ETF Price Forecaster - AWS ML Engineering Capstone Project

Marco Sampaio[1]

[1]Affiliation not available

January 18, 2025

Abstract

We develop a machine learning model to forecast Exchange Trade Fund (ETF) indices. We investigate several types of models ranging from simple baselines to ARIMA and a LightGBM models based on features extracted from the time series. The best model we find is a LightGBM model (gain of 6% over the baseline) which we deploy on AWS as an interactive dashboard to visualize the forecasts and the historical data for ETF symbols selected by the user. The dashboard will be available during the assessment of this report at http://dashboard-lb-1124043407.us-east-1.elb.amazonaws.com:8050 (if having problems accessing it try this earlier version of the deployment at http://dashboard-lb-461967644.us-east-1.elb.amazonaws.com:8050/).

Contents

1 Introduction **1**
1.1 Problem Statement 1
1.2 Structure of the report 2

2 Data Exploration **2**
2.1 ETFs selection 2
2.2 4
2.3 Exploratory time series data analysis 5

3 Methods **12**
3.1 Models 12
3.1.1 Baselines 12
3.1.2 ARIMA models 12
3.1.3 LightGBM Models 12
3.1.4 Other approaches 13
3.2 Evaluation Metrics 13
3.2.1 Loss for training 13
3.2.2 Model evaluation 13

4 Model Development **14**
4.1 Data Splitting 14
4.2 Hyperparameter Tuning 14
4.3 Test set Evaluation 18

5 Dashboard App Development 18
5.1 Architecture 18
5.2 lmplementation Details 20
5.2.1 AMI creation script 20
5.2.2 Automated model retraining script 21
5.2.3 Dashboard deployment script 21
6 Conclusion 22

1 Introduction

Exchange-Traded Funds (ETFs) [1]; [2] are investment instruments consisting of a basket of securities—such as stocks, bonds, or commodities—designed to track the performance of a specific index, sector, or asset class. ETFs are traded on stock exchanges, allowing investors to buy and sell shares throughout the day at market prices.

ETFs are an excellent tool for individuals to manage long-term savings due to their ability to provide diversified exposure to various asset classes at a low cost [3]. By investing in ETFs that track broad market indices, such as the S&P 500, individuals can achieve consistent growth over time, benefiting from market trends and compounding returns at low management fees and tax efficiency, making them a popular choice for retirement planning and other financial goals.

1.1 Problem Statement

In this project, we propose to develop an interactive dashboard that shows the evolution of a selection of long term high performing ETF indices, together with Machine Learning (ML) model based forecasts (in a relatively short time window e.g., 1 month). We have in mind the use case of an individual who:

- has some idea on the domains they want to invest in (e.g., they may be thinking about keeping half their savings in an index that tracks the overall market, like S&P 500, and distribute the remainder in the tech sector, robotics, health, etc.);
- is interested in regularly investing their savings in such funds (e.g., several times a year);
- wants to decide the best time during a period (e.g., one month) to buy, if they are investing, or sell, if they are withdrawing their funds to use them.

Thus, overall, such an individual is not interested in having a detailed real time forecast of what's happening to the market throughout the day, but it is interested in the overall trend in a window of a couple of weeks to make an informed decision on how low they should set their buying price, or how high their selling price so that their order gets executed within the time frame they have set while not executing it at a very unfavorable market fluctuation.

More specifically, the problem can be stated as follows. Given a historical time series dataset with a set of closing prices $\{x_{i,t}\}$ for the ETF index i on business day t we want to predict the future prices $\{\hat{x}_{i,t}\}$ for all the indices.

1.2 Structure of the report

This report is split in 4 main parts. In section \ref{162904} we provide a data exploration which serves two purposes: i) we select a short list of 100 symbols to work with while describing the type of investor we are focusing one, and ii) we explore the properties of the time series to build intuition on what should be relevant for modeling. In section \ref{158856} we present the types of models we will train as well as the evaluation metrics. Section \ref{106179} consists of the main results regarding model development, where we conclude on the final model parameters. Finally, in section \ref{615612} we describe the engineering of the components needed for the final deployment in AWS. Our conclusions are summarized in section \ref{370266}.

2 Data Exploration

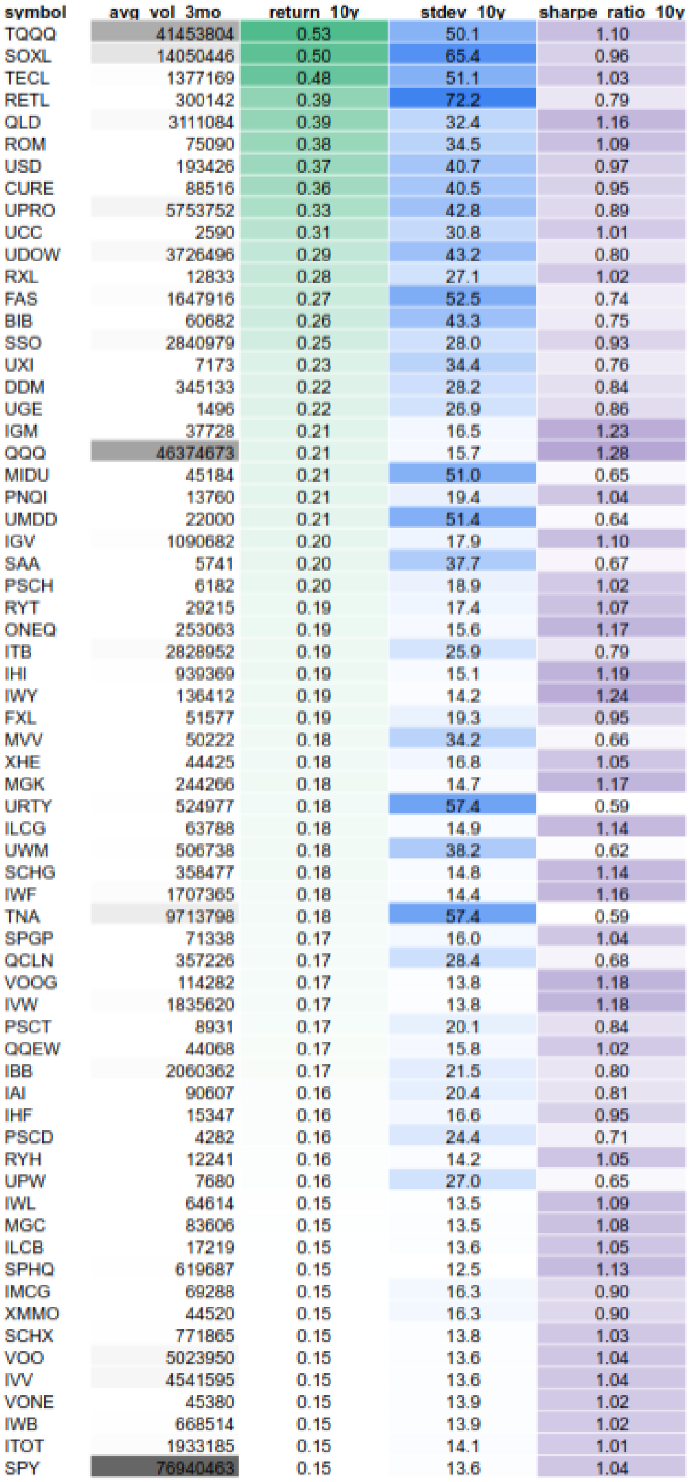
The first step in addressing any ML problem is to collect the relevant data, analyse it and clean it.

2.1 ETFs selection

In our use case we are interested in an investor who is particularly risk averse so they will want to invest in ETFs that are more main stream and with a long enough history of high long term returns. Their goal is to use ETFs as an instrument for long term savings to grow. Thus, in this section we start with an analysis of a few statistics using, for convenience, a comprehensive publicly available Kaggle dataset [4] that has been scrapped from yahoo finance with thousands of ETF symbols in the US. This dataset already contains several statistics that describe the characteristics and performance of the ETF symbols scrapped.

Though the dataset we use for this initial exploratory analysis is slightly outdated, it serves the purpose of allowing us to iterate fast and short list a reasonable set of supported symbols, avoiding the implementation overhead of several statistics used to describe asset performance. We do not expect this to affect the overall quality of the solution we will develop later (also because we focus on selecting long term stable symbols). We select about 100 symbols which should ensure a reasonable number of mainstream symbols that perform well. A notebook with the details of this analysis is found in the repo at experiments/data_exploration/etfs_selection.ipynb. experiments/data_exploration/etfs_selection.ipynb.

In figure \ref{488391} we show a summary table of the symbols with the highest annualized returns over 10 years. We also display their 10 year annualized volatility (stdev_10y) average daily volume over 3 months and 10 year sharpe ratio. We only show the top symbols with returns above or equal to the S&P 500 (SPY), for simplicity.



symbol	avg vol 3mo	return 10y	stdev 10y	sharpe ratio 10y
TQQQ	41453804	0.53	50.1	1.10
SOXL	14050446	0.50	65.4	0.96
TECL	1377169	0.48	51.1	1.03
RETL	300142	0.39	72.2	0.79
QLD	3111084	0.39	32.4	1.16
ROM	75090	0.38	34.5	1.09
USD	193426	0.37	40.7	0.97
CURE	88516	0.36	40.5	0.95
UPRO	5753752	0.33	42.8	0.89
UCC	2590	0.31	30.8	1.01
UDOW	3726496	0.29	43.2	0.80
RXL	12833	0.28	27.1	1.02
FAS	1647916	0.27	52.5	0.74
BIB	60682	0.26	43.3	0.75
SSO	2840979	0.25	28.0	0.93
UXI	7173	0.23	34.4	0.76
DDM	345133	0.22	28.2	0.84
UGE	1496	0.22	26.9	0.86
IGM	37728	0.21	16.5	1.23
QQQ	46374673	0.21	15.7	1.28
MIDU	45184	0.21	51.0	0.65
PNQI	13760	0.21	19.4	1.04
UMDD	22000	0.21	51.4	0.64
IGV	1090682	0.20	17.9	1.10
SAA	5741	0.20	37.7	0.67
PSCH	6182	0.20	18.9	1.02
RYT	29215	0.19	17.4	1.07
ONEQ	253063	0.19	15.6	1.17
ITB	2828952	0.19	25.9	0.79
IHI	939369	0.19	15.1	1.19
IWY	136412	0.19	14.2	1.24
FXL	51577	0.19	19.3	0.95
MVV	50222	0.18	34.2	0.66
XHE	44425	0.18	16.8	1.05
MGK	244266	0.18	14.7	1.17
URTY	524977	0.18	57.4	0.59
ILCG	63788	0.18	14.9	1.14
UWM	506738	0.18	38.2	0.62
SCHG	358477	0.18	14.8	1.14
IWF	1707365	0.18	14.4	1.16
TNA	9713798	0.18	57.4	0.59
SPGP	71338	0.17	16.0	1.04
QCLN	357226	0.17	28.4	0.68
VOOG	114282	0.17	13.8	1.18
IVW	1835620	0.17	13.8	1.18
PSCT	8931	0.17	20.1	0.84
QQEW	44068	0.17	15.8	1.02
IBB	2060362	0.17	21.5	0.80
IAI	90607	0.16	20.4	0.81
IHF	15347	0.16	16.6	0.95
PSCD	4282	0.16	24.4	0.71
RYH	12241	0.16	14.2	1.05
UPW	7680	0.16	27.0	0.65
IWL	64614	0.15	13.5	1.09
MGC	83606	0.15	13.5	1.08
ILCB	17219	0.15	13.6	1.05
SPHQ	619687	0.15	12.5	1.13
IMCG	69288	0.15	16.3	0.90
XMMO	44520	0.15	16.3	0.90
SCHX	771865	0.15	13.8	1.03
VOO	5023950	0.15	13.6	1.04
IVV	4541595	0.15	13.6	1.04
VONE	45380	0.15	13.9	1.02
IWB	668514	0.15	13.9	1.02
ITOT	1933185	0.15	14.1	1.01
SPY	76940463	0.15	13.6	1.04

Figure 1: Historical performance for symbols with top 10 year return down to the S&P 500 index

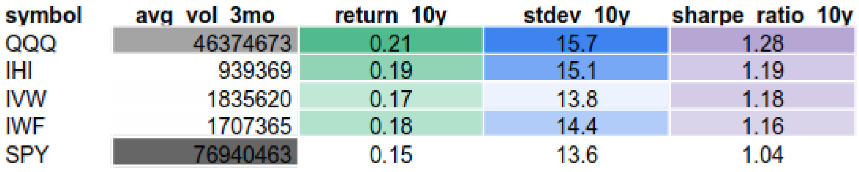
We observe that:

- The annualized returns of all these symbols range from 15% to 53%.
- A large proportion have a high volume, which means there are plenty of mainstream symbols that trade at high volume.
- The volatility tends to be larger for larger returns, which is expected (higher gains usually imply taking more risk).
- The sharpe ratio, which is a measure of tradeoff between returns and volatility is larger than 1 for the majority of symbols and it is usually high for symbols with higher return and smaller volatility.

Given these observations, we illustrate two scenarios of an investor who selects a few symbols to have in their protfolio.

Scenario 1: We consider an investor that is thinking of a very long term investment (> 10 years) and is risk averse. They want to select the symbols with large returns, large sharpe ratio and that are main stream (large average daily volume). In figure \ref{892879} we show the selection they get when applying these criteria.

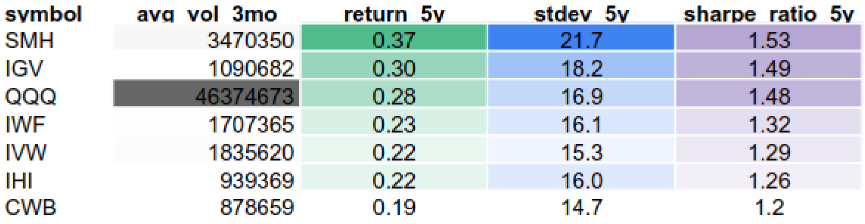
2.2



symbol	avg_vol_3mo	return_10y	stdev_10y	sharpe_ratio_10y
QQQ	46374673	0.21	15.7	1.28
IHI	939369	0.19	15.1	1.19
IVW	1835620	0.17	13.8	1.18
IWF	1707365	0.18	14.4	1.16
SPY	76940463	0.15	13.6	1.04

Figure 2: Scenario 1 - High sharpe ratio, 10 year returns >= SPY, low stdev (< 16%) and large enough volume (> 1% of SPY)

Scenario 2: We consider an investor that is thinking of a shorter term term investment (˜ 5years) and can take up a little more risk, while still wanting to select the symbols with large returns, large sharpe ratio and that are main stream (large average daily volume). In figure \ref{932441} we show the selection they get when applying these criteria.



symbol	avg_vol_3mo	return_5y	stdev_5y	sharpe_ratio_5y
SMH	3470350	0.37	21.7	1.53
IGV	1090682	0.30	18.2	1.49
QQQ	46374673	0.28	16.9	1.48
IWF	1707365	0.23	16.1	1.32
IVW	1835620	0.22	15.3	1.29
IHI	939369	0.22	16.0	1.26
CWB	878659	0.19	14.7	1.2

Figure 3: Scenario 2- High sharpe ratio , 5 year returns >= SPY, low stdev (< 22%) and large enough volume (> 1% of SPY)

In the second scenario we see that a few other symbols with higher return and risk appear in addition to those in scenario 1.

In the remainder of this report, we choose the top 100 symbols with highest 10 year returns (i.e., obtained by extending the table of figure \ref{488391} with a few more symbols below SPY).

2.3 Exploratory time series data analysis

In this section we now extract data for the 100 selected symbols up to the present day, using the yfinance library [5]. The extraction script is found in the path stock_prediction/etl/ticker_data_extractors.py of the repository.

Below, we look into a selection of statistical properties of the dataset in order to understand the best way of modeling the problem as well as to cleanup and select the data for modeling. The notebook with the full analysis is found at experiments/data_exploration/data_understanding_and_cleaning.ipynb.

In figure \ref{980651} we start by observing the distribution of first timestamps for each of the 100 selected symbols.

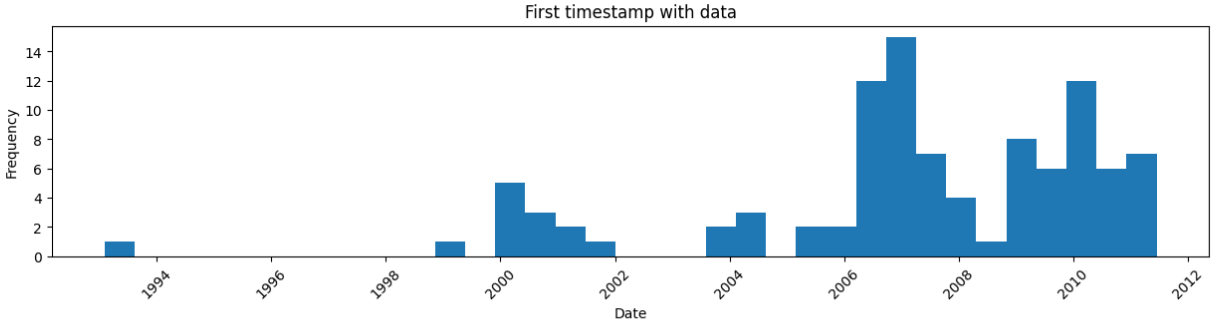


Figure 4: Distribution of first timestamp for the selected symbols

We can see that all symbols start more than 13 year ago, so we choose to clip the datasets to the period when all symbols have data (which already gives us more than 13 years of data, up to the present, for modeling).

Next, we look at distributions of daily returns. The daily return of the price time series is defined as

$$r_t = \frac{x_t - x_{t-1}}{x_{t-1}}$$

where x_t is the closing price at the end of the day at time t.

From here on, unless stated otherwise, we focus on the series of daily returns. The reason to do this is that returns compound over time resulting in an overall long term exponential growth for the prices so by working with returns we automatically remove a long term trend. Furthermore, the daily returns are not sensitive to stock splitting [6] (i.e., sometimes the units of an asset are redefined from one day to the other), which removes the extra complication of having to adjust for those events.

In figures \ref{578995} and \ref{257303} we show, respectively, the distributions of the mean daily return and the standard deviation over all symbols for our dataset.

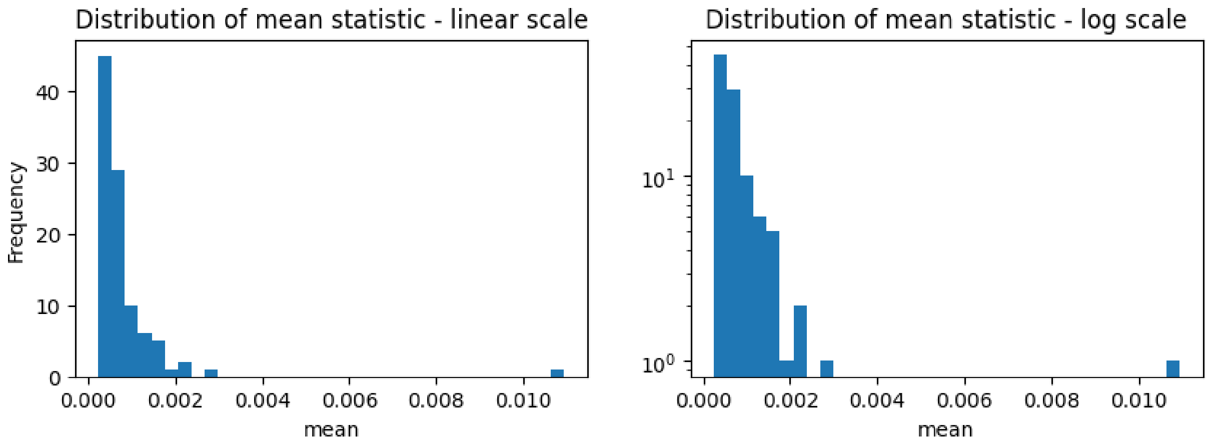


Figure 5: Distribution over symbols of the mean of the daily return per symbol in the selected period

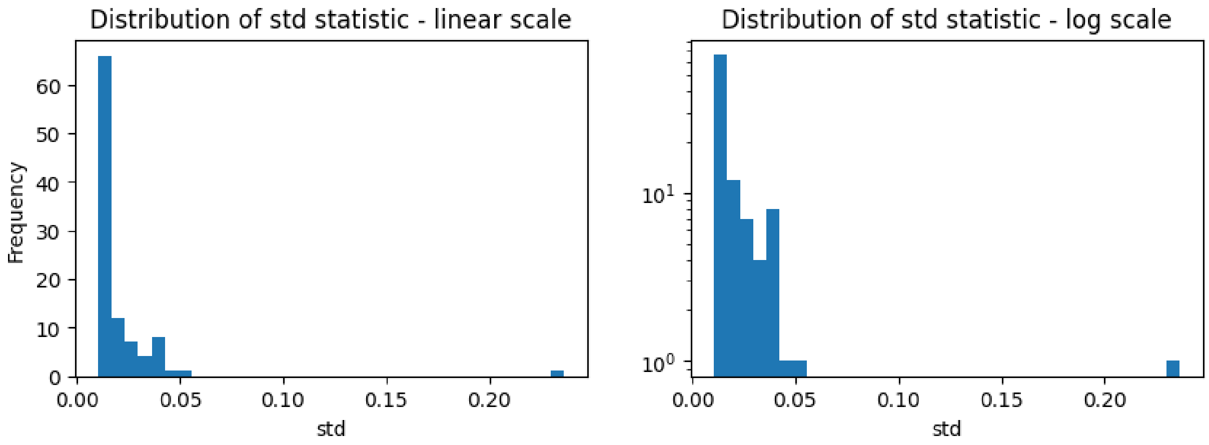


Figure 6: Distribution over symbols of the stdev of the daily return per symbol in the selected period

The main observation is that mean daily returns are peaked around a small positive value in comparison with the scale of standard deviations. This is consistent with the fact that the series of daily returns has a high volatility though in the long run there is a positive mean effect, thus justifying an overall positive growth in the long run. This already provides a hint that predicting very precisely the daily return series will not be possible, though the mean fluctuations of the series are more likely to be predictable. In any case, for an investor who is just interested in deciding a selling or buying price to set, having information on which direction the mean is likely to move and on how much is already useful to make a decision.

Next, we investigate how the 100 symbols relate to each other by observing a heatmap for the correlation matrix between any 2 symbols in figure \ref{293994} (we sort the symbols by correlation value with SPY).

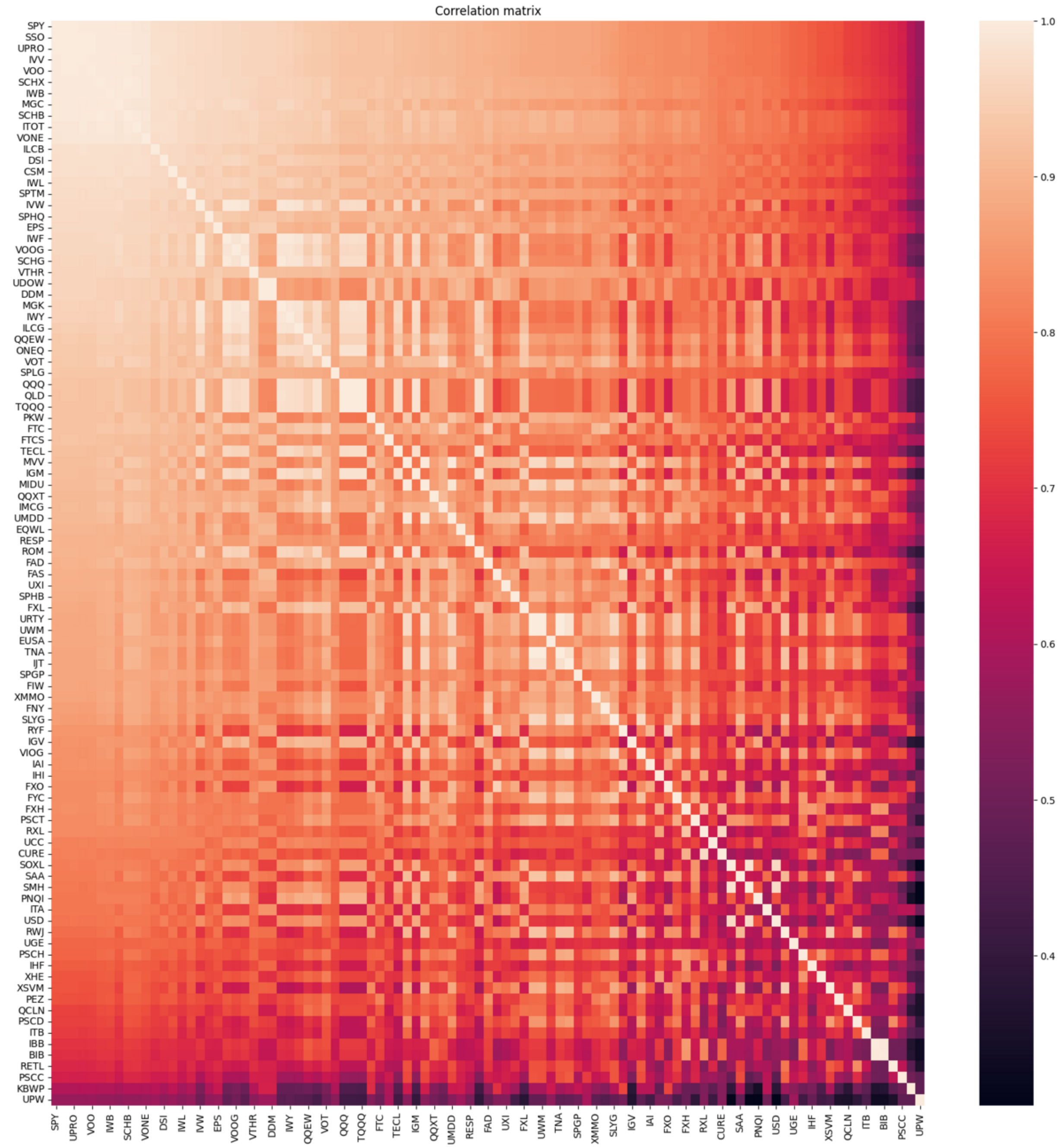


Figure 7: Correlation matrix of the time series of daily returns over the selected symbols

We observe that there is a non-trivial distribution of correlations, with some symbols being highly correlated and others not (see also figure 8). From an investors perspective, this information could be used to diversify away risk in their portfolio by choosing high performing symbols with low correlation. From a time-series modeling perspective, this motivates the possibility to model the time-series jointly instead of in a univariate way. This could exploit correlations in the data such as “if symbol 1 is growing, symbol 2 usually goes down”. Though this is an interesting possibility, in the experiments, for simplicity, we will focus on univariate modeling, i.e., we fit one time-series model per symbol.

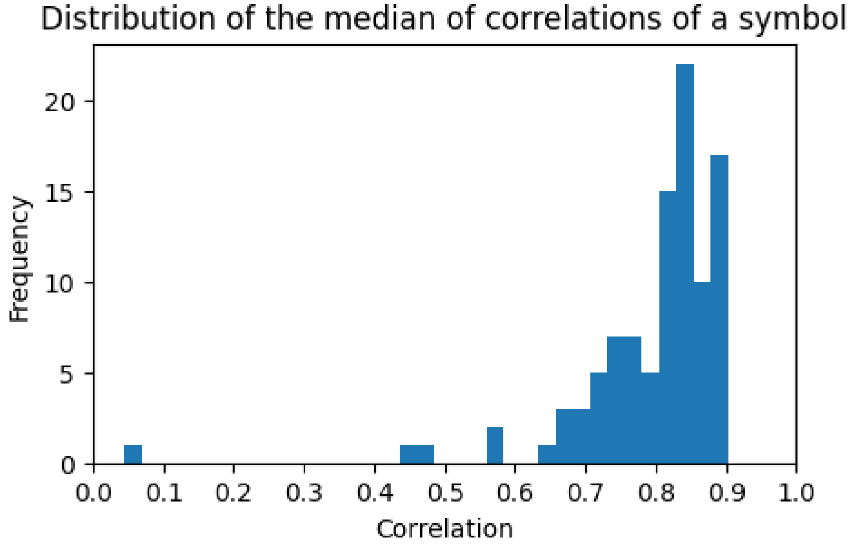


Figure 8: Distribution of the median of the correlations of a symbol with other symbols

In the following figures we now look more carefully at properties of each series. In figure \ref{273127} we show a the evolution of the cumulative daily returns, which is proportional to the real price.

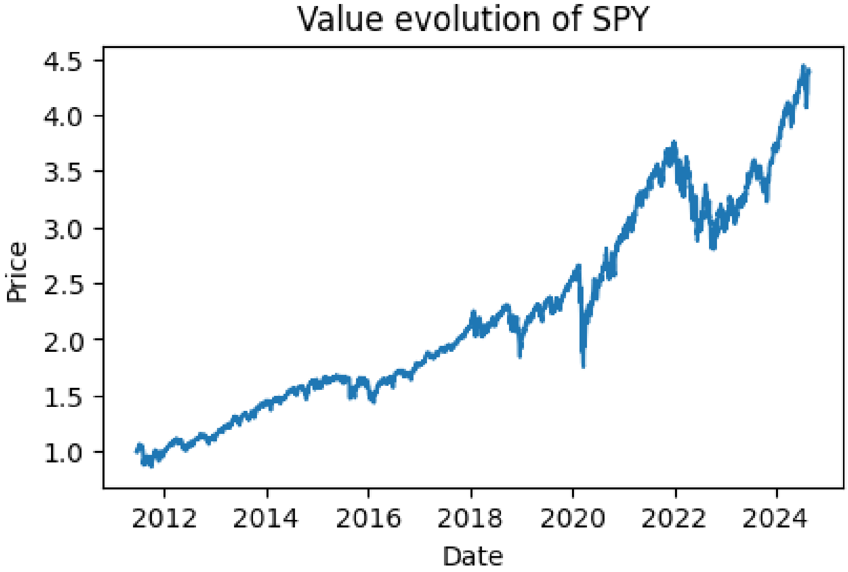


Figure 9: Relative price evolution for SPY

In figure \ref{258382} left panel we can compare it with the daily returns series. By comparing the two, we clearly see that by working with daily returns we remove a clear trend. Furthermore, we see that the daily returns fluctuate heavily around a small positive value, as already observed when analyzing the standard deviation.

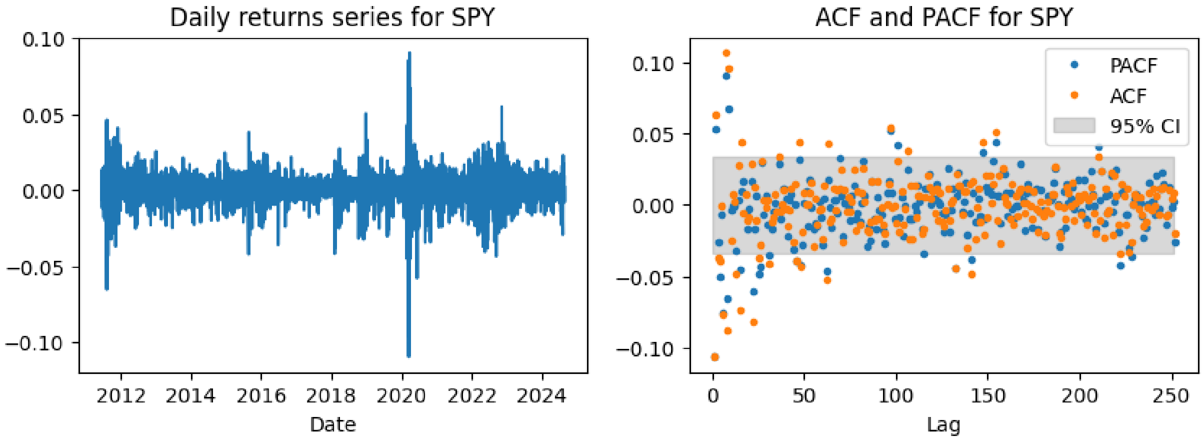


Figure 10: Example of a series of daily returns (left) and auto correlation analysis (right).

In figure \ref{258382} right panel, we observe and example of the auto-correlation function (ACF) and the partial auto-correlation functions (pACF) for SPY. These functions encode information on how patterns in the fluctuations of the series repeat themselves over time and are often used to choose hyperparameters for model building (see later sections). They are computed between the series and a lagged version of the series, so in the x axis, the lags range from 1 day to 252 (the approximate number of trading days in one year). The 95% confidence level band helps us understand if the value of the correlation function is significantly away from zero.

In figures \ref{695023} and \ref{769583} we plot the distribution of all lags with non-zero correlation (i.e., outside the 95% confidence level band) over all symbols for the pACF and ACF respectively.

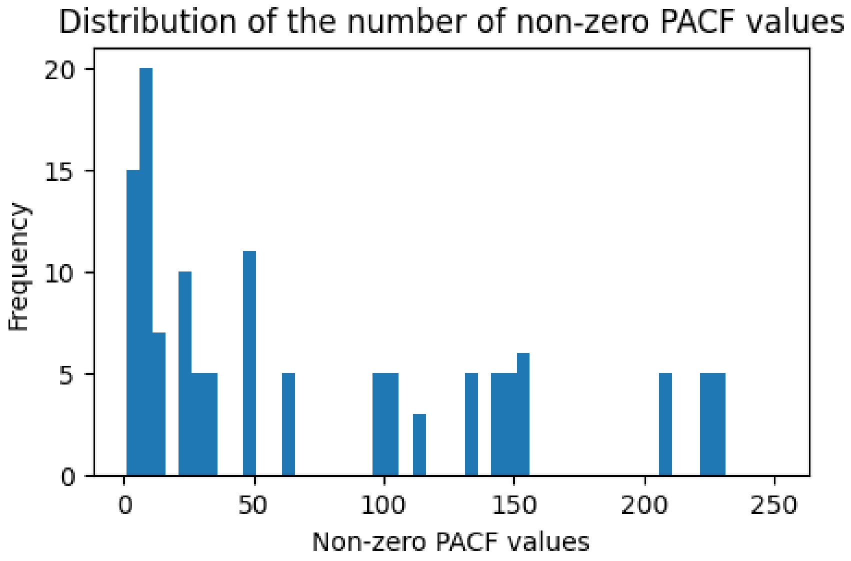


Figure 11: Distribution of the number of non-zero pACF values over the 100 symbols as a function of the lag (x-axis)

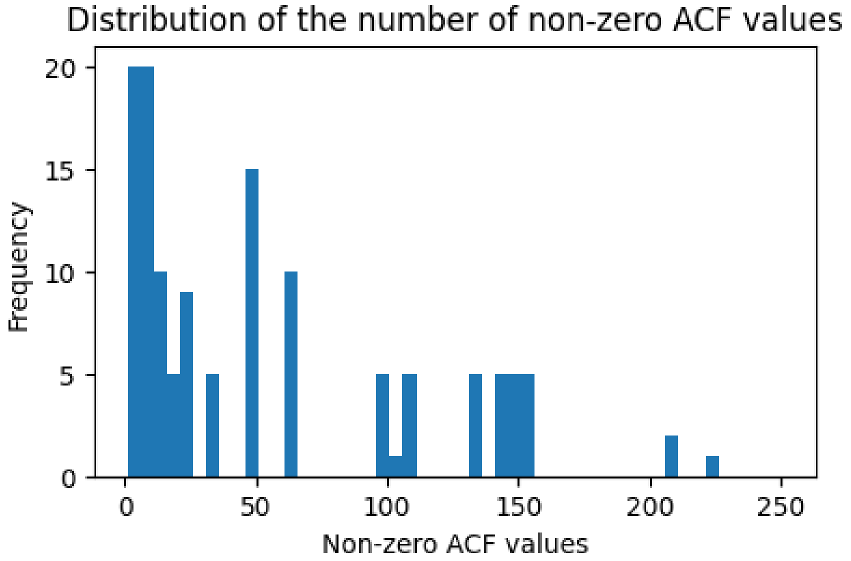


Figure 12: Distribution of the number of non-zero ACF values over the 100 symbols as a function of the lag (x-axis)

The ACF and pACF distributions indicate that the correlations are more pronounced on lags of up to 50 business days (i.e. about 2 months) and seem to have some smaller peaks at about 100, 150 and 220 business days (though with progressively lower frequency and correlation function values). On one hand this can be interpreted as a slight lack of stationarity (if we also observe the correlation function plots we see that the functions typically slowly decays to zero) and 3 seasonality frequencies.

Another measure of the oscillation periods of a signal is given by the Fourier spectrum, which decomposes a signal in sine and cosine signals of all frequencies (see figure \ref{392293}). We see that the amplitude of the spectrum has the highest peaks up to about 20 or 30 days, with a heavy tail beyond that.

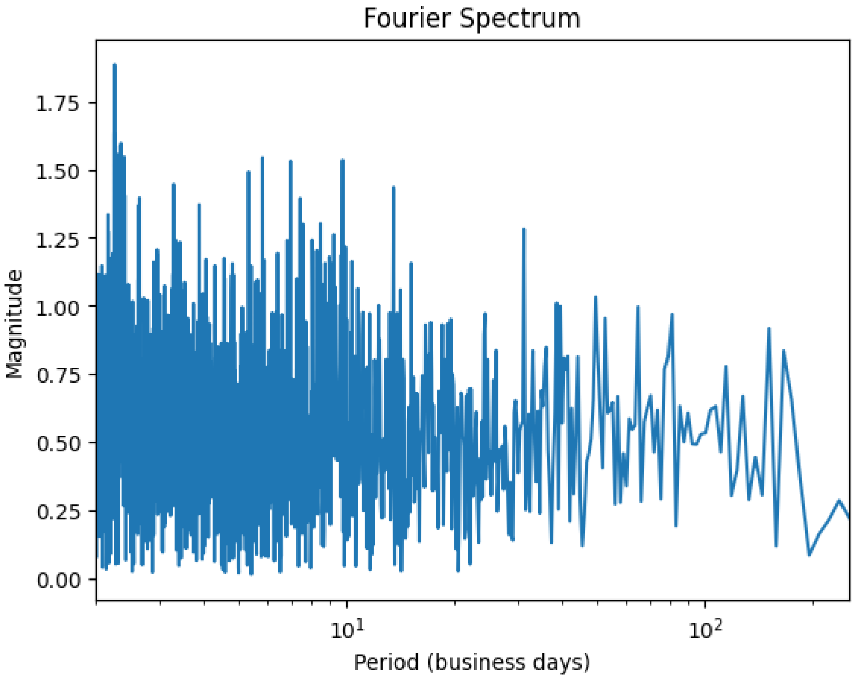


Figure 13: Fourier spectrum for SPY

To finish this section, we look into a very simple statistic that will both motivate the features we will design for model building as well as a simple baseline that we will use to compare our models against. In figure \ref{957318} we show the SPY price series against a prediction with a shift $\delta t = 1$, 5, 10, 20 and 40 days ahead, obtained given by compounding, at each time t, the average daily return computed with the 20 previous days, on top of the price at time t.

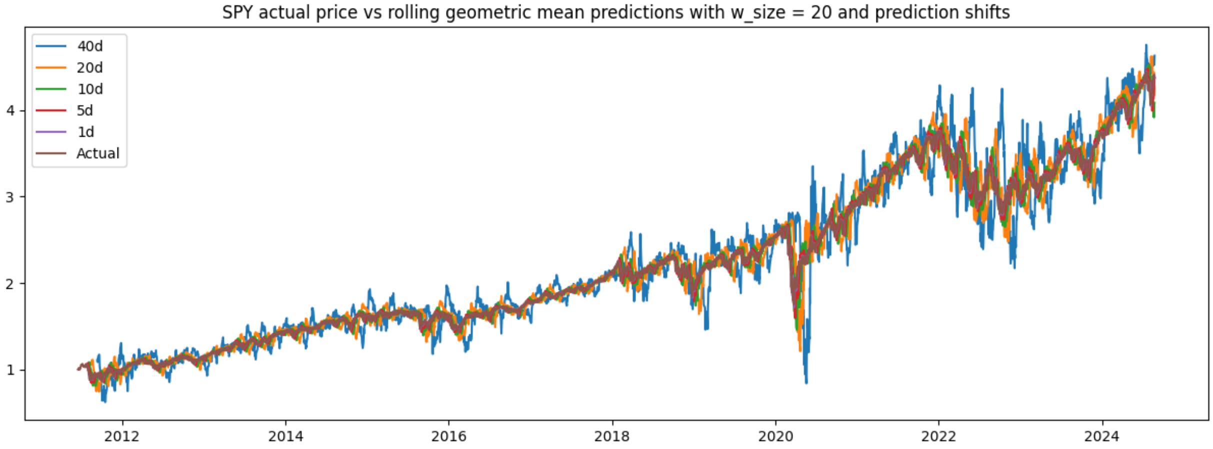


Figure 14: Rolling geometric averages vs the actual series of daily returns

We observe that all curves capture some of the overall trend, with the shorter time shifts reproducing better the actual signal. This motivates the usage of rolling averages to capture features of the time series in model building as well as using a rolling average as a simple model.

3 Methods

In this section we describe the methods used for model building and for evaluation. As already mentioned our target is to model the series of daily returns for each symbol.

3.1 Models

3.1.1 Baselines

The simplest benchmark models that can be defined in the context of time series prediction are:

- **Predict no change in price:** Since market prices fluctuate up and down the overall mean tends to drift slowly, so we expect this to be a good reference baseline (if we do worse than such a simple model then we have a clear red flag).
- **Predict based on a long term average return rate:** In our use case, the investor is mostly interested in the overall trend to ensure they invest at a favorable time. The return rate computed, e.g., on a few weeks or a few months is typically a good indication of the trend of the evolution of the prices so it is expected to provide a decent baseline. We already have indications of this from figure 14. In the results section we will tune the window for the rolling average to minimize the prediction errors of this simple baseline.

Our implementation of these baselines is found at stock_prediction/modeling/baselines.py.

3.1.2 ARIMA models

These are classic time series models that include autoregressive terms (linearly regressing on previous series values), moving average terms (linearly regressing on residual errors) and time series differencing [7]. They are denoted by ARIMA(p,d,q) where the parameters are non-negative integers such that:

- p is the order (number of time lags),
- d is the degree of differencing, i.e., the number of times the series is subtracted by past values to further de-trend the series (in our use case we already de-trended the price series once),
- q is the order of the moving average terms.

In our exploratory analysis we already saw that the dominant periods of the signal seem to be up to 20 days or 30 days with other longer subdominant periods. For simplicity, and because running ARIMA models with much larger orders becomes substantially slower, we will restrict p and q up to 20. In addition, we use $d = 1$ to further de-trend the daily return by the mean value.

Furthermore, we are interested in predicting the value of the time series n-steps after time t. This is already supported by the ARIMA model implementation we use from the statsmodels library [8].

Our implementation of ARIMA models adjusted to our use case is found at stock_prediction/modeling/arima.py.

3.1.3 LightGBM Models

The ARIMA models above have a very specific set of assumptions regarding the signal they model so the features of the series that are used are implicitly constrained the bias of the model. To allow for further flexibility we also consider using an out of the box model that can ingest any features we might engineer from the series. We use a gradient boosting decision trees model provided by the LighGBM library [9] and feed it various features that describe the history of the time series (statistics, series values etc.) with the hope of learning additional non-linear dependencies.

As already mentioned, our target is to predict several time steps after each time t. Thus, we expand our dataset for training the LightGBM model at each time step by repeating n-times each row and assigning the value of the target to be the daily return 1, . . . , n days ahead.

Regarding features, the full set we include for each series is as follows:

- The values of the series in the last N steps (e.g., 20).
- Rolling window statistics in the last 5, 20, 60, 180, 400 days (average, std, min, max) to cover several time scales.
- Time related features to have some dependency on seasonal patterns related to the day of the week, month of the year, or part of the year. We use, the fraction of the week, month and year as such features.

We use N=20, so we end up with 44 features + label.

Our implementation of UnivariateLightGBM models to fit on our dataset with 100 symbols is found at stock_prediction/modeling/lightgbm_model.py.

3.1.4 Other approaches

As already mentioned, we could follow many more modeling approaches as well as use different algorithms to train our models, namely:

- Modeling the series in a joint way could leverage correlations between symbols to extract further signal.
- Adding exogenous information could also be interesting since market shifts are often heavily controlled by e.g., public announcements of companies performances and other events such as natural disasters, political moves etc. just to name a few.
- Non-linear autoregressive neural network models [10]: RNN models such as LSTMs provide another approach to modeling time series non-linearities which does not require explicit feature engineering. These are typically more difficult to train though.

We will leave the investigation of these possibilities for future work, since the main focus is on the ML engineering side and not on providing an exhaustive search for the best modeling approach.

3.2 Evaluation Metrics

We propose two types of metrics to evaluate the models:

3.2.1 Loss for training

In regression problems, the typical metric is root mean squared error [11], which is the average of the squared residuals of the prediction at each step of the time series. Out of the box models in standard libraries for regression typically use this as the default loss for fitting.

3.2.2 Model evaluation

To evaluate the quality of the forecasts we propose to focus on Median Absolute Percentage Error (MedianAPE) for each series - defined as the Median of the absolute residuals. This will be used in validation to select the best model hyperparameters. We choose the median because it is less sensitive to outlier residuals, which are common to happen in finance due to isolated unusual market events, which deviate a lot from the mean behavior and are very unpredictable (so we do not aim to be able to predict them).

Furthermore, for each series, we compute the MedianAPE over the n-step forecast in a very specific way, namely:

1. At each time $t + i$ with $i = 1, \ldots, n$ we compute the relative error between the forecast price and the actual price defined as
$$\frac{\hat{x}_{t+i} - x_{t+i}}{x_{t+i}} \, .$$
2. Then we average those relative errors for each time t and take those as the errors for each time step.
3. Finally we take the median over all time steps to obtain our definition of MedianAPE. We can also take the mean instead of the median, which will be more sensitive to large outlier values.

This procedure gives us an aggregate metric describing the performance of the model, for each symbol, on predicting the next 20 business days. Since we have 100 symbols, we define a second aggregation of this distribution of values over symbols to get an overall metric to optimize in validation in order to in order to select hyperparameters.

The code used for evaluation is found at stock_prediction/evaluation/analysis.py.

4 Model Development

In this section we describe our approach and results to train the various models on historical data as well as our evaluation to select the best training method to deploy our daily retrained dashboard.

4.1 Data Splitting

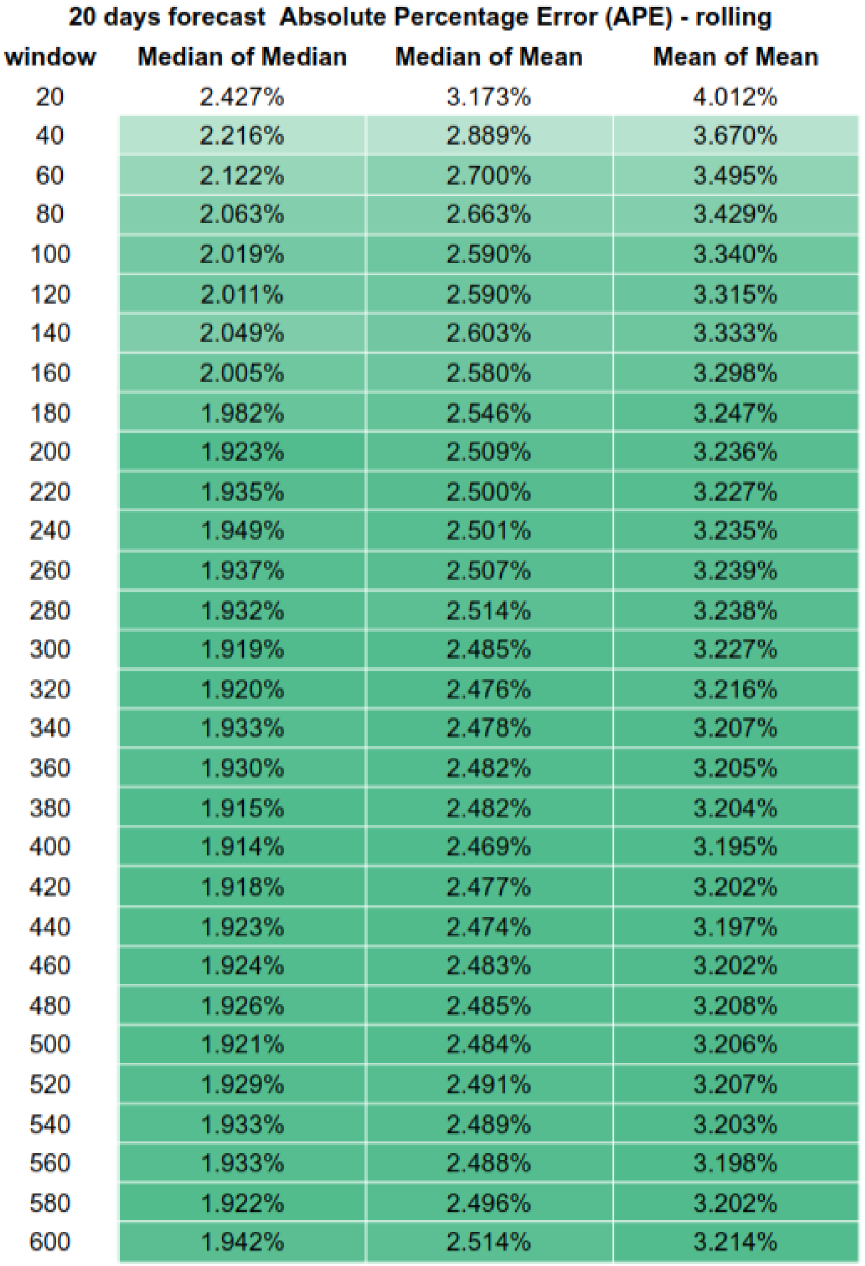
We use a standard temporal train + validation + test splitting strategy. The train set is used to train each model and the validation set to evaluate different models to select the best ones. After short listing one model per model type, we retrain each model on train+validation and re-evaluate them on the test set. This allows us to observe any temporal drift effects as well as to check if the validation set conclusions generalize, thus giving us a more unbiased evaluation (avoiding any residual risk of over fitting to the validation set).

The data splitting code is found here.

4.2 Hyperparameter Tuning

In this section we describe the results of hyperparameter tuning of the various proposed models.

In figure 15 we display the variation of our summary metrics for the rolling average baseline. Though our target evaluation metrics is the Median of the MedianAPE we also show the Median of the MeanAPE and the Mean of the MeanAPE over all the symbols. As expected the errors grow from left to right as the mean is more sensitive to large outlier values.



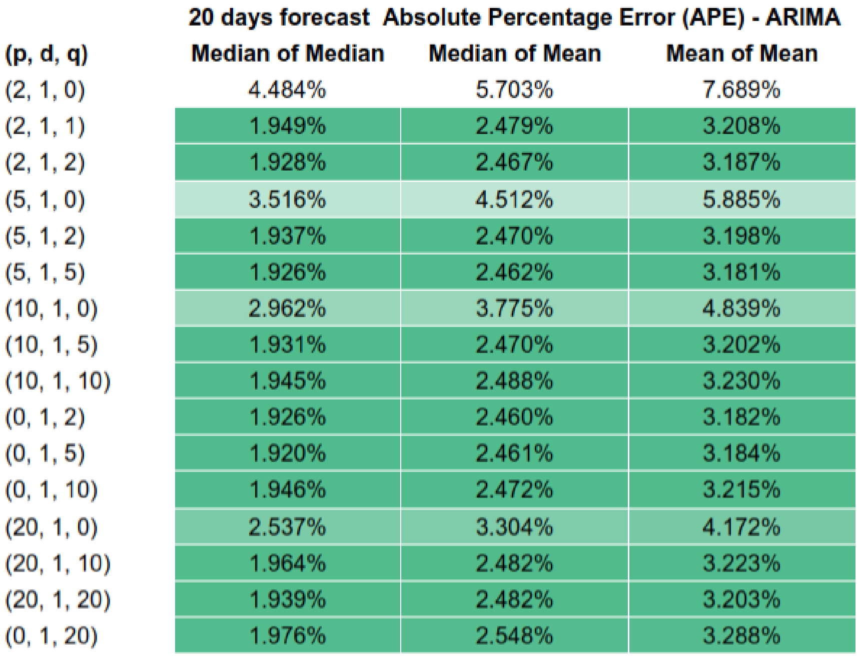
20 days forecast Absolute Percentage Error (APE) - rolling

window	Median of Median	Median of Mean	Mean of Mean
20	2.427%	3.173%	4.012%
40	2.216%	2.889%	3.670%
60	2.122%	2.700%	3.495%
80	2.063%	2.663%	3.429%
100	2.019%	2.590%	3.340%
120	2.011%	2.590%	3.315%
140	2.049%	2.603%	3.333%
160	2.005%	2.580%	3.298%
180	1.982%	2.546%	3.247%
200	1.923%	2.509%	3.236%
220	1.935%	2.500%	3.227%
240	1.949%	2.501%	3.235%
260	1.937%	2.507%	3.239%
280	1.932%	2.514%	3.238%
300	1.919%	2.485%	3.227%
320	1.920%	2.476%	3.216%
340	1.933%	2.478%	3.207%
360	1.930%	2.482%	3.205%
380	1.915%	2.482%	3.204%
400	1.914%	2.469%	3.195%
420	1.918%	2.477%	3.202%
440	1.923%	2.474%	3.197%
460	1.924%	2.483%	3.202%
480	1.926%	2.485%	3.208%
500	1.921%	2.484%	3.206%
520	1.929%	2.491%	3.207%
540	1.933%	2.489%	3.203%
560	1.933%	2.488%	3.198%
580	1.922%	2.496%	3.202%
600	1.942%	2.514%	3.214%

Figure 15: Variation of the various performance metrics with the window size in days, for the rolling average baseline

The main conclusion is that the best rolling window is actually quite large, 400 days, corresponding to about 1.5 years.

In figure \ref{176774} we display the results for the ARIMA models for a sweep over 16 configurations of hyperparameters.

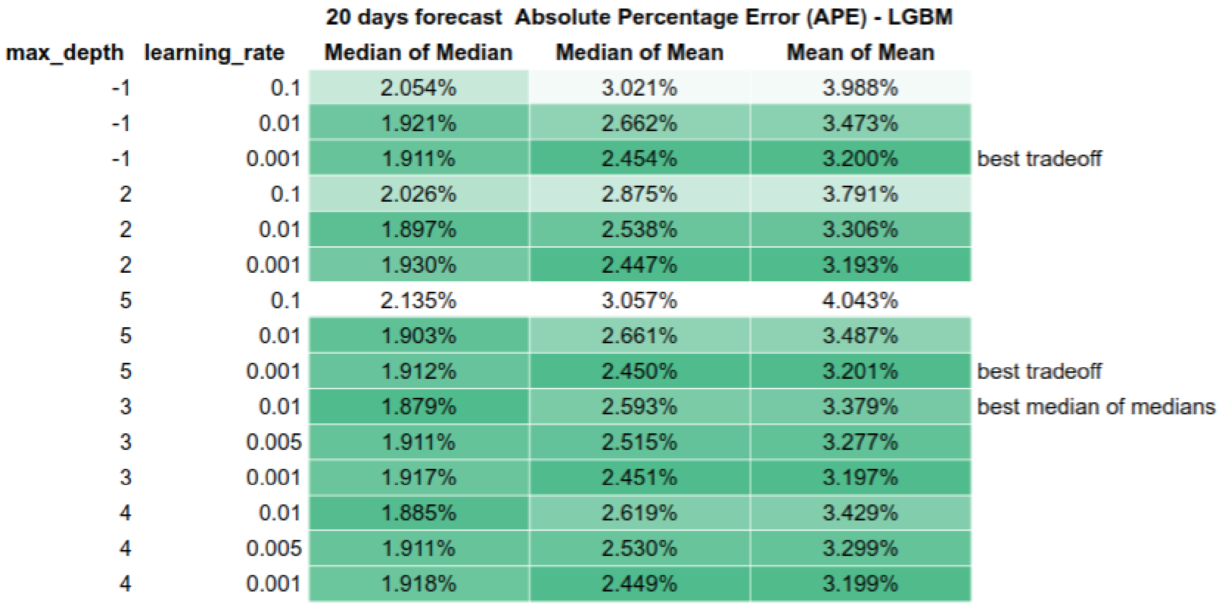


	20 days forecast	Absolute Percentage Error (APE) - ARIMA	
(p, d, q)	Median of Median	Median of Mean	Mean of Mean
(2, 1, 0)	4.484%	5.703%	7.689%
(2, 1, 1)	1.949%	2.479%	3.208%
(2, 1, 2)	1.928%	2.467%	3.187%
(5, 1, 0)	3.516%	4.512%	5.885%
(5, 1, 2)	1.937%	2.470%	3.198%
(5, 1, 5)	1.926%	2.462%	3.181%
(10, 1, 0)	2.962%	3.775%	4.839%
(10, 1, 5)	1.931%	2.470%	3.202%
(10, 1, 10)	1.945%	2.488%	3.230%
(0, 1, 2)	1.926%	2.460%	3.182%
(0, 1, 5)	1.920%	2.461%	3.184%
(0, 1, 10)	1.946%	2.472%	3.215%
(20, 1, 0)	2.537%	3.304%	4.172%
(20, 1, 10)	1.964%	2.482%	3.223%
(20, 1, 20)	1.939%	2.482%	3.203%
(0, 1, 20)	1.976%	2.548%	3.288%

Figure 16: Variation of the various performance metrics for the various ARIMA models as function of the tried hyperparameters

The best model is for (p,q,d) = (0, 1, 5) i.e. a moving average model with an order of 5, though there are other models with moving average terms with similar performance. The main conclusion seems to be that not including moving average terms should not be done, since the performance degrades considerably in those cases.

In figure \ref{728131} we display the results for LightGBM with various hyperparameter configurations. We chose to vary the maximum depth of each base tree learner as a way to control the level of regularization (-1 for unbounded) and the learning rate to control how much each boosting step is adjusting the model (the number of boosting rounds is fixed to 100).

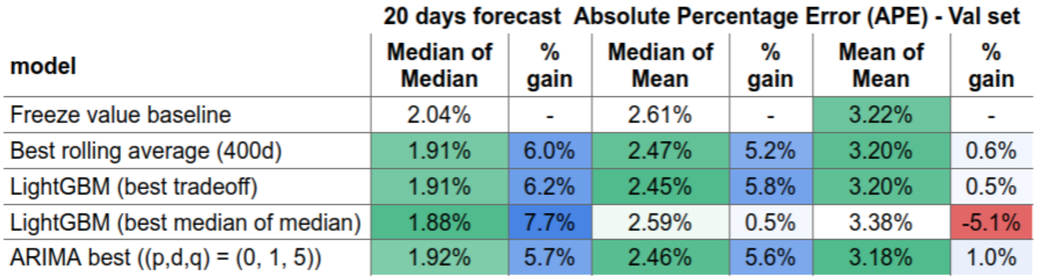


		20 days forecast	Absolute Percentage Error (APE) - LGBM		
max_depth	learning_rate	Median of Median	Median of Mean	Mean of Mean	
-1	0.1	2.054%	3.021%	3.988%	
-1	0.01	1.921%	2.662%	3.473%	
-1	0.001	1.911%	2.454%	3.200%	best tradeoff
2	0.1	2.026%	2.875%	3.791%	
2	0.01	1.897%	2.538%	3.306%	
2	0.001	1.930%	2.447%	3.193%	
5	0.1	2.135%	3.057%	4.043%	
5	0.01	1.903%	2.661%	3.487%	
5	0.001	1.912%	2.450%	3.201%	best tradeoff
3	0.01	1.879%	2.593%	3.379%	best median of medians
3	0.005	1.911%	2.515%	3.277%	
3	0.001	1.917%	2.451%	3.197%	
4	0.01	1.885%	2.619%	3.429%	
4	0.005	1.911%	2.530%	3.299%	
4	0.001	1.918%	2.449%	3.199%	

Figure 17: Variation of the various performance metrics for the various LightGBM models as function of the tried hyperparameter variations

Overall the results improve considerably over the other models with the best model having a Median of MedianAPE of 1.88% for a maximum depth of 3 and a learning rate of 0.01, i.e., using well regularized trees. We also indicate in the table other options trading off between a low median error and mean errors (i.e., reducing the other metrics that are sensitive to outliers). Note, however, that the differences between the best model and the alternatives are not large.

In figure \ref{343251} we show a table summarizing the performance of the best models for each model type where we also compare with the simplest baseline where we freeze the daily return rate.



	20 days forecast	Absolute Percentage Error (APE) - Val set				
model	Median of Median	% gain	Median of Mean	% gain	Mean of Mean	% gain
Freeze value baseline	2.04%	-	2.61%	-	3.22%	-
Best rolling average (400d)	1.91%	6.0%	2.47%	5.2%	3.20%	0.6%
LightGBM (best tradeoff)	1.91%	6.2%	2.45%	5.8%	3.20%	0.5%
LightGBM (best median of median)	1.88%	7.7%	2.59%	0.5%	3.38%	-5.1%
ARIMA best ((p,d,q) = (0, 1, 5))	1.92%	5.7%	2.46%	5.6%	3.18%	1.0%

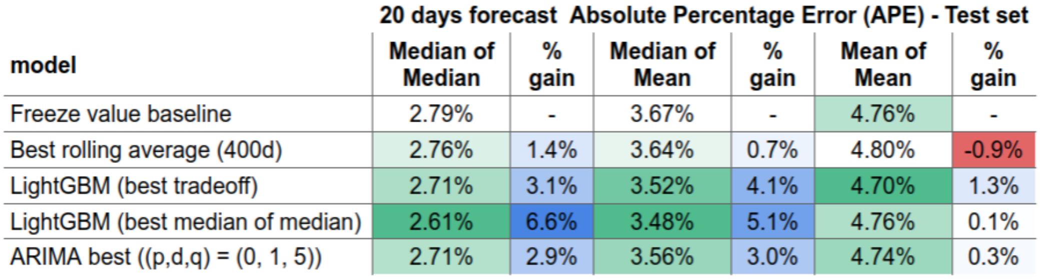
Figure 18: Comparison of the best models for each model type (Validation set)

All models achieve an improvement larger than 5% for our target evaluation metric while for the other metrics the gains are either smaller or there are no gains. In particular we note that there are some outlier symbols with larger dominating errors since the gains for Mean of MeanAPE over the simple baseline are close to null or negative. Also note that the best model for LightGBM actually performs very negatively for those alternative metrics, which could indicate some over fitting in validation.

The analysis notebooks for this hyperparameter tuning experiments are found at experiments/hpt.

4.3 Test set Evaluation

Finally we evaluate the short listed models on the Test set as displayed in figure 19.



	20 days forecast Absolute Percentage Error (APE) - Test set					
model	Median of Median	% gain	Median of Mean	% gain	Mean of Mean	% gain
Freeze value baseline	2.79%	-	3.67%	-	4.76%	-
Best rolling average (400d)	2.76%	1.4%	3.64%	0.7%	4.80%	-0.9%
LightGBM (best tradeoff)	2.71%	3.1%	3.52%	4.1%	4.70%	1.3%
LightGBM (best median of median)	2.61%	6.6%	3.48%	5.1%	4.76%	0.1%
ARIMA best ((p,d,q) = (0, 1, 5))	2.71%	2.9%	3.56%	3.0%	4.74%	0.3%

Figure 19: Comparison of the best models for each model type (Test set)

Looking at our target metric, we first observe that the relative gains over the baseline are smaller, which indicates some over fitting to validation. Nevertheless, we now see that the best LightGBM model in validation remains the best in test and much more significantly over the other models than in validation (6.6% gain over the baseline versus 2.9% for the best ARIMA model). It is also reassuring to see that the performance on the other two metrics is good, actually better than in validation (this might also be due to temporal drift in the data).

Therefore, we choose to deploy a LightGBM model with a maximum depth of 3 and a learning rate of 0.01, which we will retrain daily on all past data.

The notebook with this final evaluation is found at experiments/best_models.ipynb.

5 Dashboard App Development

In this section we describe our implementation of the daily retraining pipeline to update model forecasts and the dashboard in AWS.

5.1 Architecture

In figure 20 we provide a high level diagram of the architecture we developed for the deployment

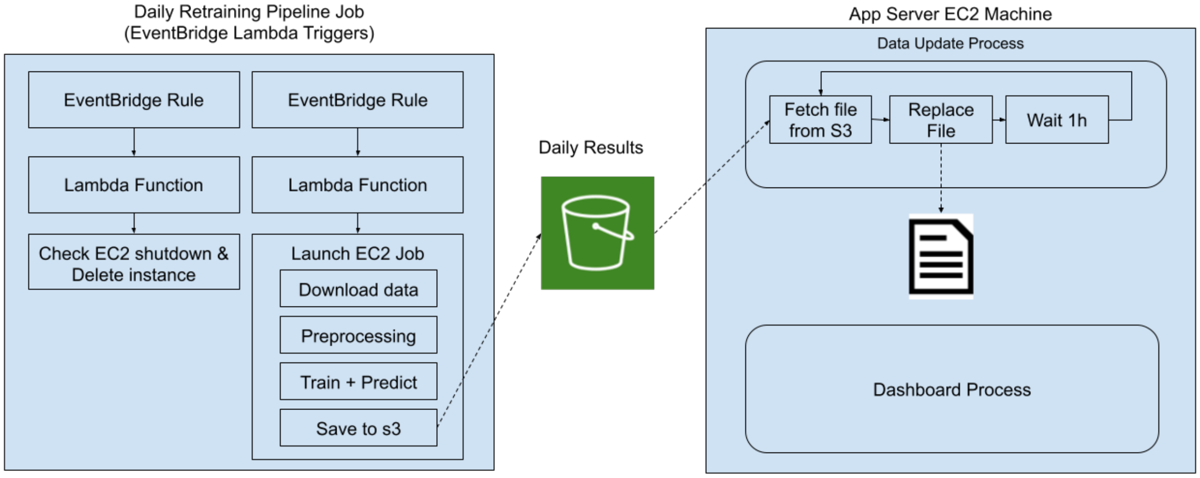
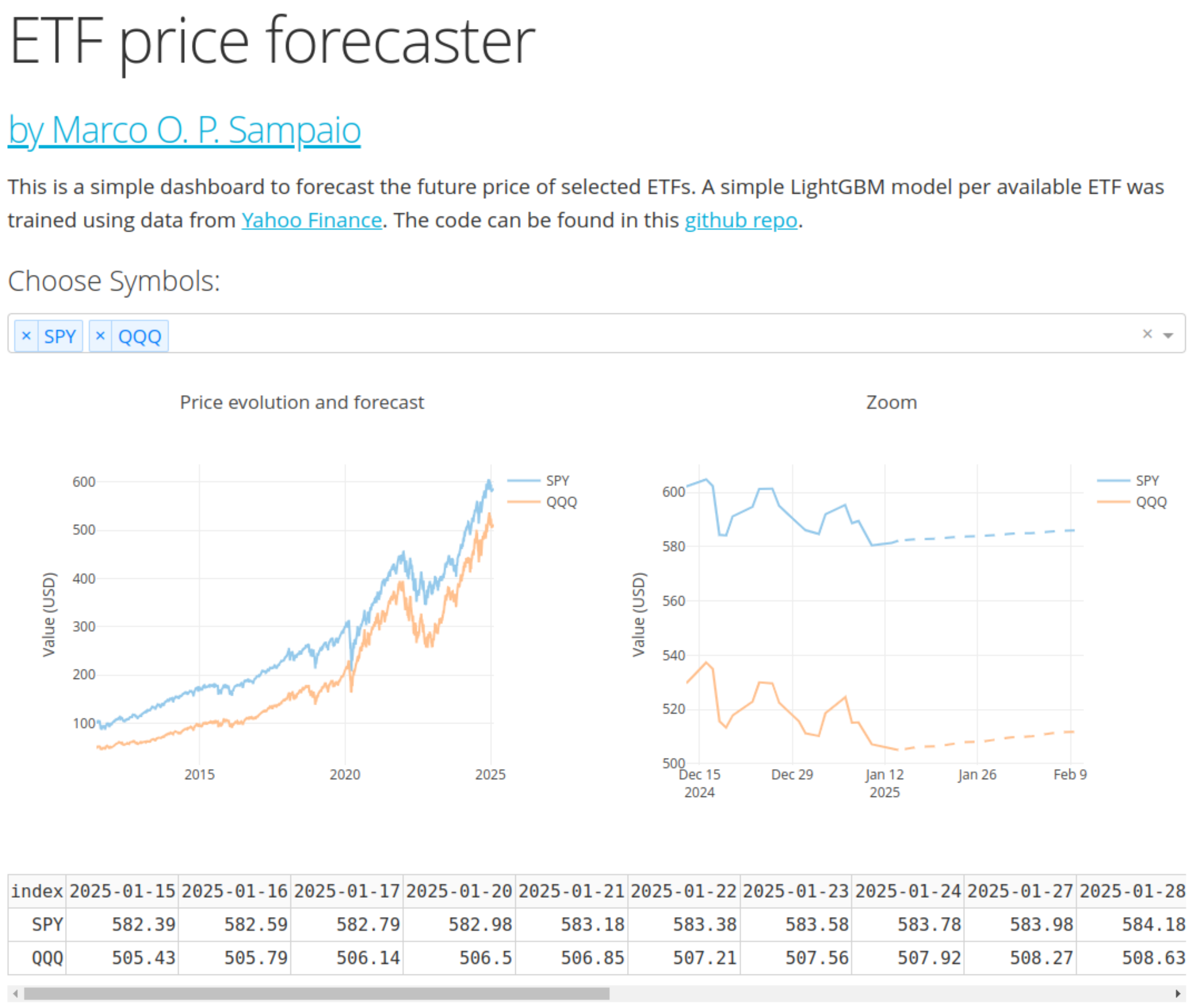


Figure 20: Overall deployment architecture

In our use case forecasts do not need to be updated in real time, since we are only predicting daily returns, and the model re-training of all symbols is relatively quick (less than one hour). So we choose to join the model retraining and inference in one single batch job avoiding having to serve the model in real time. Thus:

- The model is retrained daily to produce up to date forecasts without any manual intervention. We manage this by launching an EC2 instance daily, through an EventBridge rule triggering a lambda function, that executes the whole data extraction + model training + inference pipeline and writes the results to a file in s3. To reduce costs we have an additional EventBridge rule triggering a lambda function to check for stopped EC2 instances every hour and delete them.
- The dashboard app is deployed in an auto scaling group and available at a public URL, allowing for a user to select the ETFs they want to visualize. The dashboard fetches the data directly from S3. It allows for the user to clearly see the forecasts and the historical data, zoom specific data periods and consult the specific values of prices at specific dates in a table. An example of the final deployed dashboard can be seen in figure \ref{445301}. The live dashboard will remain active during the assessment of this project at http://dashboard-lb-1124043407.us-east-1.elb.amazonaws.com:8050 (if having problems accessing it try this earlier version of the deployment at http://dashboard-lb-461967644.us-east-1.elb.amazonaws.com:8050/).



ETF price forecaster

by Marco O. P. Sampaio

This is a simple dashboard to forecast the future price of selected ETFs. A simple LightGBM model per available ETF was trained using data from Yahoo Finance. The code can be found in this github repo.

Choose Symbols:

× SPY × QQQ

index	2025-01-15	2025-01-16	2025-01-17	2025-01-20	2025-01-21	2025-01-22	2025-01-23	2025-01-24	2025-01-27	2025-01-28
SPY	582.39	582.59	582.79	582.98	583.18	583.38	583.58	583.78	583.98	584.18
QQQ	505.43	505.79	506.14	506.5	506.85	507.21	507.56	507.92	508.27	508.63

Figure 21: Final deployed dashboard

5.2 Implementation Details

The implementation of the architecture above involves several ingredients as described below (see also the README file in the deployment directory of the repository)

5.2.1 AMI creation script

In our architecture, for better control, we choose to launch custom EC2 instances containing the project repository installed with all dependencies. To avoid wasting compute time pulling the repository and installing all necessary dependencies every time (which both delays deployment and can be wasteful in model retraining) we create a custom AMI. This AMI is used for all instances we launch for different purposes and contains all scripts that need to be run for model retraining as well as the script to launch the dashboard app.

The script that fully automates the AMI creation is found at stock_prediction/deployment/ami_creator.py. The basic logic of the script includes (all done programatically with boto3):

- Run a ”launch-instance” command that launches an instance based on a standard ubuntu AMI and runs a custom bash script that installs all dependencies and the repository.
- Run the ”make-ami” command (after checking the instance finished the installation by connecting to the instance in the AWS console).
- Check that the AMI is available in the AWS console.
- Manually delete the instance in the AWS console to avoid wasting resources.

This creates a info.yaml file that contains information on the AMI created that is used by the other deployment scripts.

Note that for AMI creation we use a t2.micro instance which is one of the cheapest ones and still has enough compute resources.

5.2.2 Automated model retraining script

For the model retraining we developed a script to create the resources which includes setting up triggers to launch an instance for retraining, managing the associated IAM policies and a trigger to delete shutdown instances that have completed re-training to avoid extra costs. This is achieved by using EventBridge rules and AWS Lambda functions.

The script that manages the deployment of the retraining pipeline is found at stock_prediction/deployment/daily_retraining_pipeline_deployer.py. When this deployment script runs it does the following:

- Creates a lambda execution role to allow for access to AWS Lambda, EC2, CloudWatch and EventBridge services.
- Then creates lambda functions to launch and delete EC2 instances used for model retraining. The lambda functions involve python scripts that are responsible either for launching an EC2 instance that runs the retraining or checking if there are instances eligible for deletion (using the EC2 boto3 client).
- Creates an EventBridge rule that runs daily the lambda that runs the model retraining daily
- Creates an EventBridge rule that runs hourly the lambda that checks for shutdown instances that are eligible for deletion.

The script also contains a function to automatically cleanup the deployment and all its resources.

Note that the model retraining script that runs when the EC2 instance is launched by the lambda function is found at stock_prediction/modeling/train.py. After the latter runs, the predictions file is copied over to S3 to be accessible by the dashboard.

For retraining we use t2.small instances since it requires a bit more memory.

5.2.3 Dashboard deployment script

The final ingredients are the script to run the dashboard and the corresponding deployment script.

We developed a dashboard app python script based on the dash library [12], which leverages plotly for the plotting. The script is found at stock_prediction/dashboard/dashboard.py and it directly reads the required data from S3

As for the script to deploy the dashboard for serving it does the following (see the script at stock_prediction/deployment/dashboard_deployer.py):

- Creates a security group for the dashboard, with the right permission to allow for HTTP requests so that the dashboard is accessible via the browser from anywhere.
- Creates an IAM profile for the dashboard instance that allows access to S3 (to fetch the latest predictions).

- Creates a launch template. This is necessary because we deploy through an autoscaling group assuming a use case where the dashboard could become popular and get spikes in requests, thus requiring spinning up more machines.
- Makes a load balancer to distribute the request through the launched instances that serve the dashboard.
- Creates the autoscaling group with a minimum of 1 instance and a maximum of 3. We use t2.micro instances.
- Creates scale up and down policies and corresponding cloud watch alarms to trigger the policies to either add or delete instances.

The script also prints out the URL where the dashboard is available after deployment and if ran with the option ”--cleanup” it cleans up the deployment by killing all machines and deleting all resources.

6 Conclusion

In this project we developed a model to forecast a selection of ETF prices in a 20 business days window using several ML model types. We focused on the use case of an investor that is interested in a long term investment with high returns and low risk.

In data exploration we found that the level of noise of the time series of daily returns that we aimed to model is quite noisy, but that predicting the general local trend should be possible. We observed symbols with various degrees of correlation which motivates further exploiting it in model building in the future. We also observed self correlations, which motivated using lags of about 20 days in modeling with ARIMA models, but also observed residual correlations on much longer periods up to (or even above) 1 year.

We then explored various types of models, from simple baselines, such as predicting the last value or compounding a rolling average estimate of the return, to ARIMA and LightGBM based models. After running hyperparameter tuning, we selected one model from each type and when evaluating the short listed models in the test set we found a LightGBM model with the best performance gains over the simplest baseline (about 6%).

We then developed code to deploy a dashboard in an autoscaling group to serve requests and a daily retraining pipeline so that a user consulting the dashboard can see up to date forecasts for the next 20 days, together with the historical time series, for the symbols of their choosing.

Note: The deployed dashboard will remain available during the assessment of this report at http://dashboard-lb-1124043407.us-east-1.elb.amazonaws.com:8050 (if having problems accessing it try this earlier version of the deployment at http://dashboard-lb-461967644.us-east-1.elb.amazonaws.com:8050/)

References

[1] “Exchange-Traded Fund (ETF): How to Invest and What It Is”. https://www.investopedia.com/terms/e/etf.asp.

[2] “Exchange-traded fund - Wikipedia”. https://en.wikipedia.org/wiki/Exchange-traded_fund.

[3] “What are exchange traded funds (ETFs)? — Vanguard”. https://investor.vanguard.com/investor-resources-education/etfs/what-is-an-etf.

[4] “US Funds dataset from Yahoo Finance”. https://www.kaggle.com/datasets/stefanoleone992/mutual-funds-and-etfs.

[5] “yfinance”. https://pypi.org/project/yfinance/.

[6] “What a Stock Split Is, Why Companies Do It, and How It Works, With an Example”. https://www.investopedia.com/terms/s/stocksplit.asp.

[7] “Autoregressive integrated moving average - Wikipedia”. https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average.

[8] “statsmodels.tsa.arima.model.ARIMA - statsmodels 0.14.4”. https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.model.ARIMA.html.

[9] “lightgbm.LGBMRegressor — LightGBM 4.5.0.99 documentation”. https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMRegressor.html.

[10] “Recurrent neural network - Wikipedia”. https://en.wikipedia.org/wiki/Recurrent_neural_network.

[11] “Root mean square deviation - Wikipedia”. https://en.wikipedia.org/wiki/Root_mean_square_deviation.

[12] “Dash Documentation & User Guide — Plotly”. https://dash.plotly.com/.