Predicting 7-day Stock Prices

By Olha Maslova

Table of Contents

Project Definition

Project Overview

Pandemic has changed the way we live, the way we work, and the way we make money. According to Devin Ryan, equity research analyst at JMP Securities, more than 10 million new brokerage accounts were opened by individuals in 2020 — more than ever in a year. These days, not only investment firms, hedge funds, and portfolio managers are looking to understand the market to make profitable investments, but regular individuals too.

For this project, I decided to use Yahoo Finance data requested through the web API. The goal is to build a stock price predictor that takes daily trading data over a certain date range as input and outputs projected estimates for the Adjusted Closing Price for the next 7-days.

The final deliverable of a project takes the form of a simple web app running on Streamlit. Users can select a stock from the list of 5 FAANG stocks and see the historical prices and a 7-day forecast.

Problem Statement

Given the historical stock market data, can we forecast the adjusted closing price of a particular stock for the next seven days with reasonable accuracy?

I will attempt to solve this problem in two steps:

1. Developing a forecasting model that produces a 7-day forecast Adjusted Closing Prices with reasonable accuracy.
   Models to be implemented:
   a. Linear model
   b. Polynomial model
   c. K-Nearest Neighbors
   d. CNN
   e. LSTM (single-shot)

f. LSTM (autoregressive)

2. Building a Streamlit web application that will allow users to select a stock and get a 7-day forecast for it.

Metrics

To evaluate the performance of the forecasting models, I decided to use the following metrics:

1. **MAE** - Mean Absolute Error. Since the stock market is highly volatile, I chose, to use MAE as it is less sensitive to outliers and will target the median rather than the mean of a dataset. In other words, it tells us how big of an error we can expect from the forecast on average.

$$MAE = \frac{1}{n} \sum |e_t|$$

2. **MAPE** - Mean Absolute Percentage Error - one of the most commonly used KPIs to measure forecast accuracy. It is the average of the percentage errors. MAPE is very easy to interpret, however, it is extremely sensitive to outliers and can produce skewed results.

$$MAPE = \frac{1}{n} \sum_{1}^{n} \left| \frac{e_t}{a_t} \right|$$

3. **RMSE** - Root Mean Square Error. RMSE emphasizes the most significant errors, whereas, for example, MAE gives the same importance to each error. It has the useful property of being in the same units as the response variable. Lower values of RMSE indicate a better fit.

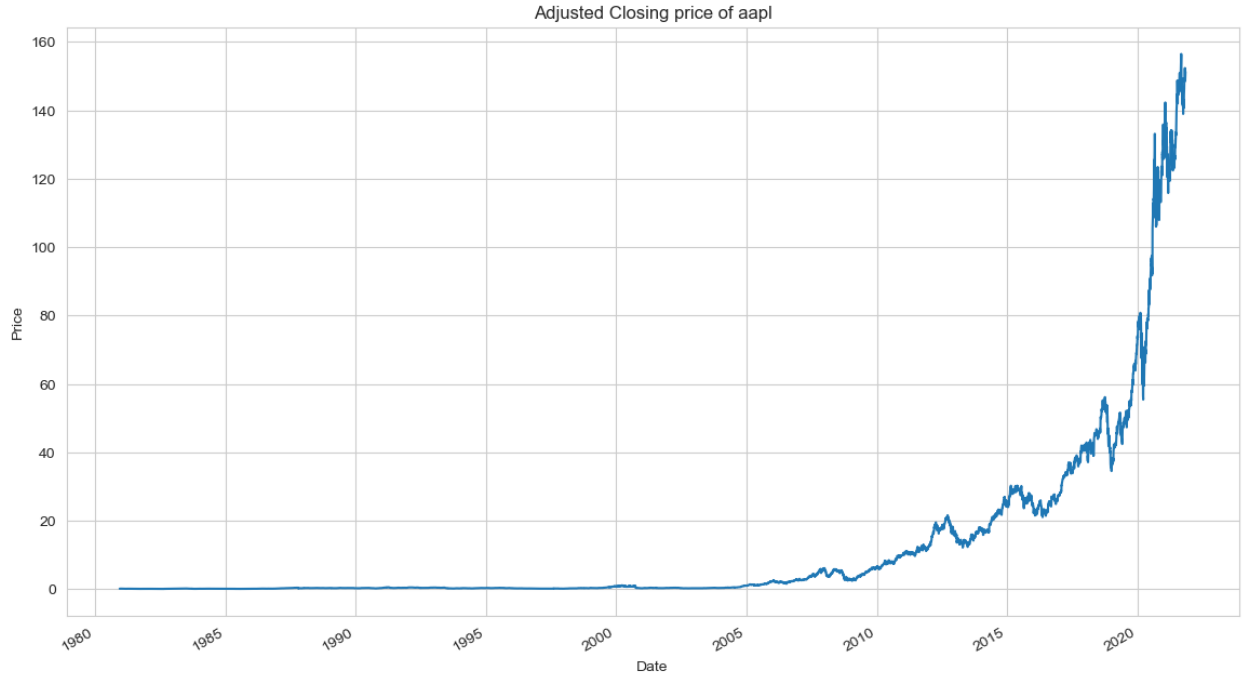
$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{e_t^2}{n}}$$

Analysis

Data Exploration & Data Visualization

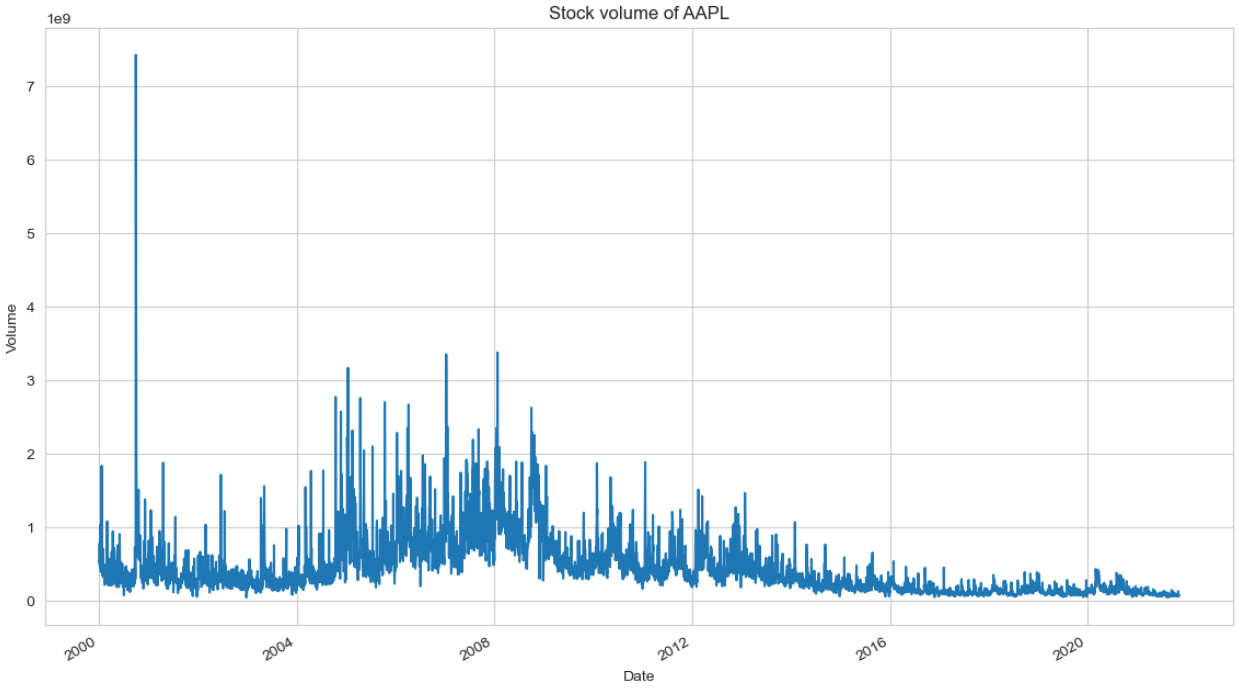
This data exploration and visualization part will be specifically for Apple stock with ticket AAPL. Other FAANG stocks follow a similar process.

The dataset contains over 10K entries. Adjusted closing price (ACP) data is highly skewed to the left. Minimal ACP is 0.04 and maximum price 156.46 while mean is only 11.95.



The original dataset contains no NULL values, which is great because we don't need to come up with various strategies to deal with it.

Next, I wanted to visualize the stock volume. As can be seen on the graph below, it became smaller and less volatile which also means it is much more predictable and stable.


Stock volume of AAPL
1e9
7
6
5
4
3
2
1
0
Volume
2000
2004
2008
2012
2016
2020
Date

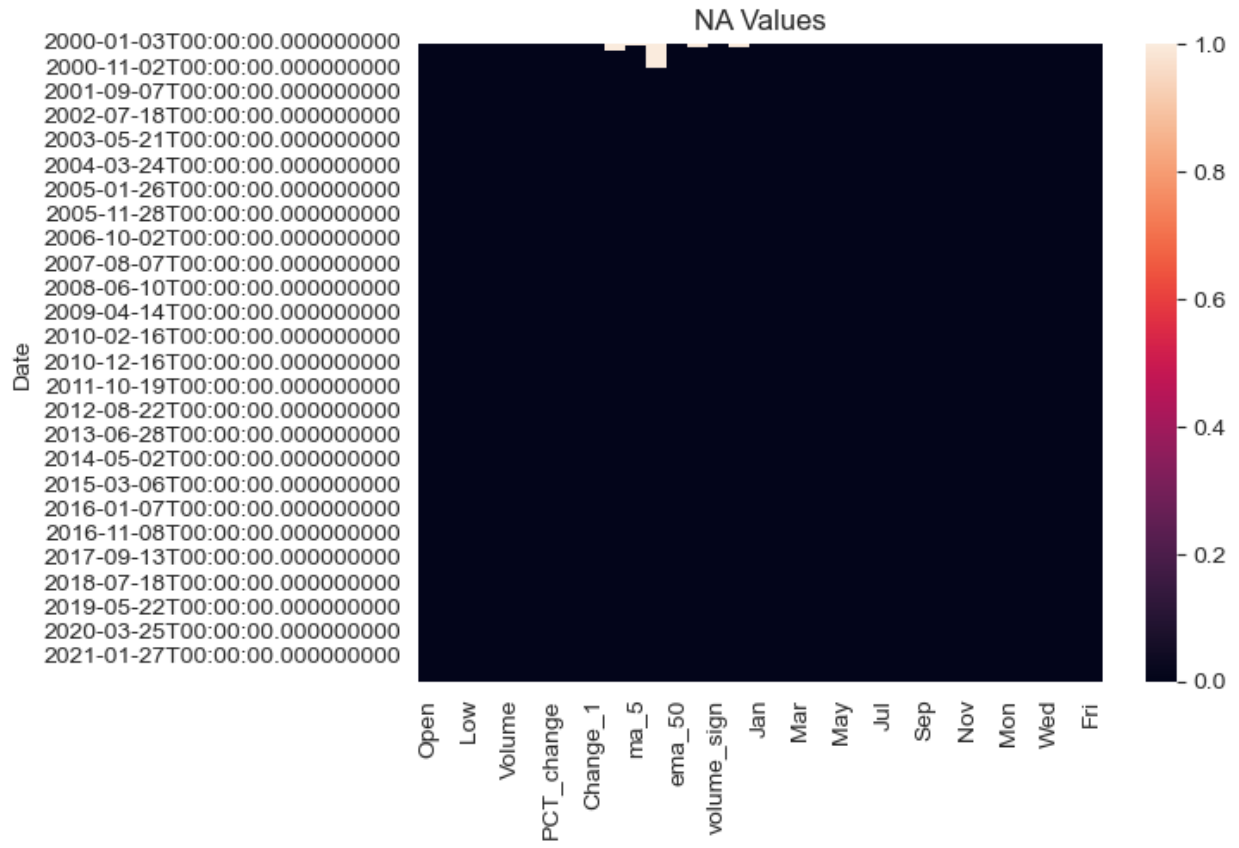
Methodology

Feature Engineering

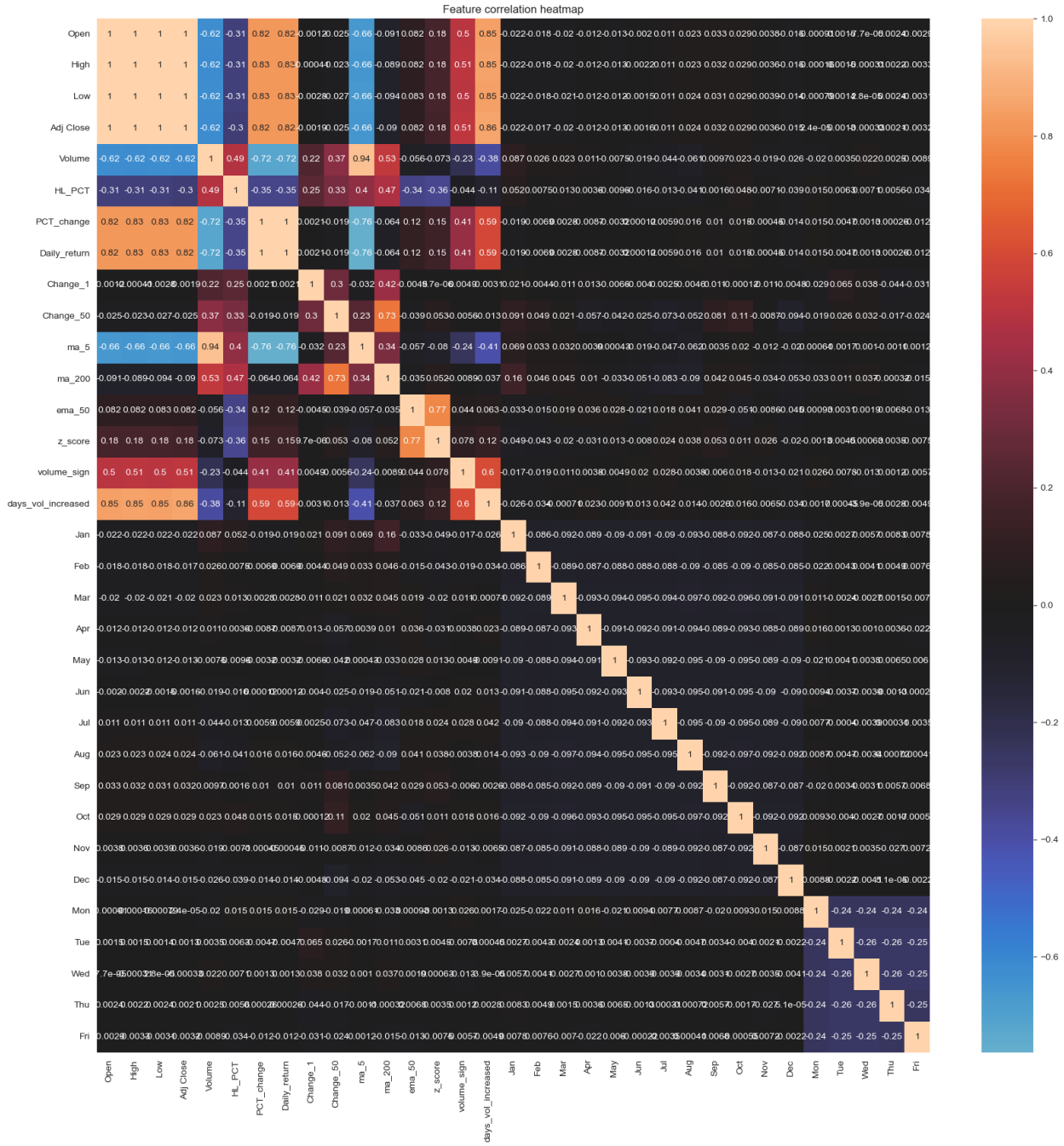
For feature engineering, I followed this tutorial and computed the following features:

- **HL_PCT**: (High - Low) / Adj Close * 100 %
- **PCT_change**: Percentage change from the immediately previous row.
- **Daily_return**: Measures the dollar change in a stock's price as a percentage of the previous day's closing price.
- **Volume:** Logarithmic transformation of volume maps better to the prediction target if put into log space:
- **Change_1**: 1-day differencing. It's often more important to know how a value is changing than to know the value itself.
- **Change_50**: 50-day differencing
- **ma_5**: 5-day moving average
- **ma_200**: 200-day moving average
- **ema_50**: daily closing price vs 50-day exponential moving average
- **Z_score:** A very popular/useful transformation for financial time series data is the z-score. It indicates how many standard deviations an element is from the mean.
- **Volume_sign:** Was volume increasing or decreasing today?
- **days_vol_increased**: How many days in a row a volume has increased?
- Dummy variables for **months** (12 additional features)
- Dummy variables for **weekdays** (5 additional features)

Next, I had to drop rows that contained NA values. These values appeared during the feature engineering process when computing rolling averages.



Finally, I visualized the correlation between all the columns. As can be seen on the heatmap below, there are multiple highly correlated features. Before we proceed further, we need to drop some of them to avoid multicollinearity: Open, Low, High, z_score, Volume, Daily_return.

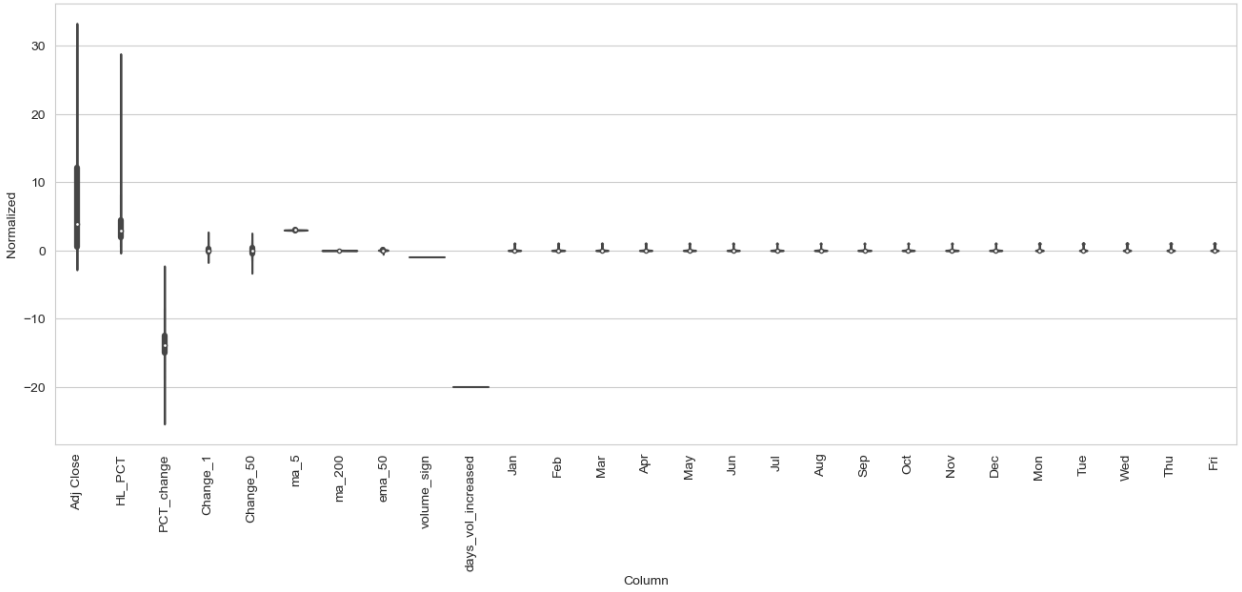


Data Preprocessing

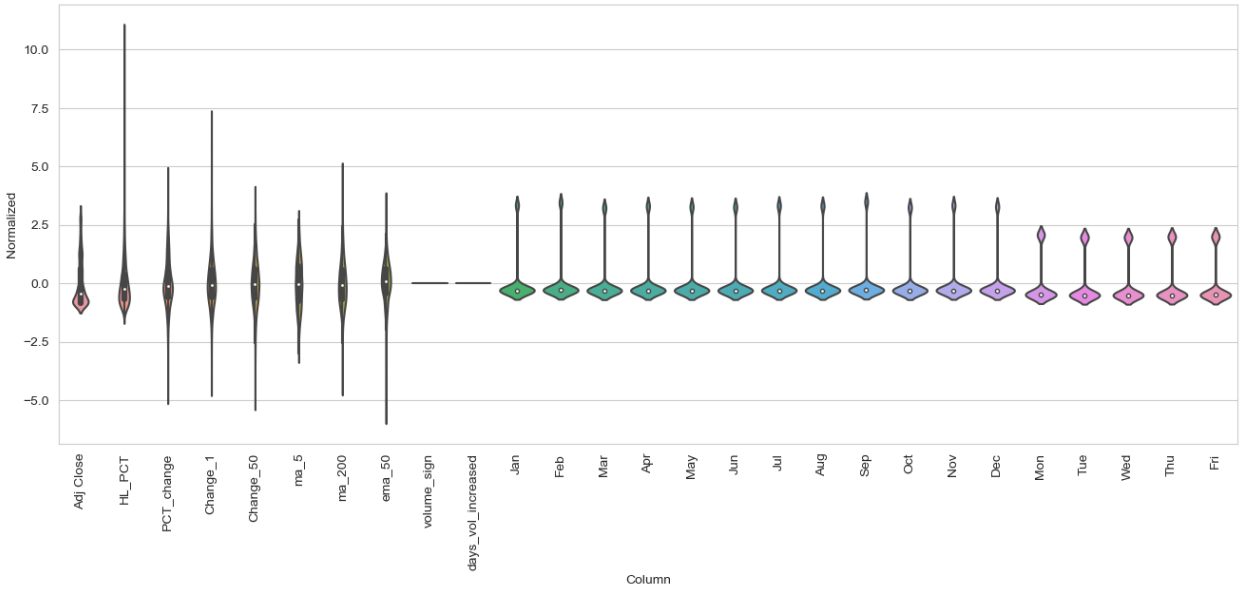
1. Train Test Split.
   I allocated the first 70% of TS data for training purposes; the next 20% for validation purposes and the last 10% for testing purposes. Note, I did not randomize the split because the data is time-dependent.

2. Scaling the data.
   It is important to scale the data before training the ML model. I used a standard scaler to scale all the features to have 0-mean and unit variance. Some features do have long tails, but there are no obvious errors.

Distribution of each feature: Before Scaling

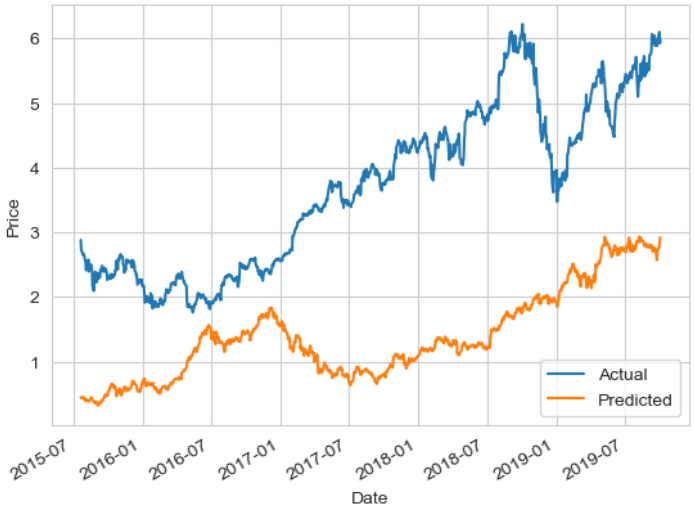


Distribution of each feature: After Scaling

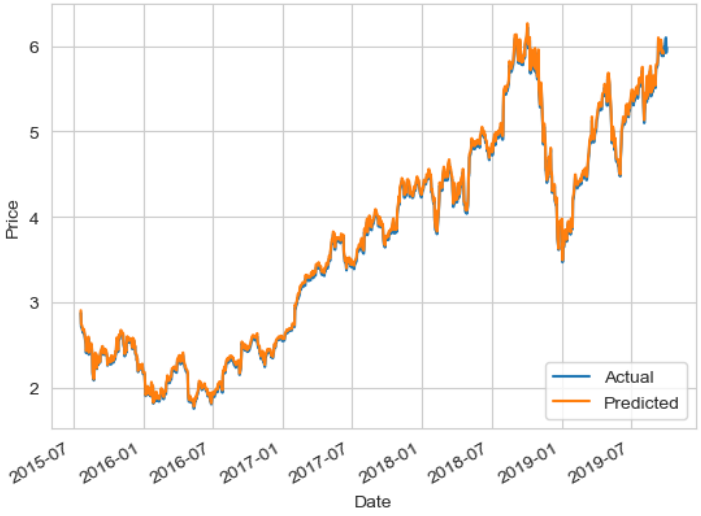


Implementation

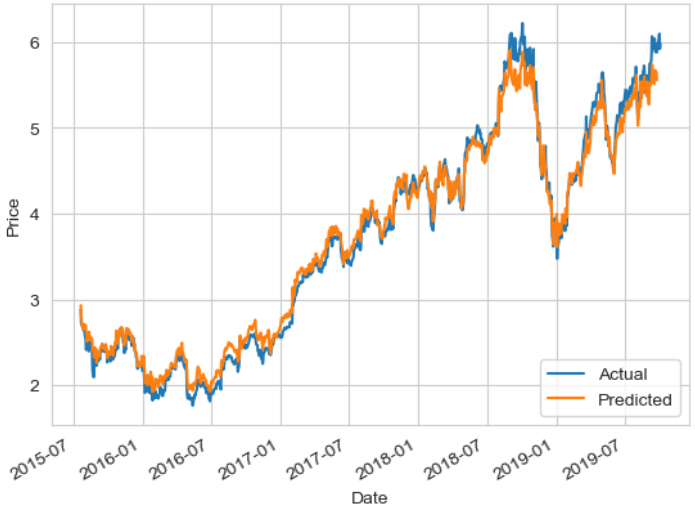
Baseline Model - stock prices are the same as the year before.



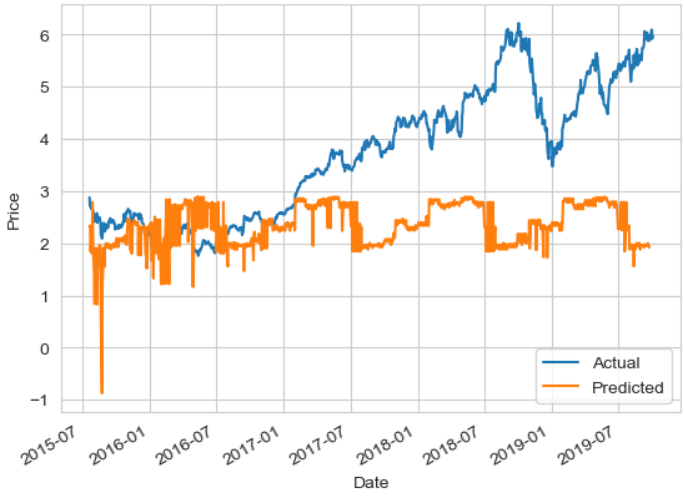
Linear Regression: predicts dependent variables (y) as the outputs given independent variables (x) as the inputs.



Polynomial Regression with Ridge regularization:
Ridge regression shrinks the coefficients and it helps to reduce the model complexity and multi-collinearity.



KNN: uses feature similarity to predict values of data points.



Refinement

Although there are numerous resources online on time series forecasting, it took me a while to

(a) engineer useful features,
(b) choose models that can be useful in this case, and
(c) understand the output produced by the deep neural networks.

In the first iterations of this project, I was trying to build simple models using sklearn. I wanted to make sure that I do not have a data leak or overfitted model. That's where shifting and regularization came into play.

Afterward, I also wanted to try deep learning models as I expected them to perform significantly better than typical simple machine learning models. First, I was looking for ways to avoid data leakage in deep learning, that's when I found, implemented, and tested a WindowGenerator class.

The implementation itself wasn't the most challenging part for me, unlike understanding how to interpret the result of a model. CNN and both of LSTM models returned the following shape (424, 7, 27).
Here,

7 - represents a forecast period (7 days)
27 - represents a number of features + target column

But was is 424? This is where I got stuck.
So, 531 - number of testing samples. 100 - window size. 7 - prediction.
Putting it all together we get: 531 - 100 - 7 = **424**

After implementing forecasting DNN models I was surprised to see that LSTM performed so poorly. I tried to find a reason for that but my only assumption lies in the data itself. During the past 1.5 years, the market was very unstable and unpredictable. However, it doesn't explain why simpler models have better predictions.

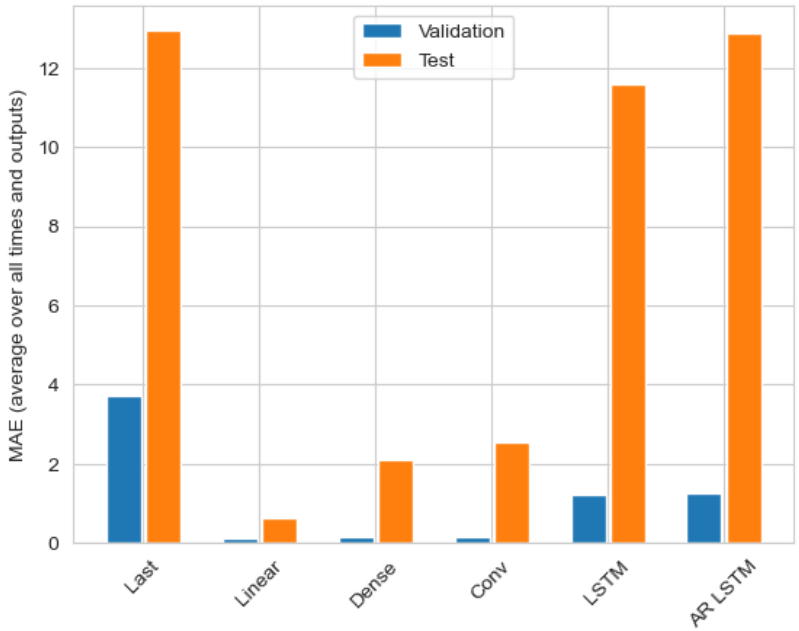
Finally, these predictions can't be blindly used to place stocks because different companies have different price trajectories over time. Therefore, this should be taken into account in the next iteration of this project.

Results

Model Evaluation and Validation (validation set)

Model	Implementation	MAE	RMSE	MAPE %
Baseline	Manual	2.32	2.52	61.6
Linear	Sklearn	0.14	0.19	3.945
Polynomial	Sklearn	0.16	0.22	4.65
KNN	Sklearn	1.48	1.86	34.9
Baseline	TensorFlow	3.72	4.12	98.13
Linear	TensorFlow	0.12	0.16	3.08
Dense	TensorFlow	0.14	0.19	3.83
CNN	TensorFlow	0.13	0.18	3.60
LSTM (single-shot)	TensorFlow	1.19	1.49	26.7
LSTM (autoregressive)	TensorFlow	1.24	1.58	27.16

Test vs Validation MAE for Each Model



Justification

From the table above, we can see that CNN performed the best out of all stocks on the validation dataset. On the bar graph, however, CNN has a higher MAE on the test dataset.

First, as was mentioned before, it is surprising why both LSTM models did so poorly on the test data compared to simpler models. A recurrent model can learn to use a long history of inputs if relevant to the predictions the model is making. It is possible that both of the LSTM models were overfitted (they learned the behavior of a stable stock market and were unable to catch the volatility).

Second, Linear, Dense, and CNN seem to perform approximately the same on the validation dataset. However, the test error grows as the complexity of the model increases.

Nevertheless, I decided to use CNN as my model of choice. It makes predictions based on a fixed-width history, which may lead to better performance than the dense model since it can see how things change over time.

Conclusion

Reflection

I decided to do this project to learn more about forecasting and time series. Being super interested in finance and the stock market I thought it will be an interesting project to complete. I knew it is going to be a challenging process but I underestimated the complexity of stock price prediction. Nevertheless, I was able to learn a lot about financial data and various forecasting algorithms.

During my learning, I found that the stock market follows a random walk which makes it extremely hard to predict. The new news is the only thing that can change the price of a stock, and since the news cycle is unpredictable, stock prices, therefore, move randomly.

It would be interesting to see if training the model on a bigger dataset either by adding multiple tickers or incorporating more data (see improvements section) will increase the accuracy.

Improvement

1. Scrape, analyze and incorporate more data:
    a. News headlines
    b. Reddit discussions
    c. Earnings reports
    d. Dividend yield
2. Feature engineering.
    a. Try more MA options
3. Improve modeling:
    a. Ensemble learning
4. Improved functionality of a web application:
    a. Add more stocks
    b. Select number of days to predict

References

1. Tensorflow - Time-series: https://www.tensorflow.org/tutorials/structured_data/time_series
2. Feature engineering: https://alphascientist.com/feature_engineering.html
3. In 12 minutes stock analysis with pandas and scikit learn: https://towardsdatascience.com/in-12-minutes-stocks-analysis-with-pandas-and-scikit-learn-a8d8a7b50ee7
4. Keep it simple, keep it linear: https://towardsdatascience.com/keep-it-simple-keep-it-linear-a-linear-regression-model-for-time-series-5dbc83d89fc3