Forex Value Prediction

A MACHINE LEARNING BASED APPROACH TO FOREX TRADING

Arthur Sumague, Weiwei Wang, Zhaoye Liu, Stephen Bonsu

Data 608 - Developing Big Data Applications

2025-04-10

Contents

1.0: Problem Statement and Results

1.1: PROBLEM STATEMENT

In an increasingly globalized world marked by rising migration and international commerce, individuals and businesses frequently face the challenge of deciding when and which currencies to exchange. While the U.S. dollar ($USD) is often treated as the global standard, ongoing economic and political fluctuations, particularly under new U.S. administrations, continue to introduce volatility into the global foreign exchange (forex) market.

This uncertainty directly affects people who transfer money internationally or travel abroad, as they seek to maximize the value of their currency exchanges. To help navigate this issue, we propose developing a predictive model that estimates the daily exchange rates of major currency pairs. For our prototype, we will focus on the three most frequently traded currency pairs in the forex market: EUR/USD, JPY/USD, and GBP/USD [10].

By applying this model, we aim to provide users with actionable insights that support smarter currency exchange decisions, whether for personal or business use.

1.2: RESULTS

In evaluating the forecasting performance of our model, we tested several configurations of the Prophet time series model using the EUR/USD exchange rate, specifically targeting the prediction of the *High* and *Low* daily price of the Forex-Trading Pair. In our tests we show 4 possible models. We have evidence to show that including sentiment polarity scores into the model decreases the performance of the model (Table 2). From our tests, a base model off just financial data performs at RMSE = 0.13363 and a sentiment-enhanced model performs at RMSE = 0.13455. There is a marginal improvement for when sentiment is excluded from the model. Based on these findings, our deployed application includes both the baseline and sentiment-enhanced models, enabling users to compare results and make informed decisions according to their preferences.

A model is individually trained on each Forex ticker, for both targets FX-trading-pair ratio's daily *High* and *Low*. However, to generate forecasts dependent on external regressors (please see Section 2.5.4: Data Output for more information), we must build individual Prophet forecasting models, which generates forecasts for our external regressors which is then collected and used to forecast our targets.

Our application is deployed on 3 EC2 instances, on which a data-scraper compute instance is triggered hourly to aggregate news-article data (Fig 1). A second compute instance handles most ETL processing, which is triggered every 2 hours to get the most

up-to-date financial data from Yahoo Finance. Data is served by a third EC2 instance to host our front-end StreamLit app.

2.0: Data Engineering Life-Cycle

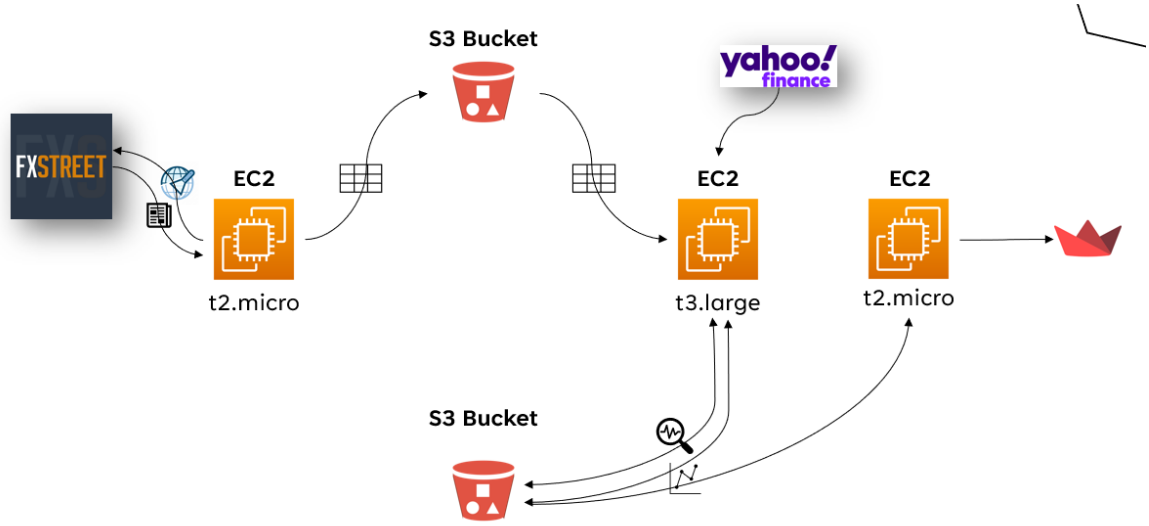


FIGURE 1: Data Engineering Life-Cycle of FX-Predict on Amazon Web Services.

2.1: DATA GENERATION

We find that the application deployed on Amazon Web Services (AWS) constitutes 3 compute instances and 2 storage instances. The pipeline begins on the left hand side of the figure (Fig. 1); displaying our primary source of news information from FXStreet.com [1]. Our scraper, consisting of python libraries selenium and BeautifulSoup, with aid from Regular Expression to navigate through the HTML code and hosted on the EC2 via ChromeDrivers to navigate through web-pages via scrolling and closing popups to access news article links and their associated text-body. The raw scraped data consisting of news article publishing datetime, title, link, and text-body are stored into an S3 Bucket as a .parquet file. This process is repeated via a cron-job on the EC2 to run hourly, everyday.

This process could be better streamlined via server-less solutions provided by AWS, such as using a lambda function to host our compute but we came across the limitation of <250 mb uncompressed file allocated to lambda functions. Our file-size did not meet this requirements because of the size of the ChromeDriver, and the associated libraries to run our python web-scraper. We understand we could have simply made a custom lambda layer to run our dependencies or used a DockerImage. Instead of a cron-job to initialize our scraper-program, we could have implemented Amazon EventBridge to trigger the lambda periodically. Due to the focus of engineering the backend of our

application and model-optimization, we were constrained of time for implementing these solutions. These would provide a more cost-effective method of running our program on the cloud.

2.2: DATA INGESTION

The process of curating and formatting data from FXStreet.com [1] and Forex (FX)-pair trading ratios from Yahoo Finance via yfinance API [2] required a two-step approach. Firstly, scrape and temporarily store news-article data. Secondly, when we generate the daily news-sentiment, we need to map FX-pair trading ratio data to the news data. Our primary source for up-to-date and reliable data in this regard is sourced from yfinance. Conveniently; when we aggregate our news data to daily-sentiment (using ProsusAI's finBERT model via HuggingFace [3]), we map the daily-sentiment to daily *High, Low, Open,* and *Close* data of our respective FX-pair. Furthermore, from advice provided by peers with similar projects, we calculated simple moving average (window of 20 dates, adjusted to 5 days for trailing dates) for feature engineering.

To achieve hosting ProsusAI's finBERT model on our cloud compute instance; we had to expand our compute power, RAM and file storage. Due to the additional dependencies, such as python libraries torch of ~2 Gb of file storage, we opted for a t3.large, a compute instance of 8Gb RAM, 25 Gb of storage with 2 vCPUs [4] with an on-demand cost of $0.0835/hr . Data ingestion is mostly handled by this EC2.

Theres a whole host of other features we could have implemented, but due to time constraints we were unable to implement them. These other features are explored in section 3.0. Limitations and Next Steps.

2.3: DATA STORAGE

As outlined in Figure 1; we used 2 S3 buckets, with our data in .parquet files for a compressed size and quicker access than .csv files. Although this process could be further streamlined using Relational Databases (RDS) such as a PostgreSQL database. Despite the use of S3 Buckets, that rewrite our entire file every-time we append new rows as the data-frames being updated real-time, the sizes of our compressed .parquet files are at maximum 734.8 kb, and that's the raw news-article dataset. Our S3 bucket that holds our raw news-article data is 1.8 Mb (top bucket Fig 1), and 50.3 kb for the S3 bucket that holds our processed data.

2.4: DATA TRANSFORMATION

Overall, the data transformation consists of; 1. Generating sentiment from each news article, 2. Aggregate to average-daily news sentiment [4], 3. Map the FX-pair data to each date, 4. Calculate the simple moving average [5]. This pipeline then pushes the data to our modelling scripts that will fit the data to a Prophet model and forecast the 7-day (one week) FX-pair trading ratio.

2.4.1: FinBERT Sentiment Analysis

ProsusAI's finBERT model was recommend by our TA, Amir Mirzai Golpayegani. We utilized this model to generate sentiment polarity, the probability of a text document being classified into positive, neutral or negative, quantified into a scale from -1 to 1 for negative to positive respectively. We opted for the use of sentiment polarity because of the richer information that is brought by continuous data, rather than categorical data. We did not test our hypothesis however, due to time constraints of the project. FinBERT reported accuracy from Prosus is 97% accuracy, with an accuracy for separating positive-negative, negative-neutral and positive-neutral are 98.7%, 94.2% and 75.2% respectively [6]. We had hoped that the finBERT model would be able to extract sentiment in regards to FX-pair trading ratios, but we found that most daily-sentiment polarity scores would be evaluated between 0 – 0.90 (neutral to positive). The error of this is further explained in Section 3.0: Limitations and Future Steps. Nonetheless, we find the noise introduced through the daily-sentiment polarity to cause a slight underperformance in our Prophet model. As observed in Section 2.5: Data Output, the model performance slightly worse when sentiment polarity and sentiment polarity simple moving average (SMA) and the historical data of the target are used in forecasting (RMSE = 0.1361) (Fig. 3) compared to when it is only the historical data of the target (RMSE = 0.1338) (Fig. 4) or when all additional predictive/ regressor variables are introduced except sentiment data (RMSE = 0.1336) (Fig. 5). Because of these tests, we found that the best viable solution was to offer (by default in our front-end of the application) the model that best performed (RMSE = 0.1336). Including sentiment-data in forecasts is toggle-able by the user.

2.4.2: Aggregating News Article Sentiment to Daily Sentiment

Mapping sentiment to ticker values was an interesting challenge. The issue lies in that some articles would be published on weekends or non-trading days. Therefore, for those days, we wouldn't have a ticker value to associate with that sentiment. The possible solutions were if published on a weekend; 1. Map to previous Friday's values, 2. Map to the following Monday's values, 3. Discard the sentiment. We find in Table 1, that the correlation is maximized when we exclude the weekend sentiment. The reason we only tested the correlation of sentiment with, the lag of itself, *High,* and *Low*, is essentially because *Sentiment Polarity* is proposed to serve as a strong regressor to our targets, *High* and *Low*. We now know this hypothesis to be false with our current model & application.

Table 1. Pearson correlation of *lag of Sentiment Polarity*, *Sentiment Polarity* and *High*, and *Sentiment Polarity* and *Low* of Differing Methods of Handling Weekend News Sentiment. Lag of Sentiment Polarity is calculated via from time *t, t-1* and *t+1*. Pearson correlation was calculated using `.corr()` function offered through Pandas' DataFrame object. In context, *High* and *Low* are our targets in the application.

2	**Pearson Correlation of Lag of**	**Pearson Correlation of *Sentiment***	**Pearson Correlation of *Sentiment***	**Total Correlation Sum(|correlations|)**

	***Sentiment Polarity* (t-1), (t+1)**	***Polarity* and *High***	***Polarity* and *Low***	
Pushed to previous Friday	0.042087	-0.092077	-0.074192	0.208356
Pushed to following Monday	0.042087	-0.090461	-0.073865	0.206413
Discarded	0.07055	-0.084655	-0.067640	0.222846

2.4.3: Simple Moving Average

We decided to include simple moving in hopes to improve the model performance in forecasting. As the results in Section 2.5: Data Output; the model improves marginally from an RMSE of 0.13384 (Fig. 3) to 0.1336 (Fig. 4). OLS regression results (Supplementary Material 1 and Supplementary Material 2) show a significant relationship between the target and their simple moving average. Interestingly, the least significant relationship (highest p-value via one-tailed t-test) is *Sentiment Polarity,* but improves with *Sentiment Polarity Simple Moving Average* (`sentiment_polarity_sma`). All other regressors show non-significant relationship with the target.

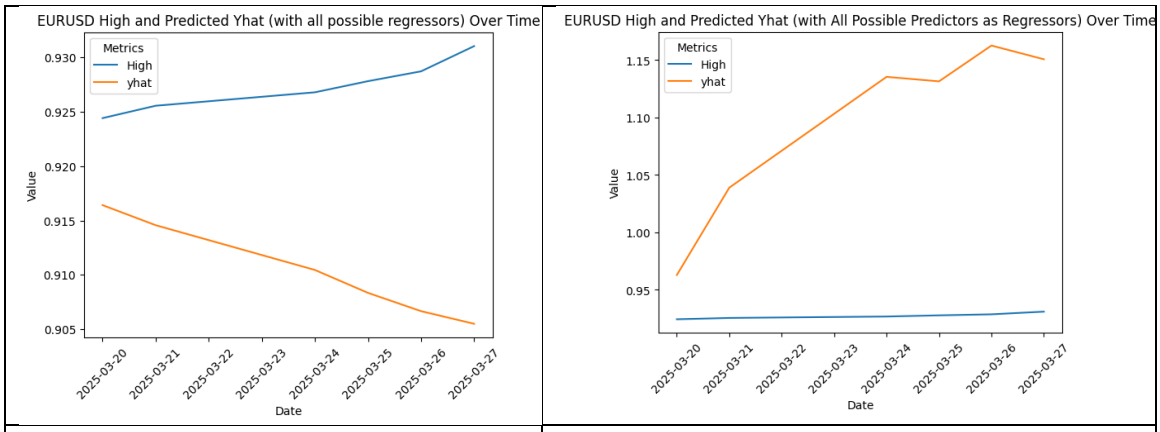
We could have explored other feature engineering implementations in hopes to improve our model. These further steps are explored in Section 3.0: Limitations and Next Steps.

2.5: DATA OUTPUT

2.5.1 Prophet vs. SARIMAX

We initially presented our results with a Prophet model. Prophet was built off scikit-learn's API and is an easy-to-integrate modelling platform [8]. Through inspiration from peers who presented stock price forecasting with SARIMAX, we experimented with this model as well. SARIMAX (Seasonal Autoregressive Integrated Moving Average + exogenous variables), built off StatsModels python library, offered a customizable forecasting model. Hyperparameter tuning was determined using Autocorrelation, Partial-Autocorrelation and Seasonal Decomposition graphs [7], explored in `/testers/news_article_modelling.ipynb` in the GitHub repository or in Supplementary Material Figures 1 - 5. As you can see in Figure 2, that the performance of Prophet and SARIMAX in forecasting *High* from March 20 – 27, 2025, the RMSE is lower (better) for Prophet than hyperparameter-tuned-SARIMAX.

Prophet	**SARIMAX**



RMSE = 0.1345589487230545	RMSE = 0.4286046492239469

Figure 2. Performance of Prophet vs. SARIMAX Model Performance. Both are tested from March 20 – 27 forecasting *High* Forex-Pair Trading Ratio.

2.5.2: Introduction to the Prophet Model

Prophet, developed by Facebook, is an open-source time series forecasting model based on additive decomposition [8]. It assumes that time series data is composed of trend, seasonality, and holiday effects [8]. Prophet automatically fits these components to generate forecasts. It is particularly well-suited for time series with strong seasonal patterns, and has been widely applied in areas such as financial markets and sales forecasting [8].

2.5.3: Why Prophet

In addition to its strong performance in testing, we have some theoretical reasons to choose Prophet model:

- Strong trend modeling capabilities: Our exchange rate data is long-term and influenced by macroeconomic factors, exhibiting a clear trend that Prophet can effectively capture.
- Insensitive to missing and anomalous data: Financial data often contains missing values or outliers, and Prophet handles these issues well. This also helps in maintaining our web app, reducing the need for special handling of missing data. This is more an error-handling fail-safe; we ensured null values were dropped prior to pushing the data to be modelled.
- Strong interpretability for seasonal and cyclical time series data: Our exchange rate data exhibits clear seasonality and cyclic patterns, and Prophet excels in modeling and interpreting such data. For proof of seasonality in our data, please see Supplementary Figures 1 – 5 in Section 6.0: Supplementary Material.

- Ability to automatically detect and define change points such as holidays and policy shifts: Financial data is often impacted by policy changes or other sudden events. Prophet can automatically detect these change points, adapt to them, and provide more accurate predictions.

2.5.4: Model Training and Model Performance

In this part we will show some examples of model training, testing and forecasting performance using different regressors.

To better reflect the forecasting process of our model and our web app, in all the tests we choose data prior to March 19, 2025, as our training set and data from March 20, 2025, onward as the test set. And we implemented the simple moving average as additional regressors to reduce noise in the data and highlight the trend in the data. We implement the model with date and high price of the forex (EUR/USD) as our independent variables in the test examples. And following we will show our model test examples with different external regressors.

Prophet model with no external regressors and only high price:

First, we only use our independent variables to train the model, and the prediction plot:

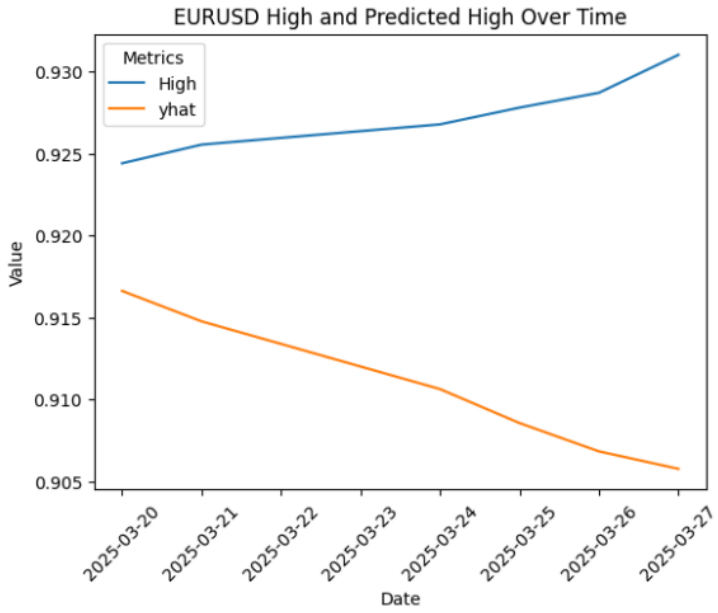


Fig. 3. Prophet performance with no external regressors. The RMSE of this prediction is 0.13384402252790 76.

Prophet model with sentiment polarity as an external regressor:

We predict sentiment polarity through financial texts such as financial news and reports, and we use the predicted sentiment polarity as an external regressors of the Prophet model. The performance of this model:

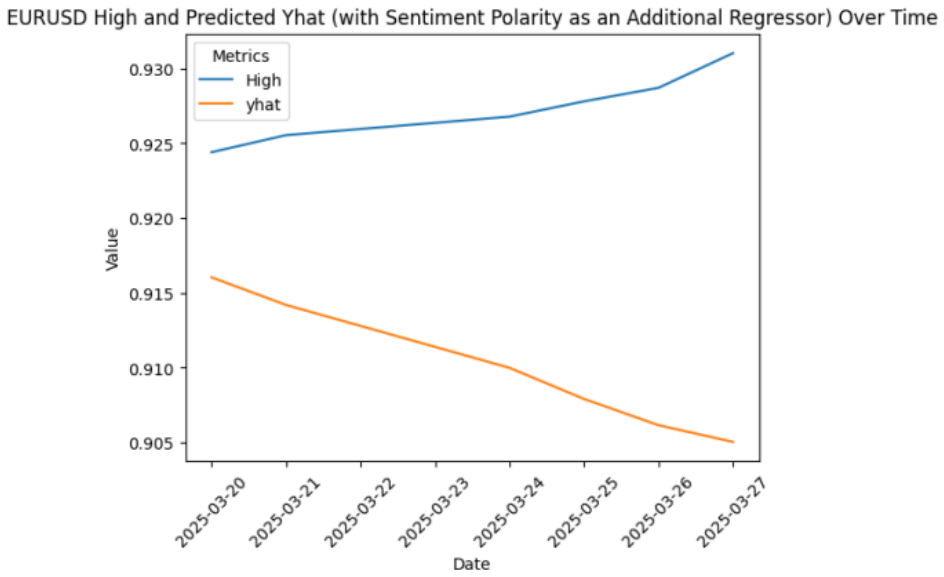


Fig. 4. Prophet Performance with Sentiment Polarity as an External Regressor. The RMSE of this prediction is 0.13613003827216 61 .

Prophet model with simple moving averages as additional regressors, excluding sentiment data

In this model, we selected several external regressors to help predict the High variable in the EUR/USD exchange rate, including predicted High, Low, Open, and Close price and their forecasted simple moving averages. And these regressors were not taken as actual future values but were generated by individually training separate Prophet models for each of them and then forecasting their future values. The plot is shown below:

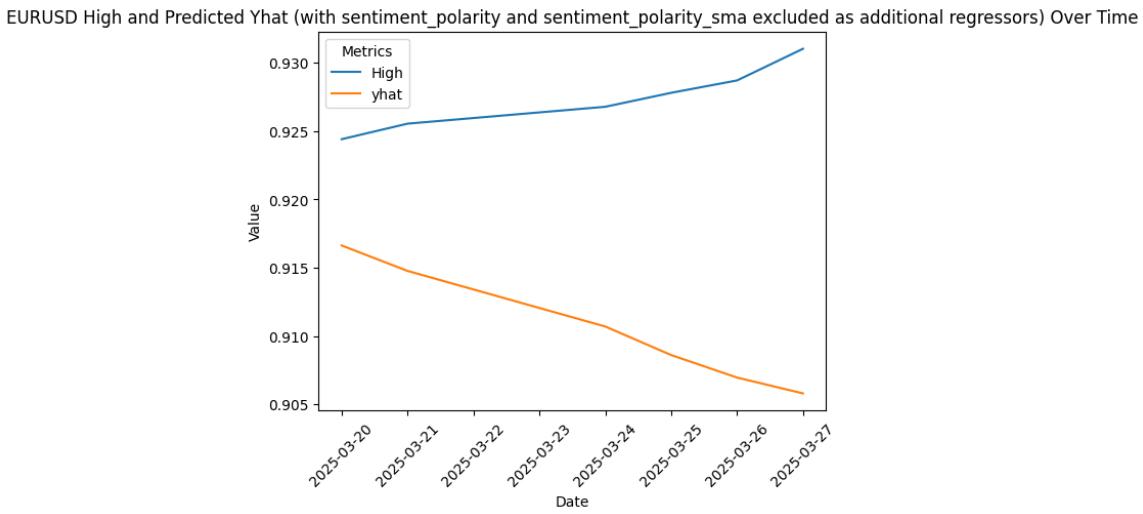


Fig. 5. Prophet Performance with model with simple moving averages as additional regressors, excluding sentiment data. The RMSE is 0.13363061582894326.

Prophet model with simple moving averages as additional regressors, including sentiment data

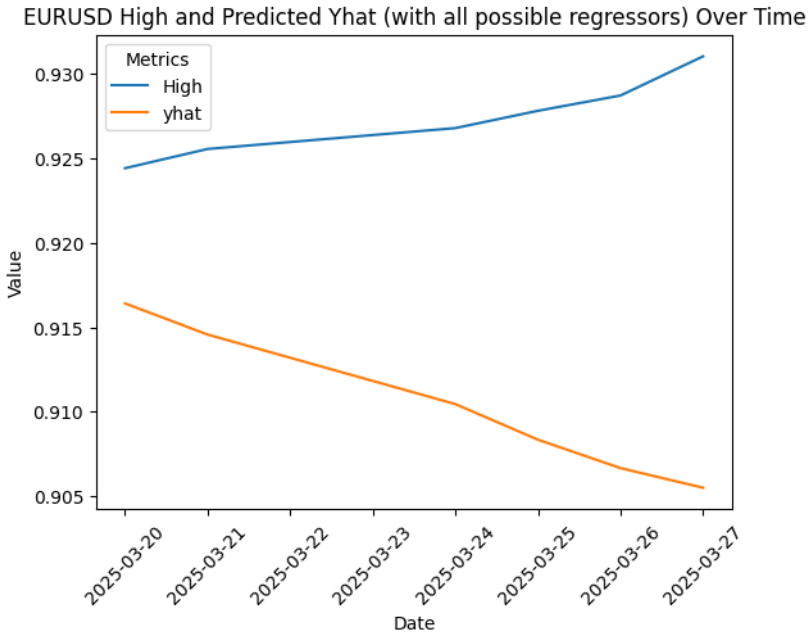


Fig. 6. Prophet Performance with model with simple moving averages as additional regressors, including sentiment data. The RMSE is 0.1345589487230545.

Table 2: Summary of Model Optimization RMSE scores with Additive Model Equations through Prophet.

	High	***High* + *Sentiment Polarity* + *Sentiment Polarity (SMA)***	***High* + *Low* + *Open* + *Close* + *High (SMA)* + *Low (SMA)* + *Open (SMA)* + *Close (SMA)* +**	***High* + *Low* + *Open* + *Close* + *High (SMA)* + *Low (SMA)* + *Open (SMA)* + *Close (SMA)* + + *Sentiment Polarity* + *Sentiment Polarity (SMA)***
RMSE	0.13384	0.13613	0.13363	0.13455

In these tests, the trend of the predicted values did not align with the trend of the actual values. Moreover, the model that included sentiment polarity as a regressor performed worse than the model without it. However, based on our observations over several days of real-world forecasting, the model with sentiment polarity as a regressor often provided more accurate predictions for other Forex Pairs we included (USD/JPY & EUR/USD). Therefore, in the web app, we included both models to allow users to either choose for themselves or compare the results to make their own judgment.

2.6: DATA SERVING

In our Streamlit web app, we include the exchange rates of the Euro, Japanese Yen, and British Pound against the US Dollar. At the top of each exchange rate forecasting page, there is a checkbox that allows users to toggle whether to include the sentiment factor—extracted from financial news and reports—in the predictions. This feature enables users to view and compare the forecasting results and recommendations with and without sentiment taken into account.

What follows is a direct buy/sell suggestion based on our 7-day forecast of market trends, indicating whether we recommend buying or selling a particular currency in the given exchange rate pair. These determinations are within the full confidence of our model. To explain, we give our users an expected range of the daily maxima (High) and minima (Low) of the Forex trading pair ratio. The reason for this is through the advice of Dr. Bonnel; the longer the period/duration in time-series forecasting (from seconds to days), the more accurate your forecasts become. Our proposed set of actions for the user was determined in this manner; in the case of EURUSD (USD per EUR) if the forecasted *Low Simple Moving Average* for the next 7 days is higher than the current value; tell the user to sell USD (because the value decreases) and buy EUR (because the value increases).

Right after this, we present a bar chart showing historical exchange rate movements over the past few months alongside our forecasted changes. This helps users

understand the rationale behind our recommendation and gives them the opportunity to analyze the data themselves and make more detailed, personalized long-term investment plans.

Next, we display the Root Mean Squared Error (RMSE) of historical forecasts along with a line chart showing its trend, to demonstrate the reliability of our predictions and help users assess how much they can trust our model's recommendations.

3.0: Limitations and Next Steps

3.1: FinBERT

The first point of optimization is to optimize the usage of FinBERT in extracting sentiment from our scraped news articles. Further testing with categorical outputs (negative, neutral, positive) can be tested and see if using sentiment polarity introduces noise in the data, impairing the forecasts. We can see that this might be the case, with evidence in Supplementary Material 1 and 2, OLS Regression output indicates that the simple moving average of sentiment polarity have a lower p-value than sentiment polarity itself.

Additionally, we should test other sentiment-extracting models. We believe the biggest limitation in finBERT is that is unable to discern sentiment in the context of the trading pair. For example; EURUSD ticker is a ratio of USD numerator, and EUR denominator. If the sentiment displays positive sentiment for EUR value, it should actually give a score of negative sentiment, as EUR is in the denominator and will drive the ratio downwards in relation to the ratio's value. Another point of confusion for the model is that if the ratio decreases, it actually means an increase in value for the currency. To explain, take for example as the EURUSD decreases, this mean the ratio of USD per 1 EUR decreases, which indicates an increase in the USD value. Further uses can be an implementation of Agentic AI, utilizing LLM's and prompt engineering to provide context to LLM models to extract the precise sentiment analysis we are seeking in our use-case.

3.2: Prophet

Although the current use of the Prophet model with external regressors has improved the prediction value to some extent compared to other time series forecasting models, there are still limitations of our Prophet model. Firstly, Prophet is an additive model primarily built to capture trend and seasonality [8], which may not sufficiently reflect the nonlinear behaviors common in complex financial markets. In addition, the model does not manually define or incorporate important external factors such as macroeconomic indicators, geopolitical events, or central bank policies that may significantly impact exchange rate movements.

Next steps include experimenting with more advanced time series forecasting models such as ARIMA or SARIMA to better capture nonlinear relationships; exploring rolling window–based model updating strategies to enhance adaptability to real-time market changes and improve the accuracy of long-term forecasts; and analyzing policy shifts and economic events extracted from financial text data to manually define change points in the model, thereby improving its ability to capture key turning points and boosting overall forecasting performance. A possible solution can be to use an ensemble of model in our application with other ARIMA/SARIMA based models.

Additionally, we can include other factors in our model such as but not limited to:

- Volatility
- Relative Strength Index
- Bollinger Bands
- Moving Average Convergence Divergence
- Average True Range
- GDP (Country of)
- Unemployment Rates (Country of)
- Interest Rates (Country of)

3.3: AWS Serverless Approach

The biggest flaw within our AWS Cloud Application is that its heavily reliant on sever compute instances. We can cut costs by a large margin by not having a running compute service 24/7. As foreshadowed in Section 2.1, we can opt to use Lambda functions and AWS EventBridge to trigger our scraper. We encountered multiple issues, mostly in the file-size limitation of lambda functions. However, solutions are existent online for hosting ChromeDrivers on a lambda function, or overcoming the file-size limitation of 250Mb on lambdas through a DockerImage.

We could also utilize AWS SageMaker to train and host our AI model on the AWS platform. Alongside which, AWS Glue can be used to monitor the resources used throughout our ETL process. To coordinate our data scraper, then our ETL process altogether, AWS Step Functions can be used in that regard. Potentially for querying in our S3 buckets with our .parquet files; we could have used AWS Athena to query.

AWS hosts a large tool-kit that would be perfect in streamlining our application. We are unfortunately pressed for time to exploring these options.

4.0. Conclusions

Building this application deepened our understanding of the complete machine learning deployment pipeline—from data scraping and sentiment analysis to model evaluation and front-end deployment. Through this project, we learned to work with various AWS services such as EC2 and S3 to manage compute resources and data storage.

We also explored the limitations of serverless computing when handling large dependencies like ChromeDriver and PyTorch, which led us to appreciate the balance between performance and infrastructure design.

On the modeling side, experimenting with the Prophet model taught us both its strengths in trend detection and its limitations in handling nonlinear behaviors. By integrating sentiment analysis using finBERT and comparing different model configurations, we gained insight into the impact of financial news on currency forecasting and how auxiliary signals can both enhance and complicate predictions.

Deploying the model via Streamlit allowed us to translate our backend work into a user-friendly interface, offering real-time forecasts with adjustable sentiment-based toggling. The experience taught us not only how to serve predictions but also how to present results transparently, providing users with RMSE scores and historical trends for better decision-making.

Overall, this project has enhanced our technical fluency across data engineering, model tuning, and web deployment, while also highlighting practical limitations and future improvement opportunities in time series forecasting for financial markets.

5.0. Code Resources

Working code is hosted on AWS Learner Lab. However, we invite interested parties to view the GitHub repository [9].

- **To run locally:** all code can be started with `/app.py`. To run the streamlit, run the command `streamlit run app.py` from the main project folder. Please install the dependencies in `requirements.txt`.
- **To view the AWS code:** we have provided readers with sub-directory `/AWS_adapted_py_programs` to view code. Adaptations are mostly for reading and writing files from S3 buckets.
- **Dependencies and Requirements:** ChromeDrivers are sourced from the official site from Google Chrome [11]. Version depends on the Selenium version used. The version used in this project was 113.0.5672.63

6.0: Supplementary Material

Supplementary Material 1: OLS Regression output with *High* as Target.

```
 Intercept                   0.004111
sentiment_polarity          -0.002877
Open                       -17.049263
Close                       17.981852
```

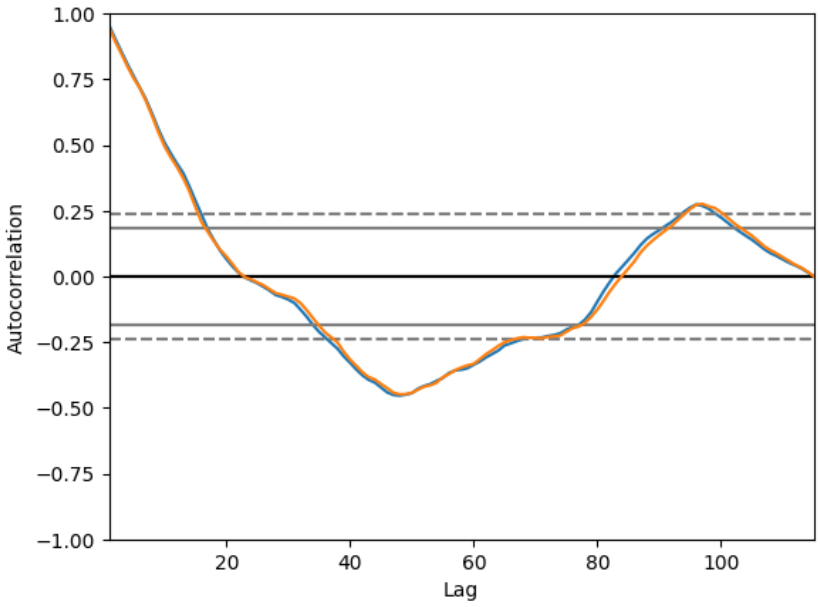
```
sentiment_polarity_sma      0.019109
High_sma                    1.511537
Low_sma                     1.251182
Open_sma                  125.622495
Close_sma                -128.331345
dtype: float64
                            OLS Regression Results

==============================================================================
==========
Dep. Variable:                    High   R-squared:
0.978
Model:                             OLS   Adj. R-squared:
0.976
Method:                  Least Squares   F-statistic:
579.5
Date:                 Mon, 07 Apr 2025   Prob (F-statistic):
8.52e-84
Time:                         21:49:36   Log-Likelihood:
514.95
No. Observations:                 115   AIC:
-1012.
Df Residuals:                     106   BIC:
-987.2
Df Model:                           8

Covariance Type:            nonrobust

==============================================================================
======================
                               coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
----------------------
Intercept                    0.0041      0.021      0.196      0.845
-0.037       0.046
sentiment_polarity          -0.0029      0.003     -0.862      0.390
-0.009       0.004
Open                       -17.0493     33.389     -0.511      0.611
-83.246      49.147
Close                       17.9819     33.393      0.538      0.591
-48.224      84.187
sentiment_polarity_sma       0.0191      0.021      0.899      0.371
-0.023       0.061
High_sma                     1.5115      0.450      3.361      0.001
0.620       2.403
Low_sma                      1.2512      0.560      2.236      0.027
0.142       2.361
Open_sma                   125.6225    191.130      0.657      0.512
-253.311     504.556
Close_sma                 -128.3313    191.084     -0.672      0.503
-507.173     250.510
```

```
==============================================================================
==========
Omnibus:                       80.454   Durbin-Watson:
1.870
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
642.133
Skew:                           2.240   Prob(JB):
3.65e-140
Kurtosis:                      13.674   Cond. No.
2.74e+06
==============================================================================
==========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors
is correctly specified.
[2] The condition number is large, 2.74e+06. This might indicate
that there are
strong multicollinearity or other numerical problems.
```

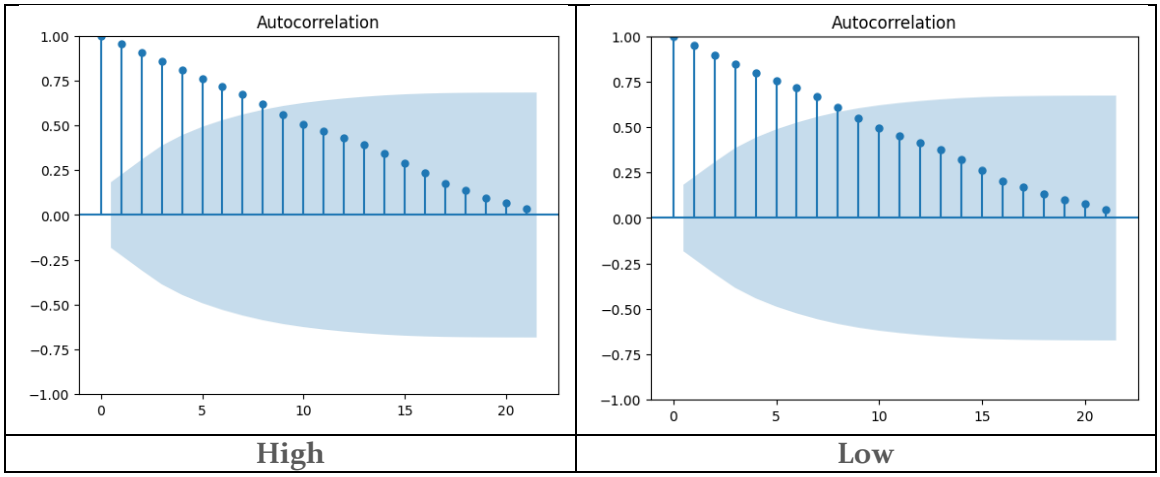
Supplementary Material 2: OLS Regression output with *Low* as Target.

```
Intercept                   0.010658
sentiment_polarity          0.001599
Open                      -22.566623
Close                      23.484464
sentiment_polarity_sma      0.025775
High_sma                    0.610194
Low_sma                     1.581789
Open_sma                 -255.934168
Close_sma                 253.792727
dtype: float64
                            OLS Regression Results

==============================================================================
==========
Dep. Variable:                    Low   R-squared:
0.978
Model:                            OLS   Adj. R-squared:
0.977
Method:                 Least Squares   F-statistic:
600.3
Date:                Mon, 07 Apr 2025   Prob (F-statistic):
1.36e-84
Time:                        21:49:36   Log-Likelihood:
520.55
No. Observations:                 115   AIC:
-1023.
Df Residuals:                     106   BIC:
-998.4
```

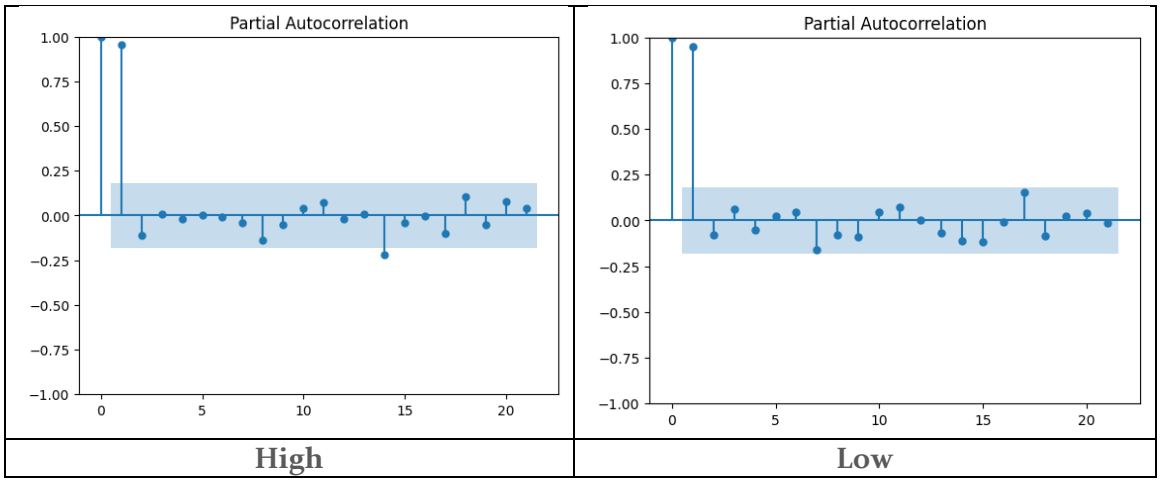
```
Df Model:                           8

Covariance Type:            nonrobust

==============================================================================
======================
                                coef    std err          t      P>|t|
[0.025      0.975]
------------------------------------------------------------------------------
----------------------
Intercept                    0.0107      0.020      0.535      0.594
-0.029       0.050
sentiment_polarity           0.0016      0.003      0.503      0.616
-0.005       0.008
Open                       -22.5666     31.802     -0.710      0.480
-85.616      40.483
Close                       23.4845     31.806      0.738      0.462
-39.574      86.542
sentiment_polarity_sma       0.0258      0.020      1.273      0.206
-0.014       0.066
High_sma                     0.6102      0.428      1.425      0.157
-0.239       1.459
Low_sma                      1.5818      0.533      2.968      0.004
0.525       2.639
Open_sma                  -255.9342    182.043     -1.406      0.163
-616.853     104.985
Close_sma                  253.7927    182.000      1.394      0.166
-107.039     614.624
==============================================================================
==========
Omnibus:                       37.224   Durbin-Watson:
1.671
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
71.098
Skew:                          -1.376   Prob(JB):
3.64e-16
Kurtosis:                       5.696   Cond. No.
2.74e+06
==============================================================================
==========

Notes:
[1] Standard Errors assume that the covariance matrix of the errors
is correctly specified.
[2] The condition number is large, 2.74e+06. This might indicate
that there are
strong multicollinearity or other numerical problems.
```



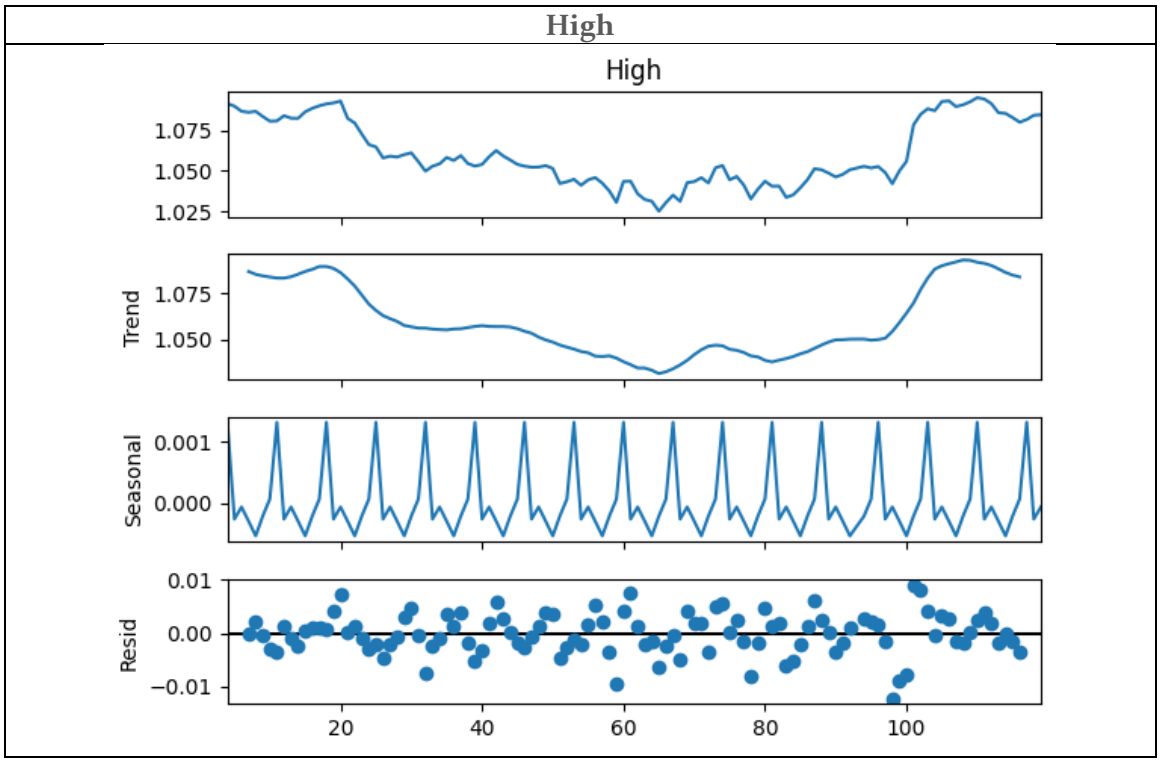
Supplementary Figure 1. Autocorrelation Plot of Target Variables: *High (Yellow)* and *Low (Blue)*. There is a seasonality component to our target variables. There is a damped sinusoidal wave with one frequency occuring every ~100 lags.



Supplementary Figure 2: Autocorrelation Function of Targets *High* and *Low*. A steep linear decrease in the autocorrelation function graphs display a strong, but temporary, autocorrelation within 20 time-periods (dates in our case).

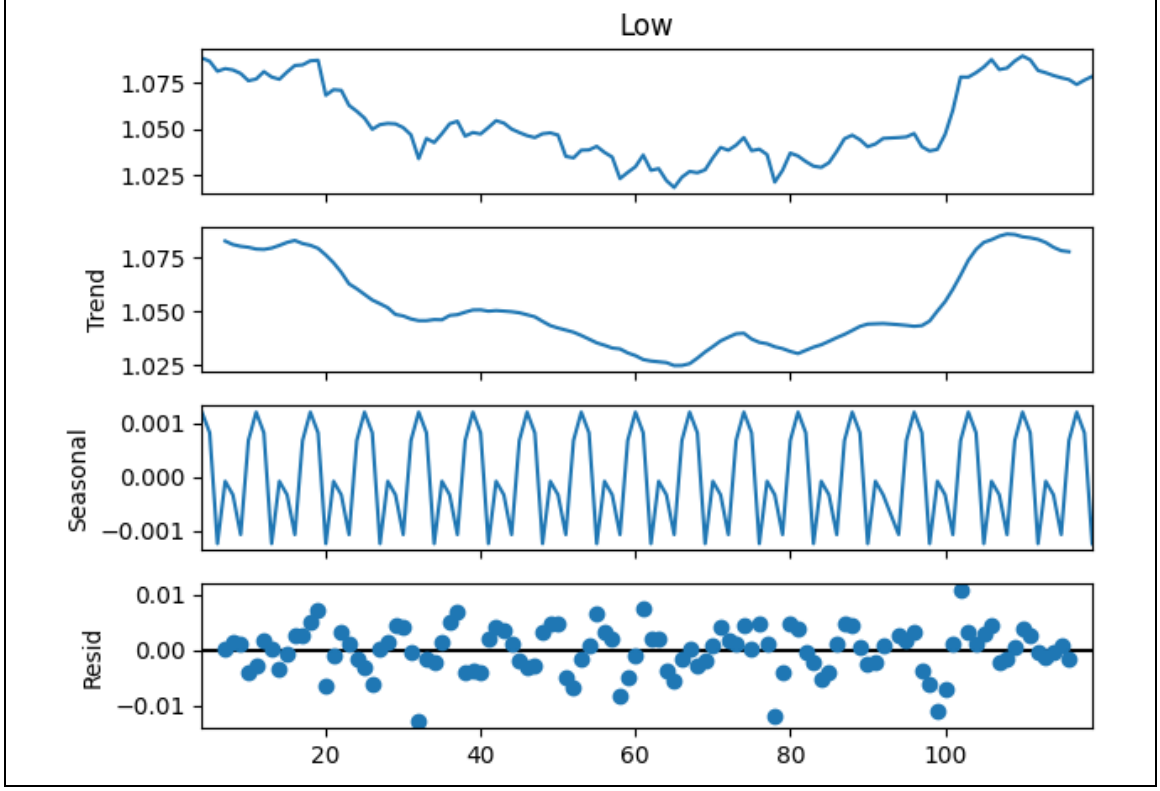


Supplementary Figure 3: Partial-Autocorrelation Plots of Targets *High* and *Low*.



Supplementary Figure 4: Seasonal Decomposition of Target *High*. The following decomposition was conducted with an additive model, for a period of 7 days.

Low


Low
1.075
1.050
1.025
Trend
1.075
1.050
1.025
Seasonal
0.001
0.000
−0.001
Resid
0.01
0.00
−0.01
20
40
60
80
100

Supplementary Figure 5: Seasonal Decomposition of Target *Low*. The following decomposition was conducted with an additive model, for a period of 7 days.

7.0: References

1. "FXStreet," *FXStreet*. [Online]. Available: https://www.fxstreet.com/. [Accessed: Apr. 10, 2025].

2. "yfinance," *PyPI*. [Online]. Available: https://pypi.org/project/yfinance/. [Accessed: Apr. 10, 2025].

3. "FinBERT – Financial Sentiment Analysis with BERT," *Hugging Face*. [Online]. Available: https://huggingface.co/ProsusAI/finbert. [Accessed: Apr. 10, 2025].

4. "Amazon EC2 Instance Types," *Amazon Web Services (AWS)*. [Online]. Available: https://aws.amazon.com/ec2/instance-types/t3/. [Accessed: Apr. 10, 2025].

5. "Moving Average with Pandas," *FavTutor*. [Online]. Available: https://favtutor.com/articles/moving-average-pandas/. [Accessed: Apr. 10, 2025].

6. "FinBERT: Financial Sentiment Analysis with BERT," *Prosus*. [Online]. Available: https://www.prosus.com/news-insights/group-updates/2020/finbert-financial-sentiment-analysis-with-bert. [Accessed: Apr. 10, 2025].

7. "Time-Series Forecasting Using SARIMA in Python," *Medium*. [Online]. Available: https://medium.com/@tirthamutha/time-series-forecasting-using-sarima-in-python-8b75cd3366f2. [Accessed: Apr. 10, 2025].

8. "Quick Start: Python API," *Facebook Prophet*. [Online]. Available: https://facebook.github.io/prophet/docs/quick_start.html#python-api. [Accessed: Apr. 10, 2025].

9. "fxTickerPredict," *GitHub*. [Online]. Available: https://github.com/asumag4/fxTickerPredict. [Accessed: Apr. 10, 2025].

10. "Top 10 most traded currency pairs," IG, Dec. 06, 2019. [Online]. Available: https://www.ig.com/en/trading-strategies/top-10-most-traded-currency-pairs-191206. [Accessed: Mar. 11, 2025]

11. "ChromeDriver Downloads," *Chrome Developers*. [Online]. Available: https://developer.chrome.com/docs/chromedriver/downloads. [Accessed: Apr. 11, 2025].