Investment Portfolio Prediction

Tanner Kunz

Springboard Data Science Career Track

Branko Kovac

June 2022

Problem Statement

With the stock market being so volatile, many advise that now is the best time for young investors and people in general to start investing in the stock market[1]. Investing in the stock market is often thought to be one of the best ways to start to build and grow wealth. The problem to this is that investing can seem like a daunting task especially if you are just starting out and can feel like gambling if you don't know what to do. There is some general advice that can help people to get started such as create a portfolio of investments to mitigate risk, but this can feel daunting as well. It is difficult to know which companies to invest in to offset the movements of other investments. Some say the best way to start investing is to invest in Exchange Traded Funds (ETF) as these are investments that are created to follow different markets. This is seen to be one of the best ways to mitigate risk when investing. One of the most popular ETFs is the Vanguard S&P 500 ETF which is modeled to track the S&P 500 Index. This is not bad advice as the S&P 500 Index has had average annual growth of around 10% since 1926[2]. However ETFs do have some fees associated with them and could potentially limit your gains. These concerns with investing led me to wonder if I could create a simpler way to generate a reliable portfolio as a means to start investing. I then wanted to accurately predict future returns of these portfolios using past data.

Methods

The way I went about solving these problems was starting by generating 100 random portfolios of 20 different stocks. The list of stocks used to generate these random portfolios came from a list of stocks in the S&P 100 Index in 2016[3]. I used this list of stocks because the S&P 100 index is an index of 100 of the biggest blue chip companies across multiple industry sectors.

I used the closing day price as the price of the stock for each day. To calculate the returns of the stock, I divided the price of the stock minus the purchase price by the purchase

price. The value that I used as the purchase price was the initial closing day price of the data.

$$return = \frac{price - purchase\ price}{purchase\ price}$$

Risk is difficult to quantify, so I used the beta value which is widely used as a measure of risk in the finance industry. This is calculated by dividing the covariance of the return of an investment and the market by the variance of the market. This value is measuring how an

$$\beta = \frac{Cov(return, market\ return)}{Var(market\ return)}$$

investment moves compared to the market. If the investment moves the same as the market then it will have a beta of 1. If it is more volatile than the market then it will have a beta greater than 1. If it is less volatile than the market then it will have a beta less than 1. If the investment moves inverse to the market then it will have a negative beta, but this is unlikely considering the stocks that will comprise these portfolios.

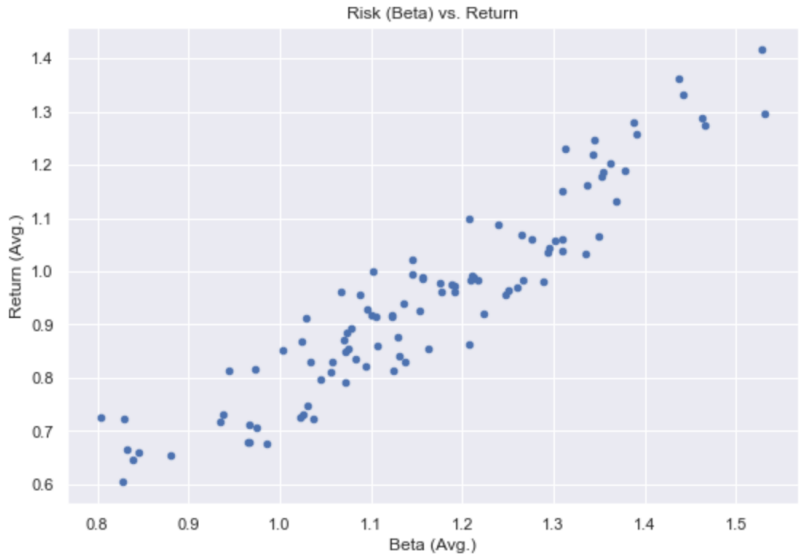
The return and the beta of the portfolio was calculated by taking the average of the returns and betas within the portfolio. This assumes that each stock has equal weight in the portfolio. This is the preferred method of calculating the beta for a portfolio as calculating it from the returns of the portfolio includes the unsystematic risk that is associated with stock specific betas[4].

Dataset

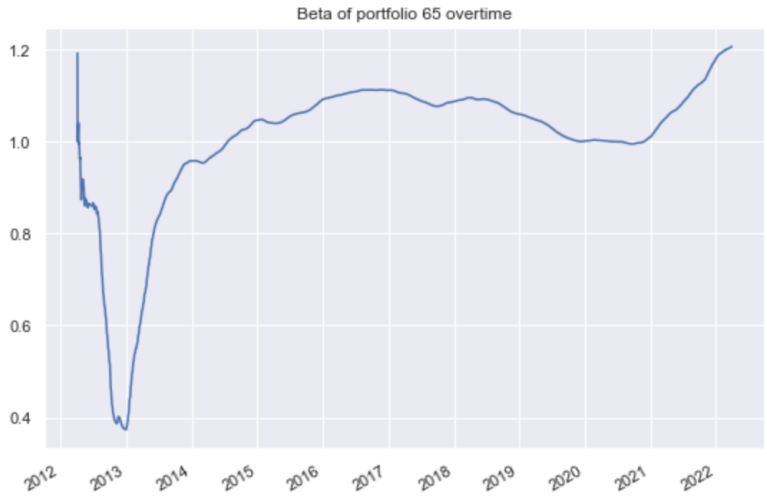
The dataset contains the closing price of stocks on the S&P 100 for the past 10 years from Yahoo Finances' website which was accessed using the Yahoo Finance API[5]. I also used the S&P 500 Index closing price for the past 10 years downloaded from FRED's website[6].

Exploratory Data Analysis

I started this analysis by visualizing the average returns and average betas for each portfolio. This positive linear trend shows that higher risk/volatility is associated with higher returns. The market has a beta of 1, so for this project I want to focus on the portfolios around 1 that also have higher returns.

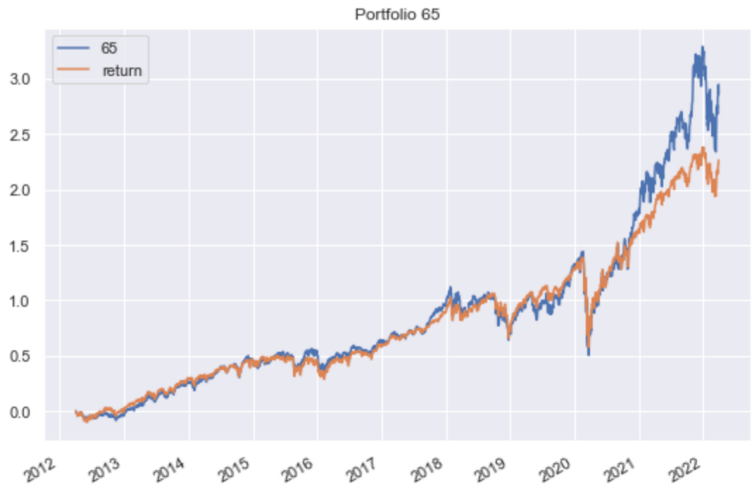


The next step in the EDA is to use a t-test to compare the betas of the portfolio against 1 which is the beta of the market. I used a two-sided t-test as I wanted to find the portfolios that had a mean beta that wasn't statistically different from 1. This statistical test will help me to find which portfolios most similarly follow the movements of the market. The null hypothesis of this



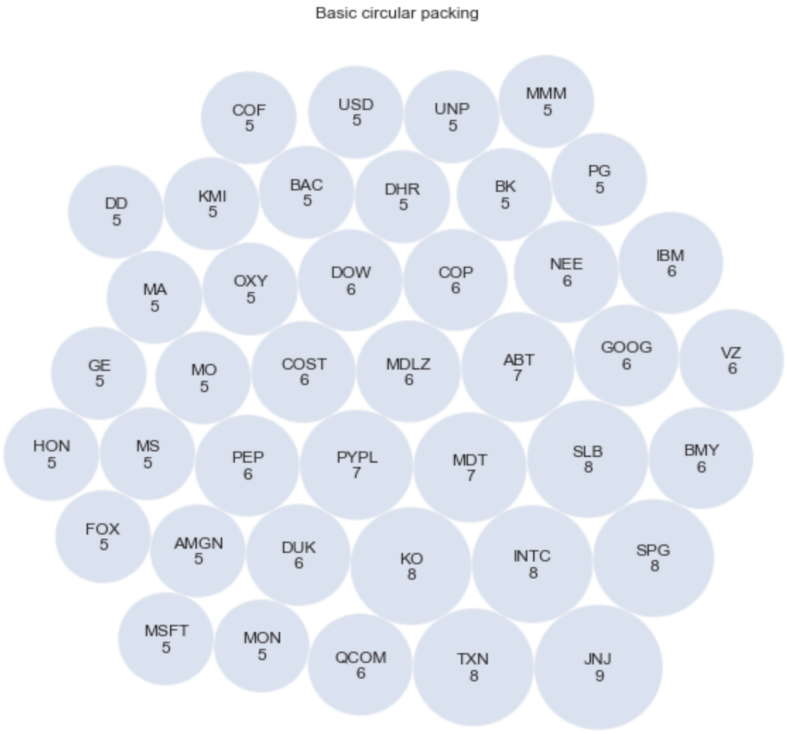
test is that the mean beta of the portfolio is equal to 1. The alternative hypothesis is that the mean beta is greater than or less than 1. This test resulted in only one portfolio having a p-value greater than the alpha of 0.05, which was portfolio 65 with a p-value of ~0.247.

I then conducted another t-test to compare the average returns of the portfolios against the average market return. I wanted to find which returns had a higher average return than the market. This required a one-sided t-test. The null hypothesis of this test is that the average return of the portfolio is less than or equal to the average market return. The alternative hypothesis is that the average return of the portfolio is greater than the average market return. This test resulted in 75 different portfolios that had a p-value less than the alpha of 0.05. One of these portfolios was portfolio 65, which makes it the primary portfolio after meeting the criteria that I was looking for in both statistical tests.



I also created a visualization of the most common stocks in the 20 portfolios that had the lowest average beta that also outperformed the market. This helps to understand which different stocks can diversify the portfolio. Some of the most frequent stocks in these portfolios are Johnson & Johnson (JNJ), Simon Property Group (SPG), Intel (INCT), Texas Instruments (TXN), Coca-Cola (KO), and Schlumberger Limited (SLB). Besides Texas Instruments and Intel which operate in the technology industry, these companies all operate in different industries. I am interested in optimizing the weight of each stock to minimize risk, but to also maximize

returns and this information could be valuable for that process. However, this seems to be something that is outside the scope of this project.



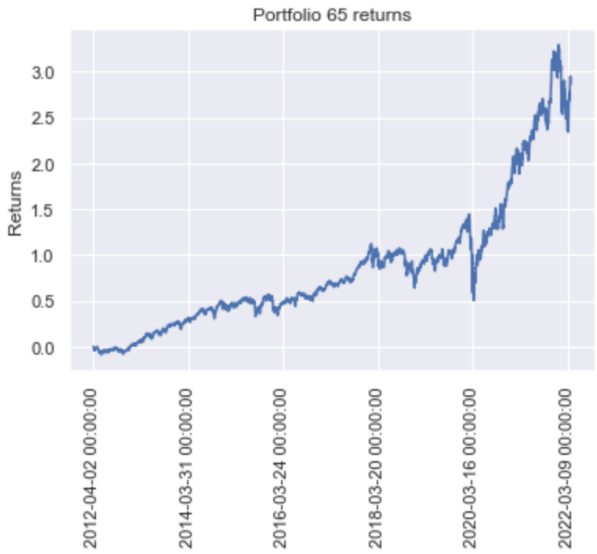
Feature Engineering

The only features that this project has are the returns, beta values, and the dates which make the feature engineering relatively straightforward. Before I apply feature engineering to this data, I need to define what I am exactly trying to predict. I am trying to forecast future returns with accuracy, but I want to forecast far enough ahead that the model will be useful. I also do not want to forecast so far ahead that the model isn't accurate. I chose to forecast 6 months into the future which is about 126 trading days. This meant that I would use 95% of my data for the training set and 5% of the data as the test set. The other feature engineering that I needed to perform was to create offsets of the target values for the machine learning model's inputs. I created offsets of 7 months to 6 months prior to the target value as well as a rolling min, max, mean, and standard deviation using the Featuretools library[7].

Modeling

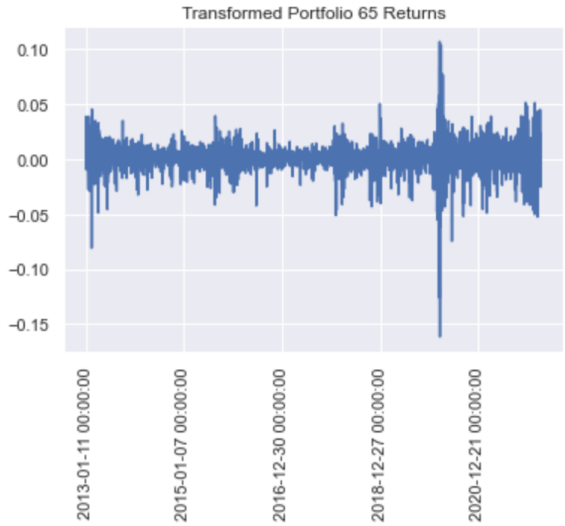
This part of the project will focus on predicting future returns of the portfolios. To do this I use three different methods: traditional forecasting methods, machine learning modeling, and deep learning. These methods each have their own strengths and weaknesses. These models also may perform better on some types of time series data rather than other types of data as different time series have different attributes.

The traditional forecasting model requires different assumptions in order to create a good model. The autoregressive moving average (ARMA) model is one of the most commonly used forecasting models, but requires that the data be relatively stationary meaning that the mean and variance don't change overtime. Looking at the returns the mean and the variance do change over time which implies that the data isn't stationary. This is also proven by using the

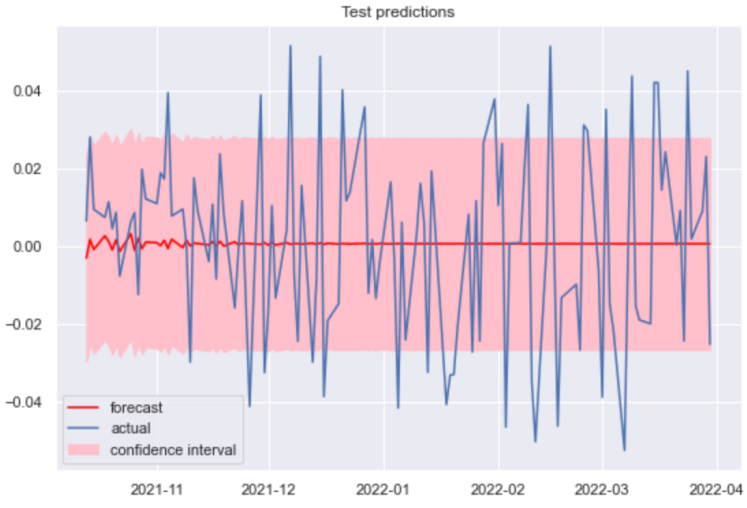


Adjusted Dickey-Fuller Test which results in a p-value of ~0.995 meaning that the data is non-stationary data. The autoregressive integrated moving average (ARIMA) model is a model that can be applied to non-stationary data that is made stationary by transformation. I transformed the data by first taking the square root of the returns and then took the first difference of the values. After the transformations it appears to have a more constant mean and

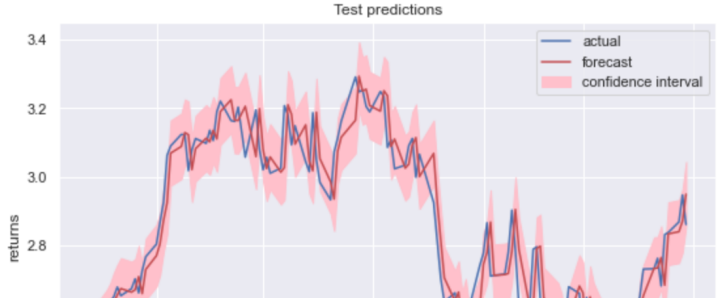
the variance largely stays the same. Using the Adjusted Dickey-Fuller Test on the transformed



data results in a p-value of ~0.000 meaning that the data is stationary. Because it isn't clear what order to use for the ARIMA model, I used the pmdarima[8] library which has an auto_arima function that looks for the order with the best Akaike information criterion score which is a measure of prediction error. This gave me an ARIMA order of (5, 0, 4). The model seems to try to predict the first few time steps into the future, but predicts a constant value after that. This is also the transformed data so after using a reverse transformation the results predictions look



more accurate. However this model assumes that I already have the data for the day before.

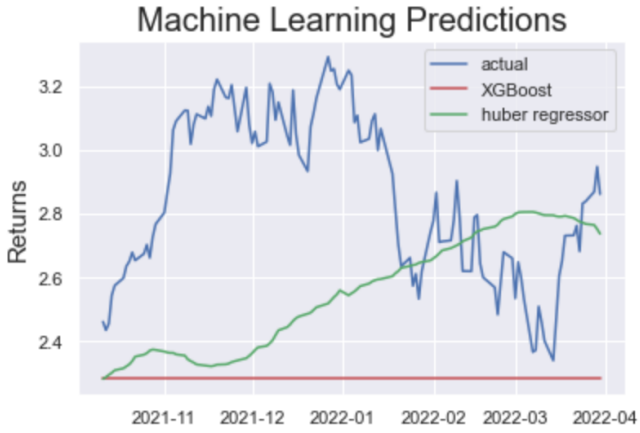


Even though this does create a good model, it is unrealistic and not productive as I wouldn't be able to make accurate predictions 6 months into the future. Traditional forecasting is a good method for forecasting for certain types of data, but it doesn't seem to be the best method to answer the scope of this project.

Machine learning modeling uses the feature engineered data with the value offsets because it uses all the data in a row for prediction and doesn't look at previous information. I decided to use 10-fold cross validation to choose which models to use. The models being compared are: linear regression, Huber regressor, random forest, and XGBoost. These are the

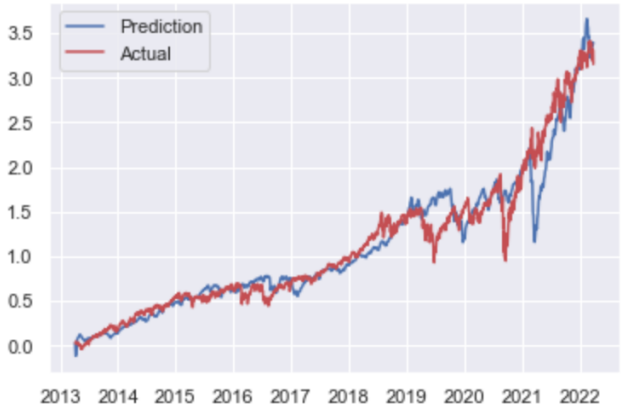
	Linear_Regression	Huber_Regressor	Random_Forest	XGBoost
MSE	0.090857	0.069689	0.141852	0.133780
MAE	0.193027	0.178599	0.205327	0.206221

results of the cross validation and the linear models outperformed the tree-based models. However, I thought it would be interesting to hyperparameter tune a linear model as well as a tree-based model. I chose to use the Huber regressor and XGBoost models because they performed the best in their given model categories. The Huber regressor is similar to ridge regression, in that there is a regularization parameter applied. This makes it more robust to outliers which is one of the reasons that I think that it performs well on this data. XGBoost is a very versatile boosting algorithm and it could perform well with hyperparameter tuning. After using random grid search cross validation for both models the Huber regressor performed better on the test set with a mean squared error (MSE) of ~0.231 and the XGBoost model had a MSE of ~0.385. The Huber regressor seems to follow the general trend of the model, but not follow



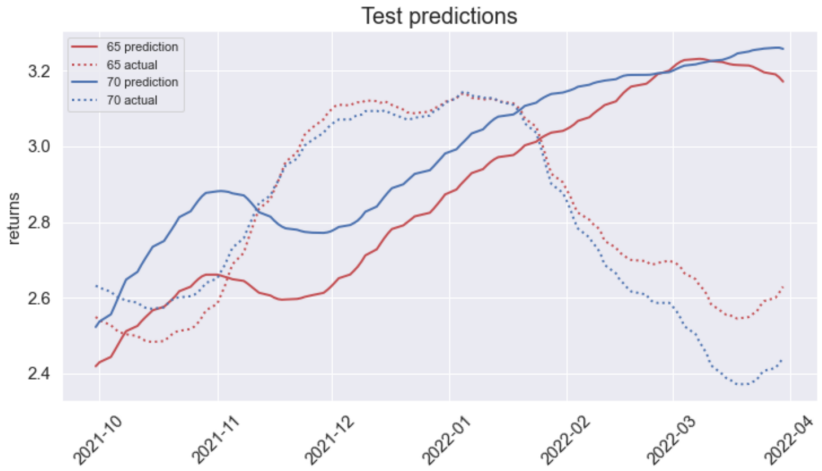
the specific movements. The XGBoost model performs similarly to the traditional forecasting model in that it predicts a constant value in the long run. The Huber regressor performs decently, but not well enough that it can accurately create reliable future predictions.

The deep learning model that I used is a specific type of recurrent neural network (RNN) called a Long Short Term Memory (LSTM) model. RNN are able to use previous values for future predictions. LSTMs are a more specialized RNN that also take into account previous values, but give weight to how much previous data is used for the predictions. The set up of my LSTM model uses two LSTM cells with a linear output cell. I also used an MSE loss function and Limited-memory BFGS (LBFGS) optimization method for training the model. The LBFGS optimization algorithm is used as it works well with lots of data. This allowed me to use not just one portfolio for training, but I was able to use all the portfolios that I generated to train the model. The training takes a relatively long amount of time, but after the model is trained it is able to quickly make predictions on new input data. This model also is able to produce fairly accurate results compared to the other models. Looking at the graph of the predictions, it is clear that this model's predictions are much more organic than the other model's predictions. There are obvious flaws to the model where it isn't able to predict sudden drops in the market. However,



where there is more consistency the model performs more accurately. The performance on the test set is also better than any of the other models with a MSE of ~0.0380. Based on the graph it

does not look like the best model, but this is because of the range of the y-axis. The model also performs better on portfolio 65 which has a lower overall beta value than portfolio 70.



Results

These models were all interesting to look at as there are pros and cons to looking at each different model. The traditional modeling performs well under certain conditions, such as stationarity and predicting a few steps into the future. The traditional modeling also has relatively low computation time compared to some of the other models. Of the machine learning models the Huber regressor was an impressive model as it was able to predict the general trend of the portfolio also with low computation time. The XGBoost model had high computation time and wasn't able to produce accurate predictions. The LSTM model was the most impressive as it was able to create accurate and more organic predictions compared to the other models. The only downside to the LSTM model is that there was high computational time. Comparing all the models against each other it is clear that the LSTM model is the closest to following the actual

trend of the portfolio and the Huber regressor is able to follow the general trend. If this model was to be used in production, I would recommend using the LSTM model for accuracy. However, if the concern was just to find the general trend in a limited amount of time the Huber regressor would be a viable option for answering this type of problem.

Conclusion

The goal of the project was to be able to generate a small portfolio that was not more risky than the market and accurately forecast a portfolio's future returns. I was able to solve the first aspect by randomly generating 100 random portfolios of 20 different stocks listed on the S&P 100 Index. During the exploratory data analysis phase, I used a two-sided t-test to find a portfolio that had a mean beta value that was not statistically different from the market beta of 1. I used a one-sided t-test to find the portfolios that had an average return that was significantly higher from the market average return with one of these portfolios being the one from the previous test. This process and analysis showed that it is possible to create a small portfolio that relatively follows the market risk and also produces better returns than the market.

The second aspect is answered during the modeling phase of the project. After evaluating several different models created using traditional forecasting methods, machine learning models, and deep learning I was able to create an accurate model using an LSTM neural network. This model was able to generate predictions that move organically and is more accurate than the other models. This model is not flawless as it seems to perform poorly when there are sudden drops in the returns. But when the portfolio returns are more consistent, the model predictions are very accurate.

In the future, I would like to productionize this project by creating a regularly updating dashboard that tracks the best performing randomly generated portfolios and also shows future projections. This would be a great tool for new investors to get information on how to create a portfolio, how risk is measured by beta, and what they might be able to expect from investing. I

would also want to improve the LSTM prediction model by incorporating some news/social media analysis that could be beneficial to improving the model's performance when there are sudden drops in the returns. Another aspect that I would want to improve would be to create a portfolio optimization aspect to optimize the weights of each investment in the portfolio. Overall, I was able to solve the main goals of the project through the use of exploratory data analysis and deep learning and see that this project could become a tool to inspire people to start their investing journey.

Bibliography

1. CNBC. 2022. “Why it’s a good time for young investors to put money in the market.” https://www.cnbc.com/2022/05/19/why-its-a-good-time-for-young-investors-to-put-money-in-the-market.html
2. Investopedia. 2022. “What Is the Average Annual Return for the S&P 500?” https://www.investopedia.com/ask/answers/042415/what-average-annual-return-sp-500.asp
3. Mometic. 2016. “S&P 100 Stock Symbol Listings” https://blog.mometic.com/s-p-100-stock-symbol-listings/
4. Corporate Finance Institute. 2022. “Beta Coefficient” https://corporatefinanceinstitute.com/resources/knowledge/finance/beta-coefficient/
5. Yahoo Finance API. https://www.yahoofinanceapi.com
6. FRED. Economic Research Federal Reserve Bank of St. Louis https://fred.stlouisfed.org
7. Featuretools.https://featuretools.alteryx.com/en/stable/
8. Pmdarima. ARIMA estimators for Python. http://alkaline-ml.com/pmdarima/