model preparation. The combination of a RBF kernel and a Matern kernel with nu = 0.5 is able to fit to both smooth and non smooth features in the dataset (Li et al., 2022) giving better results for datasets relating to securities but this increases the time complexity to find the optimal parameters using grid search from O(N^2) to O(N^4) which the current system cannot handle, hence the combination of RBF Kernel with Linear and Constant Kernel is used.

5.3 Choosing the Prior Mean Function

The ideal prior function should closely resemble the underlying function of the given data (Snoek, Larochelle and Adams, 2012). The prior function is represented as:

$$f(x) \sim \mathcal{N}(m(x), k(x, x'))$$

Where:

- $f(x)$ represents the function value at input x.
- $m(x)$is the prior mean function.
- $k(x, x') = \text{Cov}(f(x), f(x'))$ is the covariance function.

6 different prior mean functions (Hwang et al., 2023) were fit to each of the dataset to find out the best fit prior mean functions, here are the different functions examined (Note the prior mean function is denoted as $Pr\bar{i}or$ in all formulas below):

5.3.1 Mean function as the Prior Mean

This is the simplest prior mean function where the mean function is equal to the mean of all closing value of the security over the given time period

$$\text{Pr}\bar{\text{i}}\text{or} = \text{Mean}(Y_{\text{observed}})$$

$$\bar{\text{Prior}} = \frac{\sum Y_i}{\text{number of observations}}$$

Figure 4 shows the prior mean function graphs for the selected securities, the prior mean function is in blue while the actual value of the securities is in black.

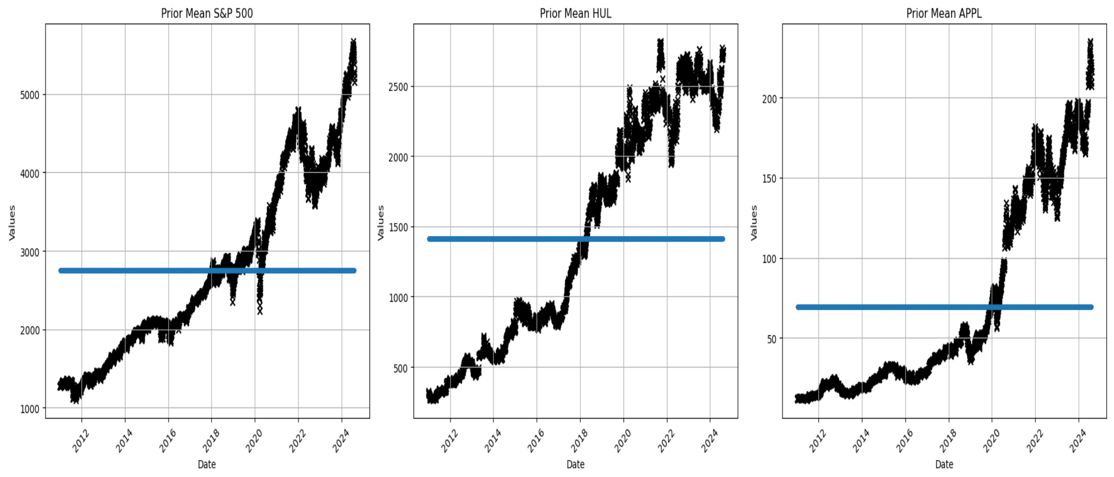


Figure 4: Mean as the Prior Mean Function.

5.3.2 Linear Regression as the Prior Mean

This prior mean function is taken as the best-fit linear regression line to the data, modeling the relationship between the independent variable X and the dependent variable Y. The linear regression equation is given by:

$$Y = \beta_0 + \beta_1 X + \epsilon$$

where β_0 represents the intercept, β_1 the slope of the line, and ϵ the error term.
Figure 5 shows the prior mean function graphs for the selected securities, the prior mean function is in blue while the actual value of the securities is in black.

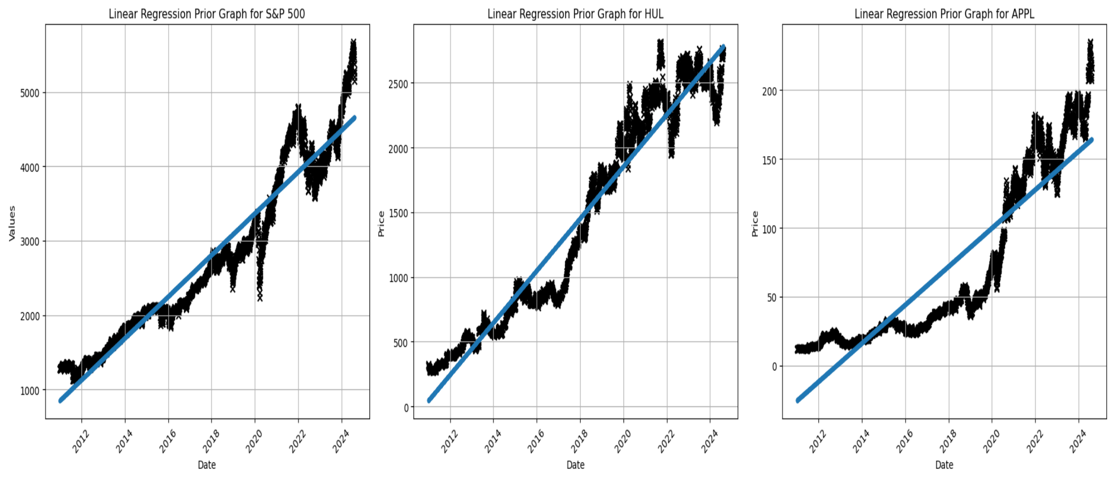


Figure 5: Linear Regression as the Prior Mean.

5.3.3 Exponential Function as the Prior Mean

The XIRR (Extended Internal Rate of Return) is calculated by the given formula:

$$\text{XIRR} = \left(\frac{C_n}{-C_0}\right)^{\frac{1}{d_n - d_0}} - 1$$

where,

- C_0 is the initial value.
- C_n is the final value.
- d_0 is the date of the initial observation.
- d_n is the date of the final observation.
- XIRR is the daily return rate that is being calculated.

The prior mean function in this case is given by:

$$y_i = C_0 \times (\text{XIRR})^{X_i - d_0}$$

where$X_i - d_0$ is the difference between the observed date and the initial investment date.Figure 6 shows the prior mean function graphs for the selected securities, the prior mean function is in blue while the actual value of the securities is in black.

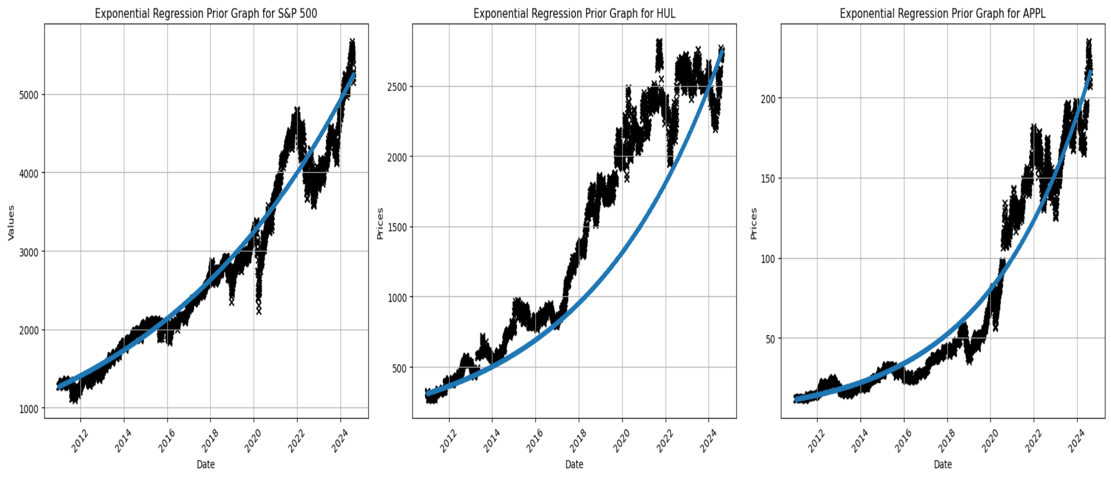


Figure 6: Exponential Function as the Prior Mean.

5.3.4 Disjointed Linear Functions as Prior Mean

This prior mean function partitions the total time period into monthly intervals and fits the best fit linear regression line based on best fit line of the the past 3 intervals, the function is disjointed as each interval has a unique linear regression line passing through it. The formula is given by:

$$Y_i = \beta_{0i} + \beta_{1i} X_i + \epsilon_i, \text{ where } i = 1, 2, \ldots, n.$$

where $\beta_0 i$ represents the intercept of the given time interval, $\beta_1 i$ the slope of the line on the given time interval, and ϵ_i is the error term.

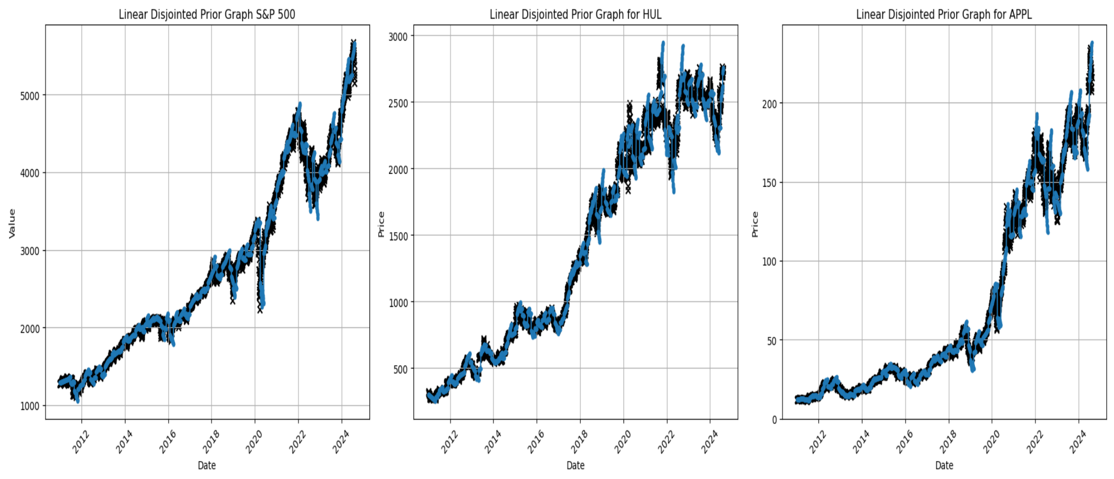


Figure 7: Disjointed Linear Functions as the Prior Mean.

Figure 7 shows the prior mean function graphs for the selected securities, the prior mean function is in blue while the actual value of the securities is in black.

5.3.5 Disjointed Quadratic Function as a Prior Mean

Similar to section 5.3.4, the prior mean function partitions the total time period into monthly intervals and fits the best fit 2nd Degree Polynomial line based on best fit 2nd Degree Polynomial line of the the past 3 intervals The formula is given by:

$$Y_i = \beta_{0i} + \beta_{1i} X_i + \beta_{2i} X_i^2 + \epsilon_i, \text{ where } i = 1, 2, \ldots, n.$$

where β_{0i} represents the intercept of the given time interval, β_{1i} represents the coefficient of the linear term, β_{2i} represents the coefficient of the quadratic term, and ϵ_i is the error term.
Figure 8 shows the prior function graphs for the selected securities, the prior function is in blue while the actual value of the securities is in black.

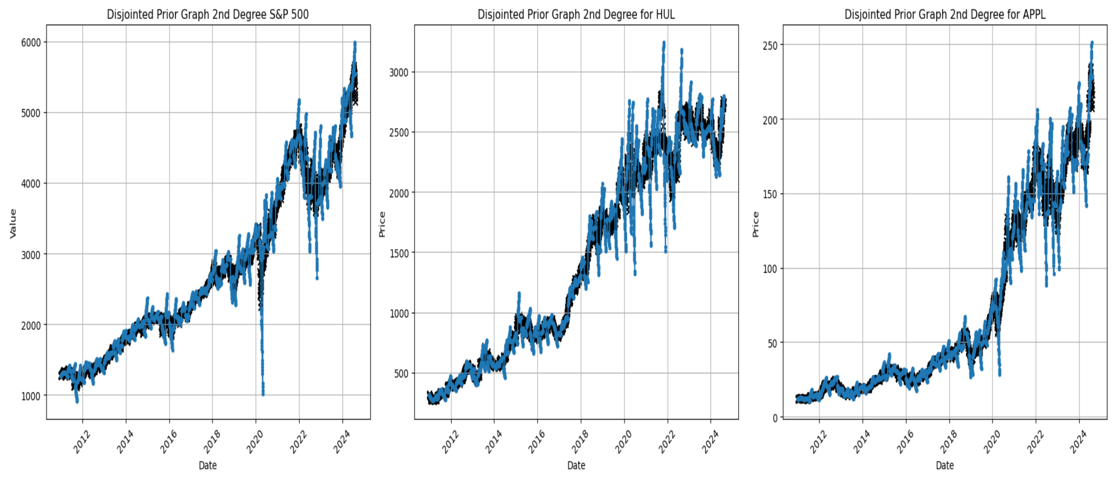


Figure 8: Disjointed Quadratic Functions as the Prior Mean.

5.3.6 Disjointed Cubic Function as a Prior Mean

Similar to section 5.3.5, the prior mean function partitions the total time period into monthly intervals and fits the best fit 3rd Degree Polynomial line based on best fit 3rd Degree Polynomial line of the the past 3 intervals The formula is given by:

$$Y_i = \beta_{0i} + \beta_{1i}X_i + \beta_{2i}X_i^2 + \beta_{3i}X_i^3 + \epsilon_i, \text{ where } i = 1, 2, \ldots, n.$$

where β_{0i} represents the intercept of the given time interval, β_{1i} represents the coefficient of the linear term, β_{2i} represents the coefficient of the quadratic term, β_{3i} represents the coefficient of the cubic term, and ϵ_i is the error term.
Figure 9 shows the prior mean function graphs for the selected securities, the prior mean function is in blue while the actual value of the securities is in black.

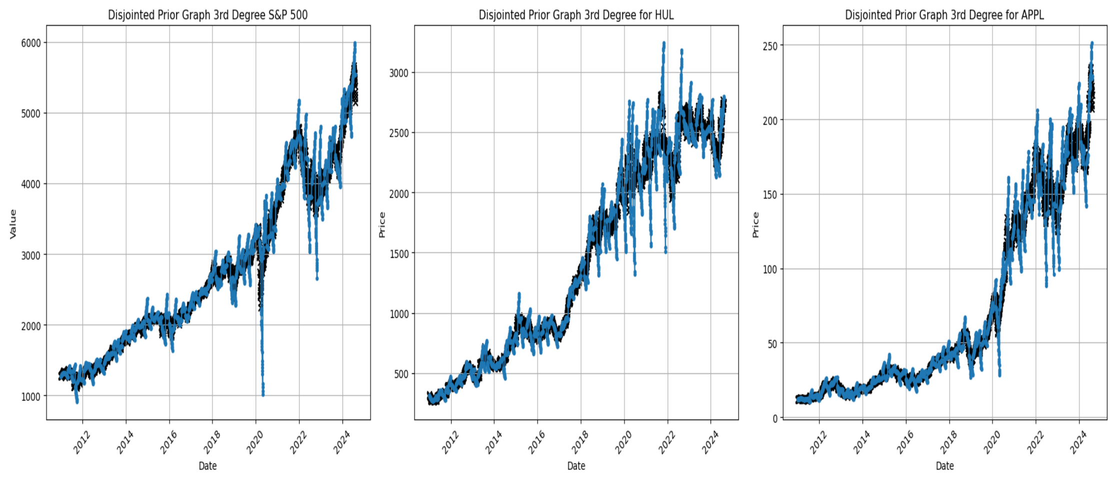


Figure 9: Disjointed Cubic Functions as the Prior Mean.

5.3.7 Summary

Figures 4 through 9 demonstrate that the disjointed linear function provides the best fit for the underlying data and effectively captures recent trends and according to (Hwang et al., 2023) the best fit mean function is beneficial to preparing the posterior function. This is further supported by Table 4, which presents the RMSE values for all prior functions, showing that the disjointed linear function consistently achieves the lowest RMSE across all cases. Additionally, it is evident that disjointed functions outperform other priors, aligning with the findings of (Kim, Mallick and Holmes, 2005). Their research suggests that disjointed priors enhance Gaussian process models, making them more effective at handling sharp transitions and variability. Despite the advantages there is a drawback to using the disjointed functions as in this case the prior function can only be calculated for 1 month into the future and no further.

Prior Function	**RMSE S&P 500**	**RMSE HUL**	**RMSE APPL**
Mean	1299755.85	662103.28	3668.86
Linear	95345.42	38203.74	670.91
Exponential	64169.70	168661.45	232.93
Disjointed Linear	20453.52	12060.41	63.17
Disjointed Quadratic	45846.39	24215.92	103.70
Disjointed Cubic	45848.57	24217.14	103.71

Table 4: RMSE for different prior functions across various securities

5.4 Choosing the Time Period

Table 5 shows the RMSE values for different time periods in predicting one month of data. The kernel used is a RBF kernel with addition of constant and linear kernels. The prior mean used is the disjointed linear function from section 5.3.4 and the parameters are optimized by finding the log likelihood.

5.4.1 Finding Optimal parameters for Combined Kernel Function

To find the optimal parameters for a Gaussian Process Regression (GPR) model using a combination of the RBF kernel, linear kernel, and constant kernel, the log likelihood method is employed. The process involves maximizing the log marginal likelihood of the observed data with respect to the kernel hyper-parameters.

Log Marginal Likelihood

Given a set of training data $\{X, Y\}$, where X represents the input data and Y the observed outputs, the Gaussian Process assumes that the observations Y are drawn from a multivariate normal distribution:

$$Y \sim \mathcal{N}(0, K(X, X) + \sigma_n^2 I)$$

where:

- $K(X, X)$ is the covariance matrix computed using the combined kernel (RBF + linear + constant) with hyper-parameters θ.

- σ_n^2 represents the noise variance (part of the constant kernel).
- I is the identity matrix.

The log marginal likelihood function $\log p(y \mid X, \theta)$ is given by:

$$\begin{aligned}\log p(Y \mid X, \theta) = &-\frac{1}{2}(Y - \bar{\text{Prior}}(X))^T (K(X,X) + \sigma_n^2 I)^{-1}(Y - \bar{\text{Prior}}(X)) \\ &- \frac{1}{2}\log\det(K(X,X) + \sigma_n^2 I) - \frac{n}{2}\log 2\pi \end{aligned} \tag{1}$$

where:

- $(Y^T - \bar{Prior}(X))(K(X,X) + \sigma_n^2 I)^{-1}(Y^T - \bar{Prior}(X))$ represents the data fit term.
- $\log\det(K(X,X) + \sigma_n^2 I)$ represents the complexity penalty term.
- n is the number of observations.

Kernels Involved

The combined kernel $K(X,X)$ is the sum of three kernels:

- RBF kernel:
$$k_{\text{RBF}}(x_i, x_j) = \sigma_f^2 \exp\left(-\frac{(x_i - x_j)^2}{2\ell^2}\right)$$
Hyperparameters: σ_f^2 (variance), ℓ (length scale).
- Linear kernel:
$$k_{\text{linear}}(x_i, x_j) = \sigma_b^2 (x_i \cdot x_j)$$
Hyperparameter: σ_b^2 (slope).
- Constant kernel:
$$k_{\text{constant}}(x_i, x_j) = c$$
Hyperparameter: c (constant).

So, the combined kernel is:

$$K(X,X) = \sigma_f^2 \exp\left(-\frac{(x_i - x_j)^2}{2\ell^2}\right) + \sigma_b^2(x_i \cdot x_j) + c + \sigma_n^2 \delta_{ij}$$

where δ_{ij} is adding noise variance to the diagonal.

Optimization of Hyperparameters

The goal is to find the optimal hyperparameters $\theta = \{\sigma_f^2, \ell, \sigma_b^2, c, \sigma_n^2\}$ by maximizing the log marginal likelihood:

$$\theta_{\text{opt}} = \arg\max_{\theta} \log p(y \mid X, \theta)$$

This optimization is performed using gradient-based methods.This is done in Python, with the help of minimize function from the scipy.optimize module. The method used for optimization is the L-BFGS-B method (Limited-memory Broyden-Fletcher-Goldfarb-Shanno with Box constraints).

Gradient Computation

The gradient of the log marginal likelihood with respect to a hyperparameter θ_k is given by:

$$\frac{\partial \log p(y \mid X, \theta)}{\partial \theta_k} = \frac{1}{2}(Y^T - \bar{Prior}(X))K^{-1}\frac{\partial K}{\partial \theta_k}K^{-1}(Y^T - \bar{Prior}(X)) - \frac{1}{2}\text{tr}\left(K^{-1}\frac{\partial K}{\partial \theta_k}\right)$$

where $K = K(X, X) + \sigma_n^2 I$.
The optimization process involves iteratively adjusting θ_k to increase $\log p(y \mid X, \theta)$, thereby finding the best fit to the observed data.

5.4.2 Summary

Table 5 shows the RMSE values for the securities across different time periods within the training data. The test set comprises the period of January 2024, while the past time intervals are utilized for training. The analysis indicates that the period spanning from 8 to 11 years yields the most favorable results, leading to the selection of a training period between 9 and 10 years for the future analysis.

No. Of Years	S&P 500	HUL	APPL
1 Year	2206215.62	7920.67	315.95
2 Years	9530.65	6067.68	376.19
3 Years	60139.31	7154.71	315.62
4 Years	8229.75	6751.95	300.11
5 Years	27278.33	7539.59	282.89
6 Years	39876.27	6918.98	274.68
7 Years	22232.27	6597.17	267.02
8 Years	10251.87	7070.98	270.89
9 Years	6170.44	10773.47	271.92
10 Years	3991.25	9791.22	290.71
11 Years	4554.62	6575.33	247.06
12 Years	6765.06	7097.18	287.41

Table 5: RMSE Values for S&P 500, HUL, and APPL over Different Time Periods

5.5 Finding the Optimal Parameters using Grid Search

The training set will be further divided into training and validation subsets. This division aims to train the Gaussian Process Regression model using the Grid Search algorithm. Grid Search involves systematically evaluating all possible parameter values to identify the optimal parameters that provide the best fit for the model, as the best fit model will also be able to identify the underlying trend correctly.

5.5.1 Splitting the Datasets

The model needs to be tested under two conditions: during normal market activities and during market crashes. The training and testing datasets for all securities will differ in both scenarios.

The dataset is divided as seen in Table 6 for normal market activities. (Note: The date is in YYYY/mm/dd format).

Security	**Training**	**Validation**	**Testing**
S&P 500	2015/01/01 to 2024/04/30	2024/05/01 to 2024/05/31	2024/06/01 to 2024/06/30
HUL	2015/01/01 to 2024/03/31	2024/04/01 to 2024/04/30	2024/05/01 to 2024/05/31
APPL	2015/01/01 to 2024/02/29	2024/03/01 to 2024/03/31	2024/04/01 to 2024/04/30

Table 6: Data split for each security during normal market activities

The dataset is divided as seen in Table 7 for market crashes. (Note: The date is in YYYY/mm/dd format).

Security	**Training**	**Validation**	**Testing**
S&P 500	2011/01/01 to 2020/01/31	2020/02/01 to 2020/02/29	2020/03/01 to 2020/03/31
HUL	2012/01/01 to 2021/08/31	2021/09/01 to 2021/09/30	2021/10/01 to 2021/10/31
APPL	2011/01/01 to 2020/01/31	2020/02/01 to 2020/02/29	2020/03/01 to 2020/03/31

Table 7: Data split for each security during market crashes

5.5.2 Finding the best Fit Posterior Prediction for Validation Set

To determine the best-fit posterior function, a grid search is employed to optimize the parameters of the RBF Kernel, specifically the length scale and variance. Simultaneously, the parameters of the constant and linear kernels are derived using the log-likelihood method.
The time complexity for the grid search is O(N^2), and the optimal curve is identified by minimizing the square root of the error.The resulting posterior function for the observed data of the securities can be found in Figures 10,11 and 12 the posterior function derived from the observed data is indicated by the black dashed line meanwhile the observed data is indicated by the red cross.
The posterior functions derived in Figures 10, 11, and 12 represent the posterior functions for normal market conditions. The posterior fit for both normal market conditions and market crashes, based on the observed data, results in well-fitted functions.

Posterior Functions Over Different Securities

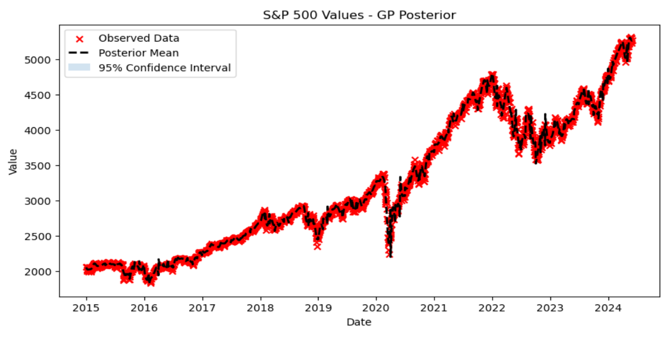


Figure 10: S&P 500 Posterior Function Fit On Observed Data.

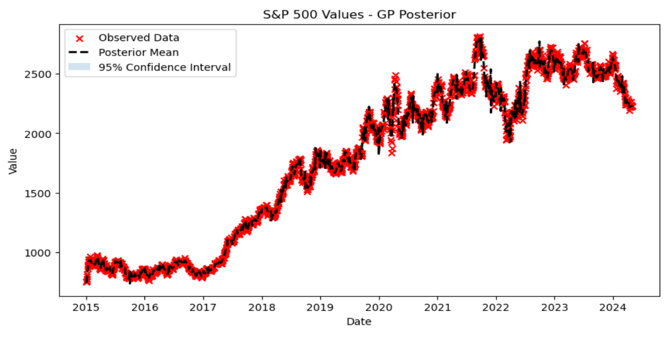


Figure 11: HUL Posterior Function Fit On Observed Data.

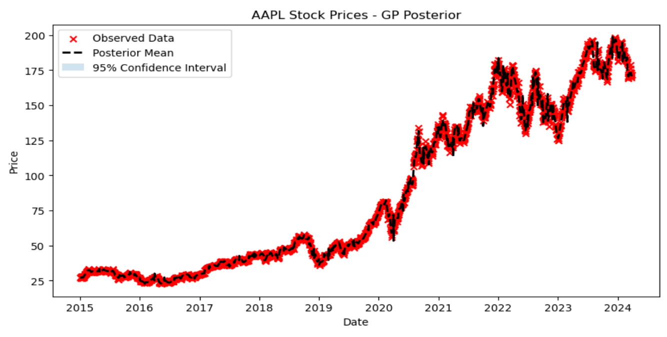


Figure 12: APPL Posterior Function Fit On Observed Data.

6 Results

This section of the dissertation focuses on the results generated by the Gaussian Process Regression (GPR) models for each security under varying market conditions. The outcomes will be observed and examined individually, here is the mathematical calculations used to make the model which has been detailed below (The code can be found in the appendix):

The covariance matrix for the observed data X_{obs} is given by:

$$K_{\text{obs}} = K(X_{\text{obs}}, X_{\text{obs}}) + \sigma_n^2 I$$

where $K(X_{\text{obs}}, X_{\text{obs}})$ is the covariance matrix computed using the combined kernel, and σ_n^2 represents the noise variance. The inverse of this matrix is:

$$K_{\text{obs}}^{-1} = \left(K(X_{\text{obs}}, X_{\text{obs}}) + \sigma_n^2 I\right)^{-1}$$

The posterior mean μ_{obs} for the observed data is computed as:

$$\mu_{\text{obs}} = \mu_{\text{prior}}(X_{\text{obs}}) + K_{\text{obs}} K_{\text{obs}}^{-1}(y_{\text{obs}} - \mu_{\text{prior}}(X_{\text{obs}}))$$

The posterior covariance Σ_{obs} for the observed data is computed as:

$$\Sigma_{\text{obs}} = K_{\text{obs}} - K_{\text{obs}} K_{\text{obs}}^{-1} K_{\text{obs}}^T$$

To sample from the posterior distribution for the observed data, we use the Cholesky decomposition:

$$L_{\text{post}} = \text{Cholesky}(\Sigma_{\text{obs}} + \sigma_n^2 I)$$

where L_{post} is the lower triangular matrix resulting from the Cholesky decomposition. Samples from the posterior distribution can then be generated as:

$$f_{\text{post}} \sim \mathcal{N}(\mu_{\text{obs}}, \Sigma_{\text{obs}})$$

which can be computed as:

$$f_{\text{post}} = \mu_{\text{obs}} + L_{\text{post}} \cdot \text{Normal}(0, I)$$

where $\text{Normal}(0, I)$ represents samples from a standard normal distribution. The covariance vector between the new (unobserved) data X_* and the observed data X_{obs} is given by:

$$K_* = K(X_*, X_{\text{obs}})$$

The covariance matrix of the unobserved data with itself is:

$$K_{**} = K(X_*, X_*)$$

The posterior mean μ_* for the unobserved data is computed as:

$$\mu_* = \mu_{\text{prior}}(X_*) + K_* K_{\text{obs}}^{-1}(y_{\text{obs}} - \mu_{\text{prior}}(X_{\text{obs}}))$$

The posterior covariance Σ_* for the unobserved data is computed as:

$$\Sigma_* = K_{**} - K_* K_{\text{obs}}^{-1} K_*^T$$

The models for each of the securities were evaluated under two different market conditions and all the models have run through the same training and testing loop to identify the optimal hyperparameters for each case. To achieve this, the training data was split into separate training and validation datasets, as shown in Tables 6 and 7. The models were initially trained on the training dataset and then tested using the validation dataset. Grid Search method was used to determine the best-fit curve for the validation set, focusing on minimizing the least square error to enhance the model's accuracy and find the optimal hyperparameters.

6.1 Applying the Model over HUL Dataset

Posterior Function and Prediction During Normal Market Conditions

Figure 13 shows the posterior mean in black and the observed data as red crosses, the posterior mean can be seen to fit the observed data well, the recent trend also suggests the stock price is reducing.

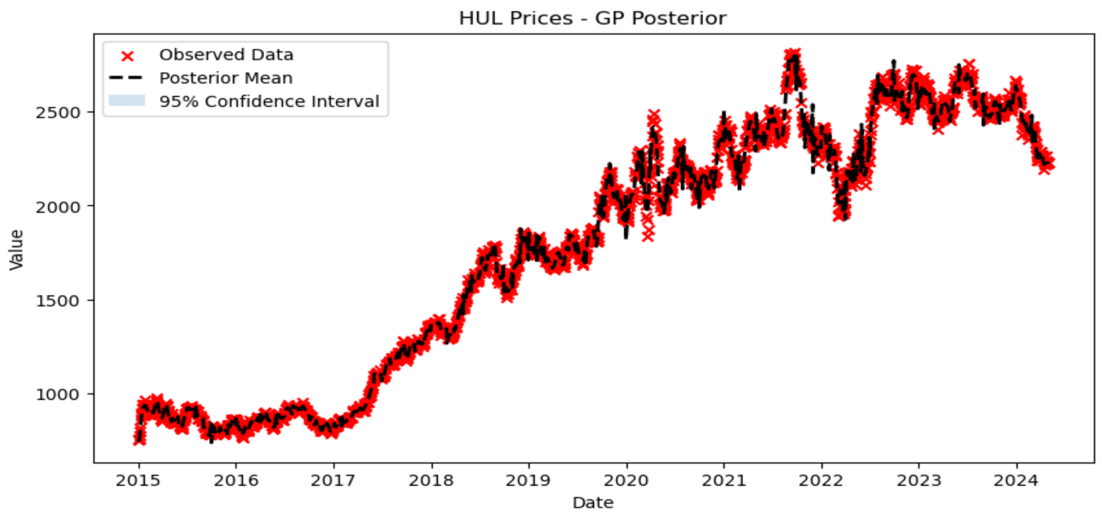


Figure 13: HUL Posterior Function for Training Data 1.

Figure 14 illustrates the posterior predictions for unobserved data, with the test data represented by black crosses and the prediction indicated by the red line. The grey shading along the red line shows the 95% confidence interval. The overall trend appears to be moving upwards; however, the posterior prediction does not capture this trend accurately.

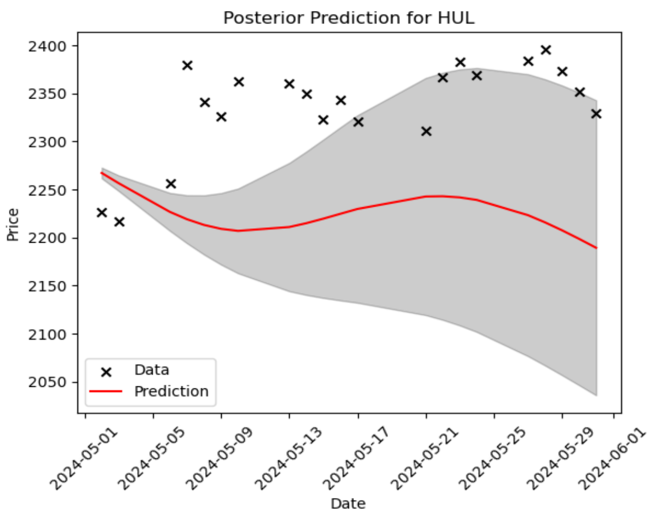


Figure 14: HUL Posterior Prediction for Normal Market Conditions.

Posterior Function and Prediction During Market Crashes

Figure 15 shows the posterior mean in black and the observed data as red crosses, the posterior mean can be seen to fit the observed data well, the recent trend also suggests the stock price has a increasing trend with a slight sharp decrease towards the end.

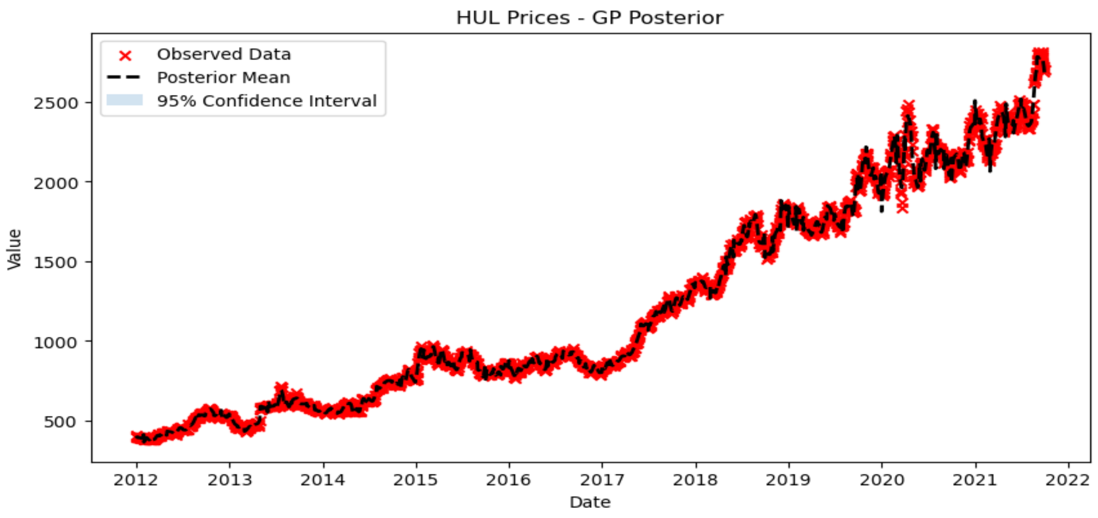


Figure 15: HUL Posterior Function for Training Data 2.

Figure 16 illustrates the posterior predictions for unobserved data, with the test data represented by black crosses and the prediction indicated by the red line. The grey shading along the red line shows the 95% confidence interval. The overall trend appears to be moving heavily downwards; however, the posterior prediction does not capture this trend accurately, it predicts a reverse trend.

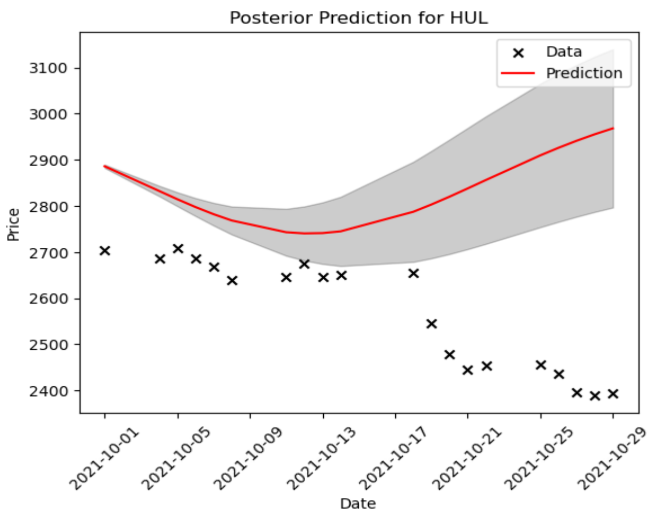


Figure 16: HUL Posterior Prediction for Crashes.

6.2 Applying the Model over S&P 500 Dataset

Posterior Function and Prediction During Normal Market Conditions

Figure 17 shows the posterior mean in black and the observed data as red crosses, the posterior mean can be seen to fit the observed data well, the recent trend also suggests the stock price is increasing.

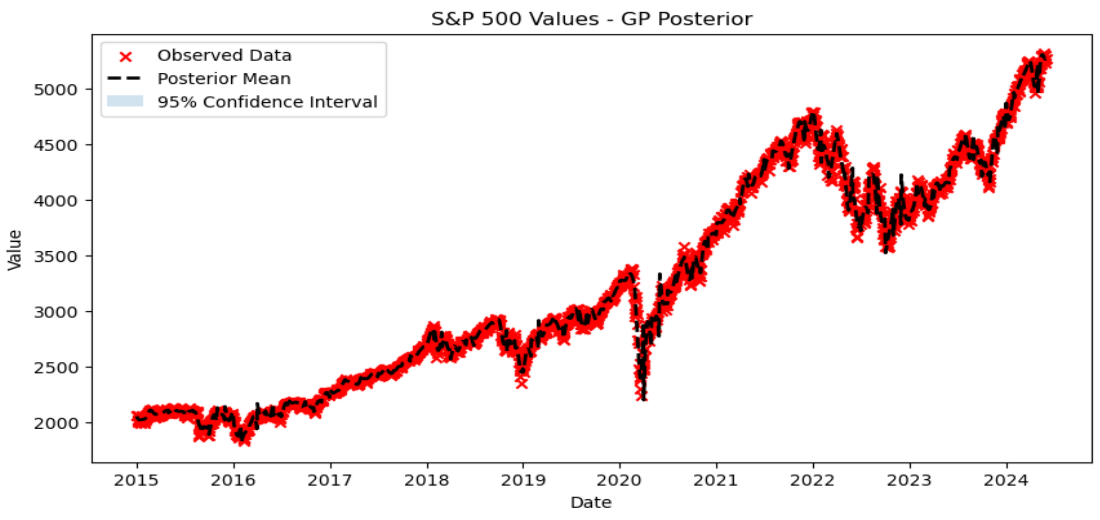


Figure 17: S&P 500 Posterior Function for Training Data 1.

Figure 18 illustrates the posterior predictions for unobserved data, with the test data represented by black crosses and the prediction indicated by the red line. The grey shading along the red line shows the 95% confidence interval. The overall trend appears to be moving upwards and the posterior prediction captures this trend accurately in this case giving a good fit to the unobserved data.

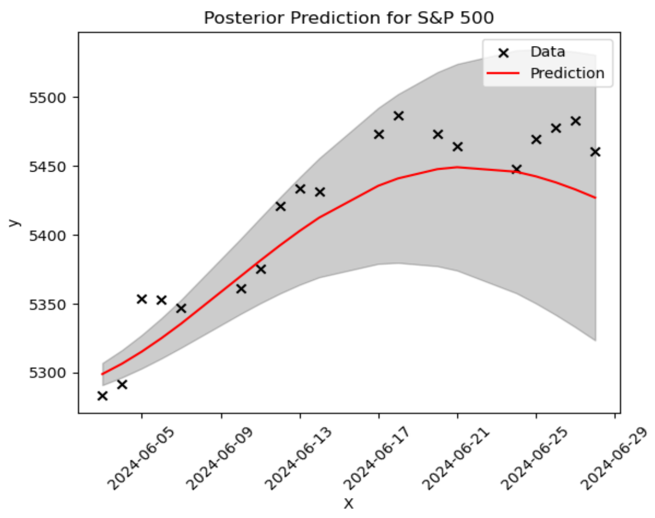


Figure 18: S&P 500 Posterior Prediction for Normal Market Conditions.

Posterior Function and Prediction During Market Crashes

Figure 19 shows the posterior mean in black and the observed data as red crosses, the posterior mean can be seen to fit the observed data well, the recent trend also suggests the stock price has a increasing trend with a very sharp decrease towards the end.

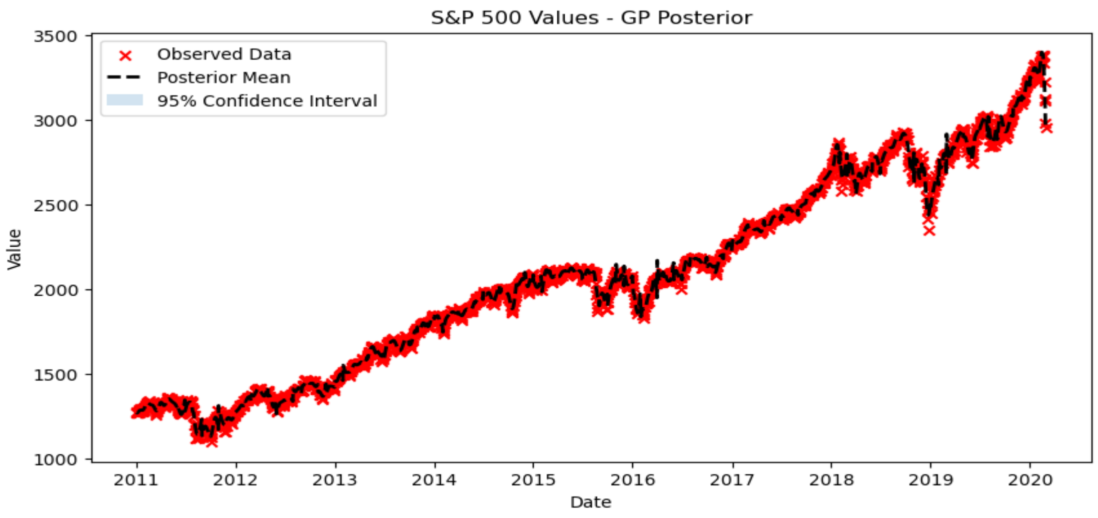


Figure 19: S&P 500 Posterior Function for Training Data 2.

Figure 20 illustrates the posterior predictions for unobserved data, with the test data represented by black crosses and the prediction indicated by the red line. The grey shading along the red line shows the 95% confidence interval which in this case cannot be seen clearly due to very small standard deviation. The overall trend appears to be moving heavily downwards; however, The posterior prediction exhibits a more heavy downward trend compared to the actual movement of the S&P 500 index. This result is likely due to the model overemphasizing the sharp decline in the observed data, leading to an exaggerated capture of the downward trend.

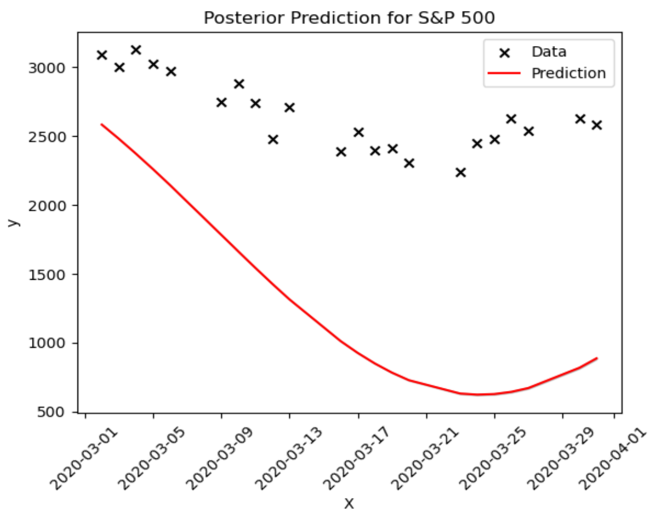


Figure 20: S&P 500 Posterior Prediction for Crashes.

6.3 Applying the Model over APPL Datset

Posterior Function and Prediction During Normal Market Conditions

Figure 21 shows the posterior mean in black and the observed data as red crosses, the posterior mean can be seen to fit the observed data well, the recent trend also suggests the stock price is moving in a cyclical trend since early 2023 where it is increasing and then decreasing within a range.

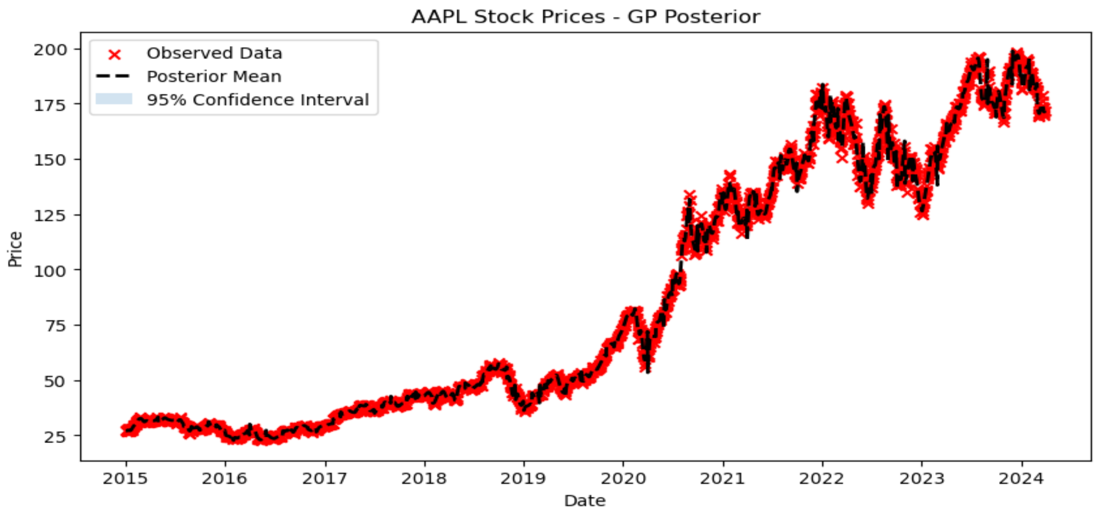


Figure 21: APPL Posterior Function for Training Data 1.

Figure 22 illustrates the posterior predictions for unobserved data, with the test data represented by black crosses and the prediction indicated by the red line. The grey shading along the red line shows the 95% confidence interval. The overall trend shows relatively stable price movements with minimal oscillation. However, the prediction seems to replicate the earlier trend of the price chart, leading to a less accurate fit for the unobserved data. This indicates that the model has failed to adapt to the current trend and is instead projecting past patterns onto the new data, the model does give decent predictions for the start and the end or the price movement in this case

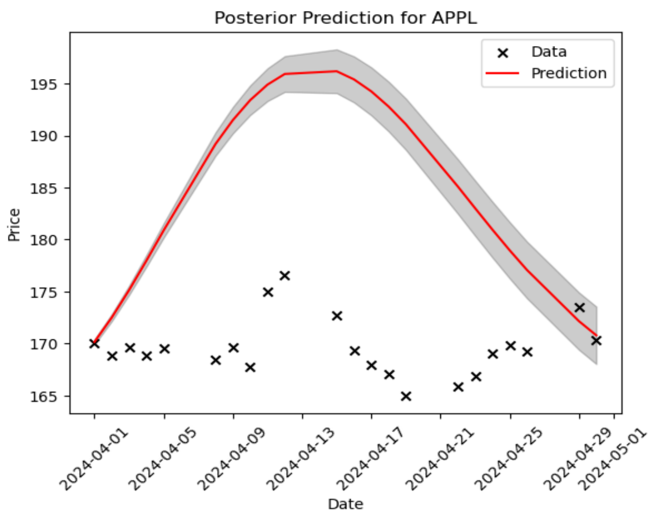


Figure 22: APPL Posterior Prediction for Normal Market Conditions.

Posterior Function and Prediction During Market Crashes

Figure 23 shows the posterior mean in black and the observed data as red crosses, the posterior mean can be seen to fit the observed data well, the recent trend also suggests the stock price has a very sharp decrease towards the end.

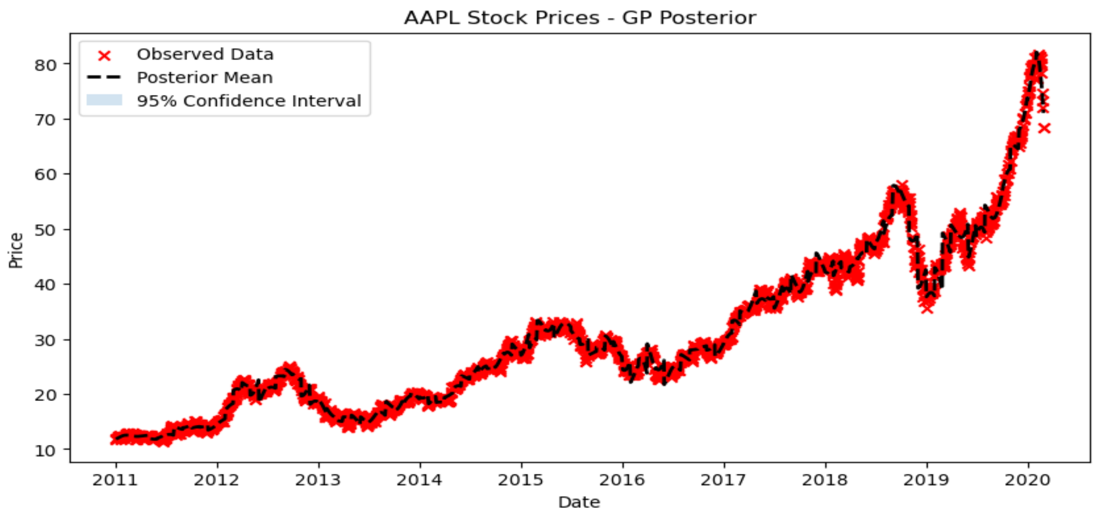


Figure 23: APPL Posterior Function for Training Data 2.

Figure 24 illustrates the posterior predictions for unobserved data, with the test data represented by black crosses and the prediction indicated by the red line. The grey shading along the red line shows the 95% confidence interval which in this case cannot be seen clearly due to very small standard deviation. The overall trend appears moving downwards; however, The posterior prediction exhibits a more bigger spiral downward compared to the actual price movement of the APPL share price. This result is likely due to the model overemphasizing the sharp decline in the observed data towards the end of the observed data which lead to an exaggerated capture of the downward trend.

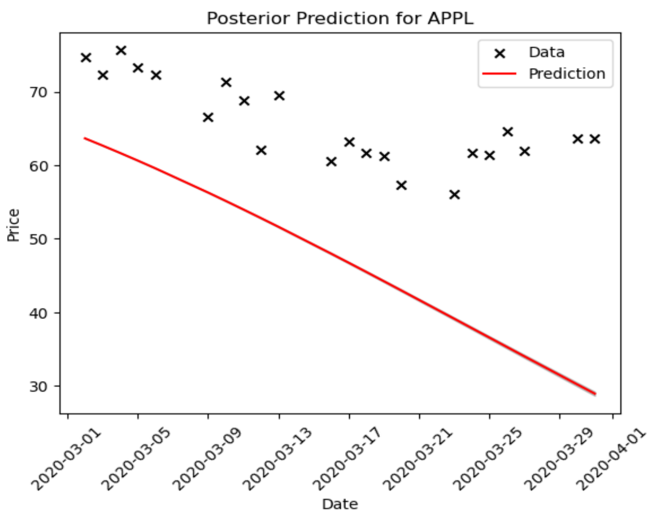


Figure 24: APPL Posterior Prediction for Crashes.

6.4 Summary of results

Gaussian Process Regression (GPR) may struggle to predict sudden, rapid changes in data as seen in the analysis of this dissertation, this occurs due to the fundamental assumptions and characteristics of Gaussian Processes. GPR assumes smoothness and continuity in the underlying function, which makes it less suitable for capturing abrupt shifts or trends. Consequently, GPR is often used in combination with other models or processes to improve its predictive performance in such scenarios. For example, GPR has been combined with ARIMA models to handle time series with rapid changes (Tu et al., 2024), or enhanced using Long Short-Term Memory (LSTM) networks to refine traditional GP kernels for better trend prediction (Shi et al., 2021).
The results obtained in this dissertation align with these observations. While simple kernels are computationally efficient and easier to model, they fall short in predicting new trends or rapid changes, reinforcing the need for more sophisticated approaches when dealing with such data patterns. The models perform better while the prices follow the recent trends as in the case of S&P 500 during regular market conditions, this is due to the fact that grid search method is employed to find the hyperparameters which represent the recent trends.
Table 8 shows the percentage error for each security in different market conditions, for normal market conditions the percentage error is least in the case of S&P 500 at 0.46% followed by HUL at 5.12% and APPL at 8.88%. For market crashes the percentage error is the least for HUL at 10.61% followed by APPL at 27.91% and S&P 500 at 52.41%.

Security Name	Market Condition	Percentage Error
S&P 500	Regular	0.46%
S&P 500	Crash	52.41%
HUL	Regular	5.12%
HUL	Crash	10.64%
APPL	Regular	8.88%
APPL	Crash	27.91%

Table 8: Percentage Error of each Security by Market Condition

In normal market conditions, the percentage error remains below 9%, as outlined in sections 6.1, 6.2, and 6.3. The model's predictions perform better when markets adhere to recent trends, such as in the case of the S&P 500 during stable periods. However, during market crashes, the percentage error ranges from 10% to 53%. This increased error is due to the model's tendency to exaggerate the impact of sharp declines during market crashes, leading to far worse predictions of crashes.
Table 9 shows the values of hyper-parameters that were found and used while preparing the model for each case. The Combined Kernel used is a combination of RBF, Constant and Linear Kernel as discussed in section 5.2, The RBF kernel has two hyperparamters, the length scale and the variance meanwhile the constnt kernel and linear kernel have 1 hyperparameter each. The mathematical formulas are available in section 3.6

Security Name	Market Condition	Length Scale	Variance	Constant Term	Linear Term
S&P 500	Regular	0.02	0.5	0.999	1.041
S&P 500	Crash	0.02	21	1741.47	898.71
HUL	Regular	0.01375	2.5	1.051	0.999
HUL	Crash	0.014	3.5	0.999	0.999
APPL	Regular	0.01225	2	0.999	0.999
APPL	Crash	0.07	1.5	1.003	0.999

Table 9: Optimal Hyperparameters for the Combined Kernel Found

7 Future Work

The computational complexity of Gaussian Process Regression (GPR), particularly its O(N^3) scaling, limits the feasibility of using more advanced kernel combinations which are much more suitable for data related to the stock market, such as RBF kernel combined with Matern kernel with nu = 0.5 (Li et al., 2022), without significantly increasing computational costs. In this dissertation, computations were performed on a single laptop (Acer Predator with a 9th-generation Intel i7 processor and NVIDIA 1660Ti GPU). To address these limitations, future work could leverage distributed computing systems, which would reduce computation time by running calculations in parallel across multiple systems, enhancing efficiency as suggested by (Chen et al., 2023).

Moreover, as discussed in section 2, the exploration of hybrid models that integrate GPR with other approaches, such as ARIMA, GARCH, or LSTM networks, offers a promising idea for improving predictive performance. For instance, combining GPR with a GARCH model could better capture both linear and nonlinear trends in stock prices, as highlighted by (Shi et al., 2021) and (Tu et al., 2024). This approach would allow for more appropriate kernel selection and model complexity, facilitated by the computational power of distributed systems.

By implementing these strategies, future research could overcome current computational constraints and further refine the predictive capabilities of GPR in financial market analysis.

8 Conclusion

Using the available tools and resources, for this dissertation it was possible to develop prediction models using Gaussian Process Regression (GPR) that performed well during regular market periods by capturing the previous trends, giving reasonable accuracy when the markets followed similar trends. However, the model struggled to capture uncertainty during rapid price movements, such as market crashes. Due to computational limitations, advanced kernel combinations with larger number of hyperparameters were not explored, leading to the use of simpler models. Future research could benefit from integrating GPR with more sophisticated techniques and exploring advanced kernels to potentially improve predictive accuracy during volatile market conditions and during times where expected trends are not followed due to external circumstances.

References

Alibrahim, H. and Ludwig, S.A. (2021). Hyperparameter Optimization: Comparing Genetic Algorithm against Grid Search and Bayesian Optimization. 2021 IEEE Congress on Evolutionary Computation (CEC). doi:https://doi.org/10.1109/cec45853.2021.9504761.

Banerjee, A., Dunson, D.B. and Tokdar, S.T. (2012). Efficient Gaussian process regression for large datasets. Biometrika, 100(1), pp.75–89. doi:https://doi.org/10.1093/biomet/ass068.

Bollen, J., Mao, H. and Zeng, X. (2011). Twitter mood predicts the stock market. Journal of Computational Science, 2(1), pp.1–8. doi:https://doi.org/10.1016/j.jocs.2010.12.007.

Bollerslev, T. (1986). Generalized autoregressive conditional heteroskedasticity. Journal of Econometrics, 31(3), pp.307–327.

Chen, W.-N., Wei, F.-F., Zhao, T.-F., Tan, K.C. and Zhang, J. (2023). A Survey on Distributed Evolutionary Computation. arXiv (Cornell University), 1(1). doi:https://doi.org/10.48550/arxiv.2304.05811.

Ghahramani, Z. (2013). Bayesian non-parametrics and the probabilistic approach to modelling. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 371(1984), p.20110553. doi:https://doi.org/10.1098/rsta.2011.0553.

Graham, B. and Dodd, D.L. (2009). Security analysis : principles and technique. New York: Mcgraw-Hill.

Hwang, S., L'Huillier, B., Keeley, R.E., M. James Jee and Arman Shafieloo (2023). How to use GP: effects of the mean function and hyperparameter selection on Gaussian process regression. Journal of cosmology and astroparticle physics, 2023(02), pp.014–014. doi:https://doi.org/10.1088/1475-7516/2023/02/014.

Khan, A. (n.d.). Predicting Short Term Stock Trends using Gaussian Regression. [online] Available at: https://web.lums.edu.pk/ adnan.khan/quantfin.pdf [Accessed 20 Jun. 2024].

Kim, H.-M., Mallick, B.K. and Holmes, C. (2005). Analyzing Nonstationary Spatial Data Using Piecewise Gaussian Processes. Journal of the American Statistical Association, 100(470), pp.653–668. doi:https://doi.org/10.1198/016214504000002014.

Krauth, K., Bonilla, E.V., Cutajar, K. and Filippone, M. (2017). AutoGP: Exploring the Capabilities and Limitations of Gaussian Process Models. arXiv (Cornell University). doi:https://doi.org/10.48550/arxiv.161

Kumari, J., Sharma, V. and Chauhan, S. (2021). Prediction of Stock Price using Machine Learning Techniques: A Survey. 2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N). doi:https://doi.org/10.1109/icac3n53548.2021.9725685.

Leung, C.K.-S., MacKinnon, R.K. and Wang, Y. (2014). A machine learning approach for stock price prediction. Proceedings of the 18th International Database Engineering & Applications Symposium on - IDEAS '14. [online] doi:https://doi.org/10.1145/2628194.2628211.

Li, T., Davies, G.R., Lyttle, A.J., Ball, W.H., Carboneau, L.M. and García, R.A. (2022). Modelling stars with Gaussian Process Regression: augmenting stellar model grid. Monthly Notices of the Royal Astronomical Society, 511(4), pp.5597–5610. doi:https://doi.org/10.1093/mnras/stac467.

Murphy, J.J. (1999). Technical analysis of the financial markets: a comprehensive guide to trading methods and applications. Choice Reviews Online, 36(07), pp.36–401636–4016. doi:https://doi.org/10.5860/choic
4016.

Natsume, Y. (2022). Gaussian Process Kernels. [online] Medium. Available at: https://towardsdatascience.com/gaussi
process-kernels-96bafb4dd63e [Accessed 25 Aug. 2024].

Park, S., Lee, J. and Son, Y. (2016). Predicting Market Impact Costs Using Nonparametric Machine Learning Models. PLOS ONE, 11(2), p.e0150243. doi:https://doi.org/10.1371/journal.pone.0150243.

Saad, S. and Saberi, B. (2017). Sentiment Analysis or Opinion Mining: A Review. International Journal on Advanced Science, Engineering and Information Technology, 7(5), p.1660. doi:https://doi.org/10.18517/ijaseit.7.5.2137.

Scikit-learn developers (2024). sklearn.gaussian_process. [online] scikit-learn. Available at: https://scikit-learn.org/stable/api/sklearn.gaussian_process.html [Accessed 19 Aug. 2024].

Shi, Y., Dai, W., Long, W. and Li, B. (2021). Deep Kernel Gaussian Process Based Financial Market Predictions. arXiv (Cornell University). doi:https://doi.org/10.48550/arxiv.2105.12293.

Snoek, J., Larochelle, H. and Adams, R.P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. arXiv (Cornell University), 2. doi:https://doi.org/10.48550/arxiv.1206.2944.

Tu, S., Huang, J., Mu, H., Lu, J. and Li, Y. (2024). Combining Autoregressive Integrated Moving Average Model and Gaussian Process Regression to Improve Stock Price Forecast. Mathematics, 12(8), pp.1187–1187. doi:https://doi.org/10.3390/math12081187.

Wang, J. (2023). An Intuitive Tutorial to Gaussian Processes Regression. Computing in Science and Engineering, 25(4), pp.1–8. doi:https://doi.org/10.1109/mcse.2023.3342149.

Wright Hoffman, G. (1935). GRAHAM, B., and D. L. DODD. Security Analysis. Pp. xi, 725. New York: McGraw-Hill Book Company, 1934. $4.00. The ANNALS of the American Academy of Political and Social Science, 177(1), pp.276–277. doi:https://doi.org/10.1177/000271623517700152.

Yi, W. (2022). Understanding Gaussian Process, the Socratic Way. [online] Medium. Available at: https://towardsdatascience.com/understanding-gaussian-process-the-socratic-way-ba02369d804.

Appendix

APPL Prediction

August 31, 2024

```
[103]: #Importing all the required libraries
       import os
       import random
       import numpy as np
       import matplotlib.pyplot as plt
       from sklearn.gaussian_process import GaussianProcessRegressor
       from sklearn.gaussian_process.kernels import RBF, ConstantKernel, Matern,␣
        ↪WhiteKernel
       from sklearn.model_selection import GridSearchCV
       from sklearn.metrics import mean_squared_error
       import pandas as pd
       from scipy.linalg import cho_solve, cho_factor
       import gpytorch
       from datetime import datetime
       import torch
       from scipy.linalg import cholesky,solve_triangular
       from scipy.optimize import minimize
       from sklearn.linear_model import LinearRegression
       import sklearn
       from sklearn.model_selection import KFold
```

```
[104]: #Reading the required files, each dataset is stored in a seperate file and has␣
        ↪to be read accordingly
       appl_df = pd.read_csv("Historical_Data_APPL.csv")
       appl_df.head()
```

```
[104]:          Date   Close     Volume    Open    High     Low
       0  08/09/2024  216.24   42201650  212.10  216.78  211.97
       1  08/08/2024  213.31   47161150  213.11  214.20  208.83
       2  08/07/2024  209.82   63516420  206.90  213.64  206.39
       3  08/06/2024  207.23   69660490  205.30  209.99  201.07
       4  08/05/2024  209.27  119548600  199.09  213.50  196.00
```

```
  [3]: #Converting the Date column to the required format
       appl_df['Date'] = pd.to_datetime(appl_df['Date'], format='%m/%d/%Y')
       appl_df.head()
```

Appendix

```
[3]:         Date   Close     Volume    Open    High     Low
     0 2024-08-09  216.24   42201650  212.10  216.78  211.97
     1 2024-08-08  213.31   47161150  213.11  214.20  208.83
     2 2024-08-07  209.82   63516420  206.90  213.64  206.39
     3 2024-08-06  207.23   69660490  205.30  209.99  201.07
     4 2024-08-05  209.27  119548600  199.09  213.50  196.00
```

```
[4]: #Making a copy of our dataframe.
     model_df = appl_df.copy()
```

```
[5]: #Sorting the columns by the Daye column in ascending order
     model_df = model_df.sort_values(by='Date').reset_index(drop=True)
```

```
[6]: #Defining the combined Kernel
     def combined_kernel(x1, x2, length_scale=1.0, variance=1.0, constant=1.0,␣
      ↪linear_variance=1.0):
         sqdist = np.sum(x1**2, 1).reshape(-1, 1) + np.sum(x2**2, 1) - 2 * np.dot(x1,␣
      ↪x2.T)
         rbf_part = variance * np.exp(-0.5 * sqdist / length_scale**2)
         constant_part = constant
         linear_part = linear_variance * np.dot(x1, x2.T)
         return rbf_part + constant_part + linear_part
```

```
[41]: #Defining the same Combined Kernel but this one runs with the help of the GPU,␣
       ↪used while GRID SEACRH for faster computing
      def combined_kernel2(x1, x2, length_scale=1.0, variance=1.0, constant=1.0,␣
       ↪linear_variance=1.0):
          # Ensure inputs are 2D arrays/tensors
          if isinstance(x1, np.ndarray):
              sqdist = np.sum(x1**2, axis=1).reshape(-1, 1) + np.sum(x2**2, axis=1) -␣
       ↪2 * np.dot(x1, x2.T)
              rbf_part = variance * np.exp(-0.5 * sqdist / length_scale**2)
              linear_part = linear_variance * np.dot(x1, x2.T)
          elif isinstance(x1, torch.Tensor):
              sqdist = torch.sum(x1**2, dim=1).reshape(-1, 1) + torch.sum(x2**2,␣
       ↪dim=1) - 2 * torch.matmul(x1, x2.T)
              rbf_part = variance * torch.exp(-0.5 * sqdist / length_scale**2)
              linear_part = linear_variance * torch.matmul(x1, x2.T)
          else:
              raise ValueError("Inputs should be either numpy arrays or torch tensors.
       ↪")

          constant_part = constant  # Assuming constant is a scalar

          return rbf_part + constant_part + linear_part
```

2

Appendix

```
[7]: #Dividing the dataset into different groups by month to prepare disjointed prior␣
      ↪function
     model_df = model_df[(model_df['Date'] >= '2015-01-01')]
     # Define the start date
     start_date = pd.to_datetime('2015-01-01')
     # Calculate the number of months since the start date
     model_df['Months_Since_Start'] = model_df['Date'].apply(lambda x: (x.year -␣
      ↪start_date.year) * 12 + (x.month - start_date.month))
     # Create the 'Group' column by dividing the months by 6 and adding 1 (to start␣
      ↪from group 1)
     model_df['Group'] = (model_df['Months_Since_Start'] // 1) + 1
     #Drop the 'Months_Since_Start' column as it's no longer needed
     model_df.drop('Months_Since_Start', axis=1, inplace=True)
     # Display the dataframe with the new 'Group' column
     print(model_df)
```

```
           Date     Close     Volume      Open      High       Low  Group
100  2015-01-02   27.3325  212575080   27.8475   27.8600   26.8375      1
101  2015-01-05   26.5625  256843520   27.0725   27.1625   26.3525      1
102  2015-01-06   26.5650  262729000   26.6350   26.8575   26.1575      1
103  2015-01-07   26.9375  159933400   26.8000   27.0500   26.6737      1
104  2015-01-08   27.9725  236675040   27.3075   28.0375   27.1750      1
...         ...       ...        ...       ...       ...       ...    ...
2512 2024-08-05  209.2700  119548600  199.0900  213.5000  196.0000    116
2513 2024-08-06  207.2300   69660490  205.3000  209.9900  201.0700    116
2514 2024-08-07  209.8200   63516420  206.9000  213.6400  206.3900    116
2515 2024-08-08  213.3100   47161150  213.1100  214.2000  208.8300    116
2516 2024-08-09  216.2400   42201650  212.1000  216.7800  211.9700    116

[2417 rows x 7 columns]
```

```
[8]: # Making a Ordinal Date column to change from date format to integer for␣
      ↪calculation purposes

     model_df['Date_ordinal'] = model_df['Date'].apply(lambda x: x.toordinal())
     print(model_df)
```

```
           Date     Close     Volume      Open      High       Low  Group  \
100  2015-01-02   27.3325  212575080   27.8475   27.8600   26.8375      1
101  2015-01-05   26.5625  256843520   27.0725   27.1625   26.3525      1
102  2015-01-06   26.5650  262729000   26.6350   26.8575   26.1575      1
103  2015-01-07   26.9375  159933400   26.8000   27.0500   26.6737      1
104  2015-01-08   27.9725  236675040   27.3075   28.0375   27.1750      1
...         ...       ...        ...       ...       ...       ...    ...
2512 2024-08-05  209.2700  119548600  199.0900  213.5000  196.0000    116
2513 2024-08-06  207.2300   69660490  205.3000  209.9900  201.0700    116
2514 2024-08-07  209.8200   63516420  206.9000  213.6400  206.3900    116
2515 2024-08-08  213.3100   47161150  213.1100  214.2000  208.8300    116
```

Appendix

```
2516 2024-08-09  216.2400   42201650  212.1000  216.7800  211.9700    116

      Date_ordinal
100         735600
101         735603
102         735604
103         735605
104         735606
...            ...
2512        739103
2513        739104
2514        739105
2515        739106
2516        739107

[2417 rows x 8 columns]
```

[9]:
```python
# Preparing the Linear Disjointed prior
# Initialize the prior_values column with NaN
model_df['prior_values'] = np.nan

# Get unique groups in the dataframe
groups = sorted(model_df['Group'].unique())

# Iterating over the groups
for i, group in enumerate(groups):
    if (i < 3) & (i > 1):
        group_df = model_df[(model_df['Group'] == 1) | (model_df['Group'] == 2)|␣
↪(model_df['Group'] == 3) ]
        x = group_df['Date_ordinal']
        y = group_df['Close']

        # Fiting a linear line
        coefs = np.polyfit(x, y, 1)
        poly = np.poly1d(coefs)

        # Predicting prior values using the fitted polynomial
        model_df.loc[(model_df['Group'] == 1)|(model_df['Group'] == 2) |␣
↪(model_df['Group'] == 3), 'prior_values'] = poly(x)
    elif i >= 3 :
        # For subsequent groups, use the previous three groups to fit the␣
↪polynomial
        prev_groups = groups[i-3:i]
        prev_df = model_df[model_df['Group'].isin(prev_groups)]
        x = prev_df['Date_ordinal']
        y = prev_df['Close']
```

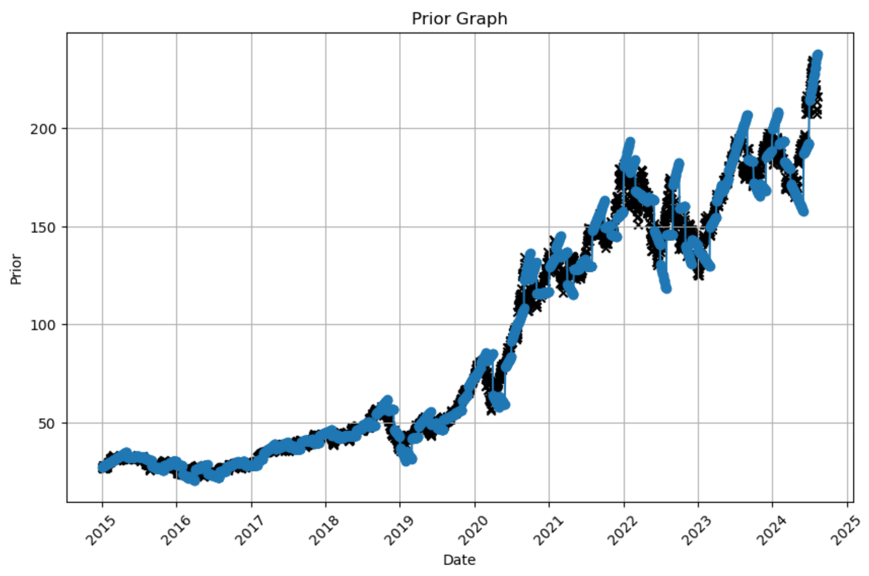
Appendix

```
        # Fit a linear line
        coefs = np.polyfit(x, y, 1)
        poly = np.poly1d(coefs)

        # Predict prior values for the current group using the fitted polynomial
        current_group_df = model_df[model_df['Group'] == group]
        x_current = current_group_df['Date_ordinal']
        model_df.loc[model_df['Group'] == group, 'prior_values'] =␣
 ↪poly(x_current)

# Print the dataframe to check the prior_values column
print(model_df[['Date', 'Close', 'Group', 'prior_values']])
```

```
           Date     Close  Group  prior_values
100  2015-01-02   27.3325      1     27.374777
101  2015-01-05   26.5625      1     27.564256
102  2015-01-06   26.5650      1     27.627415
103  2015-01-07   26.9375      1     27.690575
104  2015-01-08   27.9725      1     27.753735
...         ...       ...    ...           ...
2512 2024-08-05  209.2700    116    235.610621
2513 2024-08-06  207.2300    116    236.202646
2514 2024-08-07  209.8200    116    236.794671
2515 2024-08-08  213.3100    116    237.386697
2516 2024-08-09  216.2400    116    237.978722

[2417 rows x 4 columns]
```

[10]:
```
os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'

# Plot the graph
plt.figure(figsize=(10, 6))
plt.plot(model_df['Date'], model_df['prior_values'], marker='o')
#plt.plot(model_df['Date'], model_df['prior_values'])
plt.scatter(model_df['Date'], model_df['Close'], c='k', marker='x', label='Data')
plt.title('Prior Graph')
plt.xlabel('Date')
plt.ylabel('Prior')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.grid(True)
plt.show()
```

Appendix



[11]:
```
#Divide The data into train and validation set
model_train2 = model_df[['Date','Close','Date_ordinal','prior_values']].copy()
model_train_subset2 = model_train2[(model_train2['Date'] >= '2015-01-01') &␣
↪(model_train2['Date'] < '2024-03-01')]
model_train_subset_test2 = model_train2[(model_train2['Date'] >= '2024-03-01') &␣
↪(model_train2['Date'] <= '2024-03-31')]
model_train_subset2['Date'] = pd.to_datetime(model_train_subset2['Date'])
model_train_subset_test2['Date'] = pd.
↪to_datetime(model_train_subset_test2['Date'])
model_train_subset_test2.head(10)
```

```
C:\Users\aman1\AppData\Local\Temp\ipykernel_24032\776334061.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  model_train_subset2['Date'] = pd.to_datetime(model_train_subset2['Date'])
C:\Users\aman1\AppData\Local\Temp\ipykernel_24032\776334061.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

Appendix

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  model_train_subset_test2['Date'] =
pd.to_datetime(model_train_subset_test2['Date'])
```

[11]:

```
            Date   Close  Date_ordinal  prior_values
2405 2024-03-01  179.66        738946    182.846461
2406 2024-03-04  175.10        738949    182.445027
2407 2024-03-05  170.12        738950    182.311215
2408 2024-03-06  169.12        738951    182.177404
2409 2024-03-07  169.00        738952    182.043593
2410 2024-03-08  170.73        738953    181.909781
2411 2024-03-11  172.75        738956    181.508347
2412 2024-03-12  173.23        738957    181.374536
2413 2024-03-13  171.13        738958    181.240725
2414 2024-03-14  173.00        738959    181.106913
```

[12]:

```
#Further transformations of columns to aid in calulations and storing the prior␣
↪values for training and validation seperately
model_train_final2 = model_train_subset2.copy()
model_train_final_test2 = model_train_subset_test2.copy()

mean = model_train_final2['Date_ordinal'].mean()
std = model_train_final2['Date_ordinal'].std()


prior2 = model_train_final2['prior_values']
prior_test2 = model_train_final_test2['prior_values']

model_train_final2['Date_ordinal_normalized'] =␣
↪(model_train_final2['Date_ordinal'] - mean)/std
model_train_final_test2['Date_ordinal_normalized'] =␣
↪(model_train_final_test2['Date_ordinal'] - mean)/std

X_train3 = model_train_final2['Date_ordinal_normalized'].values
y_train3 = model_train_final2['Close'].values

X_test3 = model_train_final_test2['Date_ordinal_normalized'].values
y_test3 = model_train_final_test2['Close'].values

prior_test2
```

[12]:

```
2405    182.846461
2406    182.445027
2407    182.311215
2408    182.177404
2409    182.043593
```

Appendix

```
2410    181.909781
2411    181.508347
2412    181.374536
2413    181.240725
2414    181.106913
2415    180.973102
2416    180.571668
2417    180.437857
2418    180.304046
2419    180.170234
2420    180.036423
2421    179.634989
2422    179.501178
2423    179.367366
2424    179.233555
Name: prior_values, dtype: float64
```

[13]:
```
#Further Transformations
X_obs2 = X_train3
y_obs2 = y_train3

X_obs2 = X_obs2.reshape(-1, 1)
```

[14]:
```
#Finding Optimal parameters through Bayesian method
random.seed(10)
def log_marginal_likelihood(params, X, y, prior_mean):
    length_scale, variance, constant1, constant2 = params
    K = combined_kernel(X, X, length_scale, variance, constant1,constant2) +␣
 ↪1e-3 * np.eye(len(X))  # Add small value for numerical stability
    L = np.linalg.cholesky(K)
    y_centered = y - prior_mean
    S1 = np.linalg.solve(L, y_centered)
    S2 = np.linalg.solve(L.T, S1)
    log_likelihood = -0.5 * np.dot(y_centered.T, S2)
    log_likelihood -= np.sum(np.log(np.diagonal(L)))
    log_likelihood -= 0.5 * len(X) * np.log(2 * np.pi)
    return -log_likelihood  # We minimize negative log likelihood

# Define initial hyperparameters
initial_params = [1.0, 1.0, 1.0,1.0]  # [length_scale, variance, constant1,␣
 ↪constant2]

# Optimize hyperparameters
res = minimize(log_marginal_likelihood, initial_params, args=(X_obs2,␣
 ↪y_obs2,prior2), bounds=((None, None), (1e-5, None), (1e-5, None) ,(1e-5,␣
 ↪None)), method='L-BFGS-B')
optimal_params = res.x
```

Appendix

```
print("Optimal parameters:", optimal_params)

optimized_length_scale = optimal_params[0]
optimized_variance = optimal_params[1]
optimized_constant = optimal_params[2]
optimized_constant2 = optimal_params[3]
```

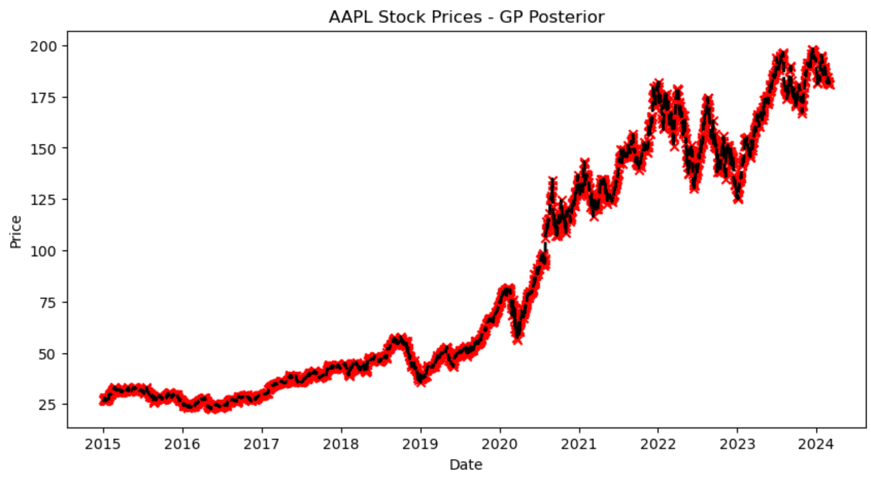
```
Optimal parameters: [0.00180051 1.05998141 0.99999292 0.99999292]
```

```
[15]: #Calculating Posterior
optimized_length_scale = optimal_params[0]
optimized_variance = optimal_params[1]
optimized_constant = optimal_params[2]
optimized_constant2 = optimal_params[3]


noise = 1e-3
num_samples = 25

# Compute covariance matrices for predictions
K_obs = combined_kernel(X_obs2, X_obs2, optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)
#K_s = combined_kernel(X_obs2, X_obs2, optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)
#K_ss = combined_kernel(X_obs2, X_obs2, optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)
K_obs_inv = np.linalg.inv(K_obs + noise * np.eye(len(X_obs2)))

# Compute the mean and covariance of the posterior
mu_s = prior2 + np.dot(K_s, np.dot(K_obs_inv, (y_obs2 - prior2)))
cov_s = K_ss - np.dot(K_s, np.dot(K_obs_inv, K_s.T))
cov_s_actual = []
mu_s = mu_s.to_numpy()

# Sample from the posterior
L_post = cholesky(cov_s + noise * np.eye(len(X_obs2)), lower=True)
f_post = mu_s.reshape(-1, 1) + np.dot(L_post, np.random.
 ↪normal(size=(len(X_obs2), num_samples)))
```

```
[17]: os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'

# Plot samples from the posterior
plt.figure(figsize=(10, 5))
#plt.plot(model_train_final2['Date'], f_post, label="Posterior Samples")
plt.scatter(model_train_final2['Date'], y_obs2, c='red', marker='x',␣
 ↪label="Observed Data")
```

Appendix

```
plt.plot(model_train_final2['Date'], mu_s, 'k--', lw=2, label="Posterior Mean")
plt.fill_between(model_train_final2['Date'],
                 mu_s.flatten() - 1.96 * np.sqrt(np.diag(cov_s)),
                 mu_s.flatten() + 1.96 * np.sqrt(np.diag(cov_s)),
                 alpha=0.2, label="95% Confidence Interval")
plt.title("AAPL Stock Prices - GP Posterior")
plt.xlabel("Date")
plt.ylabel("Price")
#plt.legend()
plt.show()
```



[18]:
```
# Compute the covariance vector between the new date and observed dates
K_new = combined_kernel(X_test3.reshape(-1, 1), X_obs2, optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)
K_new_new = combined_kernel(X_test3.reshape(-1, 1), X_test3.reshape(-1, 1),␣
 ↪optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)

# Compute the mean and variance of the posterior at the new date
#mu_new = np.dot(K_new, np.dot(K_obs_inv, y_obs))
mu_new = np.dot(K_new, np.dot(K_obs_inv, (y_obs2 - prior2) )) + prior_test2
mu_new = mu_new.to_numpy()
cov_new = K_new_new - np.dot(K_new, np.dot(K_obs_inv, K_new.T))

# Compute the standard deviation
sigma_new = np.sqrt(np.diag(cov_new))
```

Appendix

```
# Print the predicted means and standard deviations
for i in range(len(X_test3)):
    print(f"Predicted mean at new date {X_test3[i]}: {mu_new[i]}")
    print(f"Actual value at new date {X_test3[i]}: {y_test3[i]}")
    print(f"Predicted standard deviation at new date {X_test3[i]}:␣
 ↪{sigma_new[i]}")
```

```
Predicted mean at new date 1.7339104196097341: 172.52719427196757
Actual value at new date 1.7339104196097341: 179.66
Predicted standard deviation at new date 1.7339104196097341: 0.2762431142878672
Predicted mean at new date 1.7370170074081155: 181.58797179247745
Actual value at new date 1.7370170074081155: 175.1
Predicted standard deviation at new date 1.7370170074081155: 1.0193639229317708
Predicted mean at new date 1.7380525366742425: 182.03770131791703
Actual value at new date 1.7380525366742425: 170.12
Predicted standard deviation at new date 1.7380525366742425: 1.0313183999201834
Predicted mean at new date 1.7390880659403696: 182.03985716313764
Actual value at new date 1.7390880659403696: 169.12
Predicted standard deviation at new date 1.7390880659403696: 1.0322029291720485
Predicted mean at new date 1.7401235952064966: 181.92836884037786
Actual value at new date 1.7401235952064966: 169.0
Predicted standard deviation at new date 1.7401235952064966: 1.0322479414802554
Predicted mean at new date 1.7411591244726239: 181.79711881705987
Actual value at new date 1.7411591244726239: 170.73
Predicted standard deviation at new date 1.7411591244726239: 1.0322529157187879
Predicted mean at new date 1.744265712271005: 181.39573178181126
Actual value at new date 1.744265712271005: 172.75
Predicted standard deviation at new date 1.744265712271005: 1.0322603594589077
Predicted mean at new date 1.745301241537132: 181.2618581971758
Actual value at new date 1.745301241537132: 173.23
Predicted standard deviation at new date 1.745301241537132: 1.0322627748716102
Predicted mean at new date 1.746336770803259: 181.12798459521764
Actual value at new date 1.746336770803259: 171.13
Predicted standard deviation at new date 1.746336770803259: 1.0322651916981245
Predicted mean at new date 1.7473723000693864: 180.99411099289205
Actual value at new date 1.7473723000693864: 173.0
Predicted standard deviation at new date 1.7473723000693864: 1.0322676099516388
Predicted mean at new date 1.7484078293355134: 180.86023739053712
Actual value at new date 1.7484078293355134: 172.62
Predicted standard deviation at new date 1.7484078293355134: 1.0322700296358531
Predicted mean at new date 1.7515144171338946: 180.45861658353988
Actual value at new date 1.7515144171338946: 173.72
Predicted standard deviation at new date 1.7515144171338946: 1.0322772972554406
Predicted mean at new date 1.7525499464000218: 180.32474298119314
Actual value at new date 1.7525499464000218: 176.08
Predicted standard deviation at new date 1.7525499464000218: 1.032279722651806
Predicted mean at new date 1.7535854756661489: 180.19086937885652
```

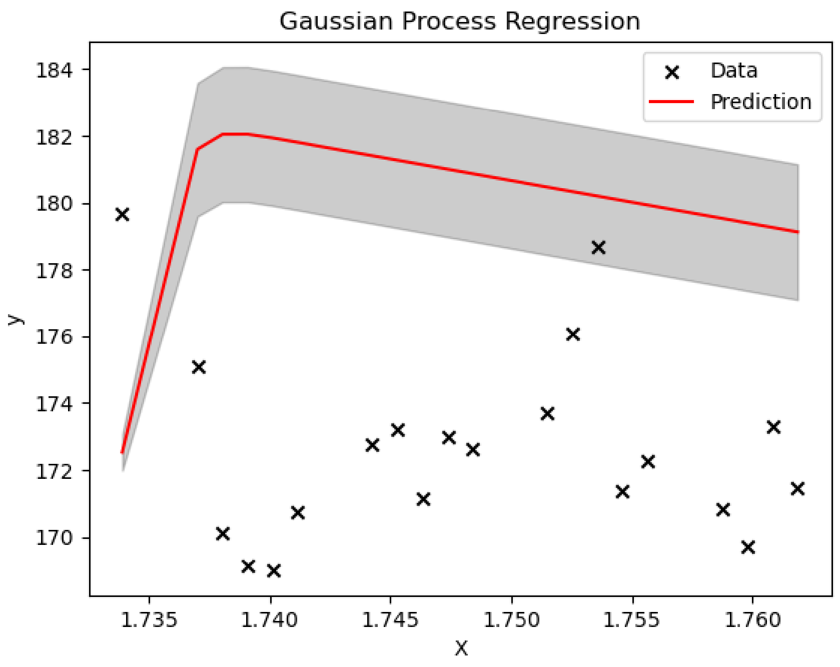
Appendix

```
Actual value at new date 1.7535854756661489: 178.67
Predicted standard deviation at new date 1.7535854756661489: 1.0322821494796819
Predicted mean at new date 1.754621004932276: 180.05699577653115
Actual value at new date 1.754621004932276: 171.37
Predicted standard deviation at new date 1.754621004932276: 1.0322845777334804
Predicted mean at new date 1.755656534198403: 179.92312217416395
Actual value at new date 1.755656534198403: 172.28
Predicted standard deviation at new date 1.755656534198403: 1.0322870074170785
Predicted mean at new date 1.7587631219967843: 179.5215013671684
Actual value at new date 1.7587631219967843: 170.85
Predicted standard deviation at new date 1.7587631219967843: 1.0322943050388373
Predicted mean at new date 1.7597986512629114: 179.38762776483395
Actual value at new date 1.7597986512629114: 169.71
Predicted standard deviation at new date 1.7597986512629114: 1.0322967404366716
Predicted mean at new date 1.7608341805290384: 179.25375416247653
Actual value at new date 1.7608341805290384: 173.31
Predicted standard deviation at new date 1.7608341805290384: 1.03229917726232
Predicted mean at new date 1.7618697097951654: 179.1198805601573
Actual value at new date 1.7618697097951654: 171.48
Predicted standard deviation at new date 1.7618697097951654: 1.0323016155165028
```

[19]:
```
os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'

plt.figure()
plt.scatter(X_test3, y_test3, c='k', marker='x', label='Data')
plt.plot(X_test3, mu_new, c='r', label='Prediction')
#plt.fill_between(X_test2.ravel(),y_test2 + y_test2*0.025, y_test2 - y_test2*0.
↪025, alpha=0.2, color='k')
plt.fill_between(X_test3.ravel(), mu_new - 1.96*sigma_new, mu_new + 1.
↪96*sigma_new, alpha=0.2, color='k')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Gaussian Process Regression')
plt.legend()
plt.show()
```

12

Appendix



[20]:
```
#list of values to check for grid search.
optimized_length_scale_list = [x / 20000 for x in range(0, 2001, 5)]
optimized_variance_list = [x / 10 for x in range(0, 1001,5)]

optimized_length_scale_list.remove(0)
optimized_variance_list.remove(0)
```

[21]:
```
#Performing Grid Search normal
dict_scores = {'Length_Scale':[],'Variance':[],'RMSE':[],'RMCE':[],'RMQE':[]}

for optimized_length_scale in optimized_length_scale_list:
    for optimized_variance in optimized_variance_list:
        try:
            noise = 1e-3
            num_samples = 25
            optimized_constant = optimal_params[2]
            optimized_constant2 = optimal_params[3]

            K_obs = combined_kernel(X_obs2, X_obs2, optimized_length_scale,␣
↪optimized_variance,optimized_constant,optimized_constant2)
```

13

Appendix

```
            #K_s = combined_kernel(X_obs2, X_obs2, optimized_length_scale,␣
↪optimized_variance,optimized_constant,optimized_constant2)
            #K_ss = combined_kernel(X_obs2, X_obs2, optimized_length_scale,␣
↪optimized_variance,optimized_constant,optimized_constant2)
            K_obs_inv = np.linalg.inv(K_obs + noise * np.eye(len(X_obs2)))
            # Compute the covariance vector between the new date and observed␣
↪dates
            K_new = combined_kernel(X_test3.reshape(-1, 1), X_obs2,␣
↪optimized_length_scale,␣
↪optimized_variance,optimized_constant,optimized_constant2)
            K_new_new = combined_kernel(X_test3.reshape(-1, 1), X_test3.
↪reshape(-1, 1), optimized_length_scale,␣
↪optimized_variance,optimized_constant,optimized_constant2)

            # Compute the mean and variance of the posterior at the new dates
            mu_new = np.dot(K_new, np.dot(K_obs_inv, (y_obs2 - prior2) )) +␣
↪prior_test2
            mu_new = mu_new.to_numpy()
            #cov_new = K_new_new - np.dot(K_new, np.dot(K_obs_inv, K_new.T))

            # Compute the standard deviation
            #sigma_new = np.sqrt(np.diag(cov_new))

            mse = sklearn.metrics.mean_squared_error(y_test3, mu_new)
            mce = np.mean(np.abs(y_test3 - mu_new) ** 3)
            mqe = np.mean(np.abs(y_test3 - mu_new) ** 4)
            dict_scores['Length_Scale'].append(optimized_length_scale)
            dict_scores['Variance'].append(optimized_variance)
            dict_scores['RMSE'].append(mse)
            dict_scores['RMCE'].append(mce)
            dict_scores['RMQE'].append(mqe)

        except Exception as e:
            print(f"Error with length scale {optimized_length_scale} and␣
↪variance {optimized_variance}: {e}")
            continue
```

```
[42]: #Performing GRID Search with parameters in CUDA

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
dict_scores = {'Length_Scale':[], 'Variance':[], 'RMSE':[], 'RMCE':[], 'RMQE':[]}

for optimized_length_scale in optimized_length_scale_list:
    for optimized_variance in optimized_variance_list:
        try:
            noise = 1e-3
```

Appendix

```
            num_samples = 25
            optimized_constant = optimal_params[2]
            optimized_constant2 = optimal_params[3]

            # Move data to GPU
            X_obs2_torch = torch.tensor(X_obs2, dtype=torch.float32).to(device)
            X_test3_torch = torch.tensor(X_test3, dtype=torch.float32).to(device)
            y_obs2_torch = torch.tensor(y_obs2, dtype=torch.float32).to(device)
            prior2_torch = torch.tensor(prior2.to_numpy(), dtype=torch.float32).
↪to(device)
            prior_test2_torch = torch.tensor(prior_test2.to_numpy(), dtype=torch.
↪float32).to(device)

            # Compute kernels
            K_obs = combined_kernel2(X_obs2_torch, X_obs2_torch,␣
↪optimized_length_scale, optimized_variance, optimized_constant,␣
↪optimized_constant2)
            K_obs_inv = torch.inverse(K_obs + noise * torch.
↪eye(len(X_obs2_torch)).to(device))

            K_new = combined_kernel2(X_test3_torch.reshape(-1, 1), X_obs2_torch,␣
↪optimized_length_scale, optimized_variance, optimized_constant,␣
↪optimized_constant2)
            K_new_new = combined_kernel2(X_test3_torch.reshape(-1, 1),␣
↪X_test3_torch.reshape(-1, 1), optimized_length_scale, optimized_variance,␣
↪optimized_constant, optimized_constant2)

            mu_new = torch.matmul(K_new, torch.matmul(K_obs_inv, (y_obs2_torch -␣
↪prior2_torch))) + prior_test2_torch
            mu_new = mu_new.cpu().numpy()  # Move back to CPU for compatibility␣
↪with sklearn

            mse = sklearn.metrics.mean_squared_error(y_test3, mu_new)
            mce = np.mean(np.abs(y_test3 - mu_new) ** 3)
            mqe = np.mean(np.abs(y_test3 - mu_new) ** 4)

            dict_scores['Length_Scale'].append(optimized_length_scale)
            dict_scores['Variance'].append(optimized_variance)
            dict_scores['RMSE'].append(mse)
            dict_scores['RMCE'].append(mce)
            dict_scores['RMQE'].append(mqe)

        except Exception as e:
            print(f"Error with length scale {optimized_length_scale} and␣
↪variance {optimized_variance}: {e}")
            continue
```

15

Appendix

```
Error with length scale 0.0475 and variance 15.0: linalg.inv: The diagonal
element 2305 is zero, the inversion could not be completed because the input
matrix is singular.
Error with length scale 0.0645 and variance 68.5: linalg.inv: The diagonal
element 2305 is zero, the inversion could not be completed because the input
matrix is singular.
Error with length scale 0.0725 and variance 6.0: linalg.inv: The diagonal
element 2305 is zero, the inversion could not be completed because the input
matrix is singular.
Error with length scale 0.07525 and variance 19.5: linalg.inv: The diagonal
element 2305 is zero, the inversion could not be completed because the input
matrix is singular.
Error with length scale 0.08225 and variance 35.5: linalg.inv: The diagonal
element 2305 is zero, the inversion could not be completed because the input
matrix is singular.
Error with length scale 0.08725 and variance 30.5: linalg.inv: The diagonal
element 2305 is zero, the inversion could not be completed because the input
matrix is singular.
Error with length scale 0.08775 and variance 47.5: linalg.inv: The diagonal
element 2305 is zero, the inversion could not be completed because the input
matrix is singular.
Error with length scale 0.09 and variance 40.5: linalg.inv: The diagonal element
2305 is zero, the inversion could not be completed because the input matrix is
singular.
Error with length scale 0.09525 and variance 39.5: linalg.inv: The diagonal
element 2305 is zero, the inversion could not be completed because the input
matrix is singular.
Error with length scale 0.09775 and variance 94.5: linalg.inv: The diagonal
element 2305 is zero, the inversion could not be completed because the input
matrix is singular.
```

[43]:
```
df_scores = pd.DataFrame(dict_scores)
```

[ ]:
```
```

[44]:
```
# Find the index of the minimum RMSE value
min_rmse_index = df_scores['RMSE'].idxmin()
min_rmce_index = df_scores['RMCE'].idxmin()
min_rmqe_index = df_scores['RMQE'].idxmin()

# Print the row with the lowest RMSE value
print(df_scores.loc[min_rmse_index])
print(df_scores.loc[min_rmce_index])
print(df_scores.loc[min_rmqe_index])
```

```
Length_Scale      0.012250
Variance          2.000000
RMSE             11.464257
```

```
RMCE              54.469501
RMQE             282.399341
Name: 9603, dtype: float64
Length_Scale       0.012250
Variance           2.000000
RMSE              11.464257
RMCE              54.469501
RMQE             282.399341
Name: 9603, dtype: float64
Length_Scale       0.012250
Variance           2.000000
RMSE              11.464257
RMCE              54.469501
RMQE             282.399341
Name: 9603, dtype: float64
```

[89]:

```
#Checking the fit for validation set with the new found optimal hyperparameters.
optimized_length_scale = 0.012250
optimized_variance = 2.0
optimized_constant = optimal_params[2]
optimized_constant2 = optimal_params[3]

K_obs = combined_kernel(X_obs2, X_obs2, optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)
#K_s = combined_kernel(X_obs2, X_obs2, optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)
#K_ss = combined_kernel(X_obs2, X_obs2, optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)
K_obs_inv = np.linalg.inv(K_obs + noise * np.eye(len(X_obs2)))

# Compute the covariance vector between the new date and observed dates
K_new = combined_kernel(X_test3.reshape(-1, 1), X_obs2, optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)
K_new_new = combined_kernel(X_test3.reshape(-1, 1), X_test3.reshape(-1, 1),␣
 ↪optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)

# Compute the mean and variance of the posterior at the new date
#mu_new = np.dot(K_new, np.dot(K_obs_inv, y_obs))
mu_new = np.dot(K_new, np.dot(K_obs_inv, (y_obs2 - prior2) )) + prior_test2
mu_new = mu_new.to_numpy()
cov_new = K_new_new - np.dot(K_new, np.dot(K_obs_inv, K_new.T))

# Compute the standard deviation
cov_new = cov_new
sigma_new = np.sqrt(np.diag(cov_new))
```

17

Appendix

```
# Print the predicted means and standard deviations
for i in range(len(X_test3)):
    print(f"Predicted mean at new date {X_test3[i]}: {mu_new[i]}")
    print(f"Actual value at new date {X_test3[i]}: {y_test3[i]}")
    print(f"Predicted standard deviation at new date {X_test3[i]}:␣
↪{sigma_new[i]}")
```

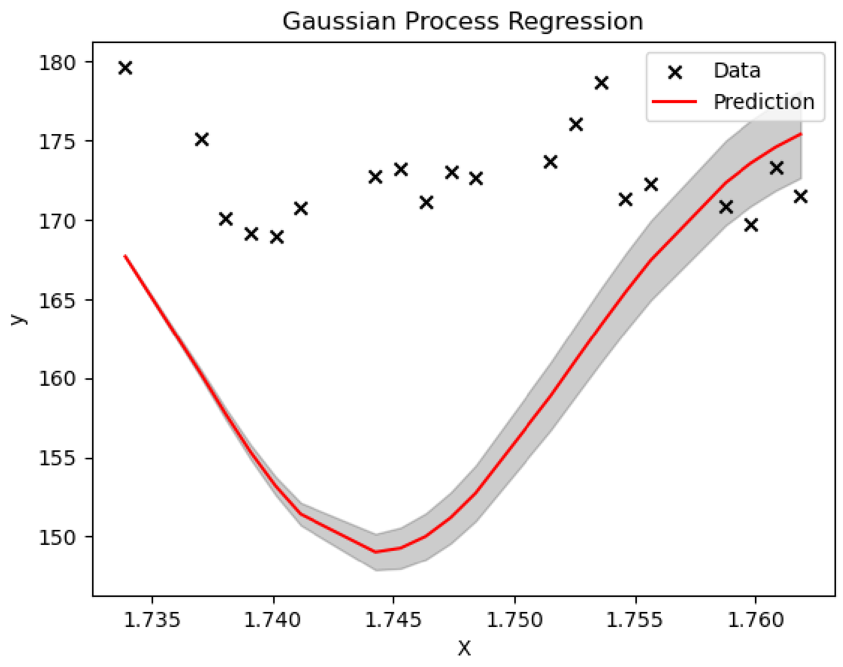
```
Predicted mean at new date 1.7339104196097341: 167.68532021138526
Actual value at new date 1.7339104196097341: 179.66
Predicted standard deviation at new date 1.7339104196097341:
0.041065331168116365
Predicted mean at new date 1.7370170074081155: 160.31189916524454
Actual value at new date 1.7370170074081155: 175.1
Predicted standard deviation at new date 1.7370170074081155: 0.13560924705587982
Predicted mean at new date 1.7380525366742425: 157.74271993029106
Actual value at new date 1.7380525366742425: 170.12
Predicted standard deviation at new date 1.7380525366742425: 0.1821740141572135
Predicted mean at new date 1.7390880659403696: 155.32841865104274
Actual value at new date 1.7390880659403696: 169.12
Predicted standard deviation at new date 1.7390880659403696: 0.23594227614994318
Predicted mean at new date 1.7401235952064966: 153.1874849138694
Actual value at new date 1.7401235952064966: 169.0
Predicted standard deviation at new date 1.7401235952064966: 0.2963914309415859
Predicted mean at new date 1.7411591244726239: 151.42149123785202
Actual value at new date 1.7411591244726239: 170.73
Predicted standard deviation at new date 1.7411591244726239: 0.36276085316723655
Predicted mean at new date 1.744265712271005: 149.02275021572132
Actual value at new date 1.744265712271005: 172.75
Predicted standard deviation at new date 1.744265712271005: 0.5868233884367116
Predicted mean at new date 1.745301241537132: 149.2674715264875
Actual value at new date 1.745301241537132: 173.23
Predicted standard deviation at new date 1.745301241537132: 0.6656428051342558
Predicted mean at new date 1.746336770803259: 150.00579862952873
Actual value at new date 1.746336770803259: 171.13
Predicted standard deviation at new date 1.746336770803259: 0.7442980450426868
Predicted mean at new date 1.7473723000693864: 151.1863146899559
Actual value at new date 1.7473723000693864: 173.0
Predicted standard deviation at new date 1.7473723000693864: 0.8214908250040904
Predicted mean at new date 1.7484078293355134: 152.7412682071008
Actual value at new date 1.7484078293355134: 172.62
Predicted standard deviation at new date 1.7484078293355134: 0.896025038665289
Predicted mean at new date 1.7515144171338946: 158.84604840520478
Actual value at new date 1.7515144171338946: 173.72
Predicted standard deviation at new date 1.7515144171338946: 1.0941067175925128
Predicted mean at new date 1.7525499464000218: 161.08576594665647
Actual value at new date 1.7525499464000218: 176.08
Predicted standard deviation at new date 1.7525499464000218: 1.149417590243002
Predicted mean at new date 1.7535854756661489: 163.30259235585982
```

```
Actual value at new date 1.7535854756661489: 178.67
Predicted standard deviation at new date 1.7535854756661489: 1.1987916210269653
Predicted mean at new date 1.754621004932276: 165.43547133149696
Actual value at new date 1.754621004932276: 171.37
Predicted standard deviation at new date 1.754621004932276: 1.242186343151669
Predicted mean at new date 1.755656534198403: 167.43535266781691
Actual value at new date 1.755656534198403: 172.28
Predicted standard deviation at new date 1.755656534198403: 1.2797366725602182
Predicted mean at new date 1.7587631219967843: 172.33251386568008
Actual value at new date 1.7587631219967843: 170.85
Predicted standard deviation at new date 1.7587631219967843: 1.3607110991701574
Predicted mean at new date 1.7597986512629114: 173.5530444211763
Actual value at new date 1.7597986512629114: 169.71
Predicted standard deviation at new date 1.7597986512629114: 1.378720876409209
Predicted mean at new date 1.7608341805290384: 174.56925255883834
Actual value at new date 1.7608341805290384: 173.31
Predicted standard deviation at new date 1.7608341805290384: 1.393136762561384
Predicted mean at new date 1.7618697097951654: 175.39274479585583
Actual value at new date 1.7618697097951654: 171.48
Predicted standard deviation at new date 1.7618697097951654: 1.4045008663014626
```

[90]:
```
os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'

plt.figure()
plt.scatter(X_test3, y_test3, c='k', marker='x', label='Data')
plt.plot(X_test3, mu_new, c='r', label='Prediction')
#plt.fill_between(X_test2.ravel(),y_test2 + y_test2*0.025, y_test2 - y_test2*0.
 ↪025, alpha=0.2, color='k')
plt.fill_between(X_test3.ravel(), mu_new - 1.96*sigma_new, mu_new + 1.
 ↪96*sigma_new, alpha=0.2, color='k')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Gaussian Process Regression')
plt.legend()
plt.show()
```

19

Appendix



```
[91]: #Splitting the data into Train and Test set
model_train3 = model_df[['Date','Close','Date_ordinal','prior_values']].copy()
model_train_subset3 = model_train3[(model_train3['Date'] >= '2015-01-01') &␣
 ↪(model_train3['Date'] < '2024-04-01')]
model_train_subset_test3 = model_train3[(model_train2['Date'] >= '2024-04-01') &␣
 ↪(model_train3['Date'] <= '2024-04-30')]
model_train_subset3['Date'] = pd.to_datetime(model_train_subset3['Date'])
model_train_subset3['Date_ordinal'] = model_train_subset3['Date'].apply(lambda x:
 ↪ x.toordinal())
model_train_subset_test3['Date'] = pd.
 ↪to_datetime(model_train_subset_test3['Date'])
model_train_subset_test3['Date_ordinal'] = model_train_subset_test3['Date'].
 ↪apply(lambda x: x.toordinal())
model_train_subset_test3.head(10)
```

```
C:\Users\aman1\AppData\Local\Temp\ipykernel_24032\1618671393.py:6:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

Appendix

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  model_train_subset3['Date'] = pd.to_datetime(model_train_subset3['Date'])
C:\Users\aman1\AppData\Local\Temp\ipykernel_24032\1618671393.py:7:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  model_train_subset3['Date_ordinal'] = model_train_subset3['Date'].apply(lambda
x: x.toordinal())
C:\Users\aman1\AppData\Local\Temp\ipykernel_24032\1618671393.py:8:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  model_train_subset_test3['Date'] =
pd.to_datetime(model_train_subset_test3['Date'])
C:\Users\aman1\AppData\Local\Temp\ipykernel_24032\1618671393.py:9:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  model_train_subset_test3['Date_ordinal'] =
model_train_subset_test3['Date'].apply(lambda x: x.toordinal())
```

[91]:

```
            Date   Close  Date_ordinal  prior_values
2425  2024-04-01  170.03        738977    171.441347
2426  2024-04-02  168.84        738978    171.218360
2427  2024-04-03  169.65        738979    170.995374
2428  2024-04-04  168.82        738980    170.772387
2429  2024-04-05  169.58        738981    170.549401
2430  2024-04-08  168.45        738984    169.880441
2431  2024-04-09  169.67        738985    169.657454
2432  2024-04-10  167.78        738986    169.434468
2433  2024-04-11  175.04        738987    169.211481
2434  2024-04-12  176.55        738988    168.988495
```

[92]:

```
model_train_final3 = model_train_subset3.copy()
model_train_final_test3 = model_train_subset_test3.copy()

mean = model_train_final3['Date_ordinal'].mean()
```

Appendix

```
std = model_train_final3['Date_ordinal'].std()

prior3 = model_train_final3['prior_values']
prior_test3 = model_train_final_test3['prior_values']

model_train_final3['Date_ordinal_normalized'] =␣
 ↪(model_train_final3['Date_ordinal'] - mean)/std
model_train_final_test3['Date_ordinal_normalized'] =␣
 ↪(model_train_final_test3['Date_ordinal'] - mean)/std


X_train4 = model_train_final3['Date_ordinal_normalized'].values
y_train4 = model_train_final3['Close'].values

X_test4 = model_train_final_test3['Date_ordinal_normalized'].values
y_test4 = model_train_final_test3['Close'].values

model_train_final3
```

[92]:

```
            Date     Close  Date_ordinal  prior_values  Date_ordinal_normalized
100   2015-01-02   27.3325        735600     27.374777                -1.730949
101   2015-01-05   26.5625        735603     27.564256                -1.727870
102   2015-01-06   26.5650        735604     27.627415                -1.726843
103   2015-01-07   26.9375        735605     27.690575                -1.725816
104   2015-01-08   27.9725        735606     27.753735                -1.724790
...          ...       ...           ...           ...                      ...
2420  2024-03-22  172.2800        738967    180.036423                 1.725600
2421  2024-03-25  170.8500        738970    179.634989                 1.728680
2422  2024-03-26  169.7100        738971    179.501178                 1.729706
2423  2024-03-27  173.3100        738972    179.367366                 1.730733
2424  2024-03-28  171.4800        738973    179.233555                 1.731760

[2325 rows x 5 columns]
```

[93]:

```
X_obs3 = X_train4
y_obs3 = y_train4

X_obs3 = X_obs3.reshape(-1, 1)
```

[94]:

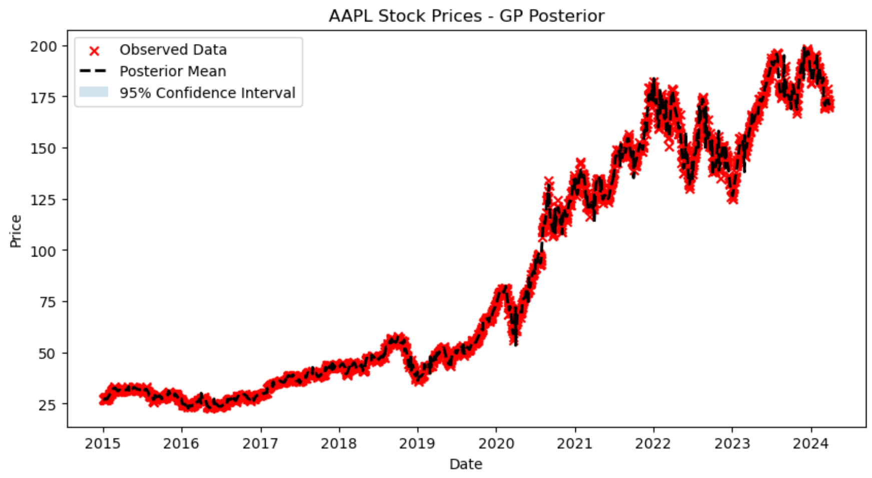
```
#Using the Optimal parameters found via GRID Search and Bayesian method
optimized_length_scale = 0.012250
optimized_variance = 2.0
optimized_constant = optimal_params[2]
optimized_constant2 = optimal_params[3]

noise = 1e-3
num_samples = 25
```

Appendix

```
# Compute covariance matrices for predictions
K_obs = combined_kernel(X_obs3, X_obs3, optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)
K_s = combined_kernel(X_obs3, X_obs3, optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)
K_ss = combined_kernel(X_obs3, X_obs3, optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)
K_obs_inv = np.linalg.inv(K_obs + noise * np.eye(len(X_obs3)))

# Compute the mean and covariance of the posterior
mu_s = prior3 + np.dot(K_s, np.dot(K_obs_inv, (y_obs3 - prior3)))
cov_s = K_ss - np.dot(K_s, np.dot(K_obs_inv, K_s.T))
cov_s_actual = []
mu_s = mu_s.to_numpy()

# Sample from the posterior
L_post = cholesky(cov_s + noise * np.eye(len(X_obs3)), lower=True)
f_post = mu_s.reshape(-1, 1) + np.dot(L_post, np.random.
 ↪normal(size=(len(X_obs3), num_samples)))
```

```
[102]: os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'

# Plot samples from the posterior
plt.figure(figsize=(10, 5))
#plt.plot(model_train_final3['Date'], f_post, label="Posterior Samples")
plt.scatter(model_train_final3['Date'], y_obs3, c='red', marker='x',␣
 ↪label="Observed Data")
plt.plot(model_train_final3['Date'], mu_s, 'k--', lw=2, label="Posterior Mean")
plt.fill_between(model_train_final3['Date'],
                 mu_s.flatten() - 1.96 * np.sqrt(np.diag(cov_s)),
                 mu_s.flatten() + 1.96 * np.sqrt(np.diag(cov_s)),
                 alpha=0.2, label="95% Confidence Interval")
plt.title("AAPL Stock Prices - GP Posterior")
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.show()
```

Appendix



[96]:
```
# Compute the covariance vector between the new date and observed dates
K_new = combined_kernel(X_test4.reshape(-1, 1), X_obs3, optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)
K_new_new = combined_kernel(X_test4.reshape(-1, 1), X_test4.reshape(-1, 1),␣
 ↪optimized_length_scale,␣
 ↪optimized_variance,optimized_constant,optimized_constant2)

# Compute the mean and variance of the posterior at the new date
#mu_new = np.dot(K_new, np.dot(K_obs_inv, y_obs))
mu_new = np.dot(K_new, np.dot(K_obs_inv, (y_obs3 - prior3) )) + prior_test3
cov_new = K_new_new - np.dot(K_new, np.dot(K_obs_inv, K_new.T))
mu_new = list(mu_new)
# Compute the standard deviation
sigma_new = np.sqrt(np.diag(cov_new))

# Print the predicted means and standard deviations
for i in range(len(X_test4)):
    print(f"Predicted mean at new date {X_test4[i]}: {mu_new[i]}")
    print(f"Actual value at new date {X_test4[i]}: {y_test4[i]}")
    print(f"Predicted standard deviation at new date {X_test4[i]}:␣
 ↪{sigma_new[i]}")
```

```
Predicted mean at new date 1.7358659631269218: 170.1620923254727
Actual value at new date 1.7358659631269218: 170.03
Predicted standard deviation at new date 1.7358659631269218: 0.13265287310248483
Predicted mean at new date 1.7368925593079267: 172.50371034833415
Actual value at new date 1.7368925593079267: 168.84
```

Appendix

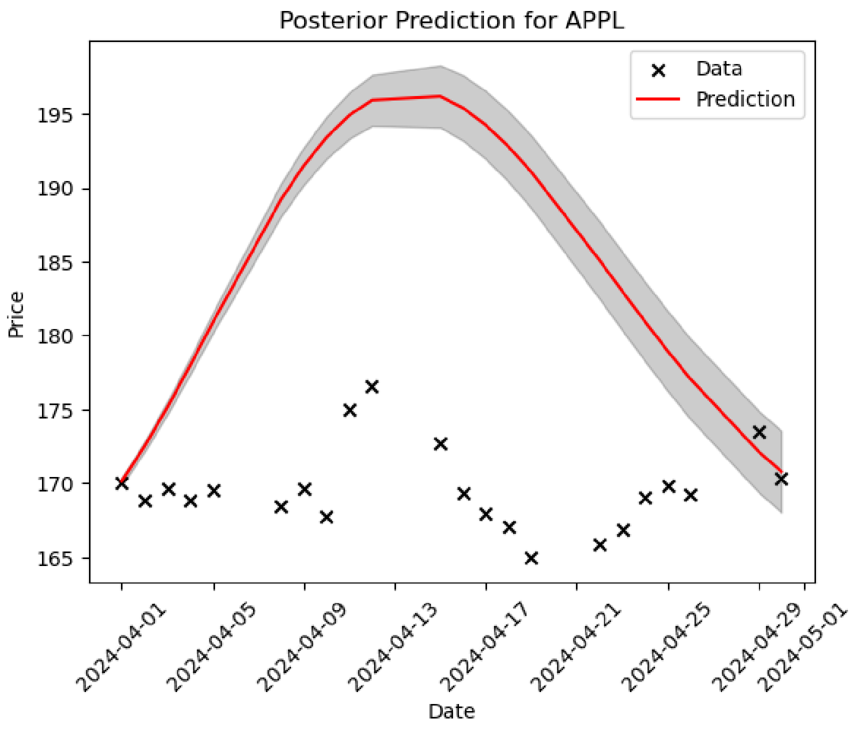
```
Predicted standard deviation at new date 1.7368925593079267: 0.17790963724050546
Predicted mean at new date 1.7379191554889315: 175.1317912101381
Actual value at new date 1.7379191554889315: 169.65
Predicted standard deviation at new date 1.7379191554889315: 0.23019186197704722
Predicted mean at new date 1.7389457516699365: 177.9578592764542
Actual value at new date 1.7389457516699365: 168.82
Predicted standard deviation at new date 1.7389457516699365: 0.28902330481466365
Predicted mean at new date 1.7399723478509415: 180.88184356201396
Actual value at new date 1.7399723478509415: 169.58
Predicted standard deviation at new date 1.7399723478509415: 0.3536959909546755
Predicted mean at new date 1.7430521363939562: 189.1918800801775
Actual value at new date 1.7430521363939562: 168.45
Predicted standard deviation at new date 1.7430521363939562: 0.5727372680263718
Predicted mean at new date 1.744078732574961: 191.48246760474217
Actual value at new date 1.744078732574961: 169.67
Predicted standard deviation at new date 1.744078732574961: 0.6500682752449397
Predicted mean at new date 1.745105328755966: 193.40019094514923
Actual value at new date 1.745105328755966: 167.78
Predicted standard deviation at new date 1.745105328755966: 0.7273898822579681
Predicted mean at new date 1.746131924936971: 194.89055531305803
Actual value at new date 1.746131924936971: 175.04
Predicted standard deviation at new date 1.746131924936971: 0.8034257853604284
Predicted mean at new date 1.7471585211179759: 195.91882813591008
Actual value at new date 1.7471585211179759: 176.55
Predicted standard deviation at new date 1.7471585211179759: 0.8769906378068333
Predicted mean at new date 1.7502383096609906: 196.1790258930714
Actual value at new date 1.7502383096609906: 172.69
Predicted standard deviation at new date 1.7502383096609906: 1.0732409507062852
Predicted mean at new date 1.7512649058419953: 195.39490100951843
Actual value at new date 1.7512649058419953: 169.38
Predicted standard deviation at new date 1.7512649058419953: 1.1282422467585598
Predicted mean at new date 1.7522915020230003: 194.24577070796687
Actual value at new date 1.7522915020230003: 168.0
Predicted standard deviation at new date 1.7522915020230003: 1.1774142660881743
Predicted mean at new date 1.7533180982040053: 192.78821108054694
Actual value at new date 1.7533180982040053: 167.04
Predicted standard deviation at new date 1.7533180982040053: 1.220685319452709
Predicted mean at new date 1.75434469438501: 191.08355329506497
Actual value at new date 1.75434469438501: 165.0
Predicted standard deviation at new date 1.75434469438501: 1.2581619029763556
Predicted mean at new date 1.7574244829280248: 185.10584943749845
Actual value at new date 1.7574244829280248: 165.84
Predicted standard deviation at new date 1.7574244829280248: 1.3389789736049036
Predicted mean at new date 1.7584510791090298: 183.01549486083172
Actual value at new date 1.7584510791090298: 166.9
Predicted standard deviation at new date 1.7584510791090298: 1.3569029363023246
Predicted mean at new date 1.7594776752900347: 180.95668084659735
Actual value at new date 1.7594776752900347: 169.02
```

Appendix

```
Predicted standard deviation at new date 1.7594776752900347: 1.3712023335895993
Predicted mean at new date 1.7605042714710395: 178.96652955279438
Actual value at new date 1.7605042714710395: 169.89
Predicted standard deviation at new date 1.7605042714710395: 1.3824183212984573
Predicted mean at new date 1.7615308676520445: 177.07414977435076
Actual value at new date 1.7615308676520445: 169.3
Predicted standard deviation at new date 1.7615308676520445: 1.391066631638907
Predicted mean at new date 1.7646106561950592: 172.1606167904469
Actual value at new date 1.7646106561950592: 173.5
Predicted standard deviation at new date 1.7646106561950592: 1.4060783741725518
Predicted mean at new date 1.7656372523760642: 170.80300117354645
Actual value at new date 1.7656372523760642: 170.33
Predicted standard deviation at new date 1.7656372523760642: 1.4086501494180257
```

[97]:
```
#Final Predictions
os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'

plt.figure()
plt.scatter(model_train_final_test3['Date'], y_test4, c='k', marker='x',␣
 ↪label='Data')
plt.plot(model_train_final_test3['Date'], mu_new, c='r', label='Prediction')
#plt.fill_between(X_test2.ravel(),y_test2 + y_test2*0.025, y_test2 - y_test2*0.
 ↪025, alpha=0.2, color='k')
plt.fill_between(model_train_final_test3['Date'], mu_new - 1.96*sigma_new,␣
 ↪mu_new + 1.96*sigma_new, alpha=0.2, color='k')
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Posterior Prediction for APPL')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```

Appendix



```
[98]: percentage = []
      total = 0
      for i in range(len(mu_new)):
          percentage.append((y_test4[i]-mu_new[i])/y_test4[i])
          total = total + abs(percentage[i])
      avg_percentage = total/len(percentage)
```

```
[99]: #Percentage error.
      avg_percentage
```

```
[99]: 0.08883277479436935
```

```
[ ]:
```

```
[ ]:
```

27