Modeling Prediction Markets: Optimizing Modified Sigmoid and LSMR Functions on Polymarket Data

Conrad Voigt

December 2024

Abstract

This study investigates the predictive accuracy of cost functions in prediction markets, with a focus on Polymarket. Prediction markets aggregate diverse participant beliefs into market prices, offering a probabilistic forecast of future events. Using hourly price data from 371 Polymarket binary outcome markets, the study implements and evaluates three models: the Modified Logarithmic Market Scoring Rule (LMSR), the Modified Sigmoid function, and Exponential Smoothing. Each model is optimized using the L-BFGS-B algorithm, minimizing residual sum of squares while ensuring parameter constraints. The results reveal that the Modified Sigmoid function outperforms the other models, achieving the lowest prediction error (MSE = 0.000998) and highest explanatory power (R^2 = 0.8090), demonstrating its suitability for capturing the non-linear dynamics of market prices. Exponential Smoothing provides a robust benchmark with competitive accuracy (R^2 = 0.7696), while the Modified LMSR highlights the dynamic role of liquidity but lags in precision (R^2 = 0.3095). Future work will bridge the theoretical sandbox model to real-world market data, refine the differentiation of data subsets, and explore advanced smoothing and predictive techniques to enhance the robustness and generalizability of the models. This research contributes to a deeper understanding of prediction market dynamics and offers practical insights for improving price forecasting in financial markets.

1 Introduction

Prediction markets are platforms that aggregate participant beliefs into probabilistic forecasts of future events. By trading contracts tied to event outcomes, these markets generate prices that reflect the likelihood of an event occurring. Prediction markets often outperform traditional forecasting methods, providing a dynamic and real-time synthesis of collective information.

Blockchain-based platforms, like Polymarket, have enhanced prediction markets by offering transparency, immutability, and accessibility. Polymarket gained significant attention during the 2024 U.S. presidential election, with a trading volume exceeding 3.5 billion dollars globally. Its detailed data on binary outcome markets provides a valuable foundation for studying market efficiency and predictive accuracy.

This study uses Polymarket data to evaluate three models: the Modified Logarithmic Market Scoring Rule (LMSR), Modified Sigmoid function, and Exponential Smoothing. A sandbox framework is developed to simulate market dynamics and analyze how these models aggregate information and respond to new data. The performance of the models is assessed using predictive metrics such as Mean Squared Error (MSE) and the Coefficient of Determination (R2R2).

The research aims to identify effective modeling approaches for prediction markets and improve understanding of how market prices reflect aggregated beliefs. By combining theoretical analysis with empirical evaluation, this study provides insights into optimizing prediction market frameworks for improved forecasting accuracy.

2 Prediction Markets and Polymarket

Prediction markets are platforms where participants trade contracts tied to the outcomes of future events. These contracts allow individuals to buy and sell positions that reflect their belief in the likelihood of specific events occurring. The prices of these contracts, normalized between $0 and 1$, represent the market's collective consensus on the probability of an event. For example, a contract priced at 0.75 dollars implies a 75 percent perceived likelihood of the event happening. Prediction markets span a wide range of topics, with sports, popular culture, and politics being the most prominent.

Polymarket, a blockchain-based prediction market, exemplifies the platform's potential. It allows users to trade on binary outcomes, such as "Will candidate X win the election?" or "Will policy Y be enacted by a specific date?" During the 2024 U.S. presidential election, Polymarket recorded over 3.5 billion dollars in global betting volume on contracts related to the election outcome alone. This volume underscores the growing importance of prediction markets in aggregating public sentiment and synthesizing information across diverse participants.

Polymarket operates on a peer-to-peer blockchain framework, ensuring trans-

parency, decentralization, and immutability. This design eliminates the need for a centralized authority, as trades are executed and recorded securely on the blockchain. The platform's data, which includes detailed price movements and trading volumes for a variety of binary outcomes, is publicly accessible. This openness makes Polymarket a valuable resource for researchers and analysts interested in exploring market behavior and efficiency.

The data used in this study was sourced from Polymarket, capturing price trajectories for numerous binary outcome contracts. These data reflect the dynamic interplay of market participants' beliefs and offer a robust foundation for evaluating the efficiency and predictive accuracy of different cost functions in prediction markets. By analyzing Polymarket's data, this research aims to provide insights into how market-based mechanisms aggregate information and process collective beliefs into actionable probabilistic forecasts.

3 Literature Review

The literature on prediction markets encompasses a variety of modeling techniques, cost functions, and forecasting methodologies that aim to enhance the accuracy and efficiency of predictions. This review will explore the general prediction techniques utilized in modeling volatile price data, with a particular focus on exponential smoothing forecasts as a benchmark. Furthermore, it will discuss the optimization of cost functions in prediction markets, detailing the various functions identified in the literature and the rationale for selecting specific models for implementation.

3.1 General Prediction Techniques

In the realm of financial forecasting, various prediction techniques are employed to model volatile price data. Among these, exponential smoothing forecasts are widely recognized for their effectiveness in capturing trends and seasonality in time series data. Exponential smoothing methods, including simple, double, and triple exponential smoothing, provide a systematic approach to forecasting by assigning exponentially decreasing weights to past observations. This characteristic makes them particularly suitable for volatile financial markets, where recent data is often more indicative of future trends than older data [5, 3].

The application of exponential smoothing in financial contexts has been extensively documented. For instance, Shiba et al. highlight the predictability of realized volatility in international stock markets, emphasizing the utility of exponential smoothing techniques in capturing market fluctuations amid uncertainty [5]. Similarly, Ma et al. discuss the importance of leveraging historical price data to forecast volatility, underscoring the relevance of exponential smoothing in their modeling framework [3]. These studies illustrate the robustness of exponential smoothing as a benchmark forecasting technique, particularly in environments characterized by high volatility.

Moreover, alternative forecasting methods, such as ARIMA (Autoregressive Integrated Moving Average) models, have also gained traction in the literature. ARIMA models are adept at capturing the autocorrelation present in time series data, making them suitable for financial applications where past price movements influence future prices [1]. However, while ARIMA models provide a comprehensive approach to forecasting, they often require more complex parameter tuning compared to exponential smoothing methods, which can be a limitation in rapidly changing markets.

In addition to ARIMA, machine learning techniques have increasingly been applied to financial forecasting. Techniques such as Support Vector Regression (SVR) and Long Short-Term Memory (LSTM) networks have shown promise in capturing non-linear relationships in price data [2, 4]. These methods leverage large datasets to identify patterns and trends that may not be immediately apparent through traditional statistical techniques. However, the complexity and computational requirements of these models can pose challenges, particularly for practitioners seeking quick and interpretable forecasts. Despite the growing interest in machine learning approaches, exponential smoothing remains a foundational technique in the forecasting toolkit. Its simplicity, ease of implementation, and effectiveness in volatile environments make it a valuable benchmark against which more complex models can be evaluated.

The aim of this paper is to evaluate the performance of prediction techniques specifically developed for modeling prediction markets, using exponential smoothing as the benchmark. By comparing the predictive accuracy of advanced cost functions tailored to prediction markets against exponential smoothing, this study seeks to assess the effectiveness of these models in accurately forecasting the outcomes represented by market prices.

3.2 Cost Functions for Prediction Markets

The optimization of cost functions in prediction markets has been a central area of research, with various functional forms proposed to improve market efficiency and enhance predictive accuracy. These cost functions are designed to aggregate market participants' information into a single price that reflects the probability of an event. While many cost functions consider multiple data inputs, such as price and volume, due to data availability this study is limited to using price data alone, which imposes certain constraints on the selection of functional forms.

Initially, nine cost functions were identified from the literature as potential candidates for implementation. These include the Logarithmic Market Scoring Rule (LMSR), quadratic scoring rule, sigmoid function, and generalized cost functions for convex optimization [11, 13, 16]. Each of these functions has unique characteristics that influence how they aggregate information and update market prices based on trades.

The LMSR, for instance, is characterized by its ability to dynamically adjust market prices based on trades, ensuring arbitrage-free pricing and balancing market liquidity [12, 6]. Modifications, such as optimizing liquidity parameters,

further enhance its forecasting precision by reducing standard errors [16]. The sigmoid function, another key cost function, effectively maps prices into probabilities, capturing non-linear relationships in the data and accommodating real-world uncertainties, such as those in renewable energy forecasts [17].

However, after careful consideration, the list was narrowed to five based on their reliance on price data. Of these, three functions—focused on regret minimization, smoothness conditions, and combinatorial frameworks—were excluded after further analysis [13, 14, 15]. These approaches rely heavily on assumptions about expected prices, which, when applied to price-only data, reduce to simple variance calculations, limiting their practical utility for this study.

As a result, two cost functions were selected for implementation: the Logarithmic Market Scoring Rule (LMSR) and the Sigmoid Function. Both functions are well-suited for price-only datasets and have distinct theoretical foundations that align with the study's objectives. The LMSR's logarithmic structure ensures proportional price updates based on trade size, maintaining market liquidity and stability. The Sigmoid function offers a flexible framework for capturing non-linear relationships, with parameters controlling curve steepness and midpoint.

By focusing on these two cost functions, the study leverages their complementary properties to evaluate market behavior effectively. This selection reflects both the practical constraints of the dataset and the theoretical strengths of these models in aggregating and interpreting market information. The integration of LMSR and the Sigmoid function into the modeling framework will facilitate a nuanced understanding of how price data can forecast market outcomes in prediction markets.

3.3 Logarithmic Market Scoring Rule (LMSR)

Agrawal et al. [12] introduce the **Logarithmic Market Scoring Rule (LMSR)** as a foundational cost function for prediction markets. The LMSR is defined as:

$$C(q) = b \ln(1 + e^{q/b}),$$

where q represents the total number of shares held, and b is a parameter that determines the liquidity of the market. The LMSR offers several significant advantages. Its logarithmic structure ensures that price updates are proportional to the size of the trades, which maintains market efficiency while avoiding arbitrage opportunities. This property makes LMSR particularly appealing for markets characterized by high levels of uncertainty, as it balances stability with responsiveness to new information.

A key strength of LMSR lies in its ability to continuously adjust prices in response to market activity. This feature not only prevents market manipulation but also promotes more accurate aggregation of information from diverse participants. The function's dependence on the liquidity parameter b allows for

customization based on market conditions. For instance, a higher b results in less volatile price adjustments, fostering stability in markets with fewer participants or less trading activity. Conversely, a smaller b creates more dynamic price updates, which may be suitable for highly liquid markets with frequent trading. These properties have established LMSR as one of the most widely used cost functions in prediction market research and practical applications.

Furthermore, Kuzheliev et al. [10] emphasize the importance of mathematical modeling in financial markets, noting that the LMSR's structure aligns well with modern econometric forecasting tools. This alignment enhances the LMSR's applicability in various market scenarios, reinforcing its role as a reliable cost function in prediction markets.

3.4 Sigmoid Function

Shamsi and Cuffe [17] propose the use of a **sigmoid function** to model the relationship between market prices and probabilities in binary outcome prediction markets. Their cost function is given by:

$$P(x) = \frac{1}{1 + e^{-k(x-x_0)}}$$

where k controls the steepness of the curve, and x_0 determines the midpoint. The sigmoid function is particularly effective in capturing the non-linear dynamics of market prices, making it a natural fit for binary outcomes where probabilities transition smoothly between extremes. The authors applied this formulation to various contexts, demonstrating its ability to translate market prices into probabilistic predictions.

The sigmoid function's flexibility allows it to adapt to various market behaviors through its parameters. The steepness parameter k can be adjusted to reflect the sensitivity of price changes to new information, while the midpoint x_0 provides a reference point for interpreting price-probability relationships. This adaptability is particularly advantageous in markets where price movements may not directly correspond to linear changes in probability. By providing a continuous and differentiable mapping of prices to probabilities, the sigmoid function enhances interpretability and predictive accuracy in binary markets.

Moreover, the application of the sigmoid function in modeling market dynamics is supported by the work of Ishii et al. [9], who highlight its utility in sociophysics analysis, where human interactions and societal trends influence market behavior. This perspective reinforces the sigmoid function's relevance in capturing the complexities of market dynamics, particularly in environments characterized by rapid information flow and participant diversity.

Both the LMSR and sigmoid functions are designed to address the unique challenges of prediction markets, particularly those involving binary outcomes. While LMSR emphasizes market stability and arbitrage-free pricing, the sigmoid function excels in scenarios requiring smooth transitions and probabilistic interpretations of prices. Together, these functions provide robust tools for

understanding and modeling market dynamics, with applications ranging from financial forecasting to renewable energy predictions.

In conclusion, the selection of the LMSR and sigmoid functions for this study reflects their theoretical strengths and practical applicability in modeling prediction markets. By leveraging these cost functions, the research aims to enhance the understanding of market behavior and improve the accuracy of price predictions in various contexts.

4 Sandbox Model for Prediction Markets

Prediction markets are designed to aggregate participant beliefs into a single price reflecting the probability of an event occurring. Before starting the empirical modeling of these markets, it is essential to first construct a simplified model to better understand how market prices arise from the aggregation of individual beliefs. This section introduces a sandbox model built upon the Logarithmic Market Scoring Rule (LMSR) and the sigmoid function as cost functions. This model provides a controlled environment to study how market prices evolve in response to trades and information updates, laying the groundwork for more complex analyses.

4.1 Market Framework

This sandbox model simulates a binary outcome prediction market, where the market price p_t at time t represents the aggregated belief about the probability of an event occurring. Traders are assumed to be rational, updating their beliefs based on Bayesian inference as new information arrives. A liquidity parameter—denoted b for LMSR and k for the sigmoid function—governs the sensitivity of price changes to trading activity.

The LMSR ensures arbitrage-free pricing by design, as established by Agrawal et al. [12], and its behavior under different liquidity conditions has been extensively analyzed [16]. Meanwhile, the sigmoid function complements LMSR by offering a flexible mapping of aggregated market beliefs to probabilities, capturing non-linear dynamics effectively [17].

4.2 Logarithmic Market Scoring Rule (LMSR)

Agrawal et al. [12] define the cost function $C(q)$ of the LMSR as:

$$C(q) = b \ln(1 + e^{q/b}),$$

where q represents the net position in the market. The price p_t of the "Yes" outcome is derived as the derivative of the cost function:

$$p_t = \frac{\partial C(q)}{\partial q} = \frac{e^{q/b}}{1 + e^{q/b}}.$$

This mapping ensures that p_t lies within the interval $[0, 1]$, making it interpretable as a probability [13]. The liquidity parameter b controls the responsiveness of price changes to trading activity. Smaller values of b result in sharper price adjustments, while larger values produce smoother transitions. This property is crucial for maintaining market stability and efficiency, as noted by Opschoor et al. [7].

4.3 Sigmoid Function

Shamsi and Cuffe [17] propose the use of a **sigmoid function** to model the relationship between market prices and probabilities. The sigmoid cost function is given by:

$$P(x) = \frac{1}{1 + e^{-k(x - x_0)}},$$

where:

- x is the input variable, representing aggregated net positions or signals.
- $k > 0$ controls the steepness of the curve.
- x_0 sets the midpoint, defining the market's initial equilibrium.

The price p_t of the "Yes" outcome is calculated from the sigmoid function:

$$p_t = P\left(\frac{q}{b}\right) = \frac{1}{1 + e^{-k(q/b - x_0)}}.$$

This formulation provides a flexible mapping of market prices to probabilities, capturing non-linear dynamics effectively, as highlighted by Archetti and Scheuring [**?**]. The sigmoid function's adaptability makes it a useful complement to LMSR, particularly in scenarios where smooth probability transitions are required.

Both LMSR and the sigmoid function offer robust theoretical frameworks for modeling prediction markets. Their combined use in this sandbox model facilitates an in-depth exploration of market dynamics, preparing the groundwork for data-driven investigations.

4.4 Information Arrival and Trader Behavior

Traders revise their beliefs upon observing signals s_t about the likelihood of an event occurring. Let $P(\text{Yes})$ and $P(\text{No})$ represent the true probabilities of the outcomes, and $P(s_t \mid \text{Yes})$ and $P(s_t \mid \text{No})$ denote the likelihoods of the signal. A trader's posterior belief $p_j^{\text{posterior}}$ after observing s_t is calculated using Bayes' theorem:

$$p_j^{\text{posterior}} = \frac{P(s_t \mid \text{Yes}) \cdot p_j^{\text{prior}}}{P(s_t \mid \text{Yes}) \cdot p_j^{\text{prior}} + P(s_t \mid \text{No}) \cdot (1 - p_j^{\text{prior}})},$$

where p_j^{prior} is the trader's prior belief. This Bayesian updating process is fundamental to both LMSR and sigmoid frameworks, as it allows traders to adjust their positions based on new information.

The trader submits a trade q_j^t based on the deviation of their posterior belief from the current market price:

$$q_j^t = \alpha \left(p_j^{\text{posterior}} - p_t \right),$$

where $\alpha > 0$ is a risk tolerance parameter. Positive trades ($q_j^t > 0$) reflect higher probabilities assigned to "Yes," while negative trades indicate a stronger belief in "No." This behavior is consistent with the findings of Chen and Vaughan [13], who emphasize the importance of trader incentives in shaping market dynamics.

4.5 Price Update Mechanism

The LMSR updates the market price dynamically based on aggregate trading activity. After n trades, the total quantity of shares is updated as:

$$q^{(n)} = q^{(n-1)} + \sum_{j=1}^{m} q_j^n,$$

where m is the number of traders in the n-th round. The market price is then recalculated using the LMSR price formula:

$$p_t^{(n)} = \frac{e^{q^{(n)}/b}}{1 + e^{q^{(n)}/b}}.$$

In the sigmoid-based framework, the market price is recalculated after each round of trading:

$$p_t^{(n)} = \frac{1}{1 + e^{-k(q^{(n)}/b - x_0)}}.$$

This dual approach allows for a comprehensive understanding of how price updates occur in response to trading activity and information flow.

4.6 Analytical Properties: Comparing LMSR and Sigmoid Frameworks

Both the LMSR and sigmoid cost functions offer unique advantages.

LMSR ensures arbitrage-free pricing and stability, with price changes controlled by the liquidity parameter b. The convexity of the LMSR cost function guarantees that:

$$C(q_1 + q_2) \geq C(q_1) + C(q_2).$$

This property prevents traders from exploiting price discrepancies for risk-free profit, as discussed by Pennock et al. [8].

Sigmoid provides smooth transitions and flexible probabilistic interpretations, with adaptability controlled by k and x_0. The sigmoid function's ability to capture non-linear relationships enhances its applicability in various market scenarios, as noted by Krüger et al. [**?**].

In conclusion, the sandbox model combines LMSR's arbitrage-free guarantees with the sigmoid function's flexibility, offering robust tools for analyzing binary outcome prediction markets. By incorporating Bayesian updates, trader decision-making, and LMSR-based price adjustments, the model captures essential features of price formation and information aggregation in prediction markets. This foundation enables deeper exploration of advanced functional forms and optimization techniques in subsequent sections.

5 Modifications of the Functions

5.1 Modified LMSR for Predicted Prices

The traditional LMSR relies on cumulative market activity, quantified as the total number of shares traded, to calculate prices. However, for the purpose of prediction—estimating the current price p_t based on the previous price p_{t-1}—the LMSR is reformulated to work directly with observed market prices. The derivative of the cost function, which gives the market price, is used as the basis for this modification. The modified LMSR is expressed as:

$$p_t = \frac{e^{p_{t-1}/b}}{1 + e^{p_{t-1}/b}},$$

where p_t is the predicted price at time t, p_{t-1} is the observed price at time $t-1$, and b is the liquidity parameter controlling price sensitivity.

This modification shifts the focus from cumulative shares (q) in the original LMSR formulation to observed prices (p_t), making it directly applicable for prediction purposes. By using the derivative of the LMSR cost function, the model predicts the current price p_t based on the price observed in the previous time step p_{t-1}. The original LMSR's reliance on cumulative quantities is not suitable for price prediction tasks due to the lack of available data on trading volume, and using the past observed price, p_{t-1}, allows the function to model how prices evolve over time. The reformulated LMSR predicts the current market price p_t by capturing the dynamic relationship between consecutive prices, reflecting how markets adjust to new information.

The liquidity parameter b continues to control the sensitivity of price changes to updates, with larger b values resulting in smoother predictions and smaller b values allowing for more rapid adjustments. This parameter is critical for tailoring the model to different market conditions, as it governs how prices respond to new information.

The primary goal of this modification is to predict the current price (p_t) based on the previous price (p_{t-1}). This approach assumes that price changes follow the underlying market dynamics dictated by the LMSR framework. By

iteratively applying this prediction model, it becomes possible to simulate price trajectories and analyze how information is incorporated into market prices. The modified LMSR retains key properties of the original function: predicted prices remain within the range $[0, 1]$, ensuring interpretability as probabilities. The function dynamically updates based on changes in observed prices, reflecting real-time market behavior. Additionally, the liquidity parameter b allows the model to adapt to different levels of market activity and sensitivity requirements.

This reformulation makes the LMSR more versatile for predictive analytics, providing a robust framework for understanding price dynamics in prediction markets while maintaining its foundational probabilistic and dynamic adjustment strengths.

5.2 Modified Sigmoid for Predicted Prices

The Sigmoid function provides a flexible and probabilistic framework for modeling price dynamics. In its original form, the sigmoid maps an input variable, such as share quantities or uncertainty levels, to a probability between 0 and 1. For this study, the sigmoid function was adapted to predict market prices directly, focusing on estimating the current price p_t based on the previous observed price p_{t-1}. The modified formulation is expressed as:

$$p_t = \frac{1}{1 + e^{-k(p_{t-1} - x_0)}},$$

where p_t is the predicted price at time t, p_{t-1} is the observed price at time $t - 1$, k is the steepness of the sigmoid curve, and x_0 is the midpoint price representing the reference point of the sigmoid curve.

This modification involves several key changes to the original sigmoid function. First, the generic input variable (x) was replaced with the observed price from the previous time step (p_{t-1}), aligning the function with the task of predicting price dynamics over time. Second, the normalization factor (b), originally used to scale the input variable, was omitted because the price data in this study is already normalized within the range $[0, 1]$. These adjustments allow the sigmoid function to model price transitions directly, leveraging the inherent probabilistic nature of the dataset.

The parameters k and x_0 are critical to the flexibility and accuracy of the sigmoid function. The steepness parameter (k) governs how quickly the predicted prices transition between extremes, capturing the sensitivity of the market to new information. The midpoint parameter (x_0) acts as a pivot point, centering the sigmoid curve and representing the reference level of the price distribution. By optimizing these parameters, the modified sigmoid function can effectively model the non-linear relationships often observed in market price data.

One of the advantages of the sigmoid function is its smooth and continuous mapping of prices to probabilities. This characteristic is particularly useful for normalized price data, simplifying implementation and interpretation while maintaining compatibility with binary outcome markets. Additionally, the sig-

moid's flexibility allows it to adapt to a variety of market scenarios, making it a valuable tool for capturing non-linear price dynamics.

The choice of LMSR and sigmoid functions in this study was motivated by their complementary strengths. The LMSR is well-suited for balancing market stability with dynamic responsiveness through its liquidity parameter, while the sigmoid function excels in providing a probabilistic interpretation of prices with adjustable steepness and midpoint parameters. Together, these functions offer robust frameworks for understanding and predicting price dynamics in binary outcome markets. Their modifications ensure compatibility with price-only datasets and enhance their utility for predictive analysis.

In conclusion, the adaptation of the sigmoid function to predict prices directly from past observed prices reflects its versatility and effectiveness in modeling normalized market data. By focusing on parameters that influence steepness and midpoint, the sigmoid function becomes a powerful tool for capturing market behavior, particularly in binary outcome prediction markets. These modifications provide a strong foundation for further analysis and evaluation of prediction market dynamics.

6 Data Availability and Preprocessing

The dataset used in this study originates from Polymarket, a prediction market platform. These markets are structured to reflect the implied probability of a particular outcome through their price data. The dataset comprises price data aggregated on an hourly basis, providing sufficient frequency for time-series analysis. The focus of this study is on binary outcome markets, as they align with the modeling frameworks discussed earlier.

Table 1 presents the average summary statistics across 371 markets analyzed in this study. Key metrics, such as the mean price of 0.2331, median price of 0.2252, variance of 0.0098, and standard deviation of 0.0636, represent averages calculated across all 371 markets. The observed price ranges from a minimum of 0.0005 to a maximum of 0.9995, reflecting the overall range across all markets. The table also reports that, on average, each market consisted of 486.95 observations. Additionally, 85 markets resolved to the outcome "Yes," while 285 resolved to the outcome "No." These summary statistics provide a comprehensive overview of the data's distribution and characteristics, forming the foundation for the subsequent analysis.

The raw dataset initially underwent a trading volume cutoff, where markets with total trading volumes below 1 million were excluded. This step ensured that the analysis focused on markets with sufficient liquidity and reduced noise from low-activity markets. After applying this cutoff, the dataset contained 121 distinct markets that had resolved as of October 15th. These markets were then split into binary outcome markets, such as separating multi-candidate election markets into individual markets for each candidate. This preprocessing step significantly increased the total number of markets to 428, providing a more granular dataset for analysis.

Average Statistic	Value
Mean	0.2331
Median	0.2252
Variance	0.0098
Standard Deviation	0.0636
Minimum	0.0005
Maximum	0.9995
Number of Observations	486.95
Starting Value	0.2999
Ending Value	0.2329
Number of Outcome "Yes"	85
Number of Outcome "No"	285

Table 1: Average Summary Statistics Across 371 Different Markets

Further preprocessing steps reduced the number of markets from 428 to 371 to ensure data quality and alignment with the study's objectives. First, markets with fewer than 50 observations were excluded, as they lacked sufficient data for meaningful analysis. Second, markets that did not resolve to a clear "Yes" or "No" outcome were removed. Specifically, markets ending with prices between 0.1 and 0.9 were excluded, as these prices indicate unresolved outcomes. Such cases likely resulted from errors in data collection or anomalies in market behavior. This exclusion was necessary because prediction markets are designed to resolve to definitive outcomes, with prices near 0 for "No" and near 1 for "Yes."

The remaining dataset includes 371 markets with an average duration of 487 observations per market, providing ample data for time-series modeling and analysis. These preprocessing steps ensured that the dataset is of high quality, focuses on reliable market data, and aligns with the binary outcome framework, forming a robust foundation for the subsequent analysis.

7 Implementation and Optimization

The implementation of the Modified LMSR, Modified Sigmoid, and Exponential Smoothing models was carried out in Python, utilizing the libraries `numpy`, `pandas`, `scipy`, and `sklearn`. These libraries provided the necessary tools for efficient numerical computation, data manipulation, and optimization. The detailed implementation code for these models can be found in the appendix.

To fit the prediction models, the Limited-memory Broyden–Fletcher–Goldfarb–Shanno with Box constraints (L-BFGS-B) optimization algorithm, as implemented in `scipy.optimize`, was used. This gradient-based optimization method efficiently minimizes the Residual Sum of Squares (RSS), a measure of the total squared difference between observed and predicted prices. L-BFGS-B is particularly effective for problems involving continuous variables with constraints, making it a suitable choice for the Modified LMSR and Sigmoid models.

The L-BFGS-B method offers several advantages that align with the needs of this study. First, it approximates second derivative (Hessian) information using limited memory, significantly reducing computational overhead compared to algorithms that compute the full Hessian matrix. This efficiency is crucial for processing larger datasets. Second, the algorithm supports parameter constraints, ensuring that the optimized values remain within meaningful ranges. For instance, in the Modified LMSR model, the liquidity parameter b was constrained to be positive, as negative liquidity values lack interpretability. Similarly, in the Modified Sigmoid model, constraints were applied to the steepness parameter k and midpoint parameter x_0 to maintain valid and interpretable predictions. These constraints were instrumental in preserving the mathematical and conceptual integrity of the models.

To evaluate the performance of the models and assess their predictive capabilities, several validation metrics were employed. The Mean Squared Error (MSE) was used to measure the average squared difference between observed and predicted prices, providing an overall sense of prediction accuracy. The Root Mean Squared Error (RMSE), calculated as the square root of the MSE, was also reported. Since RMSE is expressed in the same units as the observed prices, it offers a more interpretable measure of prediction error magnitude.

Additionally, the Coefficient of Determination (R^2) was calculated to evaluate how well the models explained the variability in observed prices. R^2 values range from 0 to 1, with values closer to 1 indicating stronger model performance. Negative R^2 values suggest that the model performs worse than a simple mean prediction. This metric was particularly useful for comparing the explanatory power of the models. Finally, the Residual Sum of Squares (RSS), which quantifies the total squared error between observed and predicted values, was minimized directly during the optimization process. Minimizing RSS ensures that the models closely fit the observed data within the constraints and assumptions imposed.

In conclusion, the Python-based implementation and the use of the L-BFGS-B optimization algorithm enabled efficient and effective fitting of the models while ensuring parameter constraints were respected. The combination of robust optimization techniques and comprehensive validation metrics provides a strong foundation for analyzing the predictive performance of the Modified LMSR, Sigmoid, and Exponential Smoothing models. For further details on the implementation, refer to Appendices 1, 2, and 3.

8 Preliminary Results

The implementation of the Modified Logarithmic Market Scoring Rule (LMSR), Sigmoid function, and Exponential Smoothing models across 371 binary prediction market bets provided meaningful insights into their predictive performance. Table 2 summarizes the average performance metrics, including the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Coefficient of Determination (R^2), along with the optimized parameter values for each

model.

Model	**Parameter(s)**	**MSE**	**RMSE**	R^2
Modified LMSR	Opt. $b = 0.8663$	0.00634	0.04981	0.30945
Modified Sigmoid	Opt. $k = 5.8048$, $x_0 = 0.4866$	0.000998	0.02164	0.80902
Exponential Smoothing	$\alpha = 0.3$	0.000723	0.01816	0.76958

Table 2: Performance Metrics for Modified LMSR, Modified Sigmoid, and Exponential Smoothing Models

Exponential Smoothing served as the benchmark model for this study, achieving an average MSE of 0.000723, RMSE of 0.01816, and an R^2 value of 0.7696. These results indicate that Exponential Smoothing performs well at capturing overall trends in the data while providing a reasonable level of predictive accuracy. The relatively high R^2 value suggests that the model explains approximately 77% of the variability in market prices, making it a solid reference point for assessing the newer approaches.

The Modified Sigmoid function demonstrated the best performance among the three models, with an average MSE of 0.000998, RMSE of 0.02164, and an R^2 value of 0.8090. This indicates that the Sigmoid function not only achieves a low prediction error but also explains over 80% of the variability in market prices. The optimized steepness parameter ($k = 5.8048$) and midpoint ($x_0 = 0.4866$) further enhanced the model's ability to adapt to the observed price movements, allowing it to effectively capture non-linear dynamics and probabilistic relationships in the data.

The Modified LMSR model achieved an average MSE of 0.00634, RMSE of 0.04981, and an R^2 value of 0.3095. While its R^2 value is considerably lower than that of the Sigmoid function, the LMSR still performed reasonably well, particularly in its ability to dynamically adjust prices based on the liquidity parameter. The optimized liquidity parameter ($b = 0.8663$) ensured stable price updates, but the model's higher MSE and RMSE suggest that it struggled to match the precision of the Sigmoid function and Exponential Smoothing. The lower R^2 value indicates that the LMSR explained only about 31% of the variability in market prices, making it less effective for capturing the complex dynamics of binary prediction markets.

The Modified Sigmoid function emerged as the best-performing model, with the lowest prediction errors (MSE and RMSE) and the highest R^2 value. Its strong performance likely stems from its similarity to the logistic regression function, which is specifically designed to model probabilities within the [0,1] range. This compatibility with the normalized price data in prediction markets makes the Sigmoid function particularly effective for modeling binary outcome markets. The high R^2 value suggests that the Sigmoid function not only fits the training data well but also captures the underlying market dynamics with high accuracy. However, this level of precision raises the possibility of overfitting, highlighting the importance of further validation, such as cross-validation or testing on independent datasets, to confirm its robustness.

In comparison, the Exponential Smoothing model provided a solid baseline, with reasonable prediction accuracy and an R^2 value close to 77%. While it does not match the precision of the Sigmoid function, its performance demonstrates its utility as a benchmark for time-series forecasting. Its simplicity and widespread applicability make it a reliable reference for evaluating newer, more complex models.

The Modified LMSR model, while less precise than the Sigmoid function and Exponential Smoothing, still offers valuable insights into the role of liquidity in market pricing. The lower R^2 value indicates that the LMSR struggled to explain as much of the variability in market prices, but its performance highlights its robustness and dynamic adjustment capabilities. These properties may make the LMSR more suitable for applications where market liquidity plays a central role.

Figures 1, 2, and 3 illustrate example fits of the three models applied to the "Trump wins the Presidency" prediction market. These visualizations provide further insight into how each model captures the dynamics of market prices:

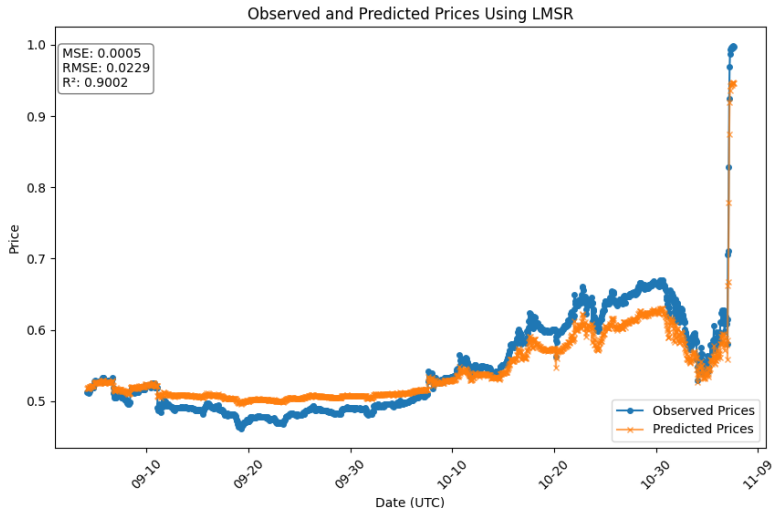

Figure 1: Example Fit: Modified LMSR on 'Trump wins the Presidency' Contract

In Figure 1, the Modified LMSR demonstrates its ability to dynamically adjust prices but shows some deviation from the observed data, particularly in periods of rapid price changes. This behavior reflects the model's limitations in capturing non-linear dynamics compared to the Sigmoid function.

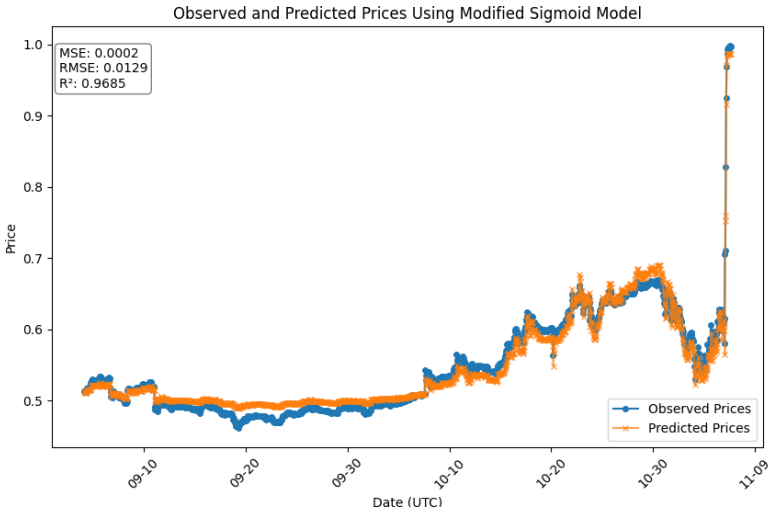

Figure 2: Example Fit: Modified Sigmoid on 'Trump wins the Presidency' Contract

Figure 2 highlights the strength of the Sigmoid function in closely tracking observed prices. Its smooth, probabilistic nature allows it to model market dynamics effectively, providing a near-perfect fit to the data. This further demonstrates the Sigmoid function's suitability for binary prediction markets.

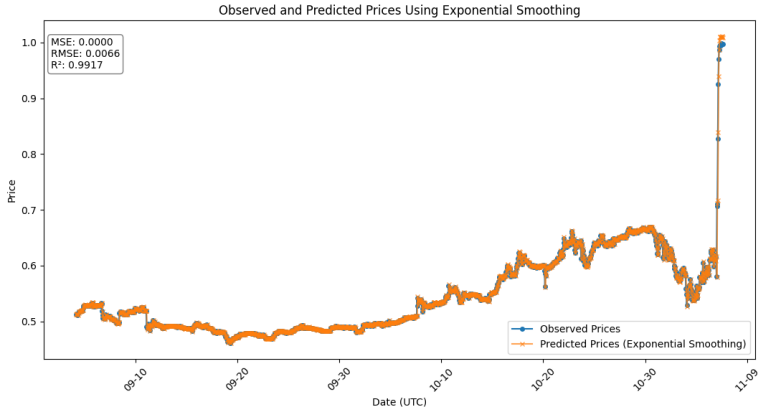

Figure 3: Example Fit: Exponential Smoothing on 'Trump wins the Presidency' Contract

In Figure 3, Exponential Smoothing captures the overall trend in market prices in this particular example exceptionally well. It even has a higher R^2 value compared to the Sigmoid function, which contradicts the average over all 371 observed markets. This highlights the limitations of relying on a single example fit for evaluation; meaningful conclusions can only be drawn by analyzing the aggregate performance across all 371 observed markets.

Table 2 highlights the comparative performance of the three models. The Modified Sigmoid function emerges as the most effective approach, with the highest predictive accuracy and explanatory power. The Exponential Smoothing model, while less precise, serves as a reliable benchmark, while the Modified LMSR provides valuable insights into market dynamics but lags behind in terms of precision and fit. These findings emphasize the importance of selecting models that balance predictive accuracy, interpretability, and generalizability for binary prediction markets.

9 Future Work

This study lays a foundation for understanding the behavior and effectiveness of models used in prediction markets. However, two key areas require further exploration: connecting the sandbox model to empirical data and making the data analysis more differentiated and robust. Future work will address these challenges to enhance both the theoretical and empirical contributions of this research.

A primary objective will be to bridge the gap between the sandbox model and the empirical findings derived from Polymarket data. The sandbox model relies on individual belief probabilities, which are not directly observable in the dataset. This lack of data presents a significant challenge in validating the sandbox's theoretical properties—such as equilibrium stability, price responsiveness, and sensitivity to information updates—against real-world market behavior. Future efforts will explore innovative methods to approximate individual beliefs, such as inference techniques from Bayesian statistics or reverse-engineering implied probabilities from trading patterns. These approaches could provide a pathway to partially connect the sandbox model to empirical data.

Another important direction is to make the data analysis more differentiated. This includes classifying markets by topics, such as politics, sports, or finance, and examining whether the models perform differently across these categories. Similarly, markets could be grouped by behavioral characteristics, such as volatility patterns or trading volume. By running the models on these differentiated subsets, future work can uncover nuances in model performance and improve their interpretability. To ensure robustness, additional techniques such as cross-validation, time-series split validation, and ensemble methods could be employed. Bootstrapping and Monte Carlo simulations can also assess the stability of the results across varied datasets.

Future work will also explore whether to focus more heavily on smoothing techniques or predictive modeling. The smoothing approach could involve ad-

vanced methods like Holt-Winters exponential smoothing or state-space models to capture trends and seasonality more effectively. On the prediction side, incorporating more sophisticated machine learning algorithms, such as recurrent neural networks (RNNs) or gradient boosting machines, could provide deeper insights into market behavior. The choice between these paths will depend on the primary objective of future research—whether it is to better understand historical price trends or improve the accuracy of forward-looking predictions.

Finally, this study could benefit from a deeper exploration of the theoretical underpinnings of the models. Concepts from probability theory, differential equations, and optimization could be used to analyze how cost functions drive price evolution and aggregate market information. This includes investigating the sensitivity of model parameters, the stability of equilibrium solutions, and the responsiveness of prices to new information. These theoretical insights would complement the empirical findings, providing a more comprehensive understanding of prediction market dynamics.

By integrating sandbox-derived findings with more robust and differentiated data analysis, and further exploring theoretical foundations, this research will aim to enhance the reliability, interpretability, and practical applicability of the models. These efforts will ensure a deeper and more nuanced understanding of prediction market behavior while addressing the challenges of both theory and practice.

10 Conclusion

This study evaluates models for prediction markets, focusing on their ability to aggregate information and predict outcomes accurately. By integrating theoretical analysis and empirical validation, the research provides insights into the strengths and limitations of different modeling approaches. I would like to thank Dr. Coulter and Dr. Vogel for their guidance and support throughout the first semester, which has been essential to the progress of this work. Their feedback has shaped the direction of this research, and I look forward to continuing to collaborate with them as the study advances.

11 Appendices: Code

11.1 Modified LMSR Optimization

The following Python code contains the implementation of the Modified LMSR function as well as the preceding data manipulation.

```
import os
import numpy as np
import pandas as pd
from scipy.optimize import minimize
from sklearn.metrics import mean_squared_error, r2_score

# Directory containing the CSV files
folder_path = r"C:\Users\conra\OneDrive - Stetson University, Inc\bildung\5th Sem Stetson\senior project\polymarket raw data"

# Output file path
combined_file = r"C:\Users\conra\OneDrive - Stetson University, Inc\bildung\5th Sem Stetson\senior project\lmsr_results.csv"

# Initialize a list to store combined results
combined_results = []

# Define the updated LMSR formula
def lmsr_price(p_t_minus_1, p0, b):
    """
    LMSR formula: p_t = e^{q/b} / (1 + e^{q/b})
    """
    q = p_t_minus_1 - p0
    return np.exp(q / b) / (1 + np.exp(q / b))

# Define the Residual Sum of Squares (RSS) function
def residual_sum_of_squares(b, p_t_minus_1, p0):
    """
    RSS: Sum of squared differences between observed prices and LMSR-mapped prices
    """
    predicted_prices = lmsr_price(p_t_minus_1, p0, b)
    return np.sum((p_t_minus_1 - predicted_prices) ** 2)

# Loop over all files in the folder
for file_name in os.listdir(folder_path):
    if file_name.endswith(".csv"):
        file_path = os.path.join(folder_path, file_name)

        # Load the CSV file
        data = pd.read_csv(file_path)

        # Extract columns after "Date (UTC)" and "Timestamp (UTC)"
        outcome_columns = data.columns[2:]

        # Loop over each outcome column
        for outcome in outcome_columns:
            prices = data[outcome].values

            # Handle NaN values
```

```
            valid_indices = ~pd.isna(prices)
            prices = prices[valid_indices]

            # Skip if there are insufficient data points
            if len(prices) < 50:
                print(f"Skipping {file_name} - {outcome}: Insufficient data points.")
                continue

            # Check for constant prices
            if np.min(prices) == np.max(prices):
                print(f"Skipping {file_name} - {outcome}: Constant prices.")
                continue

            # Compute summary statistics for this outcome
            ending_value = prices[-1]
            yes = 1 if ending_value > 0.9 else 0
            no = 1 if ending_value < 0.9 else 0

            # Skip rows where both Yes and No are 0
            if yes == 0 and no == 0:
                continue

            # Compute summary statistics
            summary = {
                'File': file_name,
                'Outcome': outcome,
                'Mean': np.mean(prices),
                'Median': np.median(prices),
                'Variance': np.var(prices),
                'Standard Deviation': np.std(prices),
                'Minimum': np.min(prices),
                'Maximum': np.max(prices),
                'Number of Observations': len(prices),
                'Starting Value': prices[0],
                'Ending Value': ending_value,
                'Yes': yes,
                'No': no
            }

            # Normalize prices to [0, 1]
            original_min, original_max = np.min(prices), np.max(prices)
            prices_normalized = (prices - original_min) / (original_max - original_min)

            # Initial reference price p0 (e.g., the first observed price)
            p0 = prices_normalized[0]

            # Shift the normalized prices to create p_t_minus_1
            p_t_minus_1 = prices_normalized[:-1]
            p_t = prices_normalized[1:]

            # Initial guess for b
            initial_b = 0.5
```

```
            # Minimize the RSS to optimize b
            result = minimize(
                residual_sum_of_squares,
                [initial_b],
                args=(p_t_minus_1, p0),
                bounds=[(0.01, 10.0)],  # Ensure b > 0
                method='L-BFGS-B',
                options={'disp': False}
            )

            # Extract the optimized liquidity parameter b
            b_optimized = result.x[0]

            # Calculate predicted prices using the optimized b
            predicted_prices_normalized = lmsr_price(p_t_minus_1, p0, b_optimized)

            # Rescale predicted prices back to the original range
            predicted_prices = predicted_prices_normalized * (original_max - original_min) + original_min
            observed_prices = p_t * (original_max - original_min) + original_min

            # Compute metrics
            mse = mean_squared_error(observed_prices, predicted_prices)
            rmse = np.sqrt(mse)
            r_squared = r2_score(observed_prices, predicted_prices)

            # Combine summary statistics and optimization results
            summary.update({
                'Optimized b': b_optimized,
                'MSE': mse,
                'RMSE': rmse,
                'R^2': r_squared
            })

            # Append to combined results
            combined_results.append(summary)

# Convert combined results to a DataFrame
combined_results_df = pd.DataFrame(combined_results)

# Save the combined results to a CSV file
combined_results_df.to_csv(combined_file, index=False)

print(f"Combined results saved to {combined_file}")
```

Listing 1: ACO Simulation Implementation

11.2 Modified Sigmoid Optimization

The following Python code contains the implementation of the Modified Sigmoid function as well as the preceding data manipulation.

```
import os
import numpy as np
import pandas as pd
from scipy.optimize import minimize
from sklearn.metrics import mean_squared_error, r2_score

# Directory containing the CSV files
folder_path = r"C:\Users\conra\OneDrive - Stetson University, Inc\bildung\5th Sem Stetson\senior project\polymarket raw data"

# Output file path
combined_file = r"C:\Users\conra\OneDrive - Stetson University, Inc\bildung\5th Sem Stetson\senior project\sigmoid_results_final.csv"

# Initialize a list to store combined results
combined_results = []

# Define the sigmoid formula
def sigmoid_price(p_t_minus_1, k, x0):
    """
    Sigmoid formula: p^t = 1 / (1 + exp(-k(p_t_minus_1 - x0)))
    """
    return 1 / (1 + np.exp(-k * (p_t_minus_1 - x0)))

# Define the Residual Sum of Squares (RSS) function
def residual_sum_of_squares(params, p_t_minus_1):
    """
    RSS: Sum of squared differences between observed prices and sigmoid-mapped prices.
    """
    k, x0 = params
    predicted_prices = sigmoid_price(p_t_minus_1, k, x0)
    return np.sum((p_t_minus_1 - predicted_prices) ** 2)

# Loop over all files in the folder
for file_name in os.listdir(folder_path):
    if file_name.endswith(".csv"):
        file_path = os.path.join(folder_path, file_name)

        # Load the CSV file
        data = pd.read_csv(file_path)

        # Extract columns after "Date (UTC)" and "Timestamp (UTC)"
        outcome_columns = data.columns[2:]

        # Loop over each outcome column
        for outcome in outcome_columns:
            prices = data[outcome].values

            # Handle NaN values
            valid_indices = ~pd.isna(prices)
            prices = prices[valid_indices]
```

```
50
51          # Skip if there are insufficient data points
52          if len(prices) < 50:
53              print(f"Skipping {file_name} - {outcome}:
    Insufficient data points.")
54              continue
55
56          # Check for constant prices
57          if np.min(prices) == np.max(prices):
58              print(f"Skipping {file_name} - {outcome}: Constant
    prices.")
59              continue
60
61          # Compute summary statistics for this outcome
62          ending_value = prices[-1]
63          yes = 1 if ending_value > 0.9 else 0
64          no = 1 if ending_value < 0.9 else 0
65
66          # Skip rows where both Yes and No are 0
67          if yes == 0 and no == 0:
68              continue
69
70          # Compute summary statistics
71          summary = {
72              'File': file_name,
73              'Outcome': outcome,
74              'Mean': np.mean(prices),
75              'Median': np.median(prices),
76              'Variance': np.var(prices),
77              'Standard Deviation': np.std(prices),
78              'Minimum': np.min(prices),
79              'Maximum': np.max(prices),
80              'Number of Observations': len(prices),
81              'Starting Value': prices[0],
82              'Ending Value': ending_value,
83              'Yes': yes,
84              'No': no
85          }
86
87          # Normalize prices to [0, 1]
88          original_min, original_max = np.min(prices), np.max(
    prices)
89          prices_normalized = (prices - original_min) / (
    original_max - original_min)
90
91          # Shift the normalized prices to create p_t_minus_1
92          p_t_minus_1 = prices_normalized[:-1]
93          p_t = prices_normalized[1:]
94
95          # Initial guesses for k and x0
96          initial_k = 1.0
97          initial_x0 = 0.5
98
99          # Minimize the RSS to optimize k and x0
100         result = minimize(
101             residual_sum_of_squares,
102             [initial_k, initial_x0],
```

```
                args=(p_t_minus_1,),
                bounds=[(0.1, 10.0), (0.0, 1.0)],  # Ensure k > 0 and x0 within normalized range
                method='L-BFGS-B',
                options={'disp': False}
            )

            # Extract the optimized parameters
            k_optimized, x0_optimized = result.x

            # Calculate predicted prices using the optimized parameters
            predicted_prices_normalized = sigmoid_price(p_t_minus_1, k_optimized, x0_optimized)

            # Rescale predicted prices back to the original range
            predicted_prices = predicted_prices_normalized * (original_max - original_min) + original_min
            observed_prices = p_t * (original_max - original_min) + original_min

            # Compute metrics
            mse = mean_squared_error(observed_prices, predicted_prices)
            rmse = np.sqrt(mse)
            r_squared = r2_score(observed_prices, predicted_prices)

            # Combine summary statistics and optimization results
            summary.update({
                'Optimized k': k_optimized,
                'Optimized x0': x0_optimized,
                'MSE': mse,
                'RMSE': rmse,
                'R^2': r_squared
            })

            # Append to combined results
            combined_results.append(summary)

# Convert combined results to a DataFrame
combined_results_df = pd.DataFrame(combined_results)

# Save the combined results to a CSV file
combined_results_df.to_csv(combined_file, index=False)

print(f"Combined results saved to {combined_file}")
```

Listing 2: ACO Simulation Implementation

11.3 Exponential Smoothing Optimization

The following Python code contains the implementation of the Exponential Smoothing function as well as the preceding data manipulation.

```
import os
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error, r2_score

# Directory containing the CSV files
folder_path = r"C:\Users\conra\OneDrive - Stetson University, Inc\bildung\5th Sem Stetson\senior project\polymarket raw data"

# Output file path
combined_file = r"C:\Users\conra\OneDrive - Stetson University, Inc\bildung\5th Sem Stetson\senior project\expsmoothing_results.csv"

# Initialize a list to store combined results
combined_results = []

# Define Exponential Smoothing function
def exponential_smoothing(p_t, alpha):
    """
    Compute Exponential Smoothing for a given series and smoothing factor alpha.
    p_t: Observed prices (array-like)
    alpha: Smoothing parameter (float between 0 and 1)
    Returns smoothed prices as an array.
    """
    smoothed = np.zeros_like(p_t)
    smoothed[0] = p_t[0]  # Initialize with the first observed price
    for t in range(1, len(p_t)):
        smoothed[t] = alpha * p_t[t] + (1 - alpha) * smoothed[t - 1]
    return smoothed

# Loop over all files in the folder
for file_name in os.listdir(folder_path):
    if file_name.endswith(".csv"):
        file_path = os.path.join(folder_path, file_name)

        # Load the CSV file
        data = pd.read_csv(file_path)

        # Extract columns after "Date (UTC)" and "Timestamp (UTC)"
        outcome_columns = data.columns[2:]

        # Loop over each outcome column
        for outcome in outcome_columns:
            prices = data[outcome].values

            # Handle NaN values
            valid_indices = ~pd.isna(prices)
            prices = prices[valid_indices]

```

```
            # Skip if there are insufficient data points
            if len(prices) < 50:
                print(f"Skipping {file_name} - {outcome}: Insufficient data points.")
                continue

            # Check for constant prices
            if np.min(prices) == np.max(prices):
                print(f"Skipping {file_name} - {outcome}: Constant prices.")
                continue

            # Compute summary statistics for this outcome
            ending_value = prices[-1]
            yes = 1 if ending_value > 0.9 else 0
            no = 1 if ending_value < 0.9 else 0

            # Skip rows where both Yes and No are 0
            if yes == 0 and no == 0:
                continue

            # Compute summary statistics
            summary = {
                'File': file_name,
                'Outcome': outcome,
                'Mean': np.mean(prices),
                'Median': np.median(prices),
                'Variance': np.var(prices),
                'Standard Deviation': np.std(prices),
                'Minimum': np.min(prices),
                'Maximum': np.max(prices),
                'Number of Observations': len(prices),
                'Starting Value': prices[0],
                'Ending Value': ending_value,
                'Yes': yes,
                'No': no
            }

            # Normalize prices to [0, 1]
            original_min, original_max = np.min(prices), np.max(prices)
            prices_normalized = (prices - original_min) / (original_max - original_min)

            # Apply Exponential Smoothing with a fixed alpha
            alpha = 0.3  # Smoothing factor (can be tuned as needed)
            smoothed_prices_normalized = exponential_smoothing(prices_normalized, alpha)

            # Rescale smoothed prices back to the original range
            smoothed_prices = smoothed_prices_normalized * (original_max - original_min) + original_min
            observed_prices = prices_normalized * (original_max - original_min) + original_min

            # Compute metrics
```

```
            mse = mean_squared_error(observed_prices,
    smoothed_prices)
            rmse = np.sqrt(mse)
            r_squared = r2_score(observed_prices, smoothed_prices)

            # Combine summary statistics and smoothing results
            summary.update({
                'Alpha': alpha,
                'MSE': mse,
                'RMSE': rmse,
                'R^2': r_squared
            })

            # Append to combined results
            combined_results.append(summary)

# Convert combined results to a DataFrame
combined_results_df = pd.DataFrame(combined_results)

# Save the combined results to a CSV file
combined_results_df.to_csv(combined_file, index=False)

print(f"Combined results saved to {combined_file}")
```

Listing 3: ACO Simulation Implementation

References

[1] Abdel-Karim, B., Benlian, A., & Hinz, O. (2020). The predictive value of data from virtual investment communities. *Machine Learning and Knowledge Extraction*, 3(1), 1–13. https://doi.org/10.3390/make3010001

[2] Haq, S. (2023). Integration of technical indicators with support vector regression analysis for improved stock market prediction. *Journal of Research, Technology, and Development*, 6(10), 2668. https://doi.org/10.53555/jrtdd.v6i10s(2).2668

[3] Ma, F., Wahab, M., & Chevallier, J. (2022). A tug of war of forecasting the US stock market volatility: Oil futures overnight versus intraday information. *Journal of Forecasting*, 42(1), 60–75. https://doi.org/10.1002/for.2903

[4] Marinovic, I., Ottaviani, M., & Sørensen, P. (2011). Modeling idea markets: Between beauty contests and prediction markets. *Routledge Studies in Business Ethics*, 24–37. https://doi.org/10.4324/9780203815526-6

[5] Shiba, S., Cuñado, J., & Gupta, R. (2022). Predictability of the realised volatility of international stock markets amid uncertainty related to infectious diseases. *Journal of Risk and Financial Management*, 15(1), 18. https://doi.org/10.3390/jrfm15010018

[6] Guariglia, A., & García-Vega, M. (2007). Volatility, financial constraints, and trade. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.1025447

[7] Opschoor, A., Van Dijk, D., & Van Der Wel, M. (2017). Combining density forecasts using focused scoring rules. *Journal of Applied Econometrics*, 32(7), 1298–1313. https://doi.org/10.1002/jae.2575

[8] Pennock, D., Lawrence, S., Nielsen, F., & Giles, C. (2001). Extracting collective probabilistic forecasts from web games. *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 174–183. https://doi.org/10.1145/502512.502537

[9] Ishii, A., & Kawahata, Y. (2018). Sociophysics analysis of the dynamics of peoples' interests in society. *Frontiers in Physics*, 6. https://doi.org/10.3389/fphy.2018.00089

[10] Kuzheliev, M., , ., Boldova, A., Zhytar, M., & Stabias, S. (2019). Modeling of structural and temporal characteristics in the corporate securities market of Ukraine. *Investment Management and Financial Innovations*, 16(2), 260–269. https://doi.org/10.21511/imfi.16(2).2019.22

[11] Abernethy, J., Chen, Y., & Vaughan, J. (2011). An optimization-based framework for automated market-making. *Proceedings of the 12th ACM*

conference on Electronic commerce, 297–306. https://doi.org/10.1145/1993574.1993621

[12] Agrawal, S., Delage, E., Peters, M., Wang, Z., & Ye, Y. (2011). A unified framework for dynamic prediction market design. *Operations Research, 59*(3), 550-568. https://doi.org/10.1287/opre.1110.0922

[13] Chen, Y., & Vaughan, J. (2010). A new understanding of prediction markets via no-regret learning. *Proceedings of the 11th ACM conference on Electronic commerce*, 189–198. https://doi.org/10.1145/1807342.1807372

[14] Iyer, K., Johari, R., & Moallemi, C. (2010). Information aggregation in smooth markets. *Proceedings of the 11th ACM conference on Electronic commerce*, 199–206. https://doi.org/10.1145/1807342.1807373

[15] Kroer, C., Dudík, M., Lahaie, S., & Balakrishnan, S. (2016). Arbitrage-free combinatorial market making via integer programming. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 161–178. https://doi.org/10.1145/2940716.2940767

[16] Lekwijit, S., & Sutivong, D. (2018). Optimizing the liquidity parameter of logarithmic market scoring rules prediction markets. *Journal of Modelling in Management, 13*(3), 736-754. https://doi.org/10.1108/jm2-06-2017-0066

[17] Shamsi, M., & Cuffe, P. (2022). Prediction markets for probabilistic forecasting of renewable energy sources. *IEEE Transactions on Sustainable Energy, 13*(2), 1244-1253. https://doi.org/10.1109/tste.2021.3112916