Forecasting Stock Prices using XGBoost A Step-By-Step Walk-Through

Photo by Jamie Street on Unsplash

There are many machine learning techniques in the wild, but extreme gradient boosting (XGBoost) is one of the most popular. Gradient boosting is a process to convert weak learners to strong learners, in an iterative fashion. The name XGBoost refers to the engineering goal to push the limit of computational resources for boosted tree algorithms. Ever since its introduction in 2014, XGBoost has proven to be a very powerful machine learning technique and is usually the go-to algorithm in many Machine Learning competitions.

In this article, we will experiment with using XGBoost to forecast stock prices. We have experimented with XGBoost in a previous article, but in this article, we will be taking a more detailed look at the performance of XGBoost applied to the stock price prediction problem. We list down the main differences between this article and the previous one below:

- In the previous article, we predict only for **1 day**, but here we predict for the next **21 days** (note there are about 21 trading days in a month, excluding weekends). To do so, we use a technique known as **recursive forecasting**.
- In the previous article, we used a simple train-validation-test split, but here we used the **moving window validation method** to perform hyperparameter tuning.
- In the previous article, the number of previous days to use as lag features (denoted as *N*) was treated as a hyperparameter to be tuned. But here we set *N*=14 and let the model infer for itself which lag period is more important for prediction.
- In the previous article, we used only the prices of the previous *N* days as features, but here we do more **feature engineering** and introduce more features.

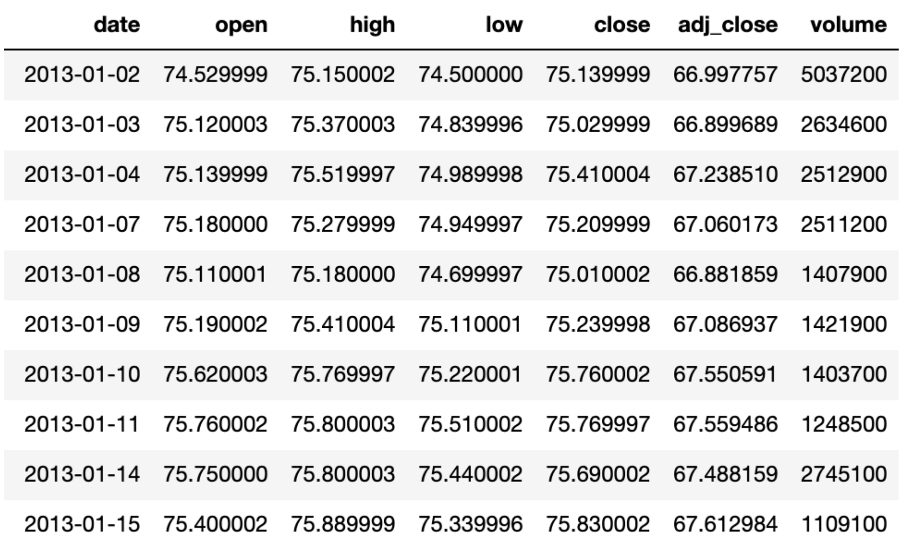
In the rest of this article, we will walk through the standard steps of a machine learning project, with a focus on our stock price prediction problem:

Problem Statement
Exploratory Data Analysis
Training, Validation, Test split
Feature Engineering
Feature Scaling
Hyperparameter Tuning
Applying the Model
Findings

It is worthwhile to be aware that there are other steps of a machine learning project not mentioned here such as data cleaning (not an issue here), missing values imputation (no missing values here), and feature selection (well, we don't have a lot of features here). These topics are important as well, but are not an issue in our problem, as you will see below.

Problem Statement

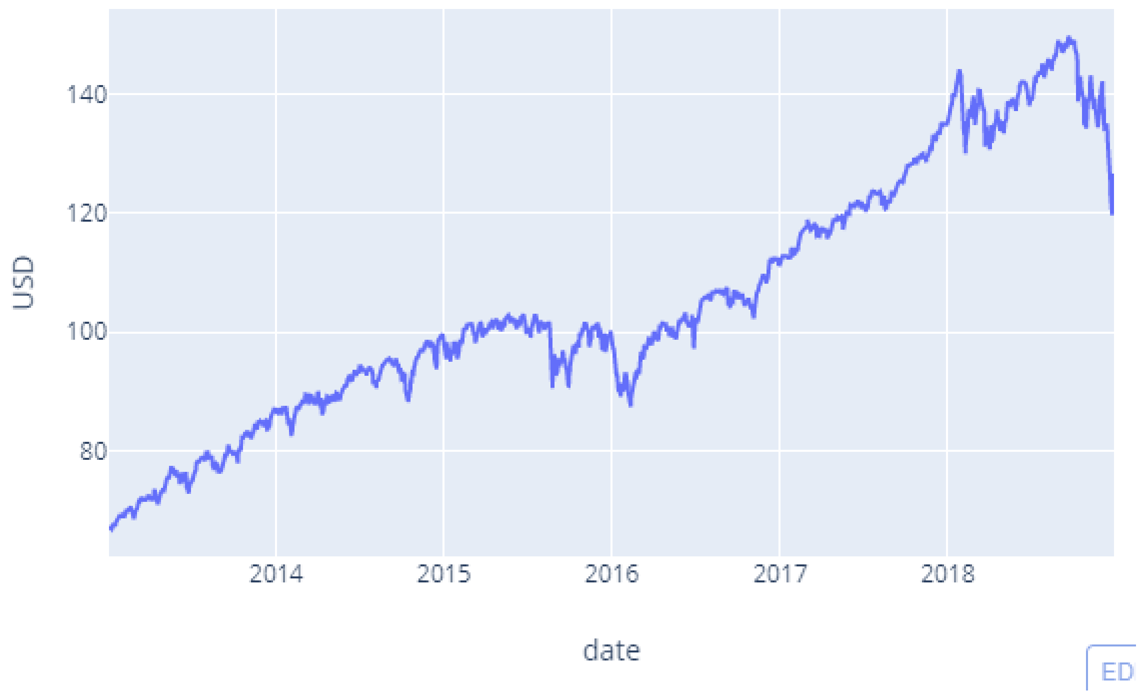
It is very important to define the problem statement clearly before you start any work. Here, we aim to predict the daily adjusted closing prices of Vanguard Total Stock Market ETF (VTI), using data from the previous *N* days. In this experiment, we will use 6 years of historical prices for VTI from 2013–01–02 to 2018–12–28, which can be easily downloaded from yahoo finance. After downloading, the dataset looks like this:



date	open	high	low	close	adj_close	volume
2013-01-02	74.529999	75.150002	74.500000	75.139999	66.997757	5037200
2013-01-03	75.120003	75.370003	74.839996	75.029999	66.899689	2634600
2013-01-04	75.139999	75.519997	74.989998	75.410004	67.238510	2512900
2013-01-07	75.180000	75.279999	74.949997	75.209999	67.060173	2511200
2013-01-08	75.110001	75.180000	74.699997	75.010002	66.881859	1407900
2013-01-09	75.190002	75.410004	75.110001	75.239998	67.086937	1421900
2013-01-10	75.620003	75.769997	75.220001	75.760002	67.550591	1403700
2013-01-11	75.760002	75.800003	75.510002	75.769997	67.559486	1248500
2013-01-14	75.750000	75.800003	75.440002	75.690002	67.488159	2745100
2013-01-15	75.400002	75.889999	75.339996	75.830002	67.612984	1109100

Downloaded dataset for VTI.

Altogether, we have 1509 days of data to play with. Note that Saturdays and Sundays are not included in the dataset above. A plot of the adjusted closing price in the entire dataset is shown below:



Adjusted closing prices from 2013–01–02 to 2018–12–28.

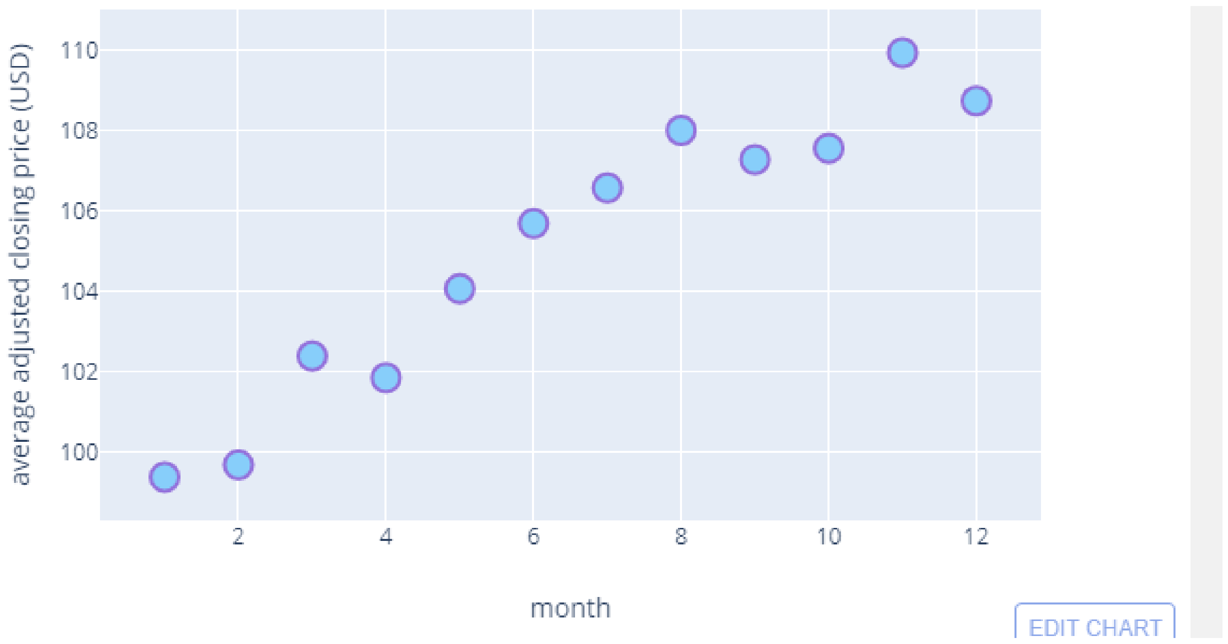
To effectively evaluate the performance of XGBoost, running one forecast at a single date is not enough. Instead, we will perform various forecasts at different dates in this dataset, and average the results.

To evaluate the effectiveness of our methods, we will use the root mean square error (RMSE), mean absolute percentage error (MAPE) and mean absolute error (MAE) metrics. For all metrics, the lower the value, the better the prediction. Similar to our previous article, we will use the Last Value method to benchmark our results.

Exploratory Data Analysis (EDA)

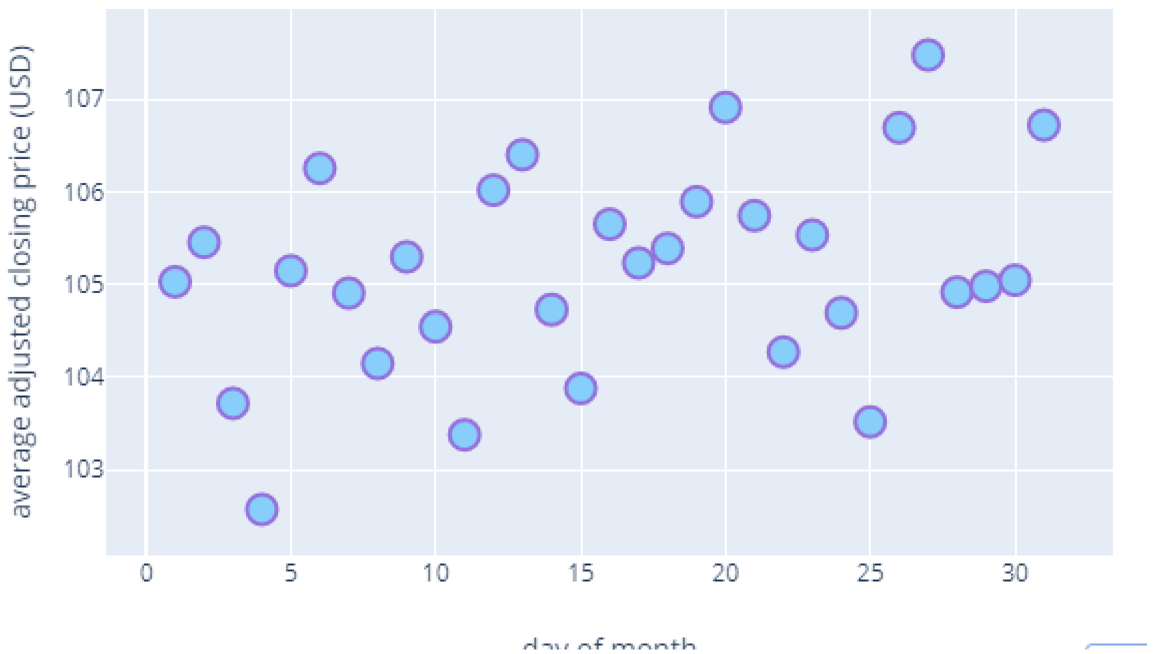
EDA is an essential part of a machine learning project to help you get a good 'feel' for a dataset. If you participate in Machine Learning competitions (or plan to), **extensive** EDA may help you generate better features or even discover 'information leaks' which can help you climb the leaderboard rankings. As we will see below, the EDA process involves creating visualizations to help you understand the dataset better.

The plot below shows the average adjusted closing price for each month. We can infer that based on our dataset, on average, later months have a higher value than earlier months.



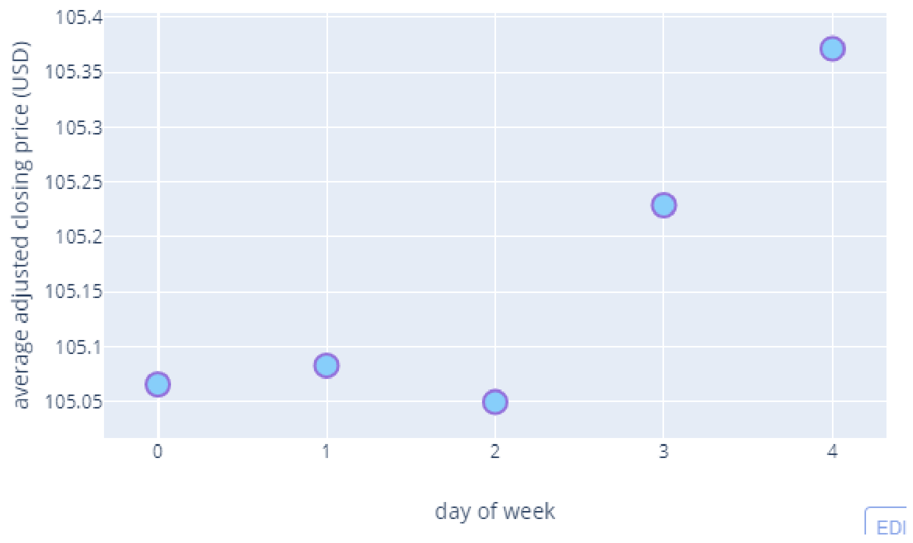
Average adjusted closing price by month.

The plot below shows the average adjusted closing price for each day of the month. On average, there is an upward sloping trend, where the later days of the month have a higher price than the earlier days.



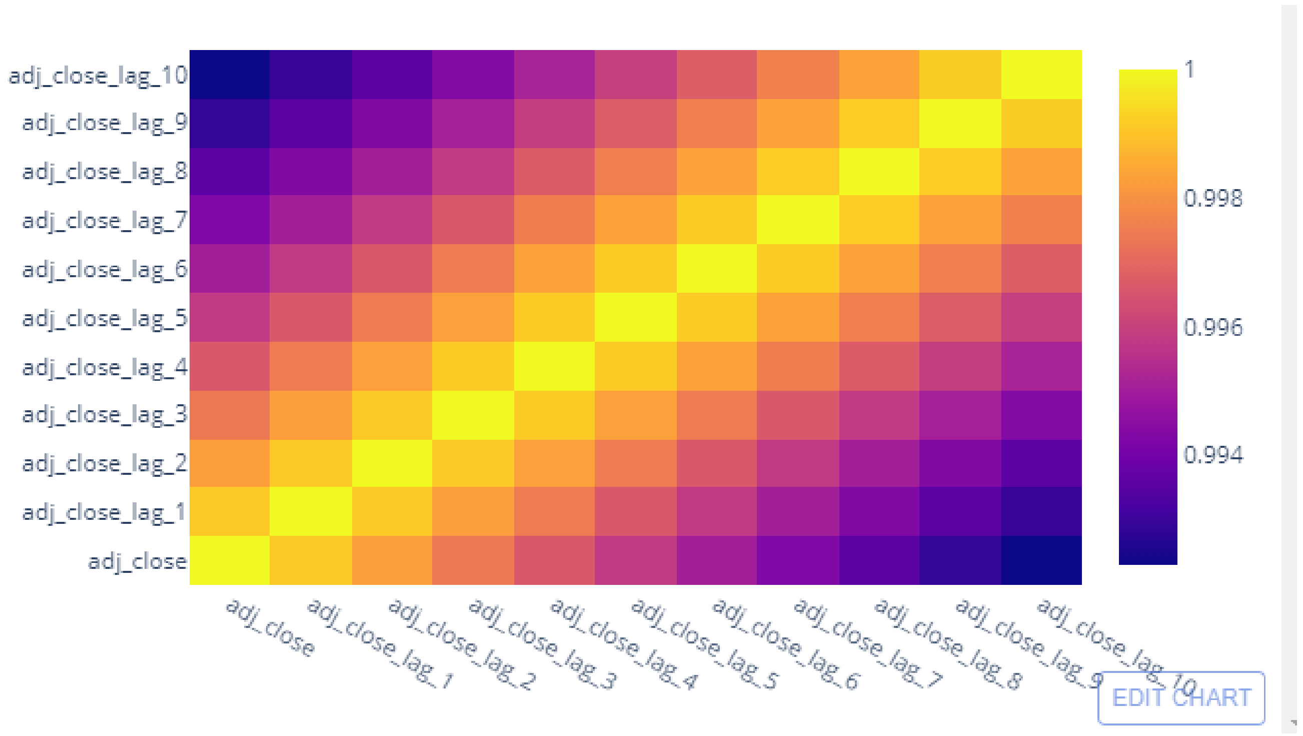
Average adjusted closing price by day of month.

The plot below shows the average adjusted closing price for each day of the week. On average, adjusted closing prices are higher for Thursdays and Fridays than other days of the week.



Average adjusted closing price by day of week.

The heatmap below shows the correlation of previous days' adjusted closing prices with the current day's. It is clear that the nearer the adjusted closing price is to the current day, the more highly correlated they are. Therefore, features relating to adjusted closing prices of the previous 10 days should be used in the prediction.



Correlation heatmap for the lag features.

Based on the EDA above, we infer that features related to dates might be helpful to the model. Further, adjusted closing prices of the previous 10 days are highly correlated to the target variable. These are important information that we will use for feature engineering below.

Feature Engineering

Feature engineering is a creative process and is one of the most important parts of any machine learning project. To highlight the importance of feature engineering, there is a nice quote from Andrew Ng which is worth sharing (from Wikipedia):

Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.

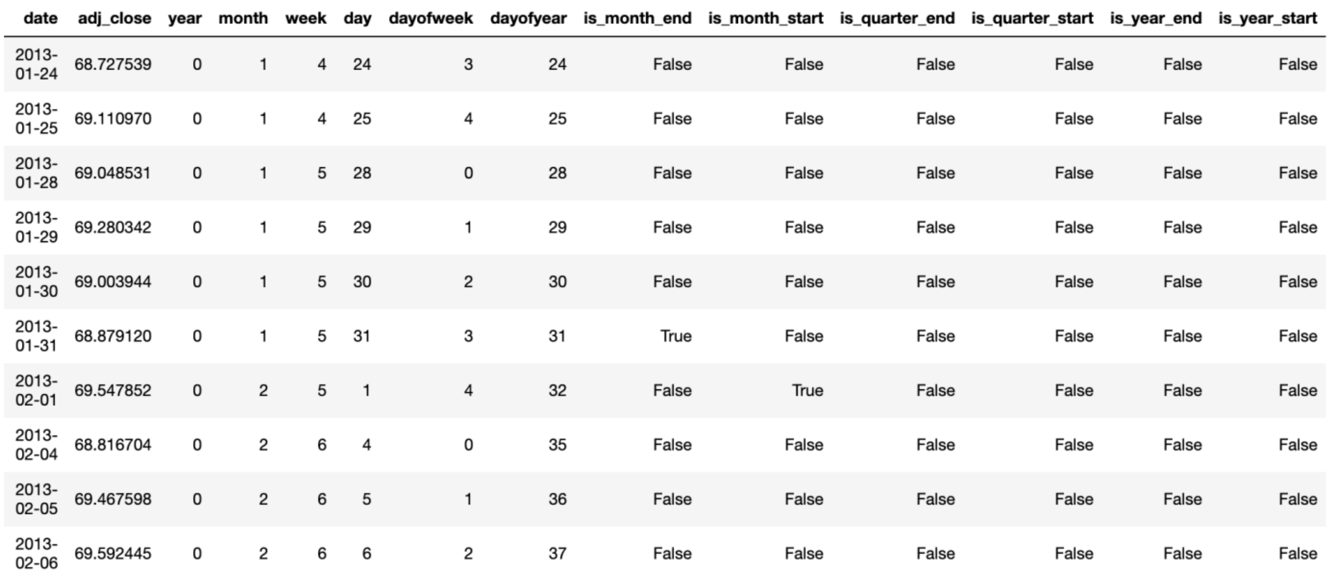
— *Andrew Ng*, Machine Learning and AI via Brain simulations

In this project, we will generate the following features:

adjusted closing prices of the last N=10 days
year
month
week
dayofmonth
dayofweek
dayofyear
is_month_end
is_month_start
is_quarter_end
is_quarter_start
is_year_end
is_year_start

The features relating to dates are easily generated using the fastai package as such:

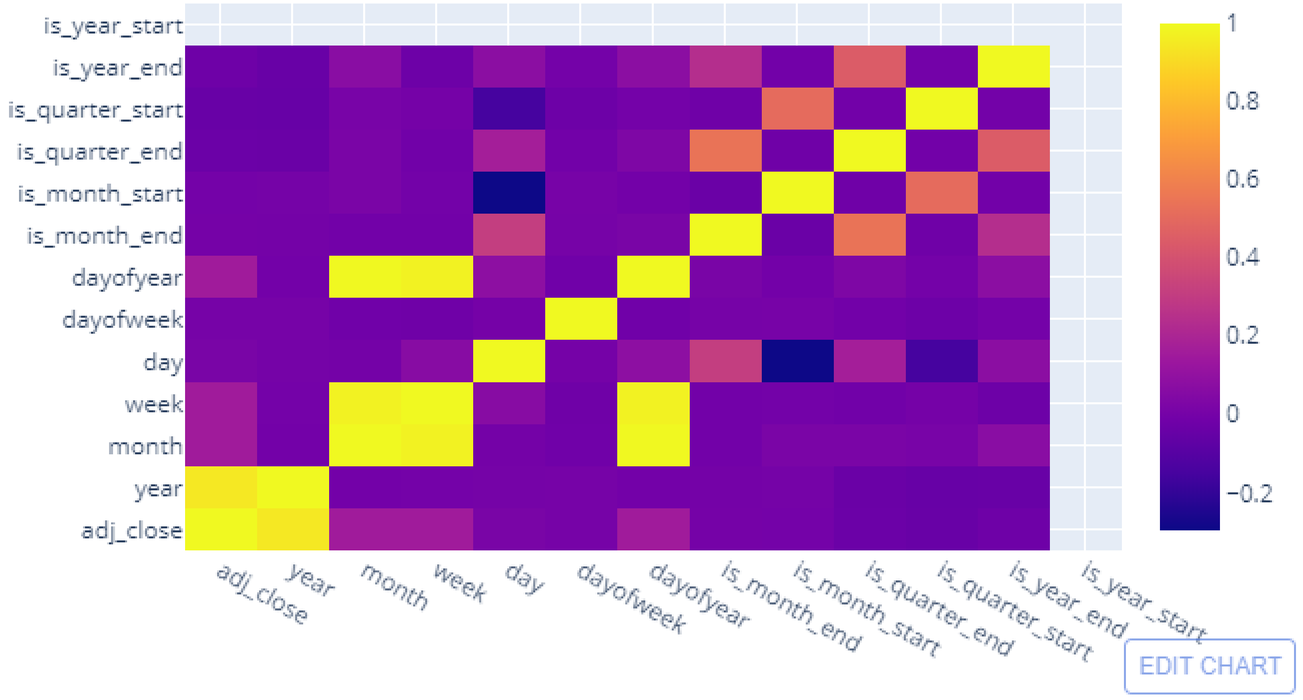
After using the code above the dataframe looks like the below. The column *adj_close* will be the target column. Features relating to adjusted closing prices of the last *N* days are omitted for brevity.



date	adj_close	year	month	week	day	dayofweek	dayofyear	is_month_end	is_month_start	is_quarter_end	is_quarter_start	is_year_end	is_year_start
2013-01-24	68.727539	0	1	4	24	3	24	False	False	False	False	False	False
2013-01-25	69.110970	0	1	4	25	4	25	False	False	False	False	False	False
2013-01-28	69.048531	0	1	5	28	0	28	False	False	False	False	False	False
2013-01-29	69.280342	0	1	5	29	1	29	False	False	False	False	False	False
2013-01-30	69.003944	0	1	5	30	2	30	False	False	False	False	False	False
2013-01-31	68.879120	0	1	5	31	3	31	True	False	False	False	False	False
2013-02-01	69.547852	0	2	5	1	4	32	False	True	False	False	False	False
2013-02-04	68.816704	0	2	6	4	0	35	False	False	False	False	False	False
2013-02-05	69.467598	0	2	6	5	1	36	False	False	False	False	False	False
2013-02-06	69.592445	0	2	6	6	2	37	False	False	False	False	False	False

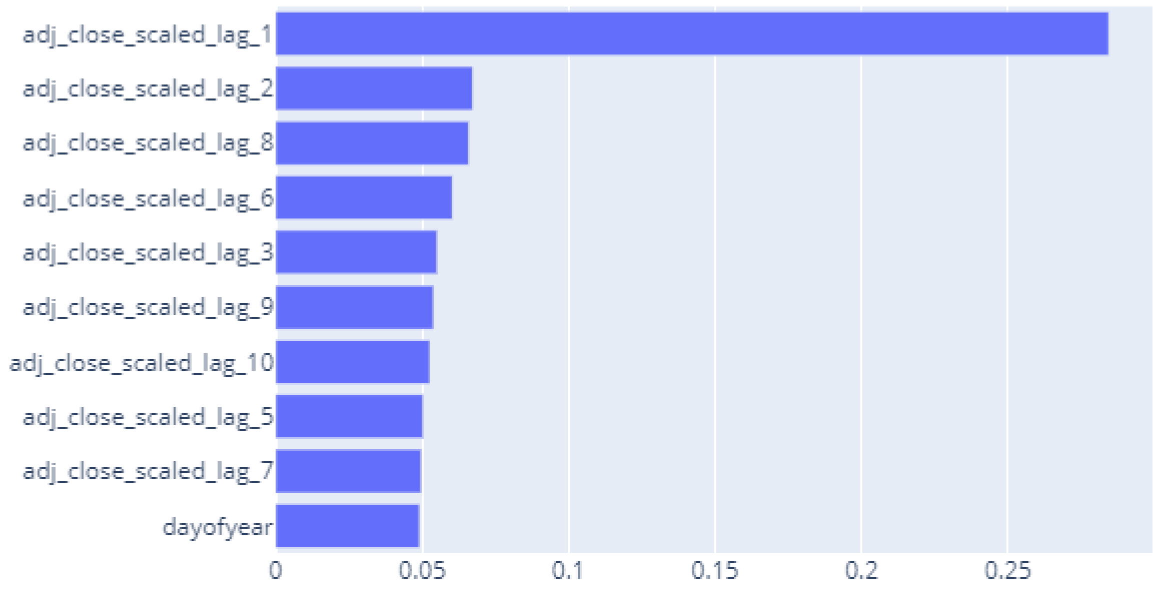
Dataframe containing the target column and date features.

The heatmap below shows the correlation of the features with the target column. The feature *year* is quite highly correlated with the adjusted closing price. This is unsurprising, because in our dataset, there is an upward-sloping trend where the larger the year, the higher the adjusted closing price. Other features do not exhibit a high correlation with the target variable. From the below, we also found that the feature *is_year_start* has all NaNs. This is because the first day of the year is never a trading day, and so we remove this feature from the model.



Correlation heatmap for date features.

Below is a bar chart showing the importance scores of the top 10 most important features. This is obtained for the forecast of 2017–01–03, and forecasts on other dates may have different ranking of the feature importance. As expected, the adjusted closing price of the previous day is the most important feature.



Top 10 most important features for the forecast of 2017–01–03.

Training, Validation, and Test

To perform a forecast, we need training and validation data. We will use 3 years of data as the train set, which corresponds to 756 days since there are about 252 trading days in a year (252*3 = 756). We will use the next 1 year of data to perform validation, which corresponds to 252 days. In other words, for each forecast we make, we need 756+252 = 1,008 days of data for model training and validation. The model will be trained using the train set, and model hyperparameters will be tuned using the validation set.

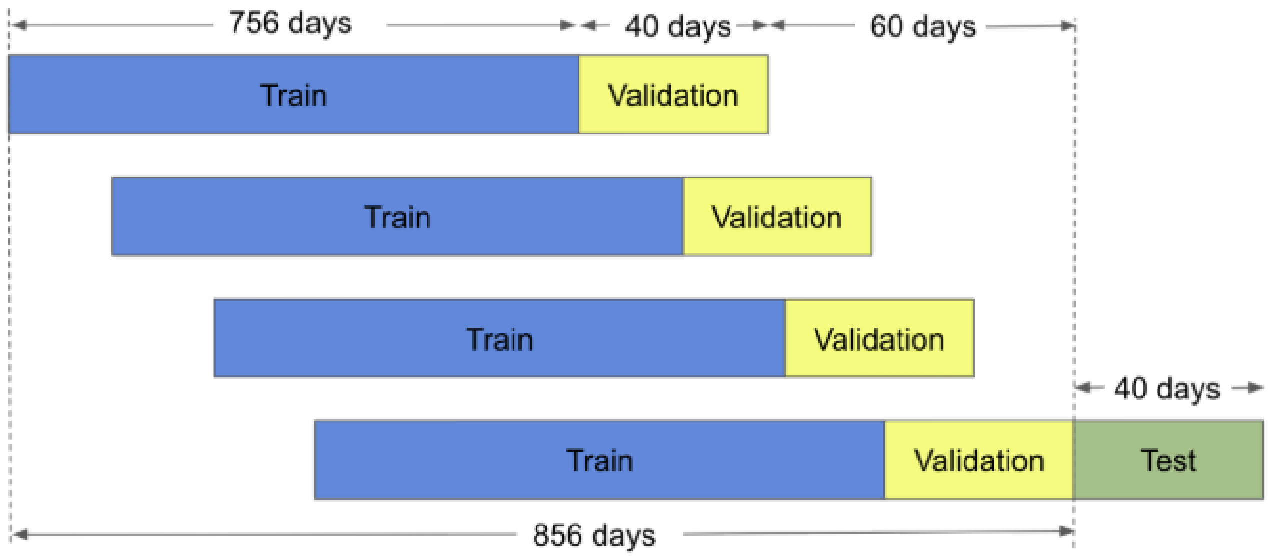
To tune the hyperparameters, we will use the moving window validation method. The moving window validation method has been described in detail in one of our previous articles:

Forecasting Stock Prices using Prophet

Forecasting is a hard science and requires substantial expertise. For these reasons Facebook open-sourced Prophet…

towardsdatascience.com

An example is illustrated below, for the case of train size of 756 days, validation size of 40 days, and a forecast horizon of 40 days.



Moving window validation example.

It is very important in time series prediction that the train, validation, test splits have to be in chronological order. Failure to do so will result in **'information leak'** in the model, which is defined as the scenario where the model is trained on data that provides information about the test set. For example, if we have the open price for today and we are trying to predict for the closing price yesterday, immediately we can set our prediction to be equal to the open price of today and we should get pretty good results. The end result is that our model will give better performance than can be expected in real life. Therefore, when building a machine learning model, we need to be very careful about information leaks.

In what follows, we will use XGBoost to perform forecasts on several days in our test set, namely:

2017–01–03
2017–03–06
2017–05–04
2017–07–05
2017–09–01
2017–11–01
2018–01–03
2018–03–06
2018–05–04
2018–07–05
2018–09–04
2018–11–01

For each of the 12 forecasts above, we will use a forecast horizon of 21 days. We will use the 1008 days immediately prior to the forecast date as training and validation set, with a 756:252 split as mentioned above.

Feature Scaling

Feature scaling is important here because if you were to look at the plot of adjusted closing prices above, splitting the train and test sets in a chronological split almost always results in the adjusted closing prices of the test set having a higher value than the train set. What this means is that a model trained without scaling the adjusted closing prices will only output predictions around the range of the prices in the train set. This has also been explained in our previous article:

Machine Learning Techniques applied to Stock Price Prediction

Machine learning has many applications, one of which is to forecast time series. One of the most interesting (or…

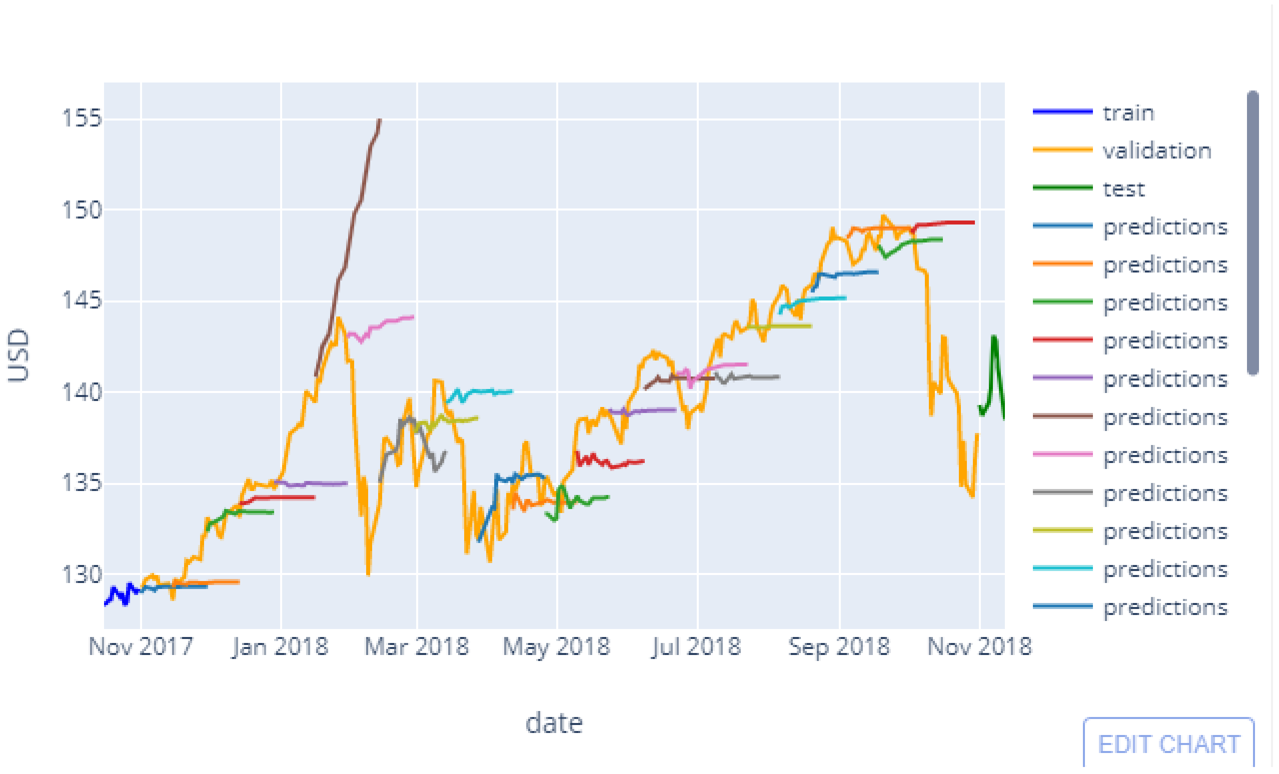
towardsdatascience.com

We have experimented with various techniques and in this article, we will use the method found from the above that has the best performance. For each feature group of adjusted closing prices (the lag features) of each sample, we will scale them to have mean 0 and variance 1. For example, if we are doing predictions on day *T*, we will take the adjusted closing prices of the last *N* days (days *T*-*N* to *T*-1) and scale them to have mean 0 and variance 1. The same is done on the train, validation, and test sets for the lag features. The date features are not scaled. We then use these scaled lag features and dates features to do prediction. The predicted values will also be scaled and we inverse transform them using their corresponding mean and variance.

Hyperparameter Tuning

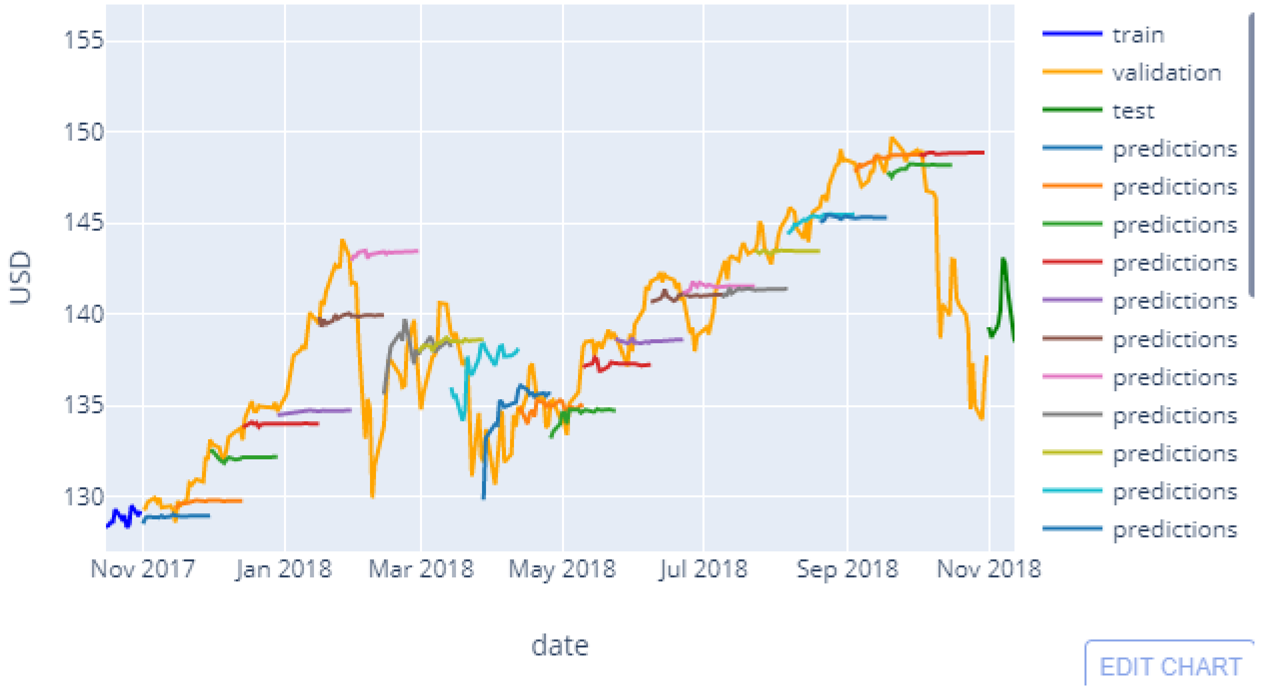
We perform hyperparameter tuning on the validation set. For XGBoost, there are several hyperparameters that can be tuned including *n_estimators*, *max_depth*, *learning_rate*, *min_child_weight*, *subsample*, *gamma*, *colsample_bytree*, and *colsample_bylevel*. For a definition of each of these hyperparameters, see here.

To look at the effectiveness of hyperparameter tuning, we can look at the predictions on our validation set for the forecast of 2018–11–01. Below show the predictions without hyperparameter tuning, where we just use the default values from the package:



Predictions on the validation set without hyperparameter tuning.

Below shows the predictions on the same validation set after hyperparameter tuning. You can see the wild prediction on 18 Jan is much more stable now.



Predictions on the validation set with hyperparameter tuning.

Below shows the hyperparameters before and after tuning:

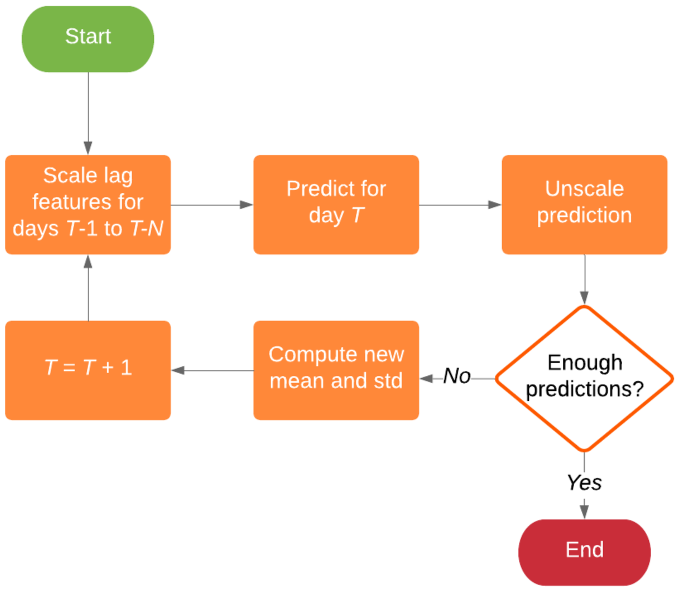
param	before_tuning	after_tuning
n_estimators	100.000	11.000
max_depth	3.000	9.000
learning_rate	0.100	0.200
min_child_weight	1.000	14.000
subsample	1.000	1.000
colsample_bytree	1.000	1.000
colsample_bylevel	1.000	1.000
gamma	0.000	0.000
val_rmse	3.395	2.984
val_mape	2.053	1.833
val_mae	2.831	2.536

XGBoost hyperparameters before and after tuning.

Clearly, the tuned hyperparameters differ a lot from the default values. Also, after tuning the RMSE, MAPE and MAE of the validation drops as expected. For example, RMSE drops from 3.395 to 2.984.

Applying the Model

Having performed EDA, feature engineering, feature scaling, and hyperparameter tuning as explained above, we are now ready to perform our forecasts on the test set. In this case, we have a forecast horizon of 21 days which means we need to generate 21 predictions for each forecast. We cannot generate all 21 predictions at one go, because after generating the prediction for day *T*, we need to feedback this prediction into our model to generate the prediction for day *T*+1, and so on until we have all 21 predictions. This is known as **recursive forecasting**. Therefore, we implement a logic like the flowchart below:



Flowchart for recursive forecasting.

For each day in the forecast horizon, we need to predict, unscale the prediction, compute the new mean and standard deviation of the last *N* values, scale the adjusted closing prices of the last *N* days, and predict again.

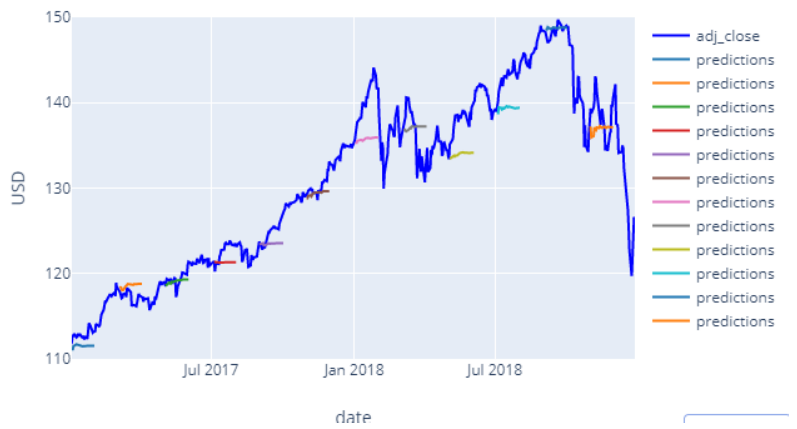
Findings

Below shows the RMSE, MAPE, and MAE of each forecast, along with their corresponding (selected) optimum hyperparameters tuned using their respective validation sets.

date	RMSE	MAPE(%)	MAE	n_estimators	max_depth	learning_rate	min_child_weight
2017-01-03	1.491535	1.227341	1.388315	33.0	3.0	0.20	15.0
2017-03-06	1.522797	1.147115	1.340978	11.0	9.0	0.30	11.0
2017-05-04	0.895876	0.553540	0.661084	13.0	7.0	0.30	6.0
2017-07-05	1.732334	1.236195	1.522898	21.0	7.0	0.10	13.0
2017-09-01	1.754289	1.217536	1.527847	37.0	7.0	0.10	11.0
2017-11-01	1.270258	0.677038	0.888327	39.0	7.0	0.20	18.0
2018-01-03	5.114899	3.334857	4.713266	7.0	9.0	0.30	17.0
2018-03-06	3.439315	2.123761	2.857635	51.0	3.0	0.05	20.0
2018-05-04	4.221913	2.950227	4.082444	23.0	2.0	0.30	20.0
2018-07-05	3.948716	2.666547	3.825811	33.0	4.0	0.30	16.0
2018-09-04	0.738313	0.373173	0.552304	29.0	5.0	0.20	5.0
2018-11-01	2.959072	1.789224	2.497811	11.0	9.0	0.20	14.0

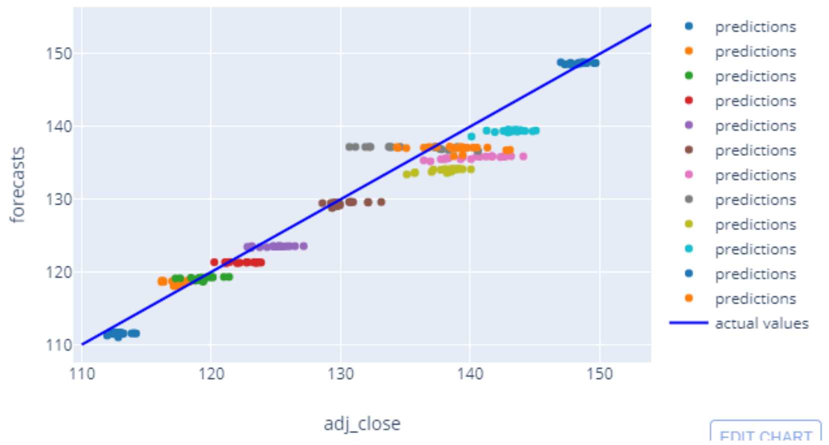
RMSE, MAPE, and MAE of each forecast made with XGBoost.

The results of applying XGBoost on our test set using the moving window validation method are shown below.



Forecasts on the test sets using XGBoost.

Another way of visualizing the forecasts is to plot each forecast with its actual value. This is shown in the plot below. If we have perfect accuracy, each forecast should lie on the diagonal y=x line.



Comparing the forecasts using XGBoost with their actual values.

Finally, here are the results of our model benchmarked against the Last Value method:

Method	RMSE	MAPE(%)	MAE
Last value	2.53	1.69	2.26
XGBoost w/o date features	2.32	1.53	2.05
XGBoost w date features	2.42	1.61	2.15

Final results.

Using XGBoost with or without the date features give a better performance over the Last Value method. Interestingly, omitting the date features gives a slightly lower RMSE than including the date features (2.32 vs. 2.42). As we have found earlier, the date features have a low correlation with the target variable and likely do not help the model much.

You can check out the Jupyter notebooks for XGBoost without date features here, and XGBoost with date features here.

We hope you enjoyed the article above, where we worked and pondered through a real-life dataset rather than a simple textbook example. The intricacies of the recursive forecasting mechanism took us longer than expected, but it was fun. Another important point of this article is to show that there are many decisions to make in building a machine learning model, which makes it very much an art as well as a science. Feel free to leave your comments below!