Final Project: Bitcoin Price Time Series Analysis

Team 7

University of San Diego

ADS-506: Time Series Analysis

12/10/2023

Bitcoin Price Time Series Analysis

In the vast world that is financial investing, cryptocurrencies are a rather new, lucrative type of investment. This new addition to a vast number of financial portfolios is known as digital currency trading (Frankenfield, 2023). The chosen project focuses on leveraging time series analysis to gain insights into this type of trading— particularly Bitcoin in the regions of Saudi Arabian Riyal (SAR) and the United States (USD). The group's collective interest in financial markets and trading serves as the driving motivation for this exploration. The project encompasses various objectives focusing on probing for seasonal patterns in Bitcoin prices, developing short-term and long-term price prediction models, and investigating historical arbitrage opportunities between SAR and USD markets.

Today, financial institutions are capable of making (somewhat naïve) inferences for Bitcoin prices based on historical data. However, very little is done to truly understand influencing factors and, with a certain level of accuracy, even predict price movements. Thus, this is the gap that this project aims to fill. Possessing such capabilities would provide a financial firm that specializes in Bitcoin with a solid advantage— allowing them to become experts in understanding the market. To measure the success of the project, specific thresholds were chosen for three primary performance measures. Specifically, a minimum of 80% for the R-squared (R2) model performance value and a maximum of $300 for the Mean Squared Error (MAE) and Root Mean Squared Error (RMSE). While no significant threats are present relative to the target data itself, overfitting is a known threat to the outcome of the project. This, in turn, is an influencing factor for the selected R2 performance measure. While the group aims to create an optimal mode, we understand that such is a unlikely reality.

**Target Datasets**

The selected dataset, ‘Digital\_Currency.csv’ lengths from May 7, 2018, to January 30, 2021, and includes ten columns and 1,000 rows of digital currency trading data. Key features within the dataset such as price information, trading volume, and extensive variability in both SAR and USD values offer a rich landscape for analysis. Seven additional datasets are also used as supplementary data— offering complementary details that have a significant potential to correlate with the pricing data. Key features within these datasets include daily treasury rates, DOW Jones industrial average, daily Federal Funds rate, crude oil prices (for both target regions), inflation rates, and Economic Policy Uncertainty (EPU) Index (U.S. only). Even at first glance, the wide range of trading volumes and prices hint towards potential correlations with the diverse market conditions that are captured, providing a compelling foundation for exploring trends and patterns.

**Literature Review**

To assess the feasibility and generalizability of the project, we undergo a review of existing literature relevant to predictive modeling for Bitcoin prices.

**Main Points**

We begin by examining a study carried out by Alpha Vintage, a financial market data specialist that focuses on analyzing real time market data. Several key points emerge that highlight both the strengths and limitations of our approach. Firstly, the dataset's time-bound scope. While rich in detail, it may not fully encapsulate the broader, evolving trends in the digital currency landscape and can potentially limit the generalizability of any findings. Methodologically, the application of time series analysis via the use of ARIMA models poses significant challenges due to the volatile and often nonlinear nature of cryptocurrency markets. In turn, necessitating a nuanced understanding of these models' assumptions and limitations.

The project's objective to explore spatial dynamics and arbitrage opportunities between the SAR and USD markets is ambitious and may be constrained by the dataset's inability to fully capture the diverse factors influencing these markets. This limitation is compounded by the lack of direct data on qualitative aspects such as regulatory changes, technological advancements, and investor sentiment which are crucial in understanding Bitcoin's market dynamics. The comparative analysis of various modeling approaches, inspired by studies in traditional financial markets and stock price predictions, suggests a need for a multifaceted approach as no single model may comprehensively address the unique characteristics of cryptocurrency volatility and market behavior. Lastly, the project's effort to adapt advanced modeling techniques from other financial domains to cryptocurrency analysis is promising but may require significant modifications to effectively capture the distinct nature of cryptocurrencies. These key points underscore the complexity and challenges inherent in analyzing the dynamic (and often unpredictable) world of digital currency trading.

**Key Findings**

Though our dataset offers a rich landscape for analysis, the exploration of leveraging time series analysis for digital currency trading has revealed compelling insights that have the potential to reshape some of the elements of this project. A study on the spatial integration of European natural gas markets to financial markets offers insights into market power and spatial arbitrage (Massol & Banal-Estañol, 2018). While we focus on digital currencies, the methodology and lessons learned from this study could potentially inform our understanding of spatial dynamics within the digital currency landscape across different regions. Although Bitcoin does not have inherent capacity constraints like a pipeline or storage facility, factors like transaction speed, scalability, and network fees may be comparable considerations.

Priya & Garg (2018) present vital insights relative to the use of Autoregressive Integrated Moving Average (ARIMA) models for Bitcoin close price analysis and prediction. While such models are effective to examine seasonality and trend, they fail to predict long-term patterns accurately. Thus, these are anticipated conclusions for our project as well. Identifying optimal model parameters, dividing data into training and testing sets, and calculating Mean Percentage Error are likely to be crucial components of our findings.

In the context of stock price prediction, studies comparing various models including the

GARCH family model, Auto ARIMA model, Long Short-Term Memory (LSTM), Bidirectional Long-short Term Memory (Bi-LSTM), and LSTM neural network model demonstrated the effectiveness of quantitative investment in predicting MSFT stock prices (Luo, 2022) & (Juairiah et al., 2022). This success story aligns with our strategy and objectives of developing short-term and long-term price prediction models for Bitcoin prices. These studies underscore the importance of exploring various models in our study to enhance the accuracy of short-term and long-term price predictions for the digital currency.

**Exploratory Data Analysis**

For our exploratory data analysis, we created a series of line charts to delve into and analyze the trends present within the dataset. We began by isolating relevant numerical columns, focusing on numeric data types ('float', 'int'), and excluding binary indicators like 'close\_USD\_change,' 'close\_SAR\_change,' and the categorical column 'Day\_of\_Week\_encoded.' We then developed a loop that iterates through these chosen numeric features by creating a chart for all columns except those identified in the 'excluded\_columns' list. Each iteration generates a specific line chart, plotting the 'Date' variable on the x-axis and the corresponding numerical column on the y-axis. We customized each chart with labels and titles, rotating the x-axis labels to make them easier to read. Visualizing numeric attributes across time through line charts unveils trends, fluctuations, and patterns ingrained within the dataset.

The line charts reveal a noticeable correlation between USD/Bitcoin close price and SAR/Bitcoin close price. Despite currency variations between the US and Saudi Arabia leading to differing scales, these charts show remarkably similar trends (see Figs. 1 & 2). Movement in USD/Bitcoin close price mirrors SAR/Bitcoin close price over time, indicating synchronized behavior and suggesting a close relationship in their valuation trends. Upon examination, all charts exhibit apparent seasonal behavior except for the treasury rates (see Fig. 3), regardless of the period length. In the USD/Bitcoin and SAR/Bitcoin price charts, discernible trends and cyclicality are evident, implying consistent patterns in their valuation trends. The absence of clear seasonality in the treasury rates contrasts with the periodic patterns seen in other variables, potentially indicating a more stable and consistent trend over time in the treasury rates compared to the other economic indicators.

Once again, we focused on numeric columns to explore autocorrelation and partial autocorrelation in our dataset. We then applied differencing to these columns to understand temporal dependencies better. Using a loop for each, we generated line charts for the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) before and after differencing. Each chart displayed the lag on the x-axis and the corresponding autocorrelation or partial autocorrelation value on the y-axis, helping us analyze temporal patterns and dependencies within the dataset.

The ACF/PACF charts revealed intriguing patterns, and most displayed lags noticeably outside statistical significance (see Fig. 4). A discernible trend emerged where only positive lags were present, except for West Texas Intermediate (WTI) crude oil price change percentage, which displayed a mix of positive and negative lags, though primarily within low statistical significance areas (see Fig. 5). Across all charts, a consistent downward trend prevailed, with the initial lag holding the most significance, followed by a diminishing pattern in subsequent lags. However, the USD/Bitcoin and SAR/Bitcoin pricing charts showcased more pronounced downward slopes. The remaining charts showed slight declines.

After differencing, the ACF/PACF charts exhibited a consistent pattern. The first two to three lags were notably significant, with subsequent lags alternating between positive and negative values (see Fig. 6). These lags mostly remained within the bounds of statistical significance or exceeded them slightly. This pattern indicates that even after differencing, the time dependencies within the dataset persisted, emphasizing the importance of the initial lags in influencing the succeeding values.

The primary takeaway from these observations lies in the significance of the initial few lags across charts following differencing. While these lags displayed significant alternation between positive and negative, they predominantly adhered to or slightly exceeded the boundaries of statistical significance. This pattern indicates immediate connections between values in the dataset over short periods, emphasizing that the initial lags significantly impact the following values.

We then initiated a regression analysis for the USD/Bitcoin and SAR/Bitcoin close prices. This analysis entailed creating scatter plots for 'Price' against each numerical column, excluding 'close USD' and 'close\_SAR.' This process lets us visualize the linear relationship between 'Price' and the selected column. These scatter plots showcase data points in blue and a red regression line demonstrating the trend between 'Price' and the respective column. Each plot is labeled to represent the examined column, offering an efficient way to assess correlations and trends between 'Price' and the numerical attributes in the dataset.

The regression charts reveal various trends among different features concerning the line of best fit and the distribution of data points. Inflation exhibits a slight positive slope, with most data points concentrated on the higher x-axis and lower y-axis regions (see Fig. 7). Conversely, Brent Crude, WTI Crude Price, WTI Crude Open, WTI Crude High, and WTI Crude Low all showcase a pronounced negative trend, concentrating most data points in the mid to high x-axis range and lower y-axis region (see Fig. 8). Similar patterns are observed for WTI Crude Volume and Fed Funds Rate, portraying pronounced negative relationships, with data points predominantly occupying the lower x-axis. Treasury Rates exhibit a pronounced negative trend, spanning low and high x-axis regions, primarily gathering data points in the low y-axis area (see Fig. 9). The regression plots for all digital currency pair charts demonstrate a strong positive fit to the line of best fit (see Fig. 10). Specifically, digital currency volume, the Dow Jones Industrial Average, and the uncertainty index also display strong positive trends, with the former featuring data points mainly in the lower x-axis and lower y-axis regions and the latter two concentrating data points in the mid to high x-axis and low y-axis areas (see Figs. 11 & 12). These trends shed light on the varying relationships between features and the close\_USD and close\_SAR cryptocurrency pairs.

The observed trends in the regression charts between features and the close\_USD and close\_SAR cryptocurrency pairs unveil interesting insights. For instance, a strong positive correlation between digital currency volume and both close\_USD and close\_SAR pairs implies trading volume's influence on these currencies. The strong positive correlation between indices like the Dow Jones Industrial Average and the uncertainty index hints at an interconnection between broader economic indices and cryptocurrency values, offering insights for strategic decisions in trading or investment strategies involving these currency pairs.

**Data Preprocessing, Preparation, & Cleaning**

The data frames are aligned by date, ensuring all information falls within a standardized temporal range and maintains consistency across multiple datasets. The process involves aligning the various data frames based on their date ranges and ensuring uniformity across all datasets. Firstly, we examined each data frame to identify their first and last recorded dates across economic indicators such as inflation rates, oil prices (Brent and WTI), federal funds rates, treasury rates, cryptocurrency data, Dow Jones Industrial Average, and economic uncertainty indices. The minimum and maximum dates are determined by iterating through each data frame, outlining the temporal span of each dataset. Subsequently, to create a unified date range for analysis, we filtered the merged DataFrame ('merged\_df') by a specified date range from '2018-05-08' to '2021-01-29'. This step involves converting dates to DateTime objects, enabling the slicing and extracting of rows within the predefined temporal boundaries. The resulting 'merged\_df' contains 998 entries and 33 columns, where each row corresponds to data falling within the specified date range.

We implemented a data imputation process to address common data gaps in financial markets arising from weekend closures, holidays, or instances where certain data is unavailable. The method involved leveraging domain expertise regarding market behaviors and applying the 'ffill' (forward fill) technique. This approach fills empty cells with information from the preceding day, effectively carrying forward the most recent available data. By adopting this strategy, our analysis captures a more accurate representation of market movements and trends. This process facilitates a more comprehensive understanding of the financial landscape, even when data remains unreported due to non-trading days or unavailability.

We introduced a dictionary that efficiently translated the existing column names to more informative labels to ensure clear and descriptive column names within our analysis. Changing feature names enabled a more thorough understanding of the dataset. Subsequently, we applied this dictionary to rename the columns in the 'merged\_df' DataFrame, enhancing readability and interpretability. Additionally, as part of refining the dataset, we removed redundant columns such as 'Day\_of\_Week' to simplify the dataset for further analysis. We replaced 'Day\_of\_Week' with 'Day\_of\_Week\_encoded' using a label encoder. This transformation allowed us to represent categorical data numerically while retaining its categorical nature, making it easier to use in subsequent analyses.

In preparation for Logistic regression modeling, we created the columns close\_USD\_change and close\_SAR\_change as new features in the dataset. These features represent whether there was an increase or decrease in the closing prices of USD and SAR cryptocurrencies compared to the preceding day. The values in these columns are binary, where '1' denotes a price increase compared to the previous day, while '0' indicates either no change or a decrease. These engineered features are valuable indicators for modeling cryptocurrency price movements over time.

Smoothing, differencing, and transformation were all deemed necessary during the data analysis process due to several factors. The ACF/PACF charts revealed trends and cyclicality, indicating non-stationary behavior within the data. Smoothing techniques were employed to stabilize the mean and variance, facilitating the identification of patterns and trends. Significant autocorrelation lags beyond statistically acceptable bounds, particularly for USD/Bitcoin and SAR/Bitcoin close prices, further necessitated differencing. This step removed these autocorrelations and rendered the data stationary, a crucial prerequisite for accurate statistical analysis and modeling.

Transformation addressed various data issues, including missing data points addressed through the "ffill" technique, newly created features like "close\_USD\_change" and "close\_SAR\_change" to depict cryptocurrency price movements, and data cleaning through redundant column removal and categorical data encoding. These techniques notably enhanced data quality, enabling a more informative and accurate analysis.

Standardizing the data was essential because many features in our dataset varied across different scales. This disparity could affect the model's performance, especially in logistic regression. By standardizing, we ensured all features were on the same scale, preventing any single feature from overpowering others due to its magnitude. This process made features comparable and enabled the model to interpret their importance based on individual contributions rather than scale.

**Logistic Regression**

In both logistic regression models presented, the primary goal was to predict the 'close\_USD\_change' target variable based on lagged features and selected predictors. The preprocessing steps involved creating lagged features for predictor columns, excluding specific columns ('Date,' 'close\_USD\_change,' 'close\_SAR\_change'), and handling missing values resulting from lag operations.

After the preprocessing steps, the data was split into training (80%) and testing (20%) sets, then the predictor variables were scaled using MinMaxScaler to maintain uniformity in their ranges. We trained logistic regression models using the training data with a maximum iteration of 10,000. Following this, we made predictions on the test set and computed subsequent evaluation metrics for both models.

The evaluation metrics for both logistic regression models showcased high accuracy in predicting the 'close\_USD\_change' target variable. The achieved accuracy was 95.98%, indicating that the models correctly classified approximately 96% of the instances in the test data. The classification report further revealed balanced precision, recall, and F1-scores for both classes (0 and 1), suggesting robust performance in identifying positive and negative instances.

After employing logistic regression to predict the binary variable 'close\_USD\_change,' which indicates whether the 'close\_USD' increased or not, we can leverage this predicted binary outcome as a feature or input within another regression model. This technique combines predictions from diverse models to enhance the overall predictive power, potentially improving the accuracy and robustness of the final prediction for 'close\_USD' or 'close\_SAR.'

**Discussion of Results**

Arbitrage Opportunities arise from recognizing significant disparities in expected prices across digital currency pairs, creating avenues to capitalize on these differences. Through line charts, our exploratory data analysis, trend assessment, and correlation exploration unveiled remarkable synchronization between USD/Bitcoin close prices and SAR/Bitcoin close prices. Despite differing scales due to currency variations between the US and Saudi Arabia, these charts demonstrated strikingly similar trends over time, suggesting a close relationship in valuation trends. Additionally, while most attributes displayed seasonal behavior, treasury rates stood apart, indicating potential stability compared to other economic indicators.

Price Prediction, a crucial aspect of our analysis, emphasized the substantial impact of specific factors on forecasting digital currency prices. Our models achieved an accuracy rate as high as 96%, offering valuable insights into when to buy, sell, or hold SAR or USD. The models also signaled potential market shifts in USD and SAR values, guiding the exploration of trading opportunities.

Our analysis underscored economic indicators such as interest rates, oil prices, and inflation as influential in currency forecasts. Despite their interconnectedness posing challenges in isolating individual impacts, correlations unveiled robust connections between high/low prices, moving averages, and digital currency pairs. Indicators like the Fed Funds Rate exhibited inverse relationships, signaling currency movements with fluctuating interest rates. Oil prices displayed varied relationships with digital currency price changes.

Additionally, our regression analysis aimed at USD/Bitcoin and SAR/Bitcoin close prices, illustrating diverse relationships between features and cryptocurrency pairs. Scatter plots and regression lines showcased trends and correlations. For instance, inflation exhibited a slight positive slope, while various indicators like Brent Crude, WTI Crude Price, and WTI Crude Open showed pronounced negative trends. These trends revealed insights into the relationships between features and USD/Bitcoin and SAR/Bitcoin close prices, providing valuable inputs for strategic trading decisions.

Our analysis utilized multiple regression techniques such as RandomForest, XGBoost, and GradientBoosting to predict close SAR and USD. These models demonstrated high accuracy rates, with ensemble models outperforming individual regressors. The ensemble models achieved mean absolute errors (MAE) of 565.7929 for close SAR and 149.2399 for close USD, indicating their effectiveness in predicting cryptocurrency prices.

The visual exploratory analysis was vital in uncovering synchronization between currency pairs, and regression models provided insights into feature correlations, both contributing to making informed decisions in cryptocurrency trading strategies. However, while promising, our analysis underscores the need for further research and analysis to reach comprehensive conclusions.

**Next Steps**

To ensure the utmost reliability of our predictive models, we plan to establish a robust data preprocessing pipeline that encompasses a series of meticulously executed steps. This pipeline will serve as the cornerstone of our data preparation process, guaranteeing the integrity and quality of the data used for analysis. The first stage of our preprocessing pipeline will focus on handling missing values, leveraging techniques like forward filling ('ffill') to maintain data continuity and completeness. We will also assess the necessity of feature scaling to ensure that no feature dominates others in magnitude. Achieving time series stationarity is another important factor, and our pipeline will incorporate techniques like differencing to remove trends and seasonality. Outlier identification and handling will be a central focus, using robust statistical methods to detect and assess outliers. Additionally, regularization techniques such as L1 (Lasso) and L2 (Ridge) will be employed, when necessary, to prevent overfitting and enhance prediction reliability. This comprehensive preprocessing pipeline ensures that our data is meticulously prepared, meeting the highest standards of quality and reliability, laying a solid foundation for the subsequent stages of our research and analysis.

We also plan to refine our feature development strategy, with a primary goal of enhancing our predictive capabilities. To achieve this, we will commence by generating lag variables for both our target variable and exogenous features. This approach enables us to capture temporal dependencies, offering a more comprehensive insight into the evolving nature of these variables over time and, consequently, improving our model's performance. Additionally, we will delve into various feature transformations to unveil concealed patterns and relationships within our dataset. Furthermore, we will conduct a thorough investigation into external economic factors that were initially presumed to have a significant impact but did not exhibit strong correlations. This exploration will entail an examination of potential lagged effects and intricate relationships that may have previously eluded our attention. Moreover, we will explore the possibility of incorporating features related to cryptocurrency market sentiment and regulatory developments, along with macroeconomic indicators that exert influence over Bitcoin and currency prices. The integration of these supplementary features will provide us with a more comprehensive perspective on the factors influencing our predictions.

In our quest for improved prediction accuracy and better insight into the relationships between out predictor variables, we will explore the possibility of incorporating supplementary external datasets related to USD/SAR Bitcoin prices. These additional datasets will serve as potential sources of valuable predictor variables, enriching our analysis and offering fresh insights. To determine their significance, we will engage in comprehensive research and data analysis, leaving no stone unturned in our pursuit of enhanced forecasting capabilities. By considering diverse external data sources, we aim to unlock hidden patterns and relationships that may have previously eluded us, ultimately contributing to a more robust and accurate prediction model.

To further bolster the effectiveness of our predictive models, we will diversify our modeling approaches. Firstly, we will consider the implementation of time series-specific models like ARIMA, SARIMA, or Prophet which are models that are specifically designed to handle time series data and may offer superior forecasting capabilities. Additionally, we will continue to explore the many different ways to implement ensemble modeling and robustly test various combinations of algorithms with systematic benchmarking of prediction accuracies across differing model types. Techniques such as stacking or blending will also be employed to leverage the unique strengths of different models, resulting in more reliable and robust predictions. By diversifying our modeling strategies, we aim to mitigate risks associated with model limitations and enhance our ability to capture complex patterns and trends in the cryptocurrency market.

In the ever-evolving cryptocurrency landscape, the ability to adapt to changing market dynamics is essential. To achieve this, we will develop a robust real-time data integration pipeline. This pipeline will enable us to continuously ingest and process real-time data, ensuring that our models remain up-to-date and capable of providing timely insights. By implementing continuous model updates, we will capture evolving trends and shifts in the market, empowering us to make informed decisions in real time. This real time data integration approach will not only enhance our project's adaptability but also position us to take advantage of emerging opportunities and respond swiftly to market changes, thereby maximizing the project's impact and utility.

**Conclusion**

In conclusion, our comprehensive Bitcoin price time series analysis has taken us on a journey through various facets of the cryptocurrency market, shedding light on its complexities and opportunities. Starting with an exploration of the dataset, we identified key trends and relationships, particularly the synchronization between USD/Bitcoin and SAR/Bitcoin close prices, which hinted at the interconnectedness of these markets. Moving forward, we delved into predictive modeling, achieving impressive accuracy rates that underscore the potential of data-driven decision-making in cryptocurrency trading. However, this analysis is just a steppingstone towards a deeper understanding of this dynamic landscape. We have outlined clear next steps to fortify our research. Our commitment to data preprocessing involves establishing a robust pipeline to handle missing values, scale features, achieve time series stationarity, and address outliers. Feature development will be a focal point, encompassing the creation of lag variables, feature transformations, and the exploration of external economic factors. Diversifying our modeling approaches with time series-specific models and ensemble techniques aims to further enhance our predictive capabilities. These steps are essential as we recognize the inherent challenges in modeling cryptocurrency volatility and market behavior. Lastly, our dedication to real-time data integration highlights our readiness to adapt to the ever-evolving cryptocurrency market. We are poised to capture emerging trends, respond swiftly to market changes, and maximize the impact of our research. As we continue this exciting journey, we remain committed to unlocking hidden patterns and contributing to a more robust and accurate prediction model in the realm of cryptocurrency trading. This paper represents not only our current findings but also the blueprint for our ongoing pursuit of knowledge and insights in this dynamic field.

In the realm of cryptocurrency trading, where uncertainty often prevails, our journey through this Bitcoin price time series analysis has illuminated a path towards data-driven informed decision making. We embarked on this exploration with a fundamental question in mind: How can we better understand and predict Bitcoin's value in the context of both Saudi Arabian Riyal (SAR) and United States Dollar (USD) markets? This question resonates with the broader financial landscape, where cryptocurrencies are becoming increasingly significant. Our findings, rooted in rigorous data analysis and predictive modeling, offer a glimpse into the intricate dynamics of this evolving market. We have uncovered remarkable correlations, harnessed the power of predictive models with high accuracy, and paved the way for comprehensive data preprocessing and feature development. This journey has equipped us to address the challenges of modeling cryptocurrency volatility and market behavior, ultimately aiming to empower stakeholders with actionable insights. As we continue our pursuit, we remain dedicated to bridging the gap between data science and cryptocurrency trading, with the goal of providing valuable guidance to those navigating this exciting and ever-changing landscape.

**Figures**

**Figure 1**

*Line Chart of close USD by Date*

A line graph with numbers and a line

Description automatically generated

**Figure 1**

*Line Chart of close SAR by Date*

A graph with blue line

Description automatically generated

**Figure 3**

*Line Chart of 52 Week Treas. Rate by Date*

A line chart of a graph

Description automatically generated

**Figure 4**

*ACF for 5-Year Inflation ExpectationA graph of a graph showing the growth of the rate of inflation

Description automatically generated with medium confidence*

**Figure 5**

*ACF for WTI Crude Price Change %*

*A graph with blue dots

Description automatically generated*

**Figure 6**

*ACF After Differencing for 5-Year Inflation ExpectationA graph with blue dots

Description automatically generated*

**Figure 7**

*Catter Plot of Price against 5-Year Inflation Expectation with Regression Line*

*A graph with blue dots and a red line

Description automatically generated*

**Figure 8**

*Scatter Plot of Price Against Brent Crude Price with Regression LineA graph with blue dots and a red line

Description automatically generated*

**Figure 9**

*Scatter Plot of Price Against 52 Week treas. Yield with Regression Line*

*A graph with blue dots and a red line

Description automatically generated*

**Figure 10**

*Scatter Plot of Price Against high USD with Regression Line*

*A graph with a red line

Description automatically generated*

**Figure 11**

*Scatter Plot of Price Against DJIA with Regression Line*

*A graph with blue dots and a red line

Description automatically generated*

**Figure 12**

*Scatter Plot of Price Against Digital Currency Volume with Regression Line*

*A graph with blue dots and red line

Description automatically generated*

References

Shmueli, G. & Lichtendahl Jr., K.C. (2018). Practical time series forecasting with R: A hands-on guide (2nd ed.). Axelrod Schnall Publishers.

Frankenfield, J. (2023, November 3). Cryptocurrency Explained With Pros and Cons for Investment.https://www.investopedia.com/terms/c/cryptocurrency.asp#:~:text=Cryptocurrencies%20are%20digital%20assets%20that,the%20risks%20involved%20before%20investing.

Juairiah, F., Mahatabe, M., Jamal, H. B., Shiddika, A., Shawon, T. R., & Mandal, N. C. (2022). Stock price prediction: A time series analysis. 2022 25th International Conference on

Computer and Information Technology (ICCIT). <https://doi.org/10.1109/iccit57492.2022.10056009>

Massol, O., & Banal-Estañol, A. (2018). Market power and spatial arbitrage between interconnected gas hubs. The Energy Journal, 39(01).

<https://doi.org/10.5547/01956574.39.si2.omas>

Priya, A., & Garg, S. (2018). Autoregressive integrated moving average model based prediction of bitcoin close price. 2018 International Conference on Smart Systems and Inventive

Technology (ICSSIT). <https://doi.org/10.1109/icssit.2018.8748423> Luo, Z. (2022). Stock price prediction based on Time Series model. 2022 6th Annual International Conference on Data Science and Business Analytics (ICDSBA).

<https://doi.org/10.1109/icdsba57203.2022.00100>

Appendix

# Import Packages  
import numpy as np  
import pandas as pd  
import os  
import matplotlib.pyplot as plt  
import seaborn as sns  
from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf  
import warnings  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import mean\_squared\_error  
from sklearn.ensemble import VotingRegressor, RandomForestRegressor, GradientBoostingRegressor  
from xgboost import XGBRegressor  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score, explained\_variance\_score  
from sklearn.preprocessing import LabelEncoder  
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler  
from sklearn.linear\_model import LogisticRegression  
import xgboost as xgb  
from sklearn import metrics  
from sklearn.decomposition import PCA  
from sklearn.impute import KNNImputer  
from statsmodels.stats.outliers\_influence import variance\_inflation\_factor  
import warnings  
warnings.filterwarnings("ignore")

### Load in all the data, apply data cleaning, and merge into singular dataframe

# folder\_path = "C:/Users/gseba/OneDrive/School\_Stuff/ADS\_506\_Applied\_Time\_Series\_Analysis/Group\_Project/Working\_Files\_Git/Bitcoin\_time\_series\_analysis\_and\_forecast/data"   
folder\_path = "C:\MIDS\ADS-506-Applied\_Time\_Series\_Analysis\Bitcoin\_time\_series\_analysis\_and\_forecast\data"   
  
# Get a list of all CSV files in the folder  
csv\_files = [f for f in os.listdir(folder\_path) if f.endswith('.csv')]  
  
# Create a dictionary to store DataFrames  
dataframes = {}  
  
# Read each CSV file into a DataFrame and store it in the dictionary  
for csv\_file in csv\_files:  
 file\_name = os.path.basename(csv\_file) # Extract the file name from the path  
 short\_name = os.path.splitext(file\_name)[0] # Remove the file extension  
 df = pd.read\_csv(os.path.join(folder\_path, csv\_file))  
 dataframes[short\_name] = df  
  
# Rename and create new DataFrames with shorter names  
inflation\_df = dataframes['5-Year Breakeven Inflation Rate - 2018-2021']  
brent\_oil\_df = dataframes['Crude Oil Prices Brent - Europe - 2018-2021']  
federal\_funds\_df = dataframes['Daily Federal Funds Rate from 2018-2021']  
treasury\_rates\_df = dataframes['daily-treasury-rates - 2018-2021']  
crypto\_df = dataframes['Digital\_Currency']  
djia\_df = dataframes['Dow Jones Industrial Average-DJIA - 2018-2021']  
uncertainty\_df = dataframes['Economic Policy Uncertainty Index for United States - 2018-2021']  
wti\_oil\_df = dataframes['Crude Oil WTI Futures Historical Data - 2018-2021']  
  
# Data Cleaning  
  
# Function to convert mixed format to floats  
def convert\_vol(value):  
 if 'K' in value:  
 return float(value.replace('K', '')) \* 1000 # Convert thousands to normal float  
 elif 'M' in value:  
 return float(value.replace('M', '')) \* 1000000 # Convert millions to normal float  
 else:  
 return float(value) # For values without K or M, return as is  
   
# Convert the 'Vol.' column to strings, apply the conversion function, and then convert back to float  
wti\_oil\_df['Vol.'] = wti\_oil\_df['Vol.'].astype(str).apply(convert\_vol).astype(float) # Convert the 'Vol.' column to strings, apply the conversion function, and then convert back to float  
wti\_oil\_df['Change %'] = wti\_oil\_df['Change %'].str.replace('%', '') # Remove '%' from the 'Change %' column  
wti\_oil\_df['Change %'] = wti\_oil\_df['Change %'].astype(float)  
  
# Rename all date columns to "Date" in all DataFrames  
inflation\_df.rename(columns={'DATE': 'Date'}, inplace=True)  
brent\_oil\_df.rename(columns={'DATE': 'Date'}, inplace=True)  
wti\_oil\_df.rename(columns={'Date': 'Date'}, inplace=True)  
federal\_funds\_df.rename(columns={'DATE': 'Date'}, inplace=True)  
treasury\_rates\_df.rename(columns={'Date': 'Date'}, inplace=True)  
crypto\_df.rename(columns={'Unnamed: 0': 'Date'}, inplace=True)  
djia\_df.rename(columns={'DATE': 'Date'}, inplace=True)  
uncertainty\_df.rename(columns={'DATE': 'Date'}, inplace=True)  
  
# List of DataFrames that need to have 'Date' column converted to datetime  
dataframes\_to\_convert = [inflation\_df, brent\_oil\_df, wti\_oil\_df, federal\_funds\_df, treasury\_rates\_df, djia\_df, uncertainty\_df, crypto\_df]  
  
# Convert 'Date' column to datetime format for each DataFrame  
for df in dataframes\_to\_convert:  
 df['Date'] = pd.to\_datetime(df['Date'])  
  
# Sort the DataFrame by Date in ascending order  
crypto\_df = crypto\_df.sort\_values('Date')  
# Calculate the increase/decrease for 'close\_USD' compared to the previous day  
crypto\_df['close\_USD\_change'] = (crypto\_df['close\_USD'].diff() > 0).fillna(0).astype(int)  
# Calculate the increase/decrease for 'close\_SAR' compared to the previous day  
crypto\_df['close\_SAR\_change'] = (crypto\_df['close\_SAR'].diff() > 0).fillna(0).astype(int)  
# Convert columns to integers  
crypto\_df[['close\_USD\_change', 'close\_SAR\_change']] = crypto\_df[['close\_USD\_change', 'close\_SAR\_change']].astype(int)  
  
# Merge all DataFrames on the 'Date' column  
merged\_df = pd.merge(inflation\_df, brent\_oil\_df, on='Date', how='outer')  
merged\_df = pd.merge(merged\_df, wti\_oil\_df, on='Date', how='outer')  
merged\_df = pd.merge(merged\_df, federal\_funds\_df, on='Date', how='outer')  
merged\_df = pd.merge(merged\_df, treasury\_rates\_df, on='Date', how='outer')  
merged\_df = pd.merge(merged\_df, crypto\_df, on='Date', how='outer')  
merged\_df = pd.merge(merged\_df, djia\_df, on='Date', how='outer')  
merged\_df = pd.merge(merged\_df, uncertainty\_df, on='Date', how='outer')  
  
# Replace cells with a dot (.) with a nan  
merged\_df = merged\_df.replace('', pd.NA)  
  
# Define a dictionary to map old column names to new column names  
column\_name\_mapping = {  
 'T5YIE': '5-Year Inflation Expectation',  
 'DCOILBRENTEU': 'Brent Crude Price',  
 'Price': 'WTI Crude Price',  
 'Open': 'WTI Crude Open',  
 'High': 'WTI Crude High',  
 'Low': 'WTI Crude Low',  
 'Vol.': 'WTI Crude Volume',  
 'Change %': 'WTI Crude Price Change %',  
 'DFF': 'Fed Funds Rate',  
 '26 WEEKS BANK DISCOUNT': '26 Week Treas. Rate',  
 '26 WEEKS COUPON EQUIVALENT': '26 Week Treas. Yield',  
 '52 WEEKS BANK DISCOUNT': '52 Week Treas. Rate',  
 '52 WEEKS COUPON EQUIVALENT': '52 Week Treas. Yield',  
 'volume': 'Digital Currency Volume',  
 'DJIA': 'DJIA',  
 'USEPUINDXD': 'Uncertainty Index',  
}  
  
# Rename the columns in merged\_df using the dictionary  
merged\_df.rename(columns=column\_name\_mapping, inplace=True)  
print(merged\_df.isna().sum())  
merged\_df.head()

Date 0  
5-Year Inflation Expectation 635  
Brent Crude Price 621  
WTI Crude Price 596  
WTI Crude Open 596  
WTI Crude High 596  
WTI Crude Low 596  
WTI Crude Volume 628  
WTI Crude Price Change % 596  
Fed Funds Rate 318  
4 WEEKS BANK DISCOUNT 317  
4 WEEKS COUPON EQUIVALENT 317  
8 WEEKS BANK DISCOUNT 515  
8 WEEKS COUPON EQUIVALENT 515  
13 WEEKS BANK DISCOUNT 317  
13 WEEKS COUPON EQUIVALENT 317  
26 Week Treas. Rate 317  
26 Week Treas. Yield 317  
52 Week Treas. Rate 317  
52 Week Treas. Yield 317  
open\_SAR 318  
open\_USD 318  
high\_SAR 318  
high\_USD 318  
low\_SAR 318  
low\_USD 318  
close\_SAR 318  
close\_USD 318  
Digital Currency Volume 318  
close\_USD\_change 318  
close\_SAR\_change 318  
DJIA 629  
Uncertainty Index 318  
dtype: int64

Date 5-Year Inflation Expectation Brent Crude Price \  
0 2018-05-07 2.11 NaN   
1 2018-05-08 2.10 74.16   
2 2018-05-09 2.13 77.60   
3 2018-05-10 2.12 77.59   
4 2018-05-11 2.13 77.37   
  
 WTI Crude Price WTI Crude Open WTI Crude High WTI Crude Low \  
0 70.73 69.85 70.84 69.51   
1 69.06 70.03 70.40 67.63   
2 71.14 70.05 71.36 69.85   
3 71.36 71.23 71.89 70.56   
4 70.70 71.45 71.63 70.45   
  
 WTI Crude Volume WTI Crude Price Change % Fed Funds Rate ... high\_USD \  
0 758920.0 1.45 1.7 ... 9689.67   
1 1250000.0 -2.36 1.7 ... 9475.70   
2 863390.0 3.01 1.7 ... 9390.00   
3 749810.0 0.31 1.7 ... 9395.12   
4 634150.0 -0.92 1.7 ... 9016.80   
  
 low\_SAR low\_USD close\_SAR close\_USD Digital Currency Volume \  
0 34432.422400 9181.00 35122.496000 9365.00 33787.0   
1 33980.649216 9060.54 34457.025024 9187.56 25533.0   
2 33622.336000 8965.00 34916.224000 9310.00 25673.0   
3 33641.088000 8970.00 33761.850880 9002.20 25055.0   
4 31282.086400 8341.00 31503.360000 8400.00 48227.0   
  
 close\_USD\_change close\_SAR\_change DJIA Uncertainty Index   
0 0.0 0.0 24357.32 96.74   
1 0.0 0.0 24360.21 100.02   
2 1.0 1.0 24542.54 62.20   
3 0.0 0.0 24739.53 74.02   
4 0.0 0.0 24831.17 78.27   
  
[5 rows x 33 columns]

#### Align the data frame by date to ensure all data is within the same range

# Store all DataFrames in a dictionary  
all\_dfs = {  
 'Inflation': inflation\_df,  
 'Brent Oil': brent\_oil\_df,  
 'WTI Oil': wti\_oil\_df,  
 'Federal Funds': federal\_funds\_df,  
 'Treasury Rates': treasury\_rates\_df,  
 'Crypto': crypto\_df,  
 'DJIA': djia\_df,  
 'Uncertainty': uncertainty\_df,  
 'Digital Currency': crypto\_df # Include Digital Currency DataFrame  
}  
  
# Find first and last dates in each DataFrame  
for df\_name, df in all\_dfs.items():  
 min\_date = df['Date'].min()  
 max\_date = df['Date'].max()  
 print(f"For {df\_name} DataFrame:")  
 print(f"First date: {min\_date}")  
 print(f"Last date: {max\_date}")  
 print()  
  
# Set the date range  
start\_date = '2018-05-08'  
end\_date = '2021-01-29'  
  
# Convert dates to datetime objects  
start\_date = pd.to\_datetime(start\_date)  
end\_date = pd.to\_datetime(end\_date)  
  
# Filter the DataFrame based on the date range  
merged\_df = merged\_df[(merged\_df['Date'] >= start\_date) & (merged\_df['Date'] <= end\_date)]  
  
# Replace cells with a dot (.) with a nan  
merged\_df = merged\_df.replace('', pd.NA)  
  
# Fill empty cells with data from the previous cell  
merged\_df = merged\_df.fillna(method='ffill')

For Inflation DataFrame:  
First date: 2018-05-07 00:00:00  
Last date: 2021-01-29 00:00:00  
  
For Brent Oil DataFrame:  
First date: 2018-05-08 00:00:00  
Last date: 2021-01-29 00:00:00  
  
For WTI Oil DataFrame:  
First date: 2018-05-07 00:00:00  
Last date: 2021-01-29 00:00:00  
  
For Federal Funds DataFrame:  
First date: 2018-05-07 00:00:00  
Last date: 2021-01-30 00:00:00  
  
For Treasury Rates DataFrame:  
First date: 2018-01-02 00:00:00  
Last date: 2021-12-31 00:00:00  
  
For Crypto DataFrame:  
First date: 2018-05-07 00:00:00  
Last date: 2021-01-30 00:00:00  
  
For DJIA DataFrame:  
First date: 2018-05-07 00:00:00  
Last date: 2021-01-29 00:00:00  
  
For Uncertainty DataFrame:  
First date: 2018-05-07 00:00:00  
Last date: 2021-01-30 00:00:00  
  
For Digital Currency DataFrame:  
First date: 2018-05-07 00:00:00  
Last date: 2021-01-30 00:00:00

## Feature Development

# Create a label encoder  
label\_encoder = LabelEncoder()  
  
# Get the day of the week feature  
merged\_df['Day\_of\_Week'] = merged\_df['Date'].dt.day\_name()  
  
# Fit and transform the 'Day\_of\_Week' column and drop orignal  
merged\_df['Day\_of\_Week\_encoded'] = label\_encoder.fit\_transform(merged\_df['Day\_of\_Week'])  
merged\_df.drop(["Day\_of\_Week"], axis=1, inplace=True)  
  
# Extract month and year from the 'Date' column  
merged\_df['Month'] = merged\_df['Date'].dt.month

#### Lagged Features, Moving Averages, Seasonality, Cyclical Encoding, and Time-Based Aggregations for Enhancing Regression Models on Time Series Data

# Calculate 7-day and 30-day moving averages  
merged\_df['close\_SAR\_7day\_ma'] = merged\_df['close\_SAR'].rolling(window=7).mean()  
merged\_df['close\_SAR\_9day\_ma'] = merged\_df['close\_SAR'].rolling(window=9).mean()  
merged\_df['close\_SAR\_15day\_ma'] = merged\_df['close\_SAR'].rolling(window=15).mean()  
merged\_df['close\_SAR\_21day\_ma'] = merged\_df['close\_SAR'].rolling(window=21).mean()  
merged\_df['close\_SAR\_30day\_ma'] = merged\_df['close\_SAR'].rolling(window=30).mean()  
merged\_df['close\_SAR\_50day\_ma'] = merged\_df['close\_SAR'].rolling(window=50).mean()  
  
merged\_df['close\_USD\_7day\_ma'] = merged\_df['close\_USD'].rolling(window=7).mean()  
merged\_df['close\_USD\_9day\_ma'] = merged\_df['close\_USD'].rolling(window=9).mean()  
merged\_df['close\_USD\_15day\_ma'] = merged\_df['close\_USD'].rolling(window=15).mean()  
merged\_df['close\_USD\_21day\_ma'] = merged\_df['close\_USD'].rolling(window=21).mean()  
merged\_df['close\_USD\_30day\_ma'] = merged\_df['close\_USD'].rolling(window=30).mean()  
merged\_df['close\_USD\_50day\_ma'] = merged\_df['close\_USD'].rolling(window=50).mean()  
  
# Apply Fourier transformations to capture seasonality  
# Assuming 'Date' is datetime type  
merged\_df['day\_of\_year'] = merged\_df['Date'].dt.dayofyear  
merged\_df['sin\_day\_of\_year'] = np.sin(2 \* np.pi \* merged\_df['day\_of\_year'] / 365)  
merged\_df['cos\_day\_of\_year'] = np.cos(2 \* np.pi \* merged\_df['day\_of\_year'] / 365)  
  
# Encode cyclical patterns in the 'Date' column  
merged\_df['day\_of\_week\_sin'] = np.sin(2 \* np.pi \* merged\_df['Date'].dt.dayofweek / 7)  
merged\_df['day\_of\_week\_cos'] = np.cos(2 \* np.pi \* merged\_df['Date'].dt.dayofweek / 7)

#### Use Knn Imputation in order to handle missing data from new features

# Extract the 'Date' column and store it separately  
dates\_column = merged\_df['Date']  
  
# Remove the 'Date' column from the DataFrame  
merged\_df = merged\_df.drop(columns=['Date'])  
  
# Initialize the KNNImputer  
knn\_imputer = KNNImputer(n\_neighbors=5) # You can adjust the number of neighbors as needed  
  
# Apply KNN imputation to your DataFrame  
imputed\_array = knn\_imputer.fit\_transform(merged\_df)  
  
# Convert the imputed array back to a DataFrame with the same columns (excluding 'Date')  
merged\_df = pd.DataFrame(imputed\_array, columns=merged\_df.columns)  
  
# Add the 'Date' column back to the DataFrame  
merged\_df['Date'] = dates\_column

# Create a new DataFrame to store columns before any dropping, to be used later in the code  
date\_df = merged\_df[["Date"]].copy()  
close\_USD\_df = merged\_df[["close\_USD"]].copy()  
close\_SAR\_df = merged\_df[["close\_SAR"]].copy()

## EDA

# Create the "plots" directory within the repo  
if not os.path.exists("plots"):  
 os.makedirs("plots")

#### Assess Correlations

# Calculate the correlation matrix  
correlation\_matrix = merged\_df.corr()  
  
# Create a heatmap  
plt.figure(figsize=(30, 24))  
heatmap = sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)  
plt.xticks(rotation=45, ha='right')   
plt.savefig("plots/correlation\_matrix.png")  
plt.title('Correlation Matrix')  
plt.show()

A colorful squares with black lines

Description automatically generated with medium confidence

#### Visualize the correlations

# Extract correlations for 'close SAR' and 'close USD' against other variables  
correlation\_with\_close\_sar = correlation\_matrix['close\_SAR'].drop(['close\_SAR', 'close\_USD']).sort\_values(ascending=False)  
correlation\_with\_close\_usd = correlation\_matrix['close\_USD'].drop(['close\_SAR', 'close\_USD']).sort\_values(ascending=False)  
  
# Create a horizontal bar plot for the correlation with 'close SAR'  
plt.figure(figsize=(8, 8))  
plt.barh(correlation\_with\_close\_sar.index, correlation\_with\_close\_sar, color='b', alpha=0.7)  
plt.title("Correlation with 'close SAR'")  
plt.ylabel("Variables")  
plt.xlabel("Correlation")  
plt.tight\_layout()  
plt.savefig("plots/correlation\_with\_close\_sar.png")  
plt.show()  
  
# Create a horizontal bar plot for the correlation with 'close USD'  
plt.figure(figsize=(8, 8))  
plt.barh(correlation\_with\_close\_usd.index, correlation\_with\_close\_usd, color='g', alpha=0.7)  
plt.title("Correlation with 'close USD'")  
plt.ylabel("Variables")  
plt.xlabel("Correlation")  
plt.tight\_layout()  
plt.savefig("plots/correlation\_with\_close\_usd.png")  
plt.show()

A graph with blue and white lines

Description automatically generated

A graph of a graph with numbers

Description automatically generated with medium confidence

## Line Charts

# Create a directory for Line Charts if it doesn't exist  
line\_charts\_dir = 'plots/Line Charts'  
os.makedirs(line\_charts\_dir, exist\_ok=True)  
  
# Select only the numerical columns  
numerical\_columns = [col for col in merged\_df.columns if col != 'Date']  
  
# Choose a specific column to plot  
selected\_column = numerical\_columns[0]  
  
# Plotting line charts for each numerical column against 'Date' and save to the "Line Charts" folder  
for column in numerical\_columns:  
 try:  
 plt.figure(figsize=(10, 6))  
 ax = sns.lineplot(x='Date', y=column, data=merged\_df)  
 plt.xlabel('Date')  
 plt.ylabel(column)  
 plt.title(f'Line chart of {column} against Date')  
 plt.xticks(rotation=45)  
 plt.tight\_layout()  
  
 # Save the plot to the "Line Charts" folder  
 plot\_filename = os.path.join(line\_charts\_dir, f'{column}\_line\_chart.png')  
 plt.savefig(plot\_filename)  
  
 # Close the plot to free up resources  
 plt.close()  
 except Exception as e:  
 print(f"Error plotting {column}: {e}")

## ACF and PACF Plots

warnings.filterwarnings("ignore")  
  
# Create folders if they don't exist  
os.makedirs('plots/ACF plots', exist\_ok=True)  
os.makedirs('plots/PACF plots', exist\_ok=True)  
  
# Select only the numerical columns  
numerical\_columns = merged\_df.select\_dtypes(include='number').columns.tolist()  
  
# Plot ACF and PACF for each numerical column  
for column in numerical\_columns:  
 # ACF Plot  
 plt.figure(figsize=(8, 4))  
 plot\_acf(merged\_df[column].dropna(), lags=50)  
 plt.title(f'ACF for {column}')  
 plt.xlabel('Lags')  
 plt.ylabel('Autocorrelation')  
   
 # Create the ACF folder path  
 acf\_folder = 'plots/ACF plots'  
 os.makedirs(acf\_folder, exist\_ok=True)  
   
 # Save ACF plot to the ACF folder  
 plot\_filename\_acf = os.path.join(acf\_folder, f'{column}\_acf.png')  
 plt.savefig(plot\_filename\_acf)  
 plt.close()   
  
 # PACF Plot  
 plt.figure(figsize=(8, 4))  
 plot\_pacf(merged\_df[column].dropna(), lags=50)  
 plt.title(f'PACF for {column}')  
 plt.xlabel('Lags')  
 plt.ylabel('Partial Autocorrelation')  
   
 # Create the PACF folder path  
 pacf\_folder = 'plots/PACF plots'  
 os.makedirs(pacf\_folder, exist\_ok=True)  
   
 # Save PACF plot to the PACF folder  
 plot\_filename\_pacf = os.path.join(pacf\_folder, f'{column}\_pacf.png')  
 plt.savefig(plot\_filename\_pacf)  
 plt.close()

## ACF and PACF Plots with Differencing

warnings.filterwarnings("ignore")  
  
# Create folders for ACF and PACF plots if they don't exist  
acf\_folder = 'plots/ACF\_after\_differencing'  
pacf\_folder = 'plots/PACF\_after\_differencing'  
  
os.makedirs(acf\_folder, exist\_ok=True)  
os.makedirs(pacf\_folder, exist\_ok=True)  
  
# Select only the numerical columns  
numerical\_columns = merged\_df.select\_dtypes(include='number').columns.tolist()  
  
# Function to save ACF and PACF plots after differencing  
def save\_acf\_pacf\_plots\_after\_differencing(column, differenced\_data, acf=True):  
 plt.figure(figsize=(8, 4))  
 if acf:  
 plot\_acf(differenced\_data, lags=50)  
 plt.title(f'ACF after differencing for {column}')  
 plt.ylabel('Autocorrelation')  
 plt.savefig(os.path.join(acf\_folder, f'{column}\_acf\_after\_differencing.png'))  
 else:  
 plot\_pacf(differenced\_data, lags=50)  
 plt.title(f'PACF after differencing for {column}')  
 plt.ylabel('Partial Autocorrelation')  
 plt.savefig(os.path.join(pacf\_folder, f'{column}\_pacf\_after\_differencing.png'))  
 plt.close()  
  
# Apply differencing to each numerical column and plot ACF and PACF after differencing  
for column in numerical\_columns:  
 differenced\_data = merged\_df[column].diff().dropna()  
 save\_acf\_pacf\_plots\_after\_differencing(column, differenced\_data, acf=True)  
 save\_acf\_pacf\_plots\_after\_differencing(column, differenced\_data, acf=False);

## Regression Plots

# Create folders for SAR and USD regression plots if they don't exist  
sar\_regression\_folder = 'plots/Regression\_Plots/close\_SAR'  
usd\_regression\_folder = 'plots/Regression\_Plots/close\_USD'  
os.makedirs(sar\_regression\_folder, exist\_ok=True)  
os.makedirs(usd\_regression\_folder, exist\_ok=True)  
  
# Function to save regression plots  
def save\_regression\_plots(df, target\_column, regression\_folder):  
 numerical\_columns = df.select\_dtypes(include='number').columns.tolist()  
 for column in numerical\_columns:  
 if column != target\_column:  
 plt.figure(figsize=(8, 5))  
 sns.lmplot(x=column, y=target\_column, data=df, height=6, scatter\_kws={'color': '#3e82fc'}, line\_kws={'color': 'red'})  
 plt.xlabel(column)  
 plt.ylabel(target\_column)  
 plt.title(f'Scatter plot of {target\_column} against {column} with Regression Line')  
 plt.tight\_layout()  
 plt.savefig(os.path.join(regression\_folder, f'{target\_column}\_{column}\_regression.png'))  
 plt.close()  
  
# Generate and save regression plots for close\_SAR  
save\_regression\_plots(merged\_df, 'close\_SAR', sar\_regression\_folder)  
# Generate and save regression plots for close\_USD  
save\_regression\_plots(merged\_df, 'close\_USD', usd\_regression\_folder)

## Modeling

#### Compute Performance Metrics of Baseline Naive Model

# Split the data into training and testing sets  
X = merged\_df.drop(columns=['close\_SAR', 'close\_USD'])  
y\_sar = merged\_df['close\_SAR']  
y\_usd = merged\_df['close\_USD']  
X\_train, X\_test, y\_sar\_train, y\_sar\_test, y\_usd\_train, y\_usd\_test = train\_test\_split(  
 X, y\_sar, y\_usd, test\_size=0.2, random\_state=42)  
  
# Calculate the mean of the target variables in the training set  
mean\_sar = y\_sar\_train.mean()  
mean\_usd = y\_usd\_train.mean()  
  
# Create predictions by using the mean as a constant value  
sar\_baseline\_predictions = [mean\_sar] \* len(y\_sar\_test)  
usd\_baseline\_predictions = [mean\_usd] \* len(y\_usd\_test)  
  
# Calculate various performance metrics  
sar\_mae = mean\_absolute\_error(y\_sar\_test, sar\_baseline\_predictions)  
sar\_rmse = mean\_squared\_error(y\_sar\_test, sar\_baseline\_predictions, squared=False)  
sar\_r2 = r2\_score(y\_sar\_test, sar\_baseline\_predictions)  
  
usd\_mae = mean\_absolute\_error(y\_usd\_test, usd\_baseline\_predictions)  
usd\_rmse = mean\_squared\_error(y\_usd\_test, usd\_baseline\_predictions, squared=False)  
usd\_r2 = r2\_score(y\_usd\_test, usd\_baseline\_predictions)  
  
# Create a bar plot to compare performance metrics for close\_SAR and close\_USD  
metrics = ['MAE', 'RMSE', 'R2']  
sar\_values = [sar\_mae, sar\_rmse, sar\_r2]  
usd\_values = [usd\_mae, usd\_rmse, usd\_r2]  
  
plt.figure(figsize=(8, 4))  
bar\_width = 0.35  
index = np.arange(len(metrics)) # Use numpy.arange to create an array for bar positions  
  
bars1 = plt.bar(index, sar\_values, bar\_width, label='close\_SAR', alpha=0.7, color='b', align='center')  
bars2 = plt.bar(index + bar\_width, usd\_values, bar\_width, label='close\_USD', alpha=0.7, color='g', align='center')  
  
plt.xlabel('Metrics')  
plt.ylabel('Values')  
plt.title('Baseline Model Performance Metrics Comparison for close\_SAR and close\_USD')  
plt.xticks(index + bar\_width / 2, metrics) # Adjust x-axis labels to be at the center of each pair of bars  
plt.legend()  
  
for bar1, bar2 in zip(bars1, bars2):  
 height1 = bar1.get\_height()  
 height2 = bar2.get\_height()  
 plt.annotate(f'{height1:.2f}', xy=(bar1.get\_x() + bar1.get\_width() / 2, height1),  
 xytext=(0, 3), # 3 points vertical offset  
 textcoords="offset points",  
 ha='center', va='bottom')  
 plt.annotate(f'{height2:.2f}', xy=(bar2.get\_x() + bar2.get\_width() / 2, height2),  
 xytext=(0, 3), # 3 points vertical offset  
 textcoords="offset points",  
 ha='center', va='bottom')  
plt.ylim(0, 22000)  
plt.tight\_layout()  
plt.savefig("plots/baseline\_model\_performance\_metrics\_comparison.png")  
plt.show()

A bar graph with numbers and a blue rectangle

Description automatically generated

## PCA

# Define the features and target variables  
features = [  
 '5-Year Inflation Expectation', 'Brent Crude Price', 'WTI Crude Price', 'WTI Crude Open', 'WTI Crude High', 'WTI Crude Low', 'WTI Crude Volume', 'WTI Crude Price Change %', 'Fed Funds Rate',   
 '4 WEEKS BANK DISCOUNT', '4 WEEKS COUPON EQUIVALENT', '13 WEEKS BANK DISCOUNT', '13 WEEKS COUPON EQUIVALENT', '26 Week Treas. Rate','26 Week Treas. Yield', '52 Week Treas. Rate',   
 '52 Week Treas. Yield', 'open\_SAR', 'open\_USD', 'high\_SAR', 'high\_USD', 'low\_SAR', 'low\_USD', 'Digital Currency Volume', 'DJIA', 'Uncertainty Index', 'Day\_of\_Week\_encoded', 'Month',   
 'close\_SAR\_7day\_ma', 'close\_SAR\_9day\_ma', 'close\_SAR\_15day\_ma', 'close\_SAR\_21day\_ma', 'close\_SAR\_30day\_ma', 'close\_SAR\_50day\_ma', 'close\_USD\_7day\_ma', 'close\_USD\_9day\_ma', 'close\_USD\_15day\_ma',   
 'close\_USD\_21day\_ma', 'close\_USD\_30day\_ma', 'close\_USD\_50day\_ma', 'day\_of\_year', 'sin\_day\_of\_year', 'cos\_day\_of\_year', 'day\_of\_week\_sin', 'day\_of\_week\_cos']  
target\_SAR = 'close\_SAR'  
target\_USD = 'close\_USD'  
  
# Split the data into train and test sets  
X\_train, X\_test, y\_train\_SAR, y\_test\_SAR, y\_train\_USD, y\_test\_USD = train\_test\_split(  
 merged\_df[features], merged\_df[target\_SAR], merged\_df[target\_USD], test\_size=0.2, random\_state=42  
)  
  
# Standardize the features  
scaler = StandardScaler()  
X\_train\_scaled = scaler.fit\_transform(X\_train)  
X\_test\_scaled = scaler.transform(X\_test)  
  
# Perform PCA  
pca = PCA()  
X\_train\_pca = pca.fit\_transform(X\_train\_scaled)  
X\_test\_pca = pca.transform(X\_test\_scaled)  
  
# Determine the number of components to use based on explained variance  
explained\_variance\_ratio = pca.explained\_variance\_ratio\_  
cumulative\_variance = explained\_variance\_ratio.cumsum()  
threshold\_variance = 0.95 # You can adjust this threshold as needed  
n\_components = len(cumulative\_variance[cumulative\_variance <= threshold\_variance])  
  
# Print the number of principal components used  
print(f"Number of Principal Components used: {n\_components}")  
  
# Retain only the top 'n\_components' components  
X\_train\_pca = X\_train\_pca[:, :n\_components]  
X\_test\_pca = X\_test\_pca[:, :n\_components]  
  
# Create a plot to visualize explained variance ratio for each component  
plt.figure(figsize=(10, 6))  
plt.plot(range(1, len(explained\_variance\_ratio) + 1), cumulative\_variance, marker='o', linestyle='--')  
plt.xlabel('Number of Components')  
plt.ylabel('Cumulative Explained Variance Ratio')  
plt.title('Cumulative Explained Variance Ratio vs. Number of Components')  
plt.grid(True)  
  
# Mark the number of components used in the final model  
plt.axvline(x=n\_components, color='r', linestyle='--', label=f'Used Components ({n\_components})')  
plt.legend()  
plt.savefig('plots/pca\_explained\_variance\_ratio.png')  
plt.show()

Number of Principal Components used: 10

A graph of a number of components

Description automatically generated

#### Use PCA components (transformed features) for modeling instead of standard features

# Create and train regression models on the PCA components  
model\_names = ['RandomForest', 'XGBoost', 'GradientBoosting']  
  
# Create lists to store metric values for each model  
r2\_SAR\_values = []  
mae\_SAR\_values = []  
rmse\_SAR\_values = []  
r2\_USD\_values = []  
mae\_USD\_values = []  
rmse\_USD\_values = []  
  
for model\_name in model\_names:  
 if model\_name == 'RandomForest':  
 model = RandomForestRegressor(random\_state=42)  
 elif model\_name == 'XGBoost':  
 model = XGBRegressor(random\_state=42)  
 elif model\_name == 'GradientBoosting':  
 model = GradientBoostingRegressor(random\_state=42)  
  
 # Train the model on the PCA components for SAR  
 model.fit(X\_train\_pca, y\_train\_SAR)  
   
 # Make predictions for SAR  
 y\_pred\_SAR = model.predict(X\_test\_pca)  
   
 # Calculate R-squared (R^2), MAE, and RMSE for close\_SAR  
 r2\_SAR = r2\_score(y\_test\_SAR, y\_pred\_SAR)  
 mae\_SAR = mean\_absolute\_error(y\_test\_SAR, y\_pred\_SAR)  
 rmse\_SAR = mean\_squared\_error(y\_test\_SAR, y\_pred\_SAR, squared=False)  
   
 # Store metric values in lists for SAR  
 r2\_SAR\_values.append(r2\_SAR)  
 mae\_SAR\_values.append(mae\_SAR)  
 rmse\_SAR\_values.append(rmse\_SAR)  
   
 # Train the model on the PCA components for USD  
 model.fit(X\_train\_pca, y\_train\_USD)  
   
 # Make predictions for USD  
 y\_pred\_USD = model.predict(X\_test\_pca)  
   
 # Calculate R-squared (R^2), MAE, and RMSE for close\_USD  
 r2\_USD = r2\_score(y\_test\_USD, y\_pred\_USD)  
 mae\_USD = mean\_absolute\_error(y\_test\_USD, y\_pred\_USD)  
 rmse\_USD = mean\_squared\_error(y\_test\_USD, y\_pred\_USD, squared=False)  
   
 # Store metric values in lists for USD  
 r2\_USD\_values.append(r2\_USD)  
 mae\_USD\_values.append(mae\_USD)  
 rmse\_USD\_values.append(rmse\_USD)  
  
# Create subplots for performance metrics (USD and SAR side by side)  
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(18, 6))  
  
# Define color schemes for bars  
sar\_bar\_colors = ['blue', 'darkblue', 'lightblue']  
usd\_bar\_colors = ['green', 'darkgreen', 'lightgreen']  
  
# Plot performance metrics for close\_SAR  
sar\_x = np.arange(len(model\_names))  
width = 0.2  
  
# Create SAR metrics  
sar\_metrics = {  
 'Model': model\_names,  
 'MAE': mae\_SAR\_values,  
 'RMSE': rmse\_SAR\_values,  
 'R2': r2\_SAR\_values  
}  
  
# Plot SAR metrics  
bar1 = axes[0].bar(sar\_x - width, sar\_metrics['MAE'], width, label='MAE', color=sar\_bar\_colors[0], alpha=0.7)  
bar2 = axes[0].bar(sar\_x, sar\_metrics['RMSE'], width, label='RMSE', color=sar\_bar\_colors[1], alpha=0.7)  
  
axes[0].set\_title('Performance Metrics for close\_SAR Models')  
axes[0].set\_xlabel('Model')  
axes[0].set\_xticks(sar\_x)  
axes[0].set\_xticklabels(sar\_metrics['Model'])  
  
# Create a twin y-axis for R2  
axes2 = axes[0].twinx()  
bar3 = axes2.bar(sar\_x + width, sar\_metrics['R2'], width, label='R2', color=sar\_bar\_colors[2], alpha=0.7)  
axes2.set\_ylabel('R2')  
axes2.set\_ylim(0, 1.2) # Set the y-axis range for R2  
  
# Add numbers on top of the bars for MAE and RMSE  
for bars in [bar1, bar2]:  
 for bar in bars:  
 height = bar.get\_height()  
 axes[0].annotate(f'{height:.4f}', xy=(bar.get\_x() + bar.get\_width() / 2, height),  
 xytext=(0, 3), # 3 points vertical offset  
 textcoords="offset points",  
 ha='center', va='bottom')  
  
# Add R2 values on top of the R2 bars  
for bar in bar3:  
 height = bar.get\_height()  
 axes2.annotate(f'{height:.4f}', xy=(bar.get\_x() + bar.get\_width() / 2, height),  
 xytext=(0, 3), # 3 points vertical offset  
 textcoords="offset points",  
 ha='center', va='bottom')  
  
# Create USD metrics  
usd\_metrics = {  
 'Model': model\_names,  
 'MAE': mae\_USD\_values,  
 'RMSE': rmse\_USD\_values,  
 'R2': r2\_USD\_values  
}  
  
# Print performance metrics for each model  
for model\_name, r2\_sar, mae\_sar, rmse\_sar, r2\_usd, mae\_usd, rmse\_usd in zip(  
 model\_names, r2\_SAR\_values, mae\_SAR\_values, rmse\_SAR\_values, r2\_USD\_values, mae\_USD\_values, rmse\_USD\_values):  
 print(f"Model: {model\_name}")  
 print(f"R-squared (R^2) for SAR: {r2\_sar:.4f}")  
 print(f"MAE for SAR: {mae\_sar:.4f}")  
 print(f"RMSE for SAR: {rmse\_sar:.4f}")  
 print(f"R-squared (R^2) for USD: {r2\_usd:.4f}")  
 print(f"MAE for USD: {mae\_usd:.4f}")  
 print(f"RMSE for USD: {rmse\_usd:.4f}")  
 print()  
   
# Plot performance metrics for close\_USD  
usd\_x = np.arange(len(usd\_metrics['Model']))  
  
# Plot USD metrics  
bar4 = axes[1].bar(usd\_x - width, usd\_metrics['MAE'], width, label='MAE', color=usd\_bar\_colors[0], alpha=0.7)  
bar5 = axes[1].bar(usd\_x, usd\_metrics['RMSE'], width, label='RMSE', color=usd\_bar\_colors[1], alpha=0.7)  
  
axes[1].set\_title('Performance Metrics for close\_USD Models using PCA Transformed Features')  
axes[1].set\_xlabel('Model')  
axes[1].set\_xticks(usd\_x)  
axes[1].set\_xticklabels(usd\_metrics['Model'])  
  
# Create a twin y-axis for R2  
axes3 = axes[1].twinx()  
bar6 = axes3.bar(usd\_x + width, usd\_metrics['R2'], width, label='R2', color=usd\_bar\_colors[2], alpha=0.7)  
axes3.set\_ylabel('R2')  
axes3.set\_ylim(0, 1.2) # Set the y-axis range for R2  
  
# Add numbers on top of the bars for MAE and RMSE  
for bars in [bar4, bar5]:  
 for bar in bars:  
 height = bar.get\_height()  
 axes[1].annotate(f'{height:.4f}', xy=(bar.get\_x() + bar.get\_width() / 2, height),  
 xytext=(0, 3), # 3 points vertical offset  
 textcoords="offset points",  
 ha='center', va='bottom')  
  
# Add R2 values on top of the R2 bars  
for bar in bar6:  
 height = bar.get\_height()  
 axes3.annotate(f'{height:.4f}', xy=(bar.get\_x() + bar.get\_width() / 2, height),  
 xytext=(0, 3), # 3 points vertical offset  
 textcoords="offset points",  
 ha='center', va='bottom')  
  
# Legend for the first subplot (close\_SAR)  
lines, labels = axes[0].get\_legend\_handles\_labels()  
lines2, labels2 = axes2.get\_legend\_handles\_labels()  
axes[0].legend(lines + lines2, labels + labels2, loc='upper left')  
  
# Legend for the second subplot (close\_USD)  
lines3, labels3 = axes[1].get\_legend\_handles\_labels()  
lines4, labels4 = axes3.get\_legend\_handles\_labels()  
axes[1].legend(lines3 + lines4, labels3 + labels4, loc='upper left')  
plt.tight\_layout()  
plt.savefig('plots/pca\_modeling\_performance\_metrics.png')  
plt.show()

Model: RandomForest  
R-squared (R^2) for SAR: 0.9809  
MAE for SAR: 1668.7773  
RMSE for SAR: 2814.5093  
R-squared (R^2) for USD: 0.9807  
MAE for USD: 449.0603  
RMSE for USD: 754.5856  
  
Model: XGBoost  
R-squared (R^2) for SAR: 0.9761  
MAE for SAR: 1735.4824  
RMSE for SAR: 3142.5561  
R-squared (R^2) for USD: 0.9763  
MAE for USD: 461.9725  
RMSE for USD: 835.9088  
  
Model: GradientBoosting  
R-squared (R^2) for SAR: 0.9752  
MAE for SAR: 1888.1573  
RMSE for SAR: 3204.6617  
R-squared (R^2) for USD: 0.9753  
MAE for USD: 501.9317  
RMSE for USD: 852.1700

A close-up of a graph

Description automatically generated

# Define a function to train and evaluate a model for a target variable  
def train\_and\_evaluate\_model(X\_train, X\_test, y\_train, y\_test, target\_variable, model\_name):  
 if model\_name == 'RandomForest':  
 model = RandomForestRegressor()  
 elif model\_name == 'XGBoost':  
 model = xgb.XGBRegressor()  
 elif model\_name == 'GradientBoosting':  
 model = GradientBoostingRegressor()  
 else:  
 raise ValueError(f"Invalid model name: {model\_name}")  
  
 # Train the model  
 model.fit(X\_train, y\_train)  
  
 # Make predictions using the model  
 predictions = model.predict(X\_test)  
  
 # Calculate the model's performance metrics  
 mae = mean\_absolute\_error(y\_test, predictions)  
 rmse = mean\_squared\_error(y\_test, predictions, squared=False)  
 r2 = r2\_score(y\_test, predictions)  
  
 print(f"\n{model\_name} Model Metrics for {target\_variable}:")  
 print(f"Mean Absolute Error (MAE): {mae:.4f}")  
 print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")  
 print(f"R-squared (R2): {r2:.4f}")  
  
 # Get feature importances if the model supports it  
 if hasattr(model, 'feature\_importances\_'):  
 feature\_importances = model.feature\_importances\_  
 return model, feature\_importances  
  
 return model, None  
  
# Drop Date column before scaling  
merged\_df.drop(["Date"], axis=1, inplace=True)  
  
# Split the data into training and testing sets  
X = merged\_df.drop(columns=['close\_SAR', 'close\_USD'])  
y\_sar = merged\_df['close\_SAR']  
y\_usd = merged\_df['close\_USD']  
  
# Scale the features and create a DataFrame  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
X\_scaled\_df = pd.DataFrame(X\_scaled, columns=X.columns)  
  
# Split the scaled data into training and testing sets  
X\_train, X\_test, y\_sar\_train, y\_sar\_test, y\_usd\_train, y\_usd\_test = train\_test\_split(  
 X\_scaled\_df, y\_sar, y\_usd, test\_size=0.2, random\_state=42)  
  
# Define dictionaries to store performance metrics  
sar\_metrics = {'Model': [], 'MAE': [], 'RMSE': [], 'R2': []}  
usd\_metrics = {'Model': [], 'MAE': [], 'RMSE': [], 'R2': []}  
  
# Create dictionaries to store models and feature importances  
sar\_models = {}  
sar\_feature\_importances = {}  
usd\_models = {}  
usd\_feature\_importances = {}  
  
# Train and evaluate models for 'close\_SAR' and 'close\_USD' using different regressors  
model\_names = ['RandomForest', 'XGBoost', 'GradientBoosting']  
  
for target\_variable in ['close\_SAR', 'close\_USD']:  
 for model\_name in model\_names:  
 model, feature\_importances = train\_and\_evaluate\_model(  
 X\_train, X\_test, y\_sar\_train if target\_variable == 'close\_SAR' else y\_usd\_train,  
 y\_sar\_test if target\_variable == 'close\_SAR' else y\_usd\_test,  
 target\_variable, model\_name)  
   
 # Store the model and feature importances in the respective dictionaries  
 models\_dict = sar\_models if target\_variable == 'close\_SAR' else usd\_models  
 feature\_importances\_dict = sar\_feature\_importances if target\_variable == 'close\_SAR' else usd\_feature\_importances  
 models\_dict[model\_name] = model  
 feature\_importances\_dict[model\_name] = feature\_importances  
  
 # Calculate performance metrics  
 y\_pred = model.predict(X\_test)  
 mae = mean\_absolute\_error(y\_sar\_test if target\_variable == 'close\_SAR' else y\_usd\_test, y\_pred)  
 rmse = mean\_squared\_error(y\_sar\_test if target\_variable == 'close\_SAR' else y\_usd\_test, y\_pred, squared=False)  
 r2 = r2\_score(y\_sar\_test if target\_variable == 'close\_SAR' else y\_usd\_test, y\_pred)  
  
 # Store metrics in the respective metrics dictionary  
 metrics\_dict = sar\_metrics if target\_variable == 'close\_SAR' else usd\_metrics  
 metrics\_dict['Model'].append(model\_name)  
 metrics\_dict['MAE'].append(mae)  
 metrics\_dict['RMSE'].append(rmse)  
 metrics\_dict['R2'].append(r2)  
  
 # Create an ensemble model using VotingRegressor  
 ensemble\_model = VotingRegressor(estimators=[  
 (model\_name, models\_dict[model\_name]) for model\_name in model\_names  
 ])  
  
 # Train and evaluate the ensemble model  
 ensemble\_model.fit(X\_train, y\_sar\_train if target\_variable == 'close\_SAR' else y\_usd\_train)  
 ensemble\_predictions = ensemble\_model.predict(X\_test)  
 ensemble\_mae = mean\_absolute\_error(y\_sar\_test if target\_variable == 'close\_SAR' else y\_usd\_test, ensemble\_predictions)  
 ensemble\_rmse = mean\_squared\_error(y\_sar\_test if target\_variable == 'close\_SAR' else y\_usd\_test, ensemble\_predictions, squared=False)  
 ensemble\_r2 = r2\_score(y\_sar\_test if target\_variable == 'close\_SAR' else y\_usd\_test, ensemble\_predictions)  
  
 print(f"\nEnsemble Model Metrics for {target\_variable}:")  
 print(f"Mean Absolute Error (MAE): {ensemble\_mae:.4f}")  
 print(f"Root Mean Squared Error (RMSE): {ensemble\_rmse:.4f}")  
 print(f"R-squared (R2): {ensemble\_r2:.4f}")  
  
# Create subplots for performance metrics  
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(18, 6))  
  
# Define color schemes for bars  
sar\_bar\_colors = ['blue', 'darkblue', 'lightblue']  
usd\_bar\_colors = ['green', 'darkgreen', 'lightgreen']  
  
# Plot performance metrics for close\_SAR  
sar\_x = np.arange(len(sar\_metrics['Model']))  
width = 0.2  
  
bar1 = axes[0].bar(sar\_x - width, sar\_metrics['MAE'], width, label='MAE', color=sar\_bar\_colors[0], alpha=0.7)  
bar2 = axes[0].bar(sar\_x, sar\_metrics['RMSE'], width, label='RMSE', color=sar\_bar\_colors[1], alpha=0.7)  
  
axes[0].set\_title('Performance Metrics for close\_SAR Models')  
axes[0].set\_xlabel('Model')  
axes[0].set\_xticks(sar\_x)  
axes[0].set\_xticklabels(sar\_metrics['Model'])  
  
# Create a twin y-axis for R2  
axes2 = axes[0].twinx()  
bar3 = axes2.bar(sar\_x + width, sar\_metrics['R2'], width, label='R2', color=sar\_bar\_colors[2], alpha=0.7)  
axes2.set\_ylabel('R2')  
axes2.set\_ylim(0, 1.2) # Set the y-axis range for R2  
  
# Add numbers on top of the bars for MAE and RMSE  
for bars in [bar1, bar2]:  
 for bar in bars:  
 height = bar.get\_height()  
 axes[0].annotate(f'{height:.4f}', xy=(bar.get\_x() + bar.get\_width() / 2, height),  
 xytext=(0, 3), # 3 points vertical offset  
 textcoords="offset points",  
 ha='center', va='bottom')  
  
# Add R2 values on top of the R2 bars  
for bar in bar3:  
 height = bar.get\_height()  
 axes2.annotate(f'{height:.4f}', xy=(bar.get\_x() + bar.get\_width() / 2, height),  
 xytext=(0, 3), # 3 points vertical offset  
 textcoords="offset points",  
 ha='center', va='bottom')  
  
# Legend for all bars  
lines, labels = axes[0].get\_legend\_handles\_labels()  
lines2, labels2 = axes2.get\_legend\_handles\_labels()  
axes[0].legend(lines + lines2, labels + labels2, loc='upper left')  
  
# Plot performance metrics for close\_USD  
usd\_x = np.arange(len(usd\_metrics['Model']))  
  
bar4 = axes[1].bar(usd\_x - width, usd\_metrics['MAE'], width, label='MAE', color=usd\_bar\_colors[0], alpha=0.7)  
bar5 = axes[1].bar(usd\_x, usd\_metrics['RMSE'], width, label='RMSE', color=usd\_bar\_colors[1], alpha=0.7)  
  
axes[1].set\_title('Performance Metrics for close\_USD Models')  
axes[1].set\_xlabel('Model')  
axes[1].set\_xticks(usd\_x)  
axes[1].set\_xticklabels(usd\_metrics['Model'])  
  
# Create a twin y-axis for R2  
axes3 = axes[1].twinx()  
bar6 = axes3.bar(usd\_x + width, usd\_metrics['R2'], width, label='R2', color=usd\_bar\_colors[2], alpha=0.7)  
axes3.set\_ylabel('R2')  
axes3.set\_ylim(0, 1.2) # Set the y-axis range for R2  
  
# Add numbers on top of the bars for MAE and RMSE  
for bars in [bar4, bar5]:  
 for bar in bars:  
 height = bar.get\_height()  
 axes[1].annotate(f'{height:.4f}', xy=(bar.get\_x() + bar.get\_width() / 2, height),  
 xytext=(0, 3), # 3 points vertical offset  
 textcoords="offset points",  
 ha='center', va='bottom')  
  
# Add R2 values on top of the R2 bars  
for bar in bar6:  
 height = bar.get\_height()  
 axes3.annotate(f'{height:.4f}', xy=(bar.get\_x() + bar.get\_width() / 2, height),  
 xytext=(0, 3), # 3 points vertical offset  
 textcoords="offset points",  
 ha='center', va='bottom')  
  
# Legend for the first subplot (close\_SAR)  
lines, labels = axes[0].get\_legend\_handles\_labels()  
lines2, labels2 = axes2.get\_legend\_handles\_labels()  
axes[0].legend(lines + lines2, labels + labels2, loc='upper left')  
  
# Legend for the second subplot (close\_USD)  
lines3, labels3 = axes[1].get\_legend\_handles\_labels()  
lines4, labels4 = axes3.get\_legend\_handles\_labels()  
axes[1].legend(lines3 + lines4, labels3 + labels4, loc='upper left')  
  
plt.tight\_layout()  
plt.savefig("plots/standard\_features\_performance\_metrics\_comparison.png")  
plt.show()  
  
# Create subplots for feature importances  
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(18, 12))  
  
# Plot individual feature importances for close\_SAR models (in green)  
for i, model\_name in enumerate(model\_names):  
 axes[0, i].barh(X\_train.columns, sar\_feature\_importances[model\_name], color='green')  
 axes[0, i].set\_title(f'Feature Importances for {model\_name} (close\_SAR)')  
 axes[0, i].set\_xlabel('Feature Importance')  
  
# Plot individual feature importances for close\_USD models (in blue)  
for i, model\_name in enumerate(model\_names):  
 axes[1, i].barh(X\_train.columns, usd\_feature\_importances[model\_name], color='blue')  
 axes[1, i].set\_title(f'Feature Importances for {model\_name} (close\_USD)')  
 axes[1, i].set\_xlabel('Feature Importance')  
  
plt.tight\_layout()  
plt.savefig(f"plots/regression\_models\_feature\_importances\_{model\_name}.png")  
plt.show()

RandomForest Model Metrics for close\_SAR:  
Mean Absolute Error (MAE): 543.9747  
Root Mean Squared Error (RMSE): 972.1629  
R-squared (R2): 0.9977  
  
XGBoost Model Metrics for close\_SAR:  
Mean Absolute Error (MAE): 581.5240  
Root Mean Squared Error (RMSE): 1076.7990  
R-squared (R2): 0.9972  
  
GradientBoosting Model Metrics for close\_SAR:  
Mean Absolute Error (MAE): 551.3066  
Root Mean Squared Error (RMSE): 1073.2355  
R-squared (R2): 0.9972  
  
Ensemble Model Metrics for close\_SAR:  
Mean Absolute Error (MAE): 493.0622  
Root Mean Squared Error (RMSE): 954.8192  
R-squared (R2): 0.9978  
  
RandomForest Model Metrics for close\_USD:  
Mean Absolute Error (MAE): 140.6355  
Root Mean Squared Error (RMSE): 245.7906  
R-squared (R2): 0.9979  
  
XGBoost Model Metrics for close\_USD:  
Mean Absolute Error (MAE): 155.4803  
Root Mean Squared Error (RMSE): 287.9565  
R-squared (R2): 0.9972  
  
GradientBoosting Model Metrics for close\_USD:  
Mean Absolute Error (MAE): 147.9848  
Root Mean Squared Error (RMSE): 293.3585  
R-squared (R2): 0.9971  
  
Ensemble Model Metrics for close\_USD:  
Mean Absolute Error (MAE): 133.4764  
Root Mean Squared Error (RMSE): 257.4048  
R-squared (R2): 0.9977

A close-up of a graph

Description automatically generated

A group of rectangular objects with different colored lines

Description automatically generated with medium confidence

## Logistic Regression Model

# Suppress warnings  
warnings.filterwarnings("ignore")  
  
# Create a new DataFrame for logistic regression without modifying merged\_df  
logistic\_df = merged\_df.copy()  
  
# Columns to exclude from lagged features and target prediction  
exclude\_columns = ['Date', 'close\_USD\_change', 'close\_SAR\_change']  
  
# Create lagged features for each predictor column  
for col in logistic\_df.columns:  
 if col not in exclude\_columns:  
 for i in range(1, 4): # Creating lag features for 3 time periods  
 new\_col\_name = f"{col}\_lag{i}"  
 logistic\_df[new\_col\_name] = logistic\_df[col].shift(i)  
  
# Drop rows with NaN resulting from the lag operation  
logistic\_df.dropna(inplace=True)  
  
# Define predictor variables and target variable  
target = 'close\_USD\_change'   
  
# Define all columns except the ones in exclude\_columns as predictors  
predictors = [col for col in logistic\_df.columns if col not in exclude\_columns]  
  
# Set up X (predictors) and y (target) variables  
X = logistic\_df[predictors]  
y = logistic\_df[target]  
  
# Scale the predictor variables using MinMaxScaler  
scaler = MinMaxScaler()  
X\_scaled = scaler.fit\_transform(X)  
  
# Split the data into training and testing sets  
split\_index = int(len(logistic\_df) \* 0.8) # 80% train, 20% test  
X\_train, X\_test = X[:split\_index], X[split\_index:]  
y\_train, y\_test = y[:split\_index], y[split\_index:]  
  
# Define and fit the logistic regression model  
logreg = LogisticRegression(max\_iter=10000)   
logreg.fit(X\_train, y\_train)  
  
# Predictions  
y\_pred = logreg.predict(X\_test)  
  
# Evaluation metrics  
print("Evaluation metrics for close\_USD\_change:")  
print("Accuracy:", metrics.accuracy\_score(y\_test, y\_pred))  
print("Confusion Matrix:\n", metrics.confusion\_matrix(y\_test, y\_pred))  
print("Classification Report:\n", metrics.classification\_report(y\_test, y\_pred))  
  
# Reset warnings to default  
warnings.resetwarnings()

Evaluation metrics for close\_USD\_change:  
Accuracy: 0.9597989949748744  
Confusion Matrix:  
 [[ 79 6]  
 [ 2 112]]  
Classification Report:  
 precision recall f1-score support  
  
 0.0 0.98 0.93 0.95 85  
 1.0 0.95 0.98 0.97 114  
  
 accuracy 0.96 199  
 macro avg 0.96 0.96 0.96 199  
weighted avg 0.96 0.96 0.96 199

# Create a directory for plots if it doesn't exist  
plots\_dir = 'plots'  
os.makedirs(plots\_dir, exist\_ok=True)  
  
# Function to create and save a Seaborn confusion matrix plot  
def plot\_confusion\_matrix(y\_true, y\_pred, title, cmap):  
 cm = metrics.confusion\_matrix(y\_true, y\_pred)  
 plt.figure(figsize=(6, 4))  
 sns.heatmap(cm, annot=True, fmt='d', cmap=cmap, cbar=False)  
 plt.title(title)  
 plt.xlabel('Predicted')  
 plt.ylabel('Actual')  
 plt.tight\_layout()  
 plot\_filename = os.path.join(plots\_dir, f'{title}\_confusion\_matrix.png')  
 plt.savefig(plot\_filename)  
 plt.show() # Display the confusion matrix in the Jupyter Notebook  
  
# Suppress warnings  
warnings.filterwarnings("ignore")  
  
# Create a new DataFrame for logistic regression without modifying merged\_df  
logistic\_df = merged\_df.copy()  
  
# Columns to exclude from lagged features and target prediction  
exclude\_columns = ['Date', 'close\_USD\_change', 'close\_SAR\_change']  
  
for target, cmap in [('close\_USD\_change', 'Greens'), ('close\_SAR\_change', 'Blues')]:  
 # Create lagged features for each predictor column  
 for col in logistic\_df.columns:  
 if col not in exclude\_columns:  
 for i in range(1, 4): # Creating lag features for 3 time periods  
 new\_col\_name = f"{col}\_lag{i}"  
 logistic\_df[new\_col\_name] = logistic\_df[col].shift(i)  
  
 # Drop rows with NaN resulting from the lag operation  
 logistic\_df.dropna(inplace=True)  
  
 # Define predictor variables and target variable  
 predictors = [col for col in logistic\_df.columns if col not in exclude\_columns]  
  
 # Set up X (predictors) and y (target) variables  
 X = logistic\_df[predictors]  
 y = logistic\_df[target]  
  
 # Scale the predictor variables using MinMaxScaler  
 scaler = MinMaxScaler()  
 X\_scaled = scaler.fit\_transform(X)  
  
 # Split the data into training and testing sets  
 split\_index = int(len(logistic\_df) \* 0.8) # 80% train, 20% test  
 X\_train, X\_test = X[:split\_index], X[split\_index:]  
 y\_train, y\_test = y[:split\_index], y[split\_index:]  
  
 # Define and fit the logistic regression model  
 logreg = LogisticRegression(max\_iter=10000)  
 logreg.fit(X\_train, y\_train)  
  
 # Predictions  
 y\_pred = logreg.predict(X\_test)  
  
 # Print all available performance metrics  
 print(f"Performance metrics for {target}:")  
 classification\_rep = metrics.classification\_report(y\_test, y\_pred)  
 print(classification\_rep)  
  
 # Create and save a Seaborn confusion matrix plot  
 plot\_confusion\_matrix(y\_test, y\_pred, f'Confusion Matrix for {target}', cmap)  
  
# Reset warnings to default  
warnings.resetwarnings()

Performance metrics for close\_USD\_change:  
 precision recall f1-score support  
  
 0.0 0.98 0.93 0.95 85  
 1.0 0.95 0.98 0.97 114  
  
 accuracy 0.96 199  
 macro avg 0.96 0.96 0.96 199  
weighted avg 0.96 0.96 0.96 199

A green squares with white text

Description automatically generated

Performance metrics for close\_SAR\_change:  
 precision recall f1-score support  
  
 0.0 0.85 0.98 0.91 84  
 1.0 0.98 0.88 0.93 114  
  
 accuracy 0.92 198  
 macro avg 0.92 0.93 0.92 198  
weighted avg 0.93 0.92 0.92 198

A blue squares with white text

Description automatically generated