Stock Price Indicator

Project Overview

Investment firms, hedge funds and even individuals have been using financial models to better understand market behaviour and make profitable investments and trades. A wealth of information is available in the form of historical stock prices and company performance data, suitable for machine learning algorithms to process.

Here, I build a stock price predictor that takes daily trading data over a certain date range as input, and outputs projected estimates for given query dates. Inputs contain multiple metrics:

- opening price (Open)
- highest price the stock traded at (High)
- how many stocks were traded (Volume)
  Whereas the prediction is made on:
- closing price adjusted for stock splits and dividends (Adjusted Close)

We want to explore, where possible, S&P500 companies. The list of the listed companies is taken from here.
The trading data comes from Quandl End-of-Day US stock prices and it is downloaded through their API and stored locally. Details are provided into the `Data Exploration` section below.

Project Format and Features

The project is developed by referring to the cross industry standard process for data mining (CRISP-DM) methodology. Here we summarise the steps undertaken:

- Business Understanding
- Data Understanding
- Data Preparation
- Modelling
- Evaluation
- Deployment

The project is developed in the form a written report.

Problem Statement

The aim of the project is to predict stock's adjusted closing prices at 1, 7, 14 and 28 days, by giving stock data a time window of n precedent days. The model should work for highly relevant companies, therefore a few from S&P500 are selected due to data availability.

Data is collected and wrangled to provide training and testing data in the format of features (open, min, max, close, split and dividend) and output vector (the adjusted closing price at the relative time shifts). Data is standardised and a experiments are conducted to test and improve performances.

Metrics

I used Mean Squared Error to measure performance of the model during training. As a reminder, it is the mean of the squared difference between predicted and true values:

$$RMSE = \frac{1}{n}\Sigma\sqrt{(Y_i - \hat{Y}_i)^2}$$

which is preferred to the mean absolute error due to its tendency to penalise more bigger errors. By doing so, The aim is to drive to model to avoid big errors rather than producing more equally distributed ones.

Requirements

Requirements for the python environment can be found in the `requirements.yml` file associated with this repository.

Analysis

Data Exploration

Features and calculated statistics relevant to the problem have been reported and discussed related to the dataset, and a thorough description of the input space or input data has been made. Abnormalities or characteristics about the data or input that need to be addressed have been identified.

	Date	Open	High	Low	Close	Volume	Dividend	Split	Adj_Open	Adj_High	Adj_Low	
0	2017-12-28	171.00	171.850	170.480	171.08	16480187.0	0.0	1.0	165.971205	166.796208	165.466497	1
1	2017-12-27	170.10	170.780	169.710	170.60	21498213.0	0.0	1.0	165.097672	165.757675	164.719142	1
2	2017-12-26	170.80	171.470	169.679	170.57	33185536.0	0.0	1.0	165.777087	166.427383	164.689053	1
3	2017-12-22	174.68	175.424	174.500	175.01	16349444.0	0.0	1.0	169.542983	170.265103	169.368277	1
4	2017-12-21	174.17	176.020	174.100	175.01	20949896.0	0.0	1.0	169.047981	170.843576	168.980040	1

Data Integrity and statistics

```
Date          0
Open          0
High          0
Low           0
Close         0
Volume        0
Dividend      0
Split         0
Adj_Open      0
Adj_High      0
Adj_Low       0
Adj_Close     0
Adj_Volume    0
Ticker        0
dtype: int64
```

And statistics result reasonable. More specifically:

- `Open`, `High`, `Low` and `Close` show similar mean and standard deviation, as we would expect from daily data. Also the `Adj_*` relative ones report a similar trend.
- `Dividend` is mostly zero, which makes sense as these are usually distributed monthly, quarterly or annually. Therefore, most of the daily data points should be zero.
- `Split` report whether stocks are split in the event of high price. This is quite of a rare event to happen, so it is reasonable that no split are present throughout almost all our data points. It is worth noting that no aggregation is performed in this dataset, as there are no data points with a split value less than 1.

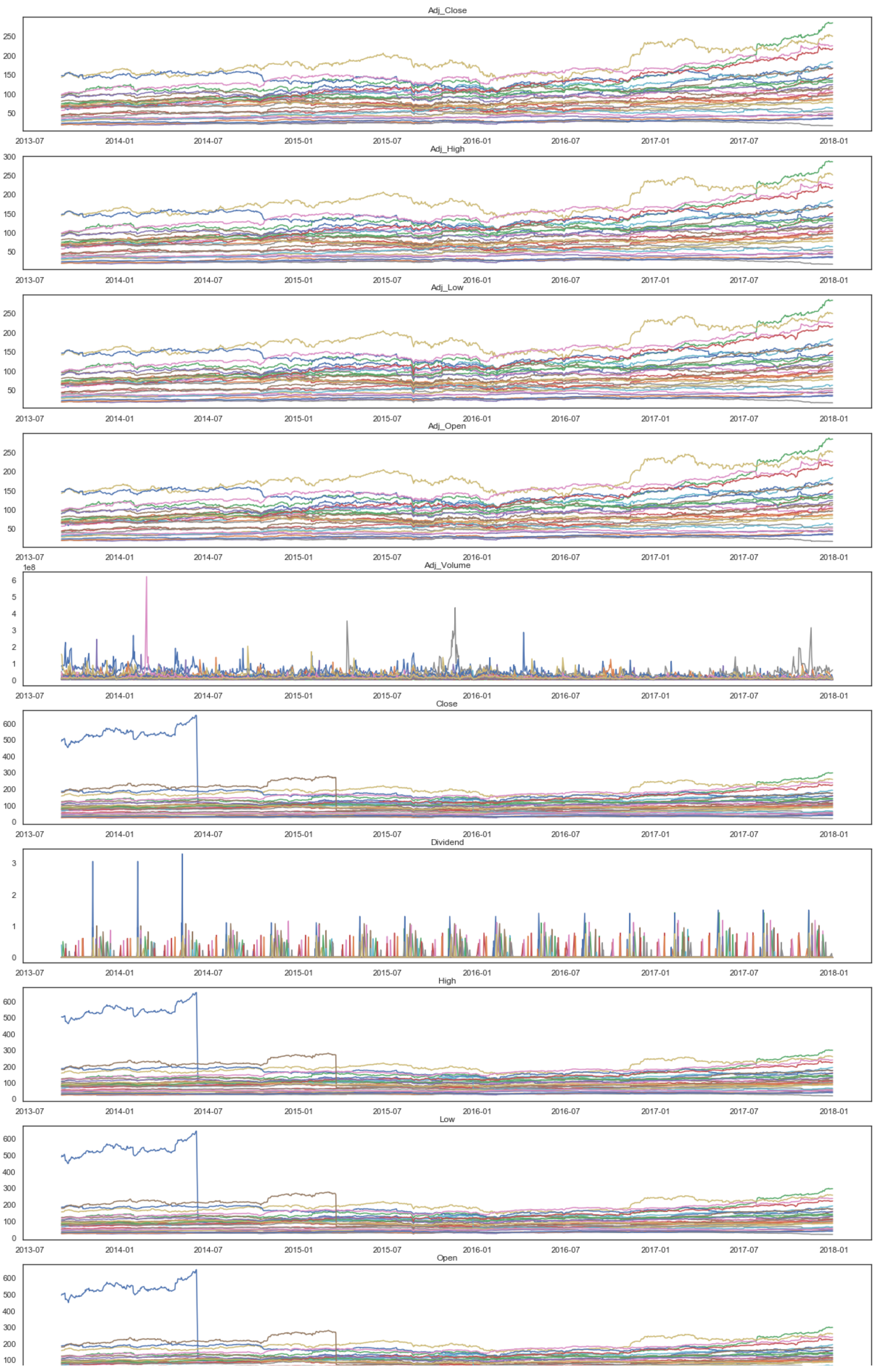
	Open	High	Low	Close	Volume	Dividend	Split	Adj_Open	Adj_High	Adj_Low
count	31610	31610	31610	31610	31610	31610	31610	31610	31610	31610
mean	95.491669	96.171887	94.818211	95.521521	1.226938e+07	0.009154	1.000316	81.437157	82.014518	80.866317
std	59.333158	59.766797	58.916946	59.355129	1.500732e+07	0.085350	0.038147	43.340742	43.637986	43.056648
min	17.350000	17.400000	17.250000	17.360000	3.053580e+05	0.000000	1.000000	16.190719	16.250819	16.102600
25%	55.372500	55.870000	54.980000	55.395000	3.847515e+06	0.000000	1.000000	47.642711	48.065195	47.263770
50%	87.845000	88.400000	87.170000	87.840000	7.197633e+06	0.000000	1.000000	76.139854	76.664216	75.628460
75%	118.560000	119.290000	117.707500	118.580000	1.526566e+07	0.000000	1.000000	105.912443	106.673449	105.180398
max	649.900000	651.260000	644.470000	647.350000	6.166205e+08	3.290000	7.000000	286.481931	286.913264	284.785353

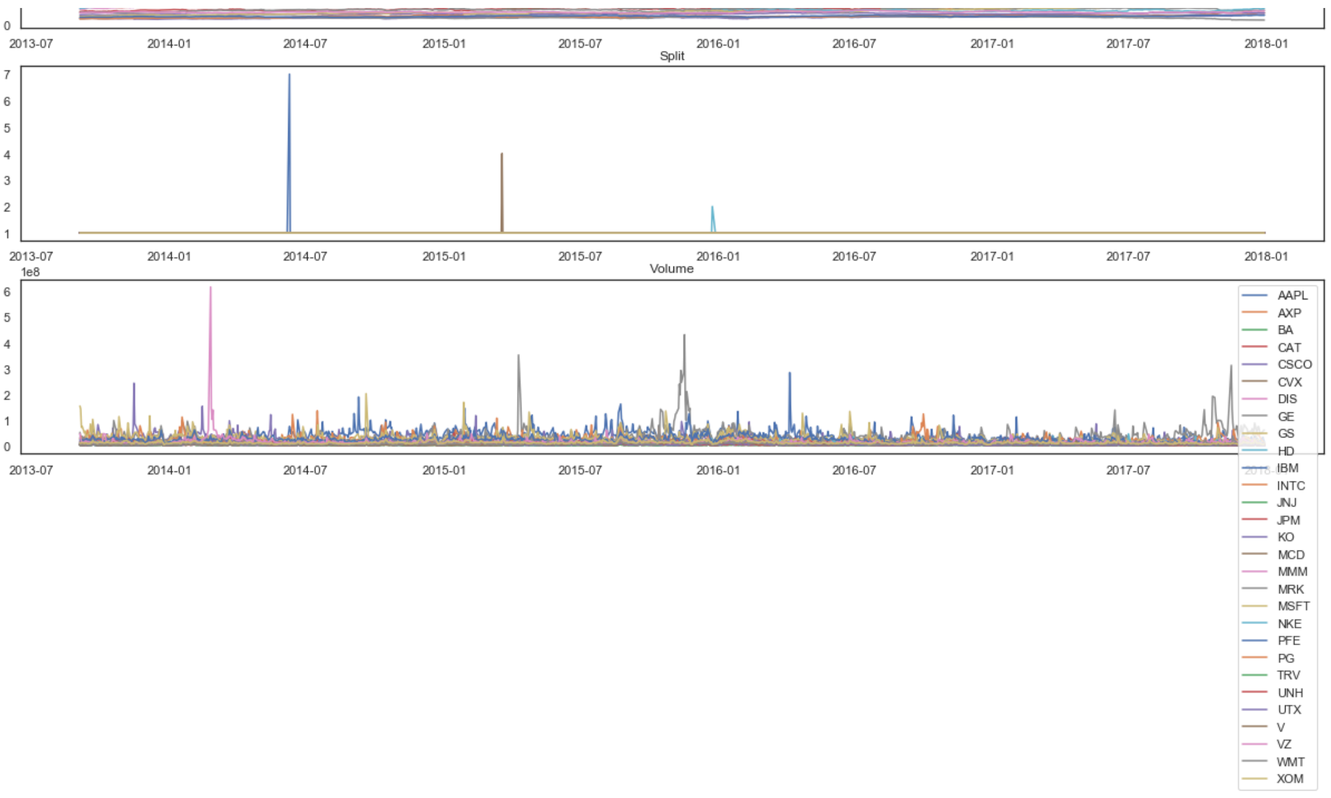
Timeframe

The time frame available from Quandl dataset is equal for all the stocks, and spans through 1577 days from 03/09/2013 until 28/12/2017 (see: appendix, table A.1).

Data Visualization

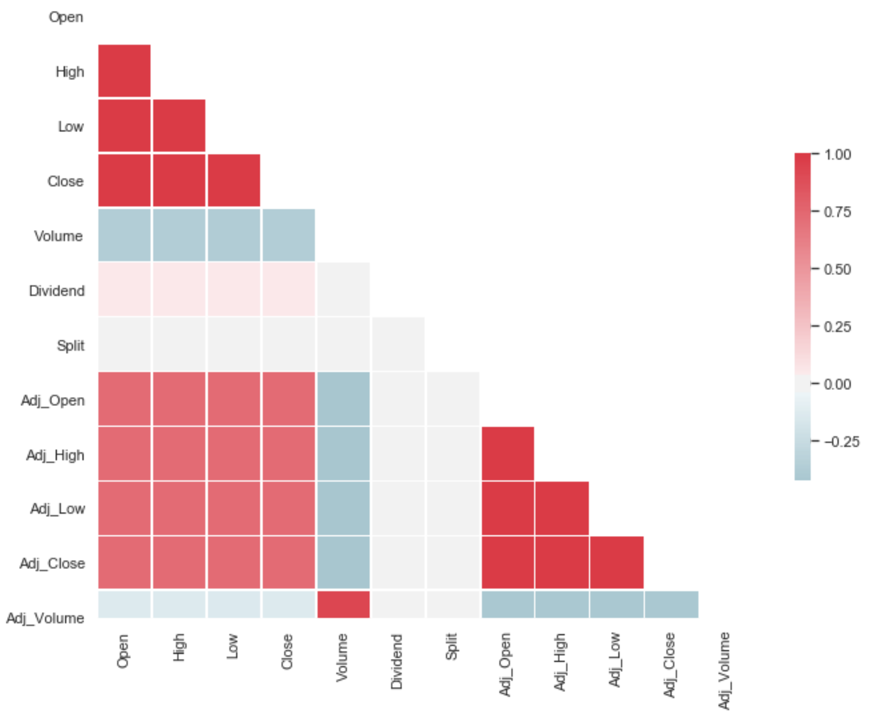
Here, all the time-series are plotted by feature. It is noticeable how the split on a few companies, most relevantly Apple (AAPL, blue) and Chevron Corporation (CVX, brown), affect the stock price and the dividend.


Adj_Close
Adj_High
Adj_Low
Adj_Open
Adj_Volume
1e8
Close
Dividend
High
Low
Open
2013-07
2014-01
2014-07
2015-01
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01



Features correlation

Here we plot the correlation matrix of our dataset. It is noticeable how prices are strongly correlated (we would not expect anything much different within a day) and how volume shows the opposite trend with price.



Methodology

Data Pre-processing

Pre-processing steps consist of a series of steps, applied per each ticker series in the dataset:

- **feature engineering**: adding technical indicators to enhance the input data.
- **windowing**: as the data comes in the form of a time series, it is required to be stored as an time-related array of features.
- **closing price projection**: we extract the adjusted closing price for each window, at 1, 7, 14 and 28 days.

- **scaling (standardization)**: whereas stock prices and dividends are of similar nature (currency), volume and split occur at a different unit.
  NOTE: the scaling is applied at the whole dataset, regardless of the ticker.
- **train-test split**: train and test datasets are created in a ratio of 70-30, where the 30 represent the last 30% of the series to avoid look-ahead bias

Technical Indicators for Feature Engineering

A series of indicators are engineered to provide more features to the dataset.

Relative Strength Index (RSI)

The RSI indicates the current and historical strength of a traded stock, based on the closing prices over a recent period. It is calculated by measuring the velocity and magnitude of oscillation, reason why it is classified as a momentum oscillator.

$RSI = 100 - \frac{100}{1+\frac{EMA(U,n)}{EMA(D,n)}}$

where

- EMA is the exponential moving average over a period n
- U is the upward change
- D is the downward change

```
def calculate_RSI(df, window=5):
    # gett al differences and separate positive and negative
    df_diff = df.diff()
    gain, loss = df_diff.copy(), df_diff.copy()
    gain[gain < 0] = 0 # set neg to zero
    loss[loss > 0] = 0 # set pos to zero
    # calc mean gains and losses
    av_gain = gain.ewm(com=window,adjust=False).mean()
    av_loss = loss.ewm(com=window, adjust=False).mean().abs()
    # RSI
    rsi = 100 - 100 / (1 + av_gain / av_loss)
    return rsi
```

Simple Moving Average (SMA)

The SMA is the arithmetic average calculated over a period

```
def calculate_SMA(df, window=5):
    # Simple Moving Average
    SMA = df.rolling(window=window, min_periods=window, center=False).mean()
    return SMA
```

Bollinger Bands (BB)

The Bollinger Bands characterise the volatility of a financial instrument or commodity by defining its likely upper and lower bounds. The measure is based on the SMA and the standard deviation $K\sigma$ over a window period N, typically 20 data points. K is typically 2.

```
def calculate_BB(df, window=5):
    # Bollinger Bands @ mean-/+2*st_dev
    # calc st_dev and mean
    STD = df.rolling(window=window,min_periods=window, center=False).std()
    SMA = calculate_SMA(df)
    upper_band = pd.DataFrame(SMA.values + (2 * STD.values),columns=df.columns)
    lower_band = pd.DataFrame(SMA.values - (2 * STD.values),columns=df.columns)
    return upper_band,lower_band
```

Windowing

The predict the closing price of a stock is required at 1,7,14 and 28 days ahead in time. To do so, the prediction is based on w previous days, the period window. Since our dataset is is made of end-of-day data, this corresponds to the number of previous days the prediction is based on.

Moreover, the process is executed on each company registered in the initial dataset.

Pseudocode:

```
def preprocess_data(data, window=8):
    # output variables
    X, y, tickers, dates = [],[],[],[]

    # Set a Technical Indicators window
    TI_window = window

    # Group by company
    groups = data.groupby('Tickers')

    for name,group in groups:
        # Technical Indicators
        group = group + group['Adj_Close']-group['Adj_Open'])/group['Adj_Open'] # returns
        group = group + calculate_RSI(group[prices],window=TI_window)
        group = group + calculate_SMA(group[prices],window=TI_window)
        group = group + calculate_BB(group[prices],window=TI_window)
        # calculate the start
        # jump ahead of the window and the technical_window
        start = (window-1)+(TI_window-1)
        for i in range(start, len(group)-28):
            # drop and transform X
            data = group[i-(window-1):i+1]
            # transform y - remember to increase dimensionality of the array
            pred = [
                group[i+1,y_label],
                group[i+7,y_label],
                group[i+14,y_label],
                group[i+28,y_label]
                ]
            # done
            X.append(data)
            y.append(pred)
            tickers.append(name)
            dates.append(group[i]['Date'])
    return X, y, tickers, dates
```

Here we report the dataset shapes in case of a window of 8 days (and so the technical window):

S	y	tickers	dates
(30384, 304)	(30384, 4)	(30384,)	(30384,)

Scaling

Here we scale the whole dataset through a standard scaling.

```
ss = StandardScaler()
X_ss = ss.fit_transform(X)
```

Train-Test and Validation split

The split is performed to avoid look-ahead bias. Therefore, for each company, the data is split into two dataset accordingly to two periods in time:

- First period (training and test),will be used to train the model by cross-validation.
- A following period (validation), which is a part of the data the model has never seen and therefore needs to be the ending part of the dataset.

Here we report an example for `'AAPL'` stock dataset with a training ratio of 0.7:

X_train	X_validation	y_train	y_validation
(728, 304)	(312, 304)	(728, 4)	(312, 4)

Implementation

A standard process is developed in order to process a object which implements `.fit(X,y)` method through a series of steps; it can be a model, grid-search or pipeline.

- The whole dataset is split between train and validation sets, by company.
- The object is fit on all the re-combined training data.
- A final validation is conducted against the whole validation set. Performance data is stored relatively to the metric chosen, the RMSE.

We run this process in three instances:

1. **Benchmark Model**: a decision tree regressor with scikit-learn standard parameters is used as a benchmark model.
2. **Model Testing**: a series of selected models (see below): models are fit and evaluated
3. **Refinements**: pipelines to improve selected models through grid-search are used to perform cross-validation over the time series.

The objective output is a 4-class vector which represent the closing stock price at 1,7,14 and 28 days.

Eventually, 4 different models which handle 1-d output might be trained for the scope. Here, Grid-Search is not performed yet, as it might lead to a data leak due to the previous scaling. Mean squared error is used to evaluate the model performances on the training and test sets.

The first run produces a model whose performances show a clear overfitting (MSE on training is about 6 times lower than the MSE on the test set).

Here, the pseudocode for the above-mentioned processes is laid down. For the extensive code and docstrings, you might want to refer to the project repository.

```
def fit_on_all_data(mdl, X, y, train_percentage=0.7):
    """ PSEUDOCODE
    """
    X_t, X_v, _, _, _ = gen_train_test_and_valid_data(unique_tickers[0], X, y, train_percentage)
    xt = np.array(X_t)
    yt = np.array(y_t)
    xv = np.array(X_v)
    yv = np.array(y_v)
    for t in unique_tickers[1:]:
        X_t, X_v, _, _, _ = gen_train_test_and_valid_data(t, X, y,0.7)
        xt = np.concatenate((xt,X_t))
        yt = np.concatenate((yt,y_t))
    mdl.fit(xt, yt)
    return mdl
```

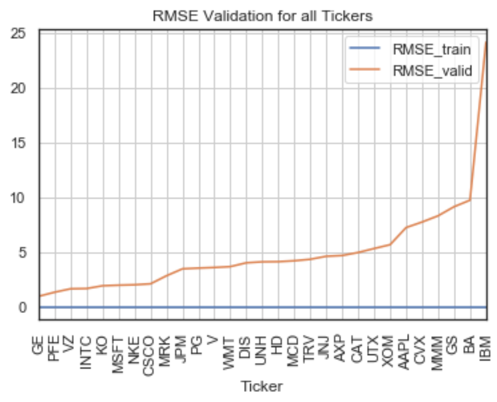
```
def validate_model(model, X, y, ticker, plot=True, print_output=True, ts_split=0.7):
    """ PSEUDOCODE
    """
    X_train, X_valid, y_train, y_valid, dates = gen_train_test_and_valid_data(ticker, X, y, ts_split=ts_split)
    # predictions
    y_pred_train = model.predict(X_train)
    y_pred_valid = model.predict(X_valid)
    # performances - scale inverse
    rmse_train = np.sqrt(mean_squared_error(y_train, y_pred_train))
    rmse_valid = np.sqrt(mean_squared_error(y_valid, y_pred_valid))
    if print_output:
        print('{:=^50}'.format("VALIDATION FOR "+ticker))
        print("TRAIN RMSE:    ",rmse_train)
        print("VALID RMSE:    ",rmse_valid)

    if plot:
        fig,axs = plt.subplots(2,2,figsize=(16,8))
        days=[1,7,14,28]
        for i in range(len(axs)):
            for j in range(len(axs[i])):
                idx = j + i*(len(axs[i]))
                y_1=y_pred_valid.T[idx]
                y_2=y_valid.T[idx]
                x=np.arange(y_1.shape[0])
                axs[i][j].plot(x,y_1,label='pred')
                axs[i][j].plot(x,y_2,label='truth')
                axs[i][j].set_title('Adjusted Close Price @ {} days'.format(days[idx]))
                axs[i][j].legend()
        plt.show();
    return rmse_train, rmse_valid
```

```
def validate_on_all_data(model, X, y, ts_split=0.7):
    """ PSEUDOCODE
    """
    results = []
    for t in unique_tickers:
        r1,r2 = validate_model(model, X, y, t, plot=plot_singles)
        results.append([r1,r2])
    results = results.transpose()
    # create a df for results and plot from most to least predictable ticker
    res = pd.DataFrame({'Ticker':unique_tickers,
                        'RMSE_train':results[0],
                        'RMSE_valid':results[1]})
    res = res.sort_values(by='RMSE_valid')
    return res
```

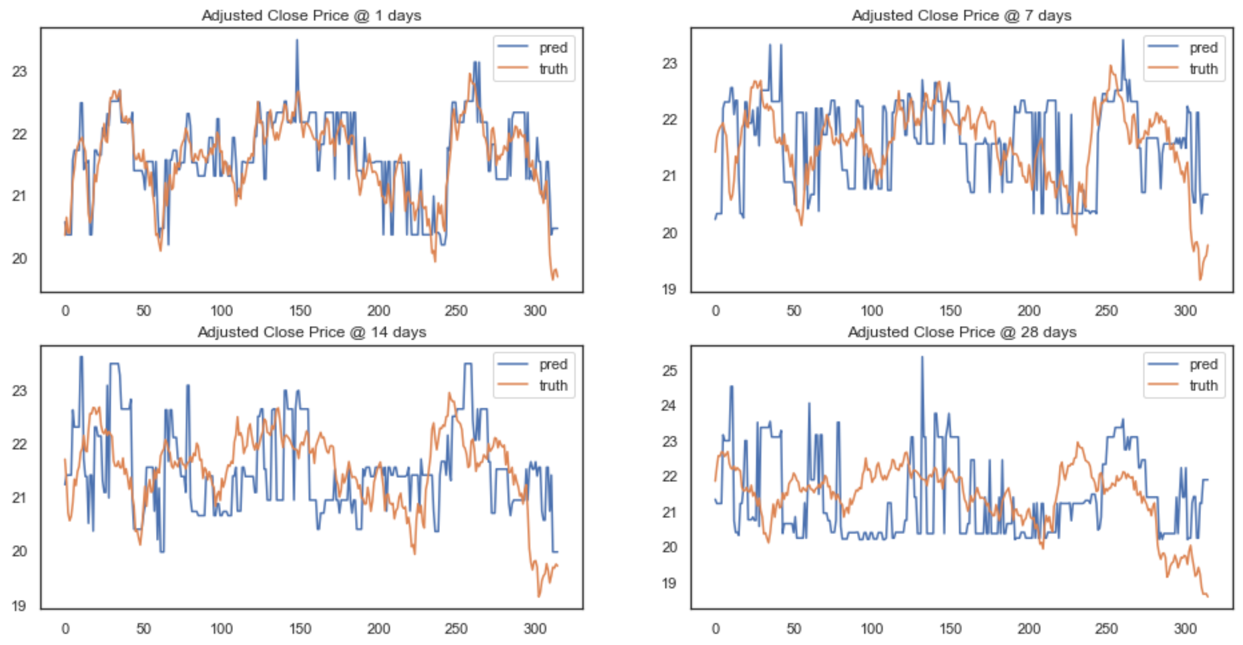
Benchmark Model

Scikit-learn's `DecisionTreeRegressor` with default parameters is used as the benchmark model for this project. Results are reported in table A.2 in the appendix and plotted here:

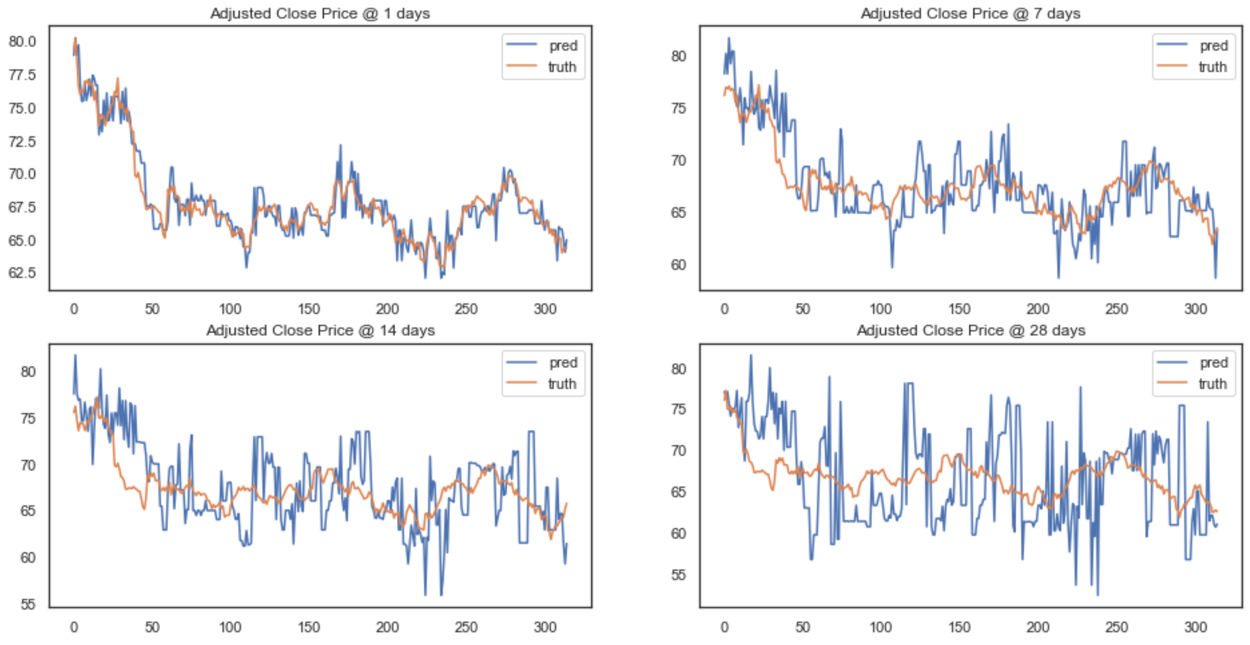


	Ticker	RMSE training	RMSE validation
Low Loss	GE	0.0	0.939871
Avg Loss	WMT	0.0	3.649151
High Loss	IBM	0.0	24.124613

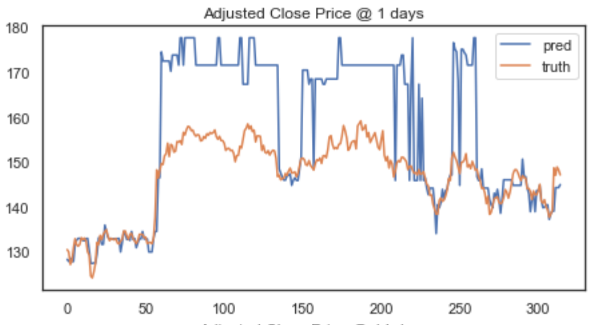
GE (lowest loss)

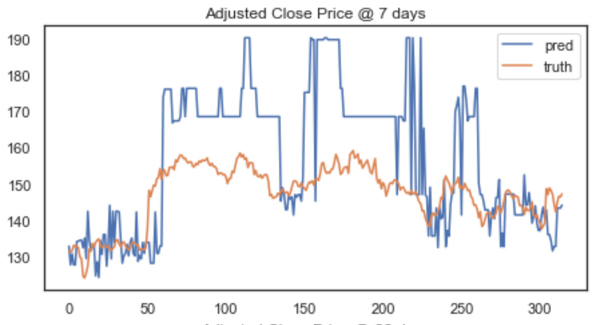


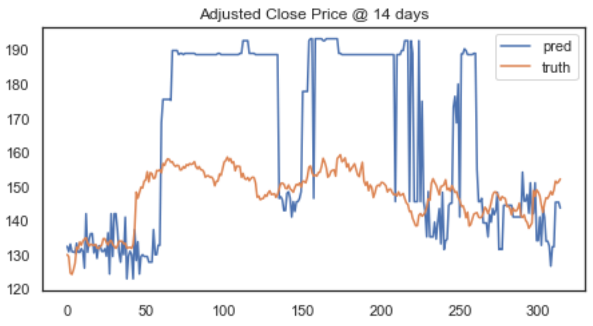
WMT (median loss)

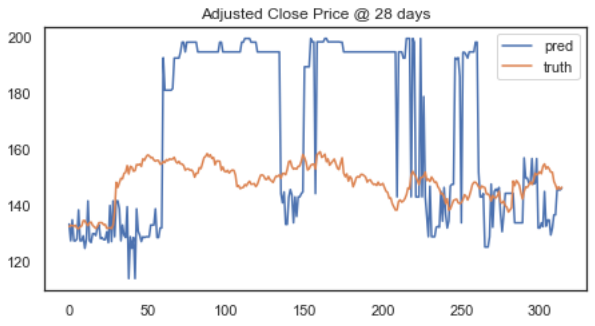


IBM (highest loss)









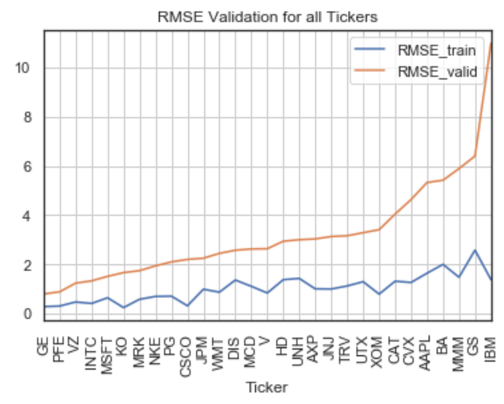
Models testing

Here we test a series of models and we try to improve them by applying grid search.

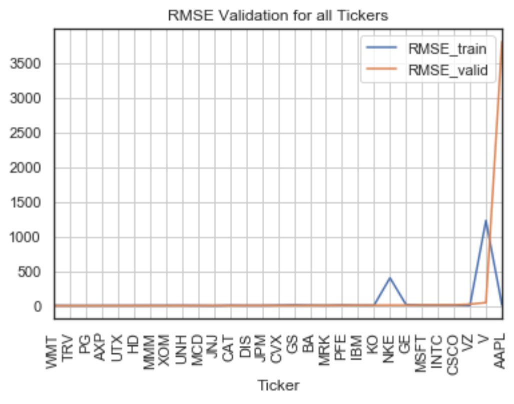
1. Random Forest Regressor
2. Stochastic Gradient Descent (multi-output)
3. Support Vector Regressor (multi-output)
4. Multi-output Lasso
5. Ridge

Models performances are reported in detail below. Details are reported in the appendix table A.3.

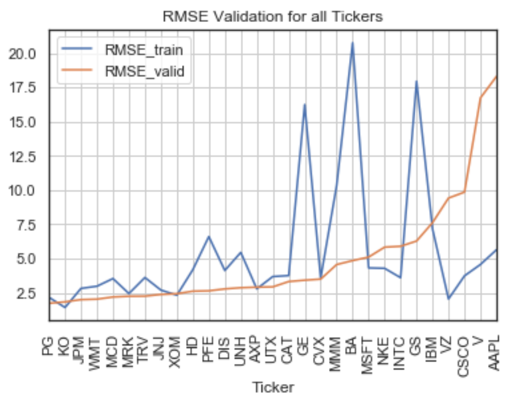
1. Random Forest Regressor



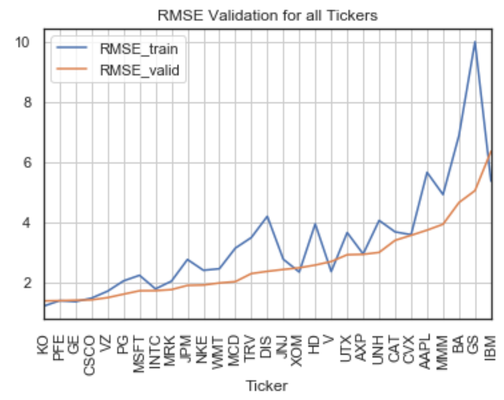
2. Multi-Output Stochastic Gradient Descent



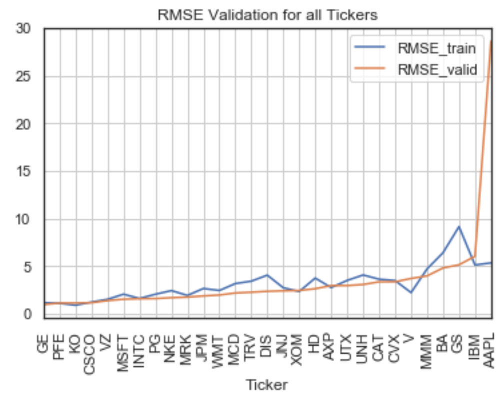
3. Multi-Output Support Vector Regressor



4. Multi-Task Lasso



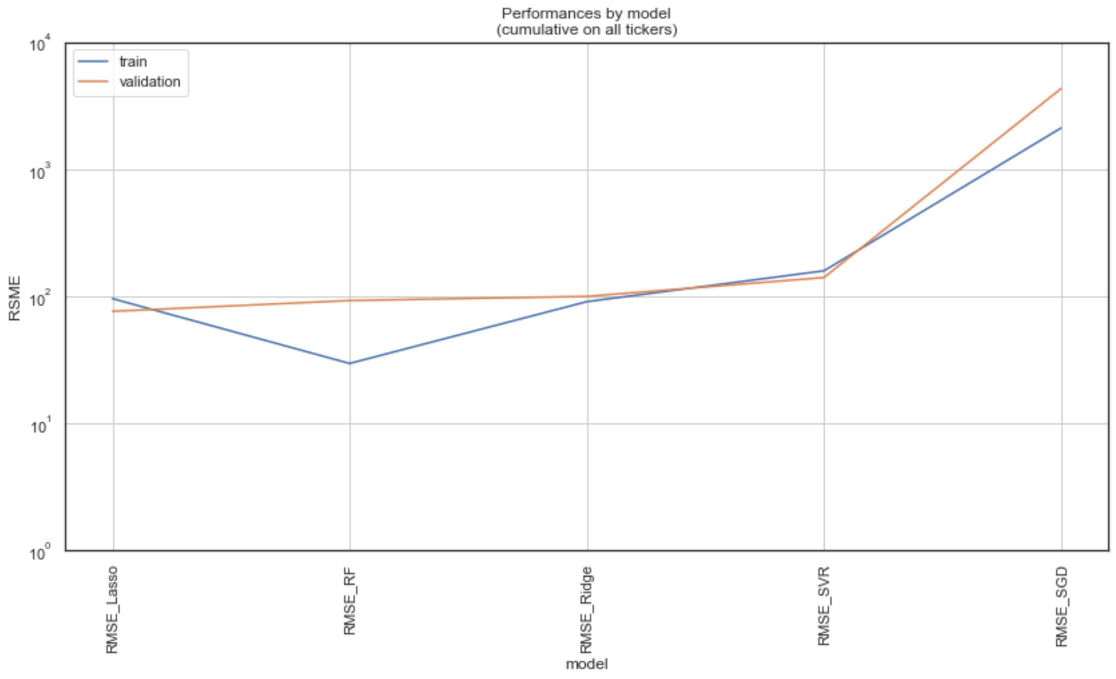
5. Ridge



Overall Validation Loss

The cumulative validation loss is calculated for each model across all the tickers as shown below. Lasso, Random Forest and Ridge perform similarly, with a clear tendency of the second to overfit. The Support Vector Regressor is still comparable, albeit reporting a performance 50% worse. Finally, the SGD reports extremely poor results.

	training RMSE	validation RMSE
Lasso (4)	96.00	76.28
Random Forest (1)	29.66	92.71
Ridge (5)	90.90	99.99
SVR (3)	158.90	140.63
SGD (2)	2115.57	4317.60



Model Testing - Conclusion

Model testing suggest that further development might be worth for models 4, 1 and 5; especially model 1 might improve its performance by fine tuning its hyper-parameters (e.g. n_estimators, max_depth, etc).

For limitation in training time (here not reported), as well as in poorer results, model 3 will be discarded. Moreover, model 2 is also discarded for poor results.

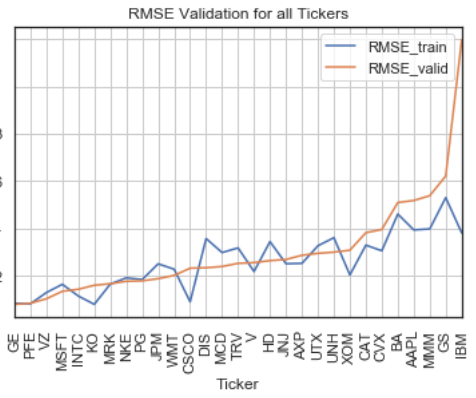
Refinements

To avoid overfitting, I repeat what above, but by building a pipeline object which comprehends a scaler to then perform a grid search to tune the hyper-parameters. Therefore, the windowing is done again, but omitting the scaling. Here we have a list of the models trained as well as the parameters to search for.

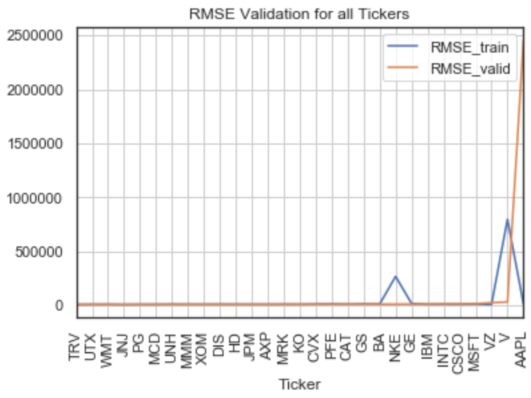
```
p1 = Pipeline([
    ('scaler', StandardScaler()),
    ('reg', RandomForestRegressor())
])
s1 = {
    'reg__n_estimators':[10,15],
    'reg__max_depth':[4,8,16],
    'reg__max_features':['auto','sqrt','log2']
}
p2 = Pipeline([
    ('scaler', StandardScaler()),
    ('reg', MultiOutputRegressor(SGDRegressor()))
])
s2 = {
    'reg__estimator__max_iter':[1000,2000],
    'reg__estimator__epsilon': [0.5,0.1,0.02],
}
p3 = Pipeline([
    ('scaler', StandardScaler()),
    ('reg', MultiOutputRegressor(SVR(gamma='auto')))
])
s3 = {
    'reg__estimator__C': [0.5,1.0],
    'reg__estimator__epsilon': [0.3,0.1],
}
p3.get_params()
```

The Random Forest reports a more balanced score between validation and training set, and so it does the Stochastic Gradient Descent. The Multi-Output Support Vector Regressor improves generally of a small amount. Values are reported in table A.4 in the appendix.

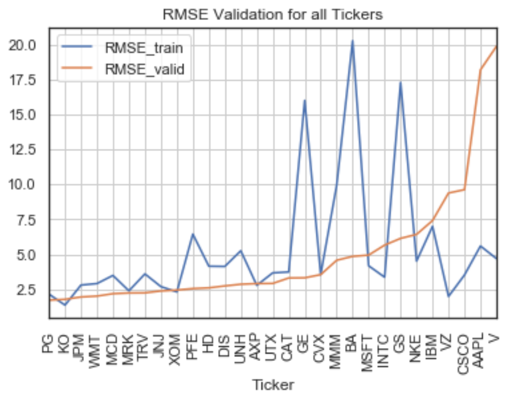
Random Forest Regressor



SGD



Multi-Output Support Vector Regressor



Results

Model Evaluation and Validation

The model is evaluated against its ability to predict returns. The returns at 1,7,14 and 28 days are calculated for the truth and the predicted prices. Then, their difference is analysed in order to show the error. More specifically:

- The mean of the difference shows if our model has the tendency to produce pessimistic/optimistic prices in the case of a predominant negative or positive error.
- The standard deviation of the difference shows how confident we can be about the model's outcome. It appears that the model requires outstanding improvements before being taken seriously, as we discuss the results below.

Data Frame for Returns

Here, we store the returns into a data frame, reporting truth, predicted and difference (error) values.

	truth - 1 day	truth - 7 days	truth - 14 days	truth - 28 days	predicted - 1 day	predicted - 7 day	predicted - 14 day	predicted - 28 day	error - 1 day	error - 7 days	error - 14 days	erro
0	0.212436	-0.861228	-1.332032	0.618887	-0.727841	-2.320279	-3.770112	-6.087364	-0.940277	-1.459051	-2.438080	-6.70
1	0.886373	-3.145194	-3.082290	0.421240	-0.735464	-2.458878	-3.753820	-5.201164	-1.621837	0.686316	-0.671530	-5.62
2	-0.605610	-3.262298	-1.119808	-0.484625	-1.190879	-2.911703	-4.182123	-6.119410	-0.585269	0.350595	-3.062315	-5.63
3	-1.521043	-3.356587	-0.451738	-0.718336	-0.671276	-2.820702	-4.447407	-6.382866	0.849767	0.535885	-3.995668	-5.66
4	-0.783275	-2.297990	0.771756	-1.007854	-0.635755	-2.910702	-4.789407	-7.153317	0.147520	-0.612712	-5.561163	-6.14

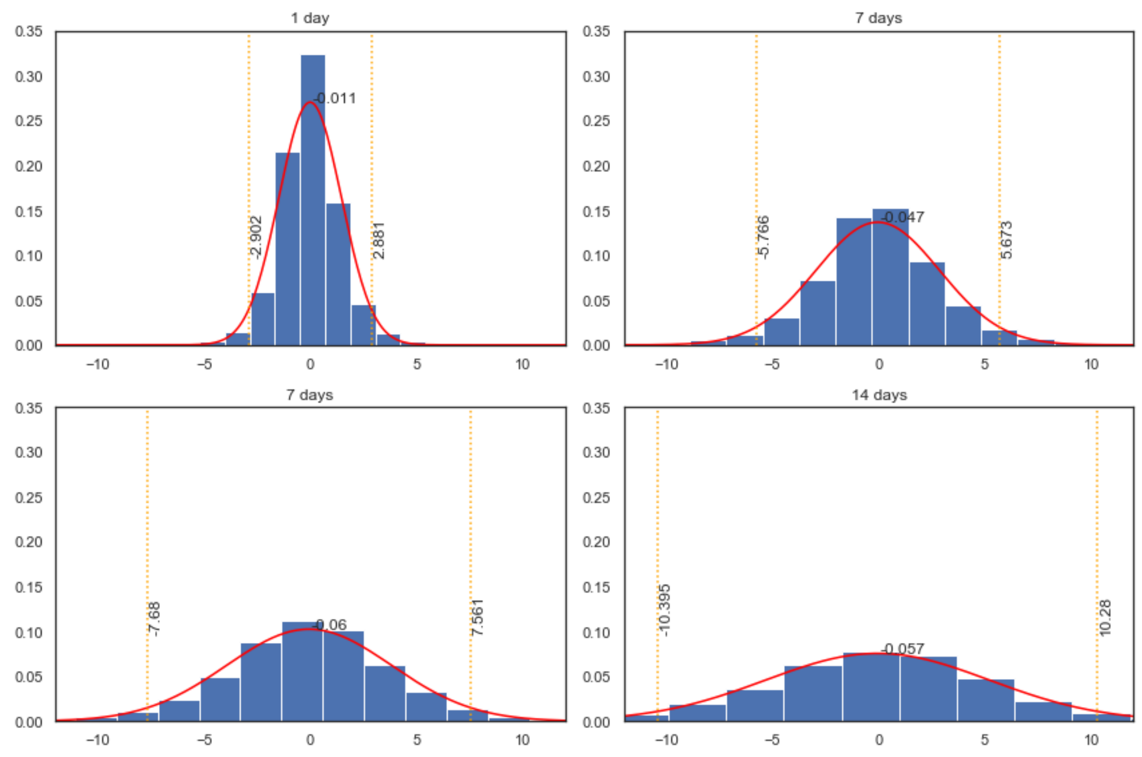
	truth - 1 day	truth - 7 days	truth - 14 days	truth - 28 days	predicted - 1 day	predicted - 7 day	predicted - 14 day	predicted - 28 day	error - 1
count	30653.000000	30653.000000	30653.000000	30653.000000	30653.000000	30653.000000	30653.000000	30653.000000	30653.000
mean	-0.064943	-0.349022	-0.660375	-1.253191	-0.075783	-0.395773	-0.720014	-1.310647	-0.010840
std	1.390063	3.087831	4.229838	5.919779	0.757015	1.101838	1.511708	2.282970	1.475334
min	-10.403576	-15.705247	-18.640535	-27.767035	-3.191394	-9.492615	-13.092403	-18.420814	-12.30225
25%	-0.749415	-2.114191	-3.228079	-5.100384	-0.550256	-1.030045	-1.554728	-2.515277	-0.831855
50%	-0.082707	-0.439099	-0.825059	-1.607244	-0.119782	-0.482586	-0.853464	-1.451513	-0.050504
75%	0.583501	1.263256	1.706667	2.224064	0.346963	0.162219	0.001345	-0.267282	0.767735
max	18.741436	21.471487	29.706575	40.148487	13.115996	14.559105	17.132684	26.654802	11.329396

Justification

Here, the final results are discussed in detail. We focus on the difference between predicted and true returns. As previously stated, we look at its mean to evaluate if our model has the tendendcy to produce pessimistic/optimistic prices in the case of a predominant negative or positive error. Its standard deviation helps us in calculating the confidence interval of the produced prediction.

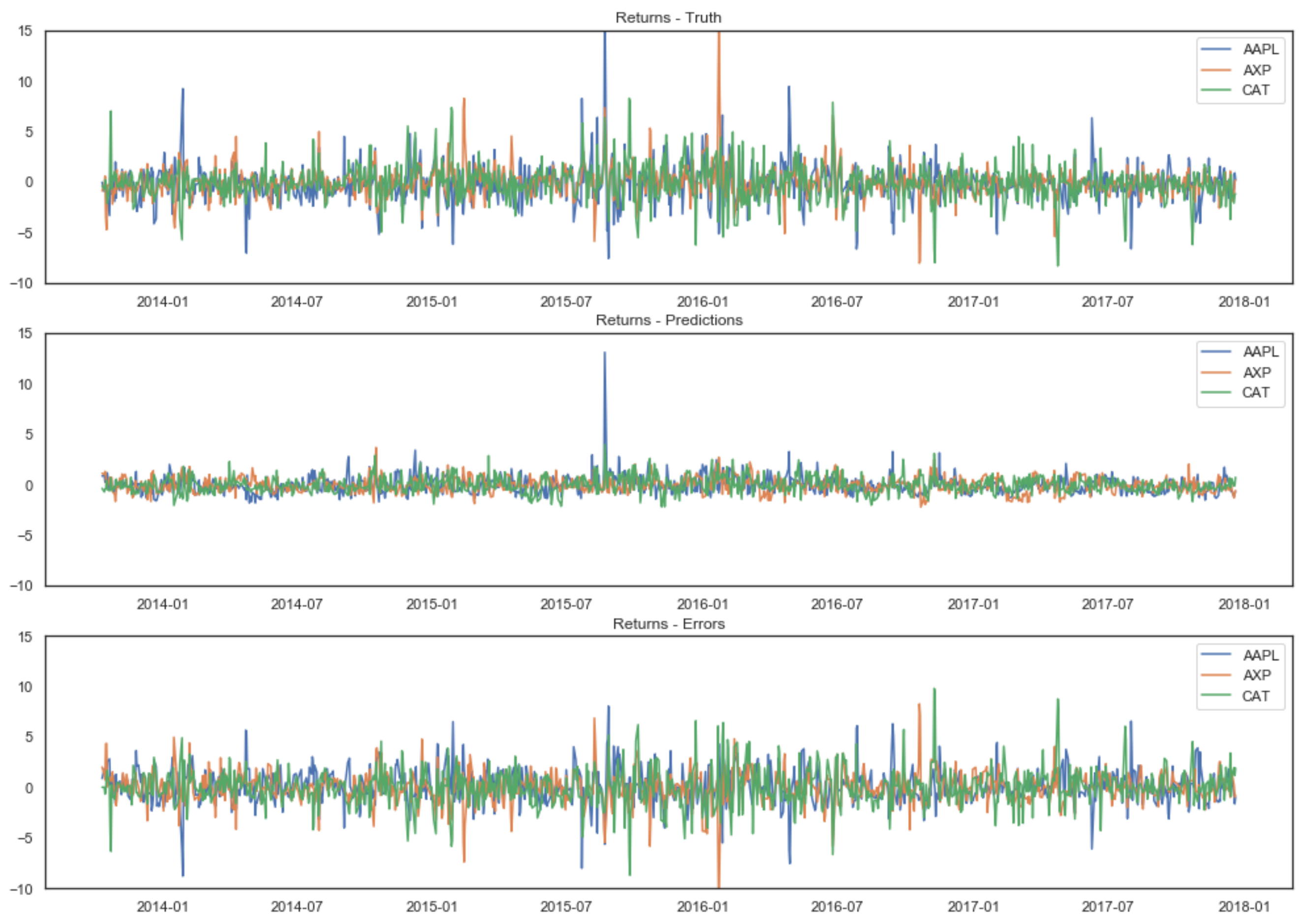
Mean and 95% confidence intervals are plotted below. The error at 1 day shows how the return might be wrong by a nominal -3/+3%. This is quite a lot, so to have an useful guess, we should look for a predicted outcome >3% (albeit further considerations are required). This makes the model being hardly useful as increments >3% in one day refers to quite specific events. A behavioural approach to investment might be considered in such case, however, the model could help in supporting that.

Then, the confidence intervals at 7, 14 and 28 days increase steadily by a further 30% for each days, making the model quite ineffective. We want also to plot a few examples about the returns, to see if it is possible to notice any patterns.



Here, returns are plotted for 3 companies. The model produces much stable outputs compared to the real returns.

```
show_returns(['AAPL','AXP','CAT'],0)
```



Conclusion

Reflection

The end-to-end product of this project tries to bring together historical data about some of the S&P500 companies, where available through Quandl API. The data did not present inconsistencies, nor missing values, requiring only standard scaling practice before proceeding through picking a model to train and test.

The selection fell on scikit-learn's random forest regressor. An initial overfitting happened due to lack of tuning hyper parameters, corrected by performing a grid search, and training the model through cross-validation (this was possible by creating a pipeline in order to avoid data leaks on cross validation).

Results were tested against analysing rate on returns, or RoR, and so by comparing the predicted with true returns. By doing so, it was possible to define a confidence interval relative to each time range. 7-days returns shows a 95% confidence interval of about -/+7% which is above the project's guidelines. Problems like reliable stock predictors requires a continuous implementation to keep the system effective by constantly digesting last-day data. In fact, by far the result has to be considered good in real-world business scenario.

Developing knowledge of time series as well as of investment principles and returns calculation was one of the most interesting aspect of the project.

Improvement

The nature of the problem, as well as the data here digested presents a high degree of non-linearity. Improvements to the system can be made by choosing a different model (e.g. SVM regressor to be trained on each time range) which is more prone to capture highly non-linear patterns. Eventually, going for a deep neural network might help: nlp-oriented techniques such as RNN or LSTM might be used to also include the temporality of the data (here, the window is digested as a flat vector by the model).

Then, further feature engineering might be required. For instance, returns might be included. Moreover, company-specific indices might help in determining the volatility of the company, or their financial status such as quarterly reports data. Moreover, such data should be daily added by undertaking incremental learning strategies in order to keep the model up-to-date.

Finally, test of the window size might be run in order to see how this affect our model. However, to avoid increasing the dimensionality of data too much without providing a time-related approach to train our model (such in the case of a LSTM), might produce worse rather than better results.

Appendix

Table A.1 - Time frame for each company in the dataset.

	start date	end date	difference
Ticker			
AAPL	2013-09-03	2017-12-28	1577 days
AXP	2013-09-03	2017-12-28	1577 days
BA	2013-09-03	2017-12-28	1577 days
CAT	2013-09-03	2017-12-28	1577 days
CSCO	2013-09-03	2017-12-28	1577 days
CVX	2013-09-03	2017-12-28	1577 days
DIS	2013-09-03	2017-12-28	1577 days
GE	2013-09-03	2017-12-28	1577 days
GS	2013-09-03	2017-12-28	1577 days
HD	2013-09-03	2017-12-28	1577 days
IBM	2013-09-03	2017-12-28	1577 days
INTC	2013-09-03	2017-12-28	1577 days
JNJ	2013-09-03	2017-12-28	1577 days
JPM	2013-09-03	2017-12-28	1577 days
KO	2013-09-03	2017-12-28	1577 days
MCD	2013-09-03	2017-12-28	1577 days
MMM	2013-09-03	2017-12-28	1577 days
MRK	2013-09-03	2017-12-28	1577 days
MSFT	2013-09-03	2017-12-28	1577 days
NKE	2013-09-03	2017-12-28	1577 days
PFE	2013-09-03	2017-12-28	1577 days
PG	2013-09-03	2017-12-28	1577 days
TRV	2013-09-03	2017-12-28	1577 days
UNH	2013-09-03	2017-12-28	1577 days
UTX	2013-09-03	2017-12-28	1577 days
V	2013-09-03	2017-12-28	1577 days
VZ	2013-09-03	2017-12-28	1577 days
WMT	2013-09-03	2017-12-28	1577 days
XOM	2013-09-03	2017-12-28	1577 days

Table A.2 - Benchmark model, training and validation loss.

	Ticker	RMSE_train	RMSE_valid
0	GE	0.0	0.939871
1	PFE	0.0	1.313880
2	VZ	0.0	1.626120
3	INTC	0.0	1.647747
4	KO	0.0	1.901515
5	MSFT	0.0	1.955191
6	NKE	0.0	1.992813
7	CSCO	0.0	2.077986
8	MRK	0.0	2.813308
9	JPM	0.0	3.446892
10	PG	0.0	3.514552
11	V	0.0	3.571304
12	WMT	0.0	3.649151
13	DIS	0.0	3.998356
14	UNH	0.0	4.085919
15	HD	0.0	4.101042
16	MCD	0.0	4.183108
17	TRV	0.0	4.318340
18	JNJ	0.0	4.591501
19	AXP	0.0	4.661659
20	CAT	0.0	4.938890
21	UTX	0.0	5.300167
22	XOM	0.0	5.650869
23	AAPL	0.0	7.213543
24	CVX	0.0	7.718927
25	MMM	0.0	8.284599
26	GS	0.0	9.113047
27	BA	0.0	9.713179
28	IBM	0.0	24.124613

Table A.3 - Performance model by company

	Ticker	RMSE_train_RF	RMSE_valid_RF	R2_train_RF	R2_valid_RF	RMSE_train_SGD	RMSE_valid_SGD
0	GE	0.289919	0.802033	0.986921	-0.080841	29.097698	20.139643
1	PFE	0.311877	0.890294	0.969730	0.038854	20.878648	17.881947
2	VZ	0.476855	1.241692	0.975719	0.226021	16.641466	33.918939
3	INTC	0.417340	1.334683	0.984059	0.893517	18.275574	22.385433
4	MSFT	0.648251	1.517920	0.996458	0.872842	19.901835	21.842127
5	KO	0.248958	1.670166	0.990425	0.097897	16.307069	19.074644
6	MRK	0.586314	1.748191	0.986352	0.832470	16.265173	16.158193
7	NKE	0.702744	1.944690	0.976167	0.684543	412.382397	19.594390
8	PG	0.713240	2.105242	0.987747	0.584425	14.022727	12.807131
9	CSCO	0.321984	2.204890	0.988218	-1.017451	18.710024	22.913129
10	JPM	0.995210	2.254704	0.994322	0.250294	15.593091	15.223622
11	WMT	0.875419	2.451230	0.985028	0.035381	15.512567	12.009910
12	DIS	1.368465	2.578931	0.960354	0.894225	15.431932	15.183528
13	MCD	1.112751	2.629494	0.997350	-0.041440	14.525599	14.334911
14	V	0.842394	2.639325	0.995043	0.525411	1237.212060	56.907097
15	HD	1.381925	2.945212	0.993554	0.809192	14.936096	13.733586
16	UNH	1.434322	3.004539	0.997661	0.841090	16.143899	14.075137
17	AXP	1.010664	3.037128	0.988497	0.596970	14.500710	13.020648
18	JNJ	1.003333	3.134837	0.994866	0.743088	12.956371	14.540930
19	TRV	1.128007	3.166511	0.986643	0.690005	14.267659	12.412839
20	UTX	1.301854	3.290615	0.983240	0.397708	14.728100	13.039407
21	XOM	0.796383	3.412235	0.966814	0.328122	15.889787	13.846503
22	CAT	1.320597	4.045199	0.993912	0.727267	16.924783	15.156559
23	CVX	1.269768	4.636693	0.986464	0.078017	18.425201	15.937431
24	AAPL	1.647009	5.328984	0.992635	0.815984	22.824478	3807.224971
25	BA	2.004267	5.421338	0.997098	-0.049844	18.576642	16.121349
26	MMM	1.482693	5.892216	0.995665	0.631618	15.502953	13.776391
27	GS	2.583861	6.403071	0.992552	0.585308	22.398443	15.983680
28	IBM	1.386201	10.982032	0.985294	-1.180905	16.737698	18.357442

	Ticker	R2_train_SGD	R2_valid_SGD	RMSE_train_SVR	RMSE_valid_SVR	R2_train_SVR	R2_valid_SVR
0	GE	-128.249774	-768.496704	16.239260	3.422879	-37.767736	-21.394799
1	PFE	-134.177393	-395.136855	6.606216	2.646302	-12.457428	-7.665084
2	VZ	-28.959778	-579.570288	2.051913	9.418686	0.548721	-46.112060
3	INTC	-27.718669	-28.012176	3.605721	5.898453	-0.085832	-1.006241
4	MSFT	-2.284966	-25.608199	4.319354	5.109776	0.846703	-0.470617
5	KO	-39.231050	-115.364163	1.430590	1.840017	0.687213	-0.094655
6	MRK	-9.514151	-13.583071	2.449079	2.249734	0.761804	0.721762
7	NKE	-8231.629426	-28.170341	4.289961	5.834492	0.101956	-1.706765
8	PG	-3.708646	-12.538335	2.169077	1.736501	0.886610	0.718926
9	CSCO	-38.028247	-215.544936	3.740059	9.853425	-0.545208	-36.178052
10	JPM	-0.374313	-33.482260	2.821956	2.005419	0.954283	0.403554
11	WMT	-3.553473	-19.224331	2.983268	2.038102	0.828084	0.344082
12	DIS	-4.115212	-2.765070	4.131485	2.786054	0.639092	0.875830
13	MCD	0.557111	-29.636184	3.541107	2.196009	0.973306	0.274316
14	V	-10656.342840	-175.218926	4.571121	16.733070	0.856735	-16.055202
15	HD	0.251502	-2.851593	4.191661	2.620976	0.940688	0.852988
16	UNH	0.707704	-2.243011	5.451365	2.873713	0.966729	0.855610
17	AXP	-1.350941	-6.747387	2.789784	2.912593	0.912017	0.632336
18	JNJ	0.159644	-4.658235	2.701113	2.381202	0.962753	0.851478
19	TRV	-1.098190	-3.334568	3.617769	2.256673	0.862432	0.844696
20	UTX	-1.122426	-8.467857	3.687257	2.934633	0.865466	0.521060
21	XOM	-12.235037	-10.845563	2.328067	2.436877	0.716505	0.652429
22	CAT	0.043864	-2.935973	3.760055	3.324160	0.950776	0.814950
23	CVX	-1.811698	-9.320279	3.573080	3.501463	0.892799	0.479152
24	AAPL	-0.392964	-91512.823096	5.644207	18.296702	0.913857	-1.075803
25	BA	0.759412	-9.229486	20.751659	4.852942	0.709631	0.137039
26	MMM	0.539068	-0.938751	10.343088	4.564458	0.797080	0.781638
27	GS	0.447075	-1.447089	17.946108	6.281350	0.645811	0.615900
28	IBM	-1.143890	-4.856899	7.162456	7.623836	0.607480	-0.033294

	Ticker	RMSE_train_Lasso	RMSE_valid_Lasso	R2_train_Lasso	R2_valid_Lasso	RMSE_train_Ridge	RMSE_valid_Ridge	R2_train_Ridge
0	GE	1.368410	1.415377	0.711794	-2.370019	1.163687	0.963598	0.787068
1	PFE	1.399961	1.400253	0.390187	-1.381807	1.096682	1.122971	0.624983
2	VZ	1.714482	1.500831	0.687086	-0.134674	1.515958	1.380802	0.755660
3	INTC	1.790876	1.731657	0.701982	0.821093	1.605749	1.578353	0.757016
4	MSFT	2.242117	1.728360	0.957323	0.834949	2.054051	1.522417	0.964206
5	KO	1.219757	1.392964	0.770268	0.371315	0.895905	1.132836	0.875354
6	MRK	2.048008	1.767645	0.833539	0.829697	1.924847	1.746910	0.852969
7	NKE	2.408349	1.919028	0.721332	0.686035	2.429223	1.686951	0.716796
8	PG	2.056726	1.616347	0.898006	0.755139	2.050876	1.593699	0.898520
9	CSCO	1.484792	1.426805	0.747446	0.135378	1.226155	1.157866	0.826928
10	JPM	2.766181	1.906850	0.955975	0.462026	2.653034	1.870894	0.959501
11	WMT	2.459952	1.987045	0.880778	0.379838	2.443884	1.966791	0.882271
12	DIS	4.199593	2.373141	0.627524	0.910179	4.054449	2.356654	0.653363
13	MCD	3.144105	2.031617	0.978811	0.379179	3.170037	2.183299	0.978452
14	V	2.366259	2.698331	0.960511	0.501541	2.220060	3.705079	0.965287
15	HD	3.948196	2.581845	0.947274	0.857124	3.749325	2.629726	0.952456
16	UNH	4.062312	3.002023	0.981204	0.842432	4.070180	3.073332	0.981134
17	AXP	2.948339	2.937734	0.901515	0.624926	2.744965	2.955127	0.914732
18	JNJ	2.774307	2.435754	0.960709	0.844519	2.727892	2.410350	0.962019
19	TRV	3.495479	2.299511	0.871327	0.838830	3.422146	2.261065	0.876600
20	UTX	3.658819	2.924552	0.867440	0.524382	3.502221	2.960691	0.878574
21	XOM	2.350650	2.490362	0.710990	0.639838	2.338461	2.443388	0.713997
22	CAT	3.682471	3.402514	0.952483	0.806557	3.627174	3.351792	0.953839
23	CVX	3.588996	3.568934	0.891500	0.455855	3.483666	3.359145	0.897786
24	AAPL	5.657462	3.740760	0.912303	0.910651	5.342053	28.626782	0.921631
25	BA	6.886914	4.657670	0.965389	0.208516	6.410956	4.809908	0.969922
26	MMM	4.920300	3.935932	0.951561	0.837189	4.708970	3.971910	0.955621
27	GS	9.991158	5.051835	0.888149	0.746767	9.152032	5.139018	0.906011
28	IBM	5.369667	6.355131	0.779302	0.238326	5.120365	6.037869	0.799313