DTSA 5509- MACHINE LEARNING INTRODUCTION – FINAL PROJECT

Applying Regression and Ensemble Techniques to Predict Stock Prices

2025-03-12

Introduction

Stock prices move every second based on a large number of reason, many of which are unknown or not quantifiable. Wouldn't it be useful to be able to predict the direction to invest wisely? What if we could predict not just the direction, but the price itself a number periods forward?
That's exactly what this paper is about. I will attempt to predict the closing price of a stock listed in the S&P500 a number of trading days forward. The unit of periods is trading days of the S&P500.

The main goal is to predict the closing price of (any) S&P500 constituent securities based on the available daily time series of open, high, low, volume, and close prices of other circa 500 securities, indices and commodities, and open, high, low, and volume of the security itself. Each security brings 4 or 5 predictors: open, high, low, close prices, and volume, with volume some times not (completely) available. This makes the total number of possible predictors a number between 2,000 and 2,500.

One (naive) strategy to build and select the best predictive model would be to test prediction accuracy for models all regression models with 1,2,3,….1999 predictors. We could build 1 model with 1999 predictors, 1999 models with 1 predictor, 1999 x 1998=3,994,002 models with 2 predictors, and 2.76 10^26 models with combinations of just 10 predictors. Producing this many models would take 8.75 10^18 years, if it took just 1 second (and it does take longer than that!) to build 1 model with 10 predictors.
Computing this number of combinations is beyond the capacity of my humble AMD processor, and the time to results beyond my life time. And if that wasn't enough bad news, the 500 or so securities I am considering for the purpose of this paper, make only about 10% of the total of 6,000 or so listed in the US markets (NYSE, NASDAQ, and AMEX combined). The scalability of such strategy is limited (if any exists at all).
While this is not strictly speaking a curse of dimensionality, it is a curse nonetheless (of astronomically large number of combinations of predictors). We have good data, and lots of dimensions (as many as predictors there are). But we can't test all possible models to select the best. We need a more pragmatic

strategy, yet an effective one able to deliver a satisfactory prediction, within reasonable (conventional, non-quantum) computational time.

The approach to solving the problem is incremental in the level of modeling complexity. I begin with a simple linear regression models of 1 predictor, and then increase complexity gradually through multi-predictor linear models, decision trees (DT), random forest (RF), Adaptive Boosting (Ada Boost), Gradient Boosting (GB), and Extreme Gradient Boost (XGB).

Once I arrive at the final models, I stack them to produce an average output, and with that I predict the close on 500 securities. This is really the final test with out-of-sample data, which enables the calculation of the error estimate of the model.

The Data Set:

The data set is composed of circa 500 .csv files containing time series with daily trading open, high, low, close prices and volume of about 500 securities including stocks (S&500 constituents), bonds (13-week, 5-year, 10-year, 30-year), commodities (gold, and crude oil), US indices (SP500, Nasdaq100, DJIA, DXY), European indices (DAX, CAC, FTSE), Asian indices (STI, HSI, Nikkei225, FCHI) with trading information dating back to the time the security was first listed. The time series for the S&P500 index goes back to 1927-12-30, more than 24,000 trading days. This security alone has 120,000 data points.

The source of the data set is yahoo finance (1), downloaded with a python the package yfinance (2) .

S&P500 headers before cleaning:

Date	Open	High	Low	Close	Volume
2025-02-04	5998.14013671875	6042.47998046875	5990.8701171875	6037.8798828125	4410160000
2025-02-05	6020.4501953125	6062.85986328125	6007.06005859375	6061.47998046875	4756250000
2025-02-06	6072.22021484375	6084.02978515625	6046.830078125	6083.56982421875	4847120000
2025-02-07	6083.1298828125	6101.27978515625	6019.9599609375	6025.990234375	4766900000
2025-02-10	6046.39990234375	6073.3798828125	6044.83984375	6066.43994140625	4458760000
2025-02-11	6049.31982421875	6076.27978515625	6042.33984375	6068.5	4324880000
2025-02-12	6025.080078125	6063	6003	6051.97021484375	4627960000
2025-02-13	6060.58984375	6116.91015625	6050.9501953125	6115.06982421875	4763800000
2025-02-14	6115.52001953125	6127.47021484375	6107.6201171875	6114.6298828125	4335190000
2025-02-18	6121.60009765625	6129.6298828125	6099.509765625	6129.580078125	4684980000
2025-02-19	6117.759765625	6147.43017578125	6111.14990234375	6144.14990234375	4562330000
2025-02-20	6134.5	6134.5	6084.58984375	6117.52001953125	4813690000

A simple ticker list with all securities is not available through yFinance, but it is available through a wikipedia page (3). The list requires a few ticker syntax adjustments --done in the pre-processing stage--, but other than that it is mostly useful.

Nomenclature is not difficult but it may get confusing. Let's take Apple Inc for example:

* security: it is the listed stock for Apple Inc. It could be gold (GLD) or in the context of this paper also a commodity –such as crude oil--, a bond index –such as ^TYX--, or a market index –such as ^HSI--.
* predictors: they are the open, high, low, close prices, and volume of the security.
* ticker: the code name of the security in the stock market. Apple Inc's ticker is AAPL.

The code:

The code is split in two files:

1) updateYFDataRev0.py: Downloads the data set to 'data' folder --which should be created before running the code--, and performs the first step of preprocessing.

2) week7EnsembleRFRev0.py: Once that the data has been downloaded, this .py assembles a portion of the data set on the fly to create the predictive models. The first section of the code sets all the functions that will later be called as needed. The linear regression piece of the code is functional but it has been commented out due to the fact that the linear models take quite a bit of time to run and are less powerful than the tree and ensemble models developed later.

Data Cleaning and Pre-Processing:

The individual data sets are generally complete, reliable, and readily available. Cleaning was mostly about identifying and fixing missing data. Open price and Volume were the main targets of the cleaning effort. In those securities that showed 100 or more missing data points in the series, the entire field was eliminated, or truncated if there was data continuity after certain date. For fewer than 100 missing data points the previous trading session was used. The process was automated in python.

Data Pre-Processing:

Different types of securities trade in different trading timelines. Stocks and bonds in the US differ in the number of trading days. The same thing happens with commodities, and with trading sessions in other geographies. Two securities in the same geography may differ in the number of trading days in their respective time series just because they listed at different times in history. Given that the prediction of the price on a future date requires consistency on dates across predictors, a baseline sequence of trading date became necessary. I used the S&P500 index trading dates as the benchmark. All other securities in the dataset referred to the dates of the S&P500 time series. If a date was not available in a security time series (due to a holiday in Europe for example), the trading data of the previous day was used in its

place. This step was crucial to ensure that trading dates were available and consistent for each and all securities after their listing date.
I also set a minimum length of trading days for a security to be considered: 1 year of trading data (circa 250 trading sessions, 250 rows in the data set) for a model forecasting 1 day forward. For a model forecasting 5 trading days forward, 5 years of data are necessary, however, this is still 250 rows since the model is built in 5 days intervals. This means that securities which have recently been listed may not be part of the set. Longer horizon forecasts, like 30 days for example, would require 30 years of data for example. I use this length of temporal data as a rule thumb --as opposed to a hard rule--, and the parameter to modify this rule can easily be adjusted before running the code.

Predicting a future closing price of a target security –let's choose AAPL (Apple) as an example-- on a future date np trading periods forward based on the values of other predictors, requires that we know the values of those predictors at a future time. But we don't. The future closing price (as well as open, high, low, and volume) of AAPL and all other securities, is, ...well…, in the future; we don't know them yet.
The solution then is to lag all predictors np periods forward. Dates and the Close Price of the target security (AAPL) remain unchanged, and the rest of the dataframe shifts np periods rows into the future. I have (arbitrarily) set the number of np periods to 5; but it can be any number. A 1-day period forward would predict Close Prices in the next trading day.

Building a model to predict np=5 trading days forward does not require that we take into account price fluctuations within each np=5 trading period. Hence, the dataframe adjusts (reduces) to np=5 by taking date increments of np=5 periods, starting from the target prediction date (2025-02-20), back to the beginning. This is done dynamically, automated in the code, depending on the input we provide to np. In other words, if we plan to predict np=5 trading days forward we build a model with dates with incremental steps of np=5 trading days: 2025-02-20, 2025-02-12, 2025-01-05….. all the way to the beginning. Note that calendar days and trading days may differ. Markets trade on trading days, and there are about 250 of them in a year. The rest are weekends or holidays in which markets do not trade.

Training Set and Testing Set split:

The purpose of all models is to predict np=5 trading periods forward. The simplest testing set is the last row of the dataframe. The testing set is the row dated 2025-02-20. The rest is the training set. The predictors data is lagged np=5 periods, except for the target stock close price AAPLClose.

The target stock name AAPL and np=5 periods are used here as examples to bring clarity. The target Stock and number of np periods can be easily changed.

We do not directly use the value of AAPLClose on 2025-02-20 in the testing set. That's the value we are trying to predict. We use AAPLClose on 2025-02-20 in the very last step to measure the prediction error. The prediction is made with predictor's test data shown in row Date 2025-02-20. That is a lagged row from 2025-02-12.

Note that we loose 1 row at the very beginning of the data set when we lag the set. This row shows NaN for predictors and it is ignored by the model.

Building the Models:

Single-predictor linear regression:

This initial (naive) first step involves fitting a linear model to all single predictors individually (circa 2,500). All predictors (Open, High, Low, Close, Volume) belong to securities with at least 250 rows of history. These models predict 5 days forward, hence, the minimum of 250 rows of data represent a minimum of about 5 years of data.

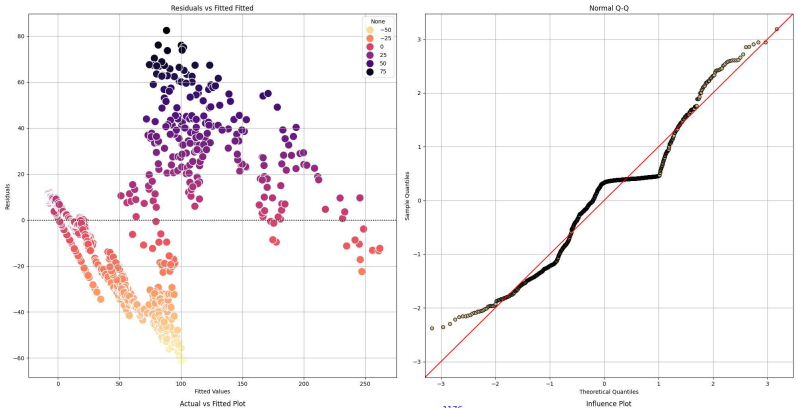
The single-predictor model is considered good if the pvalue of the predictor's coefficient is less than a significance level alpha (set at 0.01).

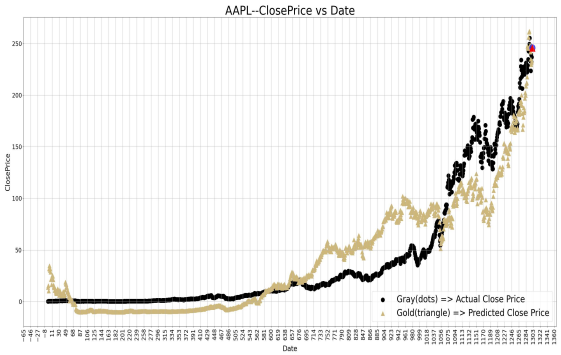
The lowest-test-error single-predictor model has the following structure:

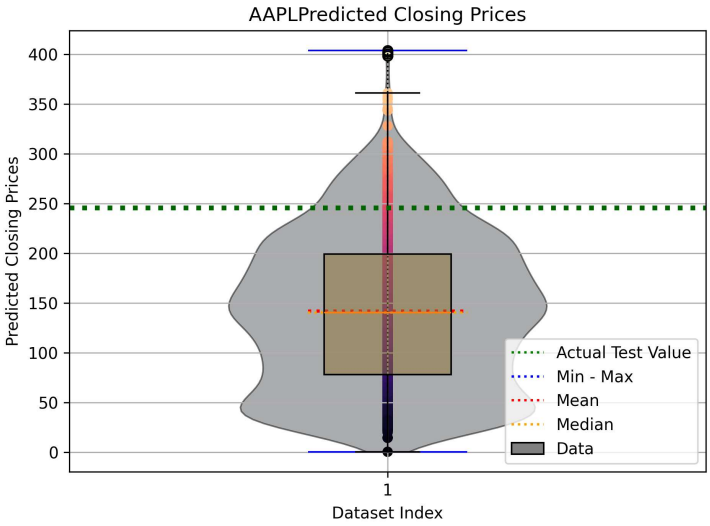
```
Line 1133:
lowestErrorSPModel                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:              AAPLClose   No. Observations:                 1301
Model:                            GLM   Df Residuals:                     1299
Model Family:                Gaussian   Df Model:                            1
Link Function:               Identity   Scale:                          667.84
Method:                          IRLS   Log-Likelihood:                -6075.9
Date:                Tue, 04 Mar 2025   Deviance:                   8.6753e+05
Time:                        10:49:39   Pearson chi2:                 8.68e+05
No. Iterations:                     3   Pseudo R-squ. (CS):             0.9896
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    -10.7748      0.982    -10.975      0.000     -12.699      -8.851
BKNGHigh       0.0511      0.001     77.069      0.000       0.050       0.052
==============================================================================
lowestTestErrorPct= 0.0577 %
Number of models= 2384
testActualClose= 245.83
meanTestPredSP= 141.86 across all models all models.
meanTestErrorPctSP= -42.2926 % ; mean test error across all models.
meanRMSETrainErrorPctSP= 13.28 %

Total Time:  2376.8 secs.
```

The test prediction of 245.69 on 2025-02-20 and its test error of 0.06% measure well against the actual value of 245.83. Linearity, as indicated in the fitter vs actual looks fine, with the caveat that under and over prediction are not evenly spread. The Residuals vs Fitted plot (below) shows no identifiable patterns that would indicate heteroskedasticity (unequal variance) in the residuals, and the QQ plot shows that there no concerning departures from normality in the residuals.

The influence plot further shows a large number of influential datapoints. While there is no immediate reason to remove these data points, we could consider making the data set shorter, supported by the notion that closing prices 40 years ago are likely to be less relevant in current days.







While the model with the lowest test error offers quite an accurate prediction, the predictions of all 2384 single predictor models (with sound pvalues) show a wide range of variation, with an average of 141.86 for the close Price on 2020-02-20, and an average prediction error of -42.29%.

While this is a good start, there is plenty of room for improvement, both in the structure of the model, and the computational time.

Multiple-predictor linear forward-step-wise regression built on random sampling of predictors:

This approach is still along the lines of linear regression and is based on the random sampling of nSeq=20 securities (each having 4 to 5 predictors: open, high, low, close and some volume), fitting a linear regression models to them, and predicting and measuring error on the test set. Prior to building the regression model, I verify collinearity amongst predictors by computing the variance inflation factor (VIF) for each predictor, and eliminating those with VIF > VIFMax (set at 5).

I then build the regression model by adding predictors one at the time in a forward-step-wise progression.

The regression model keeps the significant predictors with p-values lower than a given significance level (set at alpha=0.01), and discards those above it.

The resulting regression model makes a prediction on unseen test data, a prediction error on the testing data is measured and becomes the parameter of model selection at the end of the process.

This is the first model.

The process of random sampling nSeq is repeated a number of times (set at nModels = 500), with different sampling seeds each time –which ensures that there are no repeated random samples, yet allowing the results to be replicable--.

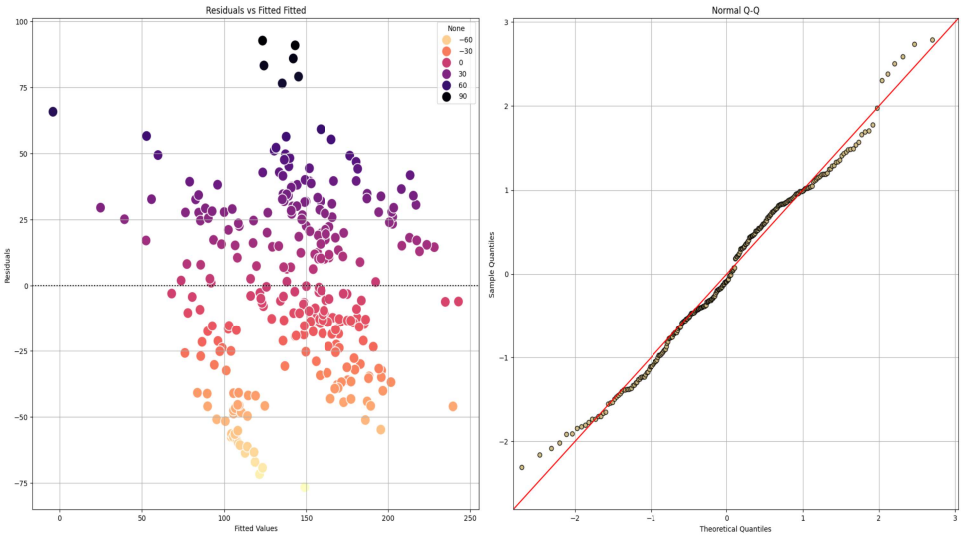
Each of the nSeq security random samples builds a model, each one delivering a prediction, and error estimates. At the end of the process we are able to select the “best” model out of the nModels created – the one with the lowest prediction error, for example--, or simply take the solution as a collection of nModels, which jointly deliver a mean (or median) prediction and an error estimate as a group.

The lowest-test-error multi-predictor model has the following structure:

```
Line 1219:
lowestModel=                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:              AAPLClose   No. Observations:                  289
Model:                            GLM   Df Residuals:                      285
Model Family:                Gaussian   Df Model:                            3
Link Function:               Identity   Scale:                          1120.5
Method:                          IRLS   Log-Likelihood:                -1422.7
Date:                Thu, 06 Mar 2025   Deviance:                   3.1933e+05
Time:                        06:37:20   Pearson chi2:                 3.19e+05
No. Iterations:                     3   Pseudo R-squ. (CS):             0.7487
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept   -104.0865     23.180     -4.490      0.000    -149.519     -58.654
AAPLVolume -3.888e-07   4.37e-08     -8.899      0.000   -4.74e-07   -3.03e-07
AEEOpen        3.5373      0.276     12.834      0.000       2.997       4.078
AEEVolume    1.17e-05   3.01e-06      3.883      0.000    5.79e-06    1.76e-05
==============================================================================
```

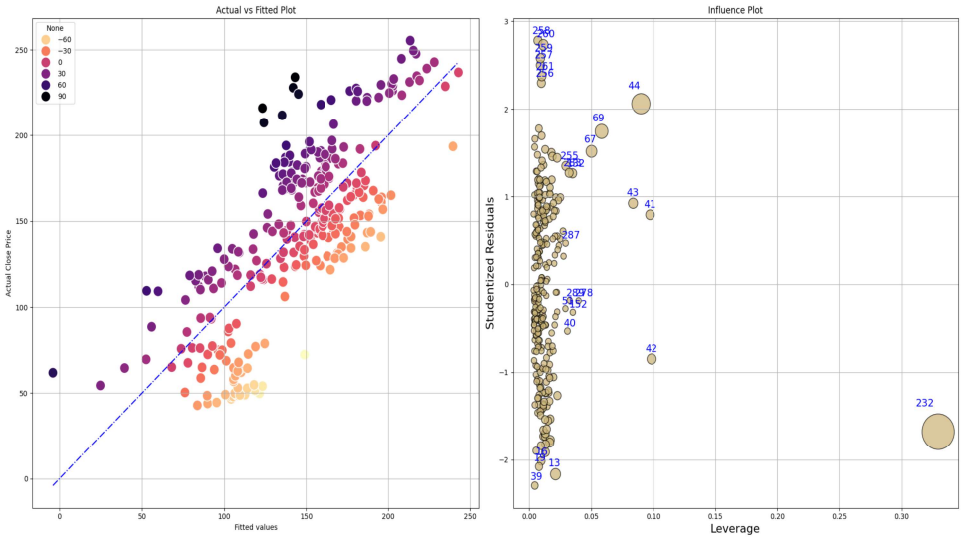
```
lowestPredError%= 0.0105 '%'on test set.
lowestRMSEPct= 13.52  '%' average error on the training set.
closeActual= 245.83  the actual closing Price on test date. 2025-02-20
lowestClosePred = 245.8  the predicted closing Price on test date.

Line 1306: This predictor= AAPLVolume R2= 0.21826264148013774 vif= 1.279202009602554 is good on vif.

Line 1306: This predictor= AEEOpen R2= 0.18640760268860057 vif= 1.2291166968922078 is good on vif.

Line 1306: This predictor= AEEVolume R2= 0.07757293239640073 vif= 1.0840965482484484 is good on vif.

Line 1232: nModels= 481 samples of random securities= 20
meanPredError= 8.38 '%'
meanRMSEPct= 8.52 '%'
meanClosePred= 231.27

Line 1304: dwStat= 0.35299861477533556

Line 1311: time= 14065.0
```

The test prediction of 245.80 on 2025-02-20 and its test error of 0.0105% measure really well against the actual value of 245.83.



Linearity, as indicated in the fitted vs actual plot looks very good, and significantly better than the initial single predictor model.

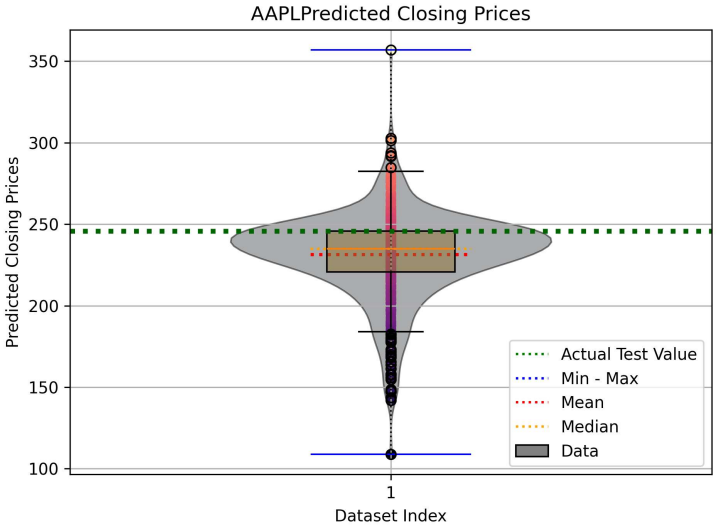
The Residuals vs Fitted plot shows no identifiable pattern that would indicate heteroskedasticity (unequal variance) in the residuals.



The Durbin-Watson's test delivers a stat of 0.35. A dw stat < 1 indicates that there could be strong autocorrelation of residuals, hence the errors are not independent from one another.

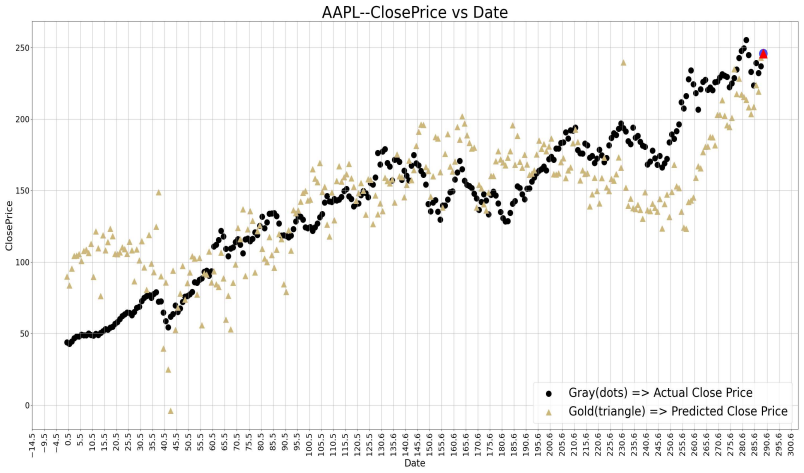
The QQ plot shows a roughly normal distribution, with no concerning departures from normality in the residuals.

The influence plot shows one large influential datapoint. Given that this is a time series, I prefer not to remove outliers. The data points in the series represent price events in the evolution of the stock over its trading life time. Even if we may consider these events black-swan type events, there is no certainty that they may not repeat in the future.



The average prediction over 500 models fitted to random samples of 20 securities (in total between 80 and 100 predictors for 20 securities) is 231.27 with an average error on the test set of 8.38%. The RMSE is computed over the training set of the lowest-test-error model on each of the nSeq models. The average over nSeq models is 8.52%.
This is an improvement over the initial single-predictor model, with the trade-off of high computational time of nearly 4 hours (for 500 models).
The fact that the RMSE (over the training set) is marginally lower than the average test error could suggest overfitting, although the plots and model output parameters suggest that there is plenty of deviance still be worked on.
It was worth the effort, but there is still room for improvement both in computational time, and average errors.

It is worth noting that the length of the time series may vary depending depending on the security/ies selected to build the model. This is due to the fact that different securities began trading for the first time at different points in history. The single predictor model has 1,301 data points in its time series, while this multi-predictor model relies on



just 289 observations. Since prediction is the purpose of the model, this is not a concern. Besides, both single and multi predictor models trading days are the same over the length of the shorter of the time series. All data series end on 2025-02-12 for the training set, and predict on 2025-02-20 (for a 5-day forward prediction), no matter how far back they extend.

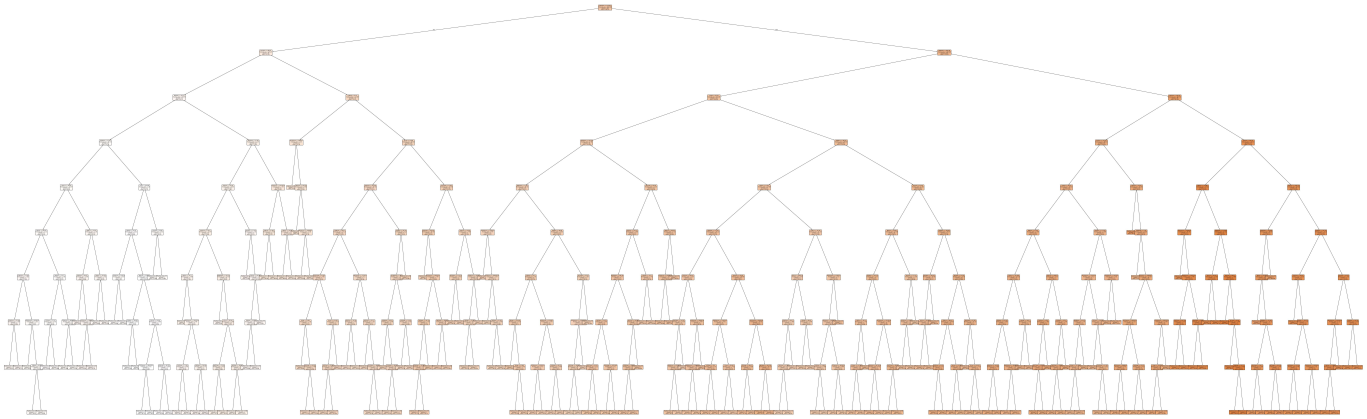
Single Decision Tree:

Given the high-dimensional nature of the data set, Decision Trees offer a suitable and efficient solution to the problem. I use the standard DecisionTreeRegressor function available in sklearn.tree (4) with the only precaution to pass a data frame at least 5 years long. Since I am predicting 5 days forward, the data set is built in increments of 5 days. The securities in the dataset have at least 250 rows of 5-day increments covering 5 years of trading.

I tune the hyperparameters error criterion, maximum depth, and minimum sample splits, minimum sample leafs, and maximum number of features through a grid seach cross validation process with a function GridSearchCV also available in sklearn (5). The cross validation cv parameter of the GridSearchCV function is set to the one suitable for time series – also a function TimeSeriesSplit available in sklearn (6)--.

```
Line 337: yTest [245.83000183]

Line 1336: DecisionTreeRegressor object complete.

Line 1359 Best Hyperparameters: {'criterion': 'friedman_mse', 'max_depth': 9, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2}
bestScore= 2.8068055606009397

Line 1362: DecisionTreeRegressor fit complete.


Line 1365:
dtPred= [244.73092651]

Line 1373:
dtrMSE= 1.21 dtrRMSE= 1.1
dtrRMSEPct= 0.45 '%'

Line 1388: time= 13.0 secs.
```

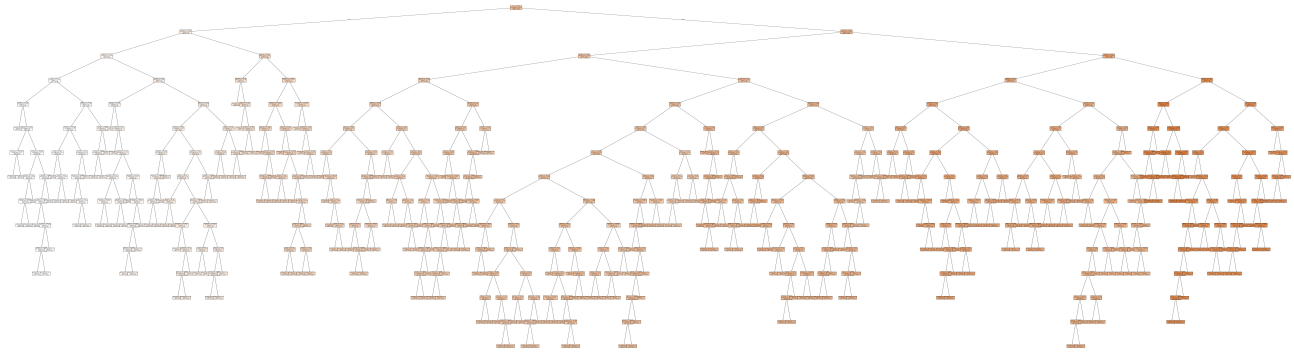
The results are very good; better than the prior initial linear regression models. The prediction of 244.73 is within 0.45% of the actual value of 245.83. The implementation in code is very simple and succinct, and the computing time efficient, about 13 seconds (may be a minute or so if we include the parameter tuning time). I spent most of time in the fine tuning of parameters.



Since the height of the tree was left free to optimize, the tree is rather large, with a height of 9, but still manageable.

Random Forest:

Random Forest is the natural evolution of the single decision tree approach above. A forest handles high-dimensional data very well. I begin with fitting a random forest of 100 trees with the standard function RandomForestRegressor available in sklearn [7]. This is the first tree in the forest.



Parameter tuning followed the initial fit.

This is the most time-consuming part in the random forest fitting process. While I decided to do it manually, to get a better feel of how the grid search works, it can be easily automated. I left the different values explored commented, and the results I got through them, for reference.

The following parameters are tuned to achieve the lowest error:

```
paramGridrfr = {
        #"n_estimators": list(np.arange(3200,3500,1)), #best 3281 3.69%
        #"n_estimators": list(np.arange(1600,5000,50)), #best 3300 3.69% # 3724 error 3.67%
        #"n_estimators": list(np.arange(100,3000,100)), #best 1800 3.78%
        #"n_estimators": [100], #error 3.60%
        #"n_estimators": [216], #list(np.arange(200,350,1)), # best is 216 error 3.53%
        #"n_estimators": list(np.arange(20,120,1)), # best is 32 error 3.62%
        "n_estimators": [32], # error 3.62%
        "criterion": ["absolute_error"], #["squared_error", "friedman_mse", "absolute_error", "poisson"],
        "max_depth": [None], #np.arange(1,10,1), #list(np.arange(1,10,1)),# 30, 40, 50], best is 4
        "min_samples_split": [2], #np.arange(1,20,1), #[None,2,3,4,5,6,7,8,9,10],# 10, 20], best is 4
        "min_samples_leaf": [1],#np.arange(1,10,1),
        "max_features": ["sqrt"], #[1,"log2", "sqrt"],
        "bootstrap": [False] # True, # No bootstrapping: the whole data set is used in each tree.
        # Max sample: this parameter controls the size of the sample for each tree. Default=whole dataset is used.
        }
```

```
Line 1423 Random Forest Best Hyperparameters: {'bootstrap': False, 'criterion': 'absolute_error', 'max_depth': None, 'max_features': 'sqrt', 'min
_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 32}
bestScore= 1056.3332014764514

Line 1444: RandomForestRegressor Tuned fit complete.


Line 1446:
rfrTunedPred= [236.9319458]

Line 1456:
rfrTunedMSE= 79.18
rfrTunedRMSE= 8.9
rfrTunedRMSEPct= 3.62 %
rfrTestErrorPct= -3.62 %

Line 1475:
TargetStock: AAPL
Random Forest time= 32.0 secs.
```

The lowest error of 3.53% is achieved with 216 estimators. This is marginally better than the 3.62% achieved with just 32 of them, which provide an estimate of 236.93.

The 32-mark is the point at which increasing the number of estimators in the forest does not meaningfully improve the error, while computational cost continues to rise.

The bootstrap parameter indicates whether the random forest will randomly select (rows of) data. I set the bootstrap flag to ‘False’ to indicate that each tree in the forest should fit all the time series data in the training set. The function randomly does select predictors randomly(with replacement) though, for each tree.

While the random forest did not achieve better accuracy than a single tree, it adjusted for overfiting that single decision tree regressors are known to produce.

Ada Boost on Random Forest:

Adaptive Boosting is an ensemble technique that focuses on improving the weak learners. The standard AdaBoostRegressor [8] function available in sklearn.

The key parameters to tune in the Ada Boost algorithm are:

```
paramGridadarfr = {
    "n_estimators": [50,100,200],
    #"estimator__max_depth": [None, 1,3,5],
    "learning_rate": [0.01, 0.1, 1.0],
    }
```

The tuning of parameters is done through Grid Search, and their optimal combination yields the following results:

```
Line 1511 AdaBoostRandom Forest Best Hyperparameters: {'learning_rate': 1.0, 'n_estimators': 50}
bestScore= 1078.3308170483674

Line 1532: Ada Boost RandomForestRegressor Tuned fit complete.


Line 1534:
adarfrTunedPred= [238.85747337]

Line 1538:
adarfrTunedMSE= 48.62
adarfrTunedRMSE= 6.97
adarfrTunedRMSEPct= 2.84 %
adarfrTestErrorPct= -2.84 %

Line 1553:
TargetStock: AAPL
Ada Boost Random Forest time= 171.0 secs.
```

This is the most computationally expensive technique amongst those ensemble techniques so far. The model delivers a prediction of 238.85 with an estimated error of 2.84%.

Gradient Boosting:

Gradient Boosting is yet another ensemble technique, and a standard function GradientBoostingRegressor [9] is also available in sklearn.

This technique reduces the error sequentially by modeling the residuals of the previous model. The key parameters to tune in Gradient Boost are:

```
paramGridGradBoost = {
    'n_estimators': [110], #[90,100,110,120], # 300, 400], #200 3.09%; 120 3.09%
    'learning_rate': [0.2], # [0.01, 0.1, 0.2], #0.1
    'max_depth': [None], #[3, 4, 5], #4
    'min_samples_split': [2], #[2, 5, 10], #5
    'min_samples_leaf': [1], #[1, 2, 4], #2
    'subsample': [1], #[0.8, 0.9, 1.0],
    'max_features': ['log2'] #[1, 'sqrt', 'log2'] #['auto', 'sqrt', 'log2'] #sqrt
}
```

The tuning of parameters was done through Grid Search. After 110 n_estimators, the error values stops dropping. The optimal combination of parameters yields the following results:

```
Line 1590 Gradient Boost Hyperparameters: {'learning_rate': 0.2, 'max_depth': None, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2,
_estimators': 110, 'subsample': 1}
bestScore= 3.412512502659875

Line 1604: Gradient Boost Regressor Tuned fit complete.


Line 1606:
gradBoostTunedPred= [238.23710476]

Line 1617:
gradBoostTunedMSE= 57.65
gradBoostTunedRMSE= 7.59
gradBoostTunedRMSEPct= 3.09 %
gradBoostTestErrorPct= -3.09 %

Line 1623:
TargetStock: AAPL
Gradient Boost Regressor time= 17.0 secs.
```

These results are similar to Ada Boost.

The model delivers a prediction of 238.24 with an estimated error of 3.09%.

eXtreme Gradient Boost (XGB):

Based on the same principles of Gradient Boost, XGB offers additional features such as regularization, speed and efficiency. The function is available in the xbgoost [10] library, which interacts well with sklearn. The key parameters to tune in Gradient Boost are:

```
paramGridXGradBoost = {
    'n_estimators': [80], #[100,200,300], # Higher number of estimators reduces error/increases computational time.
    'learning_rate': [0.1], #[0.01, 0.1, 0.3], # Lower rates lead to better generalizations.
    'max_depth': [None], #[3, 4, 5, 6], # The depth of the trees, typically between 2 and 10.
    'min_child_weight': [3], #[1, 3, 5],  #
```

```
    'subsample': [1], #[0.7, 0.8, 0.9], # Fraction of samples to train the tree. I will 1 to use all the time series.
    'colsample_bytree': [0.8], #[0.1,0.2,0.3,0.4,0.5,0.6,0.7, 0.8, 0.9], # Fraction of features (predictors).
    # Add regularization parameters:
    'reg_alpha': [0], #[0, 0.1, 0.5], # L1 Lasso regularization.
    #Adds a penalty term proportional to the value of the coefficients.
    'reg_lambda': [1] #[0.1, 1, 5] #L2 Ridge regularization.
    #Penalty to the squared value of the coefficients.
}
```

This set of parameters yields a predicted 240.71 with an error of 2.08%

```
Line 1663 XGB Hyperparameters: {'colsample_bytree': 0.8, 'learning_rate': 0.1, 'max_depth': None, 'min_child_weight': 3, 'n_estimators': 80, 'reg_alpha': 0, 'reg_lambda': 1
, 'subsample': 1}
bestScore= 1180.7493688552192

Line 1677: XGB Regressor Tuned fit complete.


Line 1679:
XGBTunedPred= [240.70522]

Line 1690:
XgradBoostTunedMSE= 26.26
XgradBoostTunedRMSE= 5.12
XgradBoostTunedRMSEPct= 2.08 %
XgradBoostTestErrorPct= -2.08 %

Line 1697:
TargetStock: AAPL
XGB Regressor time= 130.0 secs.
```

Stacking:

We have so far predicted close prices for a target stock (AAPL in this case) on a given date, a number of nperiods forward (5 in this case, with a target date of 2025-02-20), through the following models:

1) single predictor linear models.
2) multi-predictor linear models.
3) Single Decision Tree.
4) Random Forest.
5) Ada Boost Random Forest.
6) Gradient Boost.
7) Xtreme Gradient Boost.

Stacking is a technique that allows us to combine the results of multiple models and produce a final output through a final model. The function StackingRegressor [11] is available in sklearn. I configured the function with the base estimators 3 through 7 in the list above, which feed the final output into a final estimator RidgeCV. The final result predicted by RidgeCV is 261.06, which implies an error of 6.2%. This result is rather unexpected. Most of the models have underpredicted, but the combination of them overpredicts by a rather large margin. This may be partially explained by:

- the fact that the linear regression models (1) and (2) are not part of the stacking process: these two models where built with the statsmodels package [12] which requires a wrapper to be compatible with sklearn --under which all other models where built, including the stacking ensemble--.
- the stacking function performs cross validation. I have conducted cross validation during the process of construction of all models considering that the data is a time series --with an inherent sequential nature--, and passing the time-series-appropriate object to the cv parameter. The stacking function does not take this time-series-specific object, and as a consequence, the cross validation is done by selecting data (rows) at random. This is not suitable for time series.

I opted for taking an average of the tree and ensemble predictions and errors already obtained instead of using the more sophisticated stacking function.

The mean prediction of all 5 ensemble models (3 through 7) on the list above is 239.89, with an error estimate of 2.4%.

This is one stock, AAPL.

We have built 5 ensemble models, which work together to make a close price prediction nperiods=5 trading days forward. We tested and fined tuned the hyper parameters for all 5 models on the prediction day (2025-02-20). The last step is to test out-of-sample data. The set will be the rest of the S&P500, all 499 of them (except AAPL, which was the one used to fine tune hyperparameters). We will run the model on every single one of them, measure the error individually, and produce a simple statistics for the model prediction error. Given the (heavy) computational load associated with single and multi-predictor linear regression, I will rely at this stage on decision trees, random forest and ensemble-based models.

Conclusions:

While single and multi-predictor regression models provide a “quick” and easy way to get the solution of the problem started, the computational resources required to do so does away with the “quick”, delivers a very wide range of predictions --mostly out of acceptable range for the purpose of this effort--, and in the case of multi predictor models, gets conceptually too close to random forest solutions in a non-efficient manner.
The high number of dimensions in the data, evidenced in the large number of predictors involved, makes decision trees, random forests an ensemble techniques more suitable, and more accurate.

```
Line 1607: llModelsDF:
    ModelName  meanPrediction  MeanTestErrorPct
0      DTree      244.730000           -0.4471
1         RF      236.930000            3.6196
2      adaRF      238.860000            2.8363
3  gradBoost      238.240000            3.0887
4        XGB      240.710007            2.0847
meanPredAllModels= 239.89
meanErrorAllModels= -2.42 %
```

It is always tempting to take the lowest-error result. That would have been a single predictor model with a test error of 0.057%. Such a model would look too simple and too good to be true under an intuition test, and would likely exhibit high variance. While I will keep it handy in my desk's drawer, I will not take it as an acceptable final result. I find the mean prediction of all trees and ensemble models (3 to 7 on the list above), providing a mean prediction of 239.89 with an error estimate of 2.42%, the most adequate answer to this problem.

For Microsoft, the prediction and error is:

```
Line 1607: llModelsDF:
    ModelName  meanPrediction  MeanTestErrorPct
0      DTree          430.53            3.4605
1         RF          420.67            1.0912
2      adaRF          423.30            1.7236
3  gradBoost          421.19            1.2163
4        XGB          424.00            1.8917
meanPredAllModels= 423.94
meanErrorAllModels= 1.88 %
```

And for all 500 constituents of the S&P 500:

```
Close Price Prediction on 2025-02-20 for 500 S&P500 securities
============================================================
     Ticker  ActualClose    MeanPred  MeanErrorPct  CorrectedClosePred  CorrectedErrorPct
0       MOS    26.620001   26.782000      0.608561           26.968840           1.306195
1       HPE    21.740000   21.394000     -1.591535           21.543252          -0.893902
2        KO    70.040001   66.652000     -4.837237           67.116987          -4.139604
3       IQV   194.009995  211.714001      9.125307          213.190988           9.822940
4      PARA    11.470000   10.864000     -5.283350           10.939791          -4.585717
..      ...          ...         ...           ...                 ...                ...
403     ROP   581.419983  557.724000     -4.075536          561.614867          -3.377903
404    ROST   139.089996  146.738001      5.498602          147.761694           6.196235
405     RSG   230.860001  217.527999     -5.774929          219.045546          -5.077296
406     RTX   125.110001  120.234000     -3.897371          121.072792          -3.199738
407    RVTY   114.720001  114.402000     -0.277197          115.200107           0.420436

[408 rows x 6 columns]
meanErrorPct= -0.7 %
```

The estimate for the average error across all unseen data (the 499 securities in the S&P500 except AAPL) is the most important output, and it stands at -0.71%. This means that the ensemble of models we have just put together, underpredicts by -0.71%, on average.

While future work is discussed in the next section, I will correct estimates with this mean error accordingly. For APPL for example, the prediction is 239.89, which corrected (upwards) by the mean error of 0.71% becomes 241.59 and final estimate error of -1.72%.

Future Work:

The single and multi-predictor models took the largest share of the coding effort, while decision trees, random forest, and ensemble techniques relied on efficient and optimized library functions. Relying on time-series specific libraries may help manage code length and efficiency.

Predicting the closing price of 500 securities and determining the distribution of the prediction error, was done based on models built and tuned on one stock (AAPL). A cleaner solution would be tuned hyperparameters to for each specific security. This approach is supported by the fact that the dataset on which the hyper parameters is different for each stock. This is technically quite simple to code, but computationally very expensive. Solving this piece would enhance the quality of the results –and most likely their accuracy too--.

The results presented in the preceding pages are based on the 500 or so S&P500 constituents securities. They make about 10% of all available listed securities in the US, and a tiny fraction of those available worldwide. Extending this framework beyond the S&P500 in the US and later worldwide would be a natural next step.

References:

[1] https://finance.yahoo.com/quote/%5EGSPC/

[2] https://pypi.org/project/yfinance/

[3] https://en.m.wikipedia.org/wiki/List_of_S%26P_500_companies

[4] Decision Tree Regressor from sklearn.tree:
https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

[5] Grid Search Cross Validation:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[6] Cross Validation cv parameter for time-series:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html

[7] Random Forest Regressor:
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[8] Adaptive Boosting (ADA):
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html

[9] Gradient Boosting (GB):
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

[10]Xtreme Gradient Boosting XGB:
https://scikit-learn.org/stable/modules/generated/
sklearn.ensemble.GradientBoostingClassifier.htmlXtreme

[11] Stacking Regressor:
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingRegressor.html

[12] Statsmodels GLM:
https://www.statsmodels.org/stable/glm.html

[13] https://www.geeksforgeeks.org/residual-leverage-plot-regression-diagnostic/

[14] https://stackoverflow.com/questions/66493682/glm-residual-in-python-statsmodel

----------THE END----------