Non-Gaussian Bayesian Update in High Dimensional Space for Stock Prediction

Matthew Landry

2024

1 Abstract

In the ever-changing field of finance, the development of models for stock price prediction is a critical area of research. This project introduces an approach to this area by utilizing Bayesian updates in a high-dimensional, non-Gaussian framework. This is vital because financial data is typically high-dimensional and has non-Gaussian statistics. The project begins with the application of Kernel Density Estimation to model the initial distribution of stock prices. Subsequent noise reduction is achieved through the use of a modified version of Kalman Filtering. This allows for the enhancement of prediction accuracy. The final model enhancement incorporates a "bucketed" version of Bayes' Theorem by utilizing price intervals to forecast the next-day stock prices. The efficiency of this model will thus be evaluated on its ability to generate reliable predictions that could potentially offer consistent investment returns.

2 Introduction

The realm of financial trading is highly competitive, with market participants competing with incredibly different models to predict stock price movements in attempts to capture economic advantages. Effective data utilization gives these players a huge edge over the competition.

In the domain of financial markets, dozens of mathematical methods have been deployed for constructing predictive trading models. These models range from sophisticated neural network architectures—such as Back Propagation Neural Networks, Radial Basis Function Neural Networks, General Regression Neural Networks, and various forms of Support Vector Machine Regressions including standard and least squares variants [7]—to classical statistical approaches like the Auto-Regressive Integrated Moving Average model [4]. Furthermore, stochastic processes, particularly the Geometric Brownian Motion model, have also been widely applied in market prediction scenarios [2, 4]. It is evident that the application of a wide range of mathematical concepts can be used to significantly enhance trading strategies within the financial markets. Central to the use of these diverse methodologies is the fundamental concept of time series analysis [1], which provides the necessary framework for analyzing, using, and modeling time-series datasets of historical stock prices.

In this space, the Kalman Filter is occasionally employed as a sophisticated tool for estimating the hidden states of model parameters [5]. This technique is periodically adapted for predicting stock market prices [10]. However, given the productiveness and high volatility of financial markets, the standard Kalman Filter offers limited utility. To achieve a substantive competitive advantage, it is necessary to construct modified versions of the classic Kalman Filter tailored to the specific dynamics of financial datasets.

3 Methods

Predicting stock prices with high accuracy remains a formidable challenge due to the complexity and variability of market forces. Because of this, there are numerous potential routes to take in terms of methods.

The Dataset

Historical stock price datasets are necessary for analyzing market trends and predicting future movements. This project focuses primarily on time-series data from major technology firms to capture sector-specific dynamics. These firms include Apple, Google, Microsoft, Amazon, Meta, and many others (see **Appendix** for full list).

Pandas and NumPy

Pandas and NumPy are important and necessary libraries in the Python data science space, facilitating efficient data manipulation and analysis.

Initially, stock symbols and relevant company data were compiled into an Excel spreadsheet, which was later converted into a Pandas DataFrame within the Python program. Using the “yfinance” library, historical closing prices from 2015 to 2023 were selected for these stocks, ensuring the dataset to a uniform time frame (**Code 1** in **Appendix**). This time frame was to prevent biases in model training as the goal is to predict today’s stock prices. By avoiding 2024 data, over-fitting is not a potential problem during the testing phase. In preparation for the modeling phase, price normalization was conducted to standardize the value range across different stocks (**Code 1** in **Appendix**).

Simple Regression

The initial approach in this project involved employing a simple linear regression to establish a foundational model. This model incorporated features such as month, previous day closing price, S&P500 previous day closing price, and day of the week. This regression can be represented by:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n + \epsilon \tag{1}$$

where y is the price prediction, β_i are the coefficients for each feature x_i, and ϵ is the error term.

Preliminary assessments across various stock datasets revealed that simple regression models, while useful for identifying general trends, lacked the complexity needed for accurate stock prediction in a competitive market. This realization brought a transition towards more complex models, though simple regression remained as a potential option for modifying one of the future advanced models.

Correlation Heatmap

Understanding the possible relationships between companies' returns through correlation analysis is critical for aiding predictive accuracy. Initial analysis involves calculating correlation coefficients between these returns to identify predictive relationships.

This project utilized two primary visualization: one of scatter plots depicting returns between pairs (**Figure 4**), and the other as heatmaps to illustrate correlation coefficients (**Figure 5**). These correlations will be later used to determine the weights applied to companies for their prediction power in the prediction models.

Kernel Density Estimation (KDE)

Acknowledging the limitations of linear regression, this project progressed to Kernel Density Estimation (KDE), a non-parametric technique that estimates the Probability Density Function (PDF) of a random variable. KDE is a method that doesn't involve parameters and allows for the estimation of the PDF of a random variable based on kernels as weights. KDE improves on the histogram by using these kernels to mitigate data sparsity and boundary issues. For the kernel density estimate, each data point has a normal kernel. These kernels are then summed and make a kernel density [9].

There are multiple different forms of the kernel function, the most basic and common being the symmetric kernel functions. The mathematical representation of this is given by:

$$K_{sym}(x,t) = \frac{1}{h} K(\frac{x-t}{h}) \tag{2}$$

where $K(t)$ is a symmetric kernel function and h is the smoothing parameter. This project uses the Gaussian kernel function which is given by:

$$K(t) = \frac{1}{2\pi} e^{-t^2/2} \tag{3}$$

Using KDE, it is possible to provide a visual heatmap of the joint distribution of stock prices, which aids in predicting the price of one stock based on the performance of another. For example, see **Figure 1**. In this plot, the normalized closing prices for Google and Apple were utilized. The red, vertical line represents the current price of Apple and the blue, horizontal line represents the current price of Google. The intersection represents a point on the heatmap. The darker parts of the heatmap are where the most intersections for previous closes in the historical data are.

An extension of this approach involves constructing an n-dimensional KDE, where n represents the number of stocks included in the dataset. For example, say the goal is to predict the next day price of Company A. By constructing an n-dimensional KDE with previous day stock prices in comparison to Company A's next day price, it is possible to derive a "hot-zone" of predicted prices for Company A's next day price.

While this "hot-zone" method demonstrates promising potential, to implement the modified Kalman Filtering approach described in the following section, I opted for a methodology that generates conditional density plots based on past prices for a single company. This function predicts a company's price based on historical correlations between the specified company's price and the price of another company. By iterating through a list of n companies, we obtain n conditional density models, which are subsequently utilized in the filtering step.

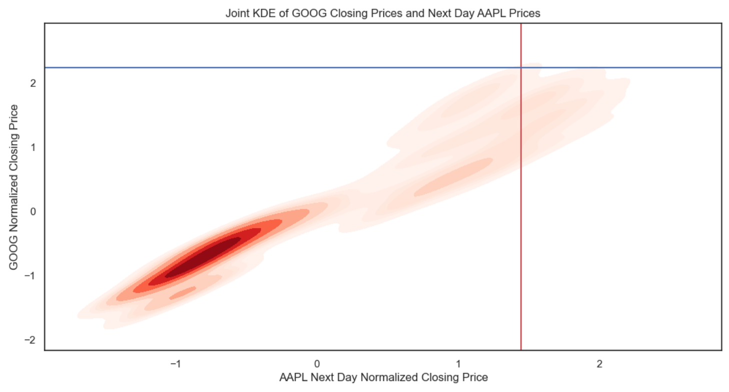

Figure 1: Normalized Closing Price KDE Heatmap

Modified Kalman Filtering

Filtering, a powerful tool for refining predictions in the presence of noisy data, forms the basis of this project's subsequent model enhancement. In general, after initially making a prediction of the true value, a filtering method can be used to limit noise, resulting in a more accurate prediction. Of the possible filters, Kalman Filtering is shown to be one of the best filters for minimizing the variance of the estimation error [8]. The Kalman Filter is thus incredibly useful in this application as the true market value of a stock is often not the actual price.

This project utilizes a modified version of the classic Kalman Filter. Kalman filters are based on linear dynamical systems and they filter based on the time domain [3] (for an example with temperature prediction, see **Figure 2** [6]). This "modified" version is not based on time steps, but instead by company. Thus, the y-axis in the example in **Figure 2** for this filtering process is company, not time.

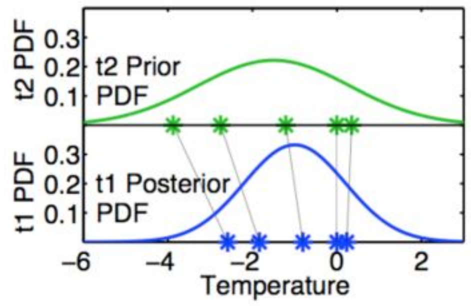

Figure 2: Example of Classic Kalman Filter with Temperature Prediction

The Kalman Filter involves three main variables: prediction from the forecast x^f with variance r^f, observation y with variance r^o, and estimation x^a with variance r^a. The filtering equations are as follows:

$$x^a = x^f + K(y - x^f) \tag{4}$$

$$r^a = (1 - K)r^f \tag{5}$$

where K is the Kalman gain:

$$K = \frac{r^f}{r^o + r^f} \tag{6}$$

Say the goal is to predict the stock price of Company A based on the prices of Company B, Company C, etc. The initial prediction for the filtering is the value that the initial model predicts the price of Company A will be based on Company B. In this case, the initial model was a basic KDE. Thus, the first step creates a resulting price estimation from the past observation prices of Company A and the current price prediction of Company A based on Company B's price. For future steps, the new prediction is the results of the filtering and the filtering model will instead use the next company in the list (ex. Company C) instead of the previous company (ex. Company B). Once every company is utilized to apply its step of filtering, the modified Kalman Filtering process is complete (see **Figure 3**).

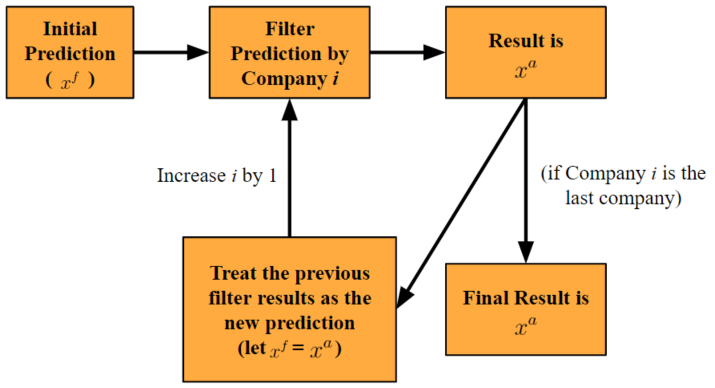
Initial Prediction (x^f)
Filter Prediction by Company i
Result is x^a
Increase i by 1
(if Company i is the last company)
Treat the previous filter results as the new prediction (let $x^f = x^a$)
Final Result is x^a

Figure 3: Modified Kalman Filtering Process

To find the predicted price of Company A based on Company B's stock price, it becomes necessary to find the conditional density distribution. This is simply done through the KDE (**Code 2** in **Appendix**). Through the conditional density, it is possible to use the NumPy function argmax() to calculate the predicted value (**Code 3** in **Appendix**). With this predicted value, it is then possible to apply the modified Kalman Filtering by using **Equations 4**, **5**, and **6** (**Code 4** in **Appendix**).

Upon constructing a newly filtered dataset, we generate conditional density plots for the specified company relative to each company on the list. These plots are then aggregated, with weighting based on the return correlations between the companies, to construct the PDF for the specified company's next-day prices (**Code 5** in **Appendix**). This PDF serves as the primary resulting model from the modified Kalman Filtering process.

Bucketed Bayes

To further refine the stock price prediction model, Bayes' Theorem was integrated to compute conditional probabilities using discretized stock prices. This adaptation involved rounding historical stock prices to the nearest dollar. By doing so, this segments the continuous price data into discrete categories, or "buckets". Through this, Bayes' Formula was applied to estimate the probability of a specific company's price, based on the price of other companies (**Code**

6 in **Appendix**). Bayes' Formula is given by:

$$Pr(X|Y) = \frac{Pr(Y|X) * Pr(X)}{Pr(Y)} \tag{7}$$

In this application, X denotes the event that Company A's stock price on the next day is x and Y denotes the event that Company B's stock price is currently price y. The theorem thus applies the calculation of the probability of X given Y. This is the likelihood of a specific next-day price for Company A, conditioned on the observed current-day price of Company B.

This probabilistic model assigns each "bucket" a probability of occurring for Company A. This probability reflects the historical frequency and relational dependency of the stock prices between companies. These probabilities will later be used to create a a Bayesian probability distribution (**Code 7** in **Appendix**). Weights determined by previously discussed correlations metrics between companies will be used to adjust this probability distribution, giving more highly correlated companies more weight in the distribution.

The Final Model

The integration of the "Bucketed" Bayesian approach alongside the filtered distributions from the initial Kernal Density Estimation yields the final predictive model. This hybrid model leverages the strengths of Bayesian statistics and advanced filtering techniques in order to provide a robust framework for stock prediction. The combined model not only accommodates the complexities of the financial datasets, but also improves forecast precision by effectively managing the inherent noise.

In the final configuration of the model, a weighting scheme is employed where the KDE derived from the discretized Bayesian method is assigned a weight of 0.325, whereas the KDE obtained via the modified Kalman Filtering technique receives a weight of 0.675. The determination of these specific weighting coefficients was the result of an extensive series of daily evaluations aimed at optimizing the accuracy of predictions for the subsequent day. This approach ensured that the final model leverages the strengths of both underlying estimators to enhance overall predictive performance.

Ultimately, the model generates a PDF representing the forecasted next-day price of a specified stock. By conducting an integration from the current stock price to infinity on this PDF (as seen in **Equation 8**), the probability that the model predicts the stock price will increase can be assessed.

$$\int_{c}^{\infty} f(x)\, dx = Pr(increase) \tag{8}$$

where

$$\begin{aligned} c &= \text{current price} \\ f(x) &= \text{final model function} \\ Pr(increase) &= \text{price increase probability} \end{aligned}$$

This methodology provides investors with a statistically grounded basis for making informed investment decisions, dependent on the predictions furnished by this final model.

4 Results

Correlations Results

The determination of filtering sequences within the Kalman Filtering framework is dependent on the results derived from the correlation coefficients between stocks. Specifically, the filtering process is initiated with stocks that exhibit high correlation to the stock of the company whose next day price is the one we wish to predict. The filtering process then moves progressively towards those with weaker correlations. Additionally, the weights assigned during the Bayesian probability distribution are determined from the correlations calculated here.

The correlation analysis utilized scatter plots and heatmaps to visually represent the strength of relationships between stock returns. For the purpose of presentation, **Figure 4** and **Figure 5** illustrate these relationships for only 5 companies each. The complete correlation plots are provided in the **Appendix**.

Initial KDE Estimation

The initial KDE estimation is executed via the predefined function *predict_price*. This function estimates the next day's stock price of a targeted company based on the historical price data of another company in the dataset. By iterating through each company and using the function, a list is constructed of predicted values. These individual predictions are then subsequently aggregated by giving weighted averages to each prediction based on the correlation. This formulates a consolidated prediction for the next day's closing price of the targeted company. This approach leverages the full spectrum of available corporate data to minimize prediction error and enhance the model's reliability.

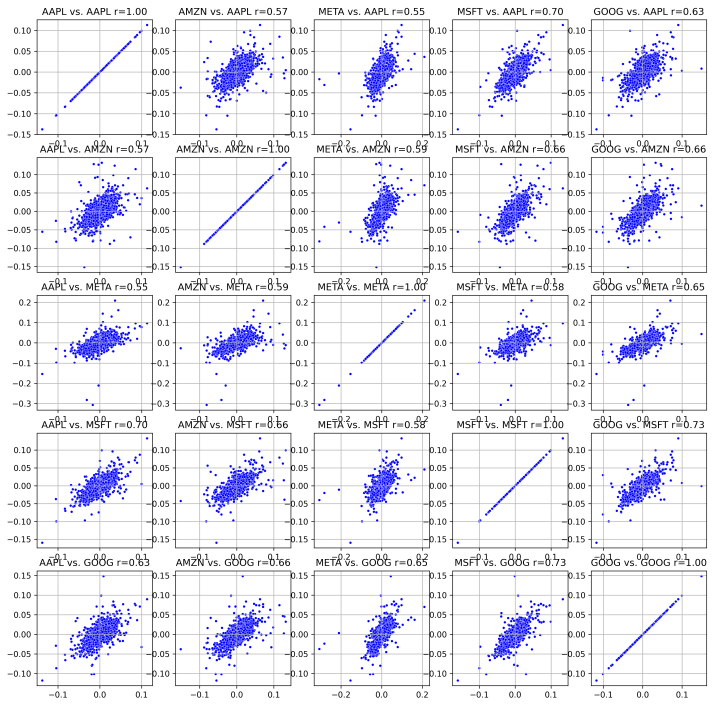

Figure 4: Returns Correlation Scatterplots

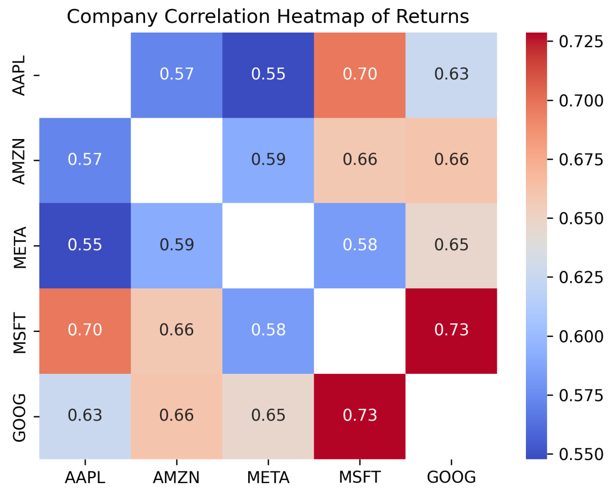

Figure 5: Returns Correlation Heatmap

Modified Kalman Filtering Prediction

As discussed previously, the modified Kalman filtering process further refines these predictions by limiting statistical noise inherent in financial datasets, as illustrated in **Figure 6**. This enhancement is depicted through a tighter concentration of data points in the heatmap, therefore implying more precise predictions. Utilizing the same *predict_price* function, this process adjusts the initial KDE predictions based on the newly filtered data. The re-calibrated predictions through this modified Kalman Filtering not only provide a sharper estimation of stock prices, but also improve the robustness of the model against volatile market conditions.

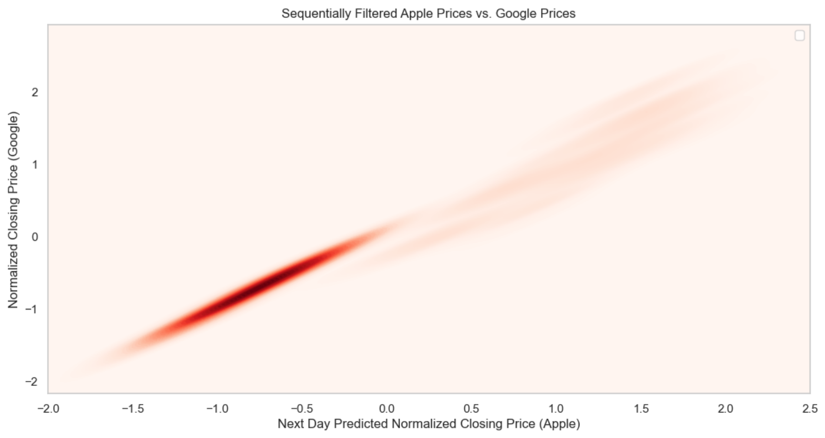

Figure 6: Normalized Closing Price KDE Heatmap with Modified Kalman Filtering Dataset

The PDF of filtered next-day prices has demonstrated a behavior of predicting the occurrence of some unlikely prices with large probability. As illustrated in **Figure 7**, there is a local maximum around a stock price of 50. In this instance, there is a predicted probability of AAPL prices that, in actuality, are highly improbable. A potential explanation for this phenomenon is provided in the **Discussion** section.

From this, the assumption may be that the model is unusable due to its inaccuracy in predicting exact next-day prices. However, it retains significant potential applications. Although the model's precision in forecasting the exact next-day price is limited, its ability to predict whether the next day's closing price will increase or decrease relative to the current day's closing price remains usable. Therefore, by integrating this model with our "Bucketed" Bayesian model, we achieve a final model capable of accurately predicting the directional movement of a stock's price by the following day's close.

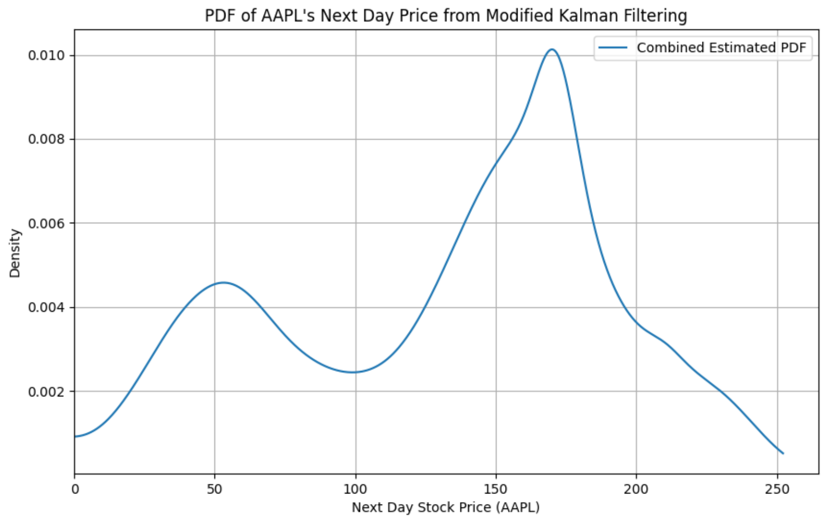

Figure 7: Filtered PDF of Next-Day Prices from Modifed Kalman Filtering Process

"Bucketed" Bayes Probability Distribution

The resulting "Bucketed" Bayes' PDF as seen in **Figure 8** was as expected. The maximum of the PDF is around the 155-165 range. By calculating the mean and median of the AAPL prices, we find that the mean is about 155.5 and the median is about 152.1. This means the PDF is almost a uniform distribution centered at the mean/median of the data.

Similarly to the PDF obtained through the modified Kalman Filtering process, the resulting PDF from the "bucketed" Bayes process also exhibits certain limitations. Although this near-uniformly distributed PDF is not particularly useful in isolation, it contributes to the construction of the final model. This was the intended objective of this approach. By constructing this model through Bayes' Formula, we can implement a slight modification to enhance the accuracy of our final model. This adjustment inherently reduces the weight assigned to extreme values in the previously resulting plot, thereby increasing the weight attributed to more probable prices.

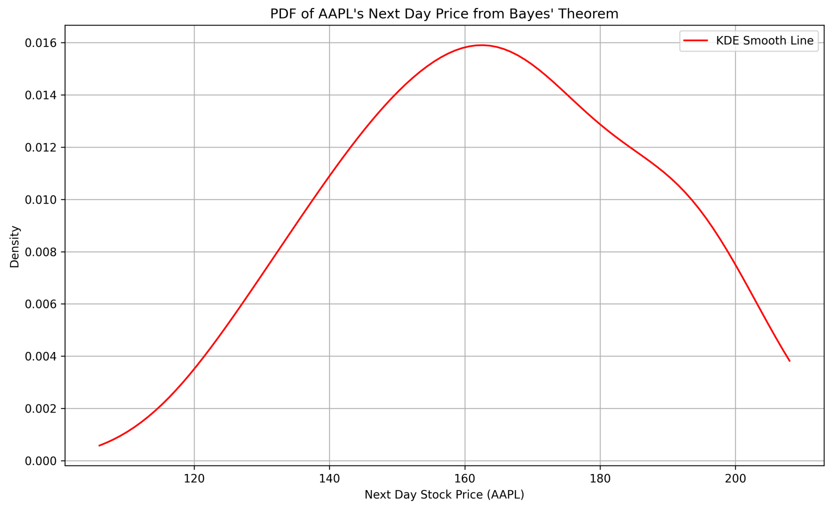

Figure 8: “Bucketed” Bayes’ Probability Distribution KDE

The Final Model

The final model, illustrated in **Figure 9**, presents the predicted probability distribution of next-day prices for AAPL. This distribution varies based on the specific day and the current prices of the stocks in the dataset. Although the model is not flawless, it provides valuable predictive insights into the likely direction of the next day’s stock price.

The most important aspect of this model lies in the information it offers to potential investors. While stock prices generally lack obvious patterns, this model delivers an informed prediction derived from the historical price data of the specified stock and its relationship to other stocks in terms of next-day prices. Making an informed investment decision involves numerous factors, and this model may contribute meaningfully by being one of these factors.

Over a week-long evaluation period, the model demonstrated the capability to accurately predict price movements on four out of the five trading days. However, this is a limited sample size, therefore requiring more comprehensive testing to robustly assess the model’s true predictive accuracy. Preliminary baseline tests yielded encouraging outcomes, yet further investigation is essential to determine whether these results were statistically significant or merely attributable to random chance.

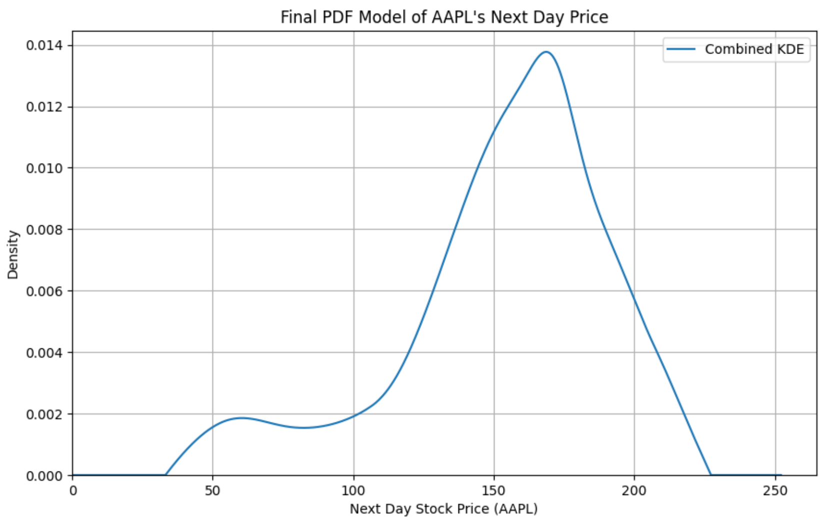

Figure 9: The Final Resulting PDF for next day AAPL Price

5 Discussion

Stock price prediction does not follow a singular, "correct" method. The primary metric for evaluating the accuracy and utility of a predictive model is its ability to consistently outperform the average market investor, and therefore generate consistent returns.

Upon observing the results from both primary models—the Modified Kalman Filter KDE and the "Bucketed" Bayes' KDE—it became evident that a fundamental flaw exists in the overall process. A critical aspect of the project is predicting the next-day price of a specified stock based on the current prices of other stocks. The inherent flaw in this approach stems from the limited number of samples where a large number of stocks historically exhibit values similar to their current prices. Consequently, the final model encounters difficulties in predicting an exact price.

For example, consider the recent prices of Apple's (AAPL) stock. During the week of May 13, 2024, AAPL's closing price consistently remained above 186. Historically, instances of AAPL's price exceeding 186 are rare, resulting in the model having insufficient data to make accurate predictions. From this, it is possible to determine that a more effective approach would involve analyzing the quantities of price increases and decreases rather than the actual prices themselves. Since these values fall within a narrower range, they provide a larger and more usable dataset. This method would likely enhance the model's accuracy, leading to more reliable predictions.

This alternative approach that utilizes changes in stock prices, rather than the absolute prices, could substantially augment the effectiveness of the “bucketed” Bayes’ PDF. As previously noted, this model was essentially a uniform distribution of the specified stock’s data. Transitioning to a model based on price changes could yield a significantly more informative and functional model. This new model would have served as a better modification to the final model.

Ultimately, the final model is reliable in certain scenarios. During the testing phase, a significant challenge arose as a majority of the companies within the dataset exhibited stock prices substantially higher than their historical averages. This anomaly led the model to exhibit a bias towards predicting lower prices, a tendency reinforced by the unusually high prices in recent stock values. Such predictions logically make sense over longer time-frames where price corrections are expected; however, they prove less accurate on a next-day pricing scale. Under more typical market conditions, where stock prices align closely with historical data, the model’s performance would likely improve due to its reliance on representative data. Despite these challenges, the mathematical methods of the model have demonstrated their reliability through unit testing. The primary limitations have arisen from the application of these methods, which have produced less ideal predictions. As mentioned previously, the primary reason for this is due to using true prices instead of price changes. Nonetheless, the structural integrity of the model remains satisfactory, with plenty of room for improvement.

6 Conclusion

Although the model did not achieve the desired performance, the mathematical methodologies employed in its construction are fundamentally correctly implemented. To further refine and enhance the model, the next phase of development would involve restarting the process with data compilation and modification. Specifically, constructing a dataset based on price changes rather than absolute prices—and subsequently adjusting the model to accommodate this change—could substantially increase its predictive capability in the stock market. This approach offers a promising avenue in the future for developing a model with robust predictive power.

While the aspiration to construct a model with exceptional predictive accuracy is commendable, achieving such a feat in practice proves to be exceedingly challenging due to the inherently unpredictable nature of financial markets. These markets lack consistent patterns, therefore leading to difficulty in prediction. Many entities in the financial sector, including market makers, hedge funds, and other investment companies, frequently develop models that ultimately falter under tests in real-market conditions. An integral component of model development involves a cyclical process of methodological evaluation and subsequent model refinement. The primary objective of this project was to immerse in this iterative process, and in the end, that is exactly what I achieved.

Acknowledgements

Main Project Advisor: Yoonsang Lee
Project Mentor: Peter Mucha

References

[1] Jonathan D Cryer and Kung-Sik Chan. *Time series analysis: with applications in R*, volume 2. Springer, 2008.

[2] Abdelmoula Dmouj. Stock price modelling: Theory and practice. *Masters Degree Thesis, Vrije Universiteit*, 2006.

[3] Ali Hirsa. *Computational Methods in Finance*. CRC Press, 2016.

[4] Mohammad Rafiqul Islam and Nguyet Nguyen. Comparison of financial models for stock price prediction. *Journal of Risk and Financial Management*, 13(8):181, 2020.

[5] Alireza Javaheri, Delphine Lautier, and Alain Galli. Filtering in finance. *Wilmott*, 3:67–83, 2003.

[6] Yoonsang Lee. Math 106 topics in applied mathematics: Data-driven uncertainty quantification, 2019. (Reference refers to material covered in MATH 106 at Dartmouth College).

[7] Yanran Ma, Nan Chen, and Han Lv. Back propagation mathematical model for stock price prediction. *Applied Mathematics and Nonlinear Sciences*, 7(1):165–174, 2022.

[8] Dan Simon. Kalman filtering. *Embedded systems programming*, 14(6):72–79, 2001.

[9] Stanisław Weglarczyk. Kernel density estimation and its application. In *ITM web of conferences*, volume 23. EDP Sciences, 2018.

[10] Curt Wells. *The Kalman filter in finance*, volume 32. Springer Science & Business Media, 2013.

Appendix

Full Company List

The full list of the companies used for stock time-series datasets is as follows:

- Apple (AAPL)
- Amazon (AMZN)
- Meta (META)
- Microsoft (MSFT)
- Google (GOOG)
- NVIDIA (NVDA)
- TSMC (TSM)
- Tesla (TSLA)
- Tencent (TCEHY)
- Oracle (ORCL)
- Salesforce (CRM)
- Netflix (NFLX)
- AMD (AMD)
- SAP (SAP)
- Adobe (ADBE)
- Cisco (CSCO)
- Texas Instruments (TXN)
- IBM (IBM)
- Uber (UBER)
- Intel (INTC)
- Dell (DELL)
- Sony (SONY)
- Airbnb (ABNB)
- Workday (WDAY)
- Nintendo (NTDOY)
- Spotify (SPOT)

Full Correlation Heatmap

The full company correlation heatmap is given by **Figure 10**. This full heatmap gives a sense of the general returns correlations between companies, but it is difficult to determine the actual specific value.

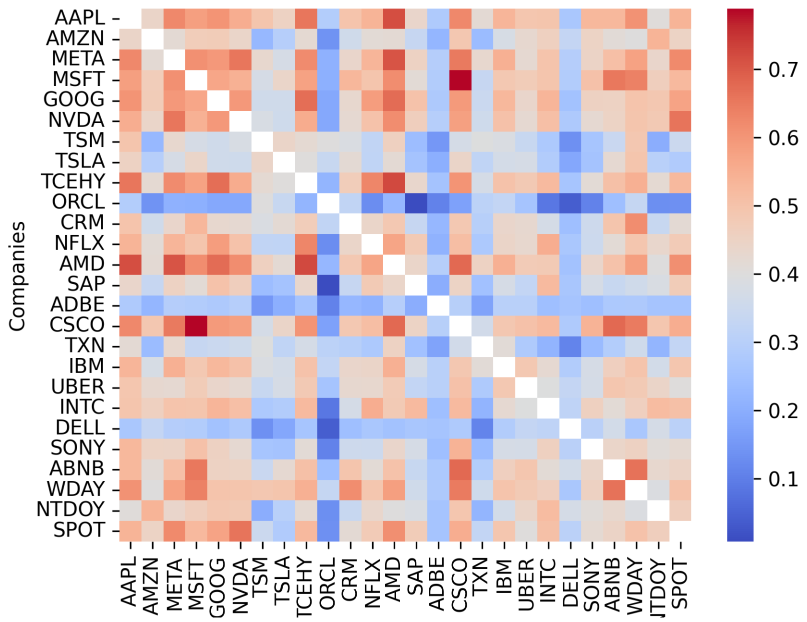

Figure 10: Full Correlation Heatmap

Code

The following figures are of the implemented code sections discussed in the report:

Code 1: Basic Data Manipulation

```
def get_and_format_data():
    company_info = pd.read_excel("../data/
        CompaniesAndTickers.xlsx")
    tickers = company_info["Ticker"].tolist()
    start, end = '2015-01-01', '2024-01-01'
    all_data = yf.download(tickers, start, end)["Close"]

    current_normalized_prices = {}
    company_initial_means_and_stds = {}

    corr_df = pd.read_csv("../outputs/
        CompanyCorrelationDataFrame.csv", index_col=0)

    for ticker in tickers:
        # remove tickers with low correlation for returns
```

```python
        if corr_df.loc[stock_to_pred, ticker] <= CORR_THRESHOLD:
            tickers.remove(ticker)

    for ticker in tickers:
        all_data[str(ticker) + "␣next␣day␣price"] = all_data[ticker].shift(-1)
        company_initial_means_and_stds[ticker] = (all_data[ticker].mean(), all_data[ticker].std())

        current_price = yf.Ticker(ticker).info["currentPrice"]
        current_normalized_prices[ticker] = (current_price - all_data[ticker].mean()) / all_data[ticker].std()

    all_data = all_data[:-1]  # drop the last row because of NA
    all_data = all_data.dropna()
    normalized_data = (all_data - all_data.mean()) / all_data.std()  # normalize the data

    return normalized_data, tickers, current_normalized_prices, company_initial_means_and_stds
```

Code 2: Calculate Conditional Density Distribution

```python
def conditional_distribution(given_price, all_data, x_grid, given_ticker, data_from_pred_ticker):
    # given_price = given_ticker's current price
    # all_data = stock data
    # x_grid = x values for KDE
    # given_ticker = ticker we know current day price of
    # data_from_pred_ticker = stock data of pred ticker that we make our prediction based on

    values = np.vstack([data_from_pred_ticker, all_data[given_ticker]])
    kde = gaussian_kde(values)
    conditional_density = kde([x_grid, np.full_like(x_grid, given_price)])
    conditional_density /= np.trapz(conditional_density, x_grid)  # Normalize

    return conditional_density
```

Code 3: Calculate the Predicted Stock Price

```
def predict_price(given_ticker, pred_ticker,
    given_stock_price_standardized, data_from_pred_ticker)
    :
    # given_ticker = ticker we know current time price of
    # pred_ticker = ticker of next day price we wish to
        predict
    # given_stock_price_standardized = given_ticker's
        current price (standardized)
    # data_from_pred_ticker = stock data of pred ticker
        that we make our prediction based on

    x_grid = np.linspace(-3, 3, 1000)  # Standardized
        price range
    conditional_density = conditional_distribution(
        given_stock_price_standardized, all_data, x_grid,
        given_ticker, data_from_pred_ticker)
    predicted_price = x_grid[np.argmax(
        conditional_density)]
    return predicted_price
```

Code 4: Modified Kalman Filter

```
def kalman_filter_modified(next_day_prices, tickers,
    primary_ticker, all_data):
    # next_day_prices = next day prices of the stock we'
        re trying to predict
    # tickers = list of tickers
    # primary_ticker = stock we wish to predict
    # all_data = list of all time-serires stock data

    Ro = next_day_prices.var()  # Observation noise
        variance
    Rf = 1  # Initial forecast error variance is assumed
        to be 1 because of standardization

    xa = np.zeros(len(next_day_prices))  # Array for
        filtered estimates
    tickers.remove(primary_ticker)

    filtered_prices = next_day_prices
    y = next_day_prices

    for i, ticker in enumerate(tickers):
        print(f"STATUS: Currently filtering {ticker}")
        if ticker != primary_ticker:
            for j, ticker_price_in_spot_j in enumerate(
                next_day_prices):
                # First, get prediction xf for
                    primary_ticker
                xf = predict_price(ticker, primary_ticker
                    , all_data[ticker][j], filtered_prices
                    )
                kalman_gain = Rf / (Rf + Ro)

                # our estimation xa is equal to:
                xa[j] = xf + kalman_gain * (y[j] - xf)
                #print("DEBUGGING: value multiplied by
                    Kalman Gain: " + str(y[j] - xf))
                #print("DEBUGGING: Kalman Gain: " + str(
                    kalman_gain))
                Rf = (1 - kalman_gain) * Rf

                # continue until all prices have an
                    estimation

            xa = (xa - np.mean(xa)) / np.std(xa)  #
                normalize
            filtered_prices = xa

    # Result is a normalized list of estimation prices
        for AAPL
    normalized_filtered_prices = (filtered_prices - np.
        mean(filtered_prices)) / np.std(filtered_prices)
    return pd.Series(normalized_filtered_prices, index=
        next_day_prices.index)
```

Code 5: Model Construction from Filtered Data

```
filtered_prices = kalman_filter_modified(all_data[f"{
    stock_to_pred} next day price"], tickers,
    stock_to_pred, all_data)
mean_std_dict = company_initial_means_and_stds
combined_pdf, x_grid = create_combined_pdf(tickers,
    stock_to_pred, current_normalized_prices,
    filtered_prices)
```

```
final_kde, final_x_grid = plot_combined_pdf(combined_pdf,
    x_grid, stock_to_pred, mean_std_dict)

def create_combined_pdf(tickers, pred_ticker,
    current_normalized_prices, filtered_data):
    x_grid = np.linspace(-3, 3, 1000)  # Standardized
        price range
    combined_density = np.zeros_like(x_grid)

    weights = {}
    corr_df = pd.read_csv("../outputs/
        CompanyCorrelationDataFrame.csv", index_col=0)

    for ticker in corr_df.index:
        if ticker != pred_ticker:
            weights[ticker] = corr_df.loc[ticker,
                pred_ticker]

    total_weight = sum(weights.values())
    normalized_weights = {ticker: weight / total_weight
        for ticker, weight in weights.items()}
    weights = normalized_weights

    for ticker in tickers:
        if ticker != pred_ticker:
            given_stock_price_standardized =
                current_normalized_prices[ticker]
            conditional_density =
                conditional_distribution(
                given_stock_price_standardized, all_data,
                x_grid, ticker, filtered_data)
            combined_density += weights[ticker] *
                conditional_density

    combined_density /= np.trapz(combined_density, x_grid
        )  # Normalize the combined density

    return combined_density, x_grid

def plot_combined_pdf(combined_density, x_grid,
    primary_ticker, mean_std_dict):
    mean, std = mean_std_dict[primary_ticker]
    true_price_grid = x_grid * std + mean

    # Filter out values where true_price_grid < 0 (
        because this is not a possible price)
    mask = true_price_grid >= 0
    true_price_grid = true_price_grid[mask]
    combined_density = combined_density[mask]

    # Set any negative values in combined_density to 0
    combined_density[combined_density < 0] = 0

    combined_density /= np.trapz(combined_density,
        true_price_grid)

    plt.figure(figsize=(10, 6))
    plt.plot(true_price_grid, combined_density, label='
        Combined Estimated PDF')
    plt.title(f"PDF of {primary_ticker}'s Next Day Price
        from Modified Kalman Filtering")
    plt.xlabel(f"Next Day Stock Price ({primary_ticker})"
        )
    plt.ylabel("Density")
    plt.legend()
    plt.xlim(left=0)
    plt.grid(True)
    plt.savefig("../outputs/kalmanKDE")
    plt.close()

    return combined_density, true_price_grid
```

Code 6: “Bucketed” Bayes’ Formula Implementation

```
def counts_for_buckets(rounded=0):
    all_data, tickers = get_and_format_data()

    # round values
    all_data = all_data.round(rounded)

    counts = []
    for ticker in tickers:
        counts.append(all_data[ticker].value_counts())

    counts_data = pd.concat(counts, axis=1, sort=True).
        fillna(0).astype(int)

    return counts_data


def bayes_formula(ticker_to_predict, known_ticker,
```

```python
    next_day_comparison_df, count):
    known_ticker_price = round(yf.Ticker(known_ticker).info["currentPrice"])

    # Pr(A): Probability of ticker_to_predict's next day price being count
    total_counts = next_day_comparison_df[str(ticker_to_predict) + "_next_day_price"].value_counts()
    Pr_A = total_counts.get(count, 0) / total_counts.sum()

    # Pr(B): Probability of known_ticker's current price occurring
    total_counts_B = next_day_comparison_df[known_ticker].value_counts()

    if known_ticker_price > total_counts_B.index.max():
        known_ticker_price = total_counts_B.index.max()
    if known_ticker_price < total_counts_B.index.min():
        known_ticker_price = total_counts_B.index.min()

    Pr_B = total_counts_B.get(known_ticker_price, 0) / total_counts_B.sum()


    # Pr(B|A): Probability that known_ticker's current price is known_ticker_price given A
    conditional_df = next_day_comparison_df[next_day_comparison_df[str(ticker_to_predict) + "_next_day_price"] == count]
    conditional_counts = conditional_df[known_ticker].value_counts()

    if conditional_counts.sum() != 0:
        Pr_B_given_A = conditional_counts.get(known_ticker_price, 0) / conditional_counts.sum()
    else:
        Pr_B_given_A = 0


    # Bayes: P(A|B) = (P(B|A) * P(A)) / P(B)
    if Pr_B != 0:
        Pr_A_given_B = (Pr_B_given_A * Pr_A / Pr_B)
    else:
            Pr_A_given_B = 0


    return Pr_A_given_B


def bucketed_bayes(ticker_to_predict):
    all_data, tickers = get_and_format_data()

    counts_data = counts_for_buckets()

    company_buckets = {}

    next_day_comparison_df = all_data.round(0)
    next_day_comparison_df[str(ticker_to_predict) + "_next_day_price"] = next_day_comparison_df[ticker_to_predict].shift(-1)

    for ticker in tickers:
        if ticker != ticker_to_predict:

            print("BAYES: " + str(ticker))

            company_buckets[ticker] = {}

            for count in counts_data.index:

                Pr_A_given_B = bayes_formula(ticker_to_predict, ticker, next_day_comparison_df, count)

                if Pr_A_given_B != 0:
                    company_buckets[ticker][count] = Pr_A_given_B

    return company_buckets
```

Code 7: "Bucketed" Bayes' Formula Probability Distribution

```
def plot_bayes_distributions(data, stock_to_pred):
    # Weights for each company
    weights = {"AMZN": 1, "META": 1, "MSFT": 1, "GOOG": 1}  # Weights section not yet implemented

    # Combined weighted probabilities
```

```python
    combined_probabilities = {}

    # Aggregate the probabilities with weights
    for company, prices_prob in data.items():
        company_weight = weights.get(company, 1)  # Default weight is 1 if not specified

        for price, probability in prices_prob.items():
            weighted_prob = probability * company_weight

            if price in combined_probabilities:
                combined_probabilities[price] += weighted_prob
            else:
                combined_probabilities[price] = weighted_prob

    # Normalize the probabilities to ensure they sum to 1
    total_prob = sum(combined_probabilities.values())
    for price in combined_probabilities:
        combined_probabilities[price] /= total_prob

    # Data for the plot
    prices = np.array(list(combined_probabilities.keys()))
    probabilities = np.array(list(combined_probabilities.values()))

    # Create KDE of the combined probabilities
    kde = gaussian_kde(prices, weights=probabilities)
    kde_x = np.linspace(prices.min() - 10, prices.max() + 10, 100)
    kde_y = kde(kde_x)

    plt.figure(figsize=(12, 7))
    plt.bar(prices, probabilities, width=1.0, color="blue", label="Probability Bars")
    plt.plot(kde_x, kde_y, color="red", label="KDE Smooth Line")
    plt.xlabel(str(stock_to_pred) + " Stock Price")
    plt.ylabel("Probability")
    plt.title("Weighted and Smoothed Probability Distribution of " + str(stock_to_pred) + " Stock Prices")
    plt.xticks(np.arange(min(prices), max(prices) + 1, 2.0))
    plt.legend()
    plt.grid(True)
    plt.show()
```

Code 8: Final Model Construction

```
kalman_model, x_grid = next_day_kde_kalman_model(
    STOCK_TO_PRED)
bucketed_bayes_model, bucketed_bayes_x =
    get_bucketed_bayes_model(STOCK_TO_PRED)

weight1 = 0.325
weight2 = 1 - weight1

# Allows for combination of models
interpolator = interp1d(bucketed_bayes_x,
    bucketed_bayes_model, kind='linear', fill_value="
    extrapolate")
bucketed_bayes_model_resampled = interpolator(x_grid)

# Combine the KDEs
combined_kde_values = weight1 *
    bucketed_bayes_model_resampled + weight2 *
    kalman_model

# Filter out values where true_price_grid < 0 (because
    this is not a possible price)
mask = x_grid >= 0
x_grid = x_grid[mask]
combined_density = combined_kde_values[mask]

# Set any negative values in combined_density to 0
combined_kde_values[combined_kde_values < 0] = 0

# Normalize
combined_kde_values /= np.trapz(combined_kde_values,
    x_grid)

# INTEGRATION
current_price = CURRENT_PRICE
upper_limit = math.inf

# Extract the indices where x is between a and b
indices = (x_grid >= current_price) & (x_grid <=
    upper_limit)
```

```
x_subset = x_grid[indices]
y_subset = combined_kde_values[indices]

area = simps(y_subset, x_subset)
print(f"The predicted probability that {STOCK_TO_PRED}'s price will increase from {current_price} tomorrow is {area}")

# Plot the combined KDE
plt.figure(figsize=(10, 6))
plt.plot(x_grid, combined_kde_values, label='Combined KDE')
plt.title(f"Final PDF Model of {STOCK_TO_PRED}'s Next Day Price")
plt.xlabel(f"Next Day Stock Price ({STOCK_TO_PRED})")
plt.ylabel("Density")
plt.text(20, 0.006, f"Next day price increase prob: {area}", fontsize=12)
plt.legend()
plt.xlim(left=0)
plt.ylim(bottom=0)
plt.grid(True)
plt.savefig(f"../outputs/finalModel_{weight}.png")
plt.close()
```