Project Code Documentation

Introduction:

In this project, we will be predicting the price of Ethereum using two different algorithms: ARIMA and Facebook Prophet model. We will be using Python programming language for coding, and the libraries we will be using are Pandas, matplotlib, statsmodel.ARIMA, datetime, Prophet.

Step 1: Importing libraries

First, we need to import the necessary libraries. Pandas is used for data manipulation and analysis, matplotlib is used for plotting graphs, statsmodel.ARIMA is used to build ARIMA models, numpy for mathematical operations and datetime is used to work with dates.

Step 2: Importing dataset and reading it with Pandas

Next, we need to import our dataset and read it with Pandas. We will be using the historical price data of Ethereum, which is in the form of a CSV file.

Step 3: Monthly forecasting using resampling function

We will forecast the monthly average prices using the resample function in Pandas. This function will allow us to resample the data to a lower frequency (monthly in this case) and take the average of each month.

Step 4: Seasonal decompose function

To understand the trend and seasonality in our data, we will be using the seasonal decompose function from statsmodels. This will help us identify if our data is stationary or not.

Step 5: Box-cox transformation

If our data is not stationary, we will need to make it stationary before building our model. To do this, we will use the Box-Cox transformation method. Diagnostics from ARIMA modeling can then be used to decide if differencing or seasonal differencing might be useful to to remove polynomial trends or seasonal trends respectively. After that the result might be an ARMA model that is stationary. If diagnostics confirm the orders p and q for the ARMA model, the AR and MA parameters can then be estimated.

Step 6: Dickey-Fuller test

The adfuller() function from the statsmodels library is used to perform the Dickey-Fuller test. We can use the Dickey-Fuller test to check whether our time series data is stationary or not. If the p-value is less than 0.05 we can conclude that the time series is stationary.

Step 7: Applying differencing

If the Dickey-Fuller test indicates that our time series is non-stationary, we can apply differencing to make it stationary. It involves subtracting the value of the time series at a certain time point from the value of the time series at a previous time point. We can apply regular differencing or seasonal differencing, depending on whether there is a seasonal pattern in the data.

Step 8: Autocorrelation plot function

To check for correlation in our time series data, we can use the autocorrelation plot function. This function plots the correlation of the time series with its lagged versions.

Step 9: Finding the best parameters

To build our ARIMA model, we need to find the best parameters. We will use the AIC (Akaike Information Criteria) value to compare different combinations of parameters and select the best one.

Step 10: Model summary function

After finding the best parameters, we can build our ARIMA model using the ARIMA function from statsmodels. We can also check the summary of our model.

Step 11: Plot diagnostics function

To check the performance of our model, we can use the plot_diagnostics function from statsmodels. This function plots the residuals, the kernel density estimate, and the normal distribution.

Step 12: Forecasting Ethereum prices using ARIMA

We fit an ARIMA model to the training data using the best parameters, and then use the predict() function to make predictions for the test data. We also use the inverse_boxcox() function to invert the Box-Cox transformation that we applied earlier. Finally, we plot the actual and predicted values using Matplotlib.
The resulting plot shows the actual and predicted Ethereum prices over time, allowing us to visually compare the accuracy of our ARIMA model.

Step 13: Validation checking error using MSE

To validate the accuracy of our ARIMA model, we can calculate the Mean Squared Error (MSE) between the actual and predicted values. The MSE measures the average squared difference between the predicted and actual values, with lower values indicating better performance.

Step 14: Implementing Facebook Prophet Model

Facebook Prophet is an open source library for time series forecasting developed by Facebook's Core Data Science team. It is designed to be easy to use and can handle a wide range of time series forecasting tasks. To implement the Facebook Prophet model, we first need to install the prophet library using !pip install prophet

Step 15: Examining Correlation between Features

Correlation analysis is an important step in any data analysis task. By examining the correlation between features, we can gain insight into the relationships between different variables and identify any patterns or trends in the data.

To examine the correlation between the features in our Ethereum dataset, we can create a correlation matrix using the corr() function in Pandas

Step 16: Forecasting with Prophet Model

To fit the Prophet model to our Ethereum dataset, we can use the Prophet() function from the fbprophet library. We first need to create a new dataframe with two columns: ds, which contains the dates of the Ethereum prices, and y, which contains the corresponding closing prices. Then, we can fit the model to the data using the fit() function. After fitting the model, we can use the make_future_dataframe() function to create a new dataframe with future dates for which we want to make predictions. We can then use the predict() function to generate forecasts for these dates.

The periods argument specifies the number of future periods for which we want to generate forecasts (in this case, 365 days or one year). The resulting forecast dataframe contains columns for the predicted values (yhat), as well as upper and lower bounds of the prediction intervals (yhat_upper and yhat_lower, respectively). We can visualize the forecasts using the plot() function.

Step 17: Cross-Validation of Prophet Model

To evaluate the performance of the Prophet model, we can use cross-validation. Cross-validation involves fitting the model to a subset of the data (the training set), and then testing it on the remaining data (the validation set). This process is repeated multiple times, with different subsets of the data used for training and validation each time.

To perform cross-validation on the Prophet model, we can use the cross_validation() function from the prophet library. This function takes as input the model, the initial training period, the length of the testing period, the length of the forecast horizon, and the size of the sliding window used for the training set. The performance_metrics() function in the Facebook Prophet library returns a dataframe containing various performance metrics for each fold of the cross-validation, including the mean squared error (MSE), the root mean squared error (RMSE), and other metrics such as the mean absolute percentage error (MAPE) and the symmetric mean absolute percentage error (SMAPE).